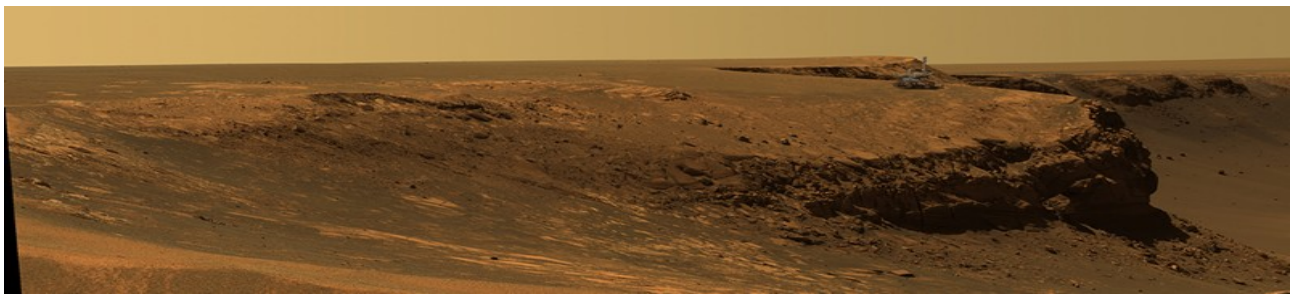
# 1 El Mundo de los Robots

*Página anterior: Rover de Marte.*
La foto es cortesía de NASA/JPL-Caltech

*No quisiera que nunca se los regresara a la Tierra. Los construimos para Marte, y en Marte es donde deberían quedarse. Pero* ***Spirit*** *(Espíritu) y* ***Opportunity*** *(Oportunidad) se han convertido en más que máquinas para mí. Los rovers son nuestros sustitutos, nuestros precursores robóticos en un mundo al cual, como humanos, aún no estamos preparados para visitar.*

-: Steve Squyres en **Roving Mars**, Hyperion, 2005.



El borde del Cráter Victoria en Marte. El rover Opportunity ha sido sobreimpreso en el borde del cráter para mostrar la escala. La foto es cortesía de JPL/NASA/Cornell University, octubre, 2006

La imagen superior es una de las miles que se han enviado desde el Spirit y el Opportunity  desde la superficie de Marte. No hace falta decir que probablemente pasen varios años, décadas o aún más, antes de que un humano pueda poner un pie en Marte. Los rovers Spirit y Opportunity aterrizaron en Marte, en enero de 2004, como robots geólogos cuya misión era analizar las rocas y los suelos del planeta rojo en busca de pistas de presencias pasadas de agua en el planeta. Se esperaba que los dos robots durarían aproximadamente 90 días. Tres años después, todavía estaban explorando la superficie del planeta, enviando invalorables datos geológicos y gráficos del planeta. En el mismo mes en que los rovers aterrizaban en Marte, en la Tierra, el robot rover Tumbleweed viajó 40 millas a través de la meseta antártica americana, transmitiendo datos meteorológicos a su estación vía satélite. Además de sobrevivir a condiciones adversas en Marte y en la meseta Ántártica, los robots están lentamente transformándose en objetos de consumo doméstico. Tomemos como ejemplo el Roomba de iRobot Corporation. Desde que se lanzaron en 2002, más de 2 millones de Roombas han sido vendidos para aspirar y limpiar pisos.

Algo en común entre los robots arriba mencionados es que han sido previamente diseñados para realizar tareas específicas: analizar piedras y suelos en la superficie de Marte, meteorología en la capa polar, o aspirar una habitación. Sin embargo, el núcleo de la tecnología robótica es casi tan sencillo de usar como las computadoras. En este curso, recibirán un robot personal. A través de este robot personal, aprenderán a impartirle instrucciones para realizar una variedad de tareas. Como los robots arriba mencionados, el robot también es un rover. Sin embargo, a diferencia de los robots anteriores, este robo personal no viene preprogramado para realizar ninguna tarea específica. Tiene ciertas capacidades básicas (sobre las cuales aprenderá) y puede ser programado para usar sus capacidades para realizar varias tareas. Esperamos que el proceso de aprender acerca de las capacidades del robot y el hecho de lograr que lleve a cabo diferentes cosas les resulte fascinante y divertido. Es este capítulo, les presentaremos al mundo de los robots y luego los introduciremos en su propio robot personal y en algunas de sus capacidades.

## ¿Qué es un robot?

El Diccionario en línea Merriam-Webster brinda las siguientes definiciones de la

palabra *robot*:

1. una máquina que parece humana y que lleva a cabo varios actos complejos (como caminar o hablar); también una máquina similar pero ficcional cuya carencia de emociones humanas es en general enfatizada; y también una persona eficiente e insensible que funciona automáticamente.
2. un dispositivo que automáticamente lleva a cabo tareas complicadas y generalmente repetitivas.
3. un mecanismo guiado por controles automáticos

En el mundo de hoy, las primeras dos definiciones serán probablemente consideradas arcaicas (la tercera interpretación en la primera definición no pertinente). Los robots fueron originalmente concebidos como entidades simil-humanas, reales o ficcionales, carentes de emociones, que realizaban tareas que eran repetitivas, aburridas o pesadas. Los robots de hoy en día vienen en distintas formas y tamaños y llevan a cabo todo tipo de tareas (ver ejemplos abajo). Mientras que muchos robots eran usados para tareas repetitivas o aburridas (incluyendo el Roomba; a menos que disfruten del costado terapéutico de pasar la aspiradora : -), los robots actuales son capaces de realizar mucho más de lo que se contempla en las definiciones de arriba. Aún en los robots ficticios, la carencia de capacidad emocional parece estar superada (ver por ejemplo la película de Steven Spielberg, *Inteligencia Artificial*).

Para nuestro propósito, la tercera definición es más abstracta y quizás más apropiada. Un robot es un mecanismo o una entidad artificial que puede ser guiado por controles automáticos. La última parte de la definición, guiado por controles automáticos, es donde nos centraremos en este curso. Es decir, dado un mecanismo capaz de ese tipo de guía, ¿qué es lo que está implicado en la creación de esos controles?

## Una breve historia de los Robots

Los robots modernos estaban inicialmente concebidos como robots industriales diseñados para asistir en tareas de manufactura automatizadas. Unimation (la primer compañía comercial de robots) fue creada hace alrededor de 50 años. A medida que crecía el uso de robots en la industria manufacturera, se comenzó a experimentar con otros usos para los mismos. Los primeros robots industriales eran principalmente largos brazos anexados a una base fija. Sin embargo, con el desarrollo de robots móviles, se les empezó a encontrar usos para otras áreas. Por ejemplo, para explorar ambientes peligrosos al alcance de sitios radiactivos,volcanes, buscar y destruir minas, etc. Comenzamos este capítulo introduciendo a dos robots de Marte. El primer robot planetario aterrizó en Marte en 1997. En la última década y en forma creciente, los robots han incursionado en áreas nuevas y más atrayentes, como la medicina (Google: cirugía robótica, silla de ruedas robótica, etc.), los juguetes y el entretenimiento (Google: Pleo, SONY Aibo, LEGO Minsdstorms, etc.), e incluso la educación (Google: IPRE). Algunos de los desarrollos más fascinantes en robótica aún están en fase de investigación, por ejemplo, en inteligencia artificial, los investigadores están intentando desarrollar robots inteligentes y están utilizando robots para comprender y explorar modelos de la inteligencia humana. Hemos provisto algunas claves de búsqueda (realizar las búsquedas mencionadas arriba) como ejemplos de varios robots y sus usos. Hay numerosos sitios web donde podrán buscar más información acerca de la historia de los robots. Lo dejaremos planteado como un ejercicio.

Hoy en día es difícil imaginar la vida sin un motor de búsqueda Web. Hay varios motores de búsqueda disponibles, pero el que provee Google Inc. ha devenido en sinónimo de búsqueda en la Web. Tanto es así que la gente utiliza la frase común “¡googlealo!”

Quizás ustedes tengan una preferencia personal de motor de búsqueda. Adelante, utilícenlo y busquen los ítems sugeridos aquí.

## Los Robots y las Computadoras

En las últimas décadas, las computadoras se han tornado crecientemente ubicuas. Lo más probable es que estén leyendo esta oración en una computadora. Si está leyendo este texto en línea, el texto está proviniendo de otra computadora (ubicada en alguna parte de la costa oeste del río Delaware en la zona sureste del estado de Pensilvana en los Estados Unidos). En este viaje desde la


Una estampilla postal titulada El mundo de la invención *(La Internet)* fue impresa por el Correo Postal Real del Reino Unido en marzo 1, 2007 como homenaje al desarrollo de la Red de Redes (World Wide Web).

computadora de Pensilvana a su computadora, el texto probablemente ha viajado a través de varias computadoras (¡varias docenas si están en las afueras del estado de Pensilvana!). Lo que hace que este viaje sea casi instantáneo es la presencia de redes de comunicación a través de las cuales operan Internet y la World Wide Web. Los avances en las tecnologías de comunicación inalámbrica hacen posible el acceso a Internet desde casi cualquier lugar del planeta. La razón por la cual ustedes están sentados frente a una computadora y aprendiendo acerca de los robots es básicamente por el advenimiento de esta tecnología. A pesar de que los robots no están tan difundidos como las computadoras, no se han quedado tan atrás.  De hecho, es precisamente el avance en las computadoras en las tecnologías de la comunicación que han permitido que ustedes se familiaricen con el mundo de los robots.

La relación entre los robots y las computadoras es la base del uso de la frase controles automatizados al describir un robot. El control automatizado de un robot casi siempre implica que hay una computadora involucrada. Por este motivo, en el proceso de aprender acerca de los robots y de jugar con los mismos, también descubrirán el mundo de las computadoras. El robot tiene una computadora embebida en él. Ustedes estarán controlando el robot a través de su computadora. Es más, lo harán a través de una tecnología inalámbrica llamada bluetooth. Inicialmente, para nuestro propósito, aprender a controlar automáticamente a un robot será sinónimo de aprender a controlar una computadora. Esto se tornará más obvio a medida que avancemos en las lecciones.

El control automático involucra la especificación, de antemano, de la serie de tareas que deberá realizar el robot o la computadora. Esto se llama programación. La programación implica el uso de un lenguaje de programación. ¡En la actualidad hay más lenguajes de programación que lenguajes humanos! Quizás hayan escuchado algo acerca de alguno de ellos: Java, C, Python, etc. En este curso, realizaremos toda la programación del robot en el lenguaje de programación Python. Python, llamado así por el popular show de televisión Monty Python, es un lenguaje moderno que es muy sencillo de aprender y de usar.

Mientras hablamos de computadoras y de lenguajes, también deberíamos mencionar al sistema de software Myro (que proviene de My robot). Myro provee un pequeño grupo de comandos de robot que extienden el lenguaje Python. Esto hace que resulte sencillo, como verán, especificar controles automáticos para los robots.

## Un Robot propio: el Scribbler

El Robot Scribbler

El robot Scribbler que se muestra aquí es también un robot. Puede moverse en su entorno. Las ruedas y sus otras funciones pueden ser controladas a través de una computadora vía una interfaz inalámbrica. Sus asistentes de laboratorio le proveerán un Scribbler y los componentes requeridos para permitir la comunicación inalámbrica. Una vez configurado, podrán controlar los movimientos del robot (y todas las otras funciones) a través de la computadora. Además de moverse, el robot también puede emitir sonidos (beeps) y, con ayuda de una lapicera insertada en el puerto de lapicera, puede dibujar una línea donde sea que vaya (por eso su nombre, Scribbler –Garabateador en español). El robot puede moverse hacia delante, hacia atrás, dar vueltas, girar, o realizar una combinación de estos movimientos, dándole

una adecuada funcionalidad para viajar a cualquier lado en la superficie del entorno. Además de desplazarse, el robot Scribbler puede también percibir ciertas características de su ambiente. Por ejemplo, es capaz de percibir una pared o un obstáculo, o una línea en el piso. Discutiremos las capacidades sensoriales del Scribbler más adelante.

## Desarrollar la siguiente actividad

Las primeras actividades les muestran como instalar la computadora y el robot y los ayudarán a familiarizarse con el Scribbler. Esto involucrará las cuatro actividades siguientes:

1- Primero lo primero: instalar Myro

2- Asignar un nombre al robot.

3- Manejar el robot por el entorno

4- Explorar un poco más allá

Quizás sea necesaria la ayuda del instructor para la primera actividad para asegurarse de que saben cómo instalar y utilizar el robot para continuar avanzando en lo que resta del texto.

### 1.- Primero lo primero: Instalar Myro

El Fluke Dongle le agrega Bluetooth y otras capacidades al Scribbler.

Cuando recibieron su robot, su software y hardware ya había sido configurado para el uso. El software que utilizaremos para controlar el robot se llama Myro (por mi robot - ”My robot”- en inglés) que trabaja en conjunción con el lenguaje Python. En este primer ejercicio, iniciaremos el robot y el software para asegurarnos que el software puede comunicarse exitosamente con el robot a través de su computadora. Si Myro no ha sido instalado en su computadora, deberían obtener una copia del mismo (insertando el CD de Myro en la computadora o siguiendo las instrucciones del manual de instalación Myro).

En una sesión típica, iniciarán el software Python, conectarán el robot a través de la librería Myro y luego controlarán al robot a través de la misma. Hemos preparado el sistema de manera que toda la comunicación entre la computadora y el robot funcione inalámbricamente con una conexión Bluetooth. La tecnología Bluetooth es una tecnología de comunicación inalámbrica común que permite que los dispositivos electrónicos se comuniquen entre sí en distancias cortas. Por ejemplo, se utiliza con frecuencia el Bluetooth en teléfonos celulares para permitir la comunicación inalámbrica entre el teléfono celular que puede estar en su bolsillo y el auricular inalámbrico. Este tipo de comunicación requiere de dos dispositivos físicos que sirven como receptores y transmisores. En el equipo Scribbler que recibieron, hay un par de estos dispositivos Bluetooth: uno se conecta al Scribbler (Fluke Dongle) y el otro al puerto USB de su computadora. Si la computadora tiene una capacidad Bluetooth incorporada, quizás no necesite el que entra en su computadora. Asegúrense de que estos dispositivos estén enchufados, y que el robot y la computadora estén

encendidos.

### 2.- Nombrar al robot

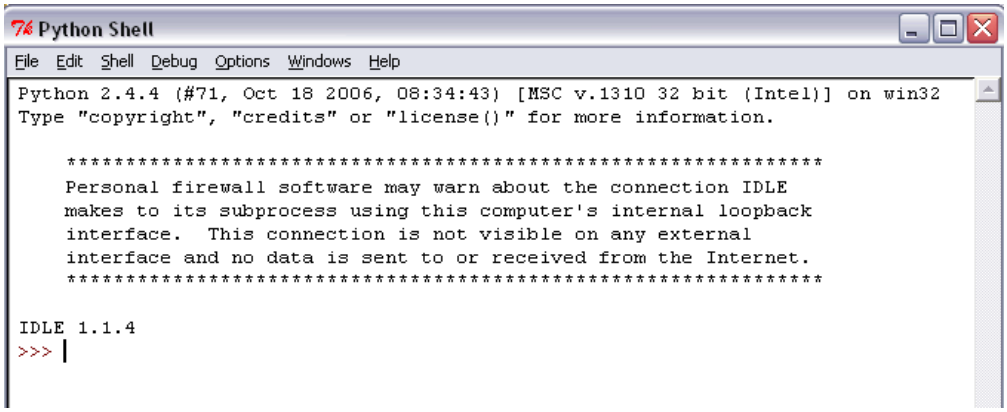
En este ejercicio, nos conectaremos con el robot y lo haremos hacer algo simple, como hacer un beep. Luego le daremos un nombre para personalizarlo. Estas tareas pueden ser llevadas a cabo siguiendo los siguientes pasos:

1. Iniciar Python
2. Conectar con el robot
3. Hacer que el robot realice un beep
4. Darle un nombre al robot.

Dado que es la primera experiencia con el uso de robots, proveeremos instrucciones detalladas para realizar la tarea delineada arriba.

**1. Iniciar Python**: Cuando iniciaron el software, se creó un archivo llamado Start Python.pyw. Deberían copiar este archivo en una carpeta donde piensen guardar todos los programas del robot. Una vez hecho esto, naveguen hasta esa carpeta y ábranla. Adentro encontrarán el ícono de Start Python. Hagan doble clic en él. La siguiente ventana deberá aparecer en la pantalla de computadora:

1.

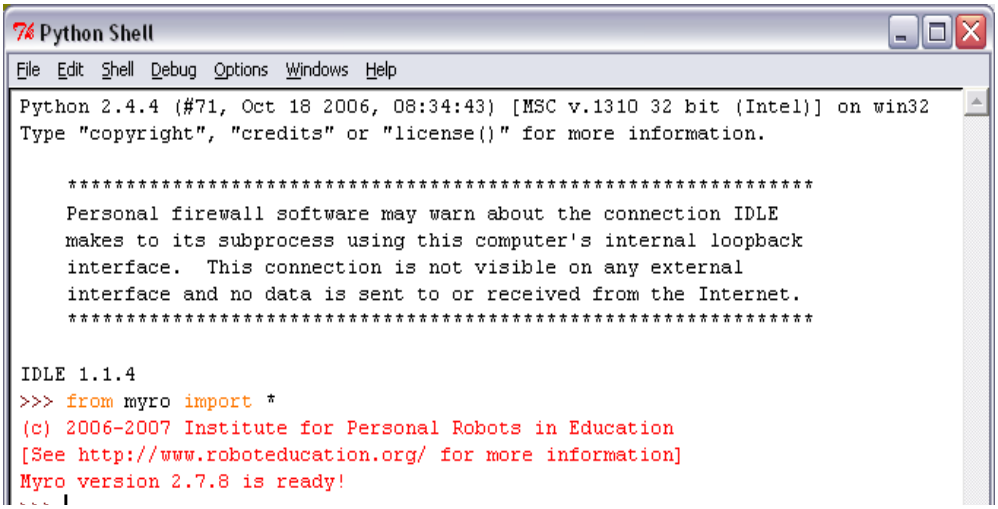



Lo que se ve arriba es la ventana de interacción de Python o el “Python Shell”. Este shell particular se llama IDLE (notar que arriba reporta que están utilizando la versión IDLE 1.1.4.) Ustedes ingresarán todos los comandos Python en esta ventana IDLE. El siguiente paso es utilizar Myro para conectar con el robot.

**2. Conectar con el robot:** Asegúrense de que el robot y la computadora tengan sus dispositivos Bluetooth insertados y que el robot esté encendido. Para conectar al robot, ingresen el siguiente comando en el shell Python:

```
>>> from myro import *
```

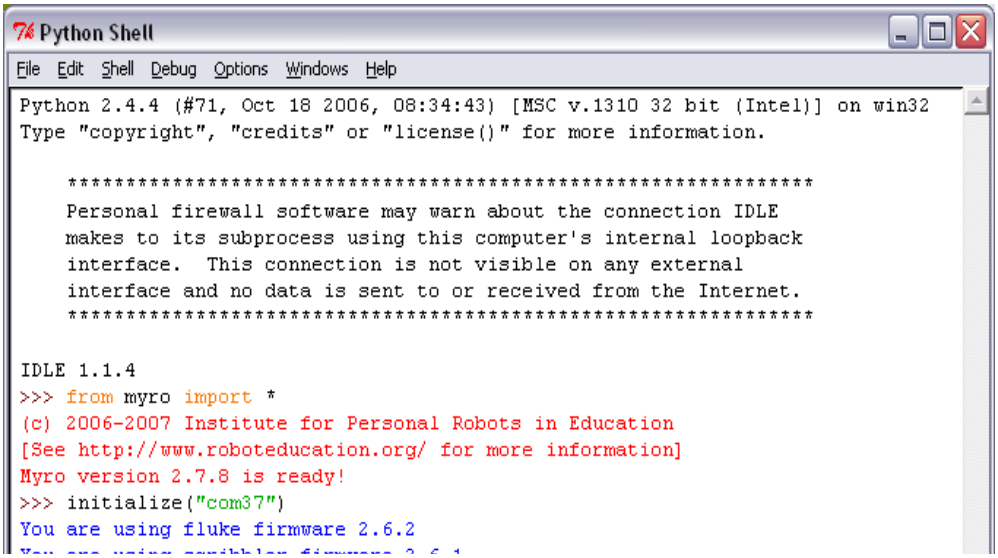
Esta interacción se muestra debajo (la versión Myro será diferente):




Esto significa que le han informado al Python Shell que estarán utilizando la librería Myro. El comando (o sentencia) import es algo que utilizarán cada vez que quieran controlar el robot. Luego de ingresar el import, se imprime cierta información útil sobre Myro y luego el Shell está preparado para el próximo comando Python. Ahora es el momento de conectar el robot ingresando el siguiente comando:

```
>>> initialize("comX")
```

donde x es el número puerto utilizando por su computadora para comunicarse con el robot. Si necesitan ayuda para averiguar el número de puerto, consulten con el instructor. El ejemplo debajo muestra como enunciar el comando cuando el puerto com5 se está utilizando:




Cuando ingresar el comando initialize, la computadora intenta comunicarse con el robot. Si esto tiene éxito, el robot responde con la línea Hello... mostrada arriba. Como se puede ver, el nombre del robot es BluePig. Ustedes pueden darle el nombre que quieran al robot. Haremos eso después. Primero, démosle un comando para que haga un beep, así sabemos que tenemos el control del robot:

3. Hacer que el robot realice un beep: En el Python Shell, entrar el comando:

```
>>> beep(1, 880)
```

El comando de arriba le da la instrucción al robot para que haga un sonido de beep en 880 Hertz por segundo. Adelante, inténtenlo. El robot hará un beep durante un segundo en 880 Hz. Intenten las siguientes variaciones para escuchar diferentes beeps:

```
beep(0.5, 880)
beep(0.5, 500)
etc.
```

Ahora, pueden darse cuenta de que tienen el control del robot. Utilizando comandos simples como los de arriba, pueden hacer que el robot se comporte de distintas maneras. Ahora podemos aprender un comando para darle al robot un nuevo nombre.

**4. Ponerle un nombre al robot**: Supongamos que le quisiéramos poner el nombre `Shrek`. Para hacerlo, todo lo que debe hacerse es ingresar el siguiente comando:

```
>>> setName("Shrek")
```

Sea cual fuere el nombre que decidan darle al robot, pueden especificarlo en el comando de arriba reemplazando las palabras `Shrek`. De ahora en más, ese será el nombre del robot. ¿Cómo lo sabemos con certeza? Intenten preguntarle el nombre:

```
>>> getName()
```

También reportará ese nombre cada vez que se conecten a él utilizando el comando de inicio:

```
>>> initialize("com5")
Waking robot from sleep...
Hello, I'm Shrek!
>>>
```

¡Felicitaciones! Han completado el primer ejercicio y están en camino de lograr más cosas divertidas con el robot. Antes de que sigamos, sería una buena idea que revisaran lo que han hecho hasta ahora. Cada sesión con un robot comienza iniciando el software Python (paso 1 arriba), seguido por la importación de la librería Myro y el encendido del robot. De ahí en más, ustedes pueden ingresar cualquier comando para el robot.

La librería Myro contiene docenas de comandos que permiten varios tipos de comportamientos del robot. En las próximas semanas aprenderemos varios comandos

de robots y también aprenderemos cómo usarlos para programar comportamientos complejos del robot. Algo para recordar es que todos los comandos se enuncian en el lenguaje Python. Por lo tanto, a medida que aprendan más acerca del robot y de su comportamiento, también aprenderán el lenguaje Python.

Una característica de los lenguajes de programación (como Python) es que tienen un modo muy estricto de tipear los comandos. Esto quiere decir, y ustedes pueden haberlo experimentado arriba, que el lenguaje es muy preciso con respecto a lo que se tipea y cómo se lo hace. Cada paréntesis, comilla, mayúscula o minúscula que construye un comando debe ser tipeado exactamente como es descrito. A pesar de que las reglas son estrictas, por suerte no son muchas. Pronto se sentirán cómodos con esta sintaxis y se convertirá en algo natural. Es necesaria la precisión en la sintaxis para que la computadora pueda determinar una interpretación exacta del comando y que resulte en la acción deseada. Por este motivo, frecuentemente se distingue a los lenguajes de computación de los lenguajes humanos caracterizándolos como lenguajes formales (opuestos a los lenguajes naturales usados por los humanos).

### 3. Manejar al robot por el entorno

En este ejercicio, los introduciremos en un modo de hacer que el robot se mueva por el entorno, controlándolo en forma manual con un dispositivo de game pad controller –controlador de juego- (ver imagen). Como lo hicimos anteriormente, ubicar el robot en un suelo abierto, encender el robot, iniciar Python y conectar el robot. Quizás ya lo tienen hecho del ejercicio 2 anterior. Además, conectar el control del joystick en el puerto USB disponible de la computadora. Seguido, ingresen el siguiente comando:


El controlador *game pad*.

```
>>> gamepad()
```

En respuesta a este comando, obtendrán un texto de ayuda impreso en la ventana IDLE mostrando lo que podría suceder si usted presionara varios botones del game pad. Si observan la imagen del controlador del game pad, notarán que tiene ocho (8) botones azules (numerados del  al 8 en la imagen), y un controlador de eje (el gran botón giratorio azul). El controlador de eje puede ser usado para mover el robot. Adelante, inténtenlo.

Presionando cada uno de los botones numerados, obtendrán diferentes comportamientos, algunos harán que el robot realice un beep, con otros, la computadora hablará o dirá cosas. El Botón #1 hará que el robot saque una foto de lo que está viendo a través de su cámara y lo mostrará en la pantalla de la computadora. El botón #8 abandonará el modo de control a través del game pad.

Tómense un tiempo para experimentar con varias de las funciones del control. Comprueben qué tan bien pueden manejar el robot para que vaya a distintos lugares, o siga una pared, o se traslade alrededor de algo (¡como ustedes!). También pueden ubicar al robot en un gran pedazo de papel, insertar una lapicera  en un portador de lapicera y luego trasladarlo alrededor para verlo haciendo garabatos. ¿Pueden escribir su nombre (o las iniciales)? Intenten con un patrón o con otras formas.

Sin crear un programa, ésta es una forma efectiva de controlar remotamente los movimientos de su robot. El siguiente ejercicio les pide que intenten enunciar comandos para que el robot se mueva.

4. **Explorar un poco más allá**

OK, ahora están por su cuenta. Inicien Python, importen Myro, conecten el robot y dénle instrucciones para moverse hacia delante, hacia atrás, para que gire hacia la derecha y a la izquierda y para que dé vueltas. Usen los comandos: forward (SPEED), backward (SPEED), turnLeft (SPEED), turnRight (SPEED), y rotate (SPEED). SPEED puede ser cualquier número entre –1.0...1.0. Estos y todos los comandos del robot se detallan en el Manual Myro de Referencia. Este sería un buen momento para revisar las descripciones de todos los comandos que se introducen en esta sección.

## Revisión Myro

`from myro import *`
Este comando importa todos los comandos de robots disponibles en la librería Myro. Usaremos esto cada vez que intentemos escribir programas que utilizan el robot.

`initialize(<PORT NAME>)`
`init(<PORT NAME>)`
Este comando establece una conexión de comunicación inalámbrica con el robot. El `<PORT NAME>` se determina al configurar el software durante la instalación. Generalmente es la palabra com seguida de un número. Por ejemplo, “com5”. Las dobles comillas (“) son esenciales y requeridas.

`beep(<TIME>, <FREQUENCY>)`
Hace que el robot realice un beep durante `<TIME>` segundos en la frecuencia especificada por `<FREQUENCY>`.

`getName()`
Da el nombre del robot.

`setName(<NEW NAME>)`
Establece el nombre del robot para `<NEW NAME>`. El nuevo nombre deberá estar encerrado en dobles comillas, sin espacios, y sin más de 16 caracteres de longitud. Por ejemplo: setName (“Bender”).

`gamepad()`
Permite el control manual de varias funciones del robot y puede ser usada para trasladar al robot.

## Revisión Python

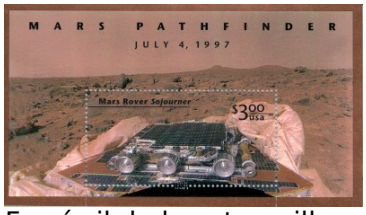
`Start Python.pyw`
Este es el ícono sobre el cual deben hacer un doble clic para iniciar un Python Shell (IDLE).

`>>>`
El mensaje Python. Aquí es donde se tipea el comando Python.

Nota: Todos los comandos que tipean (incluyendo los comandos Myro listados arriba) son comandos Python. Más adelante, en esta sección listaremos aquellos comandos que son parte del lenguaje Python.

## Ejercicios

1. ¿De dónde proviene la palabra robot? Exploren la etimología de las palabras robot y robótica y escriban un pequeño texto sobre el tema.
2. ¿Cuáles son las Leyes de Asimov sobre robótica? Escriban un ensayo con su punto de vista sobre ellos.
3. Busquen las entradas de Wikipedia sobre robots, así como la sección en Temas de IA –Inteligencia Inteligencia Artificial- (ver enlaces arriba). Escriban un pequeño ensayo sobre el actual estado de los robots.
4. Escriban un pequeño ensayo sobre un robot (real o ficcional) de su elección. Basado en lo que han aprendido de su lectura, evalúen sus capacidades.
5. Spirit y Opportunity no fueron los primeros rovers en aterrizar en Marte. El 4 de julio de 1997, el Pathfinder Mars aterrizó en Marte con una carga que incluía el rover Sojourner. El Servicio Postal de los Estados Unidos creó la estampilla que se muestra aquí para conmemorar su aterrizaje. ¡Este es quizás el primer robot real en aparecer en una estampilla postal! Averigüen lo que puedan acerca de la misión del Mars Pathfinder y comparen el rover Sojourner con el Spirit y el Opportunity.



Facsímil de la estampilla postal del Mars Pathfinder.

6. A través de los ejercicios, han experimentado un subconjunto de las capacidades del robot Scribbler y los tipos de tareas que puede hacerle llevar a cabo.
7. Insertar una lapicera en el puerto de lapicera del robot. Ubicar al robot en una superficie en la que se pueda escribir o dibujar. Manejar al robot con el controlador (joystick). Garabateará en los papeles a medida que se mueve. Observen sus garabatos moviéndolo hacia delante y hacia atrás. ¿Traza su recorrido de manera exacta? ¿Por qué o por qué no?
8. Al utilizar la operación del game pad, hagan que su robot escriba su nombre en el piso. Puede resultarles una tarea complicada por varios motivos. En lugar de eso, intenten que el robot escriba sus iniciales. Además, intenten guiar al robot para dibujar una estrella de cinco puntas. Esta tarea, en cierto sentido, no es muy diferente de controlar un robot para realizar una cirugía. Investiguen acerca de las capacidades de los robots de cirugía actuales y escriban un pequeño texto sobre el tema.
9. Utilizando el control del game pad, dibujen el logo Bluetooth (ver imagen) usando una lapicera insertada en el robot Scribbler. Realicen una búsqueda sobre Harald Blåtand y lean más acerca de los alfabetos rúnicos.

**Harald Blåtand Gormson**
¿Qué hay en un nombre?

El logo Bluetooth deriva de las letras del alfabeto rúnico H y B yuxtapuestas. HG para Harald Blåtand a Scandinavia King (del siglo 10mo AC), quien devino en leyenda por unificar Dinamarca y Noruega. La tecnología inalámbrica que hoy utilizamos se llama en su honor (Blåtand quiere decir “Bluetooth” –en español: diente azul) porque la tecnología en sí fue desarrollada por Ericsson, una compañía escandinava. La tecnología está diseñada para unificar a las computadoras y los dispositivos de telefónica. Los dispositivos Bluetooth son generalmente encontrados en teléfonos celulares. Lo estamos utilizando aquí para permitir la comunicación entre su robot y la computadora.

## Más lecturas

1. Entradas de Wikipedia  a Robots (http://en.wikipedia.org/wiki/Robot).
2. Temas de IA (Inteligencia Artificial): Robots de la Asociación Americana para la Inteligencia Artificial (AAAI) (http://www.aaai.org/AITopics/tml/robots.html).
3. Los Robots Sociales son robots que interactúan con las personas y aprenden de ellas.  Aquí hay una entrevista con Cynthia Breazeal quien dirige al Robotic Life Group en el Laboratorio de medios del MIT).(http://www.pbs.org/saf/1510/features/breazeal.htm)
4. Visiten el Hall of Fame (pasillo de la fama) en línea y averigüen más acerca de los robots verdaderos y los ficcionales que se han incorporado en el mismo. (http://www.robothalloffame.org)

# 2 Robots Personales

Cada Pleo es autónomo. Sí, cada uno de ellos empieza la vida como un bebé Camarasaurus recién salido del huevo, pero ahí es donde termina lo predecible y comienza la individualidad. Como cualquier criatura, Pleo siente hambre y fatiga provocadas por poderosas urgencias de explorar y de ser alimentado. Él pastará, tomará su siesta y andará por su cuenta –¡cuando lo desee! El dinosaurio Pelo puede cambia de opinión y de humor, al igual que ustedes.

*Fuente: www.pleoworld.com*

La mayoría de las personas asocian a la revolución de la computadora personal (la PC) con los ’80, pero la idea de una computadora personal ha estado dando vueltas casi tanto como las propias computadoras. Hoy en casi todos los colegios hay más computadoras personales que gente. La meta del proyecto One Laptop per Child (Una Laptop Por Niño- OLPC) es “proveerles a los niños del mundo nuevas oportunidades para explorar, experimentar y expresarse” (ver www.laptop.org). De igual manera, los robots personales fueron concebidos hace varias décadas. Sin embargo, la “revolución” de los robots está todavía en pañales. La imagen de arriba muestra los robots Pleo que se han diseñado para emular el comportamiento de un Camarasaurus infantil. Los Pleo se venden básicamente como juguetes o como “mascotas” mecánicas. En estos días, los robots están siendo utilizados para una gran variedad de situaciones para realizar un rango diverso de tareas: cortar el césped, pasar la aspiradora o fregar el piso, como entretenimiento, como compañía para mayores, etc. ¡El rango de aplicaciones posibles para robots hoy en día está limitado sólo por nuestra imaginación! Como ejemplo, los científicos japoneses han desarrollado una foca bebé que está siendo utilizada con fines terapéuticos para cuidar a pacientes internos.

The Paro Baby Seal Robot.
Photo courtesy of National Institute of Advanced Industrial Science and Technology, Japan (paro.jp).

El robot Scribbler es su robot personal. En este caso, se usa como un robot educativo para aprender acerca de robots y computadoras. Como ya han visto, el Scribbler es un rover, un robot que se mueve por el entorno. Este tipo de robots adquirió preponderancia en los últimos años y representa una nueva dimensión de las aplicaciones en robots. Los robots itinerantes han sido utilizados para repartir correo en grandes oficinas y como aspiradoras en los hogares: pueden rodar como pequeños vehículos (como una cortadora de césped, Roomba, Scribbler, etc.), y aún deambular sobre dos, tres o más piernas (por ej. Pleo). El robot Scribbler se mueve sobre tres ruedas, dos de las cuales tienen potencia. En este capítulo, conoceremos al Scribbler con mayor detalle y también aprenderemos acerca del uso de sus comandos y el control de su comportamiento.

## El Robot Scribbler Robot: Movimientos

En el último capítulo pudieron usar el robot Scribbler a través de Myro para realizar movimientos simples. Pudieron iniciar el software Myro, conectar con el robot, y luego le hicieron realizar un beep, le dieron un nombre y lo movieron con un joystick. Insertándole una lapicera en el porta lapicera, el Scribbler puede trazar un rastro de sus movimientos en un papel ubicado en el piso. Sería una buena idea revisar estas tareas para refrescar la memoria antes de avanzar con más detalles sobre cómo controlar al Scribbler.

Si lo sostienen al Scribbler en sus manos y lo observan, notarán que tiene tres ruedas. Dos de estas ruedas (las grandes en cada lado) están potenciadas por motores. Adelante, hagan girar las ruedas para sentir la resistencia de los motores. La tercera rueda (atrás) es una rueda libre que está allí como soporte únicamente. Todos los movimientos que hace el Scribbler están controlados por las dos ruedas impulsadas por motores. En Myro hay varios comandos para controlar los movimientos del robot. El comando que directamente controla los dos motores es el comando motors:

```
motors(LEFT, RIGHT)
```

En el comando de arriba, LEFT y RIGHT pueden cualquier valor dentro del rango [1.0... 1.0] y estos valores controlan los motores izquierdo y derecho, respectivamente. La especificación de un valor negativo moverá a los motores/ruedas hacia atrás, y uno negativo los moverá hacia delante. Por lo tanto, el comando:

```
motors(1.0, 1.0)
```

hará que el robot se mueva hacia delante a toda velocidad, y el comando:

```
motors(0.0, 1.0)
```

Detendrá el motor izquierdo y hará que el motor derecho se mueva a toda velocidad, lo cual producirá que el robot doble hacia la izquierda. De esta manera, generando una combinación de valores de motor de izquierda y derecha, pueden controlar los movimientos del robot.

Myro también ha provisto un conjunto de comandos de movimiento más utilizados que son fáciles de recordar y de usar. Algunos se listan abajo:

forward(SPEED)
backward(SPEED)
turnLeft(SPEED)
turnRight(SPEED)
stop()

Otras versiones de estos comandos tienen un segundo argumento, cierto período de tiempo en segundos:

forward(SPEED, SECONDS)
backward(SPEED, SECONDS)
turnLeft(SPEED, SECONDS)
turnRight(SPEED, SECONDS)

Si se le provee un número de SECONDS (segundos) en los comandos de arriba, se le estará especificando durante cuánto tiempo quiere que se desarrolle ese comando. Por ejemplo, si quisieran que su robot atravesara un camino cuadrado, generarían la siguiente secuencia de comandos:

```
forward(1, 1)
turnLeft(1, .3)
forward(1, 1)
turnLeft(1, .3)
forward(1, 1)
turnLeft(1, .3)
forward(1, 1)
turnLeft(1, .3)
```

Por supuesto que el hecho de que obtengan o no un cuadrado dependerá de cuánto dobla el robot en 0.3 segundos. No hay una forma directa de pedirle al robot que doble

exactamente 90 grados, o que se mueva cierta distancia especificada (digamos 2 ½ pies). Volveremos sobre esto.

También pueden utilizar los siguientes comandos de movimientos para trasladarse (por ejemplo, mover hacia delante o hacia atrás) o rotar (doblar hacia la derecha o hacia la izquierda):

```
translate(SPEED)
rotate(SPEED)
```

Adicionalmente, pueden especificar en un solo comando, la cantidad de traslación o rotación que desean utilizar:

```
move(TRANSLATE_SPEED, ROTATE_SPEED)
```

En todos estos comandos, `SPEED` (la velocidad) puede ser un valor entre `[-1.0...1.0]`

Probablemente podrán notar que en la lista de arriba hay comandos redundantes (por ejemplo, se pueden especificar varios comandos para un mismo movimiento). Esto viene de diseño. Pueden elegir el conjunto de comandos de movimientos que les parezcan más convenientes. Sería una buena idea probar estos comandos con su robot.

**Realicen la siguiente actividad**: Iniciar Myro, conectar con el robot, e intentar los siguientes comandos de movimientos en el Scribbler:

Primero, deben asegurarse de tener suficiente espacio frente al robot (ubicarlo en el suelo, con varios pies de espacio abierto frente a él.

```
>>> motors(1, 1)
>>> motors(0, 0)
```

Observen el comportamiento del robot. En particular, observen si se mueve (o no) en una línea recta después de emitir el primer comando. Pueden generar el mismo comportamiento en el robot con los siguientes comandos:

```
>>> move(1.0, 0.0)
>>> stop()
```

Adelante, pruébenlos. El comportamiento debería ser exactamente igual. A continuación, intenten hacer que el robot vaya hacia atrás usando cualquiera de estos comandos:

```
motors(-1, -1)
move(-1, 0)
backwards(1)
```

Nuevamente, observen el comportamiento de cerca. En los rovers el movimiento preciso, como trasladarse en una línea recta, es difícil de conseguir. Esto se debe a que dos motores independientes controlan los movimientos del robot. Para mover al robot hacia delante o hacia atrás en una línea recta, los dos motores deberían realizar exactamente la misma potencia en ambas ruedas. Mientras que esto es posible técnicamente, hay varios factores que pueden contribuir a desfasar la rotación de las ruedas. Por ejemplo, pequeñas diferencias en el montaje de las ruedas, diferentes resistencias del suelo en cada lado, etc. Esto no es necesariamente algo malo o indeseable en este tipo de robots. Bajo circunstancias similares, aún las personas son incapaces de moverse en líneas rectas precisas. Para ilustrar este punto, prueben el experimento mostrado a la derecha.

**¿Los humanos caminan en línea recta?**

Encuentren un pasillo largo y vacío y asegúrense de tener algún amigo que los ayude en esto. Parénse en el centro del pasillo y marquen el punto en el que están. Mirando hacia delante, caminen 10-15 pasos sin mirar el piso. Deténganse, marquen el punto de arribo y observen si han caminado en línea recta.

A continuación, regresen al punto de partida y realicen el mismo ejercicio con los ojos cerrados. Asegúrense de que su amigo esté allí para advertirles en caso de que se estén por chocar contra un objeto o una pared. Nuevamente observen si han

Para la mayoría de las personas, el experimento de arriba resultará en un movimiento variable. A menos que realmente se concentren intensamente en caminar en una línea recta, lo más probable es que desplieguen la misma variabilidad que el Scribbler. Caminar en una línea recta requiere de una retroalimentación y ajustes constantes, algo a lo que los humanos son muy adeptos. Es difícil que los robots hagan esto. Por suerte, trasladarse como el rover no requiere de movimientos tan precisos.

**Realicen la siguiente actividad:** Revisar todos los otros comandos de movimientos e intentarlos en el Scribbler. Al realizar esta actividad, es posible que se encuentren enunciando los mismos comandos repetidamente (o variaciones simples de los mismos). IDLE les provee una forma conveniente de repetir comandos previos (ver el Tip a continuación).

**Tip IDLE**

Pueden repetir un comando previo usando la función del historial del comando:

**ALT-p** recupera el comando previo

**ALT-n** recupera el siguiente (usar **CTRL-p** y **CTRL-**n en MACs)

Al presionar nuevamente **ALT-p** les dará el comando previo a ese y así sucesivamente. También podrán moverse hacia delante en la historia del comando presionando ALT-n repetidamente. También pueden cliquear en el cursor sobre cualquier comando previo y presionar ALT-ENTER para repetir ese comando.

## Definir nuevos comandos

Probar comandos simples interactivamente en IDLE es una linda manera de conocer las funcionalidades básicas del robot. Seguiremos utilizando esto cada vez que querramos intentar algo nuevo. Sin embargo, hacer que el robot lleve a cabo comportamientos más complejos requiere de varios comandos. Tener que tipearlos una y otra vez mientras el robot está operando puede resultar tedioso. Python les provee una manera conveniente de empaquetar una serie de comandos en un comando nuevo llamado *función*. Por ejemplo, si queremos que el Scribbler se mueva hacia delante y hacia atrás (como un yoyó), podemos definir un nuevo comando (función) llamado yoyo, de la siguiente manera:

```
>>> def yoyo():
        forward(1)
        backward(1)
```

La primera línea define el nombre de un comando (función) nuevo que será yoyo. Las líneas que siguen tienen una ligera sangría que contiene los comandos que hacen al comportamiento yoyo. Es decir, actuar como un yoyo, moverse hacia delante y después hacia atrás y luego parar. La sangría es importante y es parte de la sintaxis de Python. Se asegura que todos los comandos dentro de la sangría son parte de la definición del nuevo comando. Ampliaremos este tema más adelante.

Una vez definido el nuevo comando, pueden probarlo, ingresando el comando en IDLE, como se muestra abajo:

```
>>> yoyo()
```

**Realizar la siguiente actividad:** Si tienen un Scribbler listo, intenten la nueva definición de arriba; en primero lugar, deben conectar el robot y luego ingresar la definición de arriba. Notarán que ni bien tipean la primera línea, automáticamente IDLE genera una sangría en la(s) siguiente(s). Luego de ingresar la última línea, presionen un RETURN extra para terminar la definición. Esto define un nuevo comando en Python.

Observen el comportamiento del robot cuando le dan el comando `yoyo()`. Quizás vayan a necesitar repetir el comando varias veces. El robot se mueve momentáneamente y luego se detiene. Si observan cuidadosamente, notarán que se mueve hacia delante y hacia atrás.

En Python se pueden definir funciones utilizando la sintaxis `def` como se muestra arriba. Observen también que la definición de una nueva función no quiere decir que los comandos de esa función se lleven a cabo. Deben explicitar el comando para que esto ocurra. Esto es útil porque les da la posibilidad de usar la función una y otra vez (como hicieron arriba). Enunciar la función de esta manera, en Python se denomina invocación. A medida que se invoca, todos los comandos que hacen a la función se ejecutan en la secuencia dentro de la cual fueron listados en la definición.

¿Cómo podemos hacer que el comportamiento `yoyo` del robot sea más pronunciado? Es decir, hacer que se mueva hacia delante por, digamos, un segundo, y luego hacia atrás por un segundo, y que luego se detenga. Pueden usar la opción SECONDS (segundos) en los comandos de adelantarse y retroceder, como se muestra abajo:

```
>>> def yoyo():
        forward(1, 1)
        backward(1, 1)
        stop()
```

El mismo comportamiento también puede ser llevado a cabo utilizando el comando `wait` (espera) que se utiliza de la siguiente manera:

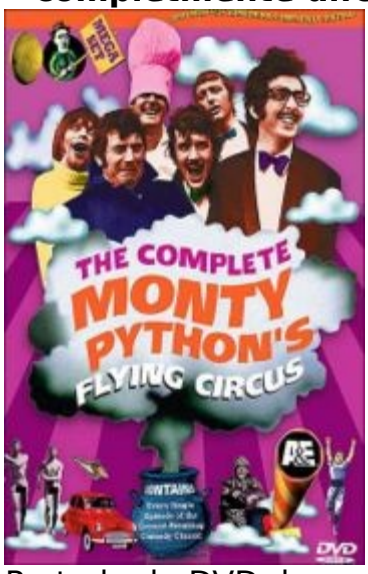
```
wait(SECONDS)
```

donde `SECONDS` especifica la cantidad de tiempo que el robot espera antes de moverse hacia el siguiente comando. En efecto, el robot continúa haciendo lo que sea que se le ha pedido que haga justo antes del comando `wait` durante la cantidad de tiempo especificado en el comando `wait`. Esto significa que si al robot se le pidió que se moviera hacia delante y luego se le pidió que esperara un segundo, se moverá hacia delante durante un segundo antes de que se aplique el comando que sigue al `wait`. Aquí hay una definición completa del yoyo que usa el comando `wait`:

```
>>> def yoyo():
        forward(1)
        wait(1)
        backward(1)
        wait(1)
        stop()
```

**Realizar la siguiente actividad:** Adelante, prueben las nuevas definiciones, exactamente como se muestran arriba, e ingresen el comando en el Scribbler. ¿Qué pueden observar? En ambos casos, deberían ver al robot moverse hacia delante durante un segundo, seguido por un movimiento hacia atrás durante un segundo y luego parar.

**Y ahora algo completmente diferente**




Portada de DVD de http:Wikipedia.com
IDLE es el nombre del programa de edición y Python shell, Cuando hacen doble clic en **Start Python** en realidad están iniciando IDLE. Python es el nombre del lenguaje que estaremos usando, su nombre proviene de Monty Python's Flying Circus (el circo volador de Monty Python). IDLE supuestamente son las siglas para Interactive Development Environment (Entorno de Desarrollo Interactivo), pero ¿saben a quién puede estar homenajeando también?

## Agregar parámetros a los comandos

Observen la definición de la función `yoyo` arriba y notarán el uso de paréntesis, (), cuando se define la función así como también cuando se utiliza. También han usado otras funciones anteriormente con paréntesis y probablemente han adivinado tu utilidad. Los comandos o funciones pueden especificar ciertos parámetros (o valores) cuando se los ubica entre paréntesis. Por ejemplo, todos los comandos de movimiento, con al excepción del `stop`, tienen uno o más números que pueden especificarse para indicar la velocidad del movimiento. El número de segundos

**Tip Scribbler:**

Recuerden que su Scribbler funciona con baterías que con el tiempo se agotarán. Cuando las baterías empiezan a descargarse, el Scribbler puede empezar a hace movimientos erráticos. Eventualmente dejará de responder. Cuando las baterías se gastan, la luz roja LED empieza a parpadear. Esta es la señal para cambiar las baterías.

que desean que el robot espere pueden especificarse como un parámetro en la invocación del comando `wait`. De igual manera, podrían haber elegido especificar la velocidad del movimiento de adelantarse y retroceder en el comando yoyo, o la cantidad de tiempo de espera. Debajo les mostramos tres definiciones del comando yoyo que usan estos parámetros:

```
>>> def yoyo1(speed):
        forward(speed, 1)
        backward(speed, 1)

>>> def yoyo2(waitTime):
        forward(1, waitTime)
        backward(1, waitTime)

>>> def yoyo3(speed, waitTime):
        forward(speed, waitTime)
        backward, waitTime)
```

En la primera definición, `yoyo1`, especificamos la velocidad del movimiento de adelantarse y retroceder como un parámetro. Al usar esta definición, pueden controlar la velocidad del movimiento con cada invocación. Por ejemplo, si quisieran que se moviera a una velocidad media, podrían enunciar el comando:

```
>>> yoyo1(0.5)
```

De modo similar, en la definición de yoyo2 hemos parametrizado el tiempo de espera. En el último caso, hemos parametrizado ambos: la velocidad y el tiempo de espera. Por ejemplo, si quisiéramos que el robot se moviera a velocidad media y durante un segundo y medio cada vez, usaríamos el comando:

```
>>> yoyo3(0.5, 1.5)
```

De este modo, podemos personalizar los comandos individuales con diferentes valores, lo cual da como resultado distintas variaciones del comportamiento yoyo. Noten que en las tres definiciones de arriba no usamos el comando `stop()` en absoluto. ¿Por qué?

## Guardar comandos nuevos en módulos

Como pueden imaginarse, mientras trabajan en los distintos comportamientos del robot, seguramente terminarán con una gran colección de funciones nuevas. Tendría sentido que no tuvieran que tipear las definiciones una y otra vez. Python les permite definir funciones nuevas y guardarlas en archivos en carpetas de su computadora. Cada archivo se llama *módulo* y puede ser usado fácilmente una y otra vez. Ilustremos esto definiendo dos comportamientos: un comportamiento yoyo parametrizado y un comportamiento de meneo que hace que el robot se menee hacia la izquierda y la derecha. Las dos definiciones se presentan a continuación:

```
# Archivo: moves.py
```

```
# Purpose: Two useful robot commands to try out as a module.

# Primero importamos myro y nos conectamos con el robot

from myro import *
init()

# Definimos las nuevas funciones

def yoyo(speed, waitTime):
    forward(speed)
    wait(waitTime)
    backward(speed)
    wait(waitTime)
    stop()

def wiggle(speed, waitTime):
    rotate(-speed)
    wait(waitTime)
    rotate(speed)
    wait(waitTime)
    stop()
```

Todas las líneas que comienzan con un signo ‘#’ se llaman comentarios. Son simplemente anotaciones que pueden ayudarnos a entender y registrar los programas en Python.

Pueden ubicar estos comentarios en cualquier lugar, incuso después de un comando. El signo # claramente marca el comienzo de un comentario y cualquier cosa que venga después en esa línea no será interpretado como un comando por la computadora. Es bastante útil y podemos usar libremente los comentarios en nuestros programas.

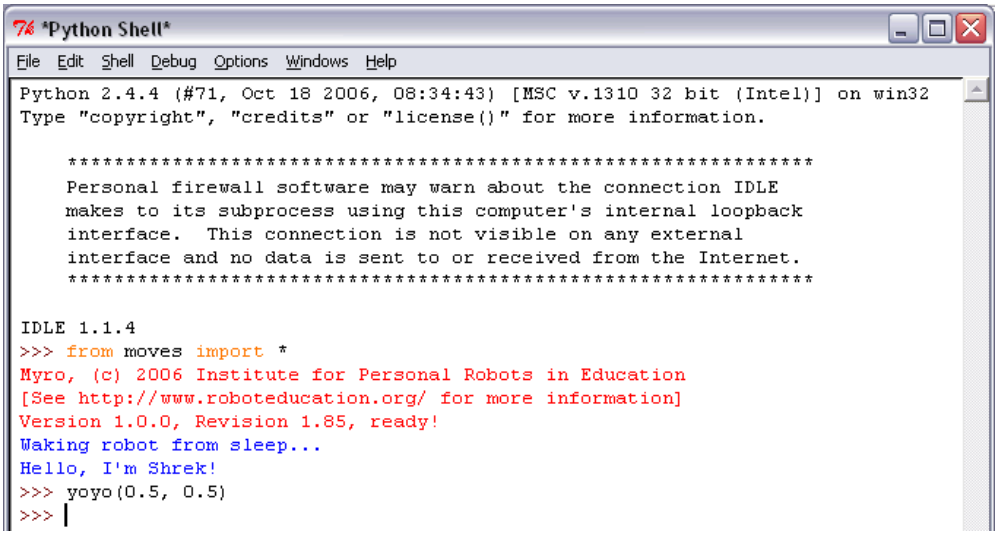
Noten que hemos agregado los comandos `import` e `init` arriba. El comando `init` siempre le pedirá que entre el número de com-port (puerto).

**Realicen la siguiente actividad**: Para guardar los comportamientos yoyo y wiggle como módulos en un archivo, pueden pedirle a IDLE una nueva ventana para el menú de archivos (file). Luego, ingresen el texto que contenga las dos definiciones y guárdenlas en un archivo (llamémoslo moves.py) en su carpeta Myro (el mismo lugar donde tienen el ícono de Start Python). Todos los módulos Python terminan con la extensión “`.py`” y deberían asegurarse de que siempre los han guardado en la misma carpeta que el archivo Start Python.pyw. Esto les facilitará (y también a IDLE) localizar los módulos cuando los quieran usar.

Una vez que han creado el archivo, siempre hay dos maneras de utilizarlo. En IDLE, solamente ingresen el comando:

```
>>> from moves import *
```

y luego intenten cualquiera de los dos comandos. El siguiente ejemplo muestra cómo usar la función yoyo después de importar el módulo `moves`:




Como pueden ver arriba, el acceso a los comandos definidos en un módulo es similar al acceso de las capacidades del módulo `myro`. Esta es una buena característica de Python. En Python, se les incentiva a que extiendan las capacidades de cualquier sistema definiendo sus propias funciones, guardándolas en módulos y luego importándolas. Por lo tanto, importar del módulo `moves` no es en absoluto distinto de importar del módulo myro. En general, el comando Python import tiene dos características que especifica: el nombre del módulo; y lo que se está importando del mismo. La sintaxis precisa se describe abajo:

```
from <MODULE NAME> import <SOMETHING>
```

donde <MODULE NAME> es el nombre del módulo del cual están importando y <SOMETHING> (Algo) especifica el comando/las capacidades que están importando.

Especificando un * para <SOMETHING>, están importando todo lo que está definido en el módulo. Volveremos sobre esto en el curso. Pero por el momento, noten que diciendo:

```
from myro import *
```

Están importando todo lo que está definido en el módulo myro. Todo lo que está definido en este módulo está listado y documentado en el Manual de Referencia Myro. Lo bueno que brinda esta facilidad es que ahora pueden definir sus propios grupos de comandos que extienden los comandos básicos disponibles en Myro para personalizar el comportamiento del su robot. Usaremos esto una y otra vez en el curso.

## Las funciones como construcciones de bloques

Ahora que han aprendido cómo definir comandos nuevos usando los que ya existen, es hora de discutir un poco más acerca de Python. La sintaxis básica para definir una función Python tiene la siguiente forma:

```
def <FUNCTION NAME>(<PARAMETERS>):
    <SOMETHING>
```

```
    ...
    <SOMETHING>
```

Esto significa que para definir una nueva función, se empieza por usar la palabra def seguida del nombre de la función (<FUNCTION NAME>) seguida de <PARMETERS> encerrados entre paréntesis y seguidos de dos puntos (:). A esta línea le siguen los comandos que conforman la definición de la función (<SOMETHING>...<SOMETHING>). Cada comando debe ser ubicado en una línea aparte, y todas las líneas que conforman la definición deben tener la misma alineación de sangría. El número de espacios de la sangría no es importante, mientras que sean los mismos. Esto puede parecer extraño y un poco restrictivo al comienzo, pero ya verán su valor. Primero, ayuda a la lectura de la definición (o de las definiciones). Por ejemplo, miren la siguiente definición para la función yoyo:

```
def yoyo(speed, waitTime):
    forward(speed)
       wait(waitTime)
    backward(speed)
       wait(waitTime)
  stop()

def yoyo(speed, waitTime):
   forward(speed); wait(waitTime)
   backward(speed); wait(waitTime)
   stop()
```

La primera definición no será aceptada por Python, como se muestra abajo:

```
>>> def yoyo(speed, waitTime):
      forward(speed)
       wait(waitTime)
      backward(speed)
        wait(waitTime)
    stop()

SyntaxError: invalid syntax
>>>
```

Reporta que hay un error de sintaxis y resalta la ubicación del error con el grueso cursor rojo (ver la tercera línea de la definición). Esto se debe a que Python refuerza estrictamente la regla descripta arriba. La segunda definición, sin embargo, es aceptable. Por dos motivos: la sangría es consistente; y los comandos en la misma línea pueden separarse por punto y coma (;). Recomendamos continuar ingresando cada comando en una línea separada en lugar de usar el punto y coma como separador, hasta que estén más cómodos con Python. Es importante destacar que IDLE les ayuda a igualar las sangrías, poniendo una sangría automática en la línea siguiente si es necesario.

Otra característica que tiene incorporada IDLE que facilita la legibilidad de los programas Python es el uso de resaltados de color. Noten que en los ejemplos de arriba (en los que usamos vistas de pantalla de IDLE), algunos fragmentos del programa aparecen en diferentes colores. Por ejemplo, la palabra def en la definición de una función aparece en rojo, el nombre de la función, yoyo, aparece en azul. Otros colores también se usan en situaciones diversas, es'ten atentos. IDLE presenta todas las palabras Python (como def) en rojo y todos los nombres que ustedes han definido, en azul.

La idea de definir funciones nuevas usando las ya existentes es muy poderosa y es central para la computación. Al definir la función yoyo como una nueva función utilizando las funciones existentes (forward, backward, wait, stop), han abstraído un nuevo comportamiento para su robot. Pueden definir funciones de más alto nivel que utilicen yoyo si lo desean. Por este motivo, las funciones sirven como construcciones básicas de bloques para definir varios comportamientos del robot. Como ejemplo, consideren un nuevo comportamiento del robot: uno que lo haga comportarse como yoyo dos veces, seguido de un meneo doble. Pueden hacerlo definiendo una función nueva, como se muestra a continuación:

```
>>> def dance():
       yoyo(0.5, 0.5)
       yoyo(0.5, 0.5)
       wiggle(0.5, 1)
       wiggle(0.5, 1)

>>> dance()
```

**Realizar la siguiente actividad**: Adelante, agreguen la función dance a su módulo moves.py. Prueben el comando dance en el robot. Ahora tienen un comportamiento simple que hace que el robot haga una pequeña danza.

## Guiado por controles automatizados

En apartados anteriores coincidimos en que un robot es “un mecanismo guiado por controles automatizados”. Pueden observar que al definir funciones que llevan a cabo funciones más complejos, pueden crear módulos para distintos tipos de comportamientos. Los módulos construyen el programa que escriben, y al ser invocados en el robot, el robot realiza el comportamiento especificado. A medida que aprendan más acerca de las capacidades del robot y cómo acceder a las mismas a través de funciones, podrán diseñar y definir muchos tipos de comportamientos automatizados.

## Resumen

En este capítulo, han aprendido varios comandos que hacen que un robot se mueva de distintas maneras. También aprendieron cómo definir nuevos comandos a través de la definición de nuevas funciones Python. Las funciones sirven como bloques de construcción básicos en computación y para definir comportamientos del robot nuevos y más complejos. Python tiene reglas de sintaxis específicas para escribir definiciones. También aprendieron cómo guardar sus definiciones de funciones en un archivo para luego usarlos como un módulo desde el cual importar. Mientras aprendían algunos comandos de robot muy simples, también aprendieron algunos conceptos de computación que permiten la construcción de comportamientos más complejos. Aunque los conceptos son simples, representan un mecanismo muy poderoso y fundamental utilizado en casi todos los desarrollos de software. En capítulos posteriores, proveeremos más detalles sobre la escritura de funciones y también sobre cómo estructurar parámetros para personalizar invocaciones de funciones individuales. Asegúrense de realizar algunos o todos los ejercicios en este capítulo para revisar estos conceptos.

## Revisión de Myro

`backward(SPEED)`
Mueve hacia atrás a cierta velocidad (valor en el rango de –1.0...1.0).

`backward(SPEED,SECONDS)`
Mueve hacia atrás a cierta velocidad (valor en el rango de –1.0...1.0) durante un tiempo determinado en segundos, luego se detiene.

`forward(SPEED)`
Mueve hacia delante a cierta velocidad (valor en el rango de –1.0...1.0).

`forward(SPEED,TIME)`
Mueve hacia adelante a cierta velocidad (valor en el rango de –1.0...1.0) durante un tiempo estipulado en segundos, luego se detiene.

`motors(LEFT,RIGHT)`
Gira el motor izquierdo a la izquierda a velocidad y el motor derecho a velocidad derecha (valor en el rango de –1.0...1.0).

`move(TRANSLATE, ROTATE)`
Mueve a las velocidades de traslado y rotación (valor en el rango de –1.0...1.0).

`rotate(SPEED)`
Rota a cierta velocidad (valor en el rango de –1.0...1.0). Los valores negativos rotan hacia la derecha (a favor de las agujas del reloj) y los valores positivos rotan hacia la izquierda (contra las agujas del reloj).

`stop()`
Detiene el robot.

`translate(SPEED)`
Mueve en una línea recta a cierta velocidad (valor en el rango de –1.0...1.0). Los valores negativos especifican el movimiento hacia atrás y los positivos el movimiento hacia delante.

`turnLeft(SPEED)`
Dobla a la izquierda a cierta velocidad (valor en el rango de –1.0...1.0).

`turnLeft(SPEED,SECONDS)`
Dobla a la izquierda a cierta velocidad (valor en el rango de –1.0...1.0) durante cierta cantidad determinada de segundos y luego se detiene.

`turnRight(SPEED)`
Dobla a la derecha a cierta velocidad (valor en el rango de –1.0...1.0)

`turnRight(SPEED,SECONDS)`
Dobla a la derecha a cierta velocidad (valor en el rango de –1.0...1.0) durante cierta cantidad de segundos y luego se detiene.

`wait(TIME)`
Pausa durante la cantidad determinada de segundos. El tiempo puede ser un número decimal.

## Revisión Python

```
def <FUNCTION NAME>(<PARAMETERS>):
    <SOMETHING>
    ...
```

```
    <SOMETHING>
```

Define una nueva función llamada <FUNCTION NAME> (nombre de función). El nombre de una función siempre debe empezar con una letra y puede ser seguido por cualquier secuencia de letras, números o guiones bajos, y no debe contener espacios. Traten de elegir nombres que describan apropiadamente la función que se está definiendo.

## Ejercicios

**1.** Comparen los movimientos del robot en los comandos turnLeft (1), turnRight (1) y rotate (1) y rotate (-1). Observen detenidamente el comportamiento del robot y luego también prueben los comandos de motor:

```
>>> motors(-0.5, 0.5)
>>> motors(0.5, -0.5)
>>> motors(0, 0.5)
>>> motors(0.5, 0)
```

Notan alguna diferencia en los comportamientos de giro? Los comandos rotate hacen que el robot doble dentro de un radio equivalente al ancho del robot (la distancia entre la rueda derecha e izquierda). El comando turn hace que el robot gire en el mismo lugar.

2. Inserten una lapicera en el puerto de lapicera del Scribbler y luego denle el comando de ir hacia delante durante 1 o más segundos y luego hacia atrás la misma cantidad de tiempo. ¿El robot viaja la misma distancia? ¿Atraviesa el mismo recorrido? Registren sus observaciones.

3. Midan el largo de la línea dibujada por el robot en el ejercicio 2. Escriban una función travel(DISTANCE- distancia) para hacer que el robot viaje determinada DISTACIA. Pueden usar centímetros o pulgadas como unidad. Testeen la función en el robot varias veces para ver cuán precisa es la línea.

4. Supongamos que quisieran hacer doblar/girar el robot cierta cantidad de grados, digamos 90. Antes de darle el comando al robot, háganlo ustedes. Es decir, deténganse en un punto, dibujen una línea que divida sus dos pies y luego doble 90 grados. Si no tienen ningún modo de medir, sus giros serán aproximados. Pueden estudiar el comportamiento del robot de igual manera, impartiendo los comandos de doblar/girar y haciendo que esperen cierta cantidad de tiempo. Intenten estimar la cantidad de tiempo de espera requerido para doblar 90 grados (ustedes deberán determinar la velocidad – speed) y escribir una función para girar esa cantidad de tiempo. Usando esta función, escriban un comportamiento del robot para recorrer un cuadrado en el piso (pueden insertar una lapicera para ver cómo sale el cuadrado).

5. Generalicen el tiempo de espera obtenido en el ejercicio 3 y escriban una función llamada degreeTurn (DEGREES -grados). Cada vez que se convoca, hará que el robot gire los grados especificados. Usen esta función para escribir un conjunto de instrucciones para dibujar un cuadrado..

6. Usando las funciones travel y degreeTurn, escriban una función para dibujar el logo Bluetooth (Ver Capítulo 1, Ejercicio 9).

7. Coreografíen una rutina de danza simple para su robot y definan las funciones para llevarla a cabo. Asegúrense de dividir las tareas en movimientos re-utilizables y establezcan parámetros para los movimientos (lo más que puedan) para que puedan ser usados en forma personalizada en distintos pasos. Usen la idea de construcción de

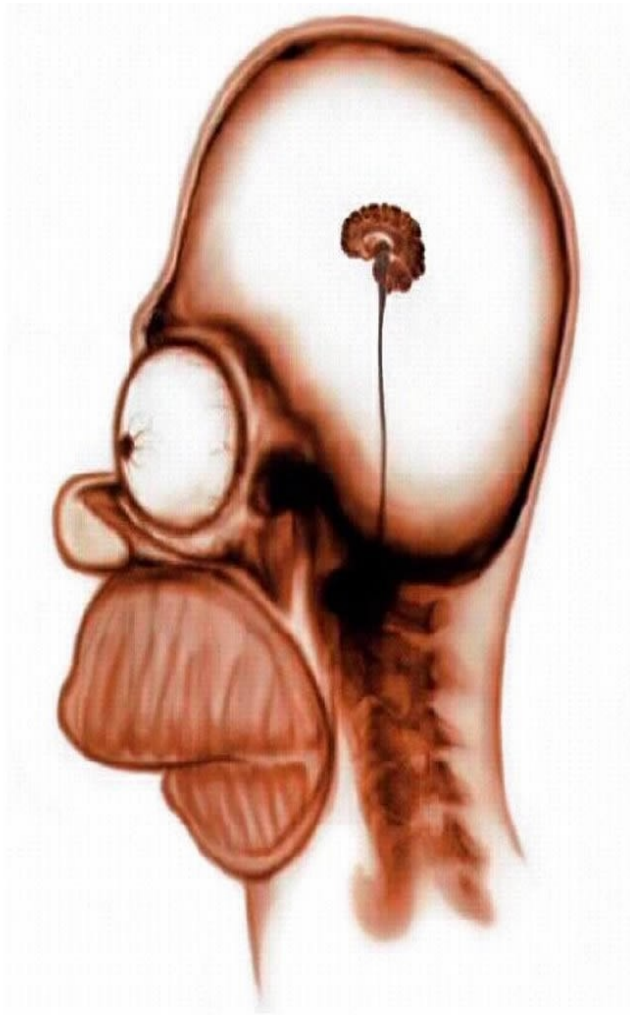
bloques para construir más y más complejos movimientos de danza. Asegúrense de que la rutina dure por lo menos varios segundos y que incluya por lo menos dos repeticiones en la secuencia completa. También pueden usar el comando beep que aprendieron en la sección previa para incorporar algunos sonidos a su coreografía.

8. Graben un video de la danza del robot y luego mézclenlo con una pista musical a elección. Usen el software de edición que más accesible les resulte. Publiquen el video en línea en sitios como YouTube para compartir con amigos.

9. Los robots que cortan el césped y aún los que pasan la aspiradora pueden usar determinados movimientos coreografiados para asegurarse de que provean una cobertura completa del área a la cual están brindando su servicio. Si asumimos que el área a la cual hay que cortarle el césped y aspirar es rectangular sin obstrucciones, ¿podrían diseñar un comportamiento para el Scribbler para que cubra toda el área? Descríbanlo por escrito. [Ayuda: piensen en cómo realizan estas acciones ustedes].

# 3

# Construyendo Cerebros de Robot



*Qué espléndida cabeza, aunque sin cerebro.*
Esopo (620 AC-560 AC)

Página opuesta: Cerebro de Homero Simpson
La foto es cortesía de The Simpson's Trivia (www.simpsonstrivia.com)

Si creen que su robot es una criatura que actúa en el mundo, entonces, al programarla, están esencialmente construyéndole un cerebro a esta criatura. El poder de las computadoras reside en el hecho de que la misma computadora o el robot pueden ser provistos de un programa o cerebro diferente para hacer que se comporten como una criatura diferente. Por ejemplo, un programa como Firefox o Explorer harán que la computadora se comporte como un navegador Web. Pero si cambian al “Media Player” (reproductor de medios), la computadora de comportará como un DVD o reproductor de CD. De la misma manera, el robot se comportará de manera diferente de acuerdo a las instrucciones en el programa sobre el cual le han pedido que funcione. En este capítulo aprenderemos acerca de la estructura de los programas Python y sobre cómo pueden organizar diferentes comportamientos de robots a través de programas.

El mundo de los robots y el de las computadoras, como han podido ver hasta ahora, están intrínsecamente conectados. Han estado usando una computadora para conectarse al robot y luego lo han controlado a través de comandos. La mayoría de los comandos que han usado hasta ahora provienen de la librería Myro que está escrita especialmente para controlar fácilmente a los robots. El lenguaje de programación que estamos usando para programar al robot es Python. Python es un lenguaje de programación de uso general. Con esto queremos decir que uno puede usar Python para crear software que controle la computadora u otro dispositivo como un robot a través de esa computadora. De esta manera, al aprender a escribir programas para robots, también están aprendiendo cómo programar computadoras. Nuestro viaje en el mundo de los robots está por lo tanto intrínsecamente ligado con el mundo de las computadoras y con la computación. Continuaremos entretejiendo conceptos relacionados con los robots y con las computadoras a través de este viaje. En este capítulo, aprenderemos más acerca de los programas de robots y de computadoras y sobre su estructura.

## La estructura básica del cerebro de un robot

La estructura básica de un programa Python (o del cerebro de un robot) se muestra abajo:

```
def main():
   <do something>
   <do something>
   ...
```

Esto es esencialmente lo mismo que definir una función nueva. De hecho, acá estamos adoptando la convención de que todos nuestros programas que representen cerebros de robots se llamarán main. En general, la estructura de los programas de su robot se mostrará como se detalla abajo (hemos provisto números de líneas –line- para que puedan hacer referencia a las mismas):

```
Line 1: from myro import *
Line 2: init()

Line 3: <any other imports>
Line 4: <function definitions>
Line 5: def main():
Line 6:     <do something>
Line 7:     <do something>
```

```
Line 8:     ...

Line 9: main()
```

Todo programa de cerebro de robot comenzará con las primeras dos líneas (Line 1 y Line 2). Estas, como habrán visto, importan la librería Myro y establecen la conexión con el robot. En el caso de estar usando otras librerías, las importarán (esto se muestra en Line 3). A esto le siguen las definiciones de las funciones (Line 4), y luego la definición de la función main. Finalmente, la última línea (Line 9) es una invocación de la función main. Esta se ubica de tal manera que al cargar el programa en el Python Shell, el programa comenzará a ejecutarse. Para ilustrar esto, escribamos un programa de robot que le hace hacer una pequeña danza usando los movimientos de yoyo y wiggle definidos en el capítulo anterior.

```
# File: dance.py
# Purpose: A simple dance routine

# First import myro and connect to the robot

from myro import *
initialize("com5")

# Define the new functions...

def yoyo(speed, waitTime):
       forward(speed, waitTime)
       backward(speed, waitTime)

def wiggle(speed, waitTime):
       motors(-speed, speed)
       wait(waitTime)
       motors(speed, -speed)
       wait(waitTime)
       stop()

# The main dance program
def main():
       print "Running the dance routine..."
       yoyo(0.5, 0.5)
       wiggle(0.5, 0.5)
       yoyo(1, 1)
       wiggle(1, 1)
       print "...Done"

main()
```

Hemos usado un nuevo comando Python en la definición de la función main: el comando print. Este comando imprimirá el texto encerrado entre las comillas dobles (“) cuando se corra el programa. Este programa no es muy diferente de la función dance definida en el capítulo anterior, excepto que acá estamos usando un movimiento de giro para hacerlo menear (wiggle). Sin embargo, en lugar de llamar a esta función dance, la llamamos main. Como dijimos antes, esta es sólo una convención que adoptamos para nombrar que nos permite identificar el principal programa en el archivo de programa.

**Realizar la siguiente actividad**: Para hacer correr este programa en el robot, pueden iniciar IDLE, crear una ventana nueva, entrar allí el programa, guardarlo como un archivo (dance.py) y luego seleccionar la opción Run Module en el menú de Run de

Window. Como alternativa, para hacer correr este programa, pueden ingresar el siguiente comando en el Python Shell:

```
>>> from dance import *
```

Básicamente, esto es equivalente a la opción de Run Module descripta arriba. Al correr el programa, notarán que el robot realiza la rutina de danza especificada en el programa main. También observen los dos mensajes que se imprimen en la ventana IDLE. Estos son los resultados del comando print. print es un comando muy útil en Python porque puede ser usado para mostrar básicamente cualquier cosa que se le pida. Mientras están en esta sesión, cambien el comando print y hagan lo siguiente:

```
speak("Running the dance routine")
```

speak es un comando Myro que permite la salida de habla desde la computadora. Cambien el otro comando print por un comando speak y prueben el programa. Una vez realizado esto, entren otros comandos de speak en el mensaje IDLE. Por ejemplo:

```
speak("Dude! Pardon me, would you have any Grey Poupon?")
```

[Traducción en español: “¡Chabón! Perdoname, ¿no tendrías un poco de pimienta?”]

La facilidad del habla está incorporada en la mayor parte de las computadoras hoy en día. Más adelante veremos cómo pueden averiguar qué otras voces están disponibles y también cómo cambiarlas.

## Hablando en Pythonés

Los hemos lanzado al mundo de las computadoras y de los robots sin darles realmente una introducción formal al lenguaje Python. En esta sección, les brindaremos más detalles acerca del lenguaje. Lo que saben hasta ahora de Python es lo necesario para controlar al robot. Los comandos del robot que tipean están integrados a Python a través de la librería Myro. Python viene con varias otras librerías o módulos útiles que probaremos y aprenderemos en este curso. Si necesitan acceder a los comandos provistos por la librería, todo lo que tienen que hacer es importarlos.

Las librerías en sí mismas están hechas en gran medida por grupos de funciones (pueden contener otras entidades pero lo desarrollaremos después). Las funciones proveen los bloques básicos de construcción para cualquier programa. En general, un lenguaje de programación (y Python no es una excepción) incluye un grupo de funciones predefinidas y un mecanismo para definir funciones adicionales. En el caso de Python, es la construcción def. Ya han visto varios ejemplos de definiciones de funciones, y ya han escrito algunas por su cuenta a esta altura. En la construcción def, cuando se define una nueva función, hay que darle un nombre a la misma.. Los nombres son un componente importante de la programación y Python tiene ciertas reglas sobre cómo se forma un nombre.

## ¿Qué contiene un nombre?

Un nombre en Python debe comenzar con una letra del alfabeto (a-z o A-Z) o con un underscore (por ej._) y se puede seguir con cualquier secuencia de letras, dígitos o guiones bajos. Por ejemplo,

```
iRobot
myRobot
jitterBug
jitterBug2
my2cents
my_2_cents
```

Son todos ejemplos de nombres Python válidos. Además, otra parte importante de la sintaxis de los nombres es que Python son “case sentitive” (sensibles a mayúsculas y minúsculas). Esto significa que los nombres myRobot y MyRobot son nombres distintos para Python. Una vez que han nombrado algo de determinada manera, deben usar consistentemente la misma forma de deletrearlo y el uso de mayúsculas o minúsculas. Bueno, eso en cuanto a la sintaxis de los nombres. La pregunta que se pueden estar haciendo es ¿qué cosas pueden (o deben ser) nombradas?

Hasta acá, han visto que los nombres pueden ser usados para representar funciones. Esto significa que lo que un robot hace cada vez que se usa un nombre de una función (como yoyo) está especificado en la definición de esa función. Entonces, al darle un nombre a una función, tienen un modo de definir funciones nuevas. Los nombres también pueden ser usados para representar otras cosas en el programa. Por ejemplo, quizás quieran representar cantidad, por ejemplo en velocidad o el tiempo, por un nombre. De hecho, lo hicieron en la definición de yoyo que también se muestra abajo:

```
def yoyo(speed, waitTime):
      forward(speed, waitTime)
      backward(speed, waitTime)
```

Las funciones pueden tomar parámetros que ayudan a personalizar lo que pueden hacer. En el ejemplo de arriba, pueden formular los dos comandos siguientes:

```
>>> yoyo(0.8, 2.5)
>>> yoyo(0.3, 1.5)
```

El primer comando le está pidiendo que lleve a cabo el comportamiento yoyo a la velocidad de 0.8 por 2.5 segundos, mientras que el segundo está especificando 0.3 y 1.5 para velocidad y tiempo, respectivamente. De esta manera, al parametrizar la función con estos dos valores, pueden producir resultados similares pero con variaciones. Es una idea similar a la de las funciones matemáticas: sine(x) por ejemplo, computa el seno de lo que sea que reemplacen por x. Sin embargo, tiene que haber una manera de definir la función en primer lugar que la haga independiente de los valores del parámetro específico. Ahí es donde entran los nombres. En la definición de la función yoyo han nombrado dos parámetros (el orden en que se listan es importante): speed y waitTime. Luego han usado esos nombres para especificar el comportamiento que hace a esa función. Esto significa que los comandos forward y backward usan los nombres speed y waitTime para especificar cualquier velocidad y tiempos de espera que se incluyan en la invocación de la función. De esta manera, los nombres speed y waitTime representan o designan valores específicos en este programa Python.

En Python, los nombres pueden representar funciones tanto como valores. Ustedes son libres de usar el nombre que quieran. Es una buena idea usar nombres que sean fáciles de leer, de tipear y también que designen apropiadamente la entidad que representan. El nombre que elijan para designar una función o un valor en su

programa es muy importante para ustedes. Por ejemplo, tendría sentido que nombraran a una función turnRight (doblar a la derecha) de manera tal que cuando se invoque, el robot girara a la derecha. No tendría sentido que en lugar de a la derecha, doblara a la izquierda, o peor aún, que hiciera los movimientos equivalentes a la danza del yoyo. Pero mantener esta consistencia semántica dependerá completamente de ustedes.

## Valores

En la última sección vimos que los nombres designan funciones tanto como valores. Es posible que la importancia de nombrar a las funciones a esta altura les resulte obvia, pero designar nombres a los valores es una característica aún más importante en la programación. Al nombrar valores, podemos crear nombres que representan valores específicos, como la velocidad del robot, o la temperatura alta promedio en el mes de diciembre en la cima del Materhorn en Suiza, o el valor actual del índice de la bolsa de Dow Jones, o el nombre del robot, etc. Los nombres que designan valores también se llaman variables. Python provee un mecanismo simple para designar variables con nombres:

```
speed = 0.75
aveHighTemp = 37
DowIndex = 12548.30
myFavoriteRobot = "C3PO"
```

Los valores pueden ser números o strings (cadenas de caracteres: cualquier cosa encerrada entre comillas, “). Arriba pueden ver ejemplos de sentencias de asignación en Python. La sintaxis exacta para esta sentencia se muestra a continuación:

```
<nombre de variable> = <expresión>
```

Deberían leer el enunciado de arriba como: que a la variable llamada <nombre de variable>  se le asigne el valor que sea el resultado del cálculo de la expresión <expresión>. ¿Y qué es <expresión>? Estos son algunos ejemplos:

```
>>> 5
5
>>> 5 + 3
8
>>> 3 * 4
12
>>> 3.2 + 4.7
7.9
>>> 10 / 2
5
```

Lo que tipean en el mensaje Python (>>>) en realidad se llama una expresión (expression). La expresión más simple que pueden tipear es un número (como se muestra arriba). Un número se da el valor a sí mismo. Esto significa que un 5 es un 5, ¡como debería ser! Y 5 + 3 es 8. Como pueden ver, cuando ingresan una expresión, Python la evalúa y muestra el resultado. También se puede usar la suma (+), la resta

(-), la multiplicación (*) y la división (/) en los números para formar expresiones que involucran números.

También habrán notado que los números pueden ser escritos como números enteros (3, 10, 1655673, etc.) o incluyendo puntos decimales (3.2, 0.5, etc.). Python (y casi todos los lenguajes de computación) puede distinguirlos. Los números enteros se llaman integers y los que tienen puntos decimales se llaman números floating point. Mientras que algunas operaciones aritméticas se definen con los dos tipos de números, hay algunas diferencias que deberían advertir. Observen los ejemplos abajo:

```
>>> 10.0/3.0
3.3333333333333335
>>> 10/3
3
>>> 1/2
0
>>> 1.0/2
0.5
```

Cuando dividen un número de punto flotante  por otro número punto flotante, obtienen un resultado punto flotante. Sin embargo, cuando dividen un entero por otro entero, obtienen un resultado entero. Verán que en el ejemplo de arriba, obtienen el resultado 3.3333333333333335 al dividir 10.0 por 3.0, pero obtienen 3 cuando dividen 3 por 10. Al saber esto, el resultado de la división de 1 por 2 (ver arriba), que da cero (0) no debería sorprenderlos. Esto significa que, aunque la operación parezca la misma (/), trata a los enteros de manera diferente que a los valores punto flotante. Sin embargo, si por lo menos uno de los números en una operación aritmética es un número punto flotante, Python les dará un resultado punto flotante (ver el último ejemplo de arriba). Deben tener esto en cuenta. Veremos más sobre números más adelante. Antes de volver a los robots, veamos rápidamente el tema de los strings.

Las computadoras se llaman así porque eran sobresalientes realizando cálculos. Sin embargo, en estos días, las computadoras son capaces de manipular todo tipo de entidades: textos, imágenes, sonidos, etc. Un texto está hecho de letras o caracteres y strings que son simplemente secuencias de caracteres. Python requiere que los strings se encierren entre comillas: que podrían ser simples (‘Soy un string’), dobles (“¡Yo también!”), e incluso triples (‘‘‘¡Yo también soy un string!’’’). El hecho de tratar a un string como un valor es una característica poderosa de Python. Python también provee operaciones con el uso de strings que les permite algunas expresiones string poderosas. Estos son algunos ejemplos:

```
>>> mySchool = "Bryn Mawr College"
>>> yourSchool = "Georgia Institute of Technology"
>>> print mySchool
Bryn Mawr College

>>> print yourSchool
Georgia Institute of Technology

>>> print mySchool, yourSchool
Bryn Mawr College Georgia Institute of Technology

>>> yourSchool+mySchool
'Georgia Institute of TechnologyBryn Mawr College'

>>> print yourSchool+mySchool
```

```
Georgia Institute of TechnologyBryn Mawr College
```

Presten especial atención a los últimos dos ejemplos. La operación +  es definida en los strings y da como resultado la concatenación de las dos strings. El comando print está seguido de un cero o más expresiones Python, separadas por comas. print evalúa todas las expresiones e imprime los resultados en la pantalla. Como han visto antes, este es un modo conveniente de imprimir los resultados o mensajes de su programa.

## Un programa de cálculo

OK, dejen su robot a un lado por unos minutos. Ya han aprendido lo suficiente de Python para escribir programas que realicen cálculos simples. Aquí hay un problema sencillo:

> *E1 1 de enero de 2008, se estimaba que la población del mundo era de aproximadamente 6.650 billones de personas. Se pronostica que con las tasas de crecimiento de la población, tendremos más de 9 billones de personas para el 2050. Una estimación del crecimiento de la población define el crecimiento anual a +1.14% (ha sido hasta de +2.2% en el pasado). Dados estos datos, ¿pueden estimar cuánto se incrementará la población mundial este año (2008)? ¿Y también cuánto se incrementará por día?*

Para responder a estas preguntas, todo lo que deben hacer es calcular 1.14% de 6.650 billones para obtener el incremento de la población de este año. Si dividen ese número por 365 (el número de días de 2009) obtendrán el incremento diario promedio. Pueden usar una calculadora para hacer estos cálculos simples. También pueden utilizar Python para hacerlo de dos maneras posibles. Pueden simplemente usar la calculadora como se muestra abajo:

```
>>> 6650000000*1.14/100.0
75810000.0
>>> 75810000.0/365.0
207131.1475409836
```

Esto significa que este año habrá un incremento de 75.81 millones en la población mundial, lo que implica un incremento diario promedio de más de 207 mil personas. ¡Así que ya conocen el resultado!

Intentemos escribir un programa para que realice los cálculos de arriba. Un programa para hacer los cálculos obviamente implicará más trabajo. ¿Para qué hace el trabajo extra si ya conocemos el resultado? Se necesitan pequeños pasos para llegar a lugares más altos. Así que démosnos el gusto y veamos cómo pueden escribir un programa Python para hacer esto. Abajo, les damos una versión:

```
#File: worldPop.py
# Purpose:
#     Estimate the world population growth in a year and
#     also per day.
#     Given that on January 1, 2008 the world's population was
#     estimated at 6,650,000,000 and the estimated growth is
#     at the rate of +1.14%
def main():
```

```
    population = 6650000000
    growthRate = 1.14/100.0
    growthInOneYear = population * growthRate
    growthInADay = growthInOneYear / 365
    print "World population on January 1, 2008 is", population
    print "By Jan. 1, 2009, it will grow by", growthInOneYear
    print "An average daily increase of", growthInADay
main()
```

El programa sigue la misma estructura y las mismas convenciones que vimos arriba. En este programa, no estamos usando librerías (no las necesitamos). Hemos definido variables con los nombres population y growthRate para designar a los valores dados en la descripción del problema. También hemos definido las variables growthInOneYear y growthInADay y las usamos para designar los resultados del calculo. Finalmente, usamos los comandos print para imprimir los resultados del cómputo.

**Realizar la siguiente actividad:** Iniciar Python, ingresar el programa y hacerlo correr (de la misma manera que harían correr el programa del robot) y observen los resultados. ¡Voilá! ¡Ahora están bien encaminados para también aprender las técnicas básicas de computación! En este sencillo programa no importamos nada, ni tuvimos la necesidad de definir una función. Aunque este es un programa trivial, sin embargo, debe servir para convencerlos de que escribir programas para realizar cómputos es esencialmente lo mismo que controlar al robot.

**El Problema de la energía**

La raíz de la causa del problema de energía es el crecimiento de l población mundial y el consumo de energía per cápita.

¿Cuánta gente más puede soportar la Tierra? La mayoría de los expertos estiman que el límite para la sustentabilidad a largo plazo es de entre 4 y 16 billones.

Fuente: Science, Policy& The Pursuit of Sustainability, Edited by Bent Orr,and Baker. illus. by Shetter. Island Press, 2002

## El uso de Input

El programa que escribimos arriba utiliza valores específicos de la población mundial y la tasa de crecimiento. Por lo tanto, este programa resuelve solamente un problema específico para los valores dados. ¿Qué pasa si quisiéramos calcular los resultados para distintas tasas de crecimiento? ¿O si no, para otra población estimada? ¿Qué pasa si quisiéramos probar el programa para variar las cantidades de ambos? Tal programa sería mucho más útil y podría ser reutilizado una y otra vez. Observen que el programa comienza asignándole valores específicos a las dos variables:

```
population = 6650000000
growthRate = 1.14/100.0
```

Algo que podrían hacer es simplemente modificar esas dos líneas para reflejar los distintos valores. Sin embargo, los típicos programas son mucho más complicados que este y pueden requerir varios valores diferentes para resolver un problema. Cuando los programas se hacen más extensos, no es una buena idea modificarlos para cada instancia de problema específico, es más bien deseable hacerlos útiles para todas las instancias de problemas. Una manera de realizar esto es utilizando las facilidades Input de Python. todos los programas de computación toman algún input (entrada de datos) para realizar algún cómputo (o algo), y luego producen algún output (salida de datos). Python tiene un comando input

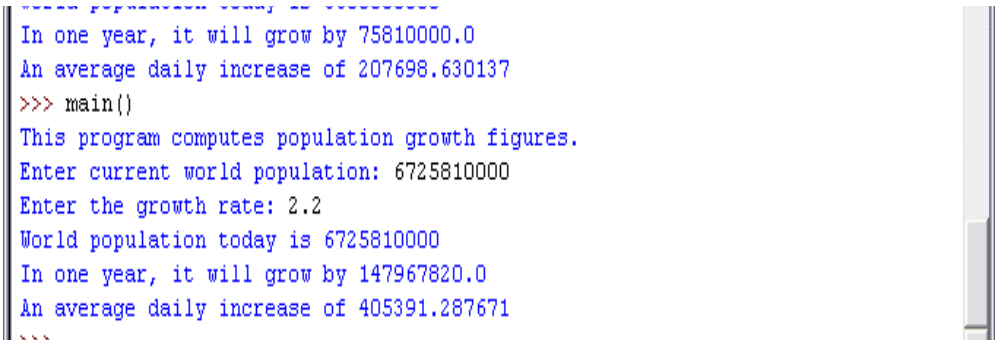
simple que puede ser usado para reescribir el programa encima, como se muestra a continuación:

```
#File: worldPop.py
# Purpose:
#     Estimate the world population growth in a year and
#     also per day.
#     Given that on Jnauray 1, 2008 the world's population was
#     estimated at 6,650,000,000 and the estimated growth is
#     at the rate of +1.14%
def main():
    # print out the preamble
    print "This program computes population growth figures."
    # Input the values
    population = input("Enter current world population: ")
    growthRate = input("Enter the growth rate: ")/100.0
    # Compute the results
    growthInOneYear = population * growthRate
    growthInADay = growthInOneYear / 365
    # output results
    print "World population today is", population
    print "In one year, it will grow by", growthInOneYear
    print "An average daily increase of", growthInADay
main()
```

Lean cuidadosamente el programa de arriba. Observen que hemos agregado comentarios adicionales además de enunciados impresos. Esto mejora la legibilidad global así también como la interacción con este programa. Observen el uso de los enunciados de print arriba. La sintaxis básica de input se muestra abajo:

```
<variable name> = input(<some prompt string>)
```

Esto significa que input es una función cuyo parámetro es un string y el valor que devuelve es el valor de la expresión que será ingresada por el usurario. Cuando se ejecuta, la computadora imprimirá el mensaje y esperará que el usuario ingrese una expresión Python. El usuario podrá ingresar cualquier expresión como respuesta al mensaje y luego presionar la tecla RETURN o ENTER. La expresión entonces es evaluada por Python y el valor resultante es devuelto por la función input. El valor entonces es asignado a la variable <variable name>. El enunciado de arriba usa la misma sintaxis que el enunciado asignado descrito arriba. Python ha hecho que resulte fácil obtener input de un usuario al definir input como una función. Ahora, observen el uso de la función input en el programa de arriba. Con este cambio, ahora tenemos un programa más general que puede ser corrido una y otra vez. Abajo, les mostramos dos ejemplos del programa funcionando:




Observen como pueden volver a correr el programa simplemente con tipear el nombre de la función main (). Hay otras formas de obtener input en Python. Veremos esto más adelante.

## Cerebros de robot

Escribir programas para controlar al robot, entonces, no es diferente de escribir un programa para realizar un cómputo. Ambos siguen la misma estructura básica. La única diferencia es que todos los programas de robot que escriban usarán la librería Myro. Habrá varios programas de robot que le requerirán obtener input del usuario (ver los ejercicios abajo). Pueden usar la función input como se describe arriba.

Una característica que distinguirá a los programas de robots de esos que solamente realizan cómputos, es el tiempo que lleva correr el programa. En general, un programa que solo realiza un cómputo, terminará ni bien se completa el cómputo Sin embargo, habrá casos en que la mayor parte del tiempo, el programa de robot le requerirá al robot que realiza una acción una y otra vez. Aquí aparece una pregunta interesante:

**Pregunta**: ¿Cuánto tiempo le toma a un robot-aspiradora aspirar una habitación de 16 ft X12 ft? (ft. es la abreviación de feet, que significa pies).

Parece una pregunta trivial, pero si lo piensan un poco más, puede implicar cuestiones más profundas. Si la habitación no tiene ningún obstáculo en ella (una habitación vacía), el robot puede planear aspirar la habitación empezando por una esquina para luego recorrer la extensión de la pared, luego girar levemente para separarse de la pared y avanzar en la otra dirección. De esta manera, finalmente llegará al otro lado de la habitación de una manera sistemática, y podría detenerse al llegar a la última esquina. Esto es similar al modo en que uno cortaría el pasto en un jardín rectangular y chato, incluso es similar al modo en que se cosecharía la siembra o en que se volvería a llenar de hielo una pista de hielo sobre hockey, usando una máquina Zamboni. Para responder a la pregunta formulada arriba, solamente deben calcular la distancia total realizada y la velocidad promedio del robot-aspiradora y usar ambos datos para computar el tiempo estimado que llevaría. Pero ¿qué ocurriría si la habitación tiene muebles y otros objetos?

Podrían intentar modificar la estrategia de pasado de aspiradora delineada arriba, pero entonces, no habría garantía de que el piso quede completamente aspirado. Quizás se verían tentados a rediseñar la estrategia de aspiración para permitir movimientos al azar y luego estimar (basándose en la velocidad promedio del robot) que después de una cantidad generosa de tiempo, se asegurarán que la habitación estará completamente limpia. Se sabe (y esto lo veremos formalmente el otro capítulo) que

los movimientos al azar llevados a cabo un largo período de tiempo terminar brindando una cobertura uniforme y casi completa. Inherentemente, esto también implica que un mismo lugar puede terminar siendo aspirado muchas veces (¡lo cual no es necesariamente algo malo!). Esto es similar a pensar que si se deja pastando un rebaño de ovejas en una colina, después de un período de tiempo, quedará una altura de pasto uniforme (piensen en las bellas colinas de Wales).

Desde una perspectiva más práctica, el iRobot Roobma usa una estrategia más avanzada (aunque está basada en el tiempo) para asegurarse de que se realice una cobertura completa. Una pregunta interesante (e importante) que podríamos preguntar sería:

**Pregunta**: ¿Cómo sabe el robot-aspiradora que ha terminado de limpiar la habitación?

La mayoría de los robots están programados para detectar ciertas situaciones de terminación o funcionan sobre una base de tiempo. Por ejemplo, funcionar durante 60 minutos y luego detenerse. Detectar situaciones es un poco más difícil y volveremos sobre ese tema en el próximo capítulo.

Hasta ahora, han programado comportamientos de robot muy simples. Cada comportamiento, que es definido por una función, al ser invocado, hace que el robot haga algo durante una cantidad de tiempo predeterminada; por ejemplo, el comportamiento yoyo del último capítulo al invocarse como:

```
>>> yoyo(0.5, 1)
```

Provocaría que el robot hiciera algo por alrededor de 2 segundos (un segundo para ir hacia delante y luego un segundo para retroceder). En general, el tiempo transcurrido realizando el comportamiento yoyo dependerá del valor del segundo parámetro provisto para la función. Por lo tanto, la invocación era:

```
>>> yoyo(0.5, 5.5)
```

El robot se movería durante un total de 11 segundos. De forma similar, el comportamiento dance definido en el capítulo anterior durará un total de seis segundos. De esta manera, el comportamiento total del robot depende directamente del tiempo que le lleve ejecutar todos los comandos que conforman el comportamiento. Saber cuánto durará un comportamiento ayudará a pre-programar el tiempo total de duración del comportamiento completo. Por ejemplo, si quisieran que el robot realizara los movimientos de danza durante 60 segundos, podrían repetir el comportamiento dance diez veces. Pueden hacer esto simplemente formulando el comando dance 10 veces. Pero nos resulta tedioso repetir el mismo comando tantas veces. Las computadoras están diseñadas para llevar a cabo tareas repetitivas. De hecho, la repetición es uno de los conceptos clave en computación y todos los programas de computación, incluyendo Python, proveen formas simples de especificar repeticiones de todo tipo.

## Realizar repeticiones en Python

Si quisieran repetir el comportamiento dance diez veces, todo lo que tienen que hace es:

```
for i in range(10):
   dance()
```

Este es una nueva sentencia en Python: la sentencia **for**. También se lo llama sentencia loop (o de repetición). Es una manera de repetir algo durante una cantidad fija de veces. La sintaxis básica para un for  en Python es:

```
for <variable> in <sequence>:
   <do something>
   <do something>
   ...
```

La sentencia for comienza con la palabra clave for, seguida por una <variable> y luego la palabra clave in y una <sequence> (secuencia) seguida de dos puntos (:). Esta línea determina el número de veces que se repetirá la repetición. Lo que sigue es un grupo de enunciados, con sangría (nuevamente, la sangría es importante), que se llaman block (bloque) y que forman el body (cuerpo) del loop (las cosas que se repiten).

Al ejecutarse, a la <variable> (que se denomina variable de repetición ) se le asignan valores sucesivos en la  <sequence>, y para cada uno de esos valores, se ejecutan los enunciados en el cuerpo del loop. Una  <sequence> en Python es una lista de valores. Las listas son centrales e Python y veremos varios ejemplos de estas listas más adelante. Por ahora, observen el ejemplo de danza arriba y noten que hemos usado la función range (10) para especificar la secuencia. Para ver lo que esta función hace, pueden iniciar IDLE e ingresarlo como una expresión:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

El resultado de ingresar el range (10) es una secuencia (una lista) de diez números 0..9. Observen cómo range deviene en una secuencia de valores comenzando del 0 hasta arriba, sin incluir el 10. De esta manera la variable i en el loop:

```
for i in range(10):
   dance()
```

tomará los valores del 0 al 9 y para cada uno de esos valores ejecutará el comando dance().

**Realizar la siguiente actividad**: Probemos esto con el robot. Modificar el programa del robot del inicio de este capítulo para incluir la función de danza y luego escribir un programa principal para usar el loop de arriba.

```
# File: dance.py
# Purpose: A simple dance routine

# First import myro and connect to the robot

from myro import *
init()

# Define the new functions...

def yoyo(speed, waitTime):
       forward(speed, waitTime)
```

```
       backward(speed, waitTime)
       stop()

def wiggle(speed, waitTime):
       motors(-speed, speed)
       wait(waitTime)
       motors(speed, -speed)
       wait(waitTime)
       stop()

def dance():
       yoyo(0.5, 0.5)
       yoyo(0.5, 0.5)
       wiggle(0.5, 1)
       wiggle(0.5, 1)

# The main dance program
def main():
       print "Running the dance routine..."

       for danceStep in range(10):
           dance()

       print "...Done"

main()
```

**Tip IDLE**

Pueden detener el programa en cualquier momento presionando las teclas CTRL-C (pronunciada como Control-Ce). Esto significa, presionando la tecla CTRL y al mismo tiempo la tecla-c.

En el caso de un programa de robot, esto también detendrá al robot.

Observen que hemos usado danceStep (un nombre más significativo que i) para representar el loop index variable (variable índice de loop). Cuando hacen correr este programa, el robot debería realizar la rutina de danza diez veces. Modifiquen el valor especificado en el comando range para probar algunos pasos nuevos. Si terminan especificando un valor muy grande, recuerden que por cada valor de danceStep, el robot realizará algo durante 6 segundos. De esta manera, si han especificado 100 repeticiones, el robot estará en acción durante 10 minutos.

Además de repetir a través del conteo, también pueden especificar la repetición usando el tiempo. Por ejemplo, si quisieran que el robot (o la computadora) realizaran algo durante 30 segundos. Pueden escribir el siguiente comando para especificar una repetición basada en tiempo:

```
while timeRemaining(10):
    <do something>
    <do something>
    ...
```

Los comandos de arriba se repetirán durante 10 segundos. De esta manera, si quisieran que la computadora dijera “¡Doh!” durante 5 segundos, pueden escribir:

```
while timeRemaining(5):
    speak("Doh!", 0)
```

Al escribir programas de robot, ¡también en que querrán que el robot siga llevando a cabo sus comportamientos por siempre! Mientras que técnicamente, con para siempre queremos decir eternamente, en realidad lo que seguramente ocurrirá es que o se

quede sin baterías, o ustedes decidan detenerlo (presionando CTRL-C). El comando Python para especificar esto utiliza un enunciado loop llamado un while-loop (loop-mientras) que puede ser escrito así:

```
while True:
   <do something>
   <do something>
   ...
```

True (verdad) también es un valor en Python (junto con False) sobre el cual aprenderemos más un poco más adelante. Por ahora será suficiente con decir que el loop de arriba está especificando que el cuerpo del loop ¡será ejecutado por siempre!

Realizar la siguiente actividad: Modificar el programa dance.py para usar cada uno de los while-loops en lugar de los for-loop. En el último caso (while TRUE: ) recuerden usar el CTRL-C para detener las repeticiones (y al robot).

Como hemos observado arriba, la repetición es uno de los conceptos clave en computación. Por ejemplo, podemos usar la repetición para predecir la población mundial dentro de diez años, repitiendo el cómputo de valores por cada año.

```
for year in range(10):
    population = population * (1 + growthRate)
```

Esto significa que repetidamente se agrega el incremento de población, basado en la tasa de crecimiento, diez veces.

Realizar la siguiente actividad: Modificar el programa worldPop.py para ingresar (input) la población actual, la tasa de crecimiento y el número de años de proyección hacia delante, y luego computar la población total resultante. Hagan correr el programa sobre varios valores diferentes.

(**Googleen**: “crecimiento de población mundial” para obtener los últimos números). Pueden estimar en qué momento la población mundial será de 9 billones?

## Resumen

Este capítulo introdujo la estructura básica de los programas de Python (y de robots). También aprendimos sobre nombres y valores en Python. Los nombres pueden ser usados para designar funciones y valores. Estos últimos también se llaman variables. Python provee varios tipos distintos de valores: integers (enteros), números floating point (punto flotante), strings, y también valores booleanps (True y False). La mayoría de los valores tienen operaciones incorporadas (como suma, resta, etc.) que realizan cálculos sobre los mismos. También se pueden formar secuencias de valores usando listas. Python provee facilidades incorporadas simples para obtener input del usuario. Todo esto nos permite escribir no sólo programas para el robot sino también programas que realizan cualquier tipo de cómputo. La repetición es central y quizás el concepto más útil en computación. En Python se pueden especificar repeticiones usando un for-loop o un while-loop. Estos últimos son útiles para escribir programas generales de cerebros de robot. En los próximos capítulos aprenderemos cómo escribir comportamientos de robot más sofisticados.

## Revisión Myro

speak(<something>)
La computadora convierte el texto en <something> (algo) para decir y lo emite en voz

alta. <something> simultáneamente también se imprime en pantalla. La generación de habla en voz alta se da en forma sincrónica. Esto significa que cualquier cosa que le siga al comando de habla se da sólo después de que la frase completa es emitida.

speak(<something>, 0)
La computadora convierte el texto en <something> para decir y lo emite en voz alta. <something> también se imprime simultáneamente en pantalla. La generación de habla en voz alta se da en forma asincrónica. Esto significa que la ejecución de los comandos subsiguientes pueden realizarse previamente a que el texto se emita en voz alta.

timeRemaining(<seconds>)
Este se usa para especificar repeticiones cronometradas en un while-loop (ver abajo).

## Revisión Python

### Valores
Los valores en Python pueden ser números (integers o números floating point) o strings. Cada tipo de valor puede ser usado solo en una expresión, o usando una combinación de operaciones definidas por ese tipo (por ejemplo +, -, *, /, %, para números). Los strings son considerados secuencias de caracteres (o letras).

### Nombres
Un nombre en Python debe comenzar con una letra alfabética (a-z o A-Z) o un underscore (por ej., _) y puede tener a continuación cualquier secuencia de letras, dígitos o letras minúsculas.

input(<prompt string>)

Esta function imprime <prompt string> en la ventana IDLE y espera a que el usuario ingrese una expresión Python. La expresión se evalúa y su resultado es devuelto como un valor de la función input.

```
from myro import *
initialize("comX")

<any other imports>
<function definitions>
def main():
    <do something>
    <do something>
    ...
main()
```

Esta es la estructura básica de un programa de control de robot en Python. Sin las primeras dos líneas, es la estructura básica de todos los programas Python.

print <expression1>, <expression2>, ...
Imprime los resultados de todas las expresiones en la pantalla (en la ventana IDLE). Pueden especificarse de ceno a más expresiones. Cuando no se especifica ninguna expresión, imprime una línea en blanco.

<variable name> = <expression>
Este es el modo en que Python le asigna un valor a las variables. El valor generado por <expression> se transformará en el nuevo valor de <variable name>.

range(10)
Genera una secuencia, una lista, de números desde 0..9. Hay otras versiones de esta función más generales. Estas se muestran abajo.

range(n1, n2)

Genera una lista de números desde el n1... (n2-1). Por ejemplo, ranges(5,10) generará la lista de números [5, 6, 7, 8, 9].

range(n1, n2, step)
Genera una lista de números desde el n1... (n2-1) en pasos de paso. Por ejemplo, range(5, 10, 2) generará la lista de números [5, 7, 9].

Repetición

```
for <variable> in <sequence>:
   <do something>
   <do something>
   ...
while timeRemaining(<seconds>):
    <do something>
    <do something>
    ...
while True:
   <do something>
   <do something>
   ...
```

Estos son modos diferentes de generar repeticiones en Python. La primera versión asignará a <variable> sucesivos valores en <sequence> y llevará a cabo el cuerpo una vez por cada valor de este tipo. La segunda versión llevará a cabo el cuerpo por <seconds> cantidad de tiempo. TimeRemaining es una función Myro (ver arriba). La última versión especifica una repetición que no termina.

## Ejercicios

1.- Escribir un programa Python para convertir una temperatura en grados Celsius a grados Fahrenheit. Aquí hay una muestra de interacción con este tipo de programa:

```
Enter a temperature in degrees Celsius: 5.0
That is equivalent to 41.0 degrees Fahrenheit.
```

La formula para convertir la temperatura de Celsius a Fahrenheit es: C/5=(F-32)/9, donde C es la temperatura en grados Celsius y F es la temperatura en grados Fahrenheit.

2.- Escribir un programa Python para convertir la temperatura de grados Fahrenheit a grados Celsius.

3. Escribir un programa para convertir una cantidad dada de dinero en dólares US a una cantidad equivalente en Euros. Busquen el cambio actual en la web (ver xe.com, por ejemplo).

4.- Modificar la versión del programa dance de arriba que usa un for-loop, utilizando el siguiente loop:

```
for danceStep in [1,2,3]:
    dance()
```

Esto significa que pueden usar una lista para especificar la repetición ( y los sucesivos valores que la variable loop tomará). Intenten de nuevo con la lista [3, 2, 6], o [3.5, 7.2, 1.45], o [“United”, “States”, “of”, “America”]. También intenten reemplazar la lista de arriba con el string “ABC”. Recuerden que los strings son también secuencias en Python. Aprenderemos más acerca de las listas más adelante.

5. Hagan correr el programa de población mundial (cualquier versión del capítulo) y cuando pida input, intenten ingresar lo siguiente y observen el comportamiento del programa. También, dado lo que han aprendido en este capítulo, intenten explicar el comportamiento resultante.

1. Usen los valores 9000000000, y 1.42 con valores de input como arriba. Pero cuando pida varios valores, ingrésenlos en cualquier orden. ¿Qué ocurre?

2. Usando los mismos valores que arriba, en lugar de ingresar el valor, digamos de 9000000000, ingresen 6000000000 + 3000000000, o 450000000*2, etc. ¿Notan alguna diferencia en el output?

3. Reemplacen cualquiera de los valores a imputarse por un string. Por ejemplo, ingresen “Al Gore”cuando les pida un número. ¿Qué ocurre?

6. Rescriban una solución para el Ejercicio 4 del capítulo anterior para usar la estructura de programa descripta arriba.

7. Se les han introducido las reglas para otorgar nombres en Python. Habrán notado que hemos hecho un uso extensivo de mixed case (mezcla de mayúsculas y minúsculas) al nombrar algunas entidades. Por ejemplo waitTime. Hay varias convenciones para nombrar usadas por los programadores y eso ha dado lugar a una interesante cultura en sí misma. Busquen la frase Came/Case controversy en su motor de búsqueda favorito para aprender acerca de convenciones para nombrar. Hallarán un interesante artículo sobre esto en Semicolon Wars (http://www.americanscientist.org/template/AssetDetail/assetid/51982) .

8. Experimenten con la función speak introducida en este capítulo. Intenten darle un número para que lo diga en voz alta (prueben con enteros y números reales). ¿Cuál es el número entero más alto que puede enunciar? ¿Qué ocurre cuando se excede ese límite? Intenten darle a la función speak una lista de números, o strings, o ambas.

9. Escriban un programa Python que cante la canción del ABC: ABCD...XYZ. Now I know my ABC’s. Next time won’t you sing with me? (se trata de una canción con la cual los chicos aprenden el abecedario en Estados Unidos).

# 4 Percibiendo desde adentro

*Página opuesta: Llama de vela*
*La foto es cortesía de Jon Sullivan (www.pdphoto.org)*

*Veo personas muertas.*
Cole Sear (por Haley Joel Osment) en *Sexto Sentido*,
M. Night Shyamalan, 1999.

Cole Sear, en la película "Sexto Sentido" de Shylamalan, no se refiere a los cuerpos muertos que están tirados frente a él (para los que no vieron la película). Los cinco sentidos con los que la mayoría de los humanos nos relacionamos son: el tacto, la visión, el equilibrio, la audición, el gusto o el olfato. En todos los casos nuestros cuerpos tienen receptores sensoriales especiales ubicados en varias partes del cuerpo que permiten la percepción. Por ejemplo, los receptores del gusto se concentran predominantemente en la lengua; los receptores del tacto son más sensibles en las manos y el rostro y menos en la espalda y las extremidades, aunque estén presentes en todo el cuerpo, etc. Además de las diferencias fisiológicas entre cada tipo de receptor, también hay diferentes vías neuronales y por lo tanto distintos mecanismos sensoriales en nuestros cuerpos. Funcionalmente, podemos decir que cada tipo de sistema sensorial comienza con los receptores que convierten aquello que perciben en señales electroquímicas que se transmiten sobre neuronas. Muchas de estas vías conducen a la corteza cerebral donde luego son procesadas (como “¡Guau, ese ají está picante!”). El sistema perceptivo de un organismo se refiere a un conjunto de receptores sensoriales, las vías neuronales y el procesamiento de la información perceptiva en el cerebro. El cerebro es capaz de combinar información sensorial de distintos receptores para crear experiencias más ricas que aquellas facilitadas por receptores individuales.

**La persepción**
Percibiendo desde adentro

Busquen algo verdaderamente delicioso para comer, como una galletita dulce, un pedazo de chocolate (¡lo que les guste!). Sosténganlo en la mano derecha y dejen que el brazo derecho cuelgue con naturalidad al costado. Ahora cierren los ojos, con los párpados bien apretados, e intenten comer lo que están sosteniendo. ¡Pan comido! (bueno, lo que sea que hayan elegido comer :-)

Dénse un momento para disfrutar de la delicia.

El sistema perceptivo de cualquier organismo incluye un conjunto de sensores *externos* (también llamados exteroreceptores) y algunos mecanismos sensoriales *internos* (interoreceptores o proprioreceptores). ¿Pueden tocar su ombligo en la oscuridad? Esto se debe a la propriocepción. El sistema sensorial del cuerpo también mantiene un registro del estado de las partes del cuerpo, cómo están orientadas, etc.

La percepción es un componente esencial del robot y cada robot viene construido con sensores internos y externos. No es poco frecuente, por ejemplo, encontrar sensores que son capaces de percibir la luz, la temperatura, el tacto, la distancia hacia otro objeto, etc. Un ejemplo de sensores internos es la medición del movimiento relativo del marco interno de referencia del robot. A veces también llamado *estimaciones muertas*, puede ser un mecanismo sensorial útil que pueden usar para diseñar comportamientos de robots.

Los robots usan sensores electromecánicos y hay dos tipos diferentes de dispositivos disponibles para percibir la misma cantidad física. Por ejemplo, un sensor común hallado en muchos robots es un sensor de *proximidad*. Éste detecta la distancia hacia un objeto o un obstáculo. Los sensores de proximidad pueden construirse usando diferentes tecnologías: luz infrarroja, sonoro e incluso láser. Dependiendo del tipo de tecnología utilizada, varía su precisión, su performance,

así como su costo: el infrarrojo (IR) es el más barato y el láser es el más caro. Veamos el sistema perceptivo del robot Scribbler, comenzando por los sensores internos.

## La percepción en el Scribbler

El Scribbler tiene tres mecanismos sensoriales internos: *atascamiento, tiempo y nivel de batería*. Cuando el programa le pide al robot que no se mueva, no implica siempre que el robot efectivamente se esté moviendo. Podría estar atascado contra una pared, por ejemplo. El sensor de atascamiento en el Scribbler permite detectar esto. Ya han visto cómo pueden usar los comportamientos de control de tiempo usando las funciones timeRemaining y wait. Además, para la mayoría de los comandos de movimiento, pueden especificar cuánto tiempo desean que se lleve a cabo el movimiento (por ejemplo, forward(1, 2.5) significa velocidad completa hacia delante durante 2.5 segundos). Finalmente, también es posible detectar el nivel de potencia de la batería para que puedan saber cuándo es el momento de cambiar las baterías del robot.

## Tiempo

Todas las computadoras vienen con un reloj interno incorporado. De hecho, los relojes son tan esenciales para las computadoras que utilizamos en la actualidad que sin ellos ¡no tendríamos computadoras en absoluto!.

El robot Scribbler puede usar el reloj de la computadora para percibir el tiempo. Con la ayuda de este reloj podemos usar funciones de tiempo como timeRemaining, wait, y otros comandos de movimiento. Sólo con estas facilidades es posible definir comportamientos automatizados interesantes.

**Realizar la siguiente actividad**: Diseñar un programa para que el Scribbler dibuje un cuadrado (digamos con lados de 6 cm). Para realizar esto, tendrán que experimentar con los movimientos del robot para correlacionarlos con el tiempo. Los dos movimientos a los cuales les tienen que prestar atención son el ritmo con el cual se mueve el robot cuando avanza en una línea recta, y el grado de cambio con respecto al tiempo. Pueden escribir una función para cada uno de estos:

```
def voyDerecho(distancia):
    # Avanzo en una linea recta distancia cm.
    ...
def giro(angulo):
    # Giro un total de angulo grados.
```

Es decir, averigüen a través de la experimentación con su propio robot (los resultados variarán de robot a robot) cuál es la correlación entre la distancia y el tiempo para un determinado tipo de movimiento dado arriba y luego usen esto para definir las dos funciones. Por ejemplo, si un robot (caso hipotético) parece trasladarse a un ritmo de 25 cm/minuto cuando enuncian el comando translate(1.0), entonces para avanzar 6 cm tendrán que trasladarlo durante un total de (6*60)/25 segundos. Intenten mover el robot hacia delante variando las cantidades de tiempo a la misma velocidad fija. Por ejemplo, intenten mover el robot hacia delante a la velocidad 0.5 durante 3, 4, 5, 6 segundos. Notarán una gran variación en la distancia aún con el mismo grupo de comandos.

Será necesario buscar un promedio entre ellos. Con estos datos, pueden estimar el tiempo promedio que le toma avanzar un cm. Pueden entonces definir voyDerecho de la siguiente manera:

```
def voyDerecho(distancia):
   # establecer la velocidad del robot
  cmPorSeg = <Inserte el valor de su robot aqui>
   # Avanzo en una linea recta distancia cm.
   forward(1, distancia/cmPorSeg)
```

De la misma manera, también pueden determinar el tiempo requerido para girar cierta cantidad de grados. Prueben hacer girar al robot a la misma velocidad variando la cantidad de tiempo. Experimenten cuánto tarda el robot en girar 30 grados, 729 grados, etc. Otra vez, busquen el promedio entre los datos que recogen para obtener la cantidad de grados por segundo. Una vez que han averiguado los detalles, utilícenlos para escribir la función giro. Luego usen el siguiente programa *main*:

```
def main():
  # Dibuja una cuadrado de lado = 6 cms.
  for lado in range(4):
     voyDerecho(6.0)
     giro(90.0)
  speak("Mira que lindo cuadrado que dibujé.")
main()
```

Hagan correr este programa varias veces. Es poco probable que obtengan un cuadrado perfecto todas las veces. Esto tiene que ver con los cálculos que han hecho así como con la variación en el motor del robot. No son precisos. Además, generalmente requiere más potencia el movimiento desde una posición quieta que cuando se continúa un movimiento. Dado que no tienen modo de controlar esto, a lo sumo pueden aproximarse a este tipo de comportamiento. Con el correr del tiempo, también notarán que el error se sumará. Esto resultará evidente con el ejercicio propuesto abajo.

**Realizar la siguiente actividad**: Basándonos en las ideas del ejercicio previo, podemos avanzar en la abstracción del comportamiento como dibujante del robot de manera que podamos pedirle que dibuje cualquier polígono regular (dadas las cantidades de lados y largos de cada lado). Escriban la función:

```
def dibujoPoligono(LADOS, LONGITUD):
   # Dibuja un poligono regular de LADOS numero de lados de LONGITUD
   # cada uno.
```

A continuación, podemos escribir un programa de dibujo de polígonos regulares de la siguiente manera:

```
def main():
    # Dado el número de lados y la longitud de cada uno
    # dibuja un poligono regular

    # Primero pregunto por el numero de lados y
    # la longitud de cada uno.
    print “Dado el número de lados de lados, voy a dibujar”
    print “un polígono para vos.. Decime la longitud de cada lado en cm.”

    nLados = input("Entre el número de lados: ")
  longLado = input("Entre la longitud de cada lado:")

    # Dibuja el poligono
    dibujoPoligono(nLados, longLado)

    speak("Mira! Puedo dibujar!")

main()
```

Para testear el programa, primero intenten dibujar un cuadrado de 6 cm como en el ejercicio previo. Luego prueben un triángulo, un pentágono, un hexágono, etc. Prueben un polígono con 30 lados de 5 cm de largo. ¿Qué ocurre cuando definen 1 como el número de lados? ¿Qué ocurre si definen cero (0) como el número de lados?

## Un ligero desvío: caminatas al azar

Una manera de hacer cosas interesantes con los dibujos del robot es incorporando un poco de azar en la cantidad de tiempo durante la cual el robot realiza determinada actividad. Python, al igual que la mayoría de los lenguajes de programación, prove una librería para la generación de números al azar. La generación de números al azar es de por sí un proceso interesante, pero dejaremos esa discusión para más adelante. Los números al azar son muy útiles para todo tipo de aplicaciones en computación, especialmente para juegos y fenómenos de simulación de la vida real. Por ejemplo, al estimar cuántos autos pueden entrar en una autopista que ya está llena en la hora pico, etc. Para acceder a las funciones de creación de números al azar en Python, deben importar la librería random (azar).

```
from random import *
```

Hay muchas funcionalidades disponibles en esta librería, pero nos restringiremos a sólo dos funciones por ahora: random y randrange. Estas se describen abajo:

**random():** Devuelve un número al azar entre 0.0 y 1.0

**randint(A, B):** Devuelve un número al azar en el rango [A...B]

Aquí hay una muestra de interacción con estas dos funciones:

```
7
>>> randint(1,10)
7
>>> randint(1,10)
7
>>> randint(1,10)
10
>>> randint(1,10)
5
>>> randint(1,10)
2
>>> random()
0.44634304047922957
>>> random()
0.19755181190206628
>>> random()
0.90671189955264253
>>> random()
0.3057456532664905
```

Como pueden ver, usar la librería de números al azar es bastante fácil y es similar al uso de la librería Myro para los comandos del robot. Dadas las dos funciones, ahora depende completamente de ustedes el modo en que las usen. Observen el programa a continuación:

```
def main():
    # Genera 10 polígono aleatorios
    for poly in range(10):
        #genero un oligono y lo dibujo
        Print “Coloca un nuevo color en el lugar para la lapicera y ....”
        userInput = input("Ingresa cualquier número: ")
        lados = randint(3, 8)
        tamanio = randint(2, 6)
        dibujoPoligono(lados, tamanio)

    # genero una caminata aleatoria de 20 pasos
    for paso in range(20):
        voyDerecho(random())
        giro(randrange(0, 360))
```

La primer *iteración* en el programa dibuja 10 polígonos de lados que van desde los 3 a los 8 lados y cada lado tiene entre 2 y 6 cm. La segund*a iteración* lleva a cabo una caminata al azar de 20 pasos.

## ¿Formulando Preguntas?

Como pueden ver arriba, es fácil programar distintos tipos de movimientos en el Scribbler. Si hay una lapicera en el porta-lapicera, el Scribbler dibujará un camino. Además, en el ejemplo de

arriba, pueden observar que podemos detener el programa temporalmente, simular que estamos tomando cierta *entrada (input)* y usar esa oportunidad para cambiar la lapicera y continuar. Arriba hemos usado el comando input para hacer esto. Hay una manera mejor de llevar esto a cabo y utiliza una función provista por la librería Myro:

```
>>> askQuestion("Estás listo?")
```

Cuando se ejecuta esta función, una ventana de diálogo aparece de la siguiente manera:


Al cliquear sobre cualquiera de las opciones (Sí/No), desaparece la ventana y la función devuelve el nombre de la tecla seleccionada por el usuario como una cadena (*string)*. Esto significa que si en la ventana de arriba presionaron la tecla *Yes* (Sí), la función devolverá el valor:

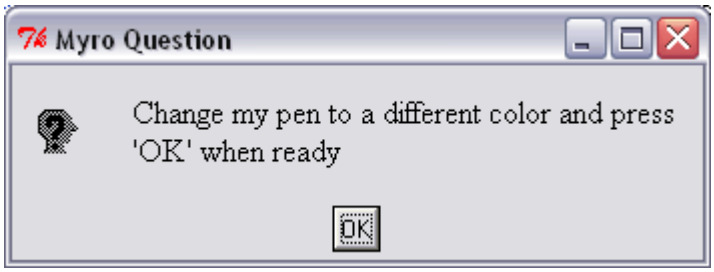
```
>>> askQuestion("Estás listo?")
'Yes'
```

El comando askQuestion puede ser usado en el programa de arriba de la siguiente manera:

```
askQuestion("Cambia la lapicera y presiona la opcion 'Si' cuando estés listo")
```

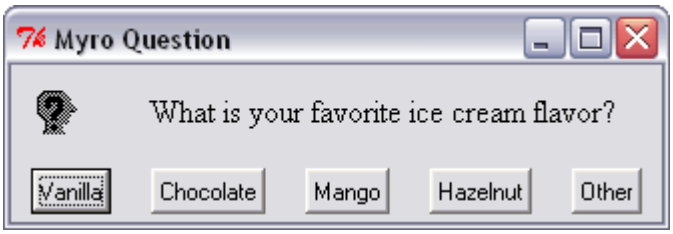
Es definitivamente más funcional que nuestra solución previa, pero podemos hacerlo aún mejor. Por ejemplo, ¿qué ocurre cuando el usuario presiona el botón *No* en la interacción de arriba? Algo que saben con certeza es que la función devolverá *la cadena 'No'*. Sin embargo, por el modo en que estamos usando esta función, realmente no importa qué tecla presione el usuario. askQuestion está diseñado para ser personalizada por ustedes, de modo que puedan especificar cuántos botones de opciones desean tener en la ventana de diálogo así como cuáles deberían ser los nombres de esos botones. Aquí hay una ilustración de cómo podrían escribir una mejor versión del comando de arriba:

```
askQuestion("Cambia la lapicera y presiona la opcion 'Si' cuando estés listo",
["OK"])
```




Sin duda, esto es mucho mejor. Observen que la función askQuestion puede ser utilizada con un parámetro o dos. Si sólo se especifica un parámetro, entonces el comportamiento predeterminado de la función es ofrecer dos botones de opciones: 'Yes' y 'No'. Sin embargo, usando el segundo parámetro pueden especificar, en una lista, cualquier cantidad de *cadenas* que se convertirán en botones de opciones. Por ejemplo,

```
askQuestion("¿Cuál es tu gusto favorito?", ["Vainilla", "Chocolate", "Mango", "Crema", "Otro"])
```




Esta será una función muy útil para usar en muchas situaciones distintas. En el siguiente ejercicio, intenten usar esta función para familiarizarse con ella.

**Realizar la siguiente actividad**: Escriban un programa que explote los movimientos del Scribbler para hacer dibujos al azar. Asegúrense de generar dibujos que usen, por lo menos tres o más colores. Debido a los movimientos al azar, es muy probable que el robot se choque con cosas y se atasque. Ayuden al robot levantándolo y ubicándolo en otro lugar cuando esto ocurra.

## Regresando al tiempo…

La mayoría de los lenguajes de programación también permiten acceder al reloj interno para mantener un registro del tiempo o del tiempo transcurrido (como en *stop watch* – detener reloj), o de cualquier otra manera que quieran hacer uso de la función del tiempo (como en el caso de wait). La librería Myro provee una función simple que puede ser usada para recuperar el tiempo actual:

```
>>> currentTime()
1169237231.836
```

El valor devuelto por currentTime es un número que representa los segundos transcurridos desde algún tiempo anterior, cualquiera que sea. Intenten emitir este comando varias veces y notarán que la diferencia entre los valores devueltos por la función representa el tiempo real en segundos. Por ejemplo:

```
>>> currentTime()
1169237351.5580001
>>> 1169237351.5580001 - 1169237231.836
119.72200012207031
```

Esto significa que 11.722 segundos han transcurrido entre los dos comandos de arriba. Esto nos provee otra forma de escribir comportamientos de robots. Hasta ahora, hemos aprendido que si queremos que el robot avance hacia delante durante 3 segundos, se podría hacer esto:

```
forward(1.0, 3.0)
```

o

```
forward(1.0, 3.0)
wait(3.0)
```

o

```
while timeRemaining(3.0):
  forward(1.0)
```

Usando la función `currentTime`, hay aún otra manera de hacer lo mismo:

```
tiempoInicial = currentTime()     # registra el tiempo de inicio
while (currentTime() - tiempoInicial) < 3.0:
  forward(1.0)
```

Esta solución usa el reloj interno. Primero registra el tiempo de inicio. Luego ingresa al *loop* que primero toma el tiempo actual y seguidamente lo chequea para ver si la diferencia entre la hora actual y la del inicio es menor a 3.0 segundos. Si es así, se repite el comando forward. Ni bien el tiempo transcurrido supera los 3.0 segundos, termina el *loop*. Esta es otra manera de usar la iteración while. que aprendieron en el capítulo anterior. En el último capítulo, aprendieron que pueden escribir un *loop* que se ejecuta por siempre, como se muestra abajo:

```
while True:
  <hacer algo>
```

La forma más general de la iteración while es:

```
while <alguna condición es verdadera>:
  <hacer algo>
```

Esto significa que pueden especificar cualquier condición en <alguna condición es verdadera>. La condición se *testea* y si resulta un valor True (verdadero), el paso o los pasos especificados en <hacer algo> son llevados a cabo. La condición se testea nuevamente y así sucesivamente. En el ejemplo de arriba, usamos la expresión:

```
(currentTime() - tiempoInicial) < 3.0
```

Si esta condición es verdadera, implica que el tiempo transcurrido desde el inicio es menor a 3.0 segundos. Si esto es falso, implica que han pasado más de 3.0 segundos y resulta un valor False (falso), por lo cual *la iteración* se detiene. Aprender acerca de la escritura de estas condiciones es esencial para escribir programas de robot más inteligentes.

Aunque puede parecer que la solución que especifica el tiempo en el comando forward, en sí misma parecía suficientemente simple (¡y lo es!), pronto, descubrirán que el hecho de poder usar el reloj interno como se muestra arriba brinda más versatilidad y funcionalidad para el diseño de comportamientos del robot. De esta manera, por ejemplo es como se podría programar un robot-aspiradora para limpiar una habitación durante 60 minutos:

```
tiempoInicial = currentTime()
while (currentTime() - tiempoInicial)/60.0 < 60.0:
 limpiarCuarto()
```

Ya han visto cómo escribir programas de robot que tienen comportamientos o comandos que pueden ser repetidos un número fijo de veces, o para siempre, o por una duración determinada.

```
# hacer algo n veces
for step in range(N):
```

```
 hacer algo...

# hacer algo siempre
while True:
 hacer algo ....

# hacer algo durante una duración
while timeRemaining(duracion):
 hacer algo ....

# hacer algo durante una duración
duracion = <algun timepo en segundos>
tiempoInicial = currentTime()
while (currentTime() - tiempoInicial) < duracion:
 hacer algo ....
```

Todos los casos de arriba son útiles para diferentes situaciones. A veces resulta una cuestión de preferencias personales.

## Escribiendo condiciones

Pasemos más tiempo aquí para aprender acerca de las condiciones que se pueden escribir en las iteraciones usando while. Lo primero que hay que notar es que todas las condiciones resultan en uno de dos valores: True o False (verdadero o falso) (o, alternativamente, un 1 o un 0). Estos son valores de Python, igual que los números. Se pueden usar de muchas maneras. Las condiciones simples pueden escribirse usando operaciones de comparación (o relacionales): < (menor que), <= (menor que o igual a), > (mayor que), >= (mayor que o igual a), = = (igual a), y != (no igual a). Estas operaciones pueden ser usadas para comparar todo tiempo de valores. Aquí hay algunos ejemplos:

```
>>> 42 > 23
True
>>> 42 < 23
False
>>> 42 == 23
False
>>> 42 != 23
True
>>> (42 + 23) < 100
True
>>> a, b, c = 10, 20, 10
>>> a == b
False
>>> a == c
True
```

```
>>> a == a
True
>>> True == 1
True
>>> False == 1
False
```

Los últimos dos ejemplos de arriba también muestran cómo los valores True y False se relacionan con 1 y 0. True es lo mismo que 1, y 0 es lo mismo que False. Pueden formar muchas condiciones útiles usando las operaciones de comparación y todas las condiciones resultan en True (o 1) o False (o 0). También pueden comparar otros valores, como los strings, usando estas operaciones:

```
>>> "Hola" == "Chau"
False
>>> "Hola" != "Chau"
True
>>> "Luna" < "Lunes"
True
>>> "Limon" < "Manzana"
True
>>> "A" < "B"
True
>>> "a" < "A"
False
```

Estudien los ejemplos de arriba cuidadosamente. Deberán notar dos cosas importantes: los strings son comparados usando un orden alfabético (por ej. lexicográficamente). Por ende, "Luna" es menor que "Lunes", dado que "a" es menor que "e" en esos strings ("Lun" es igual en ambos). Por eso "Limon" es que es menor que "Manzana" (dado que "L" es menor que "M"). La segunda cuestión importante para observar es que las letras mayúsculas son menores que su contraparte equivalente minúscula (“A” es menor que “a”). Esto viene de diseño (ver recuadro a la derecha).

**Unicode**

Los caracteres de texto tienen una codificación computacional interna o representación que hace cumplir el ordenamiento lexicográfico. Esta codificación interna es muy importante en el diseño de computadoras y es lo que permite que todas computadoras y dispositivos como iPhones, etc., puedan compartir información de manera consistente. Todos los caracteres de lenguajes del mundo tienen asignados una codificación computacional estándar. Esta se llama Unicode.

Además de operaciones relacionales, pueden construir expresiones de condiciones más complejas usando las *operaciones lógicas* (también llamadas operaciones booleanas): and, or, y not (y, o, no). Aquí hay algunos ejemplos:

```
>>> (5 < 7) and (8 > 3)
True
>>> not ( (5 < 7) and (8 > 3))
False
```

```
>>> (6 > 7) or (3 > 4)
False
>>> (6 > 7) or (3 > 2)
True
```

Podemos definir el significado de las operaciones lógicas de la siguiente manera:

- **<expresion-1> and <expresion-2>**: Tal expresión resultará en un valor True (verdadero) sólo si ambas <expresion-1> y <expresion-2> son True. En todos los otros casos, (por ej. si una o ambas de las <expresion-1 > y <expresion-2> son False ) resulta False (falso).
- **<expresion-1> or <expresion-2>**: Tal expresión resultará en un valor True si <expresion-1> o <expresion-2> son True o si ambas son True. En todos los otros casos (por ej. si ambas <expresion-1> y <expresson-2> son False), resultará False.
- **not <expresion>**: Tal expresión resultará en un valor True si <expresion> es False, o False si <expresion> es True (por ej., da vuelta o complementa el valor de la expresión).

Estos operadores pueden ser combinados con expresiones relacionales para formar expresiones condicionales arbitrariamente complejas. De hecho, cualquier toma de decisión en la creación de un programa se reduce a formar las expresiones condicionales apropiadas. Los operadores lógicos fueron creados por el lógico George Boole a mediados del siglo 19. El álgebra booleano, llamado así por Boole, define algunas leyes simples aunque importantes que gobiernan el comportamiento de los operadores lógicos. Aquí hay algunos útiles:

- `(A or True)es siempre True`
- `(not (not A))es igual a A`
- `(A or (B and C))` es lo mismo que `((A or B) and (A or C))`
- `(A and (B or C))` es lo mismo que `((A and B) or (A and C))`
- `(not (A or B))` i es lo mismo que `((not A) and (not B))`
- `(not (A and B))es lo mismo que ((not A) or (not B))`

Estas identidades o propiedades pueden ayudarlos a simplificar las expresiones condicionales. Las expresiones condicionales pueden ser usadas para escribir varias condiciones para controlar la ejecución de algunos de los enunciados del programa. Estos permiten escribir repeticiones condicionales del tipo:

```
while <alguna condicion es verdadera>:
  <hacer algo>
```

Ahora pueden entender por qué la siguiente es una manera de decir “hacé algo para siempre”:

```
while True:
```

```
  <hacer algo>
```

Dado que la condición es siempre True, los enunciados se repetirán por siempre. Se observa algo en el loop de abajo:

```
while timeRemaining(duracion):
  <hacer algo>
```

Ni bien la duracion termina, el valor de la expresión timeRemaining(duracion) se transforma en False y se detiene la repetición. El control de la repetición basado en condiciones es una idea poderosa en computación. Lo usaremos extensivamente para controlar el comportamiento de los robots.

## Detectando atascamientos

Mencionamos al inicio de este capítulo que el Scribbler también tiene un modo de detectar que está atascado al intentar moverse. Esto se logra usando la función Myro getStall:

**`getStall():`** Retorna True si el robot está atascado y False, en caso contrario.

Pueden usarlo para detectar que el robot se ha atascado e incluso pueden usarlo como una condición en un loop para controlar el comportamiento. Por ejemplo:

```
while not getStall():
  <hacer algo>
```

Esto significa, continuar haciendo <hacer algo> hasta que el robot se haya atascado. De esta manera, pueden escribir un comportamiento en que el robot vaya hacia delante hasta que se choque contra algo, digamos, una pared, y entonces se detenga.

```
while not getStall():
    forward(1.0)
stop()
speak("Ouch! Creo que choqué con algo!")
```

En el ejemplo de arriba, mientras que el robot no se haya atascado, getStall() devolverá un False y por lo tanto el robot continuará avanzando hacia delante (dado que not False es True). Una vez que se choca contra algo, getStall() devolverá True y entonces el robot se detendrá y hablará.

**Realizar la siguiente actividad:** Escriban un programa completo para el Scribbler a fin de implementar el comportamiento de arriba y luego observen el comportamiento. Muéstrenselo a amigos que no estén en el curso. Pregúntenles por sus reacciones. Notarán que la gente tiende a otorgarle cierta inteligencia al robot. Es decir, el robot está percibiendo que está atascado, y al hacerlo, deja de intentar moverse e incluso anuncia que está atascado hablando. Volveremos sobre esta idea de inteligencia artificial en un capítulo posterior.

## Detectando niveles de batería

El robot Scribbler corre sobre baterías 6 AA. Como en cualquier otro dispositivo electrónico, con el uso, las baterías finalmente se agotarán y necesitarán reemplazarlas por nuevas baterías. Myro provee una función interna de detección de nivel de batería llamada getBattery que devuelve el voltaje actual provisto por la batería. Cuando baja el nivel de la batería, obtendrán menos y menos voltaje, lo cual llevará a un comportamiento errático. Los niveles de voltaje de batería del Scribbler variarán entre 0 y 9 voltios (0 significa que está completamente agotada). Cuál es un nivel de batería bajo es algo que tendrán que experimentar y averiguar por ustedes mismos. La mejor forma de hacerlo es registrar el nivel de batería al insertar un nuevo juego. Luego, a medida que pase el tiempo, continúen registrando los niveles de batería a medida que avancen.

El Scribbler también tiene incorporadas luces que indican el nivel de batería. El LED rojo en el robot se mantiene encendido mientras que los niveles de potencia sean altos (o en un rango aceptable). Empieza a titilar cuando se agota el nivel de la batería. Hay un LED similar en el Fluke. ¿Pueden encontrarla? Esperen a que los niveles bajen y lo verán titilando.

**Eliminación de las baterías**

Asegúrense de eliminar sus baterías usadas apropiadamente y de manera responsable. Las baterías contienen materiales dañinos como cadmio, mercurio, plomo, litio, etc., que pueden ser contaminantes mortales si se arrojan en basureros. Averigüen dónde está el centro de reciclado de baterías más cercano o la opción para eliminación de esos residuos para asegurarse una eliminación apropiada.

Pueden usar el detector de nivel de batería para definir comportamientos del robot para que sean llevados a cabo sólo cuando hay suficiente potencia disponible. Por ejemplo:

```
while (getBattery() >= 5.0) and timeRemaining(duracion):
  <hacer algo>
```

Es decir, mientras que la potencia de la batería esté sobre 5.0 y el límite de tiempo no haya excedido **duracion**, <hacer algo>.

## Población Mundial, revisado

La habilidad de escribir expresiones condicionales también nos permite definir computaciones más sofisticadas. Recuerden el ejemplo de proyección de población mundial del capítulo anterior. Dadas la tasa de crecimiento de la población y la población actual, ahora pueden escribir un programa para predecir en qué año la población mundial se incrementará más allá de cierto número dado, digamos, 9 billones. Todo lo que deben hacer es escribir una repetición guiada por una condición que tenga la siguiente estructura:

```
anio = <año actual>
poblacion = <pob>lacion actual
tasaCrecimiento = <tasa de crecimiento>

# repetir mientras que la poblacion no llegue a 9000000000
while poblacion < 9000000000:
    # calcular la poblacion para el año siguiente
    anio = anio + 1
    poblacion = poblacion * (1+tasaCrecimiento )

print "En el año", anio, "tla población del mundo "
print "habrá excedido los 9 billion."
```

Es decir, agregar el crecimiento de la población al siguiente año si la población es menor a 9 billones. Seguir repitiendo esto hasta que excede los 9 billones.

**Realizar la siguiente actividad**: Completar el programa de arriba y computen el año en que la población mundial excederá los 9 billones. Para que el programa sea más útil, asegúrense de pedirle al usuario que ingrese los valores del año, la población, la tasa de crecimiento, etc. De hecho, incluso pueden pedirle al usuario que ingrese el límite de población para poder usar el programa para cualquier tipo de predicción (¿8 billones?, ¿10 billones?). ¿Cómo cambiarían el programa para que imprima la proyección de población para un año determinado, digamos el 2100?

## Resumen

En este capítulo han aprendido acerca de la percepción o los mecanismos sensoriales internos. El robot Scribbler tiene tres mecanismos sensoriales internos: tiempo, atascamiento y nivel de batería. Han aprendido cómo percibir estas cantidades y también cómo usarlas para definir comportamientos automatizados del robot. También aprendieron acerca de la generación de números al azar y lo usaron para definir comportamientos del robot impredecibles. Más adelante, también aprenderemos cómo usar números al azar para escribir juegos y simular fenómenos naturales. La percepción también puede ser usada para definir comportamientos repetitivos condicionales usando expresiones condicionales en iteraciones. Aprendieron cómo construir y escribir diferentes tipos de condiciones usando operaciones relacionales y lógicas. Estas serán valiosas para definir comportamientos que usan mecanismos sensoriales externos y nos permitirá

escribir comportamientos de toma de decisión más explícitos. Aprenderemos acerca de estos en el próximo capítulo.

## Revisión Myro

`randomNumber()`
Devuelve un número al azar en el rango de 0.0 y 1.0. Esta es una función alternativa de Myro que funciona igual que la función random de la librería random de Python (ver abajo).

`askQuestion(MESSAGE-STRING)`
Se abre una ventana con un MESSAGE-STRING con opciones: 'Yes' y 'No'. Devuelve 'Yes' o 'No' dependiendo de lo que seleccione el usuario.

`askQuestion(MESSAGE-STRING, LIST-OF-OPTIONS)`
Se abre una ventana con un MESSAGE-STRING con opciones indicadas en LIST-OF-OPTIONS. Devuelve la opción de *string* dependiendo de lo que el usuario seleccione.

`currentTime()`
El tiempo actual, en segundos, de un punto de inicio arbitrario, hace muchos años.

`getStall()`
Devuelve True si el robot está atascado al intentar moverse, o si no, False.

`getBattery()`
Devuelve el nivel de batería actual (en voltios), puede ser un número entre 0 y 9; 0 que indica sin potencia y 9 es el más alto. También hay indicadores de potencia LED en el robot. El comportamiento del robot se torna errático cuando bajan las baterías. Es entonces el momento de reemplazarlas.

## Revisión Python

`True, False`
Estos son valores lógicos o booleanos en Python. Python también define True como 1 y False como 0 y pueden ser usados en horma intercambiable.

`<, <=, >, >=, ==, !=`
Estas son operaciones relacionales en Python. Pueden ser usadas para comparar valores. Ver el texto para más detalles sobre estás operaciones.

`and, or not`
Estas son operaciones lógicas. Pueden ser usadas para combinar cualquier expresión que se someta a valores booleanos.

`random()`

Devuelve un número al azar entre 0.0 y 1.0. Esta función es parte de la librería random en Python.

`randRange(A, B)`

Devuelve un número al azar en el rango entre A (inclusivo y B(exclusivo). Esta función es parte de la librería random en Python.

## Ejercicios

**1.** Escriban un programa de robot para hacer que el Scribbler dibuje una estrella de cinco puntas. [Ayuda: cada vértice en la estrella tiene un ángulo interior de 36 grados.]

**2.** Experimenten con los comandos de movimiento Scribbler y aprendan cómo hacer que transcriba un camino de un radio determinado. Escriban un programa para dibujar un círculo de cualquier *input* de diámetro.

**3.** Escriban un programa para dibujar otras formas: la silueta de una casa, un estadio, o crear arte insertando lapiceras de diferentes colores. Escriban el programa de modo que el robot se detenga y pida un nuevo color.

**4.** Si tuvieran un rectángulo de césped (sin árboles ni obstrucciones) podrían usar un Zanboni como estrategia para cortar el césped. Empiecen en una punta del rectángulo, corten el largo completo sobre el costado más largo, giren y corten el largo completo de nuevo, junto a la zona recientemente cortada, etc., hasta terminar. Escriban un programa para el Scribbler para implementar esta estrategia (asegúrense de que el Scribbler dibuje el camino mientras avanza).

**5.** Mejoren el programa de dibujo al azar de este capítulo para usar el habla. Hagan que el robot, a medida que realiza sus movimientos al azar, hable acerca de lo que está haciendo. Como resultado, ¡tendrán un robot artista creado por ustedes mismos!

**6.** Reescriban el programa del ejercicio previo como para que el comportamiento al azar para cada lapicera se lleve a cabo por 30 segundos.

**7.** La librería Myro también provee una función llamada randomNumber(), que devuelve un número al azar en el rango de 0.0 y 1.0. Esto es similar a la función random()de la librería Python random que se introdujo en este capítulo. Pueden usar cualquiera, basándose en su preferencia personal. Tendrán que importar la librería adecuada, de acuerdo a la función que elijan usar. Experimenten con ambas para convencerse de que son equivalentes.

**8.** En realidad, sólo necesitan la función random() para generar números al azar de cualquier rango. Por ejemplo, pueden obtener un número al azar entre 1 y 6 con randRange(1,6) como se muestra abajo:

```
randomValue = 1 + int(random()*6)
```

```
La función int() toma cualquier número como su parámetro, lo trunca en un
número entero y lo devuelve como integer. Dado ese that random(), devuelve
valores entre 0.0 (inclusivo) y 1.0 (exclusivo), la expresión de arriba
asignará un valor al azar entre 1.5 (inclusivo) a randomValue. Dado este
```

ejemplo, escriban una nueva función llamada myRandRange() que funcione como randrange():

```
def myRandRange(A, B):

   # generate a random number between A..B
   # (just like as defined for randrange)
```

**9.** ¿Sobre qué tipo de cosas puede hablar el robot? Ya han visto cómo hacer que el robot/computadora hable y diga una oración o una frase. Pero el robot también puede "hablar" acerca de otras cosas, como el tiempo o el clima.

Una forma de obtener la hora actual y la fecha es importar otra librería Python llamada time:

```
>>> from time import *
```

El módulo de tiempo provee una función llamada localtime que funciona de la siguiente manera:

```
>>> localtime()
(2007, 5, 29, 12, 15, 49, 1, 149, 1)
```

`localtime` devuelve todo con referencia a:

1. año
2. mes
3. día
4. hora
5. minuto
6. segundos
7. día de la semana
8. día del año
9. sobre si está usando horario de verano o no

En el ejemplo de arriba, es el 29 de mayo de 2007, a las 12:15pm y 49 segundos. También es el primer día de la semana, el 149 día del año, y estamos usando horario de verano. Pueden asignarle cada variable a valores nombrados como se muestra abajo:

```
year, month, day,…, dayOfWeek = localtime()
```

Entonces, por el ejemplo de arriba, la variable año tendrá el valor de 2007, el mes tendrá el valor 5, etc. Escriban un programa Python que diga la fecha y hora actuales.

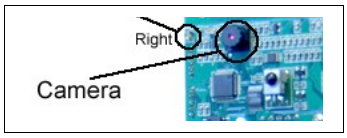
# 5 Percibiendo el Mundo

*Veo todos los obstáculos en mi camino.*
De la canción *Puedo ver claramente ahora*,
Johnny Nash, 1972.

Página opuesta: Los Sentidos
La foto es cortesía de Blogosphere (cultura.blogosfere.it)

En el capítulo anterior, aprendieron cómo  la percepción del tiempo, el atascamiento y el nivel de batería, pueden ser usados para escribir comportamientos de robot simples pero interesantes. Todos los robots, además, vienen equipados con un conjunto de sensores externos (o extroperceptores) que pueden percibir varias cosas en el ambiente. La percepción hace que el robot sea *consciente* de su entorno y puede ser usada para definir más comportamientos inteligentes. La percepción también está relacionada con otro concepto importante en computación: la entrada o input. Las computadoras actúan a partir de diferente tipo de información: números, textos, sonidos, imágenes, etc. para producir aplicaciones útiles. Generalmente se hace referencia a la adquisición de información para ser procesada como entrada o input. En este capítulo también veremos cómo otras formas de entradas pueden ser adquiridas para ser procesadas por un programa. Primero, focalicemos en los sensores del Scribbler.

## Los sensores del Scribbler




El robot Scribbler puede percibir la cantidad de luz en el ambiente, la presencia (o ausencia) de obstáculos a su alrededor, y también toma fotos con su cámara. En el Scribbler hay localizados varios dispositivos (o sensores). A continuación, se presenta una breve descripción de los mismos:

**Cámara**: La cámara puede tomar una imagen fija de cualquier cosa que el robot esté "visualizando" en ese momento.

**Luz**: Hay tres sensores de luz en el robot. Están localizados en los tres agujeros de la parte frontal del robot. Estos sensores pueden detectar los niveles de brillo (u oscuridad). Estos pueden ser usados para detectar variaciones en la luz ambiente en una habitación. Además, al usar la información proveniente de la cámara, el Scribbler tiene a disposición un conjunto de sensores de brillo alternativo (izquierda, centro y derecha).

**Proximidad**: Hay dos juegos de estos sensores en el Scribbler: Sensores IR (izquierda y derecha) en el frente; y Sensores de Obstáculos (izquierda, centro y derecha) en el *Fluke*. Pueden ser usados para detectar objetos en el frente y a los costados.

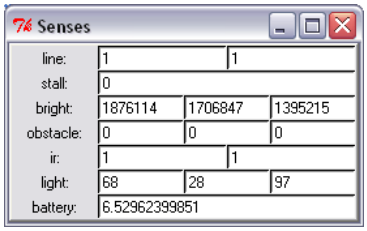
## Familiarizándose con los sensores

Percibir a través de los sensores provistos por el Scribbler es sencillo. Una vez que estén familiarizados con los detalles de comportamiento de los sensores, podrán usarlos en sus programas para diseñar comportamientos interesantes con  su Scribbler. Pero primero, debemos ocupar un poco de tiempo conociendo estos sensores; cómo acceder a la información que reportan; y cómo se ve esta información. Con respecto a los sensores internos, Myro provee varias funciones que pueden ser usadas para adquirir datos de cada dispositivo de sensor. Allí donde haya disponibles múltiples sensores, tienen la opción de obtener datos de todos los sensores, o selectivamente de un sensor individual.

**Realizar la siguiente actividad**: quizás la mejor forma de tener una mirada rápida sobre el comportamiento global de los sensores es usando la función Myro senses:

```
>>> senses()
```

**Scribbler Sensors**

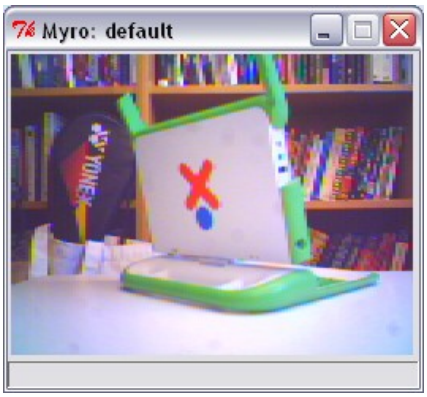



Esto da como resultado una ventana (ver la imagen adjunta) que muestra todos los valores de los sensores (exceptuando la cámara) en tiempo real. Se actualizan a cada segundo. Deberían mover el robot por el entorno par ver cómo cambian los valores de los sensores. La ventana también mostrará los valores del sensor de atascamiento y de nivel de batería. El valor que está en el extremo izquierdo en cada conjunto de sensor (luz, IR, obstáculo y brillo) es el valor del sensor de la izquierda, seguido por el del centro (si existe), y luego el de la derecha.

## La Cámara

La cámara puede ser usada para tomar fotos de la vista actual del robot. Como pueden ver, la cámara está localizada en el Fluke. La vista de la imagen tomada por la cámara dependerá de la orientación del robot (y del Fluke). Para tomar fotos con la cámara deberán usar el comando takePicture:

```
takePicture()
takePicture("color")
takePicture("gray")
```




Este comando toma una foto y devuelve una imagen. De manera predeterminada, cuando no se especifican parámetros, la foto es en color. Si se usa la opción "gray", pueden obtener una foto en escala de grises.

Por ejemplo:

```
>>> p = takePicture()
>>> show(p)
```

De manera alternativa, también pueden hacer lo siguiente:

```
>>> show(takePicture())
```

Una vez que han tomado una foto con la cámara, pueden hacer muchas cosas con ella. Por ejemplo, pueden querer ver si hay una computadora portable presente en la imagen. El procesamiento de información es un vasto sub-campo de las ciencias de la computación y tiene

aplicaciones en muchas áreas. Comprender una imagen es bastante complejo, pero es algo que hacemos con naturalidad. Por ejemplo, en la imagen de arriba, no tenemos problemas para localizar la portable la estantería de libros de fondo, e incluso un estuche de una raqueta. La cámara del Scribbler es su dispositivo más complejo y requerirá mucho esfuerzo computacional y energía para ser usado en el diseño de comportamientos. En el caso más simple, la cámara les puede servir como sus “ojos” remotos en el robot. Quizás no lo mencionamos antes, pero el rango del Bluetooth inalámbrico en el robot es de 100 metros. En el capítulo 9 aprenderemos acerca de varias maneras de usar las fotos. Por ahora, si toman una foto con la cámara y desean guardarla para un futuro uso, utilicen el comando Myro, savePicture, como en el siguiente ejemplo:

```
>>> savePicture(p, “office-scene.jpg”)
```

El archivo office-scene.jpg será guardado en la misma carpeta que su carpeta Start Python. También pueden usar savePicture para guardar una serie de imágenes de la cámara y convertirla en una “película” animada (como una imagen gif animada). Esto se ilustra con el ejemplo de abajo.

**Realizar la siguiente actividad:** En primer lugar, prueben todos los comandos para sacar y guardar fotos. Asegúrense de sentirse cómodos usándolos. Prueben sacar fotos en escala de grises también. Supongan que el robot se ha internado en un lugar donde no pueden verlo pero está aún en un rango de comunicación con la computadora. Ustedes desearían mirar alrededor para ver dónde está. Quizás está en un cuarto nuevo. Pueden pedirle al robot que se dé vuelta y tomar varias fotos que les muestren el entorno. Pueden hacerlo usando una combinación de los comandos rotate y takePicture, como se muestra abajo:

```
while timeRemaining(30):
  show(takePicture())
  turnLeft(0.5, 0.2)
```

Esto significa tomar una foto y luego girar durante 0.2 segundos, repitiendo los dos pasos durante 30 segundos. Si observan la ventana de imagen que aparece, verán las sucesivas vistas del robot. Prueben esto varias veces y observen si pueden contar cuántas imágenes diferentes aparecen. Seguidamente, cambien el comando takePicture para obtener fotos en escala de grises. ¿Pueden contar la cantidad de fotos que ha tomado esta vez? Hay, por supuesto, una manera más sencilla de hacerlo:

**Píxeles**

Cada imagen está hecha de pequeños elementos de imagen o píxeles. En una imagen color, cada píxel contiene información sobre el color que está compuesta por la cantidad de rojo, verde y azul (RGB). Cada uno de estos valores está en el rango de 0..255 y por lo tanto toma 3 bytes o 24 bits para guardarlo en la información contenida en un solo píxel. Un píxel de color rojo puro tendrá los valores RGB (255, 0, 0). Una imagen en escala de grises, por otro lado, sólo contiene el nivel de gris en un píxel, que puede ser representado por un solo byte (u 8 bits) como un número dentro del rango de 0..255 (donde 0 es negro y 255 es blanco).

```
N = 0
while timeRemaining(30):
   show(takePicture())
   turnLeft(0.5, 0.2)
   N = N + 1
print N
```

Ahora mostrará la cantidad de imágenes que toma. Notarán que es capaz de tomar muchas más imágenes en escala de grises que en color. Esto se debe a que las imágenes en color tienen mucha más información que las grises (ver el texto arriba). Una imagen color de 256x192 requiere 256x192x3 (= 147,456) bytes de datos, mientras que una imagen en escala de grises requiere solamente 256x192 (= 49, 152) bytes. Cuantos más datos se deban transferir del robot a la computadora, más tiempo tardará.

También pueden guardar un GIF animado de las imágenes generadas por el robot usando el comando savePicture, a través de la acumulación de una serie de imágenes en una lista. Esto se muestra abajo:

```
Pics = []
while timeRemaining(30):
   pic = takePicture()
   show(pic)
   Pics.append(pic)
   turnLeft(0.5, 0.2)
savePicture(Pics, “office-movie.gif”)
```

Primero creamos una lista vacía llamada Pics. Luego le anexamos las sucesivas imágenes tomadas por la cámara a la lista. Una vez que se han reunido todas las imágenes, usamos savePicture para guardar el conjunto completo como un GIF animado. Simplemente carguen el archivo en un navegador Web y mostrará las imágenes como en una película.

Hay muchas otras formas interesantes de usar las imágenes de la cámara. En el capítulo 9 exploraremos las imágenes con mayor detalle. Por ahora, veamos los otros sensores del Scribbler.

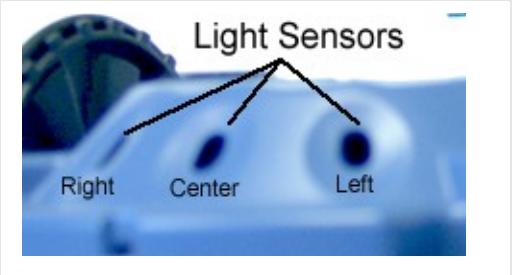
## La percepción de la luz

Las siguientes funciones están disponibles para obtener valores de sensores de luz:

**`getLight()`**: Devuelve una lista que contiene los tres valores de todos los sensores de luz.

**`getLight(<POSITION>)`** Devuelve el valor actual en el sensor de luz <POSITION>. <POSITION> puede ser uno de 'left' , 'center' , 'right' (izquierda, centro y derecha), o uno de los números 0, 1, 2. Las posiciones 0, 1 y 2 corresponden a los sensores de la izquierda, el centro y la derecha. Por ejemplo:

```
>>> getLight()
[135, 3716, 75]
>>> getLight('left')
135
>>> getLight(0)
135
>>> getLight('center')
3716
>>> getLight(1)
3716
>>> getLight('right')
75
>>> getLight(2)
75
```




Los valores que se reportan desde estos sensores pueden oscilar en el rango de [0..5000] donde los valores bajos implican una luz brillante y los valores altos implican oscuridad. Los valores de arriba fueron tomados con luz ambiental, donde un dedo tapaba completamente al sensor del centro. Por ende, cuanto más oscuro está, más alto es el valor que se reporta. Más adelante, veremos cómo podemos transformar fácilmente estos valores de muy distintas maneras para afectar el comportamiento del robot.

Sería una buena idea usar la función senses para jugar un poco con los sensores de luz y observar sus valores. Prueben mover al robot por el entorno para ver cómo se modifican los valores. Apaguen las lucen en la habitación, o cubran los sensores con los dedos, etc.

Al usar la función getLight sin ningún parámetro, obtienen una lista de tres valores de sensores (izquierda, centro y derecha). Pueden asignarlos a variables individuales de muchas maneras:

```
>>> L, C, R = getLight()
>>> print L
135
>>> Center  = getLight(“center”)
>>> print Center
3716
```

Las variables pueden, por lo tanto, utilizarse de muchas maneras para definir comportamientos del robot. Veremos varios ejemplos de esto en el siguiente capítulo.

La cámara que está en el Fluke  también puede ser usada como un tipo de sensor de brillo. Esto se hace promediando los valores de brillo de diferentes zonas de la imagen en la cámara. De alguna manera, pueden considerarlo un sensor virtual. Es decir, no existe físicamente, pero está embebido en la funcionalidad de la cámara. La función getBright es similar a la de getLight por el modo en que puede ser usada para obtener valores de brillo.

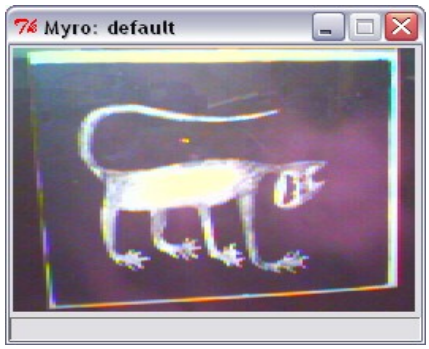
**`Getbright():`** Devuelve una lista que contiene los tres valores de todos los sensores de luz.

**`getBright(<POSITION>):`** Devuelve el valor actual en el sensor de luz <POSITION>. <POSITION> puede ser de izquierda, centro, derecha o uno de los números 0, 1, 2. Las posiciones 0, 1 y 2 corresponden a los sensores de izquierda, centro y derecha. Por ejemplo:

```
>>> getBright()
[2124047, 1819625, 1471890]
>>> getBright('left')
2124047
>>> getBright(0)
2124047
>>> getBright('center')
1819625
>>> getBright(1)
1819625
>>> getBright('right')
1471890
>>> getBright(2)
1471890
```


Los valores de arriba pertenecen a la imagen de cámara del póster de Firefox (ver imagen arriba). Los valores que reportan estos sensores pueden variar dependiendo de la vista de la cámara y de los niveles de brillo resultantes en la imagen. Pero notarán que los valores más altos implican segmentos brillantes y los más bajos implican oscuridad. Por ejemplo, aquí hay otra serie de valores basado en la imagen que se muestra a continuación.

```
>>> getBright()
[1590288, 1736767, 1491282]
```




Como podrán observar, es más probable que una imagen oscura produzca valores de brillo más bajos. Pueden ver claramente que el centro de la imagen es más brillante que sus sectores izquierdo o derecho.

También es importante notar las diferencias en la naturaleza de la información que se reporta a través de los sensores getLight y getBright. El primero reporta la cantidad de luz ambiental que percibe el robot (incluyendo la luz sobre el robot). El segundo es un promedio del brillo obtenido de la imagen vista por la cámara. Estos pueden ser usados de muchas maneras, como veremos más adelante.

**Realizar la siguiente actividad**: El programa que se muestra abajo usa una función de normalización para normalizar los valores del sensor de luz en el rango de [0.0..1.0] relativo a los valores de la luz ambiental. Entonces, los valores de sensores de luz de izquierda y derecha son utilizados para manejar los motores de izquierda y derecha del robot.

```
# registra  un promedio de los valores de luz ambiental
Ambiente = sum(getLight())/3.0

# Esta funcion normailiza los valores de los sensores de luz a valores en el
#rango de 0.0..1.0
def normalize(v):
    if v > Ambiente:
        v = Ambiente

    return 1.0 - v/Ambiente

def main():
    # Run the robot for 60 seconds
```

```
    while timeRemaining(60):
        L, C, R = getLight()
        # motors run proportional to light
        motors(normalize(L), normalize(R))
```

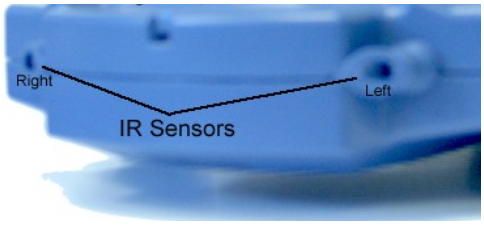
Ejecuten el programa de arriba en su robot Scribbler y observen su comportamiento. Necesitarán una linterna para provocar mejores reacciones. Mientras que el programa esté corriendo, intenten alumbrar uno de los sensores de luz con la linterna (el izquierdo o derecho). Observen el comportamiento. ¿Creen que el robot se está comportando como un insecto? ¿Cuál? Estudien el programa de arriba cuidadosamente. Hay algunas funciones nuevas de Python que discutiremos pronto. También volveremos sobre la idea de hacer que los robots se comporten como insectos en el siguiente capítulo.

## Percepción de Proximidad

El Scribbler tiene dos conjuntos de detectores de proximidad. Hay dos sensores infrarrojos IR en el frente del robot y hay tres sensores IR de obstáculos adicionales en el Fluke dongle. Las siguientes funciones están disponibles para obtener valores de los sensores IR del frente:

**`getIR():`** Devuelve una lista que contiene los dos valores de todos los sensores IR.

**`getIR(<POSITION>)`** Devuelve el valor actual del sensor IR `<POSITION>`. `<POSITION>` puede ser izquierda o derecha o uno de los números 0, 1. Las posiciones 0 y 1 corresponden a los sensores de izquierda, centro y derecha.




Ejemplos:

```
>>> getIR()
[1, 0]
>>> getIR('left')
1
>>> getIR(0)
1
>>> getIR('right')
0
>>> getIR(1)
0
```

Los sensors IR devuelven ya sea un 1 o un 0. El valor 1 implica que no hay nada en una proximidad cercana frente a ese sensor, y el 0 implica que hay algo frente a él. Estos sensores pueden ser usados para detectar la presencia o ausencia de obstáculos frente al robot. Los sensores IR de izquierda y derecha están ubicados lo suficientemente alejados como para poder detectar obstáculos individuales en cada lado.

**Realizar la siguiente actividad:** hagan correr la función `senses` y observen los valores de los sensores IR. Ubiquen varios objetos frente al robot y observen los valores de los sensores de

proximidad IR. Tomen su *notebook* y ubíquenla frente al robot a aproximadamente 60cm de distancia. Lentamente muevan la *notebook* más cerca del robot. Observen cómo cambian los valores del sensor de 1 a 0 y luego alejen nuevamente la *notebook*. ¿Pueden averiguar cuán lejos (o cerca) del obstáculo debe estar para ser detectado? Intenten mover la *notebook* de lado a lado. Nuevamente observen los valores de los sensores IR.

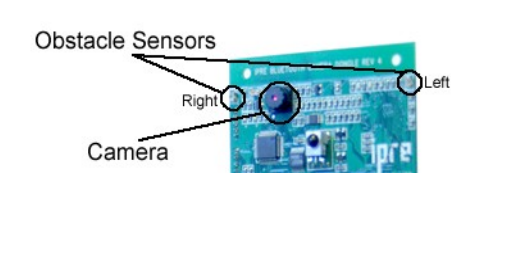
El Fluke  tiene un grupo de sensores adicionales. También son sensores IR pero se comportan de manera muy diferente en términos de los tipos de valores de reportan. Las siguientes funciones están disponibles para obtener valores de los sensores IR de obstáculos:

**`getObstacle():`** Devuelve una lista que contiene los dos valores de todos los sensores IR.

**`getObstacle(<POSITION>):`** Devuelve el valor actual del sensor IR `<POSITION>`. `<POSITION>` puede ser izquierda, centro o derecha, o uno de los números 0, 1 ó 2. Las posiciones 0, 1 y 2 corresponden a los sensores de la izquierda, el centro y la derecha.

Ejemplos:

```
>>> getObstacle()
[1703, 1128, 142]
>>> getObstacle('left')
1703
>>> getObstacle(0)
1703
>>> getObstacle('center')
1128
>>> getObstacle(1)
1128
>>> getObstacle('right')
142
>>> getObstacle(2)
142
```




Los valores reportados por estos sensores varían del 0 al 7000. Un 0 implica que no hay nada frente al sensor, mientras que un número alto implica la presencia de un objeto. Los sensores a los costados pueden ser usados para detectar la presencia (o ausencia) de paredes a los costados.

**Realizar la siguiente actividad**: Ubiquen al Scribbler en el piso, enciéndanlo, inicien Python y conéctense a él. También conecten el controlador del game pad  e inicien la operación de manejo manual ( gamepad() ). A continuación, emitan el comando senses para obtener la visualización del sensor en tiempo real. Nuestro objetivo aquí es “meternos en la cabeza del robot” y manejarlo por el entorno sin mirar al robot en ningún momento. También deben resistir la tentación de sacar una foto. Pueden usar la información brindada por los sensores para navegar el robot. Intenten conducirlo a un punto oscuro de la habitación. Intenten conducirlo de manera que en ningún momento se choque contra un objeto. ¿Pueden detectar cuándo hay algo? Si se atasca, ¡traten de maniobrar para sacarlo del atascamiento! Este ejercicio les dará una idea bastante cercana a lo que percibe el robot, al modo en que usa sus sensores, y al rango de comportamiento del que es capaz. Encontrarán que es un ejercicio difícil, pero les dará una buena idea de lo que ocurre en los cerebros de este tipo de robots cuando intentan diseñarlos. Intentaremos volver sobre este escenario a medida que construimos varios programas de robots.

**También realicen la siguiente actividad**: Prueben el programa de abajo. Es muy similar al programa de arriba que usaba los sensores de luz normalizados.

```
def main():
    # Run the robot for 60 seconds
    while timeRemaining(60):
        L, R = getIR()
        # motors run proportional to IR values
        motors(R, L)
main()
```

Dado que los sensores IR reportan valores 0 ó 1, no necesitan normalizarlos. También observen que estamos poniendo el valor del sensor izquierdo (L) en el motor derecho y el valor del sensor derecho (R) en el motor izquierdo. Ejecuten el programa y observen el comportamiento del robot. Tengan a mano una *notebook* y traten de ubicarla frente al robot. También ubíquenla levemente a la izquierda o a la derecha. ¿Qué sucede? ¿Pueden sintetizar lo que está haciendo el robot? ¿Qué ocurre cuando cambian los valores R y L de los motores?

Pueden ver cómo programas simples como los que vimos arriba pueden resultar ser interesantes estrategias de control automático para robots. También pueden definir comportamientos completamente automatizados e incluso una combinación de comportamientos automatizados y manuales para robots. En el siguiente capítulo exploraremos varios comportamientos de robots. Primero, es hora de que aprendan acerca de las listas es Python.

## Las listas en Python

Han visto arriba que varias funciones de sensores devuelven listas de valores. También utilizamos listas para acumular una serie de fotos de la cámara para generar un GIF animado. Las listas son una manera muy útil de coleccionar un grupo de información y Python provee un conjunto de operaciones y funciones útiles que permiten la manipulación de listas. En Python, una lista es una secuencia de objetos. Los objetos pueden ser cualquier cosa: números, letras, *strings*, imágenes, etc. La lista más simple que pueden tener es una lista vacía:

```
>>> []
[]
```

o

```
>>> L = []
>>> print L
[]
```

Una lista vacía no contiene nada. Aquí hay algunas listas que contienen objetos:

```
>>> N = [7, 14, 17, 20, 27]
>>> Ciudades = [“La Plata”, “Rosario”, “Carlos Paz”]
>>> NumerosPrimos = [1,2,3,5]
>>> MiAuto = [“Citroen”, 2007, “Azul”]
```

Como pueden ver arriba, una lista puede ser una colección de cualquier tipo de objeto. Python provee varias funciones útiles que permiten la manipulación de estas listas. Abajo mostraremos algunos ejemplos usando las variables definidas arriba:

```
>>> len(N)
5
>>>len(L)
```

```
0
>>> N + NumerosPrimos
[7, 14, 17, 20, 27, 1, 2, 3, 5]
>>> Ciudades[0]
La Plata
>>> NumerosPrimos[1:3]
[2, 3]
>>> 2 in NumerosPrimos
True
>>> 190 in NumerosPrimos
False
```

Del ejemplo de arriba, pueden ver que la función `len` toma una lista y devuelve el largo o el número de objetos en la lista. Una lista vacía tiene cero objetos en ella. También pueden acceder a elementos individuales en la lista usando la operación de indexación (como en Ciudades[0]). El primer elemento de una lista tiene índice 0 y el último elemento de una lista con `n` elementos tendrá un índice `n-1`. Pueden concatenar las dos listas usando el operador `'+'` para producir una nueva lista. También pueden especificar una franja en la operación de indexación (como en NumerosPrimos[1:3]) para hacer referencia a la sublista que contiene elementos del índice 1 al 2 (uno menor que 3). También pueden formar condiciones `True` / `False` para chequear si el objeto está en una lista o no, usando el operador `in`. Estas operaciones se sintetizan con más detalle al final del capítulo.

Además de las operaciones de arriba, Python también provee varias otras operaciones de listas. Aquí presentamos ejemplos de algunas operaciones de listas útiles: `sort, reverse y append.`

```
>>> NumerosPrimos
[1, 2, 3, 5]
>>> NumerosPrimos.sort()
>>> NumerosPrimos
[1, 2, 3, 5]
>>> NumerosPrimos.reverse()
>>> NumerosPrimos
[5, 3, 2, 1]
>>> NumerosPrimos.append(11)
>>> NumerosPrimos
[5, 3, 2, 1, 11]
```

`sort` reacomoda los elementos en la lista en forma ascendente. `reverse` revierte el orden de los elementos en la lista, y `append` agrega un elemento al final de la lista. Algunas otras operaciones de listas útiles se presentan al final del capítulo. Recuerden que las listas son también secuencias y por lo tanto pueden ser usadas para llevar a cabo repeticiones. Por ejemplo:

```
>>> Ciudades = ["La Plata", "Rosario", "Carlos Paz"]
>>> for ciudad in Ciudades:
        print ciudad

La Plata
Rosario
Carlos Paz
```

La variable ciudad toma valores subsecuentes en la lista Ciudades y las sentencias dentro del loop se ejecutan una vez por cada valor en ciudad. Recuerden que escribimos loops o iteraciones de la siguiente manera:

```
for I in range(5):
    <hacer algo>
```

La función range devuelve una secuencia de números:

```
>>> range(5)
[0, 1, 2, 3, 4]
```

Por lo tanto la variable `I` toma valores en la lista `[0, 1, 2, 3, 4]` y, como en el ejemplo de abajo, el *loop* se ejecuta 5 veces:

```
>>> for I in range(5):
    print I

0
1
2
3
4
```

También recuerden que los *strings* son secuencias. Esto significa que el *string:*

```
ABC = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

es una secuencia de 26 letras. Pueden escribir un *loop* que tome cada letra individual del *string* y lo diga en voz alta de la siguiente manera:

```
>>> for letra in ABC:
      speak(letra)
```

También hay algunas funciones útiles que convierten *strings* en listas. Digamos que tenemos un *string* que contiene el enunciado:

```
>>> oracion = "Mi planta de naranja lima"
```

Pueden convertir el *string* de arriba en palabras individuales usando la operación `split`.

```
>>> oracion.split()
['Mi', 'planta', 'de', 'naranja', 'lima']
```

Observando la lista de operaciones presentadas arriba, revisen algunos de los ejemplos de percepción de los inicios de este capítulo. Estaremos usando listas en muchos ejemplos de lo que queda del texto. Por ahora, volvamos al tema de la percepción.

## ¿Percepción Extrasensorial?

Han visto muchas maneras de adquirir información sensorial usando los sensores del robot. Además del robot en sí mismo, deberían ser conscientes de que su computadora también tiene varios "sensores"o dispositivos para adquirir todo tipo de datos. Por ejemplo, ya han visto cómo, usando la función de input, pueden hacer input de algunos valores en sus programas Python:

```
>>> N = input("Ingresa un numero: ")
Ingresa un numero: 42
>>> print N
42
```

Sin duda, hay otras maneras en que pueden adquirir información para sus programas Python. Por ejemplo, pueden obtener datos de un archivo en su carpeta. En el capítulo 1 también han visto cómo pudieron controlar el robot usando el controlador de *game pad*. El *game pad* fue efectivamente enchufado en su computadora y funcionó como un dispositivo de entrada. Adicionalmente, su computadora está seguramente conectada a Internet, a través de la cual pueden acceder a muchas páginas Web. También es posible obtener el contenido de cualquier página Web usando Internet. Tradicionalmente, en ciencias de la computación se hace referencia a esto como un proceso de entrada o *input*. Desde esta perspectiva, obtener información sensorial de un robot es sólo un tipo de entrada. Dado que tenemos a nuestra disposición todas las facilidades de entradas provistas por la computadora, podemos de la misma manera obtener entradas fácilmente de cualquiera de las modalidades y combinarlas con el comportamiento de robot que deseemos. Si consideran a esto como *percepción extrasensorial* o no es una cuestión de opinión. De todas maneras, el poder obtener entradas de diversos tipos de fuentes puede conducir a ciertos aplicativos de computadora y de robot muy interesantes y útiles.

## Controladores Game Pad


El controlador *game pad* que usaron en el capítulo 1 es un dispositivo típico que provee facilidades de interacción al jugar con juegos de computadora. Estos dispositivos han sido lo suficientemente estandarizados como para que, al igual que con el *mouse* de una computadora o un teclado, ustedes puedan comprarlos en un negocio y conectarlo a un puerto USB de su computadora. Myro provee algunas funciones de entradas muy útiles que pueden ser utilizadas para obtener input del controlador *game pad*. Los *game pads* vienen de todos los sabores y configuraciones con números variados de botones, ejes y otros dispositivos incluidos. En los ejemplos de abajo, nos restringiremos al *game pad* básico que se muestra en la figura.

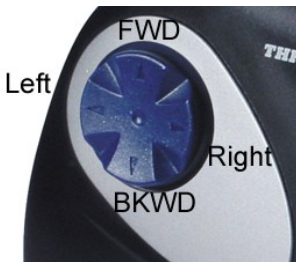
El game pad básico tiene ocho botones (numerados del 1 al 8 en la imagen) y un eje controlador (ver imagen). Los botones pueden ser presionados o soltados) on/off), lo cual se representa con 1 (para on – *encendido*) y 0 (para off –*apagado*). El eje puede ser presionado hacia muchas orientaciones distintas, representadas por un par de valores (para el eje-x y el eje-y) que van del –1.0 al 1.0, donde [0.0, 0.0] representa a ninguna actividad en el eje. Dos funciones Myro son provistas para acceder a los valores de los botones y el eje:

```
getGamepad(<device>)
getGamepadNow(<device>)
```

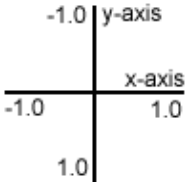
devuelve los valores que indican el estado del `<device>` (dispositivo) especificado. `<device>` puede ser `"axis"` o `"button"`.

La función `getGamepad` regresa solamente después de que `<device>` ha sido utilizado por el usuario. Esto significa que espera a que el usuario presione o utilice el dispositivo y luego devuelve los valores asociados con ese dispositivo al instante. `getGamepadNow` no espera y simplemente devuelve el estado del dispositivo enseguida. Aquí hay algunos ejemplos:

**Control de eje del Game Pad**




**Eje del Game Pad**




```
>>> getGamepadNow("axis")
[0.0, 0.0]
>>> getGamepad("axis")
[0.0, -1.0]
>>> getGamepadNow("button")
[0, 0, 0, 0, 1, 1, 0, 0]
```

Ambos, `getGamepad` y `getGamepadNow`, devuelven el mismo grupo de valores: los valores del eje (*axis*) son devueltos en forma de lista [x-axis, y-axis] (ver imagen para orientarse), y los valores son devueltos como una lista de 0 y 1. El primer valor en la lista es el estado del botón #1, seguido por 2, 3 y así sucesivamente. (Ver la imagen arriba con la numeración de los botones).

**Realizar la siguiente actividad**: Conecten el controlador *game pad* a su computadora, inicien Python e importen el módulo Myro. Prueben los comandos del *game pad* de arriba y observen los valores. Aquí hay otra manera de comprender mejor la operación del *game pad* y las funciones del mismo.

```
while timeRemaining(30):
    print getGamepad("button")
```

Prueben diferentes combinaciones de botones. ¿Qué sucede cuando presionan más de un botón? Repitan lo de arriba para el control del eje y observen los valores que devuelve (tengan a mano el diagrama del eje para poder orientarse).

El controlador *game pad* puede ser usado para todo tipo de fines interactivos, especialmente para el control del robot, así como también para escribir juegos de computadoras (ver capítulo X). Escribamos un controlador de robot basado en el *game pad*. Ingresen el siguiente programa y háganlo correr:

```
def main():
    # A simple game pad based robot controller
    while timeRemaining(30):
        X, Y = getGamePadNow("axis")
        motors(X, Y)
```

El programa de arriba correrá por 30 segundos. En ese tiempo repetidamente tomará muestras del controlador del eje y dado que esos valores están en el rango –1.0..1.0, los usará para manejar los motores. Cuando ejecuten ese programa, observen cómo se mueve el robot respondiendo a la presión de distintas partes del eje. ¿Los movimientos del robot corresponden a las direcciones que se muestran en la foto del *game pad* de la página anterior? Intenten modificar el comando de `motors` a `move` (recuerden que el movimiento requiere dos valores: traslación y rotación). ¿Cómo se comporta con respecto a los ejes? Intenten modificar el comando a `move(-X, -Y)`. Observen el comportamiento.

Como pueden ver a partir del ejemplo sencillo de arriba, es fácil combinar las entradas de un *game pad* para controlar al robot. ¿Pueden expandir el programa de arriba para que se comporte exactamente como la función del controlador `gamepad` que usaron en el capítulo 1? (ver ejercicio YY).

## La World Wide Web

Si su computadora está conectada a Internet, también pueden usar las facilidades Python para acceder al contenido de cualquier página Web y utilizarlo como input de su programa. Las páginas Web están escritas usando lenguaje *markup* como HTML, por lo tanto al acceder al contenido de una página Web, obtendrán el contenido con los *markups* incluidos. Es esta sección les mostraremos cómo acceder al contenido de una simple página Web e imprimirlo. Más adelante, veremos cómo podrían usar la información que contiene para realizar futuros procesamientos.

Accedan a un navegador Web y entren a la página:

```
http://www.fi.edu/weather/data/jan07.txt
```

Esta página Web está *hosteada* por el Instituto Franklin de Filadelfia y contiene el registro diario de datos meteorológicos de Filadelfia para enero 2007. Pueden navegar desde la dirección de arriba a otras páginas del sitio y mirar los datos del clima de otras fechas (¡los datos llegan hasta 1872!). Más abajo, les mostraremos cómo, usando la librería Python llamada `urllib`, pueden acceder fácilmente al contenido de cualquier página Web. La librería `urllib` provee una función útil llamada `urlopen`; usando esta función, pueden acceder a cualquier página de Internet de la siguiente manera:

```
>>> from urllib import *
>>> Data = urlopen("http://www.fi.edu/weather/data/jan07.txt")
>>> print Data.read()

January 2007

Day   Max   Min   Liquid      Snow  Depth

1     57    44    1.8   0     0

2     49    40    0     0     0
```

```
3      52     35     0      0      0

…      …      …      …      …      …

31     31     22     0      0      0

#days 31

Sum    1414   1005   4.18   1.80   1.10
```

Los siguientes comandos son importantes:

```
>>> Data = urlopen("http://www.fi.edu/weather/data/jan07.txt")
>>> print Data.read()
```

El primer comando usa la función `urlopen` (la cual es importada del `urllib`) para establecer una conexión entre el programa y la página Web. El segundo comando enuncia un `read` (leer) para leer de esa conexión. Lo que sea leído de esa página se imprime como resultado del comando print.

## Algo más acerca de las funciones Python

Antes de que prosigamos, sería bueno refrescar la memoria sobre la escritura de comandos y funciones Python. En el capítulo 2 aprendimos que la sintaxis básica para definir nuevos comandos/funciones es:

```
def <FUNCTION NAME>(<PARAMETERS>):
    <SOMETHING>
    ...
    <SOMETHING>
```

El módulo Myro les provee varias funciones útiles (`forward`, `turnRight`, etc.) que permiten un fácil control de los comportamientos básicos del robot. Además, usando la sintaxis de arriba, aprendieron a combinar estos comportamientos básicos para generar comportamientos más complejos (como `wiggle`, `yoyo`, etc.). Usando parámetros, pueden personalizar aún más el comportamiento de las funciones proveyendo diferentes valores para los parámetros (por ejemplo, `forward(1.0)` moverá el robot más rápido que `forward(0.5)`). También deberán notar una diferencia crucial entre los comandos de movimiento como `getLight` o `getStall, etc.` Los comandos sensoriales también siempre devuelven un valor donde sea que se hayan enunciado. Esto significa:

```
>>> getLight('left')
221
>>> getStall()
0
```

Los comandos que devuelven un valor al ser invocados se llaman funciones, dado que de hecho se comportan de manera muy similar a las funciones matemáticas. Ninguno de los comandos de

movimiento devuelve un valor, pero son útiles de otras maneras. Por ejemplo, hacen que el robot haga algo. En cualquier programa, en general se necesitan ambas funciones: aquellas que hacen algo pero no devuelven nada como resultado; y aquellas que hacen algo y devuelven un valor. En Python todas las funciones devuelven un valor. Pueden ver la utilidad de tener estos dos tipos de funciones de los ejemplos que han visto hasta ahora. Las funciones son una parte integral o crítica de cualquier programa y parte del aprendizaje de ser un buen programador es aprender a reconocer las abstracciones que pueden ser empaquetadas en funciones individuales (como `dibujoPoligono` o `giro`) que pueden ser usadas una y otra vez.

## Escribir funciones que devuelven valores

Python provee una sentencia de devolución (`return`) que pueden usar dentro de una función para devolver o retornar los resultados de una función. Por ejemplo:

```
def triple(x):
   # Retorna x*3
   return x * 3
```

La función de arriba puede ser usada igual que las que han venido usando:

```
>>> triple(3)
9
>>> triple(5000)
15000
```

La forma general de la sentencia `return es:`

```
return <expression>
```

Es decir, la función en la cual se encuentra esta sentencia devolverá el valor de <expression>. Por lo tanto, en el ejemplo de arriba, la sentencia return devuelve el valor de la expresión 3*x, tal como se muestra en las invocaciones de ejemplo. Otorgándole distintos valores al parámetro x, la función simplemente lo triplica. Esta es la idea que usamos en la normalización de valores de sensores de luz en los ejemplos anteriores en los que definimos la función normalizar para tomar los valores de los sensores de luz y normalizarlos al rango de 0.0..1.0 relativo a los valores de luz ambiental observados:

```
# This function normalizes light sensor values to 0.0..1.0
def normalize(v):
    if v > Ambient:
        v = Ambient

    return 1.0 – v/Ambient
```

Al definir la función de arriba, también estamos usando una nueva sentencia Python: la sentencia-if. Esta sentencia permite toma de decisiones simples dentro de programas de computadora. La forma más simple de una sentencia-if tiene la siguiente estructura:

```
if <CONDITION>:
   <do something>
   <do something>
   ...
```

Es decir, si la condición especificada por <CONDITION> es True, entonces lo que sea que esté especificado en el cuerpo de la sentencia-if se lleva a cabo. En el caso en que <condition> sea False, todos las sentencias para el comando if se saltean.

Las funciones pueden tener cero o más sentencias-return. Algunas de las funciones que han escrito, como wiggle, no tienen ninguno. Técnicamente, cuando una función no tiene una sentencia-return que devuelva un valor, la función devuelve un valor especial llamado None (ninguno). Esto ya está definido en Python.

Las funciones, como han visto, pueden ser usadas para empaquetar cómputos útiles y pueden ser utilizadas una y otra vez en muchas situaciones. Antes de concluir esta sección, les daremos otro ejemplo de una función. Recuerden, del capítulo 4, el comportamiento del robot que permite que el robot se mueva hacia delante hasta chocarse contra una pared. Uno de los fragmentos de programas que usamos para especificar este comportamiento se muestra abajo:

```
while not getStall():
   forward(1)
stop()
```

En el ejemplo de arriba, estamos usando el valor devuelto por `getStall` para ayudarnos a tomar la decisión de continuar avanzando o parar. Tuvimos suerte de que el valor devuelto es directamente utilizable para nuestra toma de decisión. A veces, se debe hacer un poco de interpretación de los valores del sensor para averiguar exactamente lo que el robot está percibiendo. Podrán observar esto en el caso de los sensores de luz. Aunque las sentencias de arriba son fáciles de leer, podemos mejorarlas escribiendo una función llamada `stuck()` de la siguiente manera:

```
def stuck():
   # Is the robot stalled?
   # Returns True if it is and False otherwise.

   return getStall() == 1
```

La función de arriba es bastante simple, dado que `getStall` ya nos brinda un valor utilizable (`0`/`False` o or `1`/`True`). Pero ahora, si usáramos `stuck` para escribir el comportamiento del robot, se leería así:

```
while not stuck():
   forward(1)
stop()
```

Como pueden observar, se lee mucho mejor. En este sentido, programar se parece mucho a escribir. Como en la escritura, hay varias maneras de expresar las ideas en palabras. Algunas son mejores y más legibles que otras. Algunas son directamente poéticas. De la misma manera, en programación, la expresión de algo en un programa puede ser formulada de distintas maneras, algunas mejores y más legibles que otras. La programación no se trata sólo de funcionalidad, *puede* haber poesía en el modo en que uno escribe un programa.

## Resumen

En este capítulo han aprendido todo acerca de la obtención de datos del sistema perceptivo del robot para percepción visual (fotos), percepción de luz y de proximidad. El Scribbler provee un rico conjunto de sensores que pueden ser usados para diseñar interesantes comportamientos del robot. También aprendieron que la percepción es equivalente a la operación de entrada básica en una computadora. También han aprendido cómo obtener input de un *game pad*, de la World Wide Web, y de archivos de datos. Los programas pueden ser escritos para usar creativamente las modalidades de entradas disponibles para definir comportamientos de robot, juegos de computadora, e incluso para el procesamiento de datos. En el resto del libro aprenderán cómo escribir programas que usan estas modalidades de input de muy distintas maneras.

## Revisión Myro

`getBright()`
Devuelve una lista que contiene los tres valores de todos los sensores de luz.

`getBright(<POSITION>)`
Devuelve el valor actual del sensor de luz `<POSITION>`. `<POSITION>` puede ser izquierda, centro o derecha o uno de los números 0, 1, 2.

`getGamepad(<device>)`
`getGamepadNow(<device>)`
Devuelve los valores que indican el estado del `<device>` especificado. `<device>` puede ser `"axis" o "button". La función getGamepad` espera un evento antes de devolver valores. `getGamepadNow` inmediatamente devuelve el estado actual del dispositivo.

`getIR()`
Devuelve una lista que contiene los dos valores de todos los sensores IR.

`getIR(<POSITION>)`
Devuelve el valor actual en el sensor IR `<POSITION>, <POSITION>` puede ser izquierda o derecha o uno de los números 0, 1.

`getLight()`
Devuelve una lista que contiene los tres valores de todos los sensores de luz

`getLight(<POSITION>)`
Devuelve el valor actual en el sensor de luz `<POSITION>. <POSITION>` puede ser izquierda, centro, derecha o uno de los números 0, 1, 2. Las posiciones 0, 1 y 2 corresponden a los sensores de la izquierda, el centro y la derecha.

`getObstacle()`
Devuelve una lista que contiene los dos valores de los sensors IR.

`getObstacle(<POSITION>)`
Devuelve el valor actual en el sensor IR `<POSITION>. <POSITION> puede ser` izquierda, centro o derecha, o uno de los números 0, 1, 2.

`savePicture(<picture>, <file>)`
`savePicture([<picture1>, <picture2>, …], <file>)`
Guarda la foto en el archivo especificado. La extensión del archivo debe ser `".gif"` o `".jpg"`. Si el primer parámetro es una lista de fotos, el nombre del archivo debe tener la extensión `".gif"` y se creará un GIF animado usando las fotos provistas.

`senses()`
Muestra los valores de los sensores del Scribbler en una ventana. La vista se actualiza cada segundo.

`show(<picture>)`
Muestra la foto en una ventana. Pueden clickear el mouse izquierdo en cualquier lugar de la pantalla para mostrar los valores (x, y) y (r, g, b) del punto en la barra de estado de la ventana.

```
takePicture()
takePicture("color")
takePicture("gray")
```
Toma una foto y devuelve un objeto foto. Cuando no se especifican parámetros, la foto es color.

## Revisión Python

```
if <CONDITION>:
   <statement-1>
  ...
   <statement-N>
```

Si la condición se evalúa como True, todos los enunciados se llevan a cabo. De lo contrario, todos los enunciados se saltean.

`return <expression>`
Puede ser usada dentro de cualquier función para devolver el resultado de la función.

`<string>.split()`
Divide a `<string>` en una lista.

`urlopen(<URL>)`
Establece una conexión *stream* con la <URL>. Esta función se importa del módulo Python `urlopen`.

`<stream>.read()`
Lee los contenidos enteros del `<stream>` como un *string*.

**Listas:**
[] es una lista vacía..

`<list>[i]`
Devuelve el elemento `iceavo` de la lista. La indexación comienza de 0.

`<value> in <list>`
Devuelve `True` si `<value>` está en la `<list>`, si no, `False`.

`<list1> + <list2>`
Concatena `<list1>` y `<list2>`.

`len(<list>)`
Devuelve el número de elementos en una lista.

`range(N)`
Devuelve una lista de números de 0...N

`range(N1, N2)`
Devuelve una lista de números comenzando desde N1..(N2-1)

`range(N1, N2, N3)`
Devuelve una lista de números iniciada en N1 y menor que N3, incrementándose por N3.

`<list>.sort()`
Distribuye la `<list>` en orden ascendente.

`<list>.append(<value>)`
Agrega el `<value>` al final de la `<list>`.

`<list>.reverse()`
Da vuelta los elementos de la lista.

## Ejercicios

**1** Además de texto, el comando speak también puede vocalizar números. Prueben `speak(42)` y también `speak(3.1419)`. Prueben algún número entero muy largo, como `speak(4537130980)`. ¿Cómo lo vocaliza? ¿Pueden averiguar el límite para la vocalización numérica?

**2.** La cámara Fluke devuelve fotos con 256x192 (= 49, 152) píxeles. Las típicas cámaras digitales se caracterizan por lo general por el número de píxeles que usan para sus imágenes. Por ejemplo, 8 mega píxeles. ¿Qué es un mega píxel? Realicen un poco de investigación por la Web para averiguar el resultado.

**3.** Todas las imágenes tomadas y guardadas usando la cámara Fluke también pueden ser mostradas en una página Web. En su navegador favorito, usen la opción de File Open -*abrir archivo*- (bajo el menú de File) y naveguen a la foto guardada por el Scribbler. Selecciónenla y visualícenla a través de la ventana del navegador. Inténtenlo con un archivo GIF animado.

**4** ¿Qué produce la siguiente expresión?:

```
L = [5]*10
```

Ingrésenla como un comando Python y observen el valor de `L`.

**5.** Modifiquen el programa de input del *game pad* en este capítulo para hacer que el controlador del eje se comporte de tal manera que cuando se presiona la parte superior del eje, el robot se adelante, y cuando se presiona la parte de abajo, vaya hacia atrás.

# 6 Comportamientos Símil-Insecto

*Entonces debes hacerme saber*
*¿Debo quedarme o debo irme?*
De la canción, *Debo quedarme o debo irme*, Mick Jones (The Clash), 1982.

*Página opuesta: Vaquita de San Antonio*
*La foto es cortesía de Jon Sullivan (www.pdphoto.org)*

Diseñar comportamientos para los robots es un desafío, pero es un proceso divertido. No hay una metodología formal o técnica a seguir. Implica creatividad, la habilidad para reconocer las fortalezas y las limitaciones del robot físico, el tipo de entorno en el cual el robot desarrollará su comportamiento, y por supuesto, el conocimiento de los paradigmas disponibles para programar comportamientos de robot. Ya han visto cómo, incluso un robot simple como el Scribbler, puede ser programado para llevar a cabo un diverso rango de comportamientos. También hemos ocupado un considerable esfuerzo hasta ahora explorando el abanico de posibles funciones que un robot puede realizar. El lugar en el que está el robot al funcionar, puede cumplir un rol importante al exhibir un comportamiento programado con éxito. En este capítulo, observaremos el comportamiento de los robots desde otro ángulo.

## Vehículos Braitenberg

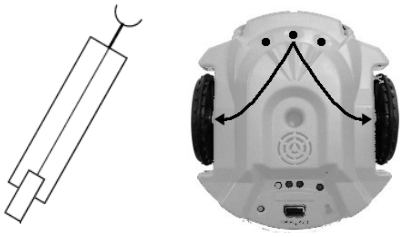
En 1984, Valentino Braitenberg escribió un libro titulado "*Vehículos: Experimentos en Psicología Sintética*" (MIT Press). En él describe varios comportamientos ideados que se centran en la creación de vehículos simples con mecanismos de control muy simples que muestran comportamientos aparentemente complejos. El objetivo de estos experimentos era ilustrar algunos aspectos esenciales de la estructura interna de los cerebros animales (y humanos). Cada experimento comprende la descripción de un vehículo simple que es provisto de un pequeño grupo de sensores (bastante similar al nuestro robot Scribbler) y el modo en que los sensores pueden estar conectados a los motores de estos vehículos imaginarios de forma análoga a las conexiones neurológicas en los animales. Muestra cómo los vehículos resultantes son capaces de realizar comportamientos complejos que pueden ser descriptos como: temor, agresión, amor, lógica, libre voluntad, etc.

Un tema central que subyace a los experimentos de Braitenberg es la demostración de lo que él llama la Ley del análisis cuesta arriba y la invención cuesta abajo: es mucho más difícil adivinar la estructura interna de una entidad con sólo observar su comportamiento, que efectivamente crear la estructura que desemboca en ese comportamiento. Es decir, intentar postular la estructura interna a partir de la pura observación es una tarea cuesta arriba (más difícil), mientras que intentar crear una entidad que exhibe cierto comportamiento es una tarea cuesta abajo (fácil). Mientras que todos los vehículos de Braitenberg eran imaginarios, y no fueron diseñados para ser efectivamente fabricados, a la gente le resultó un ejercicio divertido e intelectualmente interesante crearlos. Los robots personales como el Scribbler son excelentes plataformas para hacer esto y en lo que sigue del texto, describiremos algunos de los vehículos de Braitenberg (y de tipo-Braitenberg) y diseñaremos comportamientos para el robot basados en ellos.

## Vehículo 1: Vivo

El primer vehículo que describe Braitenberg tiene un sensor y un motor. El valor transmitido por el sensor directamente alimenta al motor. Si el valor que se reporta al sensor es una cantidad variable (digamos la luz), el vehículo se moverá a una velocidad proporcional a la suma de cantidad detectada por el sensor.

**Vehículo#1: Vivo**



Arriba se presenta un esquema del vehículo. A fin de diseñar este vehículo usando al Scribbler, pueden usar el sensor de luz del centro y conectar lo que reporte directamente a ambos motores del robot. Es decir, la misma lectura de luz está controlando directamente ambos motores en la misma cantidad. Como ya han visto, hay muchas maneras distintas de especificar comandos de movimientos del motor para el Scribbler. Supongamos que el valor obtenido del sensor de luz central es C, pueden controlar ambos motores usando este valor, al utilizar este comando:

```
motors(C, C)
```

Alternativamente, también pueden usar el comando `forward:`

```
forward(C)
```

Ahora que conocemos la estructura interna de este vehículo, podemos escribir un programa que lo implemente. Pero antes de llegar hasta este punto, debemos resolver un pequeño tema de compatibilidad: los sensores de luz reportan valores en los rangos de 0..5000, mientras que los comandos de movimiento del motor toman valores en el rango de –1.0 a 1.0. En este ejemplo, solamente nos interesan los movimientos que oscilan dentro del rango de un detenimiento completo a una velocidad máxima hacia delante, por lo tanto los valores oscilan en un rango de 0.0 a 1.0. Debemos normalizar computacionalmente, o mapear los valores del sensor de luz en este rango. Un primer intento en este sentido sería escribir una función llamada `normalizar` que opera de la siguiente manera:

```
def normalizar(v):
  # normaliza v en el rango 0.0 a 1.0
```

Una vez que tenemos esta función, podemos escribir el comportamiento del vehículo de la siguiente manera:

```
def main():
   # vehiculo de Braitenberg #1: Vivo
   while True:
      L = getLight("center")
      forward(normalizar(L))

main()
```

## Normalizar Valores del Sensor

Es hora de pensar en la tarea de la función `normalizar` (normalización). Dado un valor recibido del sensor de luz, debe transformarlo en un valor proporcional entre 0.0 y 1.0 de modo que cuanto más brillante sea la luz, más alto será el valor (ej., más cercano a 1.0). Vehículo#1 se mueve en proporción a la cantidad de luz que recibe. Es un buen momento para volver a la función `senses` de Myro para ver los valores reportados por los sensores de luz. Adelante, revisen esta función.

Luego de examinar los valores devueltos por los sensores de luz, pueden notar que reportan valores pequeños (menos de 50) para la luz brillante y valores más grandes (tanto como 3000) para la oscuridad. De alguna manera, podemos decir que el sensor de luz es en realidad un sensor de oscuridad; cuanto más oscuro está, más altos son los valores que reporta. Los sensores de luz son capaces de reportar valores entre 0 y 5000. Ahora, definitivamente podemos calibrar o normalizar usando estos valores usando la siguiente definición de `normalizar`:

```
def normalizar(v):
  # Normaliza v (en el rango 0..5000) a 0..1.0, inversamente
```

```
  return 1.0 - v/5000.0
```

Es decir, dividimos el valor del sensor de luz por su máximo valor y luego restamos esto de 1.0 (para la proporcionalidad inversa). Por lo tanto, un valor de luz más brillante, digamos un valor de 35, se normalizará como:

```
1.0 - 35.0/5000.0 = 0.9929
```

Si 0.9929 se envía a los motores (como en el programa de arriba), el robot se moverá a toda velocidad hacia delante. Computemos la velocidad del robot cuando hay total oscuridad. Cuando ponen un dedo en el centro del sensor, obtendrán valores dentro del rango de 2000-3000. Para 3000, la normalización sería:

```
1.0 - 3000.0/5000.0 = 0.40
```

El robot todavía se estará moviendo, aunque casi a media velocidad. Lo más probable es que estén operando el robot en una habitación con suficiente luz ambiental. Podrán notar que bajo condiciones de luz ambiental diurna, los valores reportados por los sensores de luz oscilan en el rango de 150-250. Al usar la normalización de arriba, obtendrán:

```
1.0 - 200.0/5000.0 = 0.9599
```

Esto es casi avanzar a toda velocidad hacia delante. Para experimentar el verdadero comportamiento del vehículo de arriba, debemos usar un esquema de normalización que tome en cuenta las condiciones de luz ambiental (variarán de habitación a habitación). Más adelante, asumamos que en condiciones de luz ambiental, veremos al robot responder a la fuente de luz que controlaremos. Una linterna funcionará bien. Entonces, para hacer que el robot sea apropiadamente sensible a la linterna bajo condiciones de luz ambiental, pueden escribir una versión mejorada de normalización como se muestra a continuación:

```
def normalizar(v):
   if v > Ambiente:
      v = Ambiente
   return 1.0 - v/Ambiente
```

Es decir, la condición más oscura está representada por el valor de luz ambiental (`Ambiente`) y luego la normalización se hace con respecto a ese valor. Pueden establecer el valor ambiental a mano, o una mejora manera es que el robot perciba la luz ambiental cuando el programa es iniciado. Esta es la misma versión de normalización que vieron en el capítulo previo. Ahora saben cómo llegamos allí. El programa completo para Vehículo#1 se muestra abajo:

```
# vehiculo de Braitenberg #1: Vivo
from myro import *
initialize("com"+ask("Qué puerto?"))

Ambiente = getLight("center")

def normalizar(v):
   if v > Ambiente:
      v = Ambiente
   return 1.0 - v/Ambiente

def main():
   #  vehiculo de Braitenberg #1: Vivo
```

```
   while True:
      L = getLight("center")
      forward(normalizar(L))
```

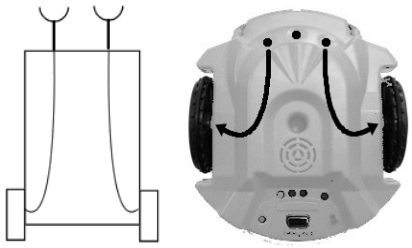
**Realizar la siguiente actividad**: Implementen el programa de arriba y observen el comportamiento del robot. ¿Responde como se describe arriba?

También habrán notado a esta altura que los tres sensores no necesariamente están sincronizados. Es decir, no reportan exactamente los mismos valores bajo las mismas condiciones. Al escribir programas para el robot que usan múltiples sensores de luz, es una buena idea promediar los valores devueltos por todos los sensores de luz para representar el valor ambiente. Modifiquen el programa de arriba para usar el promedio de los tres valores como el valor ambiente. No debería haber una diferencia notable en el comportamiento del robot. Sin embargo, esto es algo que querrán volver a usar en programas futuros.

## Vehículo 2: Cobarde y Agresivo

El siguiente grupo de vehículos utiliza dos sensores. Cada sensor directamente maneja un motor. Por lo tanto, la velocidad del motor individual es directamente proporcional a la cantidad percibida por el sensor. Hay dos formas posibles de conectar el sensor. En el primer caso, Vehículo2a, el sensor a cada lado conecta con el motor del mismo lado. En el otro caso, Vehículo2b, las conexiones están intercambiadas. Es decir, el sensor izquierdo conecta con el motor derecho y el sensor derecho conecta con el motor izquierdo. Primero diseñemos el programa de control para el Vehículo 2a:

**Vehículo 2a: Cobarde**



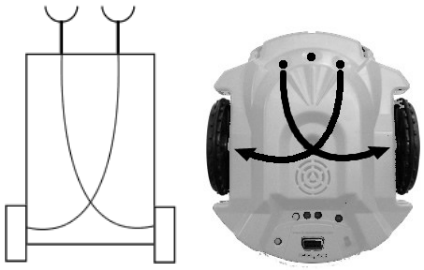
```
# vehiculo de Braitenberg #2: Cobarde
from myro import *
initialize("com"+ask("Qué puerto?"))

Ambiente = sum(getLight())/3.0

def normalizar(v):
   if v > Ambiente:
      v = Ambiente
   return 1.0 - v/Ambiente

def main():
   #  vehiculo de Braitenberg #2: Cobarde
   while True:
      L = getLight("left")
      R = getLight("right")
      motors(normalizar(L), normalizar(R))
```

La estructura del programa de arriba es muy similar a la del Vehículo#1. Hemos modificado el establecimiento del valor de la luz ambiental al de un promedio de los valores de las tres luces. También usamos el comando `motors` para manejar el motor izquierdo y el derecho en forma proporcional a los valores de los sensores de luz izquierdo y derecho (luego de su normalización).

**Realizar la siguiente actividad**: Implementar el programa de control para el Vehículo 2a como se muestra arriba. Observen el comportamiento del robot alumbrando una linterna directamente frente al robot y en cada uno de los sensores izquierdo y derecho.

Luego, escriban el programa de control para el Vehículo2b como se muestra aquí. Esto requiere un cambio simple en el programa del Vehículo2a: cambiar los parámetros del comando `motors` para reflejar las conexiones intercambiadas. Nuevamente observen los comportamientos alumbrando la linterna directamente frente al robot y también un poco en cada lado.

**Vehículo 2b: Agresivo**



Notarán que los robots se comportan del mismo modo cuando la luz se ubica frente a ellos: Ambos están atraídos a la luz y por lo tanto se mueven hacia la fuente de luz. Sin embargo, el Vehículo2a se moverá alejándose de la luz si la fuente de luz está a un costado. Dado que el sensor más cercano se excitará más, moviendo al motor correspondiente más rápido, y por lo tanto alejando al robot. En el caso del Vehículo 2b, por otra parte, siempre girará hacia la fuente de luz y se moverá hacia ella. Braitenberg llama a estos comportamientos *cobarde* (2a) y *agresivo* (2b)

## Controlando las Respuestas del Robot

Suele ser necesario, al diseñar y testear comportamientos de robot, generar apropiadamente el entorno y la orientación del robot dentro del mismo adecuadamente. En casos simples esto se logra fácilmente ubicando al robot en la orientación deseada y luego cargando y ejecutando el programa. Sin embargo, previamente al comportamiento efectivo del robot, quizás necesiten realizar algunas observaciones preliminares (por ejemplo, percibir la luz ambiental), es necesario reorientar apropiadamente al robot apropiadamente antes de iniciar la ejecución del comportamiento actual. Esto se puede lograr fácilmente incluyendo algunos comandos interactivos en el programa del robot. La estructura de programa resultante se muestra abajo:

```
# importar la librería myro, y establecer la conexión con el robot
# definir todas las funciones aquí (normalizar, etc.)
# establecer los valores para la condición de Ambiente

def main():
    # Descripción del comportamiento ...

    # Darle la oportunidad al usuario de acondicionar el robot
    askQuestion("Presione OK para comenzar...", ["OK"])

    # Escriba el comportamiento del robot aquí
```

**Realizar la siguiente actividad**: Modifiquen los programas de los vehículos 1, 2a y 2b para incluir el comando `askquestion` de arriba.

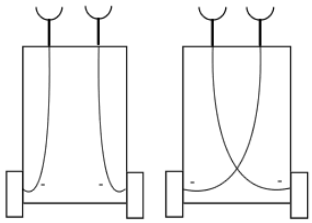
Hemos introducido algunos patrones básicos de programación arriba que pueden ser usados en muchas situaciones de programación. Lo que hay que recordar es que, en cualquier punto de la ejecución de un programa de robot, también pueden

programar apropiadamente interjecciones para realizar varias funciones experimentales o de control. Veremos varios ejemplos más adelante.

## Otras Normalizaciones

Todas las normalizaciones de los valores de sensores de luz mostradas arriba fueron usadas para normalizar los valores dentro del rango de 0.0..1.0 en proporción directa a la cantidad de luz percibida. Es decir, cuanto más oscuro está, más cercanos estarán los valores normalizados a 0.0, y cuanto más brillo haya, más cerca estarán los valores normalizados a 1.0. Esta es sólo una forma en la que podemos relacionar la cantidad percibida con la cantidad de velocidad aplicada a los motores del robot. Pueden imaginarse otro tipo de relaciones. La más obvia, por supuesto, es la relación inversa: cuanto más oscuro está, más cerca de 1.0, y viceversa. Braitenberg llama a esto una relación *inhibitoria* (como opuesto al *excitatoria*): cuanto más se percibe una cantidad, más lento funcionan los motores del robot. Como en los vehículos 2a y 2b arriba, está la opción de dos tipos de conexiones: derecha y cruzada. Estos se muestran abajo (un signo más (+) junto a un conector indica una conexión excitatoria y un signo menos (-) representa una conexión inhibitoria):

**Vehículos 3a (Amor) y 3b (Explorador)**



Escribir la función `normalize` para una conexión inhibitoria es bastante directo:

```
def normalizar(v):
   if v > Ambiente:
      v = Ambiente
   return v/Ambiente
```

Braitenberg describe el comportamiento de los vehículos resultantes como *amor* (Vehículo 3a) y *explorador* (Vehículo 3b). Es decir, si observan el comportamiento de los dos vehículos, seguramente notarán que el Vehículo 3a se detendrá de frente a la fuente de luz (muy cerca), mientras que el Vehículo 3b se detendrá de espaldas a la fuente de luz y quizás se aleje, dependiendo de la presencia de otras fuentes de luz.

En otras variantes de normalizaciones de valores de sensor, Braitenberg sugiere usar funciones matemáticas no-monotónicas. Es decir, si observan las normalizaciones excitatorias e inhibitorias, pueden ser descriptas como monotónicas: más luz, mayor velocidad del motor; o más luz, menor velocidad del motor. Observen la función presentada en la página siguiente. La relación se incrementa en proporción a la entrada sensorial pero sólo hasta cierto punto y luego decrece. La incorporación de estas relaciones en los vehículos llevará a comportamientos más complejos (Braitenberg los describe como vehículos con *instintos*). A continuación se define una función de normalización, basada en la curvatura mostrada:

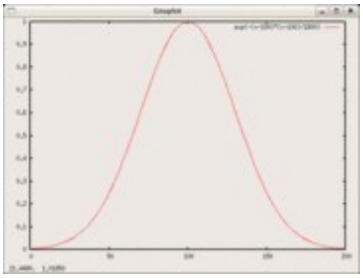
$$f(x) = e^{-(x-100)^2/1800}$$

La función de arriba está basada en una función más simple:

$$f(x) = e^{-x^2}$$

La primera definición está estirada para cubrir el rango 0..200 para valores de x con 100 como el punto donde reporta el máximo valor (por e. 1.0). Matemáticamente, generalmente esta función también se conoce como la *curva Gaussiana*. Una curva Gaussiana se define en términos de una *media* (π*) y desviación estándar* (σ) como se muestra abajo:

**Una Función No-Monotónica**



$$f(x) = e^{-(x-\pi)^2/2*\sigma^2}$$

Por lo tanto, en la función de normalización, estamos usando 100 como media y 30 como desviación estándar. Fácilmente pueden escalar la curva para el rango de los valores del sensor deseado usando la siguiente función `normalizar`:

```
def normalizar(v):
      mean = Ambiente/2.0
      stddev = Ambiente/6.0
        if v >= Ambiente:
                v = Ambiente
        return exp(-(v - mean)**2 / 2*(stddev**2))
```

`exp(x)` es una función Python que computa el valor de $e^x$ . Está disponible en la librería `math` de Python. Ahondaremos en la librería math con más detalle en el siguiente capítulo. En función de usar la función `exp` como se muestra arriba, deben importar la librería `math`:

```
from math import *
```

Hay por supuesto varias otras posibilidades que pueden probarse: una función step, un umbral; y cualquier otra combinación matemática. La idea clave es que hay un mapeo claro del rango de los valores del sensor a los valores del motor en el rango de 0.0..1.0.

Los robots que usan estas normalizaciones y otras variaciones suelen mostrar comportamientos muy interesantes y a veces impredecibles. Observadores que desconozcan los mecanismos de mapeo internos tendrán dificultades en describir con precisión el comportamiento del robot y tenderán a usar términos antropomórficos (como *amor, odio, instintos*, etc.) para describir los comportamientos del robot. Esto es lo que significa un análisis cuesta arriba.

## Sensores Múltiples

Agregar varios sensores enriquece el espacio de diseño para comportamientos del robot. Como diseñador, ahora tienen la posibilidad de elegir entre diferentes tipos de mapeos: excitatorio, inhibitorio, o más complejo; y entre conexiones: directa o cruzada. De pronto, el comportamiento de robot resultante puede parecer complejo. En el Scribbler, por ejemplo, además de los sensores de luz, también tienen el sensor de atascamiento y los sensores IR. Con la excepción de los sensores de luz, todos los otros sensores son digitales o de umbral que no son ni encendido ni apagado (ON, OFF) – por ejemplo, reportan valores que son 0 ó 1, indicando la presencia o ausencia de aquello que perciben. De alguna manera, se puede considerar que los sensores digitales ya están normalizados, pero aún es posible invertir esta relación y es necesario. Pueden diseñar varios comportamientos

interesantes combinando dos o más sensores y decidiendo si conectarlos en forma directa o cruzada.

**Realizar la siguiente actividad:** En su diseño de Vehículos 2a y 2b, sustituyan el sensor de obstáculos en lugar de los sensores de luz. Describan el comportamiento resultante de los vehículos. Intenten lo mismo para los Vehículos 3a y 3b. Luego, combinen el comportamiento de los vehículos resultantes con los sensores de luz. Prueben todas las combinaciones de conexiones, así como los mapeos inhibitorios y excitatorios. ¿Qué vehículos exhiben los comportamientos más interesantes?

## Más Vehículos

Aquí hay descripciones de varios vehículos que tienen el espíritu de los diseños de Braitenberg y también exhiben comportamientos interesantes. Usando los conceptos y las técnicas de programación de arriba, intenten implementarlos en el robot Scribbler. Una vez completado, inviten a algunos amigos a observar los comportamientos de estas criaturas y registren sus reacciones.

### *Tímido*

El tímido es capaz de moverse hacia delante en una línea recta. Tiene un sensor de luz de umbral que apunta hacia arriba. Cuando el sensor detecta luz, la criatura se mueve hacia delante, si no, se queda quieta. El umbral del sensor de luz debería estar establecido para luz ambiental. De esta manera, cuando la criatura “vea” la luz, se moverá. Al entrar en una sombra (que puede ser proyectada por una mano o cualquier otro objeto), se detendrá. Si lo que sea que está proyectando la sombra se mueve, la criatura volverá a moverse. Por eso, el tímido es un buscador de sombra.

### *Indeciso*

El indeciso es similar al tímido, excepto que nunca se detiene: sus motores siempre están funcionando, ya sea hacia delante o hacia atrás, controlados por el sensor de luz de umbral. Cuando el detector de luz percibe luz, se mueve hacia adelante, si no, se mueve hacia atrás. Al hacer funcionar a esta criatura, notarán que tiende a oscilar hacia delante y hacia atrás en los bordes de la sombra. Por eso, el indeciso es un buscador de bordes de sombra.

### *Paranoico*

El paranoico es capaz de girar. Esto se logra moviendo el motor derecho hacia delante y el motor izquierdo en dirección reversa al mismo tiempo. Tiene un solo sensor de luz de umbral. Cuando el sensor detecta luz, se mueve hacia delante. Cuando el sensor entra en la sombra, revierte la dirección del motor izquierdo, girando hacia la derecha. Pronto el sensor se dará vuelta y saldrá de la luz. Cuando esto suceda, volverá a su movimiento hacia delante. El paranoico es una criatura que teme a la sombra.

## Esto, Eso, o lo Otro

La sentencia `if` introducida en el capítulo 5 es una forma de tomar decisiones simples. Es decir, pueden controlar condicionalmente la ejecución de un conjunto de comandos basado en una condición singular. La sentencia if en Python es bastante versátil y puede ser usada para tomar decisiones múltiples. Así es como deberían usarlo para elegir entre dos grupos de comandos:

```
if <condicion>:
```

```
  <esto>
else:
  <eso>
```

Es decir, si `<condicion>` es verdadero realizará los comandos especificados en `<exto>`. Sin embargo, si `<condicion>` es falso, hará `<eso>`. De manera similar, pueden extender la sentencia if para ayudar a especificar opciones múltiples:

```
if <condicion-1>:
  <esto>
elif <condicion-2>:
  <eso>
elif <condicion-3>:
  <otra cosa >
...
else:
  <otro>
```

Noten el uso de la palabra `elif` (¡sí, se deletrea así!) para designar “else if”. Entonces, dependiendo de la condición que sea verdadera, el correspondiente `<esto> o <eso>` u `<otra cosa>` se llevará a cabo. Si el resto falla, el `<otro>` será llevado a cabo.

## Simples Comportamientos Reactivos

Al usar los tres sensores de luz, el robot puede detectar condiciones variables de luz en su entorno. Escribamos un programa de robot que lo hace detectar y orientarse hacia la luz brillante. Recuerden del capítulo 5 que los sensores de luz reportan valores bajos en condiciones de luz brillante y valores altos con luz baja. Para realizar esta tarea, solamente necesitamos mirar los valores reportados por los sensores de luz izquierdo y derecho. A continuación se describe el comportamiento del robot:

```
do para una cantidad de tiempo
  if la luz izquierda es más brillante que la derecha
    girar a la izquierda
  else
    girar a la derecha
```

De esta manera, haciendo uso de la sentencia-if, podemos refinar lo expresado arriba de la siguiente manera:

```
while timeRemaining(30):
  if la luz izquierda es más brillante que la derecha:
    turnLeft(1.0)
  else:
    turnRight(1.0)
```

Lo único que queda en los comandos de arriba es escribir la condición para detectar la diferencia entre los dos sensores de luz. Esto puede hacerse usando la expresión:

```
getLight('left') < getLight('right')
```

**Realizar la siguiente actividad**: Escriban un programa completo que implemente el comportamiento de arriba y pruébenlo en su robot.

Quizás hayan notado que aún en condiciones de luz uniformes tienden a reportar valores diferentes. Generalmente es una buena idea determinar un umbral de

diferencia al tomar la decisión de arriba. Digamos que definimos un umbral en una diferencia de por lo menos 50. Es decir que si los sensores izquierdo y derecho difieren en por lo menos 50, entonces, dobla hacia el sensor más brillante. ¿Qué ocurre si la diferencia es menor que la del umbral? Decidamos que en ese caso el robot se quedará quieto. Este comportamiento puede ser capturado de la siguiente manera:

```
umbral = 50

while timeRemaining(30):
  # Obtenemos los valores de los sensores de luz derecho e izquierdo
  L = getLight('left')
  R = getLight('right')

  # decidimos cómo actuar en base  a los valores de los sensores
  if (L - R) > umbral:
    # el izquierdo parece ser menor que el derecho
    turnRight(1.0)
  elif (R - L) > thresh:
    # el derecho parece ser menor que el izquierdo
    turnLeft(1.0)
  else:
    # la diferencia es menor que el umbral: quedarse quieto
    stop()
```

Observen cómo hemos usado la variable `umbral` para representar el valor del umbral. Esta es una buena práctica de programación. Dado que la performance de los sensores varía bajo diferentes condiciones de luz, esto les permite ajustar el umbral con simplemente cambiar ese valor. Usando el nombre `umbral` en lugar de un valor fijo, digamos 50, solamente deben hacer ese tipo de cambios en un lugar del programa.

En las sentencias de arriba hay un patrón que encontrarán recurriendo a muchos programas que definen comportamientos de robot usando decisiones simples.

```
while timeRemaing(<segundos>):
  <sensar>
  <decidir y actuar>
```

Tales comportamientos se denominan comportamientos *reactivos*. Es decir que un robot está reaccionando al cambio en su entorno tomando la decisión acerca de la manera de actuar basándose en los valores de su sensor. Un amplio rango de comportamientos de robot pueden ser escritos usando esta estructura de programa.

Abajo presentamos descripciones de varios comportamientos de robot automatizados interesantes, aunque simples. Siéntanse libres de implementar alguno de ellos en su robot.

## Comportamientos Reactivos Simples

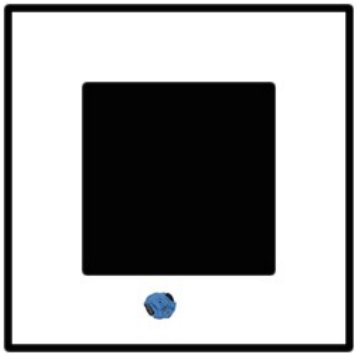
La mayoría de los comportamientos descriptos abajo requiere una selección entre alternativas usando expresiones condicionales y sentencias-if.

**Heladera detective**: Cuando eran niños ¿nunca se preguntaron si siempre estaba encendida la luz de la heladera? ¿O si se apagaba cuando cerraban la puerta? Bueno, aquí hay una forma de averiguarlo. ¡Construyan un robot detective  que se siente dentro de la heladera y les diga si la luz está encendida o apagada!

**Robot alarma contra robos**: Diseñen un robot que cuide la puerta de su pieza. Ni bien se abra la puerta, que haga sonar una alarma (un beep).

**Detector de paredes**: Escriban un programa para el robot que avance derecho y se detenga cuando detecte una pared frente a él. Estarán usando los sensores IR para esta tarea.

**Navegador de pasillo**: Imaginen a su robot en un entorno que tiene un corredor con paredes que rodea a una caja cuadrada de 60cm.  por 60cm. (ver la imagen al lado). Escriban un programa que le permita al robot avanzar alrededor de esa caja. Una estrategia que pueden usar es que el robot vaya hacia delante en una línea recta hasta que se choque contra una pared. Luego de chocar, realizará un giro de 90 grados (quizás necesiten retroceder un poco para permitir espacio para el giro) y luego continuará en línea recta.



**Dispositivo de medición**: Han calibrado al robot con respecto a cuánta distancia viaja en cierta cantidad de tiempo. Pueden usar esto para diseñar un robot que mida espacios. Escriban un programa que le permita al robot medir el ancho de un pasillo.

**El perseguidor**: Escriban un programa de robot que exhiba el siguiente comportamiento: El robot prefiere quedarse cerca de una pared (en frente). Si no tiene una pared enfrente, se mueve hacia delante hasta que la encuentra. Testen el programa en primer lugar ubicando al robot en un corralito de juego. Asegúrense de que el programa se comporte como en la descripción. A continuación, ubíquenlo en el piso y sostengan un papel blanco frente a él (lo suficientemente cerca como para que lo detecte). Ahora, lentamente, alejen el papel del robot. ¿Qué sucede?

## Diseño de Comportamientos Reactivos

La mayoría de los comportamientos que se implementan usando el estilo Braitenberg dependen de algunas cosas simples: seleccionar uno o más sensores; elegir el tipo de cableado (directo o cruzado); y seleccionar las funciones de normalización para cada sensor. Mientras que pueden adivinar el comportamiento que puede resultar de estos tres diseños, la única forma de confirmarlo es directamente observando al robot llevar a cabo estos comportamientos. También han visto cómo, usando las sentencias-if pueden diseñar comportamientos simples, aunque interesantes. En esta sección diseñaremos comportamientos reactivos adicionales.

### Seguimiento de la Luz

Para empezar, será bastante directo extender el comportamiento a partir de la orientación de luz desde arriba en otro que resulte en un robot seguidor de la luz. Es decir, con una linterna, podrán guiar al robot para que los siga. Nuevamente, tendrán que empezar por observar el rango de valores reportado por el robot bajo varias condiciones de luz. Si una linterna va a ser una fuente de luz brillante, observarán que los sensores de luz reportan muy bajos valores cuando una luz los ilumina directamente (típicamente en el rango de 0..50). Entonces, la decisión de para qué lado ir (hacia delante, doblar a la izquierda o hacia la derecha) puede tomarse basándose en las lecturas del sensor de los tres sensores de luz. La estructura del programa aparece de esta manera:

```
#Seguidor de luz

from myro import *
initialize("com"+ask("Qué puerto?"))

# Algunas configuraciones del programa

umbral = 50
velocidadAdelante = 0.8
velocidadCrucero = 0.5
velocidadGiro = 0.7     # esto girará a la izquierda, -0.7 girará a la derecha

def main():
   while True:
      # obtener los valores de los sensores izquierdo, central y derecho
      L, C, R = getLight()

      # decidimos cómo actuamos en base a los valores de los sensores
      if C < umbral:
         # luz brillante desde el centro, ir derecho
         move(velocidadAdelante, 0)
      elif L < umbral:
         # luz brillantea la izquierda, girar a la izquierda
         move(velocidadCrucero, velocidadGiro)
      elif R < umbral:
         # luz brillantea la derecha, girar a la derecha
         move(velocidadCrucero, -velocidadGiro)
      else:
         # sin luz, moverse desapacio (¿o parar?)
         move(velocidadCrucero/2, 0)
main()
```

Noten que en el programa de arriba, hemos decidido establecer valores para el umbral de luz (`umbral`) así como movimientos para valores específicos. Esto se debe a que el comando `move` nos permite mezclar movimientos de traslación y de rotación. Adicionalmente, observen que más allá de los valores del sensor, el robot siempre se mueve hacia delante un poco aún cuando está doblando. Esto es esencial dado que el robot debe seguir la luz y no sólo orientarse hacia ella. En el caso en el cual no hay una luz brillante presente, el robot aún se mueve hacia delante (a mitad de la velocidad crucero).

**Realizar la siguiente actividad**: Implementar el programa de seguimiento de luz como se describe arriba y observen el comportamiento del robot. Intenten ajustar los valores establecidos (para umbral y para velocidades de motor) y noten los cambios en los comportamientos del robot. Además, ¿observan que este comportamiento es similar a alguno de los vehículos que se describen arriba? ¿Cuál?

En el diseño del robot seguidor de luz de arriba, usamos un valor de umbral para detectar la presencia de brillo de luz. A veces es más interesante usar umbrales diferenciales para valores de sensor. Es decir, ¿el valor del sensor de luz es diferente de la luz ambiente por cierta cantidad de umbral? Pueden volver a utilizar la función `senses` y observar las diferencias de la luz ambiental y modificar el programa de arriba para usar el diferencial en lugar del umbral fijado.

Aquí hay otra idea. Junten a varios de sus compañeros de clase en una habitación con sus robots, todos corriendo el mismo programa. Asegúrense de que a habitación tenga suficiente espacio en el piso y una gran ventana con cortina.

Cierren las cortinas para bloquear temporalmente la luz externa. Ubiquen a los robots en toda la habitación e inicien el programa. El robot se escurrirá por ahí, en dirección a su orientación inicial. Ahora lentamente abran las cortinas para que entre más luz. ¿Qué ocurre?

**Esquivando Obstáculos**

Los obstáculos en el camino del robot pueden ser detectados usando los sensores IR frente al robot. Entonces, basándose en los valores obtenidos, el robot puede decidir alejarse del obstáculo que se avecina usando el siguiente algoritmo:

```
if obstacle straight ahead, turn (left or right?)
if obstacle on left, turn right
if obstacle on right, turn left
otherwise cruise
```

Esto puede ser implementado usando el programa de abajo:

```
# Esquivando Obstáculos

from myro import *
initialize("com"+ask("Qué puerto?"))

# Algunas configuraciones del programa ...

velocidadCrucero = 0.6
velocidadGiro = 0.5     # esto girará a la izquierda, -0.5 girará a la derecha

def main():
   while True:
      # obtener los valores de los sensores IR izquierdo y derechos
      L, R = getIR()
      L = 1 - L
      R = 1 - R

      # decidimos cómo actuamos en base a los valores de los sensores
      if L and R:
         # obstáculo a la vista, girar (aleatóriamente)
         move(0, velocidadGiro)
      elif L:
         # obstáculo a la izquierda, girar a la derecha
         move(velocidadCrucero, -velocidadGiro)
      elif R:
         # obstáculo a la derecha, girar a la izquierda
         move(velocidadCrucero, velocidadGiro)
      else:
         # sin obstáculos
         move(velocidadCrucero, 0)
main()
```

Como en el caso del perseguidor de luz, observen que empezamos estableciendo valores para los movimientos. Adicionalmente, hemos dado vuelta los valores de los sensores IR para que las condiciones de las sentencias-if parezcan más naturales. Recuerden que los sensores IR reportan un valor de 1 en ausencia de un obstáculo y un 0 en presencia de uno. Al invertirlos (usando valor –1), el valor para un obstáculo es 1 y en caso contrario, es 0. Estos valores hacen que resulte más natural escribir

las condiciones en el programa de arriba. Recuerden que en Python un 0 es equivalente a `False` (falso) y un 1 es equivalente a `True`. Lean el programa de arriba cuidadosamente y asegúrense de entender estas sutilezas. Más allá de esto, la estructura del programa es muy similar a la del programa del perseguidor de luz.

Otra forma de escribir un comportamiento de robot similar es usando el valor del sensor de atascamiento. Recuerden que el sensor de atascamiento detecta si el robot se ha chocado contra algo. Entonces, pueden escribir un comportamiento que no necesariamente evita obstáculos, pero que recorre el entorno chocando con las cosas. Es similar a una persona que entra en una habitación oscura e intenta sentir su avance tocando o chocándose lentamente contra las cosas. En el caso del robot, no hay forma de saber si el choque fue en su izquierda o derecha. Sin embargo, si usan el programa (mostrado abajo) observarán un comportamiento bastante robusto por parte del robot.

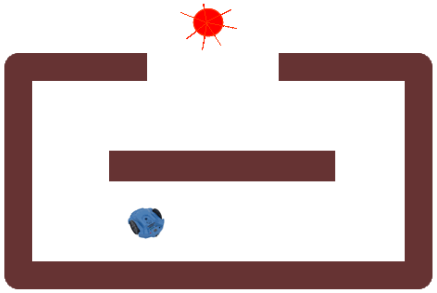
```
# Evitando obstáculos

from myro import *
initialize("com"+ask("Qué puerto?"))

# Algunas configuraciones del programa ...

velocidadCrucero = 1.0
velocidadGiro = 0.5     # esto girará a la izquierda, -0.5 girará a la derecha

def main():
   while True:

      if getStall():
         # Estoy trabado, girar (aleatoriamente?)
         move(0, velocidadGiro)
      else:
         # No estoy trabado
          move(velocidadCrucero, 0)

main()
```

En ciertos momentos, notarán que el robot se atasca aún al intentar doblar. Una solución es detener al robot, retroceder un poco y luego doblar.

**Realizar la siguiente actividad**: Implementen el programa de arriba, observen el comportamiento del robot. A continuación, modifiquen el programa sugerido arriba (al quedar atascado, detenerse, retroceder y luego doblar).

**Robot que resuelve laberintos**: Creen un laberinto simple para el robot. Ubiquen al robot en una punta y usen los programas para esquivar obstáculos de arriba (ambas versiones). ¿El robot resuelve el laberinto? Si no lo hace, observen si el laberinto es para zurdos o derechos (por ej, todos los giros son para la derecha o para la izquierda), o ambos. Modifiquen los programas para esquivar obstáculos para resolver laberintos de derechos o de izquierdos. ¿Cómo le permitirían al robot solucionar un laberinto que tiene giros hacia la derecha y la izquierda?

¿Corral OK?



## Salida del Corral

Dado que un simple programa para evitar obstáculos le permite a un robot resolver un laberinto sencillo, podemos también diseñar comportamientos más interesantes sobre esto. Imaginen un corral: un área cerrada con particiones tipo laberinto y una entrada, con una fuente de luz en la entrada (ver imagen a la derecha). Dada la posición del robot, ¿podemos diseñar un comportamiento que le permita al robot salir del corral?

Se puede diseñar una solución para el corral específico que se muestra aquí: que siga cualquier pared hasta que vea una luz brillante y luego pasar a buscador de luz. ¿Puede el Scribbler ser diseñado para seguir una pared? Recuerden que el Fluke tiene sensores de obstáculo izquierdo y derecho que apuntan a sus costados. Otro abordaje será combinar el comportamiento de esquivar obstáculos de arriba con el comportamiento de buscador de luz. Es decir, en ausencia de luz brillante, el robot se moverá alrededor del corral, evitando obstáculos, y cuando perciba una luz brillante, se dirigirá hacia ella. La parte difícil aquí será detectar que ha salido del corral y que necesita detenerse.

## Resumen

Braitenberg utiliza ideas muy simples para pensar el modo en que están conectados los cerebros y cuerpos de humanos y animales. Por ejemplo, en los humanos, los nervios ópticos (además de otros) tienen conexiones cruzadas dentro del cerebro. Es decir que los nervios del ojo izquierdo se conectan con el lado derecho del cerebro y viceversa. De hecho, se cruzan y cierta información de cada lado también está representada en el mismo lado (es decir, hay conexiones directas, además de cruzadas). Sin embargo, aún es un enigma para los científicos en por qué es así y cuáles, si es que hay alguna, son las ventajas y desventajas de este esquema. De modo similar, si observamos el comportamiento de los Vehículos 2a y 2b, fácilmente se puede ver allí paralelos con el comportamiento de varios animales, como las moscas que se orientan hacia fuentes de luz o de calor. Simples comportamientos de robot nos puede proveer profundas comprensiones del comportamiento complejo: que la observación y el análisis de algo es una tarea cuesta arriba si no se conoce la estructura interna. Construyendo estructuras internas simples podemos arribar a comportamientos de apariencia complejos. Se ha visto que estos comportamientos de apariencia complejos también influencian comportamientos grupales en los insectos (ver la imagen de artículo en la página siguiente). Es decir, que robots que no se parecen en nada a los insectos y de tamaño parecido al del Scribbler pueden ser usados para influenciar comportamientos de insectos en muchas situaciones.

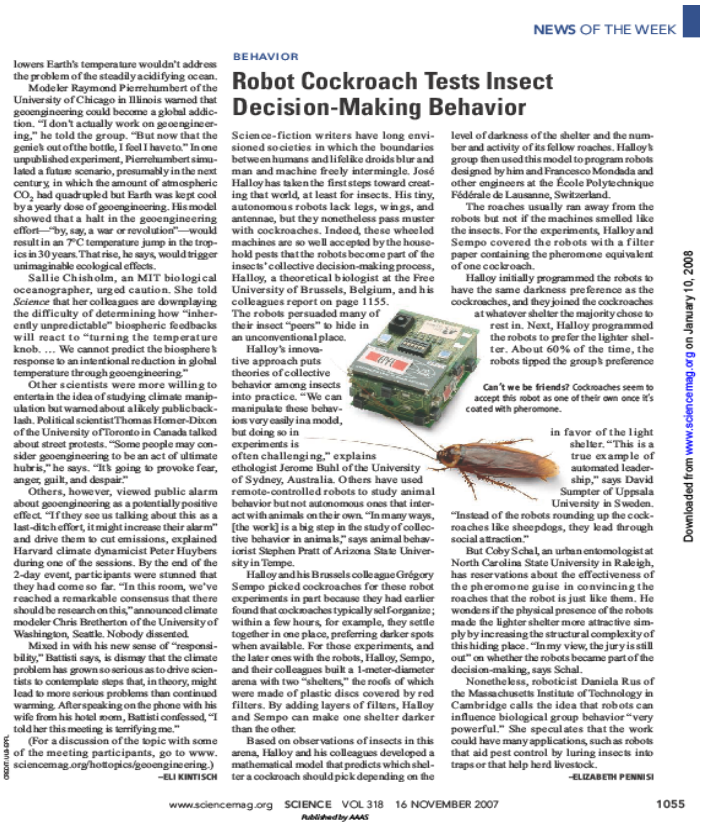
En este capítulo, hemos intentado darles un gustito de la idea de psicología sintética. A su vez, han aprendido cómo programar estructuras internas en un cerebro de robot y han aprendido varias técnicas para el control del robot.

## Referencias

Todos los vehículos numerados descriptos aquí fueron desarrollados en un conjunto de experimentos pensados diseñados por Valentino Braitenberg en su libro, "*Vehicles: Experiments in Synthetic Psychology*" *(Vehículos: Experimentos en Psicología Sintética)*, MIT Press, 1984.

Algunos de los otros vehículos descriptos aquí fueron diseñados por David Hogg, Fred Martin, y Mitchel Resnickdel Laboratorio de Medios del MIT. Hogg et al usaron ladrillos LEGO especializados para construir estos vehículos. Para más detalles, ver su artículo titulado "*Braitenberg Creatures*" *(Criaturas de Britenberg)*.

Artículo de ***Science*** **Magazine, enero 10, 2008**



NEWS OF THE WEEK

BEHAVIOR

## Robot Cockroach Tests Insect Decision-Making Behavior

lowers Earth's temperature wouldn't address the problem of the steadily acidifying ocean.

Modeler Raymond Pierrehumbert of the University of Chicago in Illinois warned that geoengineering could become a global addiction. "I don't actually work on geoengineering," he told the group. "But now that the genie's out of the bottle, I feel I have to." In one unpublished experiment, Pierrehumbert simulated a future scenario, presumably in the next century, in which the amount of atmospheric $CO_2$ had quadrupled but Earth was kept cool by a yearly dose of geoengineering. His model showed that a halt in the geoengineering effort—"by, say, a war or revolution"—would result in an 7°C temperature jump in the tropics in 30 years. That rise, he says, would trigger unimaginable ecological effects.

Sallie Chisholm, an MIT biological oceanographer, urged caution. She told *Science* that her colleagues are downplaying the difficulty of determining how "inherently unpredictable" biospheric feedbacks will react to "turning the temperature knob. … We cannot predict the biosphere's response to an intentional reduction in global temperature through geoengineering."

Other scientists were more willing to entertain the idea of studying climate manipulation but warned about a likely public backlash. Political scientist Thomas Homer-Dixon of the University of Toronto in Canada talked about street protests. "Some people may consider geoengineering to be an act of ultimate hubris," he says. "It's going to provoke fear, anger, guilt, and despair."

Others, however, viewed public alarm about geoengineering as a potentially positive effect. "If they see us talking about this as a last-ditch effort, it might increase their alarm" and drive them to cut emissions, explained Harvard climate dynamicist Peter Huybers during one of the sessions. By the end of the 2-day event, participants were stunned that they had come so far. "In this room, we've reached a remarkable consensus that there should be research on this," announced climate modeler Chris Bretherton of the University of Washington, Seattle. Nobody dissented.

Mixed in with his new sense of "responsibility," Battisti says, is dismay that the climate problem has grown so serious as to drive scientists to contemplate steps that, in theory, might lead to more serious problems than continued warming. After speaking on the phone with his wife from his hotel room, Battisti confessed, "I told her this meeting is terrifying me."

(For a discussion of the topic with some of the meeting participants, go to www.sciencemag.org/hottopics/geoengineering.)

–ELI KINTISCH

Science-fiction writers have long envisioned societies in which the boundaries between humans and lifelike droids blur and man and machine freely intermingle. José Halloy has taken the first steps toward creating that world, at least for insects. His tiny, autonomous robots lack legs, wings, and antennae, but they nonetheless pass muster with cockroaches. Indeed, these wheeled machines are so well accepted by the household pests that the robots become part of the insects' collective decision-making process, Halloy, a theoretical biologist at the Free University of Brussels, Belgium, and his colleagues report on page 1155.

The robots persuaded many of their insect "peers" to hide in an unconventional place.

Halloy's innovative approach puts theories of collective behavior among insects into practice. "We can manipulate these behaviors very easily in a model, but doing so in experiments is often challenging," explains ethologist Jerome Buhl of the University of Sydney, Australia. Others have used remote-controlled robots to study animal behavior but not autonomous ones that interact with animals on their own. "In many ways, [the work] is a big step in the study of collective behavior in animals," says animal behaviorist Stephen Pratt of Arizona State University in Tempe.

Halloy and his Brussels colleague Grégory Sempo picked cockroaches for these robot experiments in part because they had earlier found that cockroaches typically self-organize; within a few hours, for example, they settle together in one place, preferring darker spots when available. For those experiments, and the later ones with the robots, Halloy, Sempo, and their colleagues built a 1-meter-diameter arena with two "shelters," the roofs of which were made of plastic discs covered by red filters. By adding layers of filters, Halloy and Sempo can make one shelter darker than the other.

Based on observations of insects in this arena, Halloy and his colleagues developed a mathematical model that predicts which shelter a cockroach should pick depending on the level of darkness of the shelter and the number and activity of its fellow roaches. Halloy's group then used this model to program robots designed by him and Francesco Mondada and other engineers at the École Polytechnique Fédérale de Lausanne, Switzerland.

The roaches usually ran away from the robots but not if the machines smelled like the insects. For the experiments, Halloy and Sempo covered the robots with a filter paper containing the pheromone equivalent of one cockroach.

Halloy initially programmed the robots to have the same darkness preference as the cockroaches, and they joined the cockroaches at whatever shelter the majority chose to rest in. Next, Halloy programmed the robots to prefer the lighter shelter. About 60% of the time, the robots tipped the group's preference

Can't we be friends? Cockroaches seem to accept this robot as one of their own once it's coated with pheromone.

in favor of the light shelter. "This is a true example of automated leadership," says David Sumpter of Uppsala University in Sweden. "Instead of the robots rounding up the cockroaches like sheepdogs, they lead through social attraction."

But Coby Schal, an urban entomologist at North Carolina State University in Raleigh, has reservations about the effectiveness of the pheromone guise in convincing the roaches that the robot is just like them. He wonders if the physical presence of the robots made the lighter shelter more attractive simply by increasing the structural complexity of this hiding place. "In my view, the jury is still out" on whether the robots became part of the decision-making, says Schal.

Nonetheless, roboticist Daniela Rus of the Massachusetts Institute of Technology in Cambridge calls the idea that robots can influence biological group behavior "very powerful." She speculates that the work could have many applications, such as robots that aid pest control by luring insects into traps or that help herd livestock.

–ELIZABETH PENNISI

www.sciencemag.org SCIENCE VOL 318 16 NOVEMBER 2007 1055

Published by AAAS



Para leer más acerca de los robots influenciando el comportamiento de insectos, ver la revista "*Science*" de Noviembre 16, 2007. El artículo de base que se discute en la imagen de arriba es de Halloy *et al*, "*Social Integration of Robots into Groups of Cockroaches to Control Self-Organized Choices*" *(Integración social de robots en grupos de cucarachas para controlar elecciones ato-organizadas), Science*, Noviembre 16, 2007. Volumen 318, pp 1155-1158.

## Revisión Myro

No se introdujeron características nuevas de Myro en este capítulo.

## Revisión Python

La sentencia-if en Python tiene las siguientes formas:

```
if <condicion>:
  <esto>

if <condicion>:
  <esto>
else:
  <eso>

if <condicion-1>:
  <esto>
elif <condicion-2>:
  <eso>
elif <condicion-3>:
  <otra cosa>
...
```

```
...
else:
   <otho>
```

Las condiciones pueden ser cualquier expresión que resulte en un valor True, False, 1, o 0. Revisen el Capítulo 4 para más detalles en la escritura de expresiones condicionales.

## Ejercicios

**1.** Una forma aún mejor de promediar las condiciones de luz ambiente para su normalización es que el robot tome una muestra de la luz ambiente a su alrededor. Es decir, que dé un círculo completo y tome muestras de los diferentes valores del sensor de luz. El valor ambiente puede entonces establecerse como el promedio de todos los valores de luz. Escriban una función llamada `setAmbiente` que rote al robot en un círculo completo (o podrían usar el tiempo), y que tome muestras de los valores del sensor de luz a medida que rota, para finalmente devolver el promedio de todos los valores de luz. Modifiquen la línea:

```
Ambiente = sum(getLight())/3.0
```

con la línea:

```
Ambiente = setAmbiente()
```

Prueben todos los comportamientos anteriores descriptos en este capítulo para ver cómo este mecanismo nuevo afecta al comportamiento del robot.

**2.** Diseñen e implementen un programa que exhiba el comportamiento de salida del corral descrito en este capítulo.

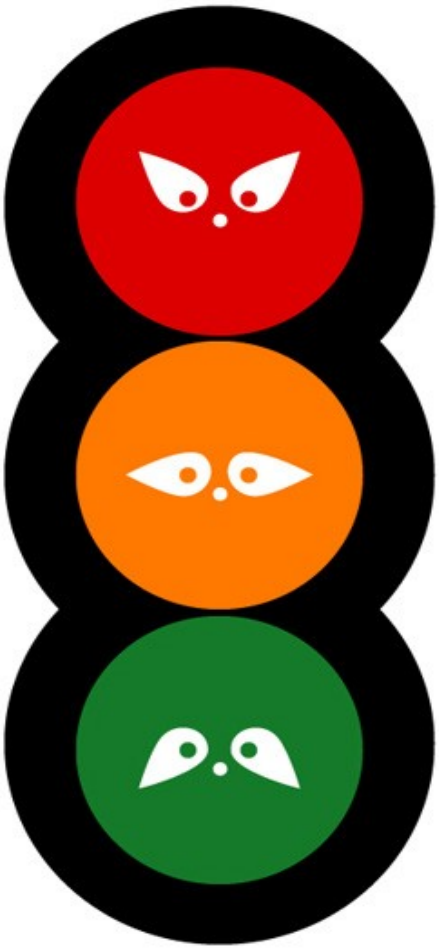
**3.** Implementen en comportamiento de detective de heladera descrito en este capítulo.

**4.** Implementen el robot de Alarma contra robo descrito en este capítulo.

**5.** Implementen el comportamiento de navegador de pasillo descrito en este capítulo.

**6.** Además de movimientos, intenten integrar música/sonido a los comportamientos del robot y observen cómo el agregado de sonido amplifica la percepción de la personalidad del robot.

# 7 Control de Comportamiento



*!, Comórtate!*
*Austin Powers (interpretado por Mike Myers) en la pelíula*
*Austin Powers: Hombre Internacional del Misterio,*
*New Line Cinema, 1997*

Página opuesta: Mis luces de tráfico
Ilustración de Yingshun Wong (yingshun.co.uk)

Escribir programas es ejercitar el control. En es caso del cerebro de un robot, el programa dirige las operaciones del robot. Sin embargo, también es importante observar que el programa en sí mismo está realmente controlando la computadora. Es decir, cuando escriben programas Python están controlando la computadora que entonces se está comunicando con el robot. Si quitan a Myro de la escena están escribiendo programas para controlar la computadora. En este sentido, el aprendizaje con robots también lleva a aprender computación. Cada programa que escriben está haciendo computación. Contrariamente a los preconceptos populares acerca de la computación, ésta no se trata solamente de realizar ecuaciones con números. Controlar el robot es también realizar computación, como también lo es predecir la población mundial, componer una imagen, etc. Este es un aspecto del control.

Al escribir programas para controlar los robots, la estructura que utilizan para organizar el programa es una estrategia de control. Programar un robot significa especificar controles automatizados. Como programadores o diseñadores de comportamiento, ustedes estructuran su programa para lograr las metas de ese comportamiento: cómo se utilizan los sensores para decidir qué hacer a continuación. Este es otro aspecto del control. Hasta ahora, han visto cómo escribir programas de control usando el estilo de cableado sensor-motor de Braitenberg. También han visto como especificar el control reactivo. Estos son ejemplos de dos paradigmas de control de robot.

En este capítulo ahondamos más en el mundo de la computación y los paradigmas de control de robots. Aprenderemos cómo escribir programas de control de robot para realizar tareas más complejas y más robustas. También veremos cómo, usando los conceptos aprendidos hasta ahora, podemos escribir aplicaciones de computadora útiles e interesantes.

## Control basado en el comportamiento

Al escribir programas de control de robots, hasta ahora, han usando una técnica muy básica para diseñar programas de control:

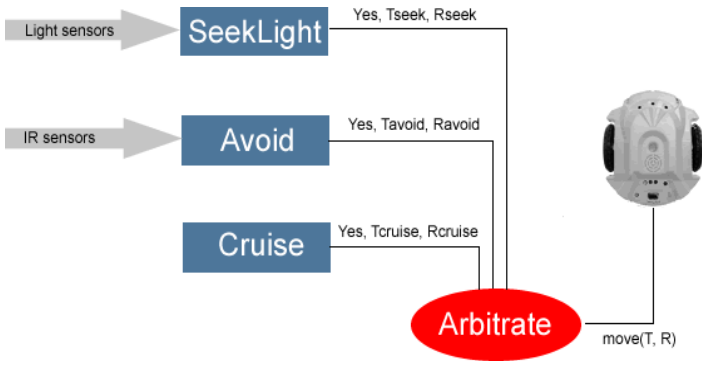
```
def main():
    # hacer por siempre o por algún tiempo
    # o hasta que se cumpla alguna condición

        # sensar y transformar los valores de los sensores
        # razonar y decidir qué hacer
        # hacerlo!
```

Como habrán observado, tales programas de control funcionan bien para tareas simples. Sin embargo, posiblemente ya se han encontrado con situaciones en las que, una vez que la tarea se pone un poco más compleja, resulta difícil estructurar un programa en términos de una línea de control como se muestra arriba. Por ejemplo, el comportamiento de salida de corral del capítulo anterior les requiere combinar dos comportamientos simples: resolver un laberinto (evitar obstáculos) y buscar la luz para salir del corral. Como han visto antes, es relativamente sencillo programar cada uno de los comportamientos individuales: evitar obstáculos; seguir la luz. Pero al combinar estos comportamientos para lograr la salida del corral, dos

cosas ocurren: se ven forzados a combinar las dos estrategias de control en una sola, y puede resultar difícil decidir de qué manera hacerlo. Además, el programa resultante no es muy bello y es difícil de leer. En realidad, casi ningún programa de robot se escribe de esa manera. En esta sección, veremos un modo distinto de estructurar los programas de robot que hace más fácil el diseño de comportamientos y además, la estructura que resulta del programa global es limpia y directa. Pueden diseñar comportamientos muy sofisticados usando estas ideas.

La gente de la comunidad robótica llama al estilo de programación mostrado arriba *control reactivo* o *control directo*. También se la llama *fusión de sensor*, y los programas resultantes son conducidos puramente por sensores y por lo tanto son demasiado "*bottom-up*", *es decir, desarrollados* de abajo hacia arriba. Es decir, los valores de los sensores llevan la lógica del control en dirección opuesta a las metas de las tareas del robot en sí mismas. En el *control basado en comportamiento*, uno se aleja de los sensores y focaliza el diseño del programa de robot sobre cierta cantidad y tipos de comportamientos que el robot puede llevar a cabo.

Veamos cómo el control basado en comportamiento nos permite diseñar el comportamiento de salida del corral. Básicamente, el robot tiene que llevar a cabo tres tipos de comportamientos: avanzar (en ausencia de cualquier obstáculo y/o luz), evitar obstáculos (si están presentes), y buscar la luz (si está presente). En el estilo de la escritura de programa basada en comportamiento, definiremos cada uno de estos comportamientos como una unidad de decisión individual. Por lo tanto, cada una se escribe de manera simple y directa. A continuación, el programa de control debe fusionar los comportamientos recomendados por cada unidad de comportamiento. Observen la imagen de abajo:




En el diagrama de arriba, hemos mostrado los tres comportamientos básicos que estamos intentando combinar: `Avanzar`, `Esquivar`, `BuscarLuz`. Cada uno de estos comportamientos da un triple: `Si`/`No`, una Velocidad de traslado,y una Velocidad de rotación. Un `Si` implica que el módulo de comportamiento tiene una recomendación. Un `No` implica que no la tiene. Es decir, permite la posibilidad de un comportamiento que no tenga recomendación. Por ejemplo, en la situación de salida del corral, en ausencia de una fuente de luz percibida por el robot, el módulo de `BuscarLuz` no tendrá una recomendación. Entonces deviene en tarea del árbitro (o el módulo de decisión) decidir cuál de las recomendaciones disponibles usar para manejar el robot. Observen que en definitiva, para controlar el robot, todo lo que uno debe hacer es decidir cuánto trasladar y rotar. Se pueden incorporar muchos esquemas de arbitraje diferentes. Usaremos uno simple pero efectivo: asignarle una prioridad

a cada módulo de comportamiento. Luego el árbitro elige la recomendación con la prioridad más alta. Este estilo de arquitectura de control también se denomina *arquitectura presuntiva.* En la figura de arriba, hemos dibujado los módulos según su orden de prioridad: cuanto más alto está el módulo en el esquema, mayor la prioridad. El comportamiento más bajo, `Avanzar`, no requiere de ningún sensor y siempre está presente: quiere que el robot siempre avance hacia delante.

Resolver el control basado en la combinación de comportamientos simples tiene varias ventajas: pueden diseñar cada comportamiento individual de manera muy sencilla; también pueden testear cada comportamiento individual agregando ese comportamiento y viendo cómo lo lleva a cabo, Pueden agregar cualquier cantidad de comportamientos uno sobre el otro. En el esquema de arriba el régimen de control implica que el robot siempre avanzará hacia delante. Pero si se presenta un obstáculo, pasará por alto el comportamiento *`Avanzar`* e intentará esquivar el obstáculo. Sin embargo, si se detecta una fuente de luz, sobrepasará a los otros comportamientos y se dedicará al comportamiento de buscador de luz. Lo ideal que podemos imaginar es que todos los comportamientos estén funcionando simultáneamente (de manera asincrónica). En esa situación, el árbitro siempre tendrá una o más recomendaciones para adoptar basándose en la prioridad.

Desarrollemos el programa que implementa el control basado en comportamiento. Primero, definimos cada comportamiento:

```
velocidadCrucero = 0.8
velocidadGiro = 0.8
umbralLuz = 80

def avanzar():
  # siempre se mueve hacia adelante
  return [True, velocidadCrucero, 0]

def esquivar():
  # busca si hay algún obstáculo
  L, R = getIR()
  L = 1 - L
  R = 1 - R

  if L:
    return [True, 0, -velocidadGiro]
  elif R:
    return [True, 0, velocidadGiro]
  else:
    return [False, 0, 0]



def buscarLuz():
  L, C, R = getLight()

  if L < umbralLuz:
    return [True, velocidadCrucero/2.0, velocidadGiro]
  elif R < umbralLuz:
    return [True, velocidadCrucero/2.0, -velocidadGiro]
  else:
    return [False, 0, 0]
```

Arriba pueden ver que cada comportamiento individual es simple y fácil de leer (y de escribir). Hay varias maneras de incorporarlos a un programa basado en comportamiento. Aquí hay una:

```
# lista de comportamientos ordenados por prioridad
# (el de la izquierda es el más alta prioridad)
comportamientos = [buscarLuz, esquivar, avanzar]

def main():

  while True:
    T, R = arbitrar()
    mover(T, R)

main()
```

El programa principal llama a la función de arbitraje que devuelve los comandos de traslación y de rotación que se aplican entonces al robot. La función `arbitrar` es lo suficientemente simple:

```
# Decidir qué comportamiento, en orden de prioridad
# recomienda para el robot
def arbitrar():

  for comportamiento in comportamientos:
    output, T, R = comportamiento()
    if output:
      return [T, R]
```

Es decir, requiere a cada comportamiento en orden de prioridades para ver si tiene una recomendación. Si la tiene, ese es el grupo de comandos de motor que se devuelve.

**Realizar la siguiente actividad:** Implementen en programa de arriba y vean cómo se comporta el robot navegando y saliendo del corral. ¿Qué ocurre si cambian la prioridad (el orden en la lista) de los comportamientos? Al escribir el programa de arriba, hemos usado dos características Python. Las veremos a continuación.

## Nombres y valores devueltos

En el capítulo 3 aprendieron que en Python los nombres pueden ser usados para representar funciones así como números y *strings*. También vieron en el Capítulo 5 que las listas, e incluso las fotos o imágenes pueden ser representados usando nombres. Un nombre es un concepto importante en programación. En Python, un nombre puede representar cualquier cosa como su valor: su número, una foto, una función, etc. En el programa de arriba, cuando definimos los comportamientos de las variables como:

```
comportamientos = [buscarLuz, esquivar, avanzar]
```

usamos los nombre `buscarLuz`, `esquivar`, y `avanzar` para denotar las funciones que representan. Nombramos a estas funciones anteriormente en el programa (usando **`def`**). De esta manera, la lista de comportamientos nombrados es una lista de nombres de funciones, cada uno de los cuales denotará la función actual y su valor. A continuación, observen el modo en que usamos la variable `comportamientos` en la función de arbitraje:

```
for comportamiento in comportamientos:
    output, T, R = comportamiento()
    ...
```

Dado que `comportamientos` es una lista, puede ser usada en el loop para representar la secuencia de objetos (en este caso funciones). Por lo tanto, en cada iteración del loop, la variable `comportamiento` toma sucesivos valores de esta lista: `buscarLuz`, `esquivar`, y `avanzar`. Cuando el valor de la variable es `buscarLuz`, la función `buscarLuz` es llamada en esta instrucción:

```
output, T, R = comportamiento()
```

No hay ninguna función llamada `comportamiento()` que esté definida en ninguna parte del programa. Sin embargo, como el valor del nombre de la variable `comportamiento` está establecida para una de las funciones en la lista `comportamientos`, la función correspondiente es invocada.

Otro aspecto nuevo de Python que hemos usado en el programa de arriba es el hecho de que una función puede devolver cualquier objeto como su valor. Por lo tanto, las funciones `avanzar`, `esquivar`, `buscarLuz`, y arbitrar todas devuelven listas como sus valores.

Ambas son características sutiles de Python. Muchos otros lenguajes de programación tienen restricciones sobre los tipos de cosas que podemos llamar como variables y también sobre los tipos de valores que puede devolver una función. Python le da un tratamiento uniforme a todos los objetos.

## La Librería Matemática de Python

La mayoría de los lenguajes de programación, incluido Python, proveen una saludable selección de librerías de funciones útiles para que no sea necesario escribirlas. Tales librerías suelen denominarse API ("Application Programming Interfaces", en inglés). Un poco más atrás se les presentó la librería `random` provista por Python. Contiene varias funciones útiles que proveen facilidades para generar números al azar. De modo similar, Python también provee una librería `math` que provee funciones matemáticas de uso frecuente. En el Capítulo 6 usamos la función `exp` de la librería `math` para normalizar los valores del sensor. Algunas de las funciones comúnmente usadas en la librería `math` se listan debajo:

### Funciones de uso frecuente en el módulo `math`:

**`ceil(x)`:** Devuelve el techo de x como un flotante, el valor entero mayor o igual a x.

**`floor(x)`:** Devuelve el piso de x como un flotante, el valor entero más grande menor o igual que x.

**`exp(x)`:** Returns $e^x$.

**`log(x[, base])`** Devuelve el logaritmo de x a la base dada. Si la base no se especifica, devuelve el logaritmo natural de x (por ej., $\log_e x$).

**`log10(x)`:** devuelve el logaritmo de base-10 de x (por ej. $\log_{10} x$).

**`pow(x, y)`:** Devuelve $x^y$.

**`sqrt(x)`:** Devuelve la raíz cuadrada de x ($\sqrt{x}$).

Algunas de las otras funciones disponibles en el módulo `math` se listan al final del capítulo. Para usar cualquiera de ellas, todo lo que deben hacer es importar el módulo `math`:

```
import math

>>> math.ceil(5.34)
6.0
>>> math.floor(5.34)
5.0

>>> math.exp(3)
20.085536923187668

>>> math.log10(1000)
3.0
>>> math.log(1024, 2)
10.0
>>> math.pow(2, 10)
1024.0

>>> math.sqrt(7.0)
2.6457513110645907
```

De manera alternativa, pueden importar el módulo `math` como  se muestra abajo:

```
>>> from math import *
>>> ceil(5.34)
6.0
>>> floor(5.34)
5.0
>>> exp(3)
20.085536923187668
>>> log10(1000)
3.0
>>> log(1024,2)
10.0
>>> sqrt(7.0)
2.6457513110645907
```

En Python, cuando importa un módulo usando el comando:

```
import <module>
```

Tienen que predeterminar todos sus comandos con el nombre del módulo (como en el caso del primer grupo de ejemplos de arriba). También hemos estado usando la forma

```
from <module> import *
```

Esto importa todas las funciones y otros objetos provistos en el módulo que pueden ser útiles sin la predeterminación. También pueden importar individualmente funciones específicas de un módulo usando:

```
from <modulo> import <esto>, <eso>, ...
```

La versión que utilicen dependerá del contexto en el cual usen las funciones importadas. Pueden encontrarse con una situación en la que dos módulos distintos

definen funciones con el mismo nombre pero hacen cosas diferentes (o para hacer cosas diferentes). Para usar ambas funciones en el mismo módulo deberán usar el módulo prefijado para que queda claro qué versión estarán utilizando.

## Realizando cómputos

Volvamos al estilo tradicional de computación por ahora. Verán que los conceptos que han aprendido hasta ahora les permitirán escribir muy distintas e interesantes aplicaciones de computación. También les dará un sentido claro de la estructura de los típicos programas de computación. Más adelante, en el Capítulo 11, volveremos también al tema más amplio de diseño general de los programas de computación.

### Una Calculadora de Préstamo

Su auto actual, un adorable 1992 SAAB 93 fue comprado usado y durante los últimos meses no han tenido más que problemas para mantener el auto en la ruta. La noche pasada la llave de arranque se rompió dentro de la ranura de la llave al intentar iniciarlo y ahora la parte rota no sale (esto sucedía mucho en los viejos SAABs). El mecánico tuvo que desmantelar todo el engranaje de arranque para sacar el pedazo roto y podría salir hasta $500. El motor del auto, que ya tiene más de 290.000 kilómetros, los ha dejado varados en la ruta muchas veces. Han decidido que es suficiente, irán en busca de un reluciente auto nuevo y confiable. Han estado trabajando de noche en un restaurante para hacer algo de dinero extra y han ahorrado $5500.00 justamente para una situación como esta. Ahora se están preguntando qué tipo de auto podrán comprar. Obviamente, deberán pedir un préstamo del banco para financiar el resto del costo, pero no están seguros acerca de cuán grande deberá ser el préstamo, y por lo tanto qué tipo de auto pueden costear. Pueden escribir un pequeño programa Python para ayudarles a probar varios escenarios.

Pueden soñar (realistamente) con el auto que quisieran comprar, o pueden ir a cualquier sitio de venta de autos en la Web (www.edmunds.com o www.demotores.com son dos buenos lugares donde comenzar). Digamos que han invertido horas mirando características y opciones y finalmente han definido su deseo en un par de posibilidades. La primera opción costará $22,000.00 y la segunda está valuada en $18,995.00. Ahora tienen que decidir cuál de estos pueden realmente comprar.

Primero, hablan con un par de bancos y miran algunas ofertas de préstamos en la Web. Por ejemplo, vayan a bankrate.com y vean las tasas actuales de préstamos para autos nuevos.

Supongan que las tasas de los préstamos para esto son de: 6.9% para 36 meses, 7.25% para 48 meses, y 7.15% para 60 meses.

Podrán ver que hay una importante variación en las tasas. Dada esta información, ahora están listos para escribir un programa que los ayude a averiguar por cuál de las dos opciones podrán optar. A fin de asegurarse el préstamo, deben garantizar que tienen suficiente dinero como para pagar los impuestos locales de venta (¡un 6% de impuestos de venta sobre un auto de $20,000 llegarán a la considerable cifra de $1200!). Luego de pagar los impuestos de venta podrán usar el resto del dinero que han ahorrado para el pago inicial. El dinero restante es la cantidad que pedirían prestada. Dependiendo del tipo de préstamo que elijan, variarán los pagos

mensuales y la cantidad de tiempo de los mismos. Hay una fórmula simple que pueden usar para estimar el pago mensual:

$$MonthlyPayment = \frac{LoanAmount * MonthlyInterestRate}{1 - e^{-LoanTerm * \log(1 + MonthlyInterestRate)}}$$

¡Whoa! Eso se ve complicado. Sin embargo, dada la fórmula, pueden ver que en realidad requiere de dos funciones matemáticas: $\log(x)$ y $e^x$, ambas disponibles en el módulo `math` de Python. De pronto, el problema no parece tan difícil.

Intentemos delinear los pasos necesarios para escribir el programa: primero, observen el costo del auto, la cantidad de dinero que han ahorrado, y la tasa de impuesto de venta. También registren los siguientes datos financieros: la tasa de interés y la duración del préstamo. La tasa de interés dada es en general el porcentaje anual. Conviértanlo en la tasa mensual (dividiéndolo por 12). Luego, computen el impuesto de venta que pagarán. Usen en dinero restante para realizar el pago inicial. Luego determinen la cantidad de dinero que tomarán como préstamo. Ingresen todos los valores en la fórmula y computen el pago mensual. Además, computen el costo total del auto. Obtengan todos los resultados. A continuación, podemos tomar cada uno de los pasos de arriba y codificarlos en un programa. Aquí tienen una primer ordenamiento:

```
def main():
  # Primero observen del costo del auto (Cost),
  # La cantidad de dinero que han ahorrado (Cash),
  # y la tasa de impuestos por venta (TaxRate)

  # También observen los datos financieros: la tasa de interés(APR),
  # y luego la duración del préstamo (Term)
  # la tasa de interés en general se fija en forma anual(APR)
  # Conviértanlo en una tasa mensual (dividiéndola por 12) (MR)

  # A continuación, computen la tasa de venta que pagarán (SalesTax)
  # Usen el dinero restante para efectuar el pago inicial (DownPayment)
  # Luego determinen la cantidad que pedirán prestada (LoanAmount)

  # Ingresen todos los valores en la fórmula y computen
  # el pago mensual (MP)

  # Calculen el costo total del auto (TotalCost)

  # Muestren los resultados

main()
```

Arriba, hemos tomado los pasos y los hemos convertido en un esqueleto de programa Python. Todos los pasos se han convertido en comentarios Python, y donde se necesitan, hemos decidido los nombres de las variables que sostendrán los valores necesarios para los cálculos. Esto es útil porque también ayuda a determinar cómo se codificará la fórmula, además de ayudar a determinar qué valores pueden ser programados al interior y cuáles deberán ser suministrados como entrada. Hacer que el programa requiera entradas les permitirá ingresar fácilmente los diferentes parámetros y luego, basándose en los salidas que reciben, pueden decidir qué auto comprar. Codifiquemos todos las entradas primero:

```
def main():

    # Primero observen del costo del auto (Cost),
    Cost = input("Ingrese el costo del auto: $")

    # La cantidad de dinero que han ahorrado (Cash),
    Cash = input("Ingrese la cantidad de dinero que ahorró: $")

    # y la tasa de impuestos por venta (TaxRate)
    SalesTaxRate = 6.0

   # También observen los datos financieros: la tasa de interés(APR),
   # y luego la duración del préstamo (Term)
   # la tasa de interés en general se fija en forma anual(APR)
    APR = input("Ingrese la tasa de Interes anual(in %): ")

    # Conviértanlo en una tasa mensual (dividiéndola por 12) (MR)

   # A continuación, computen la tasa de venta que pagarán (SalesTax)
   # Usen el dinero restante para efectuar el pago inicial (DownPayment)
   # Luego determinen la cantidad que pedirán prestada (LoanAmount)

   # Ingresen todos los valores en la fórmula y computen
   # el pago mensual (MP)

   # Calculen el costo total del auto (TotalCost)

   # Muestren los resultados

main()
```

Hemos refinado el programa incluyendo todos las entradas que se necesitarán cada vez que se ejecute el programa. Observen que elegimos no incorporar la tasa de impuestos de venta, sino que se la asignamos a la variable `SalesTaxRate`. Si ustedes hubiesen querido, podrían haberlo ingresado como otra entrada. Lo que ustedes elijan tener como entrada de su programa es su decisión de diseño. A veces el problema puede ser encuadrado de manera que explícitamente especifica las entradas, a veces deben averiguarlo. En general, donde deben hacer al programa más versátil es allí donde deben basarse las decisiones. Por ejemplo, determinar la tasa de impuestos de venta en 6.0 hará que el programa sea usable solamente en aquellas instancias en las que aplique esa tasa. Si, por ejemplo, si quisieran que un amigo en otra parte del país utilizara el programa, deberían elegir que ese también sea un valor de entrada. Avancemos hacia el siguiente paso del programa y codifiquémoslos en Python. Se trata básicamente de los cómputos. Lo primeros son sencillos. Quizás el cómputo más complicado de codificar sea la fórmula para calcular el pago mensual. Todos se muestran en la versión de abajo.

```
from math import *

def main():
    # Primero observen del costo del auto (Cost),
    Cost = input("Ingrese el costo del auto: $")

    # La cantidad de dinero que han ahorrado (Cash),
    Cash = input("Ingrese la cantidad de dinero que ahorró: $")
```

```
  # y la tasa de impuestos por venta (TaxRate)
  SalesTaxRate = 6.0

 # También observen los datos financieros: la tasa de interés(APR),
 # y luego la duración del préstamo (Term)
 # la tasa de interés en general se fija en forma anual(APR)
  APR = input("Ingrese la tasa de Interes anual(in %): ")

  # Ingrese la duración del préstamo (Term)
  term = input("Ingrese la duración del préstamo (in months): ")

  # Convertir el APR a una tasa mensual (dividir por 12) (MR)
  # también dividirlo por 100 ya que el valor de entrada es en %
  MR = APR/12.0/100.0

  # A continuación, computen la tasa de venta que pagarán (SalesTax)
  SalesTax = Cost * SalesTaxRate / 100.0

  # Usen el dinero restante para efectuar el pago inicial (DownPayment)
  DownPayment = Cash - SalesTax

  # Luego determinen la cantidad que pedirán prestada (LoanAmount)
  LoanAmount = Cost - DownPayment

  # Ingresen todos los valores en la fórmula y computen
  # el pago mensual (MP)
  MR = (LoanAmount * MR) / (1.0 - exp(-term * log(1.0+MR)))

  # Calculen el costo total del auto (TotalCost)
   TotalCost = SalesTax + DownPayment + MR * term

  # Muestren los resultados
  print "Aquí los detalles sobre tu nuevo auto ..."
  print "----------------------------------------"
  print
  print "Dinero ahorrado $", Cash
  print "Costo del auto $", Cost
  print "Tasa de impuestos por venta", SalesTaxRate, "%"
  print "Impuestos sobre la venta del autor $", SalesTax
  print "Pago Inicial $", DownPayment
  print "Dinero a pedir restado $", LoanAmount
  print "en", term, "meses a una tasa de", APR, "% APR"
  print "Tu pago mensual será de $", MP
  print "El costo total será $", TotalCost
  print

main()
```

**Realizar la siguiente actividad:** Al ingresar el programa de arriba y hacerlo correr en Python, pueden ingresar los datos de su auto. Aquí hay un ejemplo del programa corriendo:

```
Ingrese el costo del auto:$20000.00
Ingrese la cantidad de dinero que ahorró: $ 5500.00
Ingrese la tasa de Interes anual(in %): 6.9
Ingrese la duración del préstamo (in months):36
```

```
Aquí los detalles sobre tu nuevo auto ...
----------------------------------------
Dinero ahorrado $$ 5500.0
Costo del auto $ 20000.0
Tasa de impuestos por venta  $ 1200.0
Impuestos sobre la venta del autor $ 4300.0
Pago Inicial $ 15700.0
Dinero a pedir restado  en  36 meses a una tasa de 6.9 % APR
Tu pago mensual será de $ 484.052914723
El costo total será $ 22925.90493
```

Parece que para el auto de $20000.00, por un préstamo de 36 meses a una tasa del 6.9%, deberán pagar $484.05 (o $484.06, ¡dependiendo de cómo su compañía de préstamos para autos redondea los centavos!).

Cuando deben restringir los valores de la salida a espacios decimales específicos (dos espacios en el caso de los pesos  y los centavos, por ejemplo), pueden usar el string de formato de características construido en Python. Por ejemplo, en un string, pueden especificar cómo incluir un valor de punto flotante como se muestra a continuación:

```
"Valor de PI %5.3f con 3 lugares decimales" % (math.pi)
```

La de arriba es una expresión Python que tiene la siguiente sintaxis:

```
<string> % <expresion>
```

Dentro del `<string>` hay una especificación que comienza con un signo `%` y termina con `f`. Lo que sigue al signo `%` es una especificación numérica del valor para ser insertado en ese punto en el string. La `f` en la especificación hace referencia a hecho de que es para un valor de punto flotante. Entre el signo `%` y la `f` se encuentra el número 5.3. Esto especifica que el número de punto flotante a insertarse ocupará por lo menos 5 espacios, 3 de los cuales irán después del decimal. Uno de los espacios siempre está ocupado por el decimal. Por lo tanto, la especificación `%5.3f` especifica un valor a ser insertado de la siguiente manera:

```
"Este es el valor de  PI -.--- expresado con tres lugares decimales"
```

Abajo pueden ver el resultado de ejecutar esto en Python:

```
>>> "Valor de PI %5.3f con 3 lugares decimales" % (math.pi)
Valor de  PI 3.142 con 3 lugares decimales'

>>> "Valor de PI %7.4f con 4 lugares decimales" % (math.pi)
'Valor de PI  3.1416 con 4 lugares decimales'
```

En el segundo ejemplo de arriba hemos reemplazado la especificación con `%7.4f`. Noten que el string resultante define 7 espacios para imprimir ese valor. Si hay más espacio que los necesarios, se rellenan con blancos en el borde inicial (noten el espacio extra antes del 3 en el ejemplo de arriba). Si el espacio especificado es menor, siempre se expandirá para acomodar el número. Por ejemplo:

```
>>> " Valor de PI %1.4f con 4 lugares decimales" % (math.pi)
Valor de PI 3.1416 con 4 lugares decimales'
```

Deliberadamente especificamos que el valor sea de 1 ancho de espacio con 4 espacios después del decimal (por ej. `%1.4f`). Como pueden ver, el espacio se expandió para acomodar el valor. Lo que se asegura es que el valor siempre se imprime usando el exacto número de espacios después del decimal. Aquí hay otro ejemplo:

```
>>> "5 es también  %1.3f con  3 lugares decimales." % 5
'5 es también 5.000 con  3 lugares decimales .'
```

De esta manera, el valor impreso como 5.000 (por ejemplo, los tres espacios después del decimal siempre se consideran relevantes en la especificación, como `%1.3f`). De modo similar para especificar un número entero o valores *enteros*, pueden usar la letra `d`, y para strings pueden usar la letra `s`:

```
>>> "Hola %10s, Cómo estás?" % "Juan"
'Hola       Juan, Cómo estás?'
```

De manera predeterminada, las especificaciones más extensas se justifican hacia la derecha. Pueden usar una especificación `%` para justificar hacia la izquierda. Por ejemplo:

```
>>> "Hola %-10s, Cómo estás?" % "Juan"
'Hola Juan      , Cómo estás?'
```

Tener este tipo de control sobre los valores impresos es importante cuando están intentando sacar tablas de valores alineados. Modifiquemos nuestro programa de arriba para usar estas características de formato:

```
from math import *

def main():
    # Primero observen del costo del auto (Cost),
    Cost = input("Ingrese el costo del auto: $")

    # La cantidad de dinero que han ahorrado (Cash),
    Cash = input("Ingrese la cantidad de dinero que ahorró: $")

    # y la tasa de impuestos por venta (TaxRate)
    SalesTaxRate = 6.0

   # También observen los datos financieros: la tasa de interés(APR),
   # y luego la duración del préstamo (Term)
   # la tasa de interés en general se fija en forma anual(APR)
    APR = input("Ingrese la tasa de Interes anual(in %): ")

    # Ingrese la duración del préstamo (Term)
    term = input("Ingrese la duración del préstamo (in months): ")

    # Convertir el APR a una tasa mensual (dividir por 12) (MR)
    # también dividirlo por 100 ya que el valor de entrada es en %
    MR = APR/12.0/100.0

    # A continuación, computen la tasa de venta que pagarán (SalesTax)
    SalesTax = Cost * SalesTaxRate / 100.0

    # Usen el dinero restante para efectuar el pago inicial (DownPayment)
    DownPayment = Cash - SalesTax

    # Luego determinen la cantidad que pedirán prestada (LoanAmount)
```

```
    LoanAmount = Cost - DownPayment

    # Ingresen todos los valores en la fórmula y computen
    # el pago mensual (MP)
    MR = (LoanAmount * MR) / (1.0 - exp(-term * log(1.0+MR)))

    # Calculen el costo total del auto (TotalCost)
     TotalCost = SalesTax + DownPayment + MR * term

   # Muestren los resultados
    print "Aquí los detalles sobre tu nuevo auto ..."
    print "----------------------------------------"
    print
    print "Dinero ahorrado $%1.2f" %  Cash
    print "Costo del auto $%1.2f" % Cost
    print "Tasa de impuestos por venta%1.2f" % SalesTaxRate
    print "Impuestos sobre la venta del autor $", SalesTax, "%"
    print "Pago Inicial $%1.2f" % DownPayment
    print "Dinero a pedir restado $%1.2f" % LoanAmount
    print "en %2d meses a una tasa de  %1.2f APR" % (term, APR)
    print "Tu pago mensual será de $%1.2f" % MP
    print "El costo total será $%1.2f" % TotalCost
    print

main()
```

Al ejecutarlo  de nuevo (digamos para términos de préstamos levemente diferentes), obtienen:

```
Ingrese el costo del auto:$20000.00
Ingrese la cantidad de dinero que ahorró: $ 5500.00
Ingrese la tasa de Interes anual(in %): 7.25
Ingrese la duración del préstamo (in months): 48

Aquí los detalles sobre tu nuevo auto ...
----------------------------------------
Dinero ahorrado $ 5500.00
Costo del auto $ 20000.00
Tasa de impuestos por venta  $ 1200.0 %
Impuestos sobre la venta del autor $ 4300.00
Pago Inicial $ 15700.00
Dinero a pedir restado  en  48 meses a una tasa de 7.25 APR
Tu pago mensual será de $377.78
El costo total será $23633.43
```

Pueden ver que para el mismo monto, si toman el préstamo por un período más largo pueden reducir sus pagos mensuales en más de $100 (aunque pagan aproximadamente $700 más al final).

## Toma de Decisiones en programas de Computación

La toma de decisiones es central para todos los programas de computación. De hecho, hay un famoso teorema en ciencias de la computación que dice que si cualquier dispositivo programable es capaz de ser programado para realizar *ejecuciones secuenciales*, *toma de decisiones* y *repetición*, será capaz de expresar cualquier algoritmo computable. Esta es una idea muy poderosa presentada en términos de ideas muy simples. Dado un problema, si pueden describir su solución

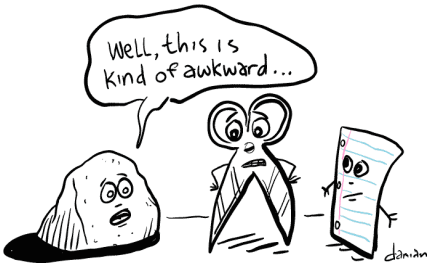
en términos de la combinación de tres modelos de ejecución, entonces pueden lograr que cualquier computadora resuelva ese problema. En esta sección, veremos la toma de decisiones en una clásica situación de juegos.

Las computadoras han sido extensamente usadas para juegos: para jugar contra otras personas y contra otras computadoras. El impacto de las computadoras en los juegos es tan tremendo que actualmente varios productores construyen consolas de juegos con computadoras adentro que se dedican exclusivamente a juegos.

Diseñemos un juego simple que seguramente han jugado cuando eran chicos: ¡Piedra papel o Tijera!

En este juego, dos jugadores juegan como contrincantes. A la cuenta de tres cada jugador realiza un gesto con la mano que indica que han seleccionado uno de los tres ítems: papel (la mano se extiende plana), tijera (se extienden dos dedos para indicar tijeras), o piedra (indicada con un puño). Las reglas para decidir quién es el ganador son simples: si ambos jugadores eligen el mismo objeto es un empate; de lo contrario, el papel le gana a la piedra, las tijeras le ganan al papel y la piedra le gana a las tijeras. Escribamos un programa de computación para jugar este juego contra la computadora. Aquí hay un delineado para jugar el juego una vez.

```
# La computadora y el jugador realizan una selección
# Decidir el resultado de las selecciones (empate
o quién gana)
# Informar al jugador de la decisión
```




*El nombre exacto del juego puede variar, con los tres componentes apareciendo en diferente orden, o con "roca" en lugar de "piedra". Los que hablan idiomas que no son el español pueden conocer al juego por los términos locales para "piedra, papel, tijera", aunque también se conoce como **Jankenpon** en Japón, **Rochambeau** en Francia, y el Sudáfrica como **Ching-Chong-Cha**...*
*Fuente:talkingtotan.wordpress.com/2007/08/*

Si presentamos los tres ítems en una lista, podemos hacer que la computadora elija uno de ellos al azar usando las facilidades de generación de números al azar provistas por Python. Si estos ítems se presentan como:

```
items = ["Papel", "Tijera", "Piedra"]
```

Entonces podemos seleccionar cualquiera de los ítems de arriba como elección de la computadora usando la expresión:

```
# La computadora elige
miOpcion = items[<0 o 1 o 2 seleccionado
aleatoriamente>]
```

Es decir `items[0]` representa la elección `"Papel"`, `items[1]` representa `"Tijera"`, e `items[2]` es `"Piedra"`. Hemos visto en el Capítulo 4 que podemos generar un número al azar dentro de cualquier rango usando la función `randint` del módulo de librería `random` en Python. Por lo tanto podemos modelar la computadora haciendo para hacer una selección al azar usando la siguiente sentencia:

```
from random import *

#  La computadora elige
miOpcion = items[randint(0,2)]
```

Recuerden que `randint(n, m)` devuelve un número al azar dentro del rango `[n..m]`. Entonces, `randrange(0, 2)` devolverá o 0, 1, o 2. Podemos usar el comando `askQuestion` en Myro para pedirle al jugador que indique su selección (ver Capítulo 3).

```
# El Jugador elige
tuOpcion = askQuestion("Elige un item.", items)
```

Ahora que sabemos cómo la computadora y el jugador toman su decisión, necesitamos pensar sobre la decisión del resultado. Aquí hay un esquema:

```
Si ambos eligieron el mismo ítem entonces es un empate
Si la computadora gana entonces se le informa al jugador
Si el jugador gana entonces se le informa al jugador
```

Si reescribimos lo de arriba usando sentencias `if` obtenemos el siguiente borrador:

```
if miOpcion == tuOpcion:
   print "Empatamos!"
if < miOpcion gana a  tuOpcion>:
   print "Gané!"
else:
   print "Ganaste!"
```

Todo lo que debemos hacer es averiguar cómo escribir la condición `<miOpcion gana a  tuOpcion>`. La condición debe capturar las reglas del juego mencionado arriba. Podemos codificar todas las reglas en una expresión condicional de la siguiente manera:

```
  if ( miOpcion == "Papel" and tuOpcion == "Piedra")
      or ( miOpcion == "Tijera" and tuOpcion == "Papel")
      or ( miOpcion == "Piedra" and tuOpcion == "Tijera"):
     print "Gané!"
  else:
     print "Ganaste!"
```

La expresión condicional de arriba captura todas las posibilidades que deben ser examinadas a fin de tomar la decisión. Otra forma de escribir la decisión de arriba sería usando lo siguiente:

```
  if miOpcion == "Papel" and tuOpcion == "Piedra":
     print "Gané!"
  elif miOpcion == "Tijera" and tuOpcion == "Papel":
     print "Gané!"
  elif miOpcion == "Piedra" and tuOpcion == "Tijera":
    print "Gané!"
  else:
     print "Ganaste!"
```

Es decir, cada condición se examina por vez hasta que se encuentra una que confirma que la computadora gana. Si ninguna de esas condiciones es verdadera, la parte `else` de la sentencia `if` se alcanzará para indicar que el jugador ha ganado.

Otra alternativa para escribir la decisión de arriba es codificar la decisión en una función. Asumamos que ha una función `gana` que devuelve `True` o `False`, dependiendo de las elecciones. Podríamos escribir lo de arriba de la siguiente manera:

```
if miOpcion == tuOpcion:
   print "Empatamos!"
```

```
if gana(miOpcion, tuOpcion):
   print "Gané!"
else:
   print "Ganaste!"
```

Miremos más de cerca cómo podemos definir la función `gana`. Necesita devolver `True` si `miOpcion` le gana a tuOpcion. Así que todo lo que debemos hacer es codificar las reglas del juego descrito arriba. Aquí hay una versión borrador de la función:

```
def gana(yo, vos):
  # Te gano? Si es así, retornamos True, sino, retornamos False

  if yo == "Papel" and vos == "Piedra":
    # Papel r le gana a la Piedra
    return True
  elif yo == "Tijera" and vos == "Papel":
    # Tijera le gana a Papel
    return True
  elif yo == "Piedra" and vos == "Tijera":
    # Piedra le gana a Tijera
    return True
  else:
    return False
```

Una vez más, hemos usado la sentencia `if` en Python para codificar las reglas del juego. Ahora que hemos resuelto todas las partes críticas del programa, podemos unirlas como se muestra abajo:

```
#Un programa que  juega al juego Piedra, Papel y Tijeras!
from myro import *
from random import randrange

def gana(yo, vos):
  # Te gano? Si es así, retornamos True, sino, retornamos False

  if yo == "Papel" and vos == "Piedra":
    # Papel r le gana a la Piedra
    return True
  elif yo == "Tijera" and vos == "Papel":
    # Tijera le gana a Papel
    return True
  elif yo == "Piedra" and vos == "Tijera":
    # Piedra le gana a Tijera
    return True
  else:
    return False

def main():
   #  Juega una ronda del juego Piedra, Papel y Tijeras!
   print "Dime: Piedra, Papel o Tijeras!"
   print "Haz tu elección en la ventana que aparecerá"

   items = ["Papel", "Tijera", "Piedra"]

   # la computadora y el Jugador hacen sus selecciones ...
   # la computadora elige
   miOpcion = items[randint(0,2)]
```

```
  # El Jugador elige
  tuOpcion = askQuestion("Elige un item.", items)

   # informamos las opciones de los jugadores
   print
   print "Yo elegí: ", miOpcion
   print "Vos elegiste: ", tuOpcion

   # Decidimos si hay un empate o alguno gana
   if miOpcion == tuOpcion:
      print "Elegimos la misma opción!"
      print "Es un empate."
   elif gana(miOpcion, tuOpcion):
      print "Ya que", miOpcion, "le gana a", tuOpcion, "..."
      print "Gane!"
   else:
      print"Ya que", tuOpcion, "le gana a", miOpcion, "..."
      print "Ganaste!"

   print "Gracias por Jugar. Adiós!"

main()
```

Se agregaron unos comando print más para que la interacción sea más natural.

**Realizar la siguiente actividad:** Implementen el programa de Piedra, Papel o Tijera de arriba y ejecútenlo varias veces para asegurarse de que lo comprenden completamente. Modifiquen el programa de arriba para jugar varias vueltas. Además, incorporen un sistema de puntaje que mantiene un registro de la cantidad de veces que ganó cada jugador y de la cantidad de empates.

## Resumen

En este capítulo han visto una variedad de paradigmas de control: control de robot basado en el comportamiento, escribir un programa computacional simple pero útil, y escribir un juego de computadora simple. El control basado en el comportamiento es una forma poderosa y efectiva de describir sofisticados programas para controlar los robots. Este es el paradigma usado en muchas aplicaciones comerciales de robots. Deberían probar algunos de los ejercicios propuestos debajo para sentirse cómodos con esta técnica de control. Los otros dos programas ilustran cómo, usando los conceptos que han aprendido, pueden diseñar otras aplicaciones de computadora útiles. Los otros dos programas ilustran cómo, usando los conceptos que han aprendido, pueden diseñar otras aplicaciones útiles de computación. En varios Capítulos siguientes, exploraremos varias dimensiones de la computación: computación de medios, escribir juegos, etc.

## Revisión Myro

No hubo nada nuevo acerca de Myro en este Capítulo.

## Revisión Python

El módulo de librería de matemática provee varias funciones matemáticas útiles. Algunas de las más usadas se listan abajo:

**`ceil(x)`:** Devuelve el techo de x como un flotante, el valor entero más bajo mayor o igual que x. **`floor(x)`** Returns the floor of x as a float, the largest integer value less than or equal to x.

**`exp(x)`:** Devuelve $e^x$ .

**`log(x[, base])`:** Devuelve el logaritmo de x a la base dada. Si la base no está especificada, devuelve el logaritmo natural de x (por ej., $\log_e x$).

**`log10(x)`:** Devuelve el logaritmo de base -10 de x (por ej. $\log_{10} x$) .

**`pow(x, y)`:** Devuelve $x^y$ .

**`sqrt(x)`:** Devuelve la raíz cuadrada de x ($\sqrt{x}$).

Funciones trigonométricas

**`acos(x)`:** Devuelve el arco coseno de x, en radianes.

**`asin(x)`:** Devuelve el arco sine de x, en radianes.

**`atan(x)`:** Devuelve el arco tangente de x, en radianes.

**`cos(x)`:** Devuelve el coseno de x radianes.

**`sin(x)`:** Devuelve el sine de x radianes.

**`tan(x)`:** Devuelve la tangente de x radianes.

**`degrees(x)`:** Convierte el ángulo x de radianes a grados.

**`radians(x)`:** Convierte el ángulo x de radios a radianes.

El módulo también define dos constantes matemáticas:

**`pi`:** `La constante matemática` π.

**`e`:** La constante matemática *e*.

## Ejercicios

**1.** Reemplazar el módulo esquivar en el programa de control basado en comportamiento por un módulo llamado: seguir. El módulo seguir intenta detectar una pared a la derecha del robot (o izquierda) y luego intenta seguirla todo el tiempo. Observen el comportamiento resultante.

**2.** Realizar el crecimiento de población modelándolo como una ecuación diferencial.

**3.** Realizar el crecimiento de población modelándolo como una función continua de crecimiento.

**4.** En lugar de usar el enunciado:

tuOpcion = items[randint(0,2)]

usar el comando:

tuOpcion = choice(items)

**`choice(<list>)`:** es otra función definida en el módulo random. Al azar busca un elemento de la lista provista.

**5.** ¿Pueden mejorar el programa?

**6.** Implementen el juego HiLo: la computadora elige un número al azar entre 0 y 1000 y el usuario debe tratar de adivinarlo. Con cada intento, la computadora le informa al usuario si su número estuvo cerca (en inglés se utiliza el término Hi ("alto"), o lejos (en inglés se utiliza el término Low ("bajo"), o si acertaron.

**7.** Inviertan los roles en el juego HiLo de arriba. Esta vez, el usuario adivina un número entre 0 y 1000 y el programa de computadora debe adivinarlo. Piensen una buena estrategia para adivinar el número basándose en claves de alto o bajo, y luego implementen esa estrategia.

# 8 Vistas & Sonidos

*No hagan música para una audiencia vasta y desconocida o para el mercado o el rating, ni siquiera para algo tan tangible como el dinero. Aunque sea crucial ganarse la vida, esa no debería ser la inspiración. Háganlo por ustedes mismos.*
-Billy Joel

Página anterior: Be My Valentine
Arte Fractal por Sven Geier (www.sgeier.net)

Mencionamos anteriormente que la noción de computación en estos días se extiende mucho más allá de los simples cálculos numéricos. Escribir programas de control de robots es computación, al igual que hacer proyecciones acerca de la población mundial. Utilizando dispositivos como los iPods, pueden disfrutar música, videos y shows de radio y de televisión. La manipulación de sonido e imagen también pertenece al reino de la computación y en este capítulo les introduciremos el tema. Ya han visto cómo, usando el robot, pueden tomar fotos de varias escenas. También pueden tomar fotos similares de su cámara digital. Usando las técnicas computacionales básicas que han aprendido hasta ahora, verán en este capítulo cómo pueden realizar computación sobre formas y sonidos. Aprenderán a crear imágenes usando computación. Las imágenes que crearán pueden ser utilizadas para visualizar datos e incluso para fines estéticos para explorar sus facetas creativas. También presentaremos algunos fundamentos del sonido y de la música y les mostraremos cómo también pueden llevar a cabo formas similares de computación utilizando la música.

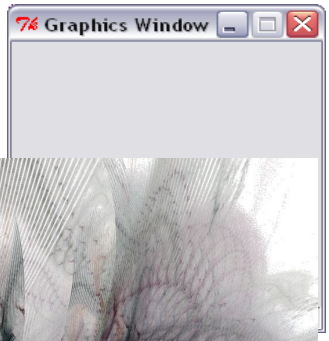
## Vistas: Dibujar

Si han usado una computadora durante cualquier cantidad de tiempo, deben haber usado algún tipo de aplicación para dibujar. Usando una típica aplicación de dibujo, pueden dibujar varias formas, colorearlas, etc. También pueden crear dibujos usando los comandos de dibujo provistos en el módulo de la librería Myro. Para

dibujar cualquier cosa, primero deben tener un lugar donde dibujarlo: un bastidor o una ventana. Pueden crear esa ventana usando el comando:

```
myCanvas = GraphWin()
```

**Una ventana de gráficos**



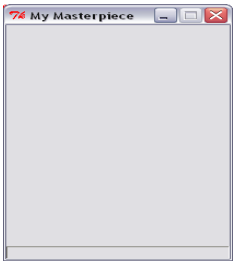

Si han ingresado el comando de arriba en el IDLE, inmediatamente verán emerger una pequeña ventana gris (ver imagen abajo a la derecha). Esta ventana les servirá como el bastidor para crear dibujos. Por defecto, la ventana creada por el comando `GraphWin` es de 200 píxeles de alto y 200 píxeles de ancho y su nombre es



`"Graphics Window"`. No es una forma muy inspiradora para comenzar, pero el comando `GraphWin` tiene algunas otras variaciones. Primero, para cerrar la pantalla, pueden usar el comando:

```
myCanvas.close()
```

**Mi obra maestra 200x300**




Para crear una ventana de gráficos de cualquier tamaño y con un nombre especificado por ustedes, pueden usar el comando de abajo:

```
myCanvas = GraphWin("Mi obra maestra", 200, 300)
```

El comando de arriba crea una ventana llamada `"Mi obra maestra"` que tendrá 200 píxeles de ancho y 300 de altura (ver imagen a la derecha). Pueden cambiar el color de fondo de un gráfico como se muestra abajo:

```
myCanvas.setBackground("white")
```

Pueden nombrar cualquiera de los colores en el comando de arriba, desde los colores comunes como “rojo”, “azul”, “gris”, “amarillo”, o colores más exóticos, desde “AntiqueWhite” (blanco antiguo) hasta “LavenderBlush” (toque de lavanda) o “WhiteSmoke” (humo blanco). Los colores se pueden crear de muchas maneras como veremos abajo. Varios miles de nombres de colores han sido pre-asignados (**Google**: lista de nombres de colores) y pueden ser usados en el comando de arriba.

Ahora que saben cómo crear una paleta (y hacerla desaparecer) e incluso cómo determinar un color de fondo, es hora de ver qué se puede dibujar dentro de ella.

Pueden crear y dibujar todo tipo de objetos geométricos: puntos, líneas, círculos, rectángulos, e incluso textos e imágenes. Dependiendo del tipo de objeto que deseen dibujar, primero deben *crearlo* y luego dibujarlo. Sin embargo, antes de hacer esto, también deben conocer el sistema de coordenadas de la ventana de gráficos.

En una ventana de gráficos con un ancho W y una altura H (por ejemplo. 200x300 píxeles), el píxel (0, 0) está en la esquina derecha superior y el píxel (199, 299) estará en la esquina derecha inferior. Es decir, las coordenadas-x aumentan a medida que van hacia la derecha y las coordenadas-y aumentan a medida que van hacia la izquierda.

El objeto más simple que pueden crear es un *punto*. Esto se hace de la siguiente manera:

```
p = Point(100, 50)
```

Es decir, p es un objeto que es un Punto cuya coordenada-x está en 100 y cuya coordenada-y está en 50. Esto sólo crea un objeto Punto. Para dibujarlo, deben enunciar el comando:

```
p.draw(myCanvas)
```

La sintaxis del comando de arriba puede parecer un poco extraña al principio. Lo vieron brevemente cuando presentamos las listas en el Capítulo 5 pero no lo desarrollamos. Definitivamente es diferente a lo que han visto hasta ahora. Pero si piensan acerca de los objetos que han visto hasta el momento: números, strings, etc., la mayoría tienen operaciones estándares definidas (como +, *, /, etc.). Pero al pensar en objetos geométricos, no hay notaciones estándares. Los lenguajes de programación como Python proveen facilidades para modelar cualquier tipo de objetos y la sintaxis que usamos acá es la sintaxis estándar que puede ser aplicada a todo tipo de objetos. La forma general del comando enunciado para un objeto es:

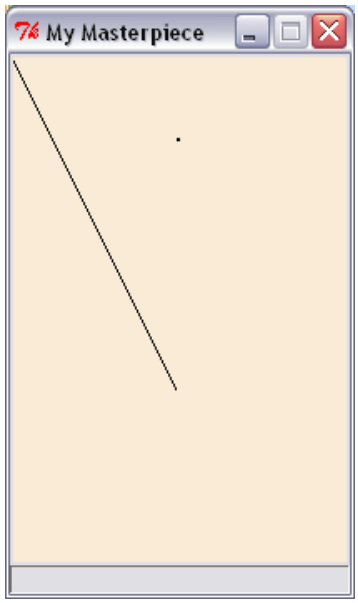
```
<objecto>.<funcion>(<parametros>)
```

Por lo tanto, en el ejemplo de arriba, <objecto> es el nombre p que anteriormente fue definido para ser un objeto Point, <funcion> es dibujar, y <parametros> es myCanvas. La función draw requiere a la ventana de gráficos como parámetro. Es decir que le están pidiendo al punto representado por p que sea dibujado en la

ventana especificada como su parámetro. Los objetos `Point` tienen otras funciones disponibles:

```
>>> p.getX()
100
>>> p.getY()
50
```

Es decir, dado un objeto `Point`, pueden obtener sus coordenadas x- e y-. Los objetos se crean usando sus *constructores* como el constructor `Point(x, y)` de arriba. Usaremos muchos constructores en esta sección para crear los objetos gráficos. Un objeto línea puede ser creado de manera similar a los objetos puntos. Una línea requiere que se especifiquen los dos puntos extremos. Por lo tanto una línea de (0, 0) a (100, 200) puede ser creada como:




```
L = Line(Point(0,0), Point(100,200))
```

Y pueden dibujar la línea usando el mismo comando de dibujo de arriba:

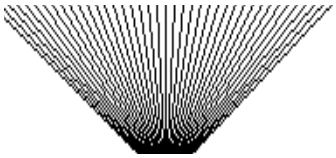
```
L.draw(myCanvas)
```

El dibujo de la derecha muestra los dos objetos que hemos creado y dibujado hasta el momento. Con respecto a `Point`, pueden obtener los valores de los puntos extremos de una línea:

```
>>> inicio = L.getP1()
>>> inicio.getX
0
>>> fin = L.getP2()
>>> fin.getY()
200
```

Aquí hay un pequeño loop de Python que puede ser usado para crear y dibujar varias líneas:

```
for n in range(0, 200, 5):
    L=Line(Point(n,25),Point(100,100))
    L.draw(myCanvas)
```



En el loop de arriba (los resultados se muestran a la derecha), el valor de `n` comienza en 0 y aumenta de a 5 después de cada iteración hasta llegar a 200 sin incluirlo 200 (por ej. 195). Para cada valor de `n` se crea un objeto `Line` (línea) nuevo con las coordenadas de inicio `(n, 25)` y el punto final en `(100, 100)`.

**Realizar la siguiente actividad:** Prueben todos los comandos introducidos hasta el momento. Luego observen los efectos producidos por el loop de arriba. Modifiquen el incremento de 5 en el loop de arriba a diferentes valores (1, 3, etc.) y observen el efecto. Luego, prueben el siguiente loop:

```
for n in range(0, 200, 5):
    L = Line(Point(n, 25), Point(100, 100))
```

```
    L.draw(myCanvas)
    wait(0.3)
    L.undraw()
```

La función de `undraw` (desdibujar) hace exactamente lo que implica el nombre. En el loop de arriba, para cada valor que toma `n`, se crea una línea (como se observa arriba), se dibuja, y luego, después de una espera de 0.3 segundos, se borra. Nuevamente, modifiquen el valor del incremento y observen el efecto. Prueben también modificar la cantidad de tiempo en el comando `wait`.

También pueden dibujar varias formas geométricas: círculos, rectángulos, óvalos y polígonos. Para dibujar un círculo (o cualquier forma geométrica), primero deben crearla y después dibujarla:

```
C = Circle(centerPoint, radius)
c.draw(myCanvas)
```

`centerPoint` es un objeto `Point` y su radio está especificado en píxeles. Por lo tanto, para dibujar un círculo centrado en (100, 150) con un radio de 30, pueden usar los siguientes comandos:

```
C = Circle(Point(100, 150), 30)
c.draw(myCanvas)
```

Los rectángulos y los óvalos se dibujan de modo similar (ver detalles al final del capítulo). Todos los objetos geométricos tienen muchas funciones en común. Por ejemplo, pueden obtener el punto central de un círculo, un rectángulo o un óvalo utilizando el comando:

```
centerPoint = C.getCenter()
```

De manera predeterminada, todos los objetos se dibujan en negro. Hay varias formas de modificar o especificar los colores de los objetos. Para cada objeto pueden especificar n color para su contorno así como un color de relleno. Por ejemplo, para dibujar un círculo centrado en (100, 150), radio 30, contorno color rojo, y relleno color amarillo:

```
C = Circle(Point(100, 150), 30)
C.draw(myCanvas)
C.setOutline(“red”)
C.setFill(“yellow”)
```

Por cierto, `setFill` (determinar relleno) y `setOutline` (determinar contorno) tienen el mismo efecto en los objetos `Point` y `Line` (dado que no hay lugar para rellenar con ningún color). Además, la línea o el contorno dibujado es siempre se un píxel de ancho. Pueden modificar este ancho usando el comando

```
setWidth(<pixels>):
C.setWidth(5)
```

El comando de arriba modifica el ancho del contorno del círculo a 5 píxeles.

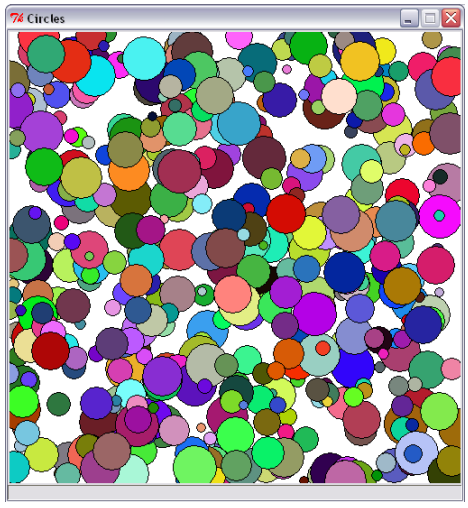
**Realizar la siguiente actividad:** Prueben todos los comandos introducidos aquí. Además, miren al final del capítulo para ver detalles sobre cómo dibujar otras formas.

Anteriormente mencionamos que varios colores tienen nombres asignados que pueden ser usados para seleccionar colores. También pueden crear colores propios especificando sus valores de rojo, verde y azule. En el Capítulo 5 mencionamos que cada color está formado por tres valores: RGB o valores de color rojo, verde y azul. Cada uno de estos valores puede estar en rango de 0..255 y se llama un valor-color 24-bit. Con esta manera de especificar colores, el color con valores (255, 255, 255) (es decir rojo = 255, verde = 255, y azul = 255) es blanco; (255, 0, 0) es rojo puro, (0, 255, 0), es azul puro, (0, 0, 0) es negro, (255, 175, 175) es rosa, etc. Pueden tener tantos colores como 256x256x256 (¡más de 16 millones de colores!). Dados los valores RGB específicos, pueden crear un color nuevo usando el comando color_rgb:

```
myColor = color_rgb(255, 175, 175)
```

**Realizar la siguiente actividad:** El programa de abajo dibuja varios círculos de tamaños al azar con colores al azar. Pruébenlo y observen los resultados. Una pantalla de muestra se puede ver a la derecha (abajo). Modifiquen el programa para incorporar un número para la cantidad de círculos que se dibujen. `randrange(m,n)` devuelve un número al azar en el rango`[m..n-1]`.

```
# Program to draw a bunch of # random colored circles
from myro import *
from random import *
def makeCircle(x, y, r):
    # creates a Circle centered at point (x, y) of radius r
    return Circle(Point(x, y), r)

def makeColor():
    # creates a new color using random RGB values
    red = randrange(0, 256)
    green = randrange(0, 256)
    blue = randrange(0, 256)
    return color_rgb(red, green,blue)
def main():
    # Create and display a
    # graphics window
    width = 500
    height = 500
    myCanvas =
GraphWin('Circles',width,height)
    myCanvas.setBackground("white")
    # draw a bunch of random
    # circles with random
    # colors.
    N = 500
    for i in range(N):
        # pick random center
        # point and radius
        # in the window
        x = randrange(0,width)
        y = randrange(0,height)
        r = randrange(5, 25)
        c = makeCircle(x, y, r)
```



```
        # select a random color
        c.setFill(makeColor())
    c.draw(myCanvas)
main()
```

Observen nuestro uso de las funciones para organizar el programa. Desde la perspectiva del diseño, las dos funciones `makeCircle` (crear círculo) y `makeColor` (crear color) se escriben de manera diferente. Esto es sólo para fines ilutrativos. Podrían, por ejemplo, definir `makeCircle` del mismo modo que `makeColor` para que no tome ningún parámetro y genere los valores de x, y, y radio de la siguiente manera:

```
def makeCircle():
    # creates a Circle centered at point (x, y) of radius r
    x = randrange(0,width)
    y = randrange(0,height)
    r = randrange(5, 25)

    return Circle(Point(x, y), r)
```

Desafortunadamente, aunque este cambio se ve muy sencillo, la función no va a andar. Para generar los valores de x y de y necesita conocer el ancho y la altura de la ventana de gráficos. Pero el ancho y la altura no están disponibles ni accesibles en la función de arriba. Este es un tema de alcance de nombres en un programa Python: ¿Cuál es el alcance de accesibilidad de un nombre en un programa?

Python define el alcance de un nombre en forma *textual* y por *léxico*. Es decir, cualquier nombre está visible en el texto del programa/función *después* de que ha sido definido. Observen que la noción de después es una noción textual. Es más, Python restringe la accesibilidad de un nombre al texto de la función en la que está definido. Es decir, los nombres ancho y altura están definidos dentro de la función main y por lo tanto no están visibles en ningún lugar fuera de main. De modo similar, la variable rojo, verde y azul se consideran locales a la definición de `makeColor` (formar color) y no están accesibles fuera de la función `makeColor`.

Entonces, ¿cómo puede `makeCircle`, si han decidido que generará los valores `x` e `y` relativos al tamaño de la ventana, obtener el acceso al ancho y a la altura de la ventana? Hay dos soluciones para esto. Primero, pueden hacerlos pasar como parámetros. En ese caso, la definición de `makeCircle` sería:

```
def makeCircle(w, h):
    # creates a Circle centered at point (x, y) of radius r
    # such that (x, y) lies within width, w and height, h
    x = randrange(0,w)
    y = randrange(0,h)
    r = randrange(5, 25)

    return Circle(Point(x, y), r)
```

Entonces, la forma en que usarían la función de arriba en el programa main sería usando el comando:

```
C = makeCircle(width, height)
```

Es decir, pasan los valores de `width` (ancho) y `height` (altura) a `makeCircle` como parámetros. La otra forma de definir `makeCircle` sería exactamente como se mostró en primera instancia:

```
def makeCircle():
    # creates a Circle centered at point (x, y) of radius r
    x = randrange(0,width)
    y = randrange(0,height)
    r = randrange(5, 25)

    return Circle(Point(x, y), r)
```

Sin embargo, moverían las definiciones de `width` y `height` afuera y antes de las definiciones de todas las funciones:

```
from myro import *
from random import *
width = 500
height = 500
def makeCircle():
    …
def makeColor():
  …
def main():
  …
```

Dado que las variables se definen fuera de cualquier función y antes de las definiciones de las funciones que las utilizan, pueden acceder a su valor. A esta altura pueden estar preguntándose ¿cuál es la mejor versión? O incluso, ¿para qué molestarse? La primera versión era igualmente buena. La respuesta a esta pregunta es similar a la forma de escribir un párrafo en un ensayo. Se pueden escribir párrafos de muchas maneras. Algunas versiones serán preferibles a otras. En programación, la primera regla que se usa en el alcance de los nombres es: asegurarse de que solamente las partes del programa  que se supone que tienen acceso al nombre tengan permitido el acceso. Esto es similar a la razón por la cual no compartirían su contraseña con nadie, o su código de tarjeta bancaria, etc. En nuestra segunda solución, hicimos que los nombres `width` y `height` estén *globalmente* visibles para el programa entero que sigue a continuación. Esto implica que aún `makeColor` puede tener acceso a ellos ya sea si lo necesita o no.

Quizás a esta altura quieran argumentar: ¿cuál es la diferencia si se hacen esas variables visibles para `makeColor` mientras uno se cuide de no usarlas en esa función? Tienen razón, no hay diferencia. Pero requiere una responsabilidad extra de su parte para asegurarse de que así sea. ¿Pero qué les garantiza que alguien que está modificando el programa no lo haga?

Usamos un programa simple para ilustrar decisiones simples pero potencialmente peligrosas que pueblan el panorama de la programación como minas de tierra. Los programas pueden romperse si nombres como estos son *mal utilizados* en un programa. Peor aún, los programas no se rompen, sino que conducen a resultados incorrectos. Sin embargo, en el nivel de decidir qué variables hacer locales y cuáles hacer globales, las decisiones son simples, y uno sólo necesita ejercitar prácticas de programación seguras. Concluiremos esta sección de gráficos con algunos ejemplos de creación de animaciones.

Cualquier objeto dibujado en la ventana de gráficos puede ser movido usando el comando `move(dx, dy)`. Por ejemplo, para mover el círculo 10 píxeles a la derecha y 5 píxeles hacia abajo, pueden usar el comando:

```
C.move(10, 5)
```

**Realizar la siguiente actividad:** Escribamos un programa que mueva el círculo de manera aleatoria en la ventana de gráficos. Primero, ingresen el siguiente programa y pruébenlo.

```
# Moving circle; Animate a circle...
from myro import *
from random import *
def main():
    # create and draw the graphics window
    w = GraphWin(“Moving Circle”, 500, 500)
    w.setBackground(“white”)
    # Create a red circle
    c = Circle(Point(250, 250), 50)
    c.setFill(“red”)
    c.draw(w)
    # Do a simple animation for 200 steps
    for i in range(200):
        c.move(randrange(-4, 5), randrange(-4, 5))
        wait(0.2)
main()
```

Observen en el programa de arriba, que estamos moviendo el círculo alrededor de manera aleatoria en las direcciones x- e y-. Prueben cambiar el rango de los movimientos y observen el comportamiento. Prueben modificar los valores para que el círculo sólo se mueva en dirección horizontal o sólo en dirección vertical. También observen que hemos tenido que *desacelerar* la animación insertando el comando wait (espera) después de cada movimiento. Comenten el comando wait y vean qué sucede. Puede parecer que no sucedió nada pero en realidad ¡los 200 movimientos pasaron tan rápido que sus ojos no pudieron registrar un solo movimiento! Usando esto de base, podemos ahora escribir un programa más interesante. Observen el programa de abajo:

```
# Moving circle; Animate a circle...
from myro import *
from random import *
def main():
    # create and draw the graphics window
    winWidth = winHeight = 500
    w = GraphWin(“Bouncing Circle”, winWidth, winHeight)
    w.setBackground(“white”)
    # Create a red circle
    radius = 25
    c = Circle(Point(53, 250), radius)
    c.setFill(“red”)
    c.draw(w)

    # Animate it
    dx = dy = 3
    while timeRemaining(15):
        # move the circle
        c.move(dx, dy)
```

```
        # make sure it is within bounds
        center = c.getCenter()
        cx, cy = center.getX(), center.getY()
        if (cx+radius >= winWidth) or (cx-radius <= 0):
            dx = -dx
        if (cy+radius >= winHeight) or (cy-radius <= 0):
            dy = -dy
wait(0.01)
main()
```

Durante 15 segundos, verán un círculo rojo rebotando alrededor de la ventana. Estudien el programa de arriba para ver cómo mantenemos el círculo dentro de la ventana todo el tiempo y cómo la dirección del rebote de a pelota se modifica. Cada vez que cambien la dirección, hagan que la computadora emita un beep:

```
computer.beep(0.005, 440)
```

Si los entusiasma la posibilidad de animar un círculo, imaginen lo que pueden hacer si tienen muchos círculos y otras formas animadas. Además, conecten el controlador *game pad* y vean si pueden controlar el círculo (o cualquier otro objeto) usando los controles del mismo. Esto es muy similar a controlar el robot. Diseñen un juego interactivo de computadora que aproveche esta nueva modalidad de input. También pueden diseñar juegos multi-usuarios dado que pueden conectar múltiples controladores game pad a su computadora. Vean la documentación de Referencias para más detalles sobre cómo obtener input de varios controladores game pad.

## Dibujar Textos & Imágenes

Así como se pueden ubicar formas, también se pueden colocar textos en la ventana de gráficos. La idea es la misma. Primero crean el texto usando el comando:

```
myText = Text(<anchor point>, <string>)
```

y luego lo dibujan.

Pueden especificar la tipografía, tamaño y estilo del texto. No entraremos en esos detalles aquí. Se sintetizan en la referencia al final del texto. Cuando tengamos la oportunidad, usaremos estas funciones abajo en otros ejemplos.

Las imágenes en este sistema son tratadas igual que cualquier otro objeto. Pueden crear una imagen usando el comando `Image`:

```
myPhoto = Image(<centerPoint>, <file name>)
```

Deben tener preparada una imagen en uno de los formatos de imágenes comunes (como JPEG, GIF, etc.) y alojado en un archivo (`<file name>`). Una vez que el objeto imagen se ha creado, puede ser dibujado, borrado o movido, al igual que las otras formas.

Lo que ustedes hagan con esta funcionalidad recién descubierta depende de su creatividad. Pueden usar gráficos para visualizar algunos datos: ploteo del crecimiento de la población mundial, por ejemplo; o crear arte, un juego interactivo, o incluso una historia animada. Pueden combinar su robot, el controlador game pad, e incluso sonido para enriquecer la experiencia multi-media. Los ejercicios al final

del capítulo presentan algunas ideas para explorar esto aún más. Abajo, ahondaremos en los sonidos.

## Sonido

Como hemos visto, pueden hacer que el robot haga *beeps* recurriendo a la función `beep()`, así:

```
beep(1, 440)
```

Este comando lo instruye al robot a reproducir un tono de 440 Hz por 1 segundo. Tratemos primero de analizar qué hay en el tono de *440 Hz*. Primero, las letras *Hz* son una abreviación de *Hertz*. El nombre proviene de un físico alemán, Heinrich Rudolph Hertz, quien fue pionero en el trabajo de la producción de ondas electromagnéticas en el siglo 19. Hoy usamos el **Hertz** (o *Hz*) como unidad para especificar frecuencias.

$1 Hertz = 1 cycle / second$

El uso más común de las frecuencias en la actualidad es para especificar las velocidades del reloj de la CPU de la computadora. Por ejemplo, una típica PC hoy en día corre a la velocidad reloj de pocos GigaHertz (or GHz).

$1 GigaHertz = 10^9 cycles / second$

Las frecuencias se relacionan con movimientos o vibraciones periódicas (repetitivas). El tiempo que le lleva a un movimiento o vibración repetirse se llama *período de tiempo*. La frecuencia y el período de tiempo están inversamente relacionados. Es decir, se llama frecuencia al número de ciclos o repeticiones en un segundo. Por lo tanto, 1 Hertz hace referencia a cualquier movimiento o vibración que se repite cada 1 segundo. En el caso de la frecuencia del reloj de la computadora, una computadora que corre a 4 Gigahertz ¡está repitiendo 4 billones de veces por segundo! Otros ejemplos de movimientos periódicos incluyen: la rotación de la tierra sobre su eje (1 ciclo cada 24 * 60 * 60 = 86400 segundos o a la frecuencia de 0.00001157 ciclos/segundo), un típico CD de audio da vueltas 400 veces por segundo, una unidad de CD en su computadora valorado en 52x gira el CD 52 * 400 = 20800 veces por segundo, los colibríes pueden batir sus alas a frecuencias que oscilan entre 20-78 veces/segundo (¡algunas llegan hasta 200!).

El sonido es una compresión y refracción (o regreso a su estado original) periódica de aire (para simplificarlo, asumamos que el medio es el aire). Un ciclo de un sonido comprende una compresión y una refracción. Por lo tanto, un beep a 440 Hz representa 440 ciclos completos de compresión y refracción. Generalmente, un oído humano es capaz de escuchar frecuencias en un rango de 20 Hz a 20000 Hz (o 20 Kilo Hertz). Sin embargo, la capacidad varía de persona a persona. Además, muchos dispositivos electrónicos no son capaces de producir frecuencias en ese rango completo. 20-20KHz se considera alta-fidelidad para un estéreo o componentes de audio *home theater*. Examinemos primero el rango de sonidos audibles que puede producir el Scribbler. Para sacar un sonido del Scribbler, deben darle una frecuencia y duración (en segundos) en que debe ser reproducido el sonido. Por ejemplo, para emitir un sonido de 440 Hz por 0.75 segundos:

```
beep(0.75, 440)
```

El oído humano es capaz de distinguir sonidos que difieren solamente por pocos Hertz (tan poco como 1 Hz) sin embargo, esta habilidad varía de persona a persona. Prueben los comandos:

```
beep(1, 440)
beep(1, 450)
```

¿Pueden distinguir entre los dos tonos? A veces ayuda ubicarlos en un loop para que puedan escuchar los tonos alternados repetidamente como para poder distinguirlos entre sí. El siguiente ejercicio puede ayudarlos a determinar qué frecuencias son capaces de distinguir.

**Ejercicio:** Usando el ejemplo de arriba, intenten ver cuánto se acercan a distinguir frecuencias cercanas. Como se sugirió antes, quizás deseen emitir los tonos alternándolos durante 5-10 segundos. Comiencen con 440 Hz. ¿Pueden notar la diferencia entre 440 y 441? 442? etc. Una vez que han establecido su alcance, prueben otra frecuencia, digamos 800. ¿Es la misma la distancia que pueden distiguir?

**Realizar la siguiente actividad:** Pueden programar al Scribbler para crear una sirena, repitiendo dos tonos diferentes (bastante similar al ejemplo de arriba). Deberán experimentar con distintos pares de frecuencias (pueden estar cercar o muy distantes entre sí) para producir una sirena que suene realista. Escriban su programa para emitir una sirena durante 15 segundos. ¡Cuánto más fuerte el volumen mejor!

También pueden hacer que Myro emita directamente un beep desde la computadora, en lugar del robot, con el comando:

```
computer.beep(1, 440)
```

Desafortunadamente, no pueden hacer que el robot y la computadora toquen a dúo. ¿Por qué not? Prueben estos comandos:

```
beep(1, 440)
computer.beep(1, 440)
beep(1, 880)
computer.beep(1, 880)
beep(1, 440)
computer.beep(1, 440)
```

¿Qué ocurre? Prueben sus soluciones a los ejercicios de arriba emitiendo los sonidos desde la computadora en lugar de desde el Scribbler.

## Escalas Musicales

En la música occidental, una *escala* está dividida en 12 notas (de 7 notas mayores: ABCDEFG –la, si, do, re, mi, fa, sol). Además hay octavas Una octava en C se compone de las notas que se muestran abajo:

C C#/Db D D#/Eb E F F#/Gb G G#/Ab A A#/Bb B

C# (pronunciada “C sostenido”) es el mismo tono que Db (pronunciada “D bemol”).

Las frecuencias correspondientes a una nota específica, digamos C, se multiplican (o dividen) por 2 para obtener la misma nota en una octava más alta (o más baja). En un piano hay disponibles varias octavas en el despliegue de teclas. ¿Cuál es la relación entre estos dos tonos?

```
beep(1, 440)
beep(1, 880)
```

El segundo tono es exactamente una octava del primero. Para subir un tono una octava, simplemente multiplican la frecuencia por 2. Del mismo modo, para hacer que un tono sea una octava más bajo, lo dividen por 2. Las notas que indican una octava pueden anotarse de la siguiente manera:

```
C0 C1 C2 C3 C4 C5 C6 C7 C8
```

Es decir, C0es la nota para C en la octava más baja (or 0). La quinta octava (numerada 4) es comúnmente referida como la octava del medio. Por lo tanto, C es la nota C en la octava del medio La frecuencia que corresponde a C es 261.63 Hz. Intenten emitirla en el Scribbler. También prueben C5 (523.25) que es dos veces la frecuencia de C4 y C3 (130.815). En sintonías comunes (*iguales temperamentos*) las 12 notas son equidistantes. Por lo tanto, si la frecuencia duplica cada octava, cada nota sucesiva es de una distancia de $2^{1/12}$. Es decir, si C4 es 261.63 Hz, C# (o Db) será:

C#4/Db4 = $261.63 * 2^{\frac{1}{12}} = 277.18$

Por lo tanto, podemos computar todas las sucesivas frecuencias:

*D*4 = $277.18 * 2^{1/12} = 293.66$

D#4/Eb = $293.66 * 2^{\frac{1}{12}} = 311.13$

etc.

El tono más bajo que puede reproducir el Scribbler es A0 y el más alto es C8. A0 tiene una frecuencia de 27.5 Hz, y C8 tiene una frecuencia de 4186 Hz. ¡Eso es un rango bastante alto! ¿Pueden escuchar el espectro completo?

```
beep(1, 27.5)
beep(1, 4186)
```

**Ejercicio:** Escriban un programa Scribbler para reproducir las 12 notas en una octava usando la computación de arriba. Pueden asumir en su programa que C0 es 16.35 y luego usar eso para computar todas las frecuencias en una octava dada (C4 es $16.35 * 2^4$). Su programa debería utilizar una octava (un número de 0 hasta 8), produzcan todas las notas en esa octava y también impriman una tabla de frecuencia para cada nota en esa octava.

## Hacer Música

Reproducir canciones por frecuencias es bastante molesto. Myro tiene un conjunto de funciones para abstraer este proceso. Una canción Myro es un *string* de caracteres compuesto de la siguiente manera:

NOTE1 [NOTE2] WHOLEPART

donde `[]` significa opcional. Cada una de estas notas /acordes está compuesto sobre su propia línea, o separado por comas donde:

NOTE1 is either a frequency or a NOTENAME (NOTE1 es o una frecuencia o un NOTENAME- nombre de nota)

NOTE2 is the same, and optional. Use for Chords. (NOTE2 es lo mismo, y opcional. Usar para acordes).

WHOLEPART is a number representing how much of

a whole note to play. (WHOLEPART es un número que representa cuánto reproducir de una nota entera).

`NOTENAMES` son *strings* insensibles a las mayúsculas o minúsculas (*case insensitive*). Aquí hay una escala entera de `NOTENAMES`:

C C#/Db D D#/Eb E F F#/Gb G G#/Ab A A#/Bb B C

Esta es la octava predeterminada. También es la quinta octava, que también puede escribirse así:

C5 C#5/Db5 D5 D#5/Eb5 E5 F5 F#5/Gb5 G5 G#5/Ab5 A5 A#5/Bb5 B5 C6

El Formato Myro Song (Canción Myro) replica las teclas del piano, por lo cual va desde A0 hasta C8. La octava del medio en un teclado es la número 4, pero usamos la 5 como la octava predeterminada. Ver http://en.wikipedia.org/wiki/Piano_key_frequencies para más detalles. Aquí hay una escala:

“C 1; C# 1; D 1; D# 1; E 1; F 1; F# 1; G 1; G# 1; A 1; A# 1; B 1; C 1;”

La escala, una octava más baja, y tocada como una polka:

“C4 1; C#4 ½; D4 ½; D#4 1; E4 ½; F4 ½; F#4 1; G4 ½; G#4 ½; A4 1; A#4 ½; B4 ½; C4 1;”

También hay algunos nombres de notas especiales más, incluyendo `PAUSE(pausa)`, `REST (descanso)`, pueden dejar el número de octava apagado de las notas de octava predeterminadas si lo desean. Usen “#” para los sostenidos, y “b” para los bemoles.

`WHOLEPART` puede ser una notación decimal o una división. Por ejemplo:

Ab2 .125

o

Ab2 1/8

Representa el A bemol en la segunda octava (dos más abajo que del medio).

A modo de ejemplo, prueben reproducir lo siguiente:

c 1

c .5

c .5

c 1

c .5

c .5

e 1

c .5

c .5

c 2

e 1

e .5

e .5

e 1

e .5

e .5

g 1

e .5

e .5

e 2

¿Lo reconocen?

Pueden dejar líneas en blanco, y los comentarios deben comenzar con un signo #. Las líneas también pueden estar separadas por comas.

## Usando una canción

Para el siguiente ejercicio, necesitarán tener un objeto para reproducir la canción. Necesitarán iniciar el robot de una manera un poco distinta. En lugar de:

```
initialize()
```

hagan lo siguiente:

```
robot = Scribbler()
```

Ahora que tienen una canción, probablemente la querrán reproducir. Si su canción está en un archivo, pueden leerla:

```
s = readSong(filename)
```

y reproducirla en el robot:

```
robot.playSong(s)
```

o en la computadora:

```
computer.playSong(s)
```

También pueden usar `makeSong(text)` para hacer una canción. Por ejemplo:

```
s = makeSong(“c 1; d 1; e 1; f 1; g 1; a 1; b 1; c7 1;”)
```

y luego reproducirla como se muestra arriba.

Si quieren reproducir la canción más rápido o más despacio, podrían cambiar todos los números `WHOLENOTE`. Pero si sólo queremos modificar el tempo, hay una manera más sencilla:

```
robot.playSong(s, .75)
```

El segundo argumento para `playSong` es la duración de una nota entera en segundos. El tempo estándar toca una nota entera en aproximadamente .5 segundos. Los números más grandes se reproducirán más lentamente, y los números más chicos se reproducirán más rápido.

## Síntesis

Pueden usar la ventana de gráficos como modo de visualizar cualquier cosa. En la ventana de gráficos pueden dibujar pueden dibujar todo tipo de formas: puntos, líneas, círculos, rectángulos, óvalos, polígonos, textos e incluso imágenes. También pueden animar estas formas como lo deseen. Lo que hagan con estas capacidades básicas de dibujos está limitado solamente por su creatividad y sus habilidades como programadores. Pueden combinar las vistas creadas con sonidos y otras interfaces, como el controlador game pad, o incluso su robot. Las funcionalidades multimedia introducidas en este capítulo pueden ser usadas en todo tipo de situaciones para crear programas interesantes.

## Referencia Myro

Abajo sintetizamos todos los comandos de gráficos mencionados en este capítulo. Los instamos a revisar el manual de referencias para más funcionalidades de gráficos.

`GraphWin()`
`GraphWin(<title>, <width>, <height>)`
Devuelve un objeto ventana de gráficos. Crea una ventana de gráficos con titulo, `<title>` y dimensiones `<width>` x `<height>`. Si no se especifican parámetros, la ventana creada es de 200x200 píxeles.

`<window>.close()`
Closes the displayed graphics window <window>.

`<window>.setBackground(<color>)`
Establece el color de fondo de la ventana al color especificado. `<color>` puede ser un color nombrado (Google: lista de nombres de colores –*color name list*), o un nuevo color creado usando el comando `color_rgb` (ver abajo).

`color_rgb(<red>, <green>, <blue>)`
Crea un nuevo color usando los valores `<red>`, `<green>`, y `<blue>` especificados. Los valores pueden estar dentro del rango 0..255.

`Point(<x>, <y>)`
Crea un objeto punto en la ubicación (`<x>`, `<y>`) en la ventana.

`<point>.getX()`
`<point>.getY()`
Devuelve las coordenadas *x* e *y* del objeto punto `<point>`.

`Line(<start point>, <end point>)`
Crea un objeto línea en `<start point>` que termina en `<end point>`.

`Circle(<center point>, <radius>)`
Crea un objeto círculo centrado en `<center point>` con un radio de `<radius>` píxeles.

`Rectangle(<point1>, <point2>)`
Crea un objeto rectángulo con esquinas opuestas ubicadas en `<point1>` y `<point2>`.

`Oval(<point1>, <point2>)`
Crea un objeto óvalo en la caja definida por los puntos de la esquina `<point1>` y `<point2>`.

`Polygon(<point1>, <point2>, <point3>,…)`
`Polygon([<point1>, <point2>, …])`
Crea un polígono con los puntos dados como vértices.

`Text(<anchor point>, <string>)`
Crea un texto anclado (la esquina inferior izquierda del texto) en `<anchor point>`. El texto en sí mismo se define por `<string>`.

`Image(<centerPoint>, <file name>)`
Crea una imagen centrada en `<center point>` del archivo de imágenes `<file name>`. La imagen puede ser en formato GIF, JPEG, o PNG.

Todos los objetos gráficos responden a los siguientes comandos:

`<object>.draw(<window>)`
Dibuja el `<object>` en la ventana de gráficos especificada `<window>`.

`<object>.undraw()`
Deshace el dibujo `<object>`.

`<object>.getCenter()`
Devuelve el punto central del `<object>`.

`<object>.setOutline(<color>)`
`<object>.setFill(<color>)`
Establece el color de delineado y de relleno del `<object>` en el `<color>` especificado.

`<object>.setWidth(<pixels>)`
Establece el ancho del delineado del `<object>` en `<pixels>`.

`<object>.move(<dx>, <dy>)`
Mueve el objeto `<dx>`, `<dy>` de su posición actual.

Las siguientes funciones relacionadas con el sonido se presentaron en este capítulo.

`beep(<seconds>, <frequency>)`
`beep(<seconds>, <f1>, <f2>)`
Hace que el robot genere un beep durante la frecuencia de tiempo `<seconds>` especificada. Pueden especificar una sola frecuencia `<frequency>` o una mezcla de dos: `<f1>` y `<f2>`.

`<robot/computer object>.beep(<seconds>, <frequency>)`
`<robot/computer object>.beep(<seconds>, <f1>, <f2>)`
Hace que el robot o la computadora realice un beep durante el tiempo `<seconds>` en la frecuencia especificada. Pueden especificar una sola frecuencia `<frequency>` o una mezcla de los dos: `<f1>` y `<f2>`.

`robot.playSong(<song>)`
Reproduce la <song> (canción) en el robot.

readSong(<filename>)
Lee un archivo de canción de <filename>.

`song2text(song)`
Convierte una <song> al formato texto.

`makeSong(<text>)`
`text2song(<text>)`
Convierte el `<text>` (texto) en un formato de canción.

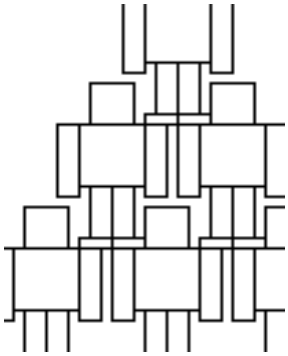
## Referencias Python

En este capítulo presentamos *alcances de las normas* informales para nombres en programas Python. Mientras que estas se pueden complicar bastante, para nuestros fines deben conocer la distinción entre un *nombre local* que es local dentro de la esfera de una función, versus un *nombre global* definido por fuera de la función. El ordenamiento del texto define lo que es accesible.

## Ejercicios

**1.** Usando los comandos de gráficos aprendidos en este capítulo, escriban un programa para generar un dibujo estacional: invierno, verano, playa o una escena festiva.

**2.** Escriban un programa que tenga la función `drawRobot(x, y, w)` de modo tal que dibuje la figura de un robot como el que se muestra al lado. El robot está ubicado en (`x`, `y`) y está dibujado en la ventana, `w`. También noten que el robot está hecho enteramente de rectángulos (8 rectángulos).



**3.** Usando la función `drawRobot` del ejercicio previo, dibujen una pirámide de robots como se muestra en el dibujo a la derecha.

**4.** Supongan que cada robot del ejercicio previo está coloreado usando uno de los colores: rojo, azul, verde y amarillo. Cada vez que se dibuja un robot, utiliza el siguiente color en la secuencia. Una vez que se terminó la secuencia, la recicla. Escriban un programa para dibujar la pirámide de arriba de manera tal que los robots aparezcan en estos colores a medida que son dibujados. Pueden decidir modificar `drawRobot` para tener un parámetro adicional: `drawRobot(x, y, c, w)` o pueden escribirlo como para que `drawRobot` decida qué color elegir. Completen ambas versiones y comparen los programas resultantes. Discutan lo méritos y/o debilidades de estas versiones con sus amigos [Una ayuda: Usen una lista de nombres de colores.]

**5.** El comando beep puede reproducir dos tonos simultáneamente. En este ejercicio demostraremos un fenómeno interesante. Primero, prueben los siguientes comandos:

```
beep(2, 660)
beep(2, 665)
```

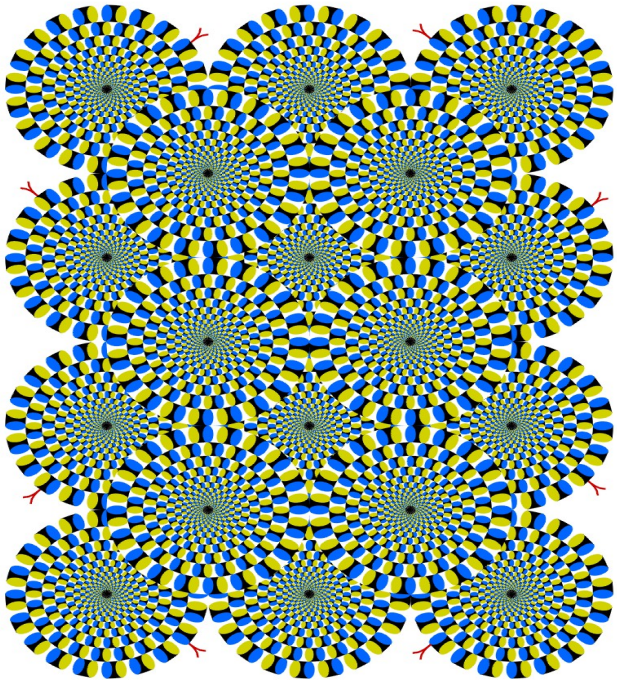
Quizás les resulte posible diferenciar los dos tonos. A continuación, prueben reproducir los dos tonos juntos usando el comando:

```
beep(2, 660, 665)
```

Escucharán latidos pulsantes. Este fenómeno se llama *pulsación* y es muy común cuando se reproducen tonos puros juntos. Expliquen por qué sucede esto.

**6.** Realicen una búsqueda en la Web acerca de *fractales* y escriban programas para dibujar algunos fractales simples. Por ejemplo, *Copos de Nieve Koch*, *Triángulos Sierpinski*, *Conjuntos de Mandelbrot*, *Conjuntos de Julia*, etc.

# 9 Procesamiento de imagen & Percepción



*Ver es creer.*
Proverbio

Página opuesta: Serpientes rotatorias (Una ilusión visual)
Creado por Akiyoshi Kitaoka (www.ritsumei.ac.jp/~akitaoka/index-e.html)

El robot tiene una pequeña cámara digital que puede ser utilizada para sacar fotos. Una foto tomada por una cámara digital se representa como una *imagen*. Como pudieron ver en el capítulo previo, las imágenes pueden ser dibujadas y se las puede mover dentro de una ventana de gráficos como si fueran cualquier otro objeto gráfico (como una línea, un círculo, etc.). También vimos en el capítulo 5 cómo una imagen tomada de la cámara del robot puede ser visualizada en sus monitores en una ventana aparte. En este capítulo aprenderemos cómo realizar computación sobre imágenes. Aprenderemos cómo se representan las imágenes y cómo podemos crearlas a través de la computación y procesarlas de maneras muy distintas. La representación de imágenes que usaremos es la misma que la que utilizan la mayoría de las cámaras digitales y teléfonos celulares, y la misma representación puede ser usada para mostrarlas en una página Web. También aprenderemos cómo una imagen tomada por la cámara del robot puede ser útil como el ojo de la cámara hacia este mundo usando algunas técnicas de compresión de imágenes. La compresión de imágenes es el primer paso hacia la percepción visual. Un robot equipado con aunque fuera las capacidades de percepción visual más rudimentarias puede ser diseñado para llevar a cabo comportamientos más importantes.

## ¿Qué es una imagen?

En Myro pueden emitir un comando para que el robot saque una foto y la muestre en el monitor de la computadora usando los comandos:

```
pic = takePicture()
show(pic)
```

La foto de la siguiente página muestra un ejemplo de una imagen tomada con la cámara del robot. Una imagen se compone de muchos pequeños elementos o *píxeles*. En una imagen color, cada píxel contiene información del color conformada por la cantidad de valores de rojo, verde y azul (también llamada RGB). Cada uno de estos valores puede estar dentro del rango [0..255] y por lo tanto se requieren 3 bytes o 24 bits para alojar la información contenida en un píxel. Un píxel de color rojo puro tendrá los valores RGB (255, 0, 0). Una imagen en blanco y negro, por otro lado, sólo contiene el nivel de gris en un píxel que puede ser representado en un solo byte (8 bits) como un número que va desde el 0..255 donde 0 es el negro y 255 es el blanco. La imagen entera es sólo una formación doble dimensional de píxeles. Por ejemplo, las imágenes obtenidas con el Scribbler tienen 256x192 (WxH) o un total de 49,152 píxeles. Dado que cada píxel requiere de 3 bytes de datos, esta imagen tiene un tamaño de 147,456 bytes.


Todas las cámaras digitales se venden especificando el número de megapixeles. Por ejemplo, la cámara que se muestra en la figura posee a 6.3 megapíxeles. Esto hace referencia al tamaño de la imagen más

grande que puede sacar. Cuantos más píxeles en una imagen, mejor será la resolución de la imagen al imprimirla. Con una imagen de 6.3 megapíxeles serán capaces de crear impresiones de buena calidad en un tamaño de 13x12 pulgadas, e incluso más. En comparación, un rollo fotográfico tiene apenas 4000x3000 o 12 millones de píxeles. Las cámaras digitales fácilmente sobrepasan esta resolución y esto es por que, en la última década hemos visto un rápido declive de la fotografía de rollo. Para mostrar imágenes nítidas en una computadora se necesita menos de la mitad de la resolución ofrecida por la cámara que se muestra aquí.

La cámara del Scribbler, con una imagen de 147,456 bytes es de solamente 0.14 megapixels. Aunque sea una resolución baja, es suficiente para realizar la percepción visual del robot.

Para alojar y transferir electrónicamente imágenes (web, e-mail, etc.) de manera rápida y conveniente, los datos en una imagen se pueden comprimir. Varios formatos están disponibles para alojar imágenes electrónicamente: JPEG, GIF, PNG, etc. JPEG es el formato más común usado por cámaras digitales, incluyendo la del Scribbler. El JPEG permite una excelente compresión, sumada a un rango más amplio de colores en comparación con el formato GIF, lo cual lo hace más útil para la mayoría de las aplicaciones de imágenes. Myro soporta formatos JPEG y GIF. Cuando tengamos la intención de procesar una imagen, siempre usaremos el formato JPEG. Usaremos el formato GIF para crear imágenes animadas.

## Cuestiones Básicas de Imágenes Myro

Después de sacar una foto con el Scribbler como hicimos arriba, pueden obtener información acerca del tamaño de la imagen con las funciones `getWidth()` y `getHeight()`:

```
picWidth = getWidth(pic)
picHeight = getHeight(pic)
print "Image WxH is", picWidth, "x", picHeight, “pixels.”
```

Si desean guardar la imagen para un uso futuro, pueden usar el comando Myro:

```
savePicture(pic, "OfficeScene.jpg")
```

El archivo `OfficeScene.jpg` será guardado en la carpeta actual. La extensión .jpg le indica al comando que guarde la imagen como JPEG. Si desean guardarla como una imagen GIF, tendrán que usar la extensión. gif, como se muestra abajo:

```
savePicture(pic, "OfficeScene.gif")
```

Más adelante, podrán cargar la foto del disco con la función `makePicture()`:

```
mySavedPicture = makePicture("OfficeScene.jpg")
show(mySavedPicture)
```

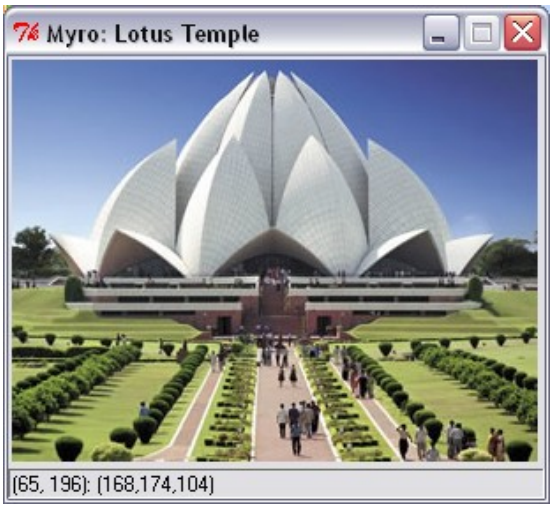
Una linda combinación de comandos que permite navegar y seleccionar la imagen a cargar es la siguiente:

```
mySavedPicture = makePicture(pickAFile())
show(mySavedPicture)
```

El comando `pickAFile` provee una caja de diálogo navegable, igual a la que se puede ver cuando abren y seleccionan archivos en cualquier otra aplicación de computadora. Pueden navegar hacia cualquier carpeta y seleccionar un archivo para abrir como imagen. De hecho, pueden usar el comando `makePicture` para cargar cualquier archivo de imagen JPEG sin importar si fue creado de su propia cámara o la han bajado de la web. Abajo les mostramos cómo cargar una imagen y mostrarla:

```
lotusTemple = makePicture(pickAFile())
show(lotusTemple, “Lotus Temple”)
```

Si mueven el mouse y hacen click en cualquier parte de la foto, también pueden obtener las coordenadas x- e y- del píxel que han seleccionado, junto con sus valores RGB. Esto se muestra en la foto a la derecha. El mouse se ha cliqueado en el píxel ubicado en (65, 196) y sus valores RGB eran (168,174,104). ¿Pueden adivinar dónde es esta ubicación en la foto? Dicho sea de paso, el comando show toma un segundo parámetro opcional que es una cadena de caracteres que se transforma en el título de la ventana de la imagen. Una ventaja de ser capaces de cargar y visualizar cualquier imagen es que también podemos aprender a procesar o manipular estas imágenes en la computadora. Volveremos al procesamiento de imágenes más adelante en el capítulo.




## Un robot explorador

Si no necesitan una imagen a color, pueden decirle a Myro que capture una imagen en escala de grises dándole a la función `takePicture()` el parámetro `"gray"`.

```
grayPic = takePicture("gray")
show(grayPic)
```

Habrán notado que sacar la foto en grises tomó menos tiempo que sacar la foto en colores. Como explicamos anteriormente, una foto en escala de grises sólo utiliza un byte por píxel, en lugar de tres. Dado que las imágenes en escala de grises pueden ser transferidas del robot a su computadora de manera más rápida que las de color, pueden ser útiles si desean que se actualicen rápidamente. Por ejemplo, pueden usar la función `joyStick()` combinada con un loop que saca la foto y la muestra para trasformar el robot en un explorador piloteado, similar a los rovers de Marte.

```
joyStick()
for i in range(25):
  pic = takePicture("gray")
  show(pic)
```

El código de arriba abrirá una ventana de joy stick para que puedan controlar al robot y luego tomar y mostrar 25 fotos, una después de la otra. Mientras que se están tomando y mostrando las fotos como en una película, pueden usar el joystick para manejar al robot, usando las imágenes para guiarlo. Por supuesto que si

sacaran el parámetro "gray" de la función takePicture(), obtendrían fotos en color en lugar de en escala de grises, pero tardarían mucho más en transferirse del robot a su computadora, y dificultarían el control del robot.

## Películas GIF animadas

La función savePicture() también les permite realizar un GIF animado, que es un tipo especial de foto que en un navegador Web mostrará varias fotos, una después de la otra, en una animación. Para guardar un GIF animado, deben darle a la función savePicture() una lista de fotos (en lugar de una sola) y un nombre de archivo que termine en ".gif". Este es un ejemplo:

```
pic1 = takePicture()
turnLeft(0.5,0.25)
pic2 = takePicture()
turnLeft(0.5,0.25)
pic3 = takePicture()
turnLeft(0.5,0.25)
pic4 = takePicture()

listOfPictures = [pic1, pic2, pic3, pic4]

savePicture(listOfPictures, "turningMovie.gif")
```

La mejor manera de visualizar un archivo GIF animado es usando un navegador Web. En su navegador favorito, usen el menú FILE->Open File, y luego elijan el archivo turningMovie.gif. El navegador Web mostrará todos los cuadros en la película y se detendrá en el último. Para volver a ver la película, presionen el botón de "Reload". También pueden usar un loop para hacer una película más larga con más imágenes:

```
pictureList = []  #Comenzamos con una lista vacía
for i in range(15):
  pic = takePicture()
  pictureList = pictureList + [pic]   #Agregamos una nueva foto
  turnLeft(0.5,0.1)

savePicture(pictureList,"rotatingMovie.gif")
```

Los comandos de arriba crean una película GIF animada de 15 fotos tomadas mientras que el robot rotaba en su lugar. Esta es una buena manera de capturar una escena completa alrededor del robot.

## Hacer imágenes

Dado que una imagen es sólo un conjunto de píxeles, también es posible crear o componer las propias imágenes coloreando cada píxel en forma individual. Myro provee comandos simples para llenar o examinar un valor de color o de grises en píxeles individuales. Pueden usar cómputos para determinar qué llenar en cada píxel. Para empezar, pueden crear una imagen en blanco de la siguiente manera:

```
W = H = 100
newPic = makePicture(W, H, black)
show(newPic)
```

La imagen de 100x100 píxeles comienza con todos sus píxeles coloreados de negro puro (por ejemplo RGB = (0,0,0)). Si preferirían que todos los píxeles tuvieran un valor diferente, pueden especificar sus valores RGB:

```
newPic = makePicture(W, H, makeColor(R,G,B))
```

También, si alguna vez lo necesitan, también pueden establecer los píxeles en cualquier color, digamos que blanco, usado el loop:

```
for x in range(W)
  for y in range(H):
    pixel = getPixel(newPic, x, y)
    setColor(pixel, white)
repaint(newPic)
```

El comando `getPixel` devuelve el píxel en las ubicaciones `x-` e `y-` especificadas en la imagen. `setColor` establece el color del píxel dado a cualquier color que se especifique. Arriba estamos usando el color blanco predeterminado. Pueden crear un color nuevo especificando sus valores RGB en el comando:

```
myRed = makeColor(255, 0, 0)
```

Para seleccionar visualmente un color y sus valores RGB correspondientes, pueden usar el comando:

```
myColor = pickAColor()
```

Se mostrará una paleta de colores de la cual podrán seleccionar un color a elección. La paleta también les muestra los valores RGB del color elegido. Luego de elegir un color y presionar OK, el valor de `myColor` será el color seleccionado.

El comando `repaint` actualiza la imagen expuesta para que puedan ver los cambios que han hecho. En el loop de ejemplo de arriba, lo estamos usando luego de que todos los píxeles se han modificado. Si desean ver los cambios a medida que se realizan, pueden incluir el comando repaint dentro del loop:

```
for x in range(W)
  for y in range(H):
    pixel = getPixel(newPic, x, y)
    setColor(pixel, white)
    repaint(newPic)
```

Podrán visualizar cada píxel que se está modificando. Sin embargo, también notarán que esto lleva una cantidad considerable de tiempo aún en la pequeña imagen que estamos creando. Por eso es una buena idea la de actualizar una vez que se hayan modificado todos los píxeles.

Sumado al comando `setColor`, Myro tiene también el comando `setRGB` que puede ser utilizado para establecer el color de un píxel. Este comando usa valores RGB en lugar de un color.

```
setRGB(pixel, (255, 0, 0))
```

También hay un comando correspondiente para obtener los valores RGB de un píxel dado:

```
r, g, b = getRGB(pixel)
```

El comando `getRGB` devuelve el triple (R,G,B) que puede ser asignado a una variable individual como se muestra arriba. Además, dado un píxel, pueden obtener los valores RGB individuales usando los comandos `getRed`, `getGreen`, y `getBlue`. Estos se describen con más detalle al final de este capítulo y se ilustran con ejemplos abajo.

Muchas situaciones de creación y procesamiento de imágenes requieren el procesamiento de cada píxel de la imagen. En los loops de arriba, estamos usando las variables x- e y- para alcanzar a cada píxel en la imagen. Una forma alternativa de lograr el mismo resultado es usando la siguiente forma del loop:

```
for pixel in getPixels(newPic):
  setColor(pixel, gray)
repaint(newPic)
```

Como en la función `timeRemaining` usada en los capítulos anteriores, `getPixels` devuelve el siguiente píxel a ser procesado cada vez alrededor del loop, garantizando de esta manera que todos los píxeles de la imagen estén procesados. Solamente para observar cómo el loop de arriba funciona, quizás deseen probar y poner repaint dentro del loop. La diferencia entre los dos métodos es que el primer loop obtiene un píxel con unas coordenadas `x-` e `y-` dadas, mientras que el último les da un píxel a la vez, sin preocuparse por sus valores `x-` e `y-`.

## Sombras de Gris

Usando los comandos de acceso y modificación de píxeles de imágenes se pueden escribir cómputos para crear imágenes interesantes y creativas. Para introducirlos en las técnicas básicas de creación y procesamiento de imágenes, crearemos algunas imágenes completamente dentro del espectro de la escala de grises. En una imagen JPEG, todas las sombras de gris tienen iguales valores RGB. La sombra más oscura de gris es (0,0,0) o negro y la más clara es (255,255,255) o blanca. En lugar de preocuparnos por los triples valores RGB, podemos pensar solamente en un valor en el rango de 0..255 para representar un espectro de grises. Podemos escribir una simple función que devolverá el tono RGB de gris de la siguiente manera:

```
def gray(v):
   # Retorna un color gris RGB para  v
   return makeColor(v, v, v)
```

Generemos una imagen que muestre toda la escala de tonos de gris usando la siguiente regla:

*pixel at x, y is colored using the grayscale x+y*

Es decir, un píxel de la imagen en 0, 0 obtendrá un tono 0+0=0 o negro, y un píxel en 50,50 obtendrá el tono

50+50=100 lo cual será un gris intermedio. Podemos lograr esto usando el siguiente loop:

```
def main():
   MAX = 255
   W = 100
   H = 100
   # Creamos una imagen en blanco
   myPic = makePicture(W, H)
   show(myPic, “Escalas de grises”)
   # Llenamos cada  pixel con  x+y escala de gris
   for x in range(W):
      for y in range(H):
         setPixel(myPic, x, y, gray((x+y)%(MAX+1))
```

Hemos usado la variable `MAX` arriba para representar el valor máximo del rango de la escala de grises. En la última línea, tomamos el restante `(x+y)%(MAX+1)` para asegurarnos de que el valor de gris estará dentro del rango 0..255. La imagen generada por el comando de arriba se muestra a la derecha. Puede parecer raro al principio que hemos usado las ubicaciones x- e y- de un píxel para computar sus valores dentro de la escala de grises o su brillo. Dado que `x-` e `y-` son números y como el valor de gris en sí es un número está bien hacer esto. Al reconocer que los valores `x-` e `y-` (y su suma) pueden ser más grandes que el rango del espectro de la escala de grises, tomamos el recordatorio de envolver los valores.

**Realizar la siguiente tarea:** Escriban un programa completo e intenten esto. Una vez que tienen el programa, pruébenlo en una imagen de un tamaño de 123x123 (¿por qué?). Luego, de nuevo pruébenlo en una imagen mucho más grande, digamos de 500x500.

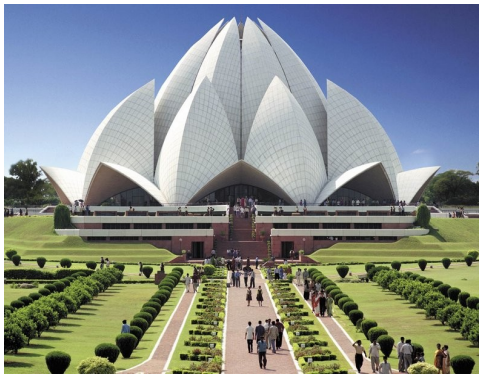
El rango de grises básicamente representa los niveles de brillo. Podemos divertirnos más si cambiamos el rango, digamos a 0..50. Entonces podemos modificar la función de gris como se muestra abajo para mapear proporcionalmente cualquier valor dentro del rango 0..50 al rango 0..255:

```
MAX = 50
def gray(v):
   # returns an rgb gray color for v
   brightness = v*255/MAX
   return makeColor(brightness, brightness, brightness)
```

**Realizar la siguiente actividad:** Reescribir el programa para usar la definición de arriba de `gray` y crear una imagen de 500x500. Modifiquen el valor de MAX a 25 y generen una imagen de 100x100 (mostrada arriba). Observen que si establecen a MAX en 255, se revertirá a la versión previa de la función.

**Realizar la siguiente actividad:** A continuación utilicen la regla:

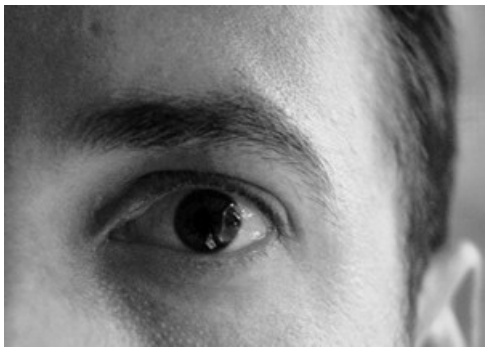
*pixel at x, y is colored using the grayscale x*y (píxel en x se colorea usando la escala de grises x*y)*



Recuerden tomar lo restante como arriba. Cambien `MAX` a 250 y generen una imagen de 500x500. Prueben otros tamaños y valores de `MAX`.

Pueden experimentar con otras reglas, transformaciones geométricas y funciones trigonométricas. Vean los ejercicios al final de este capítulo para más sugerencias.

## Procesamiento de imágenes



Uno no siempre debe comenzar con una imagen en blanco para crear una nueva. Pueden tomar imágenes ya existentes y transformarlas de formas interesantes. Los principios básicos siguen siendo los mismos: acceden a un píxel individual y a su valor de color y lo transforman de la manera que quieran. En esta sección haremos algunas transformaciones de imágenes simples para ilustrarlo. Todas las transformaciones de imágenes que aprenderemos requerirán de la misma transformación en todos los valores de píxeles de una imagen dada. También pueden, por supuesto, optar por seleccionar regiones específicas si lo desean.

Usaremos las imágenes que se muestran aquí para aprender acerca de la transformación de imágenes. Siéntanse libres de seleccionar una o más imágenes de su propia cámara digital. En la mayoría de los ejemplos hemos usado transformaciones en imágenes en escala de gris. Siéntanse libres de usar imágenes en color para experimentar con ellas. También notarán que cuanto más grande sea el tamaño de una imagen, más tardará en transformarse. Esta es una consecuencia directa de la cantidad de cómputos que deben hacerse. Por ejemplo, para transformar una imagen de 500x500 estarán operando 250,000 píxeles. Si cada transformación de píxel involucra 10 operaciones, ¡estarán especificando 25 millones de operaciones! Lo que esto demore dependerá de la computadora que estén usando. Volveremos sobre esto con más detalle más adelante en el capítulo. Por ahora, pruebe restringirse a imágenes pequeñas, digamos que no más grandes que 300x300 píxeles.

Como mencionamos antes, aún las cámaras digitales más rudimentarias en estos días proveen varios megapíxeles de resolución de imagen. Por lo tanto, lo primero que debemos aprender es cómo achicar una imagen dada a un tamaño más chico.

## Achicar & Agrandar

Imagen original (300x213)

Luego de achicarla (150x106, F=2)

Escribamos un programa que tome una imagen y la achique por un factor, digamos, F. Por ejemplo, si la imagen original es de 3000x3000 píxeles y la achicamos por un factor de 10, terminaríamos con una imagen de 300x300 píxeles. Esta transformación se logra seleccionando el valor de cada píxel de la nueva imagen basado en la imagen vieja. Podemos usar la siguiente regla para nuestra transformación:

*New pixel at x, y is a copy of the old pixel at x*F, y*F (Nuevo píxel en x, y es una copia del viejo píxel en x*F, y*F*

El programa de abajo le pide al usuario que elija una imagen, la muestre y luego le pide que especifique el factor por el cual quiere achicarla (que debe ser más grande que 1).

```
def main():
   # read an image and display it
   oldPic = makePicture(pickAFile())
   show(myPic, "Before")

   X = getWidth(oldPic)
   Y = getHeight(oldPic)
   # Input the shrink factor and computer size of new image
   F = int(ask("Enter the shrink factor."))
   newx = X/F
   newy = Y/F
   # create the new image
   newPic = makePicture(newx, newy)
   for x in range(1,newx):
      for y in range(1,newy):
         setPixel(newPic, x, y, getPixel(myPic, x*F, y*F))
   show(newPic, "After")
```

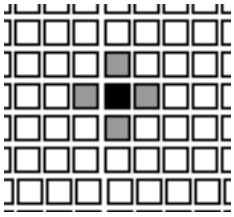
**Realizar la siguiente actividad:** Seleccionar una o más imágenes y asegúrense de que estén en formato JPEG y de que sean imágenes de alta resolución (para nuestro propósito, de  400x400 píxeles o más). Hagan correr el programa sobre cada imagen y observen el output por distintos valores de F. Modifiquen el

programa para usar el comando `savePicture` para guardar las imágenes en un archivo para usar en los ejemplos de abajo.

El agrandamiento se realiza de manera complementaria usando la regla:

*New pixel at x, y is a copy of the old pixel at x/F, y/F (el nuevo píxel en x, y es una copia del píxel viejo en x/F, y/F)*

El tamaño de la nueva imagen será F veces el tamaño de la imagen original. Cuando agrandan una imagen de una más pequeña, en realidad están generando datos donde no los había. Por lo tanto, es probable que la imagen resultante sea más borrosa que la original. Esto es similar a los reproductores de DVD que proclaman convertir una película a una definición más alta que la de la señal estándar. Si observan detenidamente, podrán ver defectos en las imágenes convertidas.



Vecindad de 4 píxeles

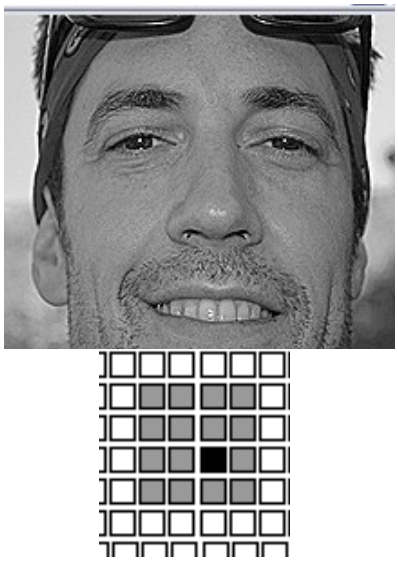
Imagen borrosa

## Borroso & Nítido

Pueden hacer más borrosa una imagen tomando el promedio de los valores de color de los píxeles vecinos. El tamaño de la vecindad determina cuán borrosa es la imagen. Abajo, mostramos cómo hacer borroso cada píxel usando una vecindad de 4 (ver imagen arriba):

```
n1 = getPixel(oldPic, x, y)
n2 = getPixel(oldPic, x, y-1)
n3 = getPixel(oldPic, x-1, y)
n4 = getPixel(oldPic, x+1, y)
n5 = getPixel(oldPic, x, y+1)
v = (getRed(n1)+...+getRed(n5))/5
setPixel(newPic, x, y, gray(v))
```

Observen arriba que `n1` es el píxel original. Pueden usar diferentes tamaños y formas de vecindades para crear efectos borrosos más elaborados. La siguiente página muestra los efectos de usar una vecindad más grande, y aún cuadrada.



vecindad de 15-píxeles

En lugar de promediar los valores de los píxeles, si le restan los valores de píxeles terminan haciendo más nítida la imagen. Es decir,

```
v = 5*getRed(n1) -
      (getRed(n2)+…+getRed(n5))
setPixel(newPic, x, y, gray(v))
```

Imagen borrosa

**Realizar la siguiente actividad:** Escriban un programa completo que use la fórmula para promediar la vecindad para hacer más borrosas y nítidas las imágenes. Dependiendo del tamaño y la forma de la vecindad que elijan, los rangos del índice del loop variarán. Por ejemplo, el loop para la vecindad de 4- píxeles aparecerá de la siguiente manera:

```
for x in range(1,X-1):
   for y in range(1,Y-1):
      # Average pixel values...
```

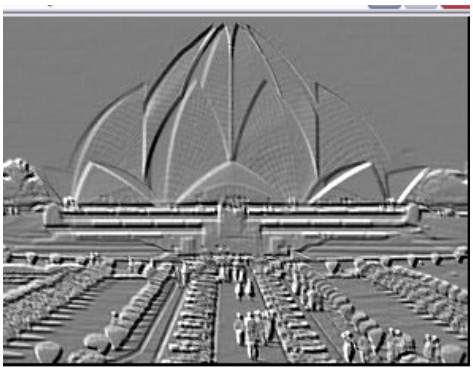
A medida que la vecindad crece, tendrán que ajustar los valores de iniciar y detener de las variables de x e y en el loop de arriba.

## Negativo & Repujado

La creación del negativo de una imagen se obtiene invirtiendo los valores de los píxeles. Para valores en escala de grises esto equivale a restar de 255. Para píxeles de color deben restar de 255 para cada uno de los valores RGB. La regla para escala de grises es:

*New pixel at x,y is 255-value of old pixel at x,y (Nuevo píxel enx, y es 255- valor del viejo píxel en x, y)*

Negativo



Repujado

Pueden especificar la transformación completa usando el loop:

```
for x in range(X):
    for y in range(Y):
        pixel = getPixel(oldPic, x, y)
        v = MAX – getRed(pixel)
        setPixel(newPic, x, y, gray(v))
```

De manera similar, pueden crear un efecto de repujado o de relieve restando del valor actual de los píxeles, el valor (o una fracción del mismo) de un píxel a dos píxeles de distancia:

```
pixel1 = getPixel(oldPic, x, y)
pixel2 = getPixel(oldPic, x+2, y+2)
v = getRed(pixel1) – getRed(pixel2)
setPixel(newPic, x, y, gray(v))
```

o

```
v = getRed(pixel1) + (MAX/2 – getRed(pixel2))
```

**Realizar la siguiente actividad:** Implementen las dos reglas mostradas arriba. Prueben crear un negativo de una imagen color, así como el de una imagen en escala de grises. Prueben distintas fracciones para el repujado para ver el efecto.

Seguramente, mientras leían y experimentaban con las técnicas de arriba, se preguntaban sobre el tipo de procesamiento de imágenes realizado por algunos de los programas de software populares de edición de fotos. Quizás ya estén familiarizados o han escuchado de Adobe Photoshop, Fireworks, GIMP, etc. La mayoría de estos programas usan técnicas similares a las que hemos descrito arriba y proveen más facilidades y opciones para la manipulación de imágenes.

## Comprensión de Imágenes & Visión del Robot

La imagen que se muestra a la derecha fue tomada de un artículo de diario de un diario local en el oeste de New York. Lean el texto que la acompaña y vean si pueden identificar a cada individuo en la foto. Luego de leer el texto, sin duda, confirmarán que la persona en el centro de la fila de adelante es el DT Kevin Starr. Sería muy difícil identificar a las personas si no estuvieran presentes los textos. De todas maneras, podrían responder a preguntas como cuántas personas hay en la foto. Este es un ejemplo del problema de *comprensión de imagen*: Dada una imagen, cómo pueden determinar, computacionalmente, que hay siete personas en la imagen, que tres de ellos están arrodillados y cuatro parados, etc. Tampoco tendrían problemas para dibujar una caja en el rostro de cada persona en la imagen. En esta sección les introducimos algunas técnicas básicas para comprensión de imágenes y para usarlas para la creación de un sistema de percepción visual para el robot Scribbler. Ya están familiarizados con los comandos básicos de procesamiento de imágenes. Los mismos comandos serán usados para la comprensión de imágenes.

"LOS MEJORES – El equipo de volley de los All-WNY rodea al DT del año, Kevin Starr de Frontier. En la hilera superior, desde la izquierda: Brian Heffernan de Eden, Tom Kait de Lake Shore, Greg Gegenfurtner de Sweet Home yGeorge Beiter de Kenmore West. Fila de abajo: desde la izquierda, están David Nagowski de Eden y Rich Bland de Orchard Park."

Para que un robot perciba su entorno visualmente debe comenzar por tomar una foto. Una vez obtenida la imagen, la percepción de los contenidos de la imagen se realiza computacionalmente. En lugar de tomar la tarea de comprensión de imagen mostrada arriba,

comenzaremos de manera sencilla: ¿cómo reconoce una pelota el Scribbler? Una vez que la reconoce, ¿puede seguirla a donde sea que vaya?

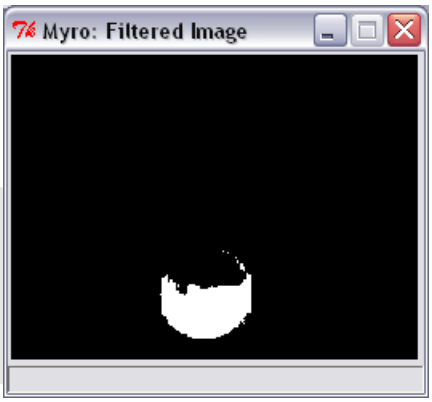
El segundo problema se hace sencillo una vez que sabemos como reconocer una pelota. Podemos escribir un programa de control simple así:

```
while timeTemaining(T):
    # take picture and locate the ball
    …
    if <ball is on the left of visual field>:
        turnLeft(cruiseSpeed)
    elif <ball is in the middle of visual field>:
        forward(cruiseSpeed)
    else:
        turnRight(cruiseSpeed)
```

El principal problema es localizar la pelota en la imagen. Dados los comandos de manipulación de imagen que aprendieron hasta ahora, piensen cómo se podría lograr esto.

Los principios básicos en la comprensión de imágenes se basan en comenzar en el nivel del píxel. Hagamos que las cosas sean un poco más concretas. Miren la imagen que se muestra arriba. Fue tomada por un robot Scribbler.

Para nuestros propósitos hemos exagerado la escena al elegir una pelota de color brillante (rosa). Cuando hagan click en el mouse en cualquier parte de la imagen les dará los valores RGB de ese píxel. La foto muestra los valores de un píxel en el centro de la pelota. Observen que el píxel rojo es 253 y el valor del píxel verde es de 66. Dicho sea de paso, el verdadero rosa tiene valores RGB (255,175,175). Podemos usar esta información como forma de identificar la pelota. Aquí hay un loop de transformación de imagen simple que convierte todos los píxeles en la imagen en píxeles blancos o negros. Si un píxel tiene un alto valor de rojo y un valor de verde más bajo, será convertido al blanco, si no, en negro.




```
for pixel in getPixels(p):
   r, g, b = getRGB(pixel)
   if r > 200 and g < 100:
      setRGB(pixel, (255, 255, 255))
   else:
      setRGB(pixel, (0, 0, 0))
```

Aquí se muestra la imagen resultante. Como pueden ver, con solo seleccionar los valores apropiados de colores individuales pueden filtrar las regiones indeseadas y focalizar únicamente en los colores que les interesan. Aunque la pelota entera no ha sido filtrada, es suficiente para usar la imagen para identificar su ubicación. Al ajustar los valores de umbral de RGB pueden refinar más este filtro. Se requerirá más procesamiento para identificar la ubicación de la pelota en el campo visual (que es la imagen).

Estamos interesados en localizar la pelota a la izquierda, derecha o en el centro de la imagen. Una forma de lograrlo es ubicar la coordenada x- en el punto central de la pelota. Una forma de hacerlo es tomar el promedio de las ubicaciones x- de todos los píxeles blancos de la imagen:

```
def locateBall(picture):
    tot_x = 0
    count = 0
    for pixel in getPixels(p):
        r, g, b = getRGB(pixel)
        if r > 200 and g < 100:
            tot_x = tot_x + getX(pixel)
            count = count + 1
    return tot_x/count
```

De forma alternativa, también pueden promediar las ubicaciones x- de todos los píxeles rosas sin usar el filtro. En la imagen mostrada, el promedio de valor de x- devuelto era 123. Dado que la imagen tiene 256 píxeles de ancho, 123 es casi justo en el centro. Abajo, mostramos las ubicaciones obtenidas en cada una de las imágenes mostradas.

ubicación = 41

ubicación = 123

ubicación = 215

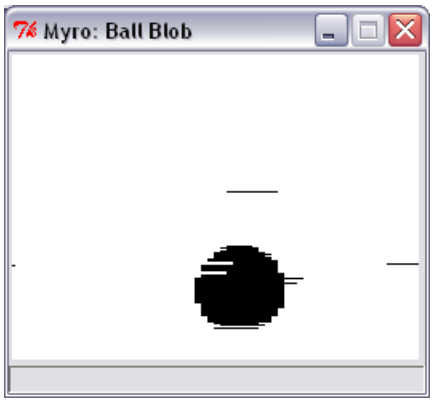
Dado que el ancho de la imagen es 256 píxeles, pueden dividir el campo visual en tres regiones: izquierda (0..85), centro (86..170), y derecha (171, 255). El control del robot para seguir a la pelota puede definirse como se muestra abajo:

```
while timeTemaining(T):
    # take picture and locate the ball
    pic = takePicture()
    ballLocation = locateBall(pic)

    if ballLocation <= 85:
        turnLeft(cruiseSpeed)
    elif ballLocation <= 170:
        forward(cruiseSpeed)
    else:
        turnRight(cruiseSpeed)
```

**Realizar la siguiente actividad:** Implementar el programa de arriba para reconocer un objeto dado en la foto. El objeto puede ser una caja, una pelota, un cesto de basura, etc. Experimenten con distintos objetos de color. ¿Pueden usar estas técnicas para reconocer cualquier objeto? ¿digamos un animal? ¿una persona? etc.

## Blobs


Estableciendo un blob en la pelota.




Resultado de `takePicture("blob")`

El tipo de filtro de umbral usado arriba es comúnmente llamado *blob filtering (filtrado blob)*. Todos los píxeles que satisfacen las propiedades del color seleccionado se identifican y el resto se elimina. Esta es una operación tan común en el procesamiento de imágenes que muchas cámaras y hardware de cámaras integran el filtrado blob como primitivo. El Scribbler también tiene esa capacidad. Usando esto, no necesitan preocuparse por identificar valores de color específicos para el umbral de filtro. Todo lo que deben hacer es tomar la foto y mostrarla. Luego, usando el mouse, definir la región rectangular que contiene su blob deseado. Hagan esto, saquen una foto, visualícenla y luego hagan click en el mouse en la esquina superior izquierda de la pelota en la imagen y arrástrenla a la esquina inferior derecha antes de soltarla. Deben asegurarse de que estén conectados al robot, porque la región blob seleccionada se transfiere de vuelta al robot. Una vez que esto esté definido, podrán usar el filtro blob predefinido sacando fotos y usando el comando:

```
pic = takePicture("blob")
```

Ni bien la foto es tomada, el detector blob de la cámara entra en funcionamiento y devuelve la imagen resultante (mostrada arriba). Ahora pueden usar esta imagen como antes para controlar el robot. Tendrán un promedio de la ubicación de todos los píxeles negros.

Si no, pueden usar la función `getBlob()` de la siguiente manera:

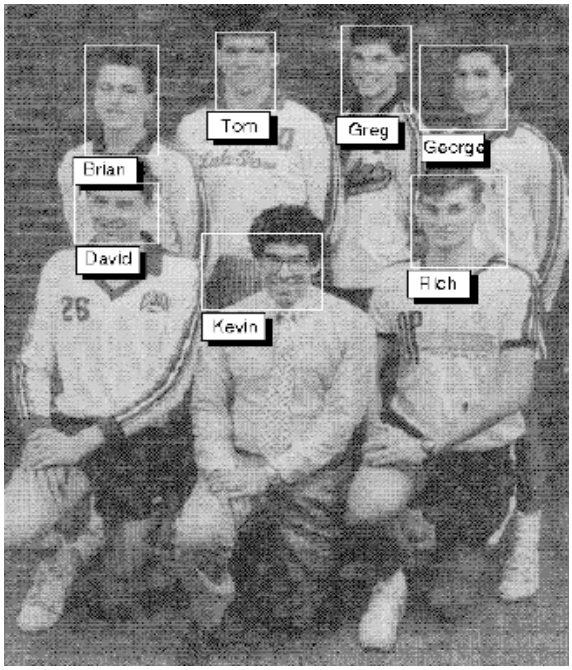
```
onPixels, avgX, avgY = getBlob()
```

La función devuelve dos valores: El número de los píxeles “encendidos” (por ejemplo, los píxeles en la imagen que estaban en el blob detectado), y la ubicación promedio x- e y- de los píxeles blob. Esta es una forma mucho mejor y más rápida de realizar reconocimiento de blob y notarán que el robot rastrea la pelota mucho mejor.

**Realizar la siguiente actividad:** Rehacer el ejemplo anterior de la pelota y aplique las características de blob itroducidas aquí.

Cálculos futuros sobre una imagen pueden resultar en la identificación de bordes de objetos. Esto se llama ***detección de borde***. Estos pueden ser usados para hacer reconocimiento de objetos: ¿Hay un auto en la foto? ¿Un rostro? etc. El

reconocimiento de objetos requiere de la habilidad para identificar características que son típicas de un objeto. El dominio del reconocimiento de imágenes y visión computacional es rica en técnicas y aplicaciones como éstas. A la derecha, mostramos los resultados de reconocer rostros e identificar individuos en la foto. A medida que los investigadores avanzan en estas técnicas y a medida que se incrementan las capacidades computacionales de las cámaras digitales, estamos comenzando a ver muchas de estas incorporadas incluso a las cámaras digitales más básicas. Tales técnicas pueden ser usadas para focalizar automáticamente en el sujeto principal y también para ajustar la exposición para obtener una foto perfecta, aún para fotógrafos amateurs.




## Resumen

En este capítulo, han visto varias facilidades Myro que pueden usar usadas para la creación, el procesamiento y la comprensión de imágenes. La mayoría de las técnicas de creación, procesamiento y comprensión de imágenes involucran la manipulación de la imagen en el nivel del píxel. Sin embargo, muchos efectos y aplicaciones interesantes pueden ser desarrollados usando estas ideas simples. Muchas de estas características se encuentran cada vez más en dispositivos cotidianos, como cámaras digitales, dispositivos de identificación biométrica, y aun sistemas de distribución automática de correo. Combinadas con comportamientos del robot estas facilidades pueden ser usadas para modelar comportamientos más inteligentes.

## Revisión Myro

`getHeight(<picture>)`
`getWidth(<picture>)`
Devuelve el alto y el ancho del objeto `<picture>` (en píxeles).

`getPixel(<picture>, x, y)`
Devuelve el pixel en `x,y` en `<picture>`.

`getPixels(<picture>)`
Cuando es usado en un loop, devuelve un píxel a la vez de `<picture>`.

`getRGB(pixel)`
`getRed(<pixel>)`
`getGreen(<pixel>)`

`getBlue(<pixel>)`
Devuelve los valores RGB del `<pixel>`.

`makeColor(<red>, <green>, <blue>)`
Crea un objeto color con los valores `<red>`, `<green>`, y `<blue>` dados (todos están en el rango [0..255]).

`makePicture(<file>)`
`makePicture(<width>, <height>)`
`makePicture(<width>, <height>, <color>)`
Crea un objeto imagen ya sea leyendo una imagen de un `<file>`, o del `<width>` y `<height>` dados. Si no se especifica `<color>`, la imagen creada tiene un fondo blanco.

`pickAColor()`
Crea una ventana de diálogo interactiva para seleccionar visualmente un color. Devuelve el objeto color correspondiente al color seleccionado.

`pickAFile()`
Crea una ventana de diálogo interactiva que le permite al usuario navegar a una carpeta y seleccionar un archivo para abrir. Nota: No puede usado para crear archivos nuevos.

`repaint()`
`repaint(<picture>)`
Actualiza la <picture> mostrada.

`savePicture(<picture>, <file>)`
`savePicture(<picture list>, <gif file>)`
Guarda `<picture>` en el arvhico especificado (un GIF o JPEG como determine la extensión del `<file>`: `.gif` o `.jpg`). `<picture list>` se guarda en un archivo GIF animado.

`setColor(<pixel>, <color>)`
`setRed(<pixel>, <value>)`
`setGreen(<pixel), <value>)`
`setBlue(<Pixel>, <value>)`
Establece el color de `<pixel>` al `<color>` o `<value>` especificados.

`show(<picture>)`
`show(<picture>, <name>)`
Muestra la `<picture>` en la pantalla en una ventana llamada `<name>` (string).

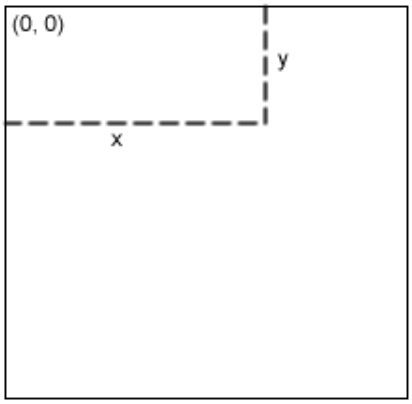
`takePicture()`
`takePicture("gray")`
`takePicture("blob")`
Saca una foto con la cámara del Scribbler. Por defecto es una foto color, o en escala de grises (`"gray"`), o una imagen filtrada basada en el blob (`"blob"`). Ver texto del capítulo para ejemplos.

## Revisión Python

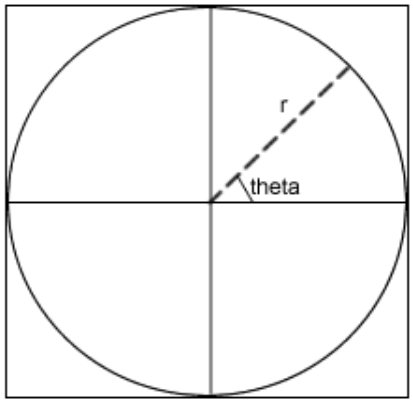
No se introdujeron características de Python en este capítulo.

## Ejercicios

**1**. En lugar de visualizar cada píxel en una imagen como valores x- e y- en el sistema cartesiano de coordenadas, imaginen que está en un sistema de coordenadas polar con su origen en el centro de la imagen (Ver la imagen abajo).



Cartesian Coordinates



Polar Coordinates

Escriban una función llamada polar(x, y) que devuelve las coordenadas polares (r, theta) de un punto dado. Usando esta función escriban una transformación de la siguiente manera:

```
for x in range (1, W):
   for y in range(1, H):
      r, theta = polar(x, y)
      setPixel(myPic, x, y,
               gray((((r+theta)%16)-8)*MAX/16+MAX/2))
```

Crear una imagen de 400x400 usando la transformación de arriba. Comiencen con una imagen negra.

**2.** Basados en el ejercicio de arriba, prueben transformar una imagen (como un retrato). Por ejemplo, usen la regla: nuevo x, y es una copia del $\sqrt{r * \frac{W}{2}}$, *theta* de la imagen original.

**3.** Escriban una transformación que gire la imagen en el sentido de las agujas del reloj (o contra las agujas del reloj) en 90 grados. Pruebenlo en una imagen cuadrada a elección.

**4.** Escriban una transformación de imagen que promedie dos imágenes (del mismo tamaño) en una sola imagen.

**5.** Escriban un programa que una dos o más imágenes juntas para crear un panorama.

**6.** Usen las técnicas de detección de blob descriptas en este capítulo para reconocer latas de Coca Cola y Pepsi. Escriban un programa para salir a la caza de una Coca o Pepsi.

**7.** Pueden expandir el ejercicio de arriba para reconocer pequeñas figuritas de juguete de personajes de dibujos animados de Disney como Mickey, Minnie, Daisy, Goofy, Pluto, y Donald?

# 10 Inteligencia artificial


**David:** Martin es el verdadero hijo de Mami y de Henry. Después de que encuentre al Hada Azul podré ir a casa. Mami va a amar a un nene de verdad. El Hada Azul me transformará en uno.
**Gigolo Joe:** El Hada Azul es Mecha, Orga, hombre o mujer?
**David:** Mujer.
**Gigolo Joe:** ¿Mujer? Yo conozco a las mujeres! A veces me piden por mi nombre. Conozco todo acerca de las mujeres. Casi todo cuanto haya que saber. No hay dos iguales, y después de que me conocen, no hay dos iguales. Y sé dónde se puede encontrar a la mayoría de ellas.
**David:** ¿Dónde?
**Gigolo Joe:** Rouge City. Cruzando el Delaware.

Diálogo entre dos entidades de inteligencia artificial: Gigolo Joe (interpretado por by Jude Law) y David (interpretado por Haley Joel Osment) en la película, *Inteligencia Artificial*, Dirigida por Steven Speilberg, Warner Bros., 2001.

Página opuesta: I.A. Inteligencia Artificial

Del póster de la película. Warner Bros., 2001.

## La cuestión de la inteligencia

El intento de comprender la inteligencia humana probablemente es uno de los interrogantes humanos más viejos que, sin embargo, aún queda por responder en su totalidad. Con el advenimiento de las computadoras y los robots, la cuestión de si los robots y las computadoras pueden ser tan inteligentes como los humanos ha llevado a las búsquedas científicas en el campo de la *Inteligencia Artificial (IA)*. La pregunta de si una computadora puede ser inteligente fue discutida lúcidamente por el Profesor Alan Turing en 1950. Para ilustrar las cuestiones subyacentes a la inteligencia de las máquinas, Turing ideó un experimento en forma de *juego de imitación*. Se juega con tres personas, un hombre, una mujer y un interrogador. Todos están en habitaciones separadas e interactúan entre ellos tipeando un texto en la computadora (bastante parecido al modo en que la gente interactúa entre sí con dispositivos de mensajería instantánea). La tarea del interrogador es identificar cuál de las personas es el hombre (o la mujer). Para que el juego resulte interesante, cualquiera de los jugadores puede intentar ser engañoso al dar sus respuestas. Turing argumenta que una computadora debería ser considerada inteligente si se la pudiera hacer jugar el rol de cualquiera de los jugadores sin darse a conocer. Este *test* de inteligencia se ha llamado el *Turing Test* y ha generado mucha actividad en la comunidad de investigadores de IA (ver Ejercicios). El diálogo que se muestra arriba, de la película *Inteligencia Artificial*, describe un aspecto del test de inteligencia diseñado por Alan Turing, basado en el intercambio entre Gigolo Joe y David, ¿pueden concluir que ambos son inteligentes? ¿Humanos?

Luego de más de cinco décadas de investigación en IA, el campo ha madurado y evolucionado de muchas maneras. En primer lugar, el foco sobre la inteligencia no se limita ya a los humanos: los insectos y otras formas de animales con variadas formas de inteligencia han sido objeto de estudio dentro de la IA. También ha habido un intercambio enriquecedor de ideas y modelos entre científicos, biólogos, psicólogos, científicos cognitivos, neurocientíficos, lingüistas y filósofos de la IA.

Han visto ejemplos de estas influencias en los modelos de los vehículos de Braitenberg introducidos anteriormente. Dada la diversidad de investigadores involucrados en la IA, también ha habido una evolución acerca de qué se trataba la IA. Volveremos sobre esto más adelante en el capítulo. Primero, les daremos algunos ejemplos de modelos que pueden ser considerados inteligentes y que son comúnmente usados por científicos de IA.

## Comprensión de Lenguaje

Un aspecto de la inteligencia reconocido por muchas personas es el uso del lenguaje. Las personas se comunican entre sí usando un lenguaje. Hay muchos (varios miles) de lenguajes en uso en este planeta. Tales lenguajes son llamados *lenguajes naturales*. Muchas teorías interesantes se han enunciado sobre los orígenes del lenguaje en sí mismo. Una cuestión interesante a considerar es: ¿puede la gente comunicarse con las computadoras usando lenguajes humanos (naturales)? En otras palabras, ¿puede hacerse una computadora para que comprenda un lenguaje? Considérenlo por un momento.

Para que la cuestión de la comprensión del lenguaje resulte más concreta, piensen en su robot Scribbler. Hasta ahora, han controlado el comportamiento del robot escribiendo programas Python para el mismo. ¿Es posible hacer que el Scribbler comprenda nuestro lenguaje (castellano o inglés) de modo tal que puedan interactuar con él? ¿Cómo sería una interacción con el Scribbler? Obviamente, no esperarían tener una conversación con el Scribbler acerca de la cena que comieron anoche. Sin embargo, probablemente tendría sentido pedirle que se mueva en cierta dirección. O preguntarle si está viendo un obstáculo frente a él.

**Realizar la siguiente actividad:** Escriban una serie de comandos breves de una palabra como: `forward`, `right`, `left`, `stop`, (adelante, derecha, izquierda, detenerse) etc. Creen un vocabulario de comandos y luego escriban un programa que ingrese un comando a la vez, lo interprete y haga que el Scribbler lo lleve a cabo. Por ejemplo:

```
Ustedes: forward
Scribbler: comienza a moverse hacia adelante
Ustedes: right
el Scribbler empieza a moverse hacia la derecha
Ustedes: stop
…
```

Experimenten con el comportamiento del robot basándose en estos comandos y reflexionen acerca de la interpretación adecuada que pueda hacer que este comportamiento sea más natural.

Se encontrarán haciendo varias hipótesis acerca de la interpretación de incluso los comandos más simples en el ejercicio de arriba. Por ejemplo, ¿qué sucede cuando después de que le ordenan al Scribbler que se mueva hacia delante, le piden que se mueva a la derecha? ¿El Scribbler debería dejar de avanzar o debería detenerse y luego doblar?

Las decisiones como éstas también nos dan una profunda conciencia sobre nuestras propias habilidades para comprender el lenguaje. También pueden ver que, como en el caso de la percepción visual, el procesamiento del lenguaje (o de un texto) comienza en un nivel muy primitivo: las palabras. Si la entrada (input) es el discurso, las unidades básicas son las señales eléctricas, quizás provenientes de un micrófono. Al igual que el procesar píxeles individuales para tratar de comprender los contenidos de una imagen, uno debe comenzar en un nivel bajo de representación para comenzar a comprender el lenguaje.

Los investigadores que trabajan en el campo de la *lingüística computacional* (o la *comprensión del lenguaje natural*) han propuesto muchas teorías acerca del procesamiento del lenguaje que forman la base de un modelo computacional para que el Scribbler comprenda un pequeño subconjunto del lenguaje inglés[1]. En esta sección examinaremos un modelo de éstos que está basado en el procesamiento de la sintaxis y la semántica de la interacción del lenguaje. Imagínense interactuando con el Scribbler utilizando el siguiente grupo de sentencias:

```
Ustedes: ¿Viste una pared?
```

1 Tengan en cuenta que en este libro se basa en una experiencia en un instituto de estados Unidos y, por lo tanto, en estas primeras experiencias se trabaja con el idioma inglés.

Scribbler: No

Ustedes: Hacé un Bip cada vez que veas una pared.
Ustedes: Doblá a la derecha cada vez que veas una pared a tu izquierda.
Ustedes: Doblá a la izquierda cada vez que veas una pared a tu derecha.
Ustedes: Movete durante 60 segundos.

[El robot Scribbler se mueve alrededor durante 60 seundos doblando cada vez que ve una pared. También hace un bip cada vez que ve una pared.]

Anteriormente, han escrito programas Python que llevan a cabo comportamientos similares. Sin embargo, ahora imagínense interactuando con el robot de la manera descripta. Desde una perspectiva física, imagínense que están sentados frente a la computadora y tienen una conexión Bluetooth con el robot. La primera pregunta que aparece es: ¿Están diciendo o tipeando los comandos de arriba? Desde la perspectiva de la IA, ambas modalidades son posibles: Podrían estar sentados frente a la computadora hablando a un micrófono; o podrían estar tipeando esos comandos en el teclado.

En primera instancia, necesitarían una capacidad de comprensión del lenguaje. Actualmente, pueden obtener software (comercial y *freeware*) que les permitirá hacerlo. Algunos de estos sistemas son capaces de distinguir acentos, entonaciones, voces femeninas o masculinas, etc. Efectivamente, la comprensión del lenguaje hablado es un campo fascinante de estudio que combina conocimientos de lingüística, procesamiento de señales, fonología, etc.

Pueden imaginar que el resultado de hablar a una computadora es un fragmento de texto que transcribe lo que han dicho. Por lo tanto, la pregunta que se le planteó al Scribbler arriba: *¿Ves una pared?* Deberá ser procesada y transcripta a un texto. Una vez que tengan el texto, es decir, un string “*¿Ves una pared?*” o en inglés: “`Do you see a wall?`”, éste puede ser posteriormente procesado o analizado para comprender el *significado* o el contenido del texto. El campo de la *lingüística computacional* provee muchas formas de análisis sintáctico y de extracción de significado de textos. Los investigadores en IA han desarrollado formas de representar el conocimiento en una computadora usando anotaciones simbólicas (por ej., la *lógica formal*). Al final, el análisis del texto resultará en un comando `getIR()` o `getObstacle()` para el robot Scribbler y producirá la respuesta mostrada arriba.

Nuestra meta de traer el escenario de arriba aquí es para presentarles varias dimensiones de la investigación en IA que pueden involucrar a personas de distintas disciplinas. En estos días, es completamente posible, incluso para ustedes, diseñar y construir programas de computación o sistemas capaces de interactuar con robots usando el lenguaje.

## Juegos

En la historia temprana de la IA, varios científicos plantearon varias tareas desafiantes que, si son realizadas por computadoras, pueden ser usadas como forma de demostrar la viabilidad de la inteligencia de las máquinas. Era una práctica común pensar en los juegos en este campo. Por ejemplo, si una computadora pudiera jugar un juego como el ajedrez o las damas en el mismo nivel

o mejor que los humanos, podríamos convencernos de que es factible pensar en una computadora como un posible candidato para la inteligencia de la máquina. Algunas de las demostraciones iniciales de la investigación en IA incluyeron pruebas de modelos de computadoras para jugar varios juegos. El ajedrez y las damas parecían ser las elecciones más populares, pero los investigadores se han dado el gusto de examinar modelos de computadoras para muchos juegos populares: Poker, Bridge, Scrabble, Backgammon, etc.



En muchos juegos, ahora es posible que los modelos de computadoras jueguen en los más altos niveles de la práctica humana. En el ajedrez, por ejemplo, aunque los primeros programas fácilmente vencían a lo novatos en los '60, no fue sino hasta 1996 que un programa de ajedrez de una computadora IBM llamada Deep Purple venció al campeón mundial Gary Kasparov en un juego a nivel competición, aunque logró ganar la partida 4-2. Un año después, Deep Blue le ganó a Kasparov en un juego de 6 partidas, era la primera vez que una computadora vencía al mejor jugador humano en un juego de ajedrez de estilo clásico. Mientras que estos logros son dignos de halagos, también es claro ahora que la búsqueda de la inteligencia artificial no se responde necesariamente con juegos de computadora. Esto ha tenido como resultado un gran progreso en los sistemas de juegos de computadora y en la tecnología de juegos que ahora es una industria multi-millonaria.

Resulta que en muchos juegos tipo ajedrez, la estrategia para que juegue una computadora es muy similar. Tales juegos se clasifican como juegos-de-dos personas-suma-cero: dos personas/computadoras juegan como oponentes y el resultado del juego es que un jugador gana  y el otro pierde, o es un empate. En muchos juegos de este tipo, la estrategia básica para dar el siguiente paso es simple: ver todos los posibles movimientos que tengo, y para cada uno, los posibles movimientos del otro jugador y así hasta el final. Entonces, rastrear hacia atrás desde la instancia de la victoria (o el empate) y hacer el siguiente movimiento basado en los resultados deseables. Pueden ver esto aún en juegos simples como Ta Te Ti, donde es fácil ver mentalmente hacia delante futuras movidas y luego tomar decisiones informadas sobre qué hacer a continuación. La mejor manera de comprender esto es realizando un programa que juegue de esta manera.

## Ta Te Ti

También conocido como *Nudos y Cruces* (*Noughts and Crosses)* o *Abrazos y Besos*, el Ta Te Ti es un juego de chicos bastante popular. Desarrollaremos un programa que pueda ser usado para jugar este juego contra una persona. Casi cualquier juego de mesa puede ser programado usando el loop (iteración) básico mostrado abajo:

```
def jugar():
    # Inicializar tablero
    tablero = inicializarTablero()

    # Determinar quién mueve: por ejemplo, X mueve siempre primero
    jugador = 'X'

    # Mostrar el tablero inicial
    mostrar(tablero)

    # El juego
    while (not gameOver(tablero)):
        mover(tablero, jugador)
        display(tablero)
        jugador = oponente(jugador)

    # fin del juego, mostrar resultados
    winningPiece = ganador(tablero)

    if winningPiece != 'Tie':
        print winningPiece, "won."
    else:
        print "It is a tie."
```

La función de arriba puede ser usada para jugar una ronda de cualquier juego de mesa de dos personas. La variable `jugador` es el jugador (o la pieza) cuyo movimiento es el siguiente. Ya estamos usando la pieza de Ta Te Ti “X” en la función de arriba. Seis funciones básicas (resaltadas arriba) conforman los bloques de construcción básicos del juego. Para el Ta Te Ti, pueden ser definidos como:

1. **`InicializarTablero()`**: Devuelve un tablero nuevo que representa el inicio del juego. Para el Ta Te Ti, esta función devuelve un tablero vacío representando nueve cuadrados.

2. **`mostrar(tablero)`**: Muestra el tablero en pantalla  para que el usuario lo visualice. La visualización puede ser tan simple o elaborada como deseen. Es bueno comenzar con el que más fácilmente puedan escribir. Más adelante lo harán más elaborado.

3. **`oponente(jugador)`**: Devuelve el oponente del actual jugador/pieza. En el Ta Te Ti, si el jugador es X, devolverá un O, y viceversa.

4. **`mover(tablero, jugador)`**: Actualiza el tablero haciendo un movimiento para el jugador. Si el jugador es el usuario, pedirá el movimiento del usuario. Si el jugador es la computadora, decidirá cómo hacer el mejor movimiento. Aquí es donde entra en escena la inteligencia.

5. **`gameOver(tablero)`**: Devuelve `True` si no hay más movimientos por hacerse, de lo contrario, devolverá `False`.

6. **`ganador(tablero)`**: Examina el tablero y devuelve la pieza ganadora o que el juego aún no ha terminado, o que hay un empate. En el Ta Te Ti, devolverá una X, un O, un blanco (representando que el juego aún no ha terminado), o un EMPATE.

Necesitaremos algunas funciones más para completar el juego pero estas seis forman en núcleo básico. Las escribiremos primero.

El juego en sí mismo consiste de nueve cuadrados que empiezan vacíos. Los jugadores eligen una pieza de juego: una “X” o una “O”. Asumiremos que “X” siempre va primero. Lo primero que debe hacerse entonces es diseñar una representación del tablero de juego. Usaremos la siguiente representación simple:

```
tablero = [' ',' ',' ',' ',' ',' ',' ',' ',' ']
```

Esta es una lista de 9 strings de un carácter. Observen que estamos usando esta representación lineal en lugar de la de 2-dimensiones.

Sin embargo, como verán, esta representación hace que sea más fácil llevar a cabo muchas manipulaciones para el juego. Durante el juego, el tablero de juego puede ser visualizado en su formato natural. Abajo, mostramos dos funciones: una crea un tablero nuevo cada vez que se la convoca, y una la muestra:

```
def inicializarTablero():
    # Representamos un tablero de 3x3 como una lista de 9 elementos.
    # Usaremos la siguiente numeración para ubicar una casilla
    #  0 | 1 | 2
    # ---|---|---
    #  3 | 4 | 5
    # ---|---|---
    #  6 | 7 | 8

    return [' ',' ',' ',' ',' ',' ',' ',' ',' ']


def mostrar(tablero):
    for i in range(0, 9, 3):
       if i > 0:
           print '---|---|---'
       print " %c | %c | %c "%(tablero[i],tablero[i+1],tablero[i+2])
    print
```

Una ventaja de escribir la función de visualización como se muestra es que nos da una forma rápida de crear y mostrar el juego. Más adelante, cuando terminan, pueden escribir una versión más elegante que permite visualizar el juego gráficamente (ver los ejercicios). Con las funciones de arriba, fácilmente podemos crear un nuevo tablero y mostrarlo de la siguiente manera:

```
tablero = inicializarTablero()
mostrar(tablero)
```

Determinar el oponente de una pieza dada es suficientemente simple:

```
def oponente(jugador):
  if jugador == "X":
```

```
    return "O"
  else:
    return "X"
```

A continuación, tenemos que escribir la función `mover`. Primero determina a quién le toca jugar. Si es el turno del usuario, tomará la jugada del usuario. Si no, determinará la mejor jugada para la computadora. Luego, efectivamente realizará la jugada. Como primer diseño, dejaremos que `mover` haga una elección al azar entre los posibles movimientos para la computadora. Luego, tomaremos una decisión más informada. La función `choice` está disponible en el módulo de librería `random`:

```
from random import choice
Vos = 'X'
Yo = 'O'

def mover(tablero, jugador):

    if jugador == Vos:       # movimiento del usuario?
        casilla = input("Ingresa tu movida: ") - 1
    else:                   # mi turno
        # el jugador es la computadora, se toma una elección aleatoria
        casilla = choice(movimientosPosibles(tablero, jugador))

    # ubica la pieza del jugador en la casilla elegida
    aplicarMovimiento(tablero, jugador, casilla)
```

Hemos establecido las variables globales `Vos` y `Yo` para piezas específicas. Esta es una simplificación provisoria. Más adelante pueden volver y reescribirlas como para que en cada juego el usuario pueda seleccionar su pieza. En el Ta Te Ti, X siempre se mueve primero. Entonces, haciendo que el usuario tenga la pieza X, asumimos que X siempre va primero. Nuevamente, más adelante podemos regresar y modificar esto (ver los Ejercicios). También observen que no estamos chequeado errores en el input del usuario para asegurarnos de que se introdujo una movida legal (ver Ejercicios).

El usuario introduce su movimiento ingresando un número del 1..9 donde la numeración de cuadrantes es como se muestra abajo. Esto es ligeramente diferente de nuestra numeración interna de cuadrantes y resulta más natural para la gente. En la función de arriba, restamos 1 del número de input para que lo ubique en el cuadrado apropiado en nuestro esquema interno.

```
 1 | 2 | 3
---|---|---
 2 | 5 | 6
---|---|---
 7 | 8 | 9
```

Nuevamente, por ahora hemos simplificado la interfaz. Los ejercicios sugieren cómo mejorar este diseño.

La función `mover` definida arriba requiere dos funciones adicionales (se muestran resaltadas). Estas son también funciones básicas para cualquier juego basado en tablero y se describen abajo:

7. **movimientosPosibles(tablero, jugador)**: Devuelve una lista de posibles movimientos para el jugador dado.

8. ***aplicarMovimiento(tablero, jugador, casilla)***: Dado un cuadrado específico y un jugador/pieza, esta función aplica el movimiento en el tablero. En el Ta Te Ti, todo lo que uno debe hacer es ubicar la pieza en el cuadrado dado. En otros juegos, como el ajedrez o las damas, hay muchas piezas que pueden ser removidas.

En el Ta Te Ti todos los cuadrados vacíos son posibles lugares donde un jugador puede moverse. Abajo, escribimos una función en la cual, dado un tablero, devuelve un listado de todos los posibles lugares donde una pieza puede ser ubicada:

```
def movimientosPosibles(tablero):
    return [i for i in range(9) if tablero[i] == ' ']
```

Arriba estamos usando *comprensiones de listas (list comprehensions)* (una característica de Python) para crear la lista de posibles movimientos. Pueden estar más cómodos escribiendo la siguiente versión:

```
def movimientosPosibles(tablero):
  movimientos = []

  for i in range(9):
     if tablero[i] == ' ':
        movimientos.append(i)

  return  movimientos
```

Para completar el programa de juego tenemos que escribir dos funciones más definidas arriba. La función `ganador` examina el tablero y determina quién ganó. Devuelve la pieza ganadora (una `'X'`, `'O'`) o el string `'Tie'`. En el caso de que el juego aún no haya terminado, devuelve un `' '`. Abajo, primero definimos todas las posiciones ganadoras en el Ta Te Ti, basadas en nuestra representación del tablero. A continuación, definimos la función para examinar el tablero.

```
# Estas ternas de casillas representan los casos ganadores
GANADORES = [[0, 1, 2],[3, 4, 5],[6, 7, 8],      # filas
            [0, 3, 6],[1, 4, 7],[2, 5, 8],       # columnas
            [0, 4, 8],[2, 4, 6]]                 # diagonales

def ganador(tablero):
    for win in GANADORES:
        posWinner = tablero[win[0]]
        if (posWinner != ' ' and
            posWinner == tablero[win[1]] and
```

```
            posWinner == tablero[win[2]]):
            return posWinner

    # No hay ganador aún, quedan casillas vacías?
    for i in range(9):
        if tablero[i] == ' ':
            return ' '

    # El tablero está lleno y no hay tres en línea
    return 'Tie'
```

Por último, la función `gameOver` puede ser escrita ya sea basándose en el hecho de que el ganador devuelve un `' '` cuando el juego aún no ha terminado, o podemos escribirlo usando `movimientosPosibles()` de la siguiente manera:

```
def gameOver(tablero):
    return len(movimientosPosibles(tablero)) == 0
```

Ahora tenemos todos los ingredientes para que un programa juegue al Ta Te Ti. La versión de arriba está simplificada de muchas maneras, y sin embargo contiene los elementos esenciales de jugar una ronda de Ta Te Ti.

**Realizar la siguiente actividad:** Implementen el programa de arriba y jueguen algunas partidas de Ta Te Ti. Cuando juegan contra una computadora, están anticipando posibles movimientos en el camino y realizan los propios movimientos con aquellos en mente.

Probablemente no tuvieron problemas en ganarle a la computadora en este juego. Esto se debe a que la computadora simplemente está eligiendo movimientos al azar entre una serie de posibles movimientos:

```
# el jugador es la computadora, se toma una elección aleatoria
casilla = choice(movimientosPosibles(tablero, jugador))
```

Esta es la línea en `mover` donde podemos poner alguna inteligencia en el programa. Sin embargo, la pregunta básica que debe hacerse es: ¿cuál de los posibles movimientos es el mejor para mí? Piensen esto por un momento. En el Ta Te Ti, se juega de dos maneras: defensivamente, como para no perder después de la siguiente jugada, u ofensivamente, para intentar ganar. Si la victoria es inminente, elegirán por supuesto hacer esa jugada, pero si no lo es (y tampoco es pérdida), ¿cómo eligen el movimiento? Intentaremos generalizar esta idea de manera que también sea aplicable a otros juegos de tablero.

Deleguemos la responsabilidad de encontrar el mejor movimiento a una función: `mejorJugada` que devolverá la mejor jugada de entre un grupo de posibles movimientos. Es decir, podemos cambiar la línea en movimiento de la siguiente manera:

```
# el jugador es la computadora, se toma una elección aleatoria
casilla = mejorJugada(tablero, jugador, movimientosPosibles(tablero, jugador))
```

Si `mejorJugada` hiciera una elección al azar (como arriba) podríamos escribirla como:

```
def mejorJugada(tablero, jugador, movimientos):
   return choice(movimientos)
```

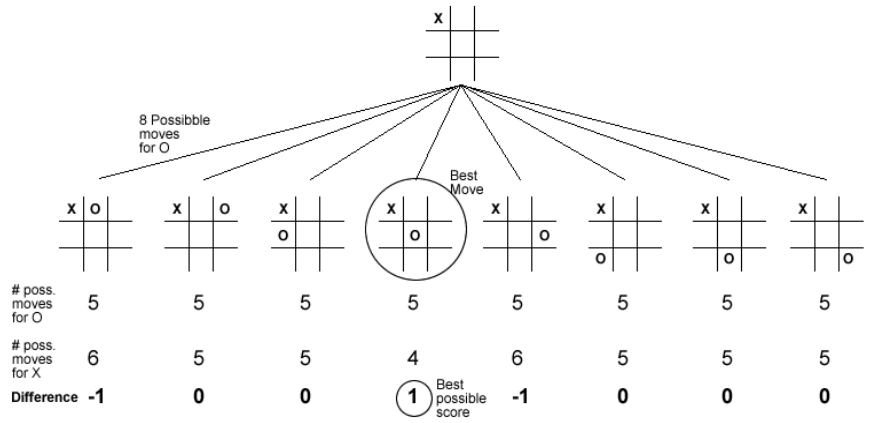
Imaginen que han decidido ir a ver a dos jugadores en una partida de ajedrez (o de Ta Te Ti). Sin embargo, llegan en el medio del juego. Entran en la habitación, miran el tablero y son capaces de evaluar cómo le está yendo a cada jugador sin saber nada de cómo se llegó a esa situación en el tablero. Por ejemplo, miren el siguiente tablero de Ta Te Ti:

```
 X | O | X       O | X | X       X | O | X       X |   |
---|---|---     ---|---|---     ---|---|---     ---|---|---
   | O |           |   |         O | X | O         |   |
---|---|---     ---|---|---     ---|---|---     ---|---|---
 O | X | X       X |   | O         | X |           |   |
     1               2               3               4
```

En todos los casos de arriba, la computadora está jugando con O y a continuación es su turno. Examinemos cada tablero y pensemos cuál sería el mejor movimiento para O.

En el caso 1, la computadora debe jugar defensivamente y ubicar una O en el cuadrado 6 para evitar perder el juego. En el caso 2, puede estar a la ofensiva y reconocer que ganará el juego ubicando una O en el cuadrado 5. En el caso 3, debe jugar para evitar perder. En el último caso, el tablero está ampliamente abierto. De las posibles ocho elecciones, ¿existe la mejor jugada? Por experiencia, probablemente vayan a ubicar al O en el centro del cuadrado (cuadrado 5). ¿Por qué? Elaboremos esto mirando hacia adelante (ver la figura en la siguiente página).

Piensen cómo podríamos cuantificar cada una de las posiciones del tablero arriba con respecto a las posibilidades de ganar que tiene O. Para cada tablero, cuenten el número de posibles posiciones ganadoras que le restan a O y compárenlas con las de X. Por ejemplo, en el primer tablero, O tiene 5 posiciones con posibilidades de ganar y X tiene 6. Podríamos decir que O es 5 – 6 = -1 tiene una posibilidad menos que X. Si hacen esto para el segundo y tercer tablero, verán que ambos X y O están parejos en la cantidad de posibles posiciones ganadoras restantes (5 cada uno). Sin embargo, en el cuarto caso, a O le restan 5 posibles posiciones ganadoras, y X tiene sólo 4. Intenten resolver las posiciones de tablero que quedan para confirmar los puntajes mostrados arriba. Claramente verán que ubicar a O en el cuadrado 5 es la mejor de todas las posibilidades. Intenten elaborar los 5 posibles movimientos para O en el caso 2 arriba. Verán que uno de los cinco conduce al triunfo para O. De modo similar, cuando elaboren las dos opciones para O en el caso 1, notarán que uno conduce a un empate y el otro a una derrota para O. Podemos capturar todas estas situaciones en una función que mire el tablero y devuelve un número que representa el puntaje a favor de O: cuanto más alto sea el valor devuelto, más deseable es el tablero.




$$\text{evaluar(tablero)} = \begin{cases} \infty, \text{ si el tablero es ganador para la computara} \\ -\infty, \text{ si el tablero es perdedor para la computadora} \\ \texttt{jugadasAbiertas}(\text{computadora}) - \texttt{jugadasAbiertas}(\text{usuario}), \text{ o/w} \end{cases}$$

Es decir que el triunfo para la computadora recibe un puntaje alto, una derrota recibe un puntaje bajo. De otra manera, es la diferencia en la cantidad de triunfos abiertos que restan para cada uno. Podemos capturar esto en Python de la siguiente manera:

```
INFINITO = 10
def evaluar(tablero):
    # si el tablero es ganador para el usuario, retornamos INFINITO
    pieza = ganador(tablero)

    if pieza == Yo:
        return INFINITO
    elif pieza == Vos:
        return –INFINITO
    else:
        return jugadasAbiertas(tablero,Yo) - jugadasAbiertas(tablero,Vos)
```

Definimos `INFINITO` como 10, un número lo suficientemente alto con relación a los otros valores que puede ser devuelto por `evaluar`. `jugadasAbiertas` mira a cada triple triunfo en la tabla y cuenta el número de aperturas para el jugador dado.

```
def jugadasAbiertas(tablero, jugador):
   possWins = 0

   for pos in wins:
      n = 0

      for i in range(3):
         if (tablero[pos[i]] == jugador) or
            (tablero[pos[i]] == ' '):
```

```
            n += 1
        if n == 3: possWins += 1

    return possWins
```

**Realizar la siguiente actividad:** Implementen las dos funciones de arriba y pruébenlas. Creen varias posiciones de tableros (usen las de los ejemplos de arriba) y confirmen que el tablero está siendo evaluado con los valores correctos.

A continuación, podemos reescribir `mejorJugada` para tomar ventaja de la evaluación:

```
def mejorJugada(tablero, jugador, movimientos):
    scores = []

    for mov in movimientos:
        b = copy(tablero)
        aplicarMovimiento(b, jugador, movimientos)
        scores.append(evaluar(b))

    return movimientos[scores.index(max(scores))]
```

Observen cómo toma cada posible jugada, crea una nueva tabla con esa jugada y luego evalúa el puntaje del tablero resultante. Finalmente, devuelve la jugada con el puntaje más alto como la mejor jugada. Modifiquen el programa de arriba para usar esta versión de `mejorJugada` y jugar varias veces. Podrán observar que hay una significativa mejora en la habilidad de juego de la computadora. ¿Pueden medirla de alguna manera? (Ver ejercicios).

La reescritura de arriba de `mejorJugada` hará que el programa juegue significativamente mejor, pero aún hay lugar para mejoras. En la mayoría de los juegos de mesa, los buenos jugadores son capaces de imaginar el partido anticipando varias jugadas. En muchos juegos, como el ajedrez, varias situaciones reconocibles conducen a resultados claramente determinados, por lo tanto, una gran parte de jugar una partida exitosa depende de la habilidad para reconocer esas situaciones.

Anticipar varias jugadas de manera sistemática es algo que las computadoras son bien capaces de hacer y por lo tanto cualquiera (¡aún ustedes!) pueden convertirlas en jugadores razonablemente buenos. El desafío reside en la cantidad de jugadas que pueden anticipar y en la capacidad limitada, si el tiempo para hacer la siguiente jugada es limitado, ¿cómo elegir entre las mejores opciones disponibles? Estas decisiones le dan un carácter interesante a los programas de juegos y continúan siendo una fuente importante de fascinación para muchas personas. Observemos cómo el programa de Ta Te Ti puede ver fácilmente todas las posibles movidas hasta el final del juego para determinar su siguiente movida (que en la mayoría de las situaciones conducen a empate, dada la simplicidad del juego).

Cuando uno mira hacia delante unas jugadas, tiene en cuenta que el oponente intentará ganar en cada jugada. El programa, al tratar de elegir la mejor jugada,

puede adelantarse a las del oponente para tenerlas en cuenta al elegir el mejor movimiento. De hecho, puede ir aún más lejos, hasta el final. La función `evaluar` que escribimos arriba puede ser usada con eficacia para evaluar futuras situaciones de tablero asumiendo que cuando es el turno de la computadora, siempre intentará elegir la jugada que prometa el mayor puntaje en el futuro. Sin embargo, al examinar los movimientos del oponente, debe asumir que el oponente realizará la jugada que resulte la peor para la computadora. En otras palabras, al mirar hacia delante, la computadora maximizará su posible puntaje mientras que el oponente minimizará las posibilidades de ganar de la computadora. Esto puede lograrse de la siguiente manera:

```
def mirarHaciaAdelante(tablero, jugador, MAX, nivel):
  movimientos = movimientosPosibles(tablero)

  if nivel <= 0 or len(movimientos)==0: # limite de mirarHaciaAdelante
     return evaluar(tablero)

  if MAX:                                          # movimiento de la computadora
     V = -INFINITO
     for m in movimientos:
        b = copy(tablero)
        b[m] = jugador
        V =max(V,mirarHaciaAdelante(b,oponente(jugador),1-MAX,nivel-1))
   else:                                           # movimiento del oponente
     V = INFINITO
     for m in movimientos:
        b = copy(tablero)
        b[m] = jugador
        V =min(V,mirarHaciaAdelante(b,oponente(jugador),1-MAX,nivel-1))
   return V
```

La función `mirarHaciaAdelante` definida arriba toma el tablero actual, el jugador al que le toca mover, ya sea la computadora (una intentando maximizar sus resultados) o la computadora (una intentando minimizar los resultados de la computadora), y los niveles aún por verse adelante, y calcula un puntaje basado en el examen de todas las jugadas hacia delante hasta el límite de la función *`mirarHaciaAdelante`*. En la función de arriba, cuando MAX es 1, representa a la computadora y 0 representa a su oponente. Por ende, dependiendo del valor de MAX, la evaluación es minimizada o maximizada respectivamente. Cada vez que mira más hacia adelante, el nivel se reduce por 1. El valor final devuelto por `mirarHaciaAdelante` puede ser usado por `mejorJugada` de la siguiente manera:

```
NIVEL = 9
def mejorJugada(tablero, jugador, movimientos):
   scores = []
   for m in movimientos:
      b = copy(tablero)
      b[m] = jugador
      scores.append(mirarHaciaAdelante(b,oponente(jugador),0,NIVEL-1)
   return movimientos[scores.index(max(scores))]
```

Como anteriormente, la jugada con el valor más alto es considerada la mejor jugada. Hemos determinado el valor de `NIVEL` arriba en 9 (por ejemplo, mirar 9 jugadas hacia delante), lo cual implica que cada vez mirará tan lejos como el final del juego antes de tomar la decisión. Sólo puede haber un máximo de 9 jugadas en un partido de Ta Te Ti. La calidad de la toma de decisiones de la computadora puede de hecho ser ajustada bajando el valor de `NIVEL`. En `NIVEL = 1`, será equivalente a la versión que escribimos atrás que sólo usaba la función `evaluar`.

Cuántos niveles hacia adelante se miren en un juego como éste depende del juego en sí mismo. ¿Pueden adivinar cuántas situaciones de tablero tendrá que ver la computadora al realizar una mirada hacia delante en `NIVEL = 9` después de la primera movida del usuario? ¡Serán 40,320 situaciones de tablero distintas! ¿Por qué? Además, para cuando es el momento de la segunda jugada de la computadora, sólo necesitará mirar 720 posiciones de tablero. Esto se debe a que en el Ta Te Ti, a medida que se llena el tablero, quedan menos posibles movimientos. De hecho, para cuando es el momento de la tercera jugada de la computadora, sólo necesita mirar un total de 24 tableros. Y, si la computadora llega a la cuarta jugada, sólo deberá ver dos posibles movimientos. Por lo tanto, en un exhaustivo examen de todas las posibles posiciones de tablero hasta el final de la partida cada vez que la computadora deba moverse, estará revisando un total de 41.066 posiciones de tablero. Sin embargo, si consideran un típico juego de ajedrez, en el cual cada jugador realiza un promedio de 32 movimientos y el número de jugadas factibles disponibles en cualquier momento es de alrededor de 10, pronto se darán cuenta de que la computadora debería examinar algo del orden de $10^{65}$ posiciones de tablero antes de realizar una jugada. Esto, aún para las computadoras más rápidas disponibles hoy en día tardaría mucho tiempo. Ampliaremos esto más adelante. Pero, para jugar un juego de dos personas-suma-cero, no es tan esencial mirar tanto hacia delante. Pueden variar la cantidad de jugadas anticipadas ajustando el valor de `NIVEL` en los programas.

**Realizar la siguiente actividad:** Implementen `mirarHaciaAdelante` como se describe arriba y comparen cuán bien juega la computadora contra ustedes. Traten de variar los niveles de 1, 2, ..., para ver si realmente hay mejoría en el juego de la computadora. ¿Considerarían a este programa inteligente?

Los ejercicios al final del capítulo los guiarán para transformar el programa de arriba en uno más robusto e incluso en un programa eficiente para el juego de Ta Te Ti. Sin embargo, estudien la estructura del programa cuidadosamente y serán capaces de usar la misma estrategia, incluyendo gran parte del corazón del programa para jugar muchos otros juegos de mesa de a dos personas.

## Un Piedra Papel o Tijera más inteligente

En el Capítulo 7 han visto el ejemplo de un programa que jugó a Piedra Papel o Tijera contra un usuario humano. En esa versión, la estrategia de elección del programa para elegir un objeto era completamente al azar. Reproducimos esa sección del programa aquí:

```
...
items = ["Papel", "Tijera", "Piedra"]
```

```
...
# La computadora elije
miEleccion = items[randint(0, 2)]
...
```

En el segmento de programa de arriba, `miEleccion` es la elección del programa.

Como pueden ver, el programa usa un número al azar para seleccionar su objeto. Es decir, las posibilidades de elegir cualquiera de los tres objetos es de 0.33 o 33%. Las estrategias para jugar y triunfar aquí han sido extensamente estudiadas. Algunas estrategias dependen de detectar patrones en el comportamiento humano de selección. Aunque no nos demos cuenta, hay patrones en nuestro aparente comportamiento al azar. Los programas de computadora fácilmente pueden rastrear estos patrones de comportamientos manteniendo largas historias de decisiones de jugadores, detectarlas y luego diseñar estrategias para ganarle a esos patrones. Esto ha demostrado ser muy efectivo. Involucra grabar las decisiones del jugador, y buscar a través de ellas. Otra estrategia es estudiar las estadísticas humanas de elecciones en este juego. Antes de presentarles algunos datos, realicen el siguiente ejercicio sugerido abajo:

**Realizar la siguiente actividad:** Jugar el juego contra algunas personas. Jugar varias rondas. Registrar las elecciones realizadas por cada jugador (sólo escribir P/P/T en dos columnas). Una vez hecho esto, computen el porcentaje de veces en que cada objeto es elegido. Ahora continúen leyendo.

Resulta que la mayoría de los jugadores humanos tienden a elegir Piedra en lugar de Tijera o Papel. De hecho, varios análisis sugieren que el 36% del tiempo, la gente tiende a elegir Piedra, el 30% Papel, y el 34% Tijera. Esto sugiere que PPT no es meramente un juego de azar y que hay espacio para algunas estrategias para ganar. Lo crean o no, hay campeonatos mundiales de PPT todos los años. Aún un simple juego como éste tiene numerosas posibilidades. Podemos usar esta información, por ejemplo, para hacer que nuestro programa sea más inteligente, o más apto para jugar el juego. Todo lo que debemos hacer es en lugar de usar una posibilidad pareja de 33% de seleccionar cada objeto, podemos sesgar las posibilidades de selección basándonos en las preferencias de la gente. Por lo tanto, si el 36% del tiempo la gente tiende a elegir Piedra, sería mejor que el programa eligiera Papel el 36% de las veces dado que el Papel le gana a la Piedra. De modo similar, nuestro programa debería elegir Tijera el 30% de las veces para equiparar las posibilidades de ganarle al Papel, y elegir Piedra el 34% de las veces para equiparar las posibilidades de ganarle al Papel.

Podemos influir en el generador de número al azar usando los siguientes porcentajes:

```
Primero generar un número al azar en el rango 0..99
Si el número generado está dentro del rango 0..29, seleccionar Tijera (30%)
Si el número generado está dentro del rango 30..63, seleccionar Piedra (34%)
Si el número generado está dentro del rango 64..99, seleccionar Papel (36%)
```

La estrategia de arriba de influir en la selección al azar puede ser implementada de la siguiente manera:

```
def miSeleccion():
    # Primero, genero un numero aleatoria en el rango de 0..99
    n = randrange(0, 100)

    # Si n esta en el rango 0..29, elegimos Tijera
    if n <= 29:
        return "Tijera"
    elif n <= 63:
        # Si n esta en el rango de 30..63, elegimos Piedra
        return "Piedra"
    else:
        return "Papel"
```

**Realizar la siguiente actividad:** Modifiquen su programa de PPT del capítulo 7 para usar esta estrategia. Jueguen el juego varias veces. ¿Funciona mejor que en la versión previa? Deberán testear esto recolectando datos de las dos versiones contra varias personas (¡asegúrense de que sean novatos!).

Otra estrategia que utiliza la gente se basa en la siguiente observación:

Luego de varias rondas, la gente tiende a hacer la jugada que le hubiera ganado a su movida previa.

Digamos que un jugador selecciona Papel. A continuación elegirá Tijera. Un programa de computadora o un jugador que esté jugando contra éste entonces debería elegir Piedra para ganarle a Tijera. Dado que la relación entre las elecciones es cíclica, la estrategia puede ser implementada eligiendo lo que le gana a lo que le gana a la jugada anterior del oponente. El Papel le gana a la Piedra. Por lo tanto, dado que la jugada anterior del jugador fue Papel, el programa puede elegir Piedra, anticipándose a la elección de Tijera por parte del jugador. Intenten pensar esto cuidadosamente, asegurándose de que la cabeza no les esté dando vueltas al final. Si el jugador puede detectarlo, puede usarlo como una estrategia ganadora. Dejaremos la implementación de esta última estrategia como un ejercicio. Los ejercicios también sugieren otra estrategia.

El punto de los ejemplos de arriba es que utilizando estrategias en sus programas pueden hacer que sean más inteligentes. Deliberadamente, hemos empezado a usar el término inteligencia de modo un poco más suelto que lo que Alan Turing sugiere en su famoso ensayo. Muchas personas argumentarán que estos programas no son inteligentes en el sentido último del término. Estamos de acuerdo. Sin embargo, escribir programas más inteligentes es una actividad natural. Si los programas incorporan estrategias o heurísticas que las personas usarían cuando realizan alguna actividad, entonces los programas tienen alguna forma de inteligencia artificial. Aún si la estrategia utilizada por el programa no tiene nada que ver con lo que usaría la gente, pero haría al programa más inteligente o mejor, lo llamaríamos inteligencia artificial. Muchas personas estarían en desacuerdo con esta observación última. Para algunos, la búsqueda por resolver la inteligencia se limita a comprenderla en los humanos (y otros animales). En la IA ambos puntos de vista prevalecen y generan debates apasionantes entre académicos.

## Discusión

La sola idea de considerar a una computadora como un dispositivo inteligente tiene sus orígenes en la naturaleza del propósito general de las computadoras. Al cambiar el programa, se puede lograr que la misma computadora se comporte de maneras diferentes. En el fondo, una computadora es sólo un manipulador de símbolos: manipula codificaciones de números o letras o imágenes, etc. Se postula que también el cerebro humano es un manipulador de símbolos. Las bases de la IA residen en el hecho de que la mayoría de los sistemas inteligentes son símbolos físicos y dado que una computadora es un manipulador de símbolos de propósitos generales, puede ser usada para estudiar o simular la inteligencia.

## Revisión Myro

No se introdujeron nuevas funciones Myro en este Capítulo.

## Revisión Python

choice(LISTA)
Devuelve un elemento seleccionado al azar de LISTA. Esta función se importará del módulo  random.

*List Comprehensions*
Una forma breve y elegante de construir listas. Ver la documentación Python para más información.

## Ejercicios

1. Leer el artículo de Alan Turing “Computing Machinery and Intelligence” (Máquinas Computadoras e Inteligencia). Fácilmente lo podrán hallar buscando en la Web.

2. Realicen una búsqueda en la Web de “Argumento de la habitación china de Searle” para ubicar los argumentos del filósofo John Searle acerca de que no importa cuán inteligente sea una computadora o programa, nunca podrá tener una ¨mente¨.

3. Reescriban la vista del juego de Ta Te Ti para ver el tablero gráficamente.

4. Diseñen un lenguaje de comandos de una palabra para el Scribbler. Escriban un programa para ingresar un comando a la vez, interpretarlo y luego hacerlo correr al comando en el Scribbler.

5. Extiendan el lenguaje del Ejercicio 4 para incluir consultas (por ej. ¿pared?) y luego modifiquen el programa para incorporar estas consultas.

6. Realicen una investigación sobre sistemas de comprensión del habla.

7. Realicen una investigación sobre lingüística computacional.

8. En el programa de Ta Te Ti diseñado en este Capítulo, asumimos que el usuario siempre juega con una X. Modifiquen el programa para que le dé la posibilidad de elegir al inicio del juego. Más adelante, al final del juego, las piezas se intercambian.

9. En la función mover definida para el Ta Te Ti, el programa acepta lo que sea que el usuario ingrese como movimiento. Prueben el programa y en lugar de ingresar un movimiento válido, ingresen su nombre. ¿Qué sucede? Tales errores pueden ser fácilmente detectados, dado que detendrán la ejecución del programa. Sin embargo, a continuación intenten ingresar un número del 1 al 9 usando una posición de repuesto que ya esté ocupada en el tablero. ¿Qué ocurre? Modifiquen la función para aceptar sólo los movimientos correctos. Si el usuario ingresa una movida incorrecta, el programa debería señalarlo y darle al usuario otra oportunidad.

10. La función gameOver puede hacer uso de la función ganador para tomar esta decisión en el programa de Ta Te Ti. Reescriban gameOver para hacer esto.

11. Una manera de medir cómo se compara una estrategia contra otra es jugarla contra otra estrategia una y otra vez registrando la cantidad de veces que se gana, se pierde y se empata. Modifiquen su programa de Ta Te Ti o PPT para reemplazar la segunda estrategia para el usuario (en lugar de tomar la entrada (input) del usuario, usa la función que implementa la segunda estrategia). Agreguen enunciados para jugar el juego muchas veces (digamos 1000) y registren las veces que se gana, se pierde y se empata. También querrán suprimir toda salida (output) de tablero dado que el juego se está jugando enteramente dentro del programa.

Realicen una comparación entre las estrategias discutidas en este Capítulo y cómo se comparan jugando entre sí.

# 11 Computadoras & Computación

*Las computadoras hogareñas están siendo convocadas para realizar muchas funciones nuevas, incluyendo la realización de la tarea escolar previamente comida por el perro.*
Doug Larson

*La Ciencia de la Computación se trata tanto de computadoras como la astronomía de telescopios.*
Edsger W. Dijkstra

¿Cuál es el centro de las ciencias de la computación? Mi respuesta es simple –es el arte de programar una computadora.. *Es el arte de diseñar métodos eficientes y elegantes para lograr que una computadora resuelva problemas, teóricos o prácticos, pequeños o grandes, simples o complejos. Es el arte de traducir este diseño en un programa de computación efectivo y preciso.*
C.A.R. Hoare

Imagen: La computadora portátil XO
La foto es cortesía del proyecto OLPC (www.olpc.org)

Hoy hay más computadoras que gente en la mayoría de los campus de las universidades en los Estados Unidos. La computadora portátil (laptop) que se muestra en la página anterior es la laptop XO desarrollada por el Proyecto Una Laptop Por Niño (OLPC). Es una computadora de bajo costo diseñada para niños. El proyecto OLPC aspira a llegar a 2 billones de niños en todo el mundo. Buscan hacer llegar las computadoras XO a sus manos para ayudar en la educación. Su misión es lograr un cambio radical en el sistema de educación global a través de las computadoras. Por supuesto que un proyecto con una misión tan grandilocuente no puede aparecer sin controversias. Ha tenido sus escollos desde sus inicios. Las cuestiones tecnológicas fueron el menor de sus problemas. Han tenido que convencer a gobiernos (es decir, políticos) para que apoyen su misión. Varios países han acordado su apoyo y luego se han echado atrás por todo tipo de razones. Las laptops no estaban bajo venta libre dentro de los Estados Unidos, lo cual ha llevado a otras controversias socio-políticas. De todas maneras, en el primer año de su lanzamiento, el proyecto apunta a tener más de 1 millón de XOs en manos de niños de muchos países en desarrollo. Algo que sucede con la tecnología que siempre ha sido verdad es que una buena idea es inefectiva. Otros jugadores han emergido que están desarrollando computadoras a bajísimo costo para niños. Pronto otros productos competitivos estarán disponibles para todos. Las preguntas que aparecen son: ¿Qué tipo de cambio traerá aparejado esto en el mundo? ¿Cómo serán usadas estas computadoras para enseñar a los niños en las escuelas primarias y secundarias? Etc. También se estarán preguntando si el robot Scribbler puede ser controlado por la XO. Sí, puede.

Han estado escribiendo programas para controlar su robot a través de una computadora. En el camino han visto muchas formas diferentes de controlar al robot y también de organizar sus programas Python. Básicamente, han estado comprometidos con la actividad generalmente entendida como *programación de computadoras* o resolución de problemas basada en computadoras. Deliberadamente hemos evitado usar esos términos porque la percepción general de los mismos convoca imágenes de personajes con protectores de bolsillos y ecuaciones matemáticas complejas que parecen ser evitables o fuera del alcance para la mayoría de las personas. Ojalá que para este momento hayan descubierto que la solución de problemas a través de una computadora puede ser bastante excitante y atrapante. El robot puede haber sido el verdadero motivo que los introdujo en esta actividad y eso fue pensado así. Les estamos confesando que usamos a los robots para atraer un poco su atención hacia la computación. Deben admitir, si están leyendo esto, ¡que el artilugio funcionó! Pero recuerden que el motivo por el cual están sosteniendo un robot propio en sus manos se debe también a las computadoras. El mismo robot es una computadora. Haya sido o no un artilugio, han asimilado muchas ideas clave de computación y de ciencia de la computación. En este capítulo haremos más explícitas estas ideas y también les daremos un sabor de lo que realmente se trata la *ciencia de la computación*. Como sostiene Dijkstra, *la ciencia de la computación es tanto sobre computadoras como la astronomía sobre telescopios*.

## Las computadoras son tontas

Supongamos que están alojando a una estudiante internacional de intercambio en su casa. Al poco tiempo de su llegada, le dan a conocer las virtudes de los sándwiches de PB&J (manteca de maní y jalea- quizás equiparable a un pan con manteca y dulce de leche en Argentina). Luego de escucharlos con atención, su boca se empieza a hacer agua y muy educadamente les piden si podrían compartir la receta con ella. Ustedes la escriben en un pedazo de papel y se la dan.

**Realizar la siguiente actividad:** Adelante, escriban la receta para hacer un sándwich de PB&J.

De verdad, traten de escribirla, ¡insistimos!

OK, ahora tienen una receta para hacer sándwiches de PB&J.

**Realizar la siguiente actividad:** Usen la receta de arriba para hacerse un sándwich de PB&J. Intenten seguir las instrucciones lo más *literalmente* que puedan. Si lograron realizarse con éxito el sándwich, ¡felicitaciones! ¡Disfrútenlo! ¿Creen que su amiga podrá seguir la receta y también disfrutar de los sándwiches de PB&J?

Deben admitir que escribir una receta para sándwiches de PB&J al principio parecía trivial, pero al sentarse a escribirlo no les queda otra opción que cuestionarse varias cosas que se han dado por supuestas: ¿Sabe ella qué es la manteca de maní? ¿Debo recomendar una marca específica? ¿Mermelada Ditto? A todo esto, se acordaron de mencionar que se trata de *jalea de uva*? ¿El tipo de pan a usar? ¿Será pre-cortado? Si no, ¿necesitan un cuchillo para cortar la rodaja de pan? ¿Especificaron cuán gruesas deberían ser las rodajas de pan? ¿Debería usar el mismo cuchillo para untar la manteca de maní y la jalea? Etc. La cuestión es que, con tanto nivel de detalle, uno puede seguir y seguir... ¿Su amiga sabe cómo usar un cuchillo para untar manteca o jalea en una rodaja de pan? De repente, una tarea aparentemente trivial se transforma en un ejercicio desalentador. En realidad, al escribir su receta, dan varias cosas por hecho: que ella sabe lo que es un sándwich, que requiere rodajas de pan, untar cosas en rodajas, y juntarlas. ¡Listo, tienen un sándwich!

Piensen en la cantidad de recetas que han sido publicadas en libros de cocina de todo el mundo. La mayoría de los buenos autores de libros de cocina empiezan con un plato, lo escriben en una receta, lo prueban varias veces y lo refinan de diferentes maneras. Previo a la publicación, la receta es probada por otros. Después de todo, la receta es para que otros la sigan y recreen un plato. La receta es revisada y ajustada basándose en el feedback (la devolución) de los probadores (o testeadores) de la receta. Lo que los autores de libros de cocina asumen es que tendrán las suficientes competencias para seguir la receta y recrear el plato para su satisfacción. Quizás nunca resulte el mismo plato que preparó el autor. Pero les dará una base sobre la cual improvisar. ¿No sería lindo que hubiera una forma exacta de seguir una receta para que terminaran con exactamente el mismo plato que el del autor del libro de cocina cada vez que prepararan ese plato? ¿Sería bueno? Eso depende de sus gustos y preferencias. Sólo por diversión, aquí hay una receta de unos ricos Brochetas de pollo al Azafrán.

**Brochetas de pollo al azafrán**

Ingredientes

1lb de pechuga de pollo deshuesada, cortada en cubos de 1-2 pulgadas.
1 media cebolla en rodajas
1 cucharadita de hebras de azafrán
1 lima
1 cucharada de aceite de oliva
Sal y pimienta negra a gusto

Preparación

1. Mezclar el pollo y las cebollas en un bowl no-reactivo.
2. Apretar con los dedos y agregar las hebras de azafrán
3. Agreguen el jugo de lima, el aceite de oliva, la sal y la pimienta.
4. Dejar reposar en la heladera por lo menos por 30 minutos (o toda la noche).
5. Precalentar una parrilla o un horno a unos 200 grados.
6. Armar los pinchos, descartando las rodajas de cebolla. O poner todo en una bandeja de hornear forrada si se usa el horno.
7. Cocinar a la parrilla o al horno por 12-15 min. hasta que estén listas.

En las recetas de cocina como la de arriba, pueden dar por hecho varias cosas: serán usadas por personas (como ustedes y yo); serán capaces de seguirlas; quizás hasta improvisen. Por ejemplo, en la receta de arriba, no especificamos que se necesitará un cuchillo para cortar el pollo, las cebollas o la lima; o que se necesitará una parrilla o un horno; etc. La mayoría de las recetas asume que serán capaces de *interpretar* y seguir la receta así como está escrita.

Los programas de computadora son también como recetas, hasta cierto punto. Piensen en el programa que escribieron para coreografiar una danza de robot, por ejemplo. Hemos reproducido la versión del Capítulo 3 aquí:

```
# File: dance.py
# Purpose: A simple dance routine
# First import myro and connect to the robot

from myro import *
initialize("com5")

# Define the new functions...

def yoyo(speed, waitTime):
      forward(speed, waitTime)
      backward(speed, waitTime)
      stop()
```

```
def wiggle(speed, waitTime):
	motors(-speed, speed)
	wait(waitTime)
	motors(speed, -speed)
	wait(waitTime)
	stop()

# The main dance program
def main():
	print "Running the dance routine..."
	yoyo(0.5, 0.5)
	wiggle(0.5, 0.5)
	yoyo(1, 1)
	wiggle(1, 1)
	print "...Done"

main()
```

En muchos sentidos, el programa de arriba es como una receta:

**Hacer una danza de robot**

Ingredientes

1 función yoyo para que el robot vaya hacia adelante y hacia atrás a una velocidad dada
1 función wiggle que le permite al robot menearse a una velocidad determinada

Preparación

1. yoyo a velocidad 0.5, esperar 0.5
2. menearse a velocidad 0.5, esperar 0.5
3. yoyo a velocidad 1, esperar 1
4. menearse a velocidad 1, esperar 1

Luego, de manera similar podrían especificar los pasos involucrados en hacer los movimientos de `yoyo` y `wiggle` en forma de receta. Este puede parecer un ejemplo trivial, pero toca dos puntos importantes: un programa de computadora es como una receta en la cual se detalla el listado de ingredientes y un método o pasos para lograr las tareas dadas; y, como una receta, sus ingredientes y los pasos requieren un cuidadoso planeamiento y reflexión previos. Es de destacar que los programas de computadora son diferentes de las recetas en un aspecto: ¡están diseñados para ser seguidos por una computadora!

Una computadora es un dispositivo tonto diseñado para seguir instrucciones/recetas. Nos ahorraremos los detalles de cómo una computadora hace lo que hace para un curso posterior. Pero es casi un conocimiento popular que todo adentro de la computadora se representa con 0s y 1s. Empezando de 0s y 1s uno puede diseñar esquemas de codificación para representar números, letras del alfabeto, documentos, imágenes, películas, música, etc. y cualquier otra entidad

abstracta que deseen manipular utilizando una computadora. Un programa de computadora es, en última instancia, también representado como una secuencia de 0s y 1s y es con esta forma que a la mayoría de las computadoras les gusta seguir recetas. No importa cuán limitante o degenerado suene esto, pero es la clave del poder de las computadoras. Especialmente al entender que esta simplificación es la que le permite a la computadora manipular cientos de millones de fragmentos de información a cada segundo. El precio que debemos pagar por este poder es que debemos especificar nuestras recetas en forma de programas de computación de una manera más bien formal y precisa. Tanto que no hay lugar para la improvisación: ninguna vaguedad al estilo ¨pizca de sal¨, como en las recetas de cocina, es aceptable. Aquí es donde entran en escena los *lenguajes de programación*. Los científicos de la computación especifican sus recetas computacionales usando *lenguajes de programación*. Ustedes han estado usando el lenguaje de programación Python para escribir sus programas para el robot. Otros ejemplos de lenguajes de programación son Java, C++ (pron.: ce más más; en inglés ¨si plus plus¨), C# (pron.: ce sharp, en inglés ¨si sharp¨), etc. ¡Existen más de 2000 lenguajes de programación!

**Realizar la siguiente actividad:** ¿Pueden averiguar cuántos lenguajes de programación hay? ¿Cuáles son los diez lenguajes de programación más usados?

Previo a la existencia de lenguajes de programación, las computadoras se programaban usando secuencias de 0s y 1s. ¡No es necesario aclarar que muchas personas se volvieron locas con esto! Los lenguajes de programación, como Python, permiten una forma más amigable para que los programadores escriban sus programas. Los lenguajes de programación proveen un acceso fácil a codificaciones que representan el tipo de cosas con las que nosotros, los humanos, nos relacionamos. Por ejemplo, el enunciado Python:

```
sentidoDeLaVida = 42
```

es un comando para que la computadora asocie el valor 42 con el nombre `sentidoDeLaVida`. De esta manera, podemos pedirle a la computadora que verifique que es efectivamente 42:

```
if sentidoDeLaVida == 42:
    speak("Eureka!")
else:
    speak("Qué hacemos ahora?")
```

Una vez más, sería bueno recordarles que la elección del nombre sentidoDeLaVida, no significa realmente que estamos hablando acerca de “*el sentido de la vida”*. Bien podríamos haberlo llamado `timbuktoo`, como en:

```
timbuktoo = 42
```

Como pueden ver, ¡las computadoras son realmente tontas! Realmente depende de nosotros, los programadores, el asegurarnos de usar los nombres consistentemente y elegirlos, en primer lugar, cuidadosamente. Pero, al crear un lenguaje como Python, hemos creado una notación formal que al traducirse a 0s y 1s, cada enunciado querrá decir solamente una cosa, sin otras interpretaciones. Esto lo hace diferente a una receta de cocina.

## El robot sale a comprar huevos frescos

Las recetas, sin embargo, forman una buena base conceptual para comenzar a pensar en un programa para resolver un problema. Digamos que tienen en mente hacer su *Strudel de manzana* favorito. Saben que necesitarán manzanas. Quizás es la temporada de manzanas lo que motivó la idea. También necesitarán masa. Pero antes que nada, necesitarán esa receta que les dio la abuela.

Cada vez que se nos pide que resolvamos un problema usando una computadora, empezamos armando un plan borrador para resolverlo. Es decir, trazamos una estrategia. Esto luego se refina en pasos específicos, quizás se identifican y nombran algunas variables, etc. Una vez que están convencidos de tener una forma de resolver el problema, lo que tienen es un *algoritmo*.

La idea de un algoritmo es fundamental para la ciencia de la computación, así que dedicaremos un tiempo aquí a desarrollar esta noción. Quizás la mejor manera de relacionarnos con esta idea es con un ejemplo. Supongamos que un robot entra en un almacén a comprar una docena de huevos frescos. Supongamos que es capaz de hacerlo, ¿cómo se asegurará de que ha seleccionado los huevos más frescos disponibles?

Su robot personal no es capaz de este tipo de tarea, pero imaginen que sí. Mejor aún, dejemos la mecánica de lado, veamos cómo *ustedes* irían a comprar los huevos más frescos. Bueno, de algún modo necesitarían saber qué fecha es la de hoy. Digamos que es 15 de septiembre de 2007 (¿por qué esta fecha? ¡Ya lo sabrán!). También saben que las cajas de huevos generalmente tienen la fecha de vencimiento. De hecho, el USDA (Departamento de Agricultura de los Estados Unidos) ofrece voluntariamente, sin costo, programas de certificación para granjas de huevos. Un granjero de huevos puede participar voluntariamente en el programa de certificación de huevos de USDA, en el cual el USDA realiza inspecciones regulares y brinda ayuda para categorizar los huevos por tamaños. Por ejemplo, los huevos generalmente se clasifican en Grado AA, Grado A o Grado B. La mayoría de los almacenes tienen huevos Grado A. También vienen en varios tamaños: Extra Grande, Grande, Pequeño etc. Pero más interesante aún es que el sistema de etiquetado de las cajas tiene una información muy útil codificada en el mismo.

**Etiquetas en cajas de huevos**


Cada caja de cartón certificada tiene por lo menos tres fragmentos de información (ver la imagen a la derecha): un ¨consumir antes de¨ o ¨fecha de vencimiento¨, un código que identifica la granja de la cual vienen los

huevos y una fecha en que los huevos fueron envasados en la caja. La mayoría de la gente compra los huevos mirando la fecha de ¨consumir antes de¨ o ¨fecha de vencimiento¨. Sin embargo, la información acerca de si son frescos está codificada en la fecha de envasado. Para agregar a la confusión, la fecha está codificada como el día del año. Por ejemplo, miren la caja de arriba mostrada en la página anterior. Su fecha de ¨consumir antes de¨ (¨sell by¨) es 4 de octubre. "P1107" es el código de granja. Este cartón fue envasado en el 248avo día del año. Además, el USDE requiere que todos los huevos certificados se envasen dentro de los 7 días de haber sido puestos. Por lo tanto, los huevos de la caja superior se pusieron en algún momento entre el día 241 y el 248 de 2007. ¿Qué fechas corresponden a esos días?

A continuación, observen la caja de abajo. Esos huevos tienen un ¨consumir antes de¨ más tardío (18 de octubre) pero una fecha de envasado anterior: 233. Es decir, esos huevos fueron puestos en algún momento entre el día 226 y el 233.

¿Cuáles son los huevos más frescos?

Aunque la fecha de ¨consumir antes de¨ en la segunda caja es de dos semanas más tarde, la primera caja contiene los huevos más frescos. De hecho, los huevos en la caja de arriba fueron puestos por lo menos dos semanas después.

La fecha de envasado está codificada en un número de 3 dígitos. Por lo tanto, los huevos envasados el 1 de enero serán etiquetados: 001; los huevos envasados el 31 de diciembre, 2007 serán etiquetados: 365.

**Realizar la siguiente actividad:** Vayan al sitio Web del USDA (www.usda.gov) y vean si pueden averiguar de qué granja vinieron las dos cajas de huevos.

Para un robot, el problema de comprar los huevos más frescos consiste en averiguar, dada la fecha de envase, ¿cuál fue la fecha en que se envasaron los huevos?

Ajusten sus cinturones, estamos por embarcarnos en un viaje computacional único...

## Diseñar un algoritmo

Hasta aquí, hemos acotado el problema a las siguientes especificaciones:

```
Entrada
    Codificación de tres dígitos de la fecha de envasado

Salida
     Fecha en que fueron envasados los huevos.
```

Por ejemplo, si la fecha de envasado fue codificada como 248, ¿cuál será la fecha?

Bueno, eso depende. Podría ser 4 ó 5 de septiembre, dependiendo de si fue un año bisiesto o no. Entonces, resulta que el problema de arriba también requiere que

sepamos de qué año se trata. Resolver uno o dos problemas como ejemplo es siempre una buena idea porque ayuda a identificar información que falta que puede ser crítica para resolver el problema. Dado que también necesitamos conocer el año, podemos pedirle al usuario que lo ingrese a la vez que ingresa el código de 3 dígitos. La especificación del problema entonces se transforma en:

Entrada
    Codificación de tres dígitos de fecha
    Año actual

Salida
    Fecha en que se envasaron los huevos

**Ejemplo**:
    Entrada: 248, 2007
    Salida: Los huevos se envasaron el 5 de septiembre de 2007

¿Alguna idea sobre cómo resolverían el problema? Siempre ayuda que intenten hacerlo ustedes, con lápiz y papel. Tomen el ejemplo de arriba y vean cómo llegarían a la fecha de salida. Mientras lo están resolviendo, traten de escribir el proceso de resolución. Su algoritmo o receta será muy similar.

**La etimología de algoritmo**

Se cree que la palabra algoritmo, un anagrama de logaritmo, deriva de Al-Khowarizmi, un matemático que vivió de 780-850 DC. Su nombre completo era Abu Ja'far Muḥammad ibn Mūsā al-Khwārizmī, (Mohammad, padre de Jafar, hijo de Moses, un Khwarizmian). Gran parte del conocimiento matemático de la Europa medieval deriva de traducciones latinas de sus trabajos.


En 1983, la Unión Soviética imprimió la estampilla que se muestra arriba en homenaje a su 1200avo aniversario.

Supongamos que estamos intentando codificar el input 248, 2007. Si lo hicieran a mano, usando una lapicera y un papel, el proceso sería algo así:

La fecha no es en enero porque tiene 31 días y 248 es mucho mayor que 31.
Restemos 31 de 248: 248 - 31 = 217

217 también es mayor que 28, la cantidad de días de febrero, 2007.
Entonces, restemos 28 de 217: 217 - 28 = 189

189 es mayor que 31, la cantidad de días de marzo.
Restemos 31 de 189: 189 - 31 = 158

158 es mayor que 30, la cantidad de días de abril.
Entonces: 158 - 30 = 128

128 es mayor que 31, la cantidad de días de mayo.
Por lo tanto: 128 - 31 = 97

97 es mayor que 30, la cantidad de días de junio.
97 - 30 = 67

67 es mayor que 31, la cantidad de días de julio.
67 - 31 = 36

```
36 es mayor que la cantidad de días de agosto (31).
36 - 31 = 5

5 es menor que la cantidad de días de septiembre.
Por lo tanto debe ser el 5to día de septiembre.

La respuesta es: 248avo día de 2007 es 5 de septiembre de 2007.
```

Eso era evidentemente demasiado repetitivo y tedioso. Pero ahí es donde entran las computadoras. Observen el proceso de arriba y vean si hay un patrón para los pasos realizados. A veces, ayuda probar con otro ejemplo.

**Realizar la siguiente actividad:** Supongamos que el día y año de entrada es: 56, 2007. ¿Cuál es la fecha?

Cuando vean los cómputos de ejemplo que han realizado, encontrarán varios patrones. Identificarlos es la clave para diseñar un algoritmo. A veces, para que esto sea más sencillo, ayuda identificar o nombrar los fragmentos de información clave que se están manipulando. Generalmente esto comienza con las entradas y salidas identificados en la especificación del problema. Por ejemplo, en este problema, las entradas son: día del año, año actual. Comiencen por asignarle estos valores a nombres de variables específicos. Es decir, asignemos el nombre `dia` para representar el día del año (248 en este ejemplo), y `anio`[1] como el nombre para registrar el año actual (2007). ¡Noten que no elegimos nombrar a ninguna de estas variables `timbuktu` o `sentidoDeLaVida`!

También noten que repetidamente deben restar la cantidad de días del mes, empezando por enero. Asignemos la variable llamada `mes` para registrar el mes que se esté considerando.

A continuación, pueden reemplazar los nombres `dia` y `anio` en el cómputo de ejemplo:

```
Entrada:
  dia = 248
  anio = 2007

# Empezar considerando enero
mes = 1

La fecha no está mes = 1 porque tiene 31 días y 248 es mucho mayor que 31.
dia = dia - 31

# siguiente mes
mes = 2

dia (= 217) es también mayor que 28, el # de días en mes = 2
dia = dia - 28
```

1 Nota del Traductor: La letra ñ no forma parte de los caracteres permitidos para nombrar variables o funciones. Sucede lo mismo con los caracteres acentuados.

```
# siguiente mes
mes = 3
dia (= 189) es mayor que 31, el # de días en mes = 3.
dia = dia - 31

# siguiente mes
mes = 4
dia (= 158) es mayor que 30, el # de días en mes = 4.
dia = dia - 30

# siguiente mes
mes = 5
dia (= 128) es mayor que 31, el # de días en mes = 5.
dia = dia - 31

# siguiente mes
mes = 6
dia (= 97) es mayor que 30, el # de días en mes = 6.
dia = dia = 30

# siguiente mes
mes = 7
dia (= 67) es mayor que 31, el # de días en mes = 7.
dia = dia - 31

# siguiente mes
mes = 8
dia (= 36) es mayor que el # de días en mes = 8.
dia = dia - 31

# siguiente mes
mes = 9
dia (= 5) es menor que el # de días en mes = 9.
Por lo tanto debe ser el 5to día de septiembre.

La respuesta es: 5/9/2007
```

Observen ahora cuán repetitivo es el proceso. La repetición puede ser expresada más concisamente como se muestra abajo:

```
Entrada:
  dia
  anio

# empezar con mes = 1, para enero
mes = 1
repetir
  si día es menor que número de días en mes
    dia = dia – numero de días en mes
    # siguiente mes
    mes = mes + 1
  si no
    terminado

Salida: día/mes/año
```

Ahora empieza a parecer una receta o un algoritmo. Pruébenlo con las entradas del ejemplo de arriba y asegúrense de obtener los resultados correctos. Además, asegúrense de que este algoritmo funcione en los casos límite: 001, 365.

## Treinta días tiene septiembre

Podemos afinar aún más el algoritmo de arriba: algo que dejamos sin especificar arriba es el cómputo de la cantidad de días en un mes. Esta información debe ser explicitada para que la computadora sea capaz de seguir la receta. Entonces, ¿cómo computamos la cantidad de días en un mes? La respuesta parece ser sencilla. Muchos recordarán el siguiente poema:

*Treinta días tiene septiembre*
*Abril, junio y noviembre*
*El resto treinta y uno*
*Excepto febrero el mocho*
*Que tiene veintiocho*
*(Y veintinueve en cada año bisiesto)*

Desde la perspectiva del diseño, podemos asumir que tenemos un ingrediente, una función en este caso, llamada `diasEnMes` que, dado un mes y un año, computará y devolverá la cantidad de días en el mes. Es decir, que podemos afinar nuestro algoritmo de la siguiente manera:

```
Ingredientes:
  1 función diasEnMes(m, a): devuelve la cantidad de días  en el mes m y en año a.

Entrada:
  dia
  anio

# empezar con mes = 1, para enero
mes = 1
repetir
  si el día es menor que la cantidad de días en el mes
        dia = dia - diasEnMes(mes, anio)
        # siguiente mes
        mes = mes + 1
  si no
    terminado

Salida: día/mes/año
```

Ahora, tenemos que resolver el problema secundario:

```
Entrada
   mes, M
   anio, Y
Salida
   cantidad de días en mes, M en año, Y
```

A primera vista parece fácil, el poema de arriba especifica que abril, junio, septiembre y noviembre tienen 30 días, y el resto, con excepción de febrero, tienen 31. Febrero tiene 28 ó 29, dependiendo de si cae en un año bisiesto o no. Por lo tanto, fácilmente podemos elaborar una receta o un algoritmo para esto de la siguiente manera:

```
Entrada:
  m, a

si m es abril (4), junio(6), septiembre(9), o noviembre (11)
  dias = 30
si no, si m es febrero
       si a es año bisiesto
           dias = 29
       si no
           dias = 28
si no
  (m es enero, marzo, mayo, julio, agosto, octubre, diciembre)
   dias = 31

Salida:
  dias
```

Aún hay un detalle más: ¿cómo sabemos si es un año bisiesto?

Primero, intenten responder a la pregunta, *¿Qué es un año bisiesto?*

Nuevamente, podemos afinar el algoritmo de arriba suponiendo que tenemos otro ingrediente, una función `esBisiesto`, que determina si un año dado es bisiesto o no. Entonces podemos escribir el algoritmo de arriba como:

```
Ingredientes:
  1 función esBisiesto(a)
    devuelve True si y es año bisiesto, de lo contrario, false


Entrada:
  m, a

si m es abril (4), junio(6), septiembre(9), o noviembre (11)
  dias = 30
si no, si m es febrero
       si esBisiesto(a)
           dias = 29
       si no
           dias = 28
si no
  (m es enero, marzo, mayo, julio, agosto, octubre, diciembre)
   dias = 31

Salida:
  dias
```

A la mayoría nos han enseñado que un año bisiesto es un año divisible por 4. Es decir, el año 2007 no es bisiesto, dado que 2007 no es divisible por 4, pero 2008 sí lo es, porque es divisible por 4.

**Realizar la siguiente actividad:** ¿Cómo determinan si algo es divisible por 4? Prueben la solución en los años 1996, 2000, 1900, 2006.

## Años bisiestos: Bula Papal

Para diseñar una receta o un algoritmo que determine si el número correspondiente a un año es un año bisiesto o no es directo si aceptan la definición de la última sección. Por lo tanto, podemos escribir:

```
Entrada
   a, un año

Salida
   True (verdadero) si a es año bisiesto, de lo contrario, false (falso)
```

Método

```
 si a es divisible por 4
    es un año bisiesto, o True
  si no
    no es un año bisiesto, o False
```

Sin embargo, ésta no es la historia completa. El calendario occidental que seguimos se llama *Calendario Gregoriano*, y fue adoptado en 1582 por una Bula Papal dictada por el Papa Gregorio XIII. El Calendario Gregoriano define un año bisiesto agregando un día extra cada cuatro años. Sin embargo, se le aplica una corrección cada 100 años, lo cual complica un poco más la situación: Los centenarios no son años bisiestos a menos que sean divisibles por 400. Es decir que los años 1700, 1800, 1900, 2100 no son años bisiestos aunque sean divisibles por 4. Pero los años 1600, 2000, 2400 son años bisiestos. Para más información sobre esto, ver los ejercicios al final del capítulo. Nuestro algoritmo para determinar si un año es bisiesto se puede afinar tal como se muestra abajo:

```
Entrada
  a, un año

  si a es divisible por 400
     es un año bisiesto, o True
  si no, si a es divisible por 100
     no es un año bisiesto, o Falso
  si no, si a es divisible por 4
     es un año bisiesto, o True
  si no
     no es un año bisiesto, o Falso
```

Finalmente, hemos logrado diseñar todos los algoritmos o recetas que se requieren para resolver el problema. Quizás habrán notado que usamos algunas construcciones familiares para elaborar nuestras recetas o algoritmos. Luego, echemos una mirada rápida a las construcciones que se usan para expresar algoritmos.

## Componentes esenciales de un algoritmo

Los científicos de la computación expresan soluciones a problemas en términos de algoritmos, que básicamente son recetas más detalladas. Los algoritmos pueden ser usados para expresar cualquier solución y sin embargo se componen de elementos muy básicos.

1. Los algoritmos son recetas paso-por-paso que identifican claramente las entradas y salidas.
2. Los algoritmos nombran las entidades que son manipuladas o usadas: variables, funciones, etc.
3. Los pasos en el algoritmo se siguen en el orden en que están escritos (de arriba a abajo).
4. Algunos pasos pueden especificar decisiones (si-entonces) sobre la elección de algunos pasos.
5. Algunos pasos pueden especificar repeticiones (loops) de pasos.
6. Todos los de arriba pueden ser combinados de cualquier manera.

Los científicos de la computación proclaman que las soluciones/algoritmos a *cualquier* problema pueden ser expresadas usando las construcciones de arriba. ¡No necesitan más! Esta es una poderosa idea y es lo que hace tan versátiles a las computadoras. Desde una perspectiva más amplia, si esto es verdad, entonces pueden ser usados como herramientas para pensar cualquier problema en el universo. Volveremos sobre esto más adelante en el capítulo.

## Lenguajes de programación

También, como han visto anteriormente al escribir programas Python, los lenguajes de programación (Python, por ejemplo) proveen vías formales para especificar los componentes esenciales de los algoritmos. Por ejemplo, el lenguaje Python provee una manera para asociar valores con variables que ustedes nombren, provee una forma secuencial de codificar los pasos, provee los enunciados condicionales si-entonces (if-then) y también provee las construcciones while y  for para expresar repeticiones. Python también provee modos de definir funciones además de maneras de organizar grupos de funciones relacionadas en librerías o módulos que pueden importar y usar según las necesidades. A modo de ejemplo, abajo les brindamos el programa Python que codifica el algoritmo `esBisiesto` mostrado arriba:

```
def esBisiesto(a):
  '''Retorna true si a es bisiesto y false en caso contrario.'''

  if a %400 == 0:
    return True
  elif a % 100 == 0:
    return False
  elif a % 4 == 0:
    return True
  else:
    return False
```

El mismo algoritmo, cuando es expresado en C++ (o Java) se verá así:

```
bool esBisiesto(int a) {
  //  Retorna true si a es bisiesto y false en caso contrario
  if (a % 400 == 0)
    return true
  else if (a % 100 == 0)
    return false
  else if (a % 4 == 0)
    return true
  else
    return false
}
```

Como pueden ver, hay variaciones sintácticas definidas entre lenguajes de programación. Pero por lo menos en los ejemplos de arriba, la codificación del mismo algoritmo se ve parecida. Sólo para dar otro sabor, aquí está la misma función expresada en el lenguaje de programación CommonLisp.

```
(defun esBisiesto (a)
  (cond
    ((zerop (mod a 400)) t)
    ((zerop (mod a 100)) nil)
    ((zerop (mod a 4)) t)
    (t nil)))
```

Nuevamente, esto puede parecer raro, pero aún está expresando el mismo algoritmo.

Lo más interesante de todo es que, dado un algoritmo, puede haber muchas formas de codificarlo, aún en el mismo lenguaje de programación. Por ejemplo, aquí hay otra manera de escribir la función `esBisiesto` en Python:

```
def esBisiesto(a):
  '''Retorna true si a es bisiesto y false en caso contrario.'''

  if ((a % 4 == 0) and (a % 100 != 0)) or (a % 400 == 0):
```

```
    return True
  else:
    return False
```

Nuevamente, éste es exactamente el mismo algoritmo. Sin embargo, combina todos, los prueba en una sola condición: y es divisible por 4 o por 400 pero no por 400. La misma condición puede ser usada para escribir una versión aún más sucinta:

```
def esBisiesto(a):
        '''Retorna true si a es bisiesto y false en caso contrario.'''

         return ((a % 4 == 0) and (a % 100 != 0)) or (a % 400== 0)
```

Es decir, devolver cual sea el resultado (`True`/`False`) de la prueba para que `a` sea un año bisiesto. En algún sentido, expresar algoritmos en un programa se parece a expresar un pensamiento o un conjunto de ideas en un párrafo o una narración. Puede haber muchas formas de codificar un algoritmo en un lenguaje de programación. Algunos parecen más naturales, y algunos más poéticos, o ambas cosas, y, como en la escritura, algunos pueden ser muy enrevesados. Como en la buena escritura, la habilidad para una buena programación viene de la práctica y, aún más importante, se aprende leyendo programas bien escritos.

## De los algoritmos a un programa en funcionamiento

Para poder resolver el problema de los huevos frescos, se deben codificar todos los algoritmos en funciones Python y luego unirlos como un programa que funciona. Abajo presentamos una versión:

```
# File: huevosFrescos.py

def esBisiesto(a):
        '''Retorna true si a es bisiesto y false en caso contrario.'''
         return ((a % 4 == 0) and (a % 100 != 0)) or (a % 400== 0)


def diasEnMes(m,a):
  '''Retorna la cantidad de días en el mes m, m (1-12)
    ien el año a.'''

  if (m == 4) or (m == 6) or (m == 9) or (m == 11):
    return 30
  elif m == 2:
    if esBisiesto(a):
      return 29
    else:
      return 28
  else:
    return 31

def main():
```

```
  '''Dado un dia del año (ejemplo. 248, 2007),
     lo convierte a la fecha correspodiente (ejemplo. 5/9/2007)'''

  #Entrada: dia, año
  dia, anio = input("Ingrese dia , año: ")

  # Comenzamos con mes = 1, para enero
  mes = 1

  while dia > diasEnMes(mes, anio):
     dia = dia - diasEnMes(mes, anio)

     # Siguiente mes
     mes = mes + 1

  # Listo, Salida: dia/mes/año
  print "La fecha es:  %1d/%1d/%4d" % (dia, mes, anio)

main()
```

Si guardan este programa en un archivo, `huevosFrescos.py`, podrán hacerlo correr y probarlo con varias fechas. Pruébenlo. Aquí hay algunas salidas  de ejemplo:

```
Ingrese dia , año: 248, 2007
La fecha es: 5/9/2007

>>> main()
Ingrese dia , año: 12, 2007
La fecha es:  12/1/2007

>>> main()
Ingrese dia , año:248, 2008
La fecha es:  4/9/2008

>>> main()
Ingrese dia , año: 365, 2007
La fecha es:  31/12/2007

>>> main()
Ingrese dia , año: 31, 2007
La fecha es:  31/1/2007
```

Todo parece andar bien. Observen cómo probamos o ¨testeamos¨ el programa para distintas entradas de valores para confirmar que nuestro programa está produciendo resultados correctos. Es muy importante probar el programa para un variado conjunto de entradas, asegurándose de incluir todas las condiciones límites: primer y último día del año, mes, etc. Probar programas es un arte en sí mismo y varios libros se han escrito sobre el tema. Uno debe asegurarse de que todos las posibles entradas están probadas para asegurarse de que el comportamiento del programa es aceptable y correcto. Ustedes hicieron esto con los programas de robot haciéndolos correr repetidamente y observando el comportamiento del robot. Lo mismo se aplica a la computación.

## Testeo y chequeo de errores

¿Qué pasa si el programa de arriba recibe entradas que están fuera de su alcance? ¿Qué pasa si el usuario ingresa valores hacia atrás (por ej. 2007, 248 en lugar de 248, 2007)? ¿Qué pasa si el usuario ingresa su nombre en lugar de la fecha? (por ejemplo. Paris, Hilton)? Este es el momento de probar todo esto. Prueben el programa y observen su comportamiento con algunos de estas entradas.

Asegurarse de que un programa provea resultados aceptables para todos las entradas es crítico en la mayoría de las aplicaciones. Aunque no hay modo de evitar lo que sucede cuando un usuario ingresa su nombre en lugar de un día y un año, ustedes deberían ser capaces de proteger a sus programas de tales situaciones. Por ejemplo:

```
>>> main()
Ingrese dia , año:  400, 2007
Que corresponde a la fecha, 4/14/2007
```

¡Obviamente, no tenemos un mes 14!

Lo que viene al rescate aquí es darse cuenta de que su programa y la computadora sólo llevarán a cabo lo que ustedes han expresado en el programa. Es decir, pueden incluir facilidades de *chequeo de errores* en su programa para dar cuenta de tales condiciones. En este caso, cualquier valor de entrada de un día que esté fuera del rango 1..365 (o 1..366 para años bisiestos) no será aceptable. Además, también pueden asegurarse de que el programa sólo acepte años mayores que 1582 para el segundo valor de entrada. Aquí está el programa modificado (sólo mostraremos la función `main`):

```
def main():
  '''Dado un dia del año (ejemplo. 248, 2007),
     lo convierte a la fecha correspodiente (ejemplo. 5/9/2007)'''

  #Entrada: dia, año
   dia, anio = input("Ingrese dia , año: ")

   # Validamos los valores ingresados
   if anio <= 1582:
      print "Lo siento. Debés  ingresar un año válido (uno después de 1582). Por favor,
intentalo de nuevo”
      return

   if dia < 1:
      print "Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo de
nuevo”
      return

   if esBisiesto(anio):
      if dia > 366:
```

```
            print "Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo
de nuevo”
          return

  elif dia > 365:
            print "Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo
de nuevo”
       return

  # las entradas son seguras.... seguimos....
  # Comenzamos con mes = 1, para enero
  mes = 1

  while dia > diasEnMes(mes, anio):
    dia = dia - diasEnMes(mes, anio)

    # Siguiente mes
    mes = mes + 1

  # Listo, Salida: dia/mes/año
  print "La fecha es:  %1d/%1d/%4d" % (dia, mes, anio)


main()
```

Aquí están los resultados de algunas de las pruebas sobre el programa de arriba.

```
Ingrese dia , año: 248, 2007
La fecha es: 5/9/2007

>>> main()
Ingrese dia , año: 0, 2007
Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo de nuevo

>>> main()
Ingrese dia , año: 366, 2007
Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo de nuevo

>>> main()
Ingrese dia , año: 400, 2007
Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo de nuevo

>>> main()
Ingrese dia , año: 248, 1492
Lo siento. Debés  ingresar un año válido (uno después de 1582). Por favor, intentalo de
nuevo

>>> main()
Ingrese dia , año: 366, 2008
La fecha es:31/12/2008
```

Empezando de la descripción de un problema, es un viaje largo y cuidadosamente planeado que involucra el desarrollo del algoritmo, la codificación del algoritmo en un programa, y finalmente el testeo y la mejora del programa. Al final son recompensados no sólo con un programa útil, también han afilado sus habilidades para resolver problemas generales. Programar los fuerza a anticipar situaciones

inesperadas y dar cuenta de ellas antes de encontrárselas, que en sí mismo puede ser una maravillosa lección de vida.

## Módulos para organizar componentes

Frecuentemente, durante el diseño de programas, terminan diseñando componentes o funciones que pueden ser usadas en muchas otras situaciones. Por ejemplo, en el problema de arriba, escribimos las funciones `esBisiesto` y `diasEnMes` para asistir en la resolución del problema. Sin duda estarán de acuerdo en que hay muchas situaciones en las cuales estas dos funciones pueden resultar útiles (ver ejercicios abajo). Python provee la facilidad *module* (módulo) para ayudarlos a organizar funciones útiles relacionadas entre sí, en un solo archivo que pueden entonces usar una y otra vez cuando lo necesiten. Por ejemplo, pueden tomar las definiciones de las dos funciones y ponerlas separadas en un archivo llamado `calendario.py`. Entonces, pueden *importar* estas funciones cuando las necesiten. Han usado el enunciado `import` de Python para importar funcionalidades de varios módulos distintos: `myro`, `random`, etc. Bueno, ahora saben crear los propios. Una vez que hayan creado el archivo `calendario.py`, pueden importarlo al programa `huevosFrescos.py` como se muestra abajo:

```
from calendario import *

def main():
  '''Dado un dia del año (ejemplo. 248, 2007),
     lo convierte a la fecha correspodiente (ejemplo. 5/9/2007)'''

  #Entrada: dia, año
   dia, anio = input("Ingrese dia , año: ")


   # Validamos los valores ingresados
   if anio <= 1582:
      print "Lo siento. Debés  ingresar un año válido (uno después de 1582). Por favor,
intentalo de nuevo”
      return

   if dia < 1:
      print "Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo de
nuevo”
      return

   if esBisiesto(anio):
      if dia > 366:
            print "Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo
de nuevo”
         return

   elif dia > 365:
            print "Lo siento. Debés  ingresar un día válido (1..365/366). Por favor, intentalo
de nuevo”
      return

   # las entradas son seguras.... seguimos....
   # Comenzamos con mes = 1, para enero
```

```
  mes = 1

  while dia > diasEnMes(mes, anio):
    dia = dia - diasEnMes(mes, anio)

    # Siguiente mes
    mes = mes + 1

  # Listo, Salida: dia/mes/año
  print "La fecha es:  %1d/%1d/%4d" % (dia, mes, anio)


main()
```

También para este momento se les puede haber ocurrido que para cualquier problema dado hay muchas diferentes soluciones o algoritmos. Ante la presencia de varios algoritmos alternativos, ¿cómo deciden cuál elegir? Los científicos de la computación han convertido en su tarea principal el desarrollar, analizar y clasificar algoritmos para ayudar a tomar estas decisiones. La decisión puede estar basada en facilidad de programación, eficiencia, o el número de recursos que lleva para un algoritmo dado. Esto también llevó a los científicos de la computación a crear una clasificación de problemas: de fácil a difícil, pero en un sentido formal. Algunas de los interrogantes abiertos más difíciles en el terreno de los problemas y la computación entran en este dominio de investigación. No entraremos en detalles aquí, pero estas cuestiones incluso han aparecido en varios programas populares de TV (ver foto arriba). Intentaremos darles un sabor acerca de esto a continuación.

**Homero contempla ¿P = NP?**


La segunda ecuación a la derecha es la ecuación de Euler.

## La Complejidad del Tiempo & Espacio

Empecemos con otro problema: Deben viajar del Estado de Washington a Washington DC en los Estados Unidos de América. Para hacer las cosas más interesantes, agreguemos la restricción de que pueden viajar solamente a través de los estados cuyos nombres empiecen con las letras de la palabra ¨mujer¨ (se trabajará con la palabra en ingles: ¨woman¨). Es decir, está bien ir de Washington a Oregon dado que ambos ¨W¨ y ¨O¨ están en la palabra ¨woman¨, pero no está bien ir de Washington a California. ¿Es posible esto? Si lo es, ¿cuántos posibles caminos hay? ¿Cuál de ellos atraviesa la menor/mayor cantidad de estados? Etc.

Si están pensando en cómo resolverlo, dependerán de su conocimiento geográfico sobre los Estados Unidos. Como alternativa, pueden googlear un mapa con los

estados y entonces llegar a la solución. Pero al hacerlo, han tropezado con los dos ingredientes claves de la computación: datos y algoritmo.

Los datos les dan una representación de la información relevante, que en este caso es una forma de conocer cuáles estados son lindantes entre sí. El algoritmo les da una manera de hallar la solución: empezar en Washington, mirar a sus vecinos. Elegir un vecino cuyo nombre satisfaga la restricción, luego mirar el vecino de ese estado, y así sucesivamente. Además, un algoritmo los obliga a hacer determinadas elecciones: si hay más de un vecino elegible, ¿cuál eligen? ¿Qué pasa si terminan en un callejón sin salida?, ¿cómo vuelven a las elecciones que dejaron atrás para explorar pasos alternativos? Etc. Dependiendo de estas decisiones, es probable que terminen con varios algoritmos distintos: uno puede sugerir explorar todas las alternativas simultáneamente; o una que los obliga a elegir, sólo para volver a las otras elecciones en el caso de fracaso. Cada uno de estos impactarán luego en la cantidad de datos que necesitarán guardar para mantener un registro de su progreso.

Desarrollar programas de computadora para resolver cualquier problema requiere que uno diseñe representaciones de datos y elija entre un conjunto de algoritmos alternativos. Los científicos de la computación caracterizan las elecciones de representaciones de datos y algoritmos abstractamente en términos de recursos de *espacio* y *tiempo* de la computadora, necesarios para implementar esas elecciones. Las soluciones varían en términos de la cantidad de espacio (o memoria de la computadora) y tiempo (segundos, minutos, horas, días, años) requeridos en una computadora. Aquí hay algunos ejemplos:

1. Computar el producto de dos números requiere tiempo constante: para una computadora no hay diferencia entre multiplicar 5 por 2 o 5.564.198 por 9.342.100. Estos se llaman *algoritmos de tiempo constante*.

2. Encontrar un número en una lista de *N* números desordenados lleva el tiempo proporcional a ***N***, especialmente si el número que están buscando no está allí. Estos se llaman *algoritmos de tiempo lineal.*

3. Encontrar un número en una lista de N números ordenados (digamos, en orden ascendente) requiere como máximo el $\mathbf{log_2\ N}$ de tiempo. ¿Cómo? Tales algoritmos se llaman algoritmos de tiempo logarítmico.

4. Transformar una imagen de cámara de *NxN* píxeles manipulando todos sus píxeles lleva un tiempo proporcional a $\mathbf{N^2}$. Estos son *algoritmos cuadráticos.*

5. Encontrar un camino de un estado a otro, en un mapa de N estados, dadas ciertas restricciones, pueden llevar un tiempo proporcional a $\mathbf{b^d}$ donde *b* es la cantidad promedio de vecinos de cada estado y *d* es la cantidad de estados que conforman la solución. Generalmente, muchos problemas entran en la categoría donde $N^k$ es el tamaño del problema. Estos se llaman *algoritmos de tiempo polinómico*.

6. También hay varios problemas irresolubles en el mundo.

En el Capítulo 9, al hacer transformaciones de imágenes, nos restringimos a imágenes de tamaños relativamente pequeños. Habrán notado que cuanto más grande es la imagen, más tiempo tarda en transformarse. Esto tiene que ver con la velocidad de la computadora que están usando, así como con la velocidad de implementación del lenguaje de programación. Por ejemplo, una computadora que corre a una velocidad de 4GHz es capaz de hacer aproximadamente 1 billón de operaciones aritméticas básicas en un segundo (o  operaciones/segundo). Un programa escrito en el lenguaje de programación C les podría llegar a dar una velocidad de ½ billón de operaciones por segundo en la misma computadora. Esto se debe a las operaciones extra requeridas para implementar el lenguaje C y las tareas adicionales que el sistema operativo está llevando a cabo detrás. Los programas Python corren aproximadamente 20 veces más lento que los programas C. Es decir, un programa Python corriendo en la misma computadora les podría llegar a dar 25 millones de operaciones por segundo como mucho. Una transformación típica, digamos computar el negativo de una imagen de `WxH` píxeles requeriría el siguiente loop:

```
for x in range(W):

   for y in range(H):

      pixel = getPixel(myPic, x, y)
      r, g, b = getRGB(pixel)
      setRGB(pixel, (255-r,255-g,255-b))
```

Si la imagen es de 1000x1000 píxeles (por ej. W=1000 y H=1000), cada uno de las sentencias en el loop se ejecuta 1 millón de veces. Cada uno de esas sentencias a su vez requiere un promedio de 8-16 operaciones de bajo nivel de computación: los cálculos actuales, más los cálculos para ubicar los píxeles en la memoria, etc. Por lo tanto, la transformación de arriba requeriría arriba de 24-48 millones de operaciones. Como pueden imaginar, llevará unos segundos completar esa tarea.

En la industria de la computación, las velocidades de cómputo se clasifican en base a cálculos de referencia oficiales que calculan la velocidad en términos de la cantidad de operaciones de punto flotante por segundo, o *flops*. Una típica laptop en estos días es capaz de llegar a velocidades de entre ½ a 1 Gflops (Giga flops). La computadora más rápida del mudo puede llegar a velocidades de cómputo de 500 Tflops (Terra flops). Es decir, es alrededor de un millón de veces más rápida (y cuesta muchos millones construirla también). Sin embargo, si se detienen a pensar sobre el programa de juego de ajedrez que mencionamos en el Capítulo 10 que requeriría aproximadamente  $10^{65}$ operaciones antes de realizar una sola jugada, ¡incluso a la computadora más rápida le llevará gazillion de años terminar ese cómputo! Tal problema sería considerado *incomputable*.

Los científicos de la computación han desarrollado un vocabulario elaborado para discutir problemas y clasificarlos como *solubles o*, *computables o incomputables,* basados en si hay modelos conocidos para resolver un problema dado o si los modelos son solucionables, computables, etc. También hay jerarquías de solución de problemas, de simple a complejo; tiempo constante a tiempo polinómico y más

largos; y clases de equivalencias, que implican que el mismo algoritmo puede resolver todos los problemas en una clase, etc. ¡Es realmente sorprendente conceptuar un algoritmo que sea capaz de resolver problemas no relacionados! Por ejemplo, los algoritmos que optimizan rutas de envíos también pueden ser usados para determinar las estructuras de plegamiento de proteínas de las moléculas de ADN. Esto es lo que hace a la ciencia de la computación intelectualmente interesante y emocionante.

## Resumen

En este capítulo hemos unido varias ideas fundamentales sobre la computación y la ciencia de la computación. Nuestro viaje se inició en el Capítulo 1 jugando con los robots personales y en el proceso, hemos adquirido un bagaje de conceptos fundamentales de computación. Como pueden ver, la computación es una disciplina de estudio profundamente intelectual que tiene implicancias en todos los aspectos de nuestras vidas. Iniciamos este capítulo señalando que ahora hay más computadoras en un típico campus universitario en Estados Unidos que cantidad de gente. Seguramente no faltará mucho tiempo para que haya más computadoras que personas en todo el planeta. Sin embargo, la idea de *algoritmo* que es esencial en computación, apenas es comprendida por la mayoría de  los usuarios a pesar de que está constituida por elementos simples e intuitivos. Muchos científicos de la computación creen que aún estamos sentados en la madrugada de la *era del algoritmo* y de que hay beneficios sociales e intelectuales mucho mayores por ser realizados. Especialmente si más gente toma conciencia de estas ideas.

## Revisión Myro

No se introdujeron nuevas características Myro en este capítulo.

## Revisión Python

La única característica Python introducida en este capítulo fue la creación de módulos. Cada programa que crean puede ser usado como un módulo de librería desde el cual pueden importar facilidades muy útiles.

## Ejercicios

1. Para computar la cantidad de días en un mes, hemos usado lo siguiente:

```
def diasEnMes(m, a):
  '''Retorna la cantidad de dias en el mes m (1-12) en el año, a.'''

  if (m == 4) or (m == 6) or (m == 9) or (m == 11):
    return 30
  elif m == 2:
    if esBisiesto(a):
      return 29
    else:
      return 28
  else:
```

```
    return 31
```

Pueden simplificar más la escritura de la condición en el enunciado-if usando listas:

```
if m in [4, 6, 9, 11]:
   return 30
…
```

Reescriban la función usando la condición de arriba.

2. Definan una función llamada `valido(dia, mes, anio)` para que devuelva true (verdadero) si el `dia`, `mes` y `anio` conforman una fecha válida como se define en este capítulo. Usen la función para reescribir el programa como se muestra a continuación:

```
def main():
  '''Dado un dia del año (ejemplo. 248, 2007),
     lo convierte a la fecha correspodiente (ejemplo. 5/9/2007)'''

  #Entrada: dia, año
   dia, anio = input("Ingrese dia , año: ")


   # Validamos los valores ingresados
   if not valido(dia, mes, anio):
         print "Por favor,ingresa una fecha válida”
         return

   # las entradas son seguras.... seguimos....
   # Comenzamos con mes = 1, para enero
   mes = 1

   while dia > diasEnMes(mes, anio):
     dia = dia - diasEnMes(mes, anio)

     # Siguiente mes
     mes = mes + 1

   # Listo, Salida: dia/mes/año
   print "La fecha es:  %1d/%1d/%4d" % (dia, mes, anio)


main()
```

Reescribir el programa como se muestra arriba para usar la función nueva. Además de desarrollar un algoritmo correcto también es importante escribir programas de manera que sean más leíbles por ustedes.

**3.** Averigüen cuán rápida es su computadora observando el reloj de velocidad de la CPU. Basados en esto, estimen cuántas operaciones aritméticas será capaz de

realizar en 1 segundo. Escriban un programa para ver cuántas operaciones realmente realiza en un segundo.

**4.** Realicen una búsqueda Web para “Chazelle age of algorithm” (Chazelle era del algoritmo). Serán recompensados con un link a un ensayo escrito por el Prof. Chazelle. Lean el ensayo y escriban un breve comentario sobre el mismo.

**5.** ¿Cuál es la computadora más rápida en uso actualmente? ¿Pueden averiguarlo? ¿Cuán rápida es comparada con la computadora que ustedes usan? ¿100 veces? ¿1000 veces? ¿100.000 veces?

**6.** ¿Qué es/fue el problema “Y2K”?

# 12 Rápido, Barato & Fuera de Control

*Sugerimos que dentro de pocos años será posible a un costo modesto, invadir un planeta con millones de pequeños robots… Con imaginación y voluntad podemos invadir el sistema solar entero.*

De un artículo titulado,
*Rápido, barato y fuera de control: Una invasión robot al sistema solar,*
Rodney Brooks y Anita M. Flynn,
Journal of the British Interplanetary Society, Volumen 42, pp 478-485, 1989.

Página opuesta: Enjambres de Orbes
Usada con permiso, Copyright Phil Spitler (www.orbswarm.com)

Hace casi dos décadas, Brooks y Flynn expusieron una visión radical para el futuro que, el 4 de julio de 1997, resultó en un exitoso aterrizaje del rover *Sojourner* en la superficie de Marte. Unos años más tarde, en 2004, dos rovers, *Spirit* y *Opportunity,* llegaron a Marte para explorar el planeta. En 2002, en su libro, *Flesh and Machines (Carne y Máquinas)*, un Rodney Brooks más reflexivo confiesa su exuberancia en los comentarios de su artículo escrito con Anita Flynn y describe a los rovers como embajadores planetarios. La visión de Brooks and Flynn consistía en reemplazar a un único y gran rover de 1000 kilos por 100 rovers más pequeños de 1 kilo para explorar superficies planetarias. Los robots más pequeños se moverían mucho más rápido y podrían ser construidos de manera más barata y producidos en masa. También ofrecerían la necesaria redundancia en el caso de que ocasionalmente fallara alguno, que en el caso de un solo robot conduciría a un total fracaso de la misión. Es más, cada robot sería autónomo (es decir, fuera del control humano), y operaría con su propia agenda, tal como estaría definida en su programa de control.

Se debate si Brooks efectivamente acuñó la frase *rápido, barato y fuera de control*. Pero el crédito por usarlo en el título de su artículo de referencia seguido por el despliegue de tres rovers en Marte llevó a su uso masivo en la cultura popular. La frase se convirtió en el título de una película documental de 1997 de Errol Morris en la que aparecía el propio Brooks. En su libro de 2002, Brooks cuenta cómo un científico en el laboratorio de la NASA de Jet Propulsión criticó la idea de rovers rápidos, baratos y autónomos. Mientras que el sueño original de 100 robots pequeños y autónomos queda aún por realizarse, hoy nos resultaría difícil argumentar en contra de la idea. Hemos usado la frase de Brooks en el título de este capítulo para sugerir que a menos de veinte años de que se acuñó la frase, estamos sumergidos en la exploración de todo el potencial de los robots pequeños, baratos y personales. Dado que varios millones de robots personales ya han sido vendidos, podríamos concluir que nuestro planeta ha sido invadido por ellos. Brooks dice que ¨la revolución de la robótica está en una fase inicial, lista para estallar en los albores del siglo veintiuno¨. En este capítulo les presentamos varios ejemplos de las formas en que los robots están pasando a formar parte de nuestra vida cotidiana.

## Los robots son mecanismos guiados por un control automatizado

Aceptamos la definición de arriba para los robots en el inicio de este texto. Usando el robot Scribbler, también hemos aprendido mucho acerca de sus mecanismos: sensores y motores, y cómo controlarlos va través de los programas Python. En el transcurso de este viaje hemos aprendido lecciones valiosas sobre la construcción de distintos cerebros de robots, cómo diseñar comportamientos simil-insectos, crear sonidos, imágenes y también nos aventuramos brevemente en el reino de la Inteligencia Artificial. La computación tiene su centro en la definición del control automatizado. Cualquier robot, no importa cuán grande o chico sea, tiene mecanismos sensores y motores y su comportamiento puede ser programado para que pueda actuar con autonomía en su ambiente. Es importante tener esto en

mente a medida que exploramos las variadas dimensiones del espectro de la robótica.

El ámbito de aplicaciones de los robots es casi tan diverso como el mundo en el que vivimos. De hecho, está limitado sólo por nuestra propia imaginación. A través de este texto, hemos presentado ejemplos de robots en varios dominios: rovers planetarios, aspiradoras, exploradores en ambientes peligrosos, juguetes y entretenimientos, educación, cirugía médica, manufacturas, etc. Abajo, presentamos otros casos interesantes de uso de la tecnología de robots. Será importante, a medida que exploramos algunos de estos ejemplos, intentar usar lo que han comprendido de este texto para averiguar qué mecanismos se están utilizando en cada caso. Traten de pensar en otras posibles áreas en las que se podrían usar mecanismos similares.

## Juguetes

Los juguetes robóticos simples están en todas partes. Algunos efectivamente parecen y se comportan como robots, pero la mayoría usa la computación y la tecnología robótica de forma simple e innovadora. Tomemos el ejemplo de Tengu, diseñado por Crispin Jones (tengutengutengu.com). Tengu se enchufa al puerto USB. Es capaz de desplegar más de una docena de formas con la boca. Está diseñado para reaccionar al sonido y a la música en su ambiente. Las expresiones faciales de boca cambian dependiendo de los sonidos que escucha. Si cantan una canción o si suena alguna música parecerá hacer la mímica con sus labios.

Los juegos robóticos de escritorio han crecido en popularidad por varias razones. La tecnología que lo permite es la presencia de computadoras con puertos USB. Los puertos USB son únicos por el hecho de que, además de proveer canales para el intercambio de datos (como hacemos al comunicarnos con el Scribbler), también proveen potencia. Muchos juguetes, como el Tengu, usan el USB sólo por la potencia. Todos los controles están construidos dentro de la unidad Tengo.


Algunos juguetes de escritorio pueden funcionar en forma independiente de una computadora (todos los controles están presentes dentro de la unidad en sí misma) y sin embargo no requieren baterías: funcionan con energía solar. Las flores y plantas Flip Flap creadas por la compañía TOMY (www.tomy.com) son buenos ejemplos de estos casos. Estos juegos incorporan mecanismos muy simples: sensores solares y generación de

potencia en conjunto con pequeños motores que se activan con corriente eléctrica. Son simples, y sin embargo ingeniosas y divertidas criaturas Braitenberg. La compañía TOMY tiene toda una línea de productos que utiliza estas ideas.

También hay cantidades de juguetes robóticos de escritorio que operan con baterías. El Facebank diseñado por Takada “come” una moneda cuando la pasas frente a sus ojos. Funciona con baterías y su mecanismo incluye censores IR, similares a los del Scribbler y un motor que lleva o empuja la ¨piel¨ de la cara desde atrás.

La mayoría de los juguetes electrónicos educativos usan mecanismos de control automatizado programado. Un juguete educativo interesante es la Tag Reading Pen creada por Leapfrog (leapfrog.com/tag). Un chico puede usar la lapicera para señalar palabras o texto en libros de cuentos especialmente creados para ésta, y la lapicera dice en voz alta la palabra o la oración completa. Diseñada para chicos en edad preescolar que recién se están empezando a interesar por la lectura, este juguete puede promover las habilidades de lectura y pronunciación. La lapicera tiene un lector óptico y un parlante unidos a una memoria que registra los patrones de lectura del chico. Los padres pueden enchufar la lapicera a una computadora para descargar y registrar el progreso de su hijo.

Los robots no necesitan ser construidos con mecanismos digitales. Uno también puede crear mecanismos de control usando circuitos analógicos. La foto a la derecha muestra dispositivos llamados Thingamapoops (bleeplabs.com). ¿Qué hacen? Pueden ser usados para producir o sintetizar bips que suenan muy locos. Los bips también pueden ser enviados como entrada para instrumentos musicales estándar para crear aún más sonidos y efectos bizarros.

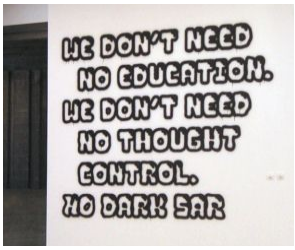
**¿Por qué crean esta cosa?**
Porque no hay suficientes monstruos sintetizadores antropomórficos, bipeadores, que hacen zapping y bixxerfoupeadores en el mundo.
From: FAQ on bleeplabs.com

## Arte

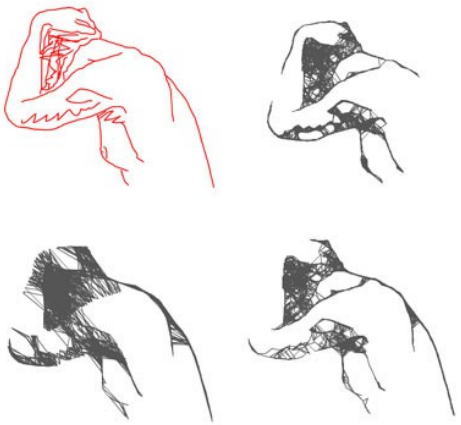
Los robots han sido activamente utilizados para crear todo tipo de arte y lo robots en sí mismos han sido objeto del arte. Existen varias organizaciones a nivel mundial que están dedicadas a crear arte con robots y dispositivos  robóticos. En este libro también han experimentado con los dibujos del Scribbler. Un par de lindos ejemplos

de robots creando arte se ilustran en los trabajos de los ingenieros suizos Jürg Lehni y Uli Franke quienes crearon el robot que dibuja graffitis, Hektor (hector.ch), en la Escuela de Arte en Lousanne, y Zefrank (zefrank.com) que ha creado dos versiones de un robot que llama Scribbler que es diferente del que ustedes tienen.

Ambos Hektor y Scribbler crean dibujos basados en dibujos existentes. Primero se crea un dibujo. El robot (programa) lee el dibujo y después lo embellece con un nuevo dibujo basado en el que se ingresó. Hektor está montado en la pared y tiene una lata de aerosol que se mueve sobre un sistema de poleas controladas por el robot. El graffiti que se muestra acá fue pintado por Hektor. Pueden visitar el sitio Web de Lehni y Franke para ver películas del robot en acción. Scribbler usa esquemas básicos como base para crear dibujos. En la imagen que se muestra aquí, se pueden ver tres esquemas que fueron creados basados en el de la equina superior izquierda. El concepto del Scribbler es interesante por el hecho de que cualquiera puede usarlo a través de la Web para crear dibujos en un navegador Web. Los creadores también han construido un robot físico que hace dibujos.




**Realizar la siguiente actividad:** Escribir un programa que copie o escanee una imagen y logre crear un dibujo gráfico basado en ella. Lean los detalles provistos en el sitio Web del Scribbler y utilícenlos para crear algunos esquemas. Observen el proceso de dibujado y piensen cómo podrían crear algo similar.



## Mostrame el camino

Los sistemas de posicionamiento global (GPS) han estado en uso por muchos años ya. Más recientemente, pequeños dispositivos de GPS portátiles comenzaron a estar disponibles para uso en autos de consumidores. Un GPS les permite ingresar un destino o punto de interés y luego les muestra el camino a tomar. Les provee una guía de giro-por-giro-en-tiempo-real basada en un mapa. La tecnología usada en estos dispositivos utiliza mecanismos que involucran señales satelitales y mapas de calles. Estos dispositivos también forman el corazón de los mecanismos de los pilotos automáticos en un avión. Un avión que opera en modo piloto automático puede ser considerado un robot según nuestra definición. Más del 90% de despegues y aterrizajes de vuelos comerciales en estos días se realizan usando sistemas de piloto automático. También habrán escuchado hablar de aviones teledirigidos de vigilancia robótica que usa el ejército para espiar territorios enemigos. Los misiles automáticos usan dispositivos similares. Hoy en día la

tecnología también existe para crear sistemas de piloto automático en autos. Es decir, es posible que su auto se maneje solo hasta donde ustedes quieren que vaya.

En algunas de estas aplicaciones, la cuestión de la implementación de la tecnología deviene en una cuestión social o ética: ¿Realmente queremos esto o no? ¿Qué implicancias tendría?

## Robots afectuosos & Sociales


En el Capítulo 10 mencionamos que uno de los desafíos de la investigación en IA es comprender y/o crear artificialmente inteligencia corporeizada de nivel humano. Muchos investigadores en IA trabajan en el avance de la comprensión de la inteligencia humana, mientras que otros trabajan en la construcción de modelos de comportamiento más inteligentes. A su vez, el campo de la robótica en sí misma se mueve rápidamente en dirección hacia robots más capaces, ágiles y símil-humanos. Una forma buena y divertida de encontrar la convergencia de estos avances puede verse en las metas de la RoboCup (RoboCopa) (www.robotcup.org). La organización RoboCup está poniendo el foco en robots que juegan al fútbol como base de prueba para IA y robótica. Organizan competiciones anuales de fútbol jugado por robots que, además de humanoides de dos piernas, incluyen jugadores de fútbol de cuatro piernas y otros con ruedas.

Además de robots que juegan al fútbol, otra área de investigación en IA y robótica que está adquiriendo atención es la Interacción Humano-Robot (HRI). Como el nombre lo sugiere, es un área de investigación que estudia modelos de interacción entre humanos y robots. Dado que ya tenemos millones de robots entre nosotros, es importante reconocer la necesidad de crear formas más amigables de interacción con los robots.

Aunque opinamos que todo ciudadano debe estar bien versado en programación y computación, reconocemos que no estamos para nada cerca de esa meta. Como mencionamos varias veces antes, pronto habrá más computadoras que personas en este planeta. ¿Quizás también robots? De todas maneras, la necesidad de interacciones más amigables con las computadoras siempre ha sido reconocida. Con el rápido incremento de aplicaciones basadas en robots, será aún más imperativo para los robots, especialmente dada su presencia física, tener rasgos de comportamiento socialmente relevantes. Aunque fuera sólo por esta razón, facilitaría la aceptación por parte de las personas en nuestra sociedad en muchos niveles.

Dentro de la HRI los investigadores estudian modelos de emoción, reconocimiento de gestos y otras modalidades sociales. Los juguetes emotivos se han estudiado en

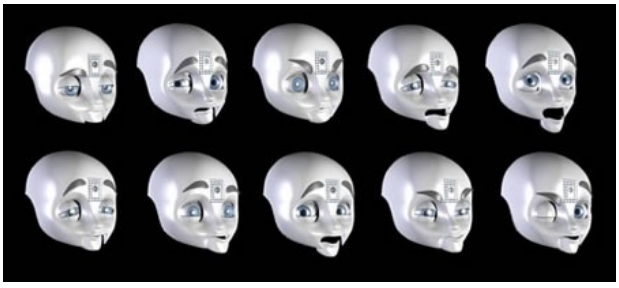
muchas aplicaciones, que van desde los juguetes hasta la terapia médica (mencionamos la foca Paro en el Capítulo 1).

Dado el interés y los avances recientes en el área de HRI, ha emergido un nuevo campo de investigación: *la robótica social*. La robótica social estudia modelos que tienen en cuenta normas sociales de comportamiento relevantes para el ambiente y/o la aplicación del robot. Las emociones pueden cumplir un rol importante en esta área de investigación. Dentro de la relativamente pequeña comunidad de investigadores hay mucho debate acerca de si la robótica social requiere que uno tenga un robot físico. Agentes simulados actuando socialmente son aceptables para algunos investigadores pero no para otros. Tomemos por ejemplo el agente robótico simulado Ananova (www.ananova.com). Ananova fue diseñado para dar noticias a través de la Web igual que un conductor de noticias lo haría en TV. Aunque no es un robot físicamente corporeizado, tiene simulada la morfología humana y es capaz de usar los mismos modelos de emoción y expresión que se han diseñado para robots corporeizados. De hecho, dadas las limitaciones físicas de los robots, la simulación es mucho más realista (ver imagen a la derecha).

iCat es una plataforma de investigación para el estudio de temas de interacción robots-humanos. El robot tiene 38 cm. de alto y está equipado con 13 servos que controlan diferentes partes de la cara, como las cejas, los ojos, los párpados, la boca y la posición de la cabeza. Con esto, el iCat puede generar muchas expresiones faciales diferentes – felicidad, sorpresa, enojo, triste- necesarios para crear diálogos de interacción social humana-robot.

Fuente: research.philips.com

No queremos dejarlos con la foto del iCat como la imagen representativa de robots físicos capaces de manifestar expresiones emocionales. Hacia fines de 1990, Cynthia Breazeal y Brian Scassellati desarrollaron un robot sociable, Kismet, para explorar expresiones e interacciones humanos-robots. Ambos, Breazeal y Scassellati, fueron estudiantes de Rodney Brooks y han participado activamente en la investigación HRI. Los esfuerzos actuales de Breazeal con relación a la computación afectiva se basan en el robot Nexi (ver imagen abajo), que fue desarrollado en el MIT en colaboración con la Universidad de Massachusetts y otros socios colaboradores.



Un nuevo robot experimental del Laboratorio de Medios del MIT puede inclinar sus cejas con enojo o arquearlas en sorpresa, y mostrar un amplio abanico de expresiones faciales para comunicarse con personas en términos centrados en humanos. Se llama Nexi, y apunta a un ámbito de aplicaciones para robots personales y trabajo en equipo humano-robot. Fuente: David Chandler, Oficina de Noticias del MIT, abril 2008.

Brian Scassellati está desarrollando un robot humanoide en la Universidad de Yale que tiene aproximadamente el tamaño de un niño de 1 año. Se está utilizando el robot para explorar tareas más fundamentales de desarrollo como la coordinación mano-ojo. También está siendo utilizado para mejorar el diagnóstico de autismo en niños. Brian y sus colegas sostienen que las señales que el robot necesita detectar y aprender son las mismas que son deficientes en niños autistas. También piensan que tales robots pueden ser usados para crear modelos funcionales de comportamiento autístico.

Los robots autónomos ya se utilizan para repartir el correo diario en oficinas grandes. Fácilmente pueden imaginarse que la misma tecnología se está usando para construir máquinas expendedoras itinerantes. Uno ya puede configurar una cafetera Express de oficia para hacer el café basado en las preferencias de cada individuo: la máquina los percibe (o algún dispositivo que esté utilizando) lo cual transmite sus gustos a la máquina y empieza a funcionar. Las posibilidades son infinitas. Aquí concluimos con otra nueva aplicación: Aquí hay una descripción de problema (fuente: www.koert.com/work): En el diario de la mañana puedo leer el reporte meteorológico así como las cotizaciones de la bolsa. Pero al mirar por mi ventana, sólo obtengo una actualización del clima pero nada de información sobre la bolsa. ¿Alguien podría por favor arreglar este *bug* en mi sistema ambiental? ¡Gracias!
La solución: Fuentes de Datos.

Diseñado por Koert van Minsvoort, Charles Mignot y sus colegas, la Data Fountain usa el medioambiente para mostrar datos de tiempo real. Ellos diseñaron el sistema para mapear los tipos de cambio a la altura de las fuentes. Llaman a esta idea decoración con información. No necesitan restringirse a pantallas de computadoras para visualizar información. La efectiva realización del concepto emplea la robótica o tecnología de control de dispositivos que obtiene datos actuales en tiempo real de la Web y luego la transforma en la actuación de los chorros de las fuentes.

## Resumen

Los robots han sido usados para limpiar sustancias peligrosas, remoción de minas, e incluso destapar profundos sistemas cloacales en grandes ciudades.
Crecientemente se encuentran aplicaciones para situaciones inusuales: exploración de pequeños pasajes en pirámides antiguas, cirugía médica, e incluso en granjas.
Los investigadores están desarrollando vehículos de agricultura inteligentes para

monitorear cosechas, pestes de plantas, y condiciones generales de crecimiento. Mientras que puede resultar aceptable e incluso deseable tener un robot que pase la aspiradora en la casa e incluso que limpie las calles, uno debe detenerse y preguntarse si la misma tecnología puede ser usada para fines destructivos. Invariablemente, como sucede con cualquier tecnología nueva, hay beneficios así como potencial para mal uso. Cuestiones éticas y sociales deben entrar en consideración para tomar decisiones importantes en todas las circunstancias.

Los robots se están poniendo rápidos, pequeños, baratos y autónomos. Sin embargo, es sólo el sentido de la autonomía que hace deseable que estén fuera de control. No habremos enviado cientos de pequeños robots a otro planeta pero sí parecemos rodeados por millones de ellos aquí en la tierra. La foto de robots tipo-orbe en un paisaje viene del proyecto Orb Swarm (orbswarm.com). Están explorando usar enjambres de estos robots que coordinan, exploran, e interactúan con su medio ambiente, similar a la visión de Brooks y Flynn. Si tienen éxito en el envío de ejércitos de robots tipo-orbe a otro planeta está por verse. Aunque una cosa es segura: ¡Aprender computación con un robot personal hace que sea mucho más divertido!