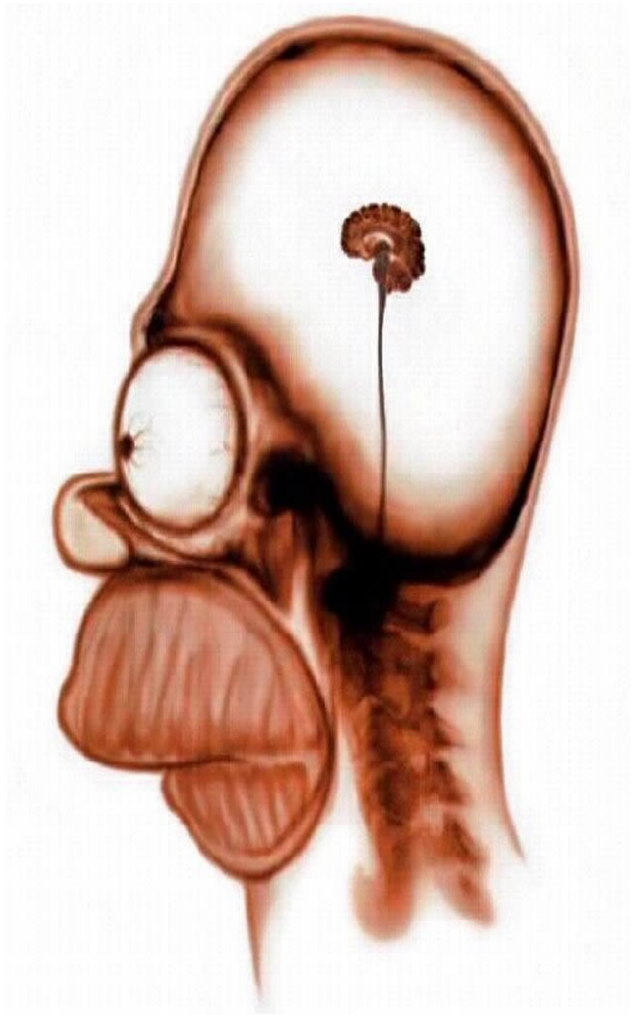
3

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

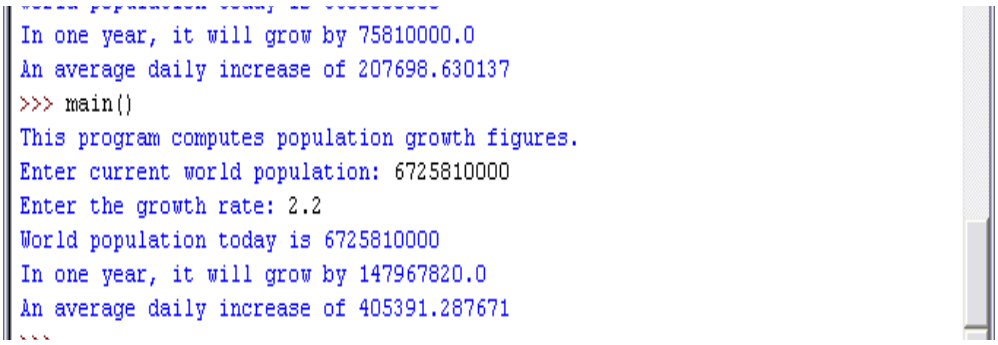
simple que puede ser usado para reescribir el programa encima, como se muestra a continuación:

```
#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).