Python

Guía para ser un Pythonista

Juan José Lozano Gómez – j2logo.com

7 Sentencias de control de flujo

Índice de contenidos

Introducción

Bueno, pues comenzamos un nuevo bloque del curso.

En el bloque anterior hemos realizado un repaso sobre los aspectos básicos del lenguaje: características, expresiones, operadores, tipos primitivos, entrada/salida, etc. Además, hemos implementado los primeros programas en Python para resolver problemas matemáticos.

Este nuevo bloque está compuesto únicamente por un tema, el que estás leyendo ahora mismo.

Aquí vamos a aprender qué sentencias de control de flujo ofrece Python.

Como ya sabrás, las sentencias o instrucciones de control de flujo permiten alterar la ejecución secuencial de un programa, de manera que es posible implementar una lógica más compleja.

Básicamente, Python pone a disposición del programador tres sentencias de este tipo: `if`, `while` y `for`.

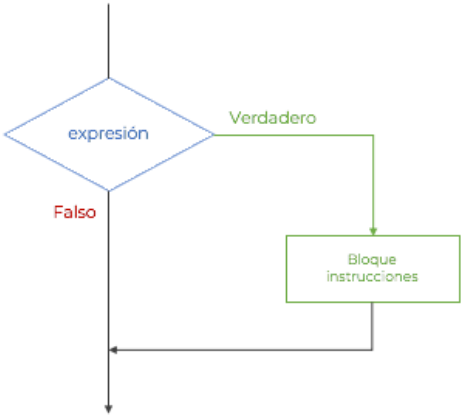
Pues no te entretengo más y doy paso al tema.

La sentencia if

En Python, la sentencia `if` se utiliza para ejecutar un bloque de código si, y solo si, se cumple una determinada condición. Por tanto, `if` **es usado para la toma de decisiones**.

La estructura básica de esta sentencia es la siguiente:

```
if expresión condicional:
    # Se ejecuta este BloqueInstrucciones
```



Es decir, solo si la expresión condicional se evalúa a `True`, se ejecutarán las sentencias que forman parte de `BloqueInstrucciones`. En caso de que se evalúe a `False` no se ejecutará ninguna sentencia perteneciente a `BloqueInstrucciones`.

La expresión condicional puede ser un literal, una variable, el resultado de una expresión o el valor devuelto por una función.

En las expresiones es muy común usar los operadores booleanos y de comparación.

IMPORTANTE: El bloque de instrucciones del `if` está indicado con un sangrado mayor. Dicho bloque termina cuando se encuentre la primera línea con un sangrado menor.

Veamos un ejemplo:

```
x = 17
if x < 20:
    print('x es menor que 20')
```

En el código anterior la variable `x` referencia al número `17`. En la línea 2, la expresión condicional evalúa si `x` es menor que `20`. Como el valor devuelto por la expresión es `True`, se ejecuta el bloque del `if`, mostrando por pantalla la cadena `x es menor que 20`.

RECUERDA: En principio, en Python todos los objetos/instancias se evalúan a `True` a excepción de `None`, `False`, `0` de todos los tipos numéricos y secuencias/colecciones vacías, que se evalúan a `False`.

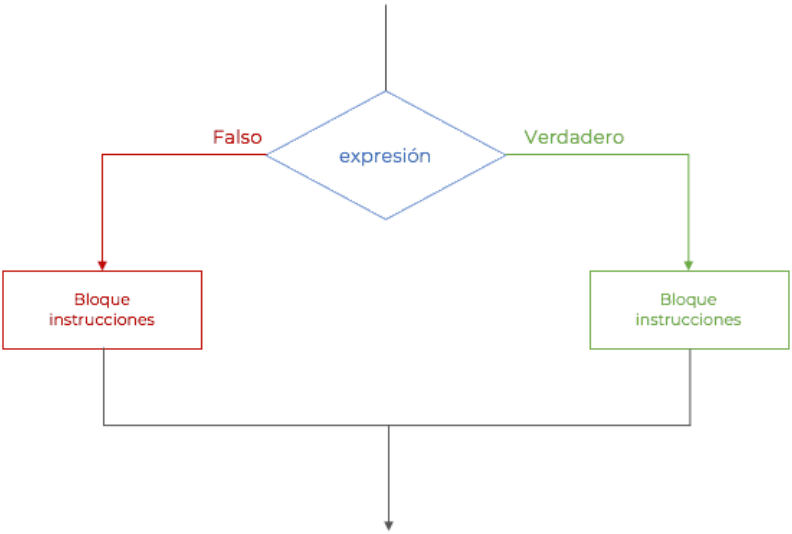
Sentencia if ... else

Hay ocasiones en que la sentencia `if` básica no es suficiente y es necesario ejecutar un conjunto de

instrucciones o sentencias cuando la expresión condicional se evalúa a `False`.

Para ello se utiliza la sentencia `if ... else...`. Su estructura es como sigue:

```
if expresión condicional:
    # Bloque de código (cuando
    # condición se evalúa a True)
else:
    # Bloque de código 2 (cuando
    # condición se evalúa a False)
```



Por ejemplo:

```
num = input('Introduce un número')
num = int(num)
if num % 2 == 0:
    print(f'{num} es par')
else:
    print(f'{num} es impar')
```

El programa anterior pide un número al usuario y muestra si el número introducido es par o impar.

if ... elif ... else

También es posible que te encuentres situaciones en que una decisión dependa de más de una condición.

En estos casos es posible utilizar una sentencia `if` compuesta, cuya estructura es como se indica a continuación:

```
if expr_cond1:
    # Bloque 1 (sentencias si expr_cond1
    # se evalúa a True)
elif expr_cond2:
    # Bloque 2 (sentencias si expr_cond1
    # se evalúa a True)
...
else:
    # Bloque n (sentencias si todas las
    # expresiones se evalúan a False)
```

Es decir, en esta ocasión, se comprueba la expresión condicional `expr_cond1`. Si se evalúa a `True`, se ejecuta el bloque de instrucciones `# Bloque 1` y después el flujo continúa tras la sentencia `if`.

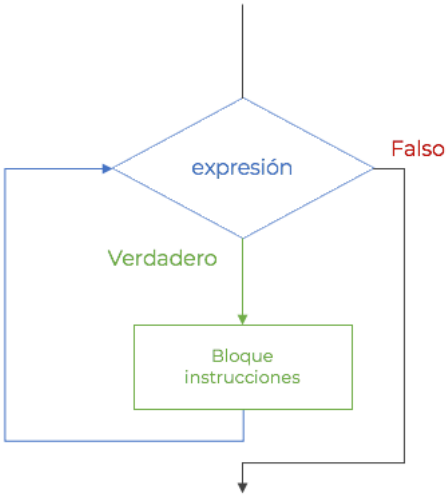
Sin embargo, si `expr_cond1` se evalúa a `False`, se comprueba la expresión `expr_cond2`. Si esta se evalúa a `True`, se ejecuta el bloque de sentencias `# Bloque 2`. En caso contrario, pasaría a comprobar la expresión del siguiente `elif` y así sucesivamente hasta que llegue al `else`, que se ejecuta si ninguna de las condiciones anteriores se evaluó a `True`.

Sentencia o bucle while

El bucle `while` se utiliza para ejecutar un bloque de instrucciones de forma continuada mientras se cumpla una condición determinada.

La estructura de la sentencia `while` es la siguiente:

```
while expresión condicional:
    # Bloque de código
```



Es decir, mientras que `expresión condicional` se evalúe a `True`, se ejecutarán las instrucciones y sentencias de `Bloque de código`.

Al igual que en la sentencia `if`, `expresión condicional` puede ser un literal, una variable, el resultado de una expresión o el valor devuelto por una función.

IMPORTANTE: El bloque de instrucciones del `while` está indicado con un sangrado mayor. Dicho bloque termina cuando se encuentre la primera línea con un sangrado menor.

En este tipo de bucle, es muy común que una variable de control forme parte de la expresión condicional. Su valor se suele actualizar en el cuerpo del bucle.

Veamos un ejemplo:

```
numero = 0
print('Tabla del 3')
while numero <= 10:
    print(f'{numero} * 3 = {numero * 3}')
    numero += 1
print('Fin')
```

En el programa anterior se inicializa una variable de control `numero` con el valor `0`.

Seguidamente se muestra un mensaje y en la línea 3 aparece una sentencia `while`. En esta sentencia la expresión condicional comprueba si el valor de la variable `numero` es menor o igual a `10`.

Como se evalúa a `True`, se muestra por pantalla el resultado de multiplicar `numero` por `3` y después se incrementa el valor de `numero` en `1`.

Tras ello, en un flujo de ejecución normal, se debería ejecutar el código de la línea 6. Sin embargo, como hemos definido un bucle `while`, el flujo vuelve a la línea 3 y de nuevo se evalúa si la expresión `numero <= 10` sigue siendo `True`. En esta ocasión el valor de `numero` es `1`, por lo que la expresión da como resultado `True` y vuelven a ejecutarse las instrucciones del bloque del `while`.

Esto será así hasta que `numero` sea igual a `11`. En esa ocasión la expresión condicional se evaluará a `False` y el flujo continuará, ahora sí, por la línea 6.

El resultado del programa anterior es el siguiente:

```
Tabla del 3
0 * 3 = 0
1 * 3 = 3
2 * 3 = 6
3 * 3 = 9
4 * 3 = 12
5 * 3 = 15
6 * 3 = 18
7 * 3 = 21
8 * 3 = 24
9 * 3 = 27
10 * 3 = 30
Fin
```

Sentencia o bucle for

En Python, el bucle `for` es un tanto especial y, seguramente, lo utilizarás con frecuencia en tus programas.

¿Por qué es especial? Porque este bucle se utiliza para recorrer, uno a uno, los elementos de un objeto de tipo iterable (*lista*, *tupla*, *conjunto*, *diccionario*, ...) y ejecutar un bloque de código. En cada paso de la iteración se tiene en cuenta a un único elemento del iterable y sobre él se suelen aplicar una serie de operaciones.

En el siguiente bloque del curso estudiaremos en detalle los tipos secuenciales e iterables, así como los diferentes usos del bucle `for`.

La estructura de la sentencia `for` es la siguiente:

```
for <elem> in <iterable>:
    # Bloque de instrucciones
```

Aquí, `elem` es la variable que referencia a un elemento del `iterable` en cada paso del bucle, el cuál, finaliza su ejecución cuando se recorren todos sus elementos.

Como te decía, es muy frecuente usar el bucle `for` para iterar sobre los elementos de listas, tuplas o diccionarios.

Imagina que tienes una lista de números y quieres mostrar por consola el cuadrado de cada uno de ellos. El código podría ser así:

```
>>> nums = [3, 5, 7]
>>> for n in nums:
…     print(f'{n}^2 = {n ** 2}')4
3^2 = 9
5^2 = 25
7^2 = 49
```

break, continue … y else

En las secciones anteriores hemos visto que:

- Las sentencias del bucle `while` se ejecutan indefinidamente hasta que la expresión condicional se evalúe a `False`.
- El bucle `for` itera sobre todos los elementos de un objeto iterable.

En Python, como en otros lenguajes de programación, es posible alterar el funcionamiento "normal" de los bucles `for` y `while` a través de las sentencias `break` y `continue`.

Veamos cómo:

- La sentencia `break` se utiliza para finalizar y salir el bucle, por ejemplo, si se cumple alguna condición.
- Por su parte, `continue` salta al siguiente paso de la iteración, ignorando todas las sentencias que le siguen y que forman parte del bucle.

- En combinación con las sentencias `break` y `continue`, a los bucles `while` y `for` se les puede añadir una cláusula `else` cuyo bloque de instrucciones se ejecutará siempre y cuando no se

haya ejecutado la sentencia `break` en el cuerpo del bucle.

Veamos un ejemplo:

```
numeros = [1, 2, 4, 3, 5, 8, 6]
for n in numeros:
    if n == 3:
        break
else:
    print('No se encontró el número 3')
```

En los videotutoriales del tema puedes ver más ejemplos de cómo usar break, continue y else con los bucles while y for.

Combinando las sentencias de control de flujo

Una vez que hemos visto las diferentes sentencias de control de flujo que tenemos disponibles, solo me queda decirte, por si no lo sabías, que estas se pueden combinar de cualquier manera.

Esto permite crear estructuras realmente potentes que resuelven una lógica compleja.

Por ejemplo, podemos anidar varias sentencias `if`:

```
num = int(input('Introduce un número'))
if num % 2 == 0:
    print(f'{num} es par')
    if num > 100:
        print(f'{num} es mayor que 100')
else:
    print(f'{num} es impar')
```

O incluir una sentencia `if` dentro de un bucle:

```
nums = [1, 3, 2, 5, 9, 4]

for n in nums:
    if n == 5:
        print('Encontrado')
```

En definitiva, puedes crear cualquier combinación que se te ocurra y/o necesites.

En los videotutoriales del tema tienes más ejemplos de cómo combinar las diferentes estructuras de control.

pass

A continuación, te voy a presentar otra sentencia especial que ofrece el lenguaje Python. Se trata de la instrucción `pass`.

`pass` es una sentencia nula, vacía, que no hace nada. A diferencia de los comentarios, que son ignorados por el intérprete, `pass` sí es tenida en cuenta. Aunque, como te digo, es una instrucción que no tiene efecto.

Se suele utilizar en aquellos sitios en los que se necesita una instrucción, pero: bien no hay nada que poner, bien se desconoce qué poner en el momento de implementar el código.

Ejemplos:

- Una clase vacía (veremos esta sentencia cuando definamos qué es una clase y en el tema de las excepciones).

```
class AppBaseError(Exception):
    pass
```

- En el cuerpo de un bucle `for`:

```
nums = [1, 28, 9, 3]
for n in nums:
    pass
```

Imagina que tienes que realizar ciertas operaciones sobre los elementos de la lista anterior, pero en dicho momento no sabes exactamente cuáles son. Si en el cuerpo del bucle no hubiéramos incluido la sentencia `pass`, el intérprete lanzaría un error.

El operador ternario

Y para finalizar el tema, te voy a presentar un nuevo operador que no incluí en el tema correspondiente a los operadores.

Lo vamos a ver aquí, ahora que sabes cómo funciona la sentencia `if`.

Aunque te pueda parecer una sentencia if en una sola línea, el operador ternario, es como su nombre indica, un operador.

La estructura de este operador es la siguiente:

```
expr1 if expresión condicional else expr2
```

Es decir, si la `expresión condicional` se resuelve como `True`, entonces se devuelve el valor de la expresión `expr1`. En caso contrario se devuelve el valor de `expr2`.

`expr1` y `expr2` pueden ser un literal, una variable, una expresión o el valor devuelto por una función.

Ejemplo:

```
>>> num = 17
>>> tipo = 'par' if num % 2 == 0 else 'impar'
>>> print(tipo)
impar
```