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:

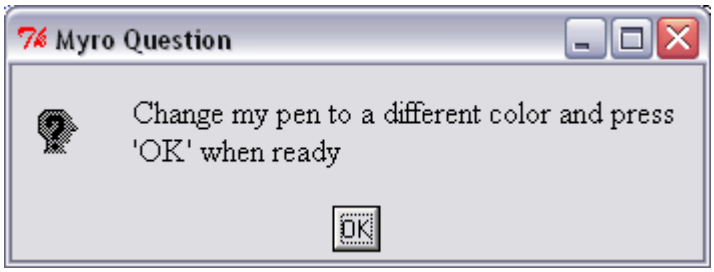
```
>>> askQuestion("Estás listo?")
'Yes'
```

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")
```

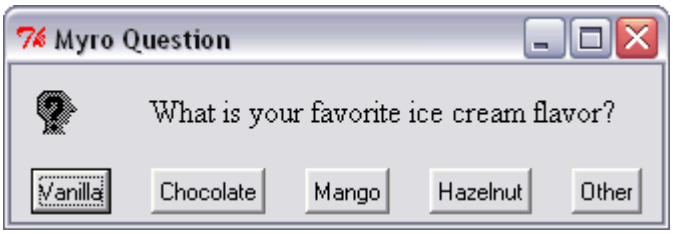
Es definitivamente más funcional que nuestra solución previa, pero podemos hacerlo aún mejor. Por ejemplo, ¿qué ocurre cuando el usuario presiona el botón *No* en la interacción de arriba? Algo que saben con certeza es que la función devolverá *la cadena 'No'*. Sin embargo, por el modo en que estamos 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.