2. Estructuras de control

May 5, 2014

1 if

```
In [3]: a = 10

        if a < 100:
            print("a es chico")
```

```
a es chico
```

En la condición del `if`, *no* son necesarias las paréntesis.

```
In [4]: a = 1000

        if a < 100:
            print("a es chico")
```

```
In [8]: a = 10

        if a < 100:
            print("a es chico")
            print("es decir, cerca de 0")

        else:
            print("a es grande")
```

```
a es chico
es decir, cerca de 0
```

En Python es obligatorio usar `:` para introducir un *bloque* de código, y sangría (indentación) para marcar el *extento* del bloque

Hacer un `switch`:

```
In [13]: if a < 1:
             print("Muy chico")

         elif a < 5:
             print("Chico")

         elif a < 10:
             print("Mediano")

         else:
             print("Grande")
```

```
Grande
```

Combinar condiciones:

```
In [14]: a = 3
         b = 10

In [15]: if a < 10 and b > 5:
             print("Bien")

Bien

In [16]: if a < 10 or b < 3:
             print("Tambien")

Tambien

In [18]: if not a < 10 or b > 3:
             print("No se que vaya a pasar")

No se que vaya a pasar

In [20]: a = 5
         if a == 10:
             print("a es 10")
         else:
             print("a no es 10")

a no es 10
```

2 while:

Empezamos *antes* del `while` con inicializar la variable (el contador)

```
In [24]: i = 0

         while i < 10:
             print(i)
             i += 1

0
1
2
3
4
5
6
7
8
9
```

3 for

Un `for` en Python itera sobre una lista

```
In [28]: l = [4, -5.5, "hola", [3, 4]]
```

```
In [29]: for i in l:
             print(i)

4
-5.5
hola
[3, 4]
```

Nota que en cada vuelta por el bucle, `i` tiene un tipo distinto

Para imprimir todos los números de 1 a 10: Necesitamos *crear* una lista de los números:

```
In [30]: range(10)

Out[30]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [33]: for i in range(10):
             print(i)

0
1
2
3
4
5
6
7
8
9
```

4 Funciones

Para *definir* una función propia:

```
In [43]: def f(x):
             return x*x

         def duplicar(x):
             return 2*x

In [39]: f

Out[39]: <function __main__.f>

In [40]: f(3)

Out[40]: 9
```

En Python, *nunca* especificamos de qué tipo son los argumentos de una función

```
In [44]: duplicar(3)

Out[44]: 6

In [46]: duplicar(17.2)

Out[46]: 34.4
```

```
In [47]: duplicar("hola")

Out[47]: 'holahola'

In [48]: duplicar(f)


    ---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

    <ipython-input-48-fc8fc22b04be> in <module>()
----> 1 duplicar(f)


    <ipython-input-43-42c712e8efc1> in duplicar(x)
      3
      4 def duplicar(x):
----> 5     return 2*x


    TypeError: unsupported operand type(s) for *: 'int' and 'function'
```

5 Tarea: El método babilónico

Es un método para calcular la raíz cuadrada de un número real, que consiste en un *algoritmo iterativo*:
$x_{n+1} = \frac{1}{2}(x_n + \frac{y}{x_n})$
Implementarlo:

1. Con un número dado de repeticiones
2. Hasta que converja

```
In []:
```