Estructuras de control y subprogramas en Python

Tabla de contenido

Estructuras de control selectivas e iterativas

Un programa durante su ejecución va pasando por los diferentes pasos que ha especificado el programador. Estos pasos pueden ser:

- Secuenciales: se ejecutará un paso o línea de programación, luego la siguiente y así sucesivamente.
- Selectivos: llegamos a un paso o línea de programación que contiene una condición o expresión lógica que en función de si es verdadera o falsa puede que se ejecuten unos pasos u otros.
- Iterativos: los pasos a ejecutar se repetirán

Este tipo de pasos son las estructuras de control de programación es lo que definirá el flujo de control del programa: la secuencia de ejecución de las instrucciones del programa.

Estructuras de control selectivas

Una estructura selectiva permite, de acuerdo a una condición, ejecutar o no ciertas instrucciones.

Como el objetivo es aprender a programar en Python, veamos como pueden ser las estructuras selectivas en este lenguaje.

if

Condición es de tipo booleano, y cuando ésta se cumple se ejecuta el bloque.

```
if condicion:
    bloque
```

Nota: Cada vez que una sentencia acaba con dos puntos Python espera que la sentencia o sentencias que le siguen aparezcan con un mayor sangrado. Es la forma de marcar el inicio y el fin de una serie de sentencias que dependen de otra.

Excepcion: si solo hay una sentencia que depende de otra, pueden escribirse ambas en la misma línea.

Ejemplo: Pedir un número al usuario y si es negativo mostrar su valor absoluto.

if else

Condición es de tipo booleano, cuando ésta se cumple se ejecuta el bloque 1, cuando no se cumple se ejecuta el bloque 2.

```
if condicion:
    bloque 1
else:
    bloque 2
```

Además, tenemos la siguiente estructura, que abrevia el if else if:

Ejemplo: Pedir un número al usuario e indicar con un mensaje si es par o impar.

if elif [else]

```
if condicion 1:
    bloque 1
elif condicion 2:
    bloque 2
```

```
elif condicion 3:
    bloque 3
else:
    bloque 4
```

Ejemplo: Pedir dos números al usuario e indicar qué número es mayor que otro o si son iguales

Estructuras de control iterativas

Una estructura iterativa (bucle) engloba un conjunto de instrucciones que se ejecutan ninguna, una o tantas veces como indique una determinada condición.

Conceptualmente existen 3 tipos de bucles:

- Desde (número de iteraciones conocido)
- Mientras (0 o más iteraciones) y
- Repetir (1 o más iteraciones)

Bucle for

- Permite ejecutar una sentencia o bloque de sentencias un numero conocido de veces.
- Itera sobre una lista de valores conocidos, bien numéricos (bastante frecuente) o de otro tipo.
- Una variable de control toma sucesivamente todos los valores de la lista.

El cuerpo del bucle se ejecuta tantas veces como elementos tenga el elemento recorrible (elementos de una lista o de un range(), caracteres de una cadena, etc.).

Veamos en Python como es la sintaxis de un *bucle for*:

for

```
for variable in elemento iterable (lista, cadena, range, etc.):
    cuerpo del bucle
```

Ejemplo: Mostrar por pantalla tres veces el mensaje Hola

Range

En la estructura de los *bucles for* a menudo se emplea una función Python que se llama *range*. Hacemos un inciso en este punto para explicar en qué consiste.

La función *range* se emplea para generar una lista de enteros

```
range(10)
```

Para comprenderlo, lo mejor es probarlo:

```
range (5) : [ 0, 1, 2, 3, 4 ] desde 0 hasta el numero menos 1
range (2,5) : [ 2, 3, 4 ] de 2 a 5 - 1
range (3, 10, 2) : [ 3, 5, 7, 9 ] de 3 a 10 menos 1 de 2 en 2
```

¿Qué obtendríamos de los siguientes ejemplos?

```
for i in range (10):
    print (i)
```

```
for i in range (1 ,20):
     print (i, end ="")
```

```
words = ['cat ', 'dog ', 'lion ']
for w in words :
    print (w, end =''/'')
```

Nota: cuando usamos la función print con el argumento end le indicamos como deseamos finalizar la línea, sino se especifica este argumento por defecto es un salto de línea "\n"

```
x = 0
for i in range (1, 20, 2):
    x += i
print (x)
```

Ejercicio 1: pedir un número al usuario y mostrar la tabla su tabla de multiplicar.

Ejercicio 2: mostrar las tablas de multiplicar de los números pares

Bucle while

El bucle while se ejecuta mientras la condición sea cierta, si la condición es falsa al inicio, el bloque no se ejecuta y se pasan a ejecutar las sentencias que le siguen.

En Python, cualquier valor entero distinto de cero es verdadero y 0 es falso. La condición también puede ser una lista o cualquier secuencia, siendo la secuencia vacía falsa.

El cuerpo del bucle debe estar sangrado, ya que de este modo Python agrupa las sentencias.

En Python la sintaxis de un bucle while es:

```
while condicion:
    bloque
```

ADVERTENCIA: habrá que modificar en algún momento la condición para que el bucle deje de ejecutarse, sino será un bucle infinito.

Ejemplo: Cuenta atrás del 10 al 1 para el despegue:

```
n = 10
while n > 0:
   print n
   n = n -1
print ('Despegue!')
```

Ejercicio: pedir números al usuario hasta que escriba un número par

Funciones

La mayoría de las sentencias en un típico programa en Pyhton están agrupadas y organizadas en funciones.

Una función es un grupo de sentencias que se ejecutan cuando se las invoca. Python proporciona muchas funciones integradas y permite a los programadores definir sus propias funciones.

Cuando llamamos o invocamos a la función podemos enviar argumentos sobre los cuales la función puede trabajar.

En Python ujna función siempre devuelve un valor de resultado, o bien None o bien un dato.

En Python las funciones son tratadas como objetos, lo que supone que se puede enviar como parámetro de una función otra función.

DEFINICIÓN DE FUNCIONES

```
def nombre-funcion (parámetros):
      instrucciones
```

Nombre-funcion: es el identificador de la función.

Parámetros es una lista opcional de identificadores que podemos enviar a la función. Si la función tiene más de un parámetro, estos se separan por comas

Por último, tenemos el cuerpo de la función, con las instrucciones necesarias para realizar su tarea. Al final de la función se devolverá el valor con la palabra return

Ejemplo de función:

```
def double (x):
      return x*2
```

PARÁMETROS

- Paso por valor: se envía simplemente el valor de la variable, en cuyo caso el módulo no puede modificar la variable, pues el módulo solo conoce su valor, pero no la variable que lo almacenaba.
- Paso por referencia: se envía la dirección de memoria de la variable, en cuyo caso el módulo s que puede modificar la variable.

En Python no se hace ni una cosa ni otra. En Python cuando se envía una variable como argumento en una llamada a un módulo lo que se envía es la referencia al objeto al que hace referencia la variable. Dependiendo de si el objeto es mutable o inmutable, el módulo podrá modificar o no el objeto.

```
def aumenta(x):
    print(id(x))
    x += 1
    print(id(x))
    return x

a = 3
print(id(3), id(4))
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

```
505894336 505894352
505894336
505894336
505894352
4
3
505894336
```

En el ejemplo siguiente, la variable enviada a la funcin es una variable que hace referencia a un objeto mutable

```
def aumenta(x):
    print(id(x))
    x += [1]
    print(id(x))
    return x

a = [3]
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

```
45688512
45688512
45688512
[3, 1]
[3, 1]
45688512
```

Funciones predefinidas

Todos los lenguajes de programación disponen de un conjunto de funciones ya creadas que nos facilitan realizar ciertas operaciones. En este apartado, se mostrarán algunas funciones que nos serán de utilidad a lo largo del curso.

Funciones de cadenas en Python

Función	Utilidad	Ejemplo	Resultado
print()	Imprime en pantalla el argumento.	**print («Hola»)**	«Hola»
len()	Determina la longitud en caracteres de una cadena.	**len(«Hola Python»)**	11
join()	Convierte en cadena utilizando una separación	**Lista = [‘Python’, ‘es’]** **‘-‘.join(Lista)**	‘Python-es’
split()	Convierte una cadena con un separador en una lista	**a = («hola esto sera una lista»)** **Lista2 = a.split()** **print (Lista2)**	[‘hola’, ‘esto’, ‘sera’, ‘una’, ‘lista’]
replace()	Reemplaza una cadena por otra	**texto = «Manuel es mi amigo»** **print (texto.replace (‘es’, ‘era’))**	Manuel era mi amigo
upper()	Convierte una cadena en Mayúsculas	**texto = «Manuel es mi amigo»** **texto.upper()**	‘MANUEL ES MI AMIGO’
lower()	Convierte una cadena en Minúsculas	**texto = «MaNueL eS mI AmIgO»** **texto.lower()**	‘manuel es mi amigo’

Funciones numéricas en Python

Función	Utilidad	Ejemplo	Resultado
range()	Crea un rango de números	**x = range (5)** **print (list(x))**	[0, 1, 2, 3, 4]
str()	Convierte un valor numérico a texto	**str(22)**	’22’
int()	Convierte a valor entero	**int(’22’)**	22
float()	Convierte un valor a decimal	**float(‘2.22’)**	2.22

Función	Utilidad	Ejemplo	Resultado
max()	Determina el máximo entre un grupo de números	**x = [0, 1, 2]** **print (max(x))**	2
min()	Determina el mínimo entre un grupo de números	**x = [0, 1, 2]** **print (min(x))**	0
sum()	Suma el total de una lista de números	**x = [0, 1, 2]** **print (sum(x))**	3

Otras funciones útiles en Python

Función	Utilidad	Ejemplo	Resultado
list()	Crea una lista a partir de un elemento	**x = range (5)** **print (list(x))**	[0, 1, 2, 3, 4]
tuple()	Crea o convierte en una tupla	**print(tuple(x))**	(0, 1, 2, 3, 4)
open()	Abre, crea, edita un elemento (archivo)	**with open(«Ejercicios/Ejercicio.py», «w») as variables:** **variables.writelines(«Eje»)**	Crea el archivo «Ejercicio.py» con el contenido «Eje»
ord()	Devuelve el valor ASCII de una cadena o carácter.	**print(ord('A'))**	65
round()	Redondea después de la coma de un decimal	**print (round(12.723))**	13
type()	Devuelve el tipo de un elemento	**type(x)**	<class 'range'>
input()	Permite la entrada de datos al usuario en Python 3	**y = int(input(«Ingrese el número»))** **print (y)**	3 3

Para consultar todas las funciones predefinidas de Python consultar: https://docs.python.org/3/library/functions.html