Clase 2: El Lenguaje Python

Grupo de Desarrollo Lihuen

Agosto 2012

Clase 2

Temario

- Consideraciones generales
- Operadores y expresiones
- Variables y Tipos
- Funciones
- Las Estructuras de Control
- Guía práctica

Primer curso de programación usando robots y Python

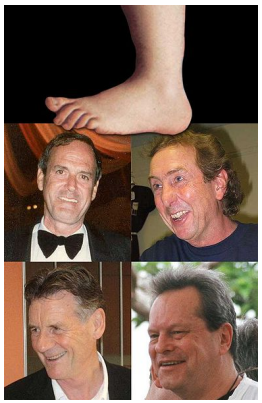
Objetivos

El objetivo de esta clase es dar las características generales sobre la sintaxis de Python

Python

Características Básicas

- Es interactivo
- Multi plataforma
- Permite una programación prolija e intuitiva
- Es posible programar aplicaciones complejas
- Muy usado
- Es **software libre**



Monty Python

Características Generales del Lenguaje Python

Operadores básicos

Operadores Aritméticos

+	Suma de números
-	Resta
*	Multiplicación
/	División
%	Resto de división entera

Ejemplos

Ejemplo	Resultado
10 / (2-3)	2
7 / 3	2
5 / 2.0	2.5

Características Generales del Lenguaje Python

Operadores básicos

Operadores Aritméticos

+	Suma de números
-	Resta
*	Multiplicación
/	División
%	Resto de división entera

Ejemplos

Ejemplo	**Resultado**
10 / (2-3)	2
7 / 3	2
5 / 2.0	2.5

Características Generales del Lenguaje Python

Operadores básicos

Operadores de Caracteres

+	Concatenación
*	Repetición

Ejemplos

“Hola“ + “ Argentina!”	”Hola Argentina!
“Hola“ * 3	“HolaHolaHola”

Características Generales del Lenguaje Python

Operadores básicos

Operadores de Caracteres

+	Concatenación
*	Repetición

Ejemplos

“Hola“ + “ Argentina!”	”Hola Argentina!
“Hola“ * 3	“HolaHolaHola”

Características Generales del Lenguaje Python

Variables

- Las variables son nombres que apuntan o representan datos
- Se asocian a los datos a través de la sentencia de **asignación** (=)
- Sus nombres pueden contener letras, números o el símbolo de subrayado(_), pero **DEBE** comienzan con una letra
- **Hay sensibilidad a mayúsculas y minúsculas**

Identificadores

Correctos	**Incorrectos**
casa12	12casa
casa_12	casa!
Casa_12	mi casa

Características Generales del Lenguaje Python

Variables

- Las variables son nombres que apuntan o representan datos
- Se asocian a los datos a través de la sentencia de **asignación** (=)
- Sus nombres pueden contener letras, números o el símbolo de subrayado(_), pero **DEBE** comienzan con una letra
- **Hay sensibilidad a mayúsculas y minúsculas**

Identificadores

Correctos	**Incorrectos**
casa12	12casa
casa_12	casa!
Casa_12	mi casa

Características Generales del Lenguaje Python

Ejemplo

Queremos calcular el área de un círculo de radio 10:

```
radio = 10
pi = 3.14159
area = pi * radio * radio
print area
```

Esto imprimirá 314.15 en la pantalla

Características Generales del Lenguaje Python

Funciones

- Una función agrupa un conjunto de sentencias.
- Puede tener argumentos
- Sintaxis:

```
def nombre(lista de parámetros):
    sentencias
```

Características Generales del Lenguaje Python

Funciones

- Para ejecutar un función se la debe invocar:

```
nombre(parametros)
```

- Y el valor retornado, puede asignarse a una variable:

```
resultado = nombre(parametros)
```

- O imprimirse en pantalla, por ejemplo.

```
print nombre(parametros)
```

Características Generales del Lenguaje Python

Funciones

- Para ejecutar un función se la debe invocar:

```
nombre(parametros)
```

- Y el valor retornado, puede asignarse a una variable:

```
resultado = nombre(parametros)
```

- O imprimirse en pantalla, por ejemplo.

```
print nombre(parametros)
```

Características Generales del Lenguaje Python

Funciones

- Para ejecutar un función se la debe invocar:

```
nombre(parametros)
```

- Y el valor retornado, puede asignarse a una variable:

```
resultado = nombre(parametros)
```

- O imprimirse en pantalla, por ejemplo.

```
print nombre(parametros)
```

Una función tiene un encabezado y un cuerpo:

```
def nombre(parametros):
    sentencia1
    sentencia2
    sentencia3
```

Encabezado de la función

Cuerpo de la función

CUIDADO CON LA INDENTACION!!!!!

Una función tiene un encabezado y un cuerpo:

```
def nombre(parametros):
```

Encabezado de la función

```
    sentencia1
    sentencia2
    sentencia3
```

Cuerpo de la función

CUIDADO CON LA INDENTACION!!!!!

Una función tiene un encabezado y un cuerpo:

```
def nombre(parametros):
```

Encabezado de la función

```
    sentencia1
    sentencia2
    sentencia3
```

Cuerpo de la función

CUIDADO CON LA INDENTACION!!!!!

¿Dibujamos un cuadrado?

```
def cuadrado():
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
```

- Siempre se mueve la misma distancia
- No retorna ningún valor

¿Dibujamos un cuadrado?

```
def cuadrado():
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
```

- Siempre se mueve la misma distancia
- No retorna ningún valor

¿Dibujamos un cuadrado?

```
def cuadrado():
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, 0.5)
        wait(1)
        mi_robot.turnRight(35, 1)
```

- Siempre se mueve la misma distancia
- No retorna ningún valor

¿Y si le pasamos parámetros?

```
def cuadrado(tiempo):
        mi_robot.forward(50, tiempo)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, tiempo)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, tiempo)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, tiempo)
        wait(1)
        mi_robot.turnRight(35, 1)
```

Podemos invocar esta función con distintos argumentos

```
cuadrado(0.5)
cuadrado(1)
```

¿Y si le pasamos parámetros?

```
def cuadrado(tiempo):
        mi_robot.forward(50, tiempo)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, tiempo)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, tiempo)
        wait(1)
        mi_robot.turnRight(35, 1)
        mi_robot.forward(50, tiempo)
        wait(1)
        mi_robot.turnRight(35, 1)
```

Podemos invocar esta función con distintos argumentos

```
cuadrado(0.5)
cuadrado(1)
```

Características Generales del Lenguaje Python

Programa - Módulo

- Guardar código en un archivo.
- Ejecutar muchas veces.
- Evitar repetir escritura de código en el intérprete.
- Reutilización.
- Uitlización de un IDE: Geany - Pyshell.

Características Generales del Lenguaje Python

Programa - Módulo

- Guardar nuestro código en el IDE.
- Extensión **.py**.

Programa script.

```
#!/usr/bin/python
print "Hola mundo"
```

Programa sin path.

```
print "Hola mundo"
```

Lo ejecutamos en la terminal

```
./hola.py
```

Lo ejecutamos en la terminal
python hola.py

Características Generales del Lenguaje Python

Programa - Módulo

Formas de importar:

```
import modulo
from modulo import *
```

Escribir nuestro código en un archivo separado.

movimientos.py

```
def dudo(rob):
  rob.forward(50,2)
  rob.backward(50,2)
```

Importar

```
import movimientos
movimientos.duda(r)
```

Realizar cambios en el módulo, modifico movimientos.py

```
def dudo(rob):
  rob.forward(50,2)
  rob.backward(50,2)
def giro(res)
   res.turnLeft(40, 4)
   res.turnRight(40, 4)
```

Importar

```
movimientos = reload(
    movimientos)
movimientos.giro(robot)
movimientos.dudo(robot)
```

Tomando decisiones

- La sentencia **if** permite ejecutar un bloque de código en forma condicional
- Sintaxis:

```
if ( expresion booleana):
      sentencia
      sentencia
```

- El bloque indentado se ejecuta sólo si la expresión lógica es verdadera

Características Generales del Lenguaje Python

Sentencia if

```
def avanzar(robot, velocidad, tiempo):
  if velocidad < 20:
    robot.forward(29, tiempo)
```

¿Cómo usamos esta función?

```
from duinobot import *
b=Board("/dev/ttyUSB0")
mi_robot=Robot(b,0)
avanzar(mi_robot, 50, 2)
```

Características Generales del Lenguaje Python

Sentencia if

```
def avanzar(robot, velocidad, tiempo):
  if velocidad < 20:
    robot.forward(29, tiempo)
```

¿Cómo usamos esta función?

```
from duinobot import *
b=Board("/dev/ttyUSB0")
mi_robot=Robot(b,0)
avanzar(mi_robot, 50, 2)
```

Características Generales del Lenguaje Python

Expresiones Booleanas

- Existe **tipo Boolean**, con valores **True** y **False**

Operadores Relacionales

$<$	Menor
$>$	Mayor
$==$	Igual
$!=$	Distinto

Operadores Lógicos

and: Retorna `True` si ambas expresiones son verdaderas	n == 4 and n > 10
or: Retorna `False` si ambas expresiones son falsas	n == 4 or n > 10
not: Invierte el valor de verdad de una expresión	not (n <> 0)

Características Generales del Lenguaje Python

Sentencia if

Muchas veces necesitamos indicar distintas acciones, de acuerdo a una condición.

```
def avanzar(robot, velocidad, tiempo):
  if velocidad < 20:
    robot.forward(20, tiempo)
  else:
    robot.forward(velocidad, tiempo)
```

Características Generales del Lenguaje Python

Sentencia if

Muchas veces necesitamos indicar distintas acciones, de acuerdo a una condición.

```
def avanzar(robot, velocidad, tiempo):
  if velocidad < 20:
    robot.forward(20, tiempo)
  else:
    robot.forward(velocidad, tiempo)
```

Cuando tenemos varias opciones

```
print "Indicanos hacia qué dirección
     te gustaría mover el robot:"
print "1.- Girar a la derecha"
print "2.- Girar a la izquierda"
print "3.- Avanzar"
print "4.- Retroceder"
opcion=raw_input("Opción:")
if opcion=='1':
  robot.turnRight(100,1)
elif opcion=='2':
  robot.turnLeft(100,1)
elif opcion=='3':
  robot.forward(100,1)
elif opcion=='4':
  robot.backward(100,1)
else:
  robot.stop()
```

La sentencia **raw_input** me permite leer caracteres desde el teclado

Características Generales del Lenguaje Python

Sentencia if

Cuando tenemos varias opciones

```
print "Indicanos hacia qué dirección
     te gustaría mover el robot:"
print "1.- Girar a la derecha"
print "2.- Girar a la izquierda"
print "3.- Avanzar"
print "4.- Retroceder"
opcion=raw_input("Opción:")
if opcion=='1':
  robot.turnRight(100,1)
elif opcion=='2':
  robot.turnLeft(100,1)
elif opcion=='3':
  robot.forward(100,1)
elif opcion=='4':
  robot.backward(100,1)
else:
  robot.stop()
```

La sentencia **raw_input** me permite leer caracteres desde el teclado

Repitiendo Instrucciones

- La sentencia **while** permite ejecutar un bloque de código mientras se cumpla una determinada condición
- Sintaxis:

```
while ( expresion booleana):
      sentencia
      sentencia
```

- El bloque indentado se ejecuta tantas veces mientras la expresión lógica es verdadera

Características Generales del Lenguaje Python

Sentencia while

Ejemplo:

```
def decido_movimiento(robot):
  print('''Acción para mover el robot:
        1.- Girar a la derecha
        2.- Girar a la izquierda
        3.- Avanzar
        4.- Salir''')
  opcion=raw_input("Opción:")
  while (opcion!= '4'):
    if opcion=='1':
      robot.turnRight(100,1)
    elif opcion=='2':
      robot.turnLeft(100,1)
    elif opcion=='3':
      robot.forward(100,1)
    else:
      print "Ingresaste una opción no válida."
    opcion = raw_input("Opción: ")
```

Repitiendo Instrucciones

- La sentencia **for** permite ejecutar un bloque de código un número fijo de veces
- Sintaxis:

```
for var in lista_de_valores:
        sentencia
        sentencia
```

- El bloque indentado se ejecuta tantas veces como elementos tenga la lista de valores
- La variable `var` toma todos los valores de la `lista_de_valores`

Características Generales del Lenguaje Python

Sentencia for

Ejemplo

```
for i in [1,2,3]:
 robot.forward(50, 0.5)
 robot.bakckward(50, 0.5)
 robot.turnRight(35, 1)
 robot.turnLeft(35, 1)
 wait(1)
```

- La variable **i** toma los valores, 1, 2 y 3.
- ¿Y si quiero repetir 1000 veces esto?

- La función **range** permite generar listas en forma automática

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(3, 8)
[3, 4, 5, 6, 7]
>>> range(0, 10, 2)
[0, 2, 4, 6, 8]
>>> range(7, 3, -1)
[7, 6  5, 4]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

Prestemos atención al valor final de la lista

Siempre es uno menos que el valor indicado

Características Generales del Lenguaje Python

Sentencia for

Ejemplo

```
for i in range(1,101):
  robot.forward(50, 0.5)
  robot.backward(50, 0.5)
  robot.turnRight(35, 1)
  robot.turnLeft(35, 1)
  wait(1)
```

- En este caso, la variable i toma los valores desde 1 hasta 100

Ejemplo

```
for i in range(1,101):
  robot.forward(50, 0.5)
  robot.backward(50, 0.5)
  robot.turnRight(35, 1)
  robot.turnLeft(35, 1)
  wait(1)
```

- En este caso, la variable i toma los valores desde 1 hasta 100

Ahora si..

A trabajar!! ..