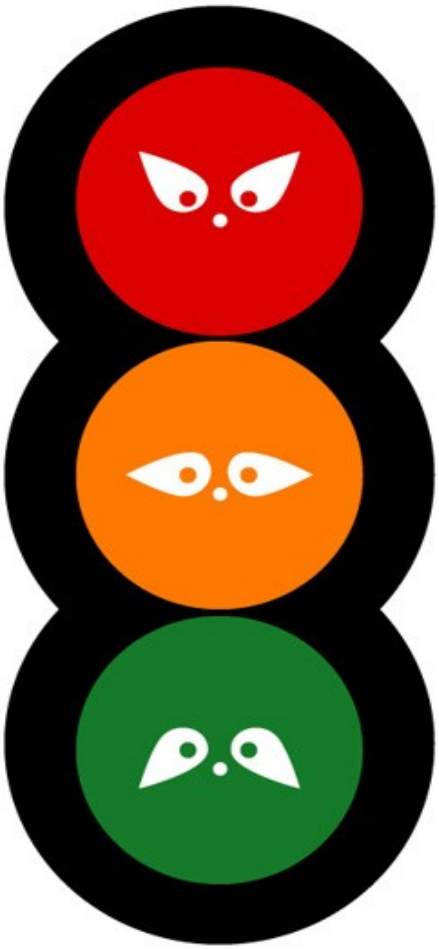
7 Control de Comportamiento



!, Comórtate!
Austin Powers (interpretado por Mike Myers) en la pelíula
Austin Powers: Hombre Internacional del Misterio,
New Line Cinema, 1997

Página opuesta: Mis luces de tráfico
Ilustración de Yingshun Wong (yingshun.co.uk)

Escribir programas es ejercitar el control. En es caso del cerebro de un robot, el programa dirige las operaciones del robot. Sin embargo, también es importante observar que el programa en sí mismo está realmente controlando la computadora. Es decir, cuando escriben programas Python están controlando la computadora que entonces se está comunicando con el robot. Si quitan a Myro de la escena están escribiendo programas para controlar la computadora. En este sentido, el aprendizaje con robots también lleva a aprender computación. Cada programa que escriben está haciendo computación. Contrariamente a los preconceptos populares acerca de la computación, ésta no se trata solamente de realizar ecuaciones con números. Controlar el robot es también realizar computación, como también lo es predecir la población mundial, componer una imagen, etc. Este es un aspecto del control.

Al escribir programas para controlar los robots, la estructura que utilizan para organizar el programa es una estrategia de control. Programar un robot significa especificar controles automatizados. Como programadores o diseñadores de comportamiento, ustedes estructuran su programa para lograr las metas de ese comportamiento: cómo se utilizan los sensores para decidir qué hacer a continuación. Este es otro aspecto del control. Hasta ahora, han visto cómo escribir programas de control usando el estilo de cableado sensor-motor de Braitenberg. También han visto como especificar el control reactivo. Estos son ejemplos de dos paradigmas de control de robot.

En este capítulo ahondamos más en el mundo de la computación y los paradigmas de control de robots. Aprenderemos cómo escribir programas de control de robot para realizar tareas más complejas y más robustas. También veremos cómo, usando los conceptos aprendidos hasta ahora, podemos escribir aplicaciones de computadora útiles e interesantes.

Control basado en el comportamiento

Al escribir programas de control de robots, hasta ahora, han usando una técnica muy básica para diseñar programas de control:

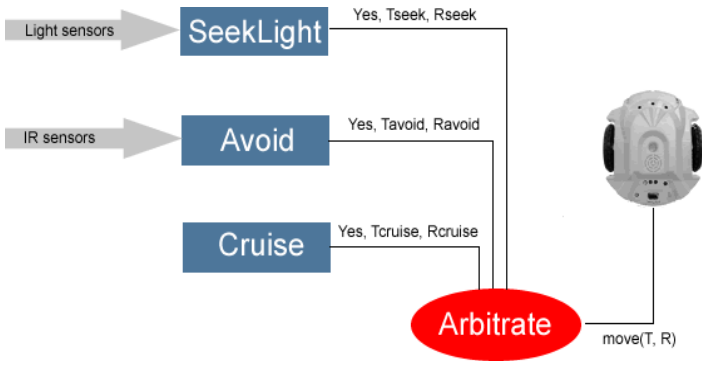
```
def main():
    # hacer por siempre o por algún tiempo
    # o hasta que se cumpla alguna condición

        # sensar y transformar los valores de los sensores
        # razonar y decidir qué hacer
        # hacerlo!
```

Como habrán observado, tales programas de control funcionan bien para tareas simples. Sin embargo, posiblemente ya se han encontrado con situaciones en las que, una vez que la tarea se pone un poco más compleja, resulta difícil estructurar un programa en términos de una línea de control como se muestra arriba. Por ejemplo, el comportamiento de salida de corral del capítulo anterior les requiere combinar dos comportamientos simples: resolver un laberinto (evitar obstáculos) y buscar la luz para salir del corral. Como han visto antes, es relativamente sencillo programar cada uno de los comportamientos individuales: evitar obstáculos; seguir la luz. Pero al combinar estos comportamientos para lograr la salida del corral, dos

cosas ocurren: se ven forzados a combinar las dos estrategias de control en una sola, y puede resultar difícil decidir de qué manera hacerlo. Además, el programa resultante no es muy bello y es difícil de leer. En realidad, casi ningún programa de robot se escribe de esa manera. En esta sección, veremos un modo distinto de estructurar los programas de robot que hace más fácil el diseño de comportamientos y además, la estructura que resulta del programa global es limpia y directa. Pueden diseñar comportamientos muy sofisticados usando estas ideas.

La gente de la comunidad robótica llama al estilo de programación mostrado arriba *control reactivo* o *control directo*. También se la llama *fusión de sensor*, y los programas resultantes son conducidos puramente por sensores y por lo tanto son demasiado "*bottom-up*", *es decir, desarrollados* de abajo hacia arriba. Es decir, los valores de los sensores llevan la lógica del control en dirección opuesta a las metas de las tareas del robot en sí mismas. En el *control basado en comportamiento*, uno se aleja de los sensores y focaliza el diseño del programa de robot sobre cierta cantidad y tipos de comportamientos que el robot puede llevar a cabo.

Veamos cómo el control basado en comportamiento nos permite diseñar el comportamiento de salida del corral. Básicamente, el robot tiene que llevar a cabo tres tipos de comportamientos: avanzar (en ausencia de cualquier obstáculo y/o luz), evitar obstáculos (si están presentes), y buscar la luz (si está presente). En el estilo de la escritura de programa basada en comportamiento, definiremos cada uno de estos comportamientos como una unidad de decisión individual. Por lo tanto, cada una se escribe de manera simple y directa. A continuación, el programa de control debe fusionar los comportamientos recomendados por cada unidad de comportamiento. Observen la imagen de abajo:



En el diagrama de arriba, hemos mostrado los tres comportamientos básicos que estamos intentando combinar: `Avanzar`, `Esquivar`, `BuscarLuz`. Cada uno de estos comportamientos da un triple: `Si`/`No`, una Velocidad de traslado,y una Velocidad de rotación. Un `Si` implica que el módulo de comportamiento tiene una recomendación. Un `No` implica que no la tiene. Es decir, permite la posibilidad de un comportamiento que no tenga recomendación. Por ejemplo, en la situación de salida del corral, en ausencia de una fuente de luz percibida por el robot, el módulo de `BuscarLuz` no tendrá una recomendación. Entonces deviene en tarea del árbitro (o el módulo de decisión) decidir cuál de las recomendaciones disponibles usar para manejar el robot. Observen que en definitiva, para controlar el robot, todo lo que uno debe hacer es decidir cuánto trasladar y rotar. Se pueden incorporar muchos esquemas de arbitraje diferentes. Usaremos uno simple pero efectivo: asignarle una prioridad

a cada módulo de comportamiento. Luego el árbitro elige la recomendación con la prioridad más alta. Este estilo de arquitectura de control también se denomina *arquitectura presuntiva.* En la figura de arriba, hemos dibujado los módulos según su orden de prioridad: cuanto más alto está el módulo en el esquema, mayor la prioridad. El comportamiento más bajo, `Avanzar`, no requiere de ningún sensor y siempre está presente: quiere que el robot siempre avance hacia delante.

Resolver el control basado en la combinación de comportamientos simples tiene varias ventajas: pueden diseñar cada comportamiento individual de manera muy sencilla; también pueden testear cada comportamiento individual agregando ese comportamiento y viendo cómo lo lleva a cabo, Pueden agregar cualquier cantidad de comportamientos uno sobre el otro. En el esquema de arriba el régimen de control implica que el robot siempre avanzará hacia delante. Pero si se presenta un obstáculo, pasará por alto el comportamiento *`Avanzar`* e intentará esquivar el obstáculo. Sin embargo, si se detecta una fuente de luz, sobrepasará a los otros comportamientos y se dedicará al comportamiento de buscador de luz. Lo ideal que podemos imaginar es que todos los comportamientos estén funcionando simultáneamente (de manera asincrónica). En esa situación, el árbitro siempre tendrá una o más recomendaciones para adoptar basándose en la prioridad.

Desarrollemos el programa que implementa el control basado en comportamiento. Primero, definimos cada comportamiento:

```
velocidadCrucero = 0.8
velocidadGiro = 0.8
umbralLuz = 80

def avanzar():
  # siempre se mueve hacia adelante
  return [True, velocidadCrucero, 0]

def esquivar():
  # busca si hay algún obstáculo
  L, R = getIR()
  L = 1 - L
  R = 1 - R

  if L:
    return [True, 0, -velocidadGiro]
  elif R:
    return [True, 0, velocidadGiro]
  else:
    return [False, 0, 0]



def buscarLuz():
  L, C, R = getLight()

  if L < umbralLuz:
    return [True, velocidadCrucero/2.0, velocidadGiro]
  elif R < umbralLuz:
    return [True, velocidadCrucero/2.0, -velocidadGiro]
  else:
    return [False, 0, 0]
```

Arriba pueden ver que cada comportamiento individual es simple y fácil de leer (y de escribir). Hay varias maneras de incorporarlos a un programa basado en comportamiento. Aquí hay una:

```
# lista de comportamientos ordenados por prioridad
# (el de la izquierda es el más alta prioridad)
comportamientos = [buscarLuz, esquivar, avanzar]

def main():

  while True:
    T, R = arbitrar()
    mover(T, R)

main()
```

El programa principal llama a la función de arbitraje que devuelve los comandos de traslación y de rotación que se aplican entonces al robot. La función `arbitrar` es lo suficientemente simple:

```
# Decidir qué comportamiento, en orden de prioridad
# recomienda para el robot
def arbitrar():

  for comportamiento in comportamientos:
    output, T, R = comportamiento()
    if output:
      return [T, R]
```

Es decir, requiere a cada comportamiento en orden de prioridades para ver si tiene una recomendación. Si la tiene, ese es el grupo de comandos de motor que se devuelve.

Realizar la siguiente actividad: Implementen en programa de arriba y vean cómo se comporta el robot navegando y saliendo del corral. ¿Qué ocurre si cambian la prioridad (el orden en la lista) de los comportamientos? Al escribir el programa de arriba, hemos usado dos características Python. Las veremos a continuación.

Nombres y valores devueltos

En el capítulo 3 aprendieron que en Python los nombres pueden ser usados para representar funciones así como números y *strings*. También vieron en el Capítulo 5 que las listas, e incluso las fotos o imágenes pueden ser representados usando nombres. Un nombre es un concepto importante en programación. En Python, un nombre puede representar cualquier cosa como su valor: su número, una foto, una función, etc. En el programa de arriba, cuando definimos los comportamientos de las variables como:

```
comportamientos = [buscarLuz, esquivar, avanzar]
```

usamos los nombre `buscarLuz`, `esquivar`, y `avanzar` para denotar las funciones que representan. Nombramos a estas funciones anteriormente en el programa (usando **`def`**). De esta manera, la lista de comportamientos nombrados es una lista de nombres de funciones, cada uno de los cuales denotará la función actual y su valor. A continuación, observen el modo en que usamos la variable `comportamientos` en la función de arbitraje:

```
for comportamiento in comportamientos:
    output, T, R = comportamiento()
    ...
```

Dado que `comportamientos` es una lista, puede ser usada en el loop para representar la secuencia de objetos (en este caso funciones). Por lo tanto, en cada iteración del loop, la variable `comportamiento` toma sucesivos valores de esta lista: `buscarLuz`, `esquivar`, y `avanzar`. Cuando el valor de la variable es `buscarLuz`, la función `buscarLuz` es llamada en esta instrucción:

```
output, T, R = comportamiento()
```

No hay ninguna función llamada `comportamiento()` que esté definida en ninguna parte del programa. Sin embargo, como el valor del nombre de la variable `comportamiento` está establecida para una de las funciones en la lista `comportamientos`, la función correspondiente es invocada.

Otro aspecto nuevo de Python que hemos usado en el programa de arriba es el hecho de que una función puede devolver cualquier objeto como su valor. Por lo tanto, las funciones `avanzar`, `esquivar`, `buscarLuz`, y arbitrar todas devuelven listas como sus valores.

Ambas son características sutiles de Python. Muchos otros lenguajes de programación tienen restricciones sobre los tipos de cosas que podemos llamar como variables y también sobre los tipos de valores que puede devolver una función. Python le da un tratamiento uniforme a todos los objetos.

La Librería Matemática de Python

La mayoría de los lenguajes de programación, incluido Python, proveen una saludable selección de librerías de funciones útiles para que no sea necesario escribirlas. Tales librerías suelen denominarse API ("Application Programming Interfaces", en inglés). Un poco más atrás se les presentó la librería `random` provista por Python. Contiene varias funciones útiles que proveen facilidades para generar números al azar. De modo similar, Python también provee una librería `math` que provee funciones matemáticas de uso frecuente. En el Capítulo 6 usamos la función `exp` de la librería `math` para normalizar los valores del sensor. Algunas de las funciones comúnmente usadas en la librería `math` se listan debajo:

Funciones de uso frecuente en el módulo `math`:

`ceil(x)`: Devuelve el techo de x como un flotante, el valor entero mayor o igual a x.

`floor(x)`: Devuelve el piso de x como un flotante, el valor entero más grande menor o igual que x.

`exp(x)`: Returns e^x.

`log(x[, base])` Devuelve el logaritmo de x a la base dada. Si la base no se especifica, devuelve el logaritmo natural de x (por ej., $\log_e x$).

`log10(x)`: devuelve el logaritmo de base-10 de x (por ej. $\log_{10} x$).

`pow(x, y)`: Devuelve x^y.

`sqrt(x)`: Devuelve la raíz cuadrada de x ($\sqrt{x}$).

Algunas de las otras funciones disponibles en el módulo `math` se listan al final del capítulo. Para usar cualquiera de ellas, todo lo que deben hacer es importar el módulo `math`:

```
import math

>>> math.ceil(5.34)
6.0
>>> math.floor(5.34)
5.0

>>> math.exp(3)
20.085536923187668

>>> math.log10(1000)
3.0
>>> math.log(1024, 2)
10.0
>>> math.pow(2, 10)
1024.0

>>> math.sqrt(7.0)
2.6457513110645907
```

De manera alternativa, pueden importar el módulo `math` como se muestra abajo:

```
>>> from math import *
>>> ceil(5.34)
6.0
>>> floor(5.34)
5.0
>>> exp(3)
20.085536923187668
>>> log10(1000)
3.0
>>> log(1024,2)
10.0
>>> sqrt(7.0)
2.6457513110645907
```

En Python, cuando importa un módulo usando el comando:

```
import <module>
```

Tienen que predeterminar todos sus comandos con el nombre del módulo (como en el caso del primer grupo de ejemplos de arriba). También hemos estado usando la forma

```
from <module> import *
```

Esto importa todas las funciones y otros objetos provistos en el módulo que pueden ser útiles sin la predeterminación. También pueden importar individualmente funciones específicas de un módulo usando:

```
from <modulo> import <esto>, <eso>, ...
```

La versión que utilicen dependerá del contexto en el cual usen las funciones importadas. Pueden encontrarse con una situación en la que dos módulos distintos

definen funciones con el mismo nombre pero hacen cosas diferentes (o para hacer cosas diferentes). Para usar ambas funciones en el mismo módulo deberán usar el módulo prefijado para que queda claro qué versión estarán utilizando.

Realizando cómputos

Volvamos al estilo tradicional de computación por ahora. Verán que los conceptos que han aprendido hasta ahora les permitirán escribir muy distintas e interesantes aplicaciones de computación. También les dará un sentido claro de la estructura de los típicos programas de computación. Más adelante, en el Capítulo 11, volveremos también al tema más amplio de diseño general de los programas de computación.

Una Calculadora de Préstamo

Su auto actual, un adorable 1992 SAAB 93 fue comprado usado y durante los últimos meses no han tenido más que problemas para mantener el auto en la ruta. La noche pasada la llave de arranque se rompió dentro de la ranura de la llave al intentar iniciarlo y ahora la parte rota no sale (esto sucedía mucho en los viejos SAABs). El mecánico tuvo que desmantelar todo el engranaje de arranque para sacar el pedazo roto y podría salir hasta $500. El motor del auto, que ya tiene más de 290.000 kilómetros, los ha dejado varados en la ruta muchas veces. Han decidido que es suficiente, irán en busca de un reluciente auto nuevo y confiable. Han estado trabajando de noche en un restaurante para hacer algo de dinero extra y han ahorrado $5500.00 justamente para una situación como esta. Ahora se están preguntando qué tipo de auto podrán comprar. Obviamente, deberán pedir un préstamo del banco para financiar el resto del costo, pero no están seguros acerca de cuán grande deberá ser el préstamo, y por lo tanto qué tipo de auto pueden costear. Pueden escribir un pequeño programa Python para ayudarles a probar varios escenarios.

Pueden soñar (realistamente) con el auto que quisieran comprar, o pueden ir a cualquier sitio de venta de autos en la Web (www.edmunds.com o www.demotores.com son dos buenos lugares donde comenzar). Digamos que han invertido horas mirando características y opciones y finalmente han definido su deseo en un par de posibilidades. La primera opción costará $22,000.00 y la segunda está valuada en $18,995.00. Ahora tienen que decidir cuál de estos pueden realmente comprar.

Primero, hablan con un par de bancos y miran algunas ofertas de préstamos en la Web. Por ejemplo, vayan a bankrate.com y vean las tasas actuales de préstamos para autos nuevos.

Supongan que las tasas de los préstamos para esto son de: 6.9% para 36 meses, 7.25% para 48 meses, y 7.15% para 60 meses.

Podrán ver que hay una importante variación en las tasas. Dada esta información, ahora están listos para escribir un programa que los ayude a averiguar por cuál de las dos opciones podrán optar. A fin de asegurarse el préstamo, deben garantizar que tienen suficiente dinero como para pagar los impuestos locales de venta (¡un 6% de impuestos de venta sobre un auto de $20,000 llegarán a la considerable cifra de $1200!). Luego de pagar los impuestos de venta podrán usar el resto del dinero que han ahorrado para el pago inicial. El dinero restante es la cantidad que pedirían prestada. Dependiendo del tipo de préstamo que elijan, variarán los pagos

mensuales y la cantidad de tiempo de los mismos. Hay una fórmula simple que pueden usar para estimar el pago mensual:

$$MonthlyPayment = \frac{LoanAmount * MonthlyInterestRate}{1 - e^{-LoanTerm * \log(1 + MonthlyInterestRate)}}$$

¡Whoa! Eso se ve complicado. Sin embargo, dada la fórmula, pueden ver que en realidad requiere de dos funciones matemáticas: $\log(x)$ y e^x, ambas disponibles en el módulo `math` de Python. De pronto, el problema no parece tan difícil.

Intentemos delinear los pasos necesarios para escribir el programa: primero, observen el costo del auto, la cantidad de dinero que han ahorrado, y la tasa de impuesto de venta. También registren los siguientes datos financieros: la tasa de interés y la duración del préstamo. La tasa de interés dada es en general el porcentaje anual. Conviértanlo en la tasa mensual (dividiéndolo por 12). Luego, computen el impuesto de venta que pagarán. Usen en dinero restante para realizar el pago inicial. Luego determinen la cantidad de dinero que tomarán como préstamo. Ingresen todos los valores en la fórmula y computen el pago mensual. Además, computen el costo total del auto. Obtengan todos los resultados. A continuación, podemos tomar cada uno de los pasos de arriba y codificarlos en un programa. Aquí tienen una primer ordenamiento:

```
def main():
  # Primero observen del costo del auto (Cost),
  # La cantidad de dinero que han ahorrado (Cash),
  # y la tasa de impuestos por venta (TaxRate)

  # También observen los datos financieros: la tasa de interés(APR),
  # y luego la duración del préstamo (Term)
  # la tasa de interés en general se fija en forma anual(APR)
  # Conviértanlo en una tasa mensual (dividiéndola por 12) (MR)

  # A continuación, computen la tasa de venta que pagarán (SalesTax)
  # Usen el dinero restante para efectuar el pago inicial (DownPayment)
  # Luego determinen la cantidad que pedirán prestada (LoanAmount)

  # Ingresen todos los valores en la fórmula y computen
  # el pago mensual (MP)

  # Calculen el costo total del auto (TotalCost)

  # Muestren los resultados

main()
```

Arriba, hemos tomado los pasos y los hemos convertido en un esqueleto de programa Python. Todos los pasos se han convertido en comentarios Python, y donde se necesitan, hemos decidido los nombres de las variables que sostendrán los valores necesarios para los cálculos. Esto es útil porque también ayuda a determinar cómo se codificará la fórmula, además de ayudar a determinar qué valores pueden ser programados al interior y cuáles deberán ser suministrados como entrada. Hacer que el programa requiera entradas les permitirá ingresar fácilmente los diferentes parámetros y luego, basándose en los salidas que reciben, pueden decidir qué auto comprar. Codifiquemos todos las entradas primero:

```
def main():

    # Primero observen del costo del auto (Cost),
    Cost = input("Ingrese el costo del auto: $")

    # La cantidad de dinero que han ahorrado (Cash),
    Cash = input("Ingrese la cantidad de dinero que ahorró: $")

    # y la tasa de impuestos por venta (TaxRate)
    SalesTaxRate = 6.0

   # También observen los datos financieros: la tasa de interés(APR),
   # y luego la duración del préstamo (Term)
   # la tasa de interés en general se fija en forma anual(APR)
    APR = input("Ingrese la tasa de Interes anual(in %): ")

    # Conviértanlo en una tasa mensual (dividiéndola por 12) (MR)

   # A continuación, computen la tasa de venta que pagarán (SalesTax)
   # Usen el dinero restante para efectuar el pago inicial (DownPayment)
   # Luego determinen la cantidad que pedirán prestada (LoanAmount)

   # Ingresen todos los valores en la fórmula y computen
   # el pago mensual (MP)

   # Calculen el costo total del auto (TotalCost)

   # Muestren los resultados

main()
```

Hemos refinado el programa incluyendo todos las entradas que se necesitarán cada vez que se ejecute el programa. Observen que elegimos no incorporar la tasa de impuestos de venta, sino que se la asignamos a la variable `SalesTaxRate`. Si ustedes hubiesen querido, podrían haberlo ingresado como otra entrada. Lo que ustedes elijan tener como entrada de su programa es su decisión de diseño. A veces el problema puede ser encuadrado de manera que explícitamente especifica las entradas, a veces deben averiguarlo. En general, donde deben hacer al programa más versátil es allí donde deben basarse las decisiones. Por ejemplo, determinar la tasa de impuestos de venta en 6.0 hará que el programa sea usable solamente en aquellas instancias en las que aplique esa tasa. Si, por ejemplo, si quisieran que un amigo en otra parte del país utilizara el programa, deberían elegir que ese también sea un valor de entrada. Avancemos hacia el siguiente paso del programa y codifiquémoslos en Python. Se trata básicamente de los cómputos. Lo primeros son sencillos. Quizás el cómputo más complicado de codificar sea la fórmula para calcular el pago mensual. Todos se muestran en la versión de abajo.

```
from math import *

def main():
    # Primero observen del costo del auto (Cost),
    Cost = input("Ingrese el costo del auto: $")

    # La cantidad de dinero que han ahorrado (Cash),
    Cash = input("Ingrese la cantidad de dinero que ahorró: $")
```

```
  # y la tasa de impuestos por venta (TaxRate)
  SalesTaxRate = 6.0

 # También observen los datos financieros: la tasa de interés(APR),
 # y luego la duración del préstamo (Term)
 # la tasa de interés en general se fija en forma anual(APR)
  APR = input("Ingrese la tasa de Interes anual(in %): ")

  # Ingrese la duración del préstamo (Term)
  term = input("Ingrese la duración del préstamo (in months): ")

  # Convertir el APR a una tasa mensual (dividir por 12) (MR)
  # también dividirlo por 100 ya que el valor de entrada es en %
  MR = APR/12.0/100.0

  # A continuación, computen la tasa de venta que pagarán (SalesTax)
  SalesTax = Cost * SalesTaxRate / 100.0

  # Usen el dinero restante para efectuar el pago inicial (DownPayment)
  DownPayment = Cash - SalesTax

  # Luego determinen la cantidad que pedirán prestada (LoanAmount)
  LoanAmount = Cost - DownPayment

  # Ingresen todos los valores en la fórmula y computen
  # el pago mensual (MP)
  MR = (LoanAmount * MR) / (1.0 - exp(-term * log(1.0+MR)))

  # Calculen el costo total del auto (TotalCost)
   TotalCost = SalesTax + DownPayment + MR * term

  # Muestren los resultados
  print "Aquí los detalles sobre tu nuevo auto ..."
  print "----------------------------------------"
  print
  print "Dinero ahorrado $", Cash
  print "Costo del auto $", Cost
  print "Tasa de impuestos por venta", SalesTaxRate, "%"
  print "Impuestos sobre la venta del autor $", SalesTax
  print "Pago Inicial $", DownPayment
  print "Dinero a pedir restado $", LoanAmount
  print "en", term, "meses a una tasa de", APR, "% APR"
  print "Tu pago mensual será de $", MP
  print "El costo total será $", TotalCost
  print

main()
```

Realizar la siguiente actividad: Al ingresar el programa de arriba y hacerlo correr en Python, pueden ingresar los datos de su auto. Aquí hay un ejemplo del programa corriendo:

```
Ingrese el costo del auto:$20000.00
Ingrese la cantidad de dinero que ahorró: $ 5500.00
Ingrese la tasa de Interes anual(in %): 6.9
Ingrese la duración del préstamo (in months):36
```

```
Aquí los detalles sobre tu nuevo auto ...
----------------------------------------
Dinero ahorrado $$ 5500.0
Costo del auto $ 20000.0
Tasa de impuestos por venta  $ 1200.0
Impuestos sobre la venta del autor $ 4300.0
Pago Inicial $ 15700.0
Dinero a pedir restado  en  36 meses a una tasa de 6.9 % APR
Tu pago mensual será de $ 484.052914723
El costo total será $ 22925.90493
```

Parece que para el auto de $20000.00, por un préstamo de 36 meses a una tasa del 6.9%, deberán pagar $484.05 (o $484.06, ¡dependiendo de cómo su compañía de préstamos para autos redondea los centavos!).

Cuando deben restringir los valores de la salida a espacios decimales específicos (dos espacios en el caso de los pesos y los centavos, por ejemplo), pueden usar el string de formato de características construido en Python. Por ejemplo, en un string, pueden especificar cómo incluir un valor de punto flotante como se muestra a continuación:

```
"Valor de PI %5.3f con 3 lugares decimales" % (math.pi)
```

La de arriba es una expresión Python que tiene la siguiente sintaxis:

```
<string> % <expresion>
```

Dentro del `<string>` hay una especificación que comienza con un signo `%` y termina con `f`. Lo que sigue al signo `%` es una especificación numérica del valor para ser insertado en ese punto en el string. La `f` en la especificación hace referencia a hecho de que es para un valor de punto flotante. Entre el signo `%` y la `f` se encuentra el número 5.3. Esto especifica que el número de punto flotante a insertarse ocupará por lo menos 5 espacios, 3 de los cuales irán después del decimal. Uno de los espacios siempre está ocupado por el decimal. Por lo tanto, la especificación `%5.3f` especifica un valor a ser insertado de la siguiente manera:

```
"Este es el valor de  PI -.--- expresado con tres lugares decimales"
```

Abajo pueden ver el resultado de ejecutar esto en Python:

```
>>> "Valor de PI %5.3f con 3 lugares decimales" % (math.pi)
Valor de  PI 3.142 con 3 lugares decimales'

>>> "Valor de PI %7.4f con 4 lugares decimales" % (math.pi)
'Valor de PI  3.1416 con 4 lugares decimales'
```

En el segundo ejemplo de arriba hemos reemplazado la especificación con `%7.4f`. Noten que el string resultante define 7 espacios para imprimir ese valor. Si hay más espacio que los necesarios, se rellenan con blancos en el borde inicial (noten el espacio extra antes del 3 en el ejemplo de arriba). Si el espacio especificado es menor, siempre se expandirá para acomodar el número. Por ejemplo:

```
>>> " Valor de PI %1.4f con 4 lugares decimales" % (math.pi)
Valor de PI 3.1416 con 4 lugares decimales'
```

Deliberadamente especificamos que el valor sea de 1 ancho de espacio con 4 espacios después del decimal (por ej. `%1.4f`). Como pueden ver, el espacio se expandió para acomodar el valor. Lo que se asegura es que el valor siempre se imprime usando el exacto número de espacios después del decimal. Aquí hay otro ejemplo:

```
>>> "5 es también  %1.3f con  3 lugares decimales." % 5
'5 es también 5.000 con  3 lugares decimales .'
```

De esta manera, el valor impreso como 5.000 (por ejemplo, los tres espacios después del decimal siempre se consideran relevantes en la especificación, como `%1.3f`). De modo similar para especificar un número entero o valores *enteros*, pueden usar la letra `d`, y para strings pueden usar la letra `s`:

```
>>> "Hola %10s, Cómo estás?" % "Juan"
'Hola       Juan, Cómo estás?'
```

De manera predeterminada, las especificaciones más extensas se justifican hacia la derecha. Pueden usar una especificación `%` para justificar hacia la izquierda. Por ejemplo:

```
>>> "Hola %-10s, Cómo estás?" % "Juan"
'Hola Juan      , Cómo estás?'
```

Tener este tipo de control sobre los valores impresos es importante cuando están intentando sacar tablas de valores alineados. Modifiquemos nuestro programa de arriba para usar estas características de formato:

```
from math import *

def main():
    # Primero observen del costo del auto (Cost),
    Cost = input("Ingrese el costo del auto: $")

    # La cantidad de dinero que han ahorrado (Cash),
    Cash = input("Ingrese la cantidad de dinero que ahorró: $")

    # y la tasa de impuestos por venta (TaxRate)
    SalesTaxRate = 6.0

   # También observen los datos financieros: la tasa de interés(APR),
   # y luego la duración del préstamo (Term)
   # la tasa de interés en general se fija en forma anual(APR)
    APR = input("Ingrese la tasa de Interes anual(in %): ")

    # Ingrese la duración del préstamo (Term)
    term = input("Ingrese la duración del préstamo (in months): ")

    # Convertir el APR a una tasa mensual (dividir por 12) (MR)
    # también dividirlo por 100 ya que el valor de entrada es en %
    MR = APR/12.0/100.0

    # A continuación, computen la tasa de venta que pagarán (SalesTax)
    SalesTax = Cost * SalesTaxRate / 100.0

    # Usen el dinero restante para efectuar el pago inicial (DownPayment)
    DownPayment = Cash - SalesTax

    # Luego determinen la cantidad que pedirán prestada (LoanAmount)
```

```
    LoanAmount = Cost - DownPayment

    # Ingresen todos los valores en la fórmula y computen
    # el pago mensual (MP)
    MR = (LoanAmount * MR) / (1.0 - exp(-term * log(1.0+MR)))

    # Calculen el costo total del auto (TotalCost)
     TotalCost = SalesTax + DownPayment + MR * term

   # Muestren los resultados
    print "Aquí los detalles sobre tu nuevo auto ..."
    print "----------------------------------------"
    print
    print "Dinero ahorrado $%1.2f" %  Cash
    print "Costo del auto $%1.2f" % Cost
    print "Tasa de impuestos por venta%1.2f" % SalesTaxRate
    print "Impuestos sobre la venta del autor $", SalesTax, "%"
    print "Pago Inicial $%1.2f" % DownPayment
    print "Dinero a pedir restado $%1.2f" % LoanAmount
    print "en %2d meses a una tasa de  %1.2f APR" % (term, APR)
    print "Tu pago mensual será de $%1.2f" % MP
    print "El costo total será $%1.2f" % TotalCost
    print

main()
```

Al ejecutarlo de nuevo (digamos para términos de préstamos levemente diferentes), obtienen:

```
Ingrese el costo del auto:$20000.00
Ingrese la cantidad de dinero que ahorró: $ 5500.00
Ingrese la tasa de Interes anual(in %): 7.25
Ingrese la duración del préstamo (in months): 48

Aquí los detalles sobre tu nuevo auto ...
----------------------------------------
Dinero ahorrado $ 5500.00
Costo del auto $ 20000.00
Tasa de impuestos por venta  $ 1200.0 %
Impuestos sobre la venta del autor $ 4300.00
Pago Inicial $ 15700.00
Dinero a pedir restado  en  48 meses a una tasa de 7.25 APR
Tu pago mensual será de $377.78
El costo total será $23633.43
```

Pueden ver que para el mismo monto, si toman el préstamo por un período más largo pueden reducir sus pagos mensuales en más de $100 (aunque pagan aproximadamente $700 más al final).

Toma de Decisiones en programas de Computación

La toma de decisiones es central para todos los programas de computación. De hecho, hay un famoso teorema en ciencias de la computación que dice que si cualquier dispositivo programable es capaz de ser programado para realizar *ejecuciones secuenciales*, *toma de decisiones* y *repetición*, será capaz de expresar cualquier algoritmo computable. Esta es una idea muy poderosa presentada en términos de ideas muy simples. Dado un problema, si pueden describir su solución

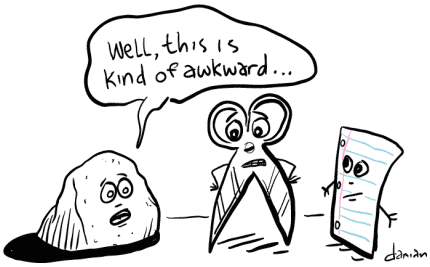
en términos de la combinación de tres modelos de ejecución, entonces pueden lograr que cualquier computadora resuelva ese problema. En esta sección, veremos la toma de decisiones en una clásica situación de juegos.

Las computadoras han sido extensamente usadas para juegos: para jugar contra otras personas y contra otras computadoras. El impacto de las computadoras en los juegos es tan tremendo que actualmente varios productores construyen consolas de juegos con computadoras adentro que se dedican exclusivamente a juegos.

Diseñemos un juego simple que seguramente han jugado cuando eran chicos: ¡Piedra papel o Tijera!

En este juego, dos jugadores juegan como contrincantes. A la cuenta de tres cada jugador realiza un gesto con la mano que indica que han seleccionado uno de los tres ítems: papel (la mano se extiende plana), tijera (se extienden dos dedos para indicar tijeras), o piedra (indicada con un puño). Las reglas para decidir quién es el ganador son simples: si ambos jugadores eligen el mismo objeto es un empate; de lo contrario, el papel le gana a la piedra, las tijeras le ganan al papel y la piedra le gana a las tijeras. Escribamos un programa de computación para jugar este juego contra la computadora. Aquí hay un delineado para jugar el juego una vez.

```
# La computadora y el jugador realizan una selección
# Decidir el resultado de las selecciones (empate
o quién gana)
# Informar al jugador de la decisión
```



*El nombre exacto del juego puede variar, con los tres componentes apareciendo en diferente orden, o con "roca" en lugar de "piedra". Los que hablan idiomas que no son el español pueden conocer al juego por los términos locales para "piedra, papel, tijera", aunque también se conoce como **Jankenpon** en Japón, **Rochambeau** en Francia, y el Sudáfrica como **Ching-Chong-Cha**...*
Fuente:talkingtotan.wordpress.com/2007/08/

Si presentamos los tres ítems en una lista, podemos hacer que la computadora elija uno de ellos al azar usando las facilidades de generación de números al azar provistas por Python. Si estos ítems se presentan como:

```
items = ["Papel", "Tijera", "Piedra"]
```

Entonces podemos seleccionar cualquiera de los ítems de arriba como elección de la computadora usando la expresión:

```
# La computadora elige
miOpcion = items[<0 o 1 o 2 seleccionado
aleatoriamente>]
```

Es decir `items[0]` representa la elección `"Papel"`, `items[1]` representa `"Tijera"`, e `items[2]` es `"Piedra"`. Hemos visto en el Capítulo 4 que podemos generar un número al azar dentro de cualquier rango usando la función `randint` del módulo de librería `random` en Python. Por lo tanto podemos modelar la computadora haciendo para hacer una selección al azar usando la siguiente sentencia:

```
from random import *

#  La computadora elige
miOpcion = items[randint(0,2)]
```

Recuerden que `randint(n, m)` devuelve un número al azar dentro del rango `[n..m]`. Entonces, `randrange(0, 2)` devolverá o 0, 1, o 2. Podemos usar el comando `askQuestion` en Myro para pedirle al jugador que indique su selección (ver Capítulo 3).

```
# El Jugador elige
tuOpcion = askQuestion("Elige un item.", items)
```

Ahora que sabemos cómo la computadora y el jugador toman su decisión, necesitamos pensar sobre la decisión del resultado. Aquí hay un esquema:

```
Si ambos eligieron el mismo ítem entonces es un empate
Si la computadora gana entonces se le informa al jugador
Si el jugador gana entonces se le informa al jugador
```

Si reescribimos lo de arriba usando sentencias `if` obtenemos el siguiente borrador:

```
if miOpcion == tuOpcion:
   print "Empatamos!"
if < miOpcion gana a  tuOpcion>:
   print "Gané!"
else:
   print "Ganaste!"
```

Todo lo que debemos hacer es averiguar cómo escribir la condición `<miOpcion gana a  tuOpcion>`. La condición debe capturar las reglas del juego mencionado arriba. Podemos codificar todas las reglas en una expresión condicional de la siguiente manera:

```
  if ( miOpcion == "Papel" and tuOpcion == "Piedra")
      or ( miOpcion == "Tijera" and tuOpcion == "Papel")
      or ( miOpcion == "Piedra" and tuOpcion == "Tijera"):
     print "Gané!"
  else:
     print "Ganaste!"
```

La expresión condicional de arriba captura todas las posibilidades que deben ser examinadas a fin de tomar la decisión. Otra forma de escribir la decisión de arriba sería usando lo siguiente:

```
  if miOpcion == "Papel" and tuOpcion == "Piedra":
     print "Gané!"
  elif miOpcion == "Tijera" and tuOpcion == "Papel":
     print "Gané!"
  elif miOpcion == "Piedra" and tuOpcion == "Tijera":
    print "Gané!"
  else:
     print "Ganaste!"
```

Es decir, cada condición se examina por vez hasta que se encuentra una que confirma que la computadora gana. Si ninguna de esas condiciones es verdadera, la parte `else` de la sentencia `if` se alcanzará para indicar que el jugador ha ganado.

Otra alternativa para escribir la decisión de arriba es codificar la decisión en una función. Asumamos que ha una función `gana` que devuelve `True` o `False`, dependiendo de las elecciones. Podríamos escribir lo de arriba de la siguiente manera:

```
if miOpcion == tuOpcion:
   print "Empatamos!"
```

```
if gana(miOpcion, tuOpcion):
   print "Gané!"
else:
   print "Ganaste!"
```

Miremos más de cerca cómo podemos definir la función `gana`. Necesita devolver `True` si `miOpcion` le gana a tuOpcion. Así que todo lo que debemos hacer es codificar las reglas del juego descrito arriba. Aquí hay una versión borrador de la función:

```
def gana(yo, vos):
  # Te gano? Si es así, retornamos True, sino, retornamos False

  if yo == "Papel" and vos == "Piedra":
    # Papel r le gana a la Piedra
    return True
  elif yo == "Tijera" and vos == "Papel":
    # Tijera le gana a Papel
    return True
  elif yo == "Piedra" and vos == "Tijera":
    # Piedra le gana a Tijera
    return True
  else:
    return False
```

Una vez más, hemos usado la sentencia `if` en Python para codificar las reglas del juego. Ahora que hemos resuelto todas las partes críticas del programa, podemos unirlas como se muestra abajo:

```
#Un programa que  juega al juego Piedra, Papel y Tijeras!
from myro import *
from random import randrange

def gana(yo, vos):
  # Te gano? Si es así, retornamos True, sino, retornamos False

  if yo == "Papel" and vos == "Piedra":
    # Papel r le gana a la Piedra
    return True
  elif yo == "Tijera" and vos == "Papel":
    # Tijera le gana a Papel
    return True
  elif yo == "Piedra" and vos == "Tijera":
    # Piedra le gana a Tijera
    return True
  else:
    return False

def main():
   #  Juega una ronda del juego Piedra, Papel y Tijeras!
   print "Dime: Piedra, Papel o Tijeras!"
   print "Haz tu elección en la ventana que aparecerá"

   items = ["Papel", "Tijera", "Piedra"]

   # la computadora y el Jugador hacen sus selecciones ...
   # la computadora elige
   miOpcion = items[randint(0,2)]
```

```
  # El Jugador elige
  tuOpcion = askQuestion("Elige un item.", items)

   # informamos las opciones de los jugadores
   print
   print "Yo elegí: ", miOpcion
   print "Vos elegiste: ", tuOpcion

   # Decidimos si hay un empate o alguno gana
   if miOpcion == tuOpcion:
      print "Elegimos la misma opción!"
      print "Es un empate."
   elif gana(miOpcion, tuOpcion):
      print "Ya que", miOpcion, "le gana a", tuOpcion, "..."
      print "Gane!"
   else:
      print"Ya que", tuOpcion, "le gana a", miOpcion, "..."
      print "Ganaste!"

   print "Gracias por Jugar. Adiós!"

main()
```

Se agregaron unos comando print más para que la interacción sea más natural.

Realizar la siguiente actividad: Implementen el programa de Piedra, Papel o Tijera de arriba y ejecútenlo varias veces para asegurarse de que lo comprenden completamente. Modifiquen el programa de arriba para jugar varias vueltas. Además, incorporen un sistema de puntaje que mantiene un registro de la cantidad de veces que ganó cada jugador y de la cantidad de empates.

Resumen

En este capítulo han visto una variedad de paradigmas de control: control de robot basado en el comportamiento, escribir un programa computacional simple pero útil, y escribir un juego de computadora simple. El control basado en el comportamiento es una forma poderosa y efectiva de describir sofisticados programas para controlar los robots. Este es el paradigma usado en muchas aplicaciones comerciales de robots. Deberían probar algunos de los ejercicios propuestos debajo para sentirse cómodos con esta técnica de control. Los otros dos programas ilustran cómo, usando los conceptos que han aprendido, pueden diseñar otras aplicaciones de computadora útiles. Los otros dos programas ilustran cómo, usando los conceptos que han aprendido, pueden diseñar otras aplicaciones útiles de computación. En varios Capítulos siguientes, exploraremos varias dimensiones de la computación: computación de medios, escribir juegos, etc.

Revisión Myro

No hubo nada nuevo acerca de Myro en este Capítulo.

Revisión Python

El módulo de librería de matemática provee varias funciones matemáticas útiles. Algunas de las más usadas se listan abajo:

`ceil(x)`: Devuelve el techo de x como un flotante, el valor entero más bajo mayor o igual que x. **`floor(x)`** Returns the floor of x as a float, the largest integer value less than or equal to x.

`exp(x)`: Devuelve e^x .

`log(x[, base])`: Devuelve el logaritmo de x a la base dada. Si la base no está especificada, devuelve el logaritmo natural de x (por ej., $\log_e x$).

`log10(x)`: Devuelve el logaritmo de base -10 de x (por ej. $\log_{10} x$) .

`pow(x, y)`: Devuelve x^y .

`sqrt(x)`: Devuelve la raíz cuadrada de x ($\sqrt{x}$).

Funciones trigonométricas

`acos(x)`: Devuelve el arco coseno de x, en radianes.

`asin(x)`: Devuelve el arco sine de x, en radianes.

`atan(x)`: Devuelve el arco tangente de x, en radianes.

`cos(x)`: Devuelve el coseno de x radianes.

`sin(x)`: Devuelve el sine de x radianes.

`tan(x)`: Devuelve la tangente de x radianes.

`degrees(x)`: Convierte el ángulo x de radianes a grados.

`radians(x)`: Convierte el ángulo x de radios a radianes.

El módulo también define dos constantes matemáticas:

`pi`: `La constante matemática` π.

`e`: La constante matemática *e*.

Ejercicios

1. Reemplazar el módulo esquivar en el programa de control basado en comportamiento por un módulo llamado: seguir. El módulo seguir intenta detectar una pared a la derecha del robot (o izquierda) y luego intenta seguirla todo el tiempo. Observen el comportamiento resultante.

2. Realizar el crecimiento de población modelándolo como una ecuación diferencial.

3. Realizar el crecimiento de población modelándolo como una función continua de crecimiento.

4. En lugar de usar el enunciado:

tuOpcion = items[randint(0,2)]

usar el comando:

tuOpcion = choice(items)

`choice(<list>)`: es otra función definida en el módulo random. Al azar busca un elemento de la lista provista.

5. ¿Pueden mejorar el programa?

6. Implementen el juego HiLo: la computadora elige un número al azar entre 0 y 1000 y el usuario debe tratar de adivinarlo. Con cada intento, la computadora le informa al usuario si su número estuvo cerca (en inglés se utiliza el término Hi ("alto"), o lejos (en inglés se utiliza el término Low ("bajo"), o si acertaron.

7. Inviertan los roles en el juego HiLo de arriba. Esta vez, el usuario adivina un número entre 0 y 1000 y el programa de computadora debe adivinarlo. Piensen una buena estrategia para adivinar el número basándose en claves de alto o bajo, y luego implementen esa estrategia.