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.

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.

¿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

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:

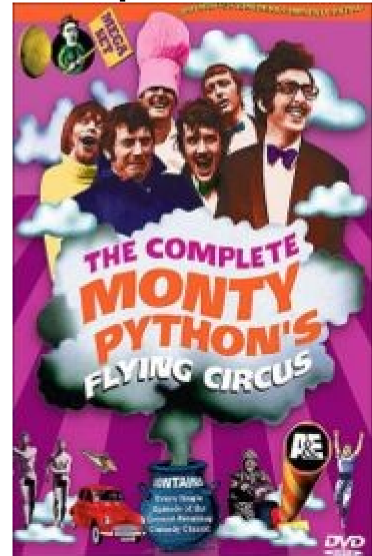
```
wait(SECONDS)
```

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`:

```
Python 2.4.4 (#71, Oct 18 2006, 08:34:43) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.1.4
>>> from moves import *
Myro, (c) 2006 Institute for Personal Robots in Education
[See http://www.roboteducation.org/ for more information]
Version 1.0.0, Revision 1.85, ready!
Waking robot from sleep...
Hello, I'm Shrek!
>>> yoyo(0.5, 0.5)
>>> 
```

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].