

## Lectura 2: El tutorial de Python (página 18)

---

En este material de lectura veremos las sentencias break, else y continue en los ciclos. Además veremos la sentencia pass.

Cantidad de páginas: 2

## Las sentencias break, else y continue en lazos

La sentencia **break**, como en C, termina el lazo **for** o **while** más anidado.

Las sentencias de lazo pueden tener una cláusula **else** que es ejecutada cuando el lazo termina, luego de agotar la lista (con **for**) o cuando la condición se hace falsa (con **while**), pero no cuando el lazo es terminado con la sentencia **break**. Se ejemplifica en el siguiente lazo, que busca números primos:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'es igual a', x, '*', n/x)
...             break
...         else:
...             # sigue el bucle sin encontrar un factor
...             print(n, 'es un numero primo')
...
2 es un numero primo
3 es un numero primo
4 es igual a 2 * 2
5 es un numero primo
6 es igual a 2 * 3
7 es un numero primo
8 es igual a 2 * 4
9 es igual a 3 * 3
```

(Sí, este es el código correcto. Fíjate bien: el **else** pertenece al ciclo **for**, no al **if**.)

Cuando se usa con un ciclo, el **else** tiene más en común con el **else** de una declaración **try** que con el de un **if**: el **else** de un **try** se ejecuta cuando no se genera ninguna excepción, y el **else** de un ciclo se ejecuta cuando no hay ningún **break**. Para más sobre la declaración **try** y excepciones, mirá [Manejando excepciones](#).

La declaración **continue**, también tomada de C, continua con la siguiente iteración del ciclo:

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Encontré un número par", num)
...         continue
...     print("Encontré un número", num)
Encontré un número par 2
Encontré un número 3
Encontré un número par 4
Encontré un número 5
Encontré un número par 6
Encontré un número 7
Encontré un número par 8
Encontré un número 9
```

## La sentencia pass

La sentencia **pass** no hace nada. Se puede usar cuando una sentencia es requerida por la sintaxis pero el programa no requiere ninguna acción. Por ejemplo:

```
>>> while True:
...     pass # Espera ocupada hasta una interrupción de teclado (Ctrl+C)
... 
```

Se usa normalmente para crear clases en su mínima expresión:

```
>>> class MyEmptyClass:
...     pass
... 
```

Otro lugar donde se puede usar `pass` es como una marca de lugar para una función o un cuerpo condicional cuando estás trabajando en código nuevo, lo cual te permite pensar a un nivel de abstracción mayor. El `pass` se ignora silenciosamente:

```
>>> def initlog(*args):
...     pass    # Acordate de implementar esto!
... 
```