Artículo

Bucles for


David Aroesti 11 de Diciembre de 2019

Los bucles, en diversos lenguajes de programación pueden ser definidos o indefinidos. Los bucles definidos preestablecen las condiciones de la iteración por adelantado. Por su parte, los bucles indefinidos establecen la condición en la que una iteración terminará. En este último tipo de bucles existe el riesgo de que el bucle se vuelva infinito (cuando la condición de suspensión nunca se cumple).

Los bucles definidos se implementan en Python a través del keyword `for`. Por su parte, los bucles indefinidos se implementan con el keyword `while`.

Sin embargo, esta no es la única forma de implementar bucles definidos. Por ejemplo, Javascript puede implementar un bucle definido mediante el siguiente constructo:

```
for (i = 0; i <= 10; i++) {
  <expresión>
}
```

El bucle se puede leer de la siguiente manera:

- Inicializa el bucle en 0
- Continua el bucle mientras `i` sea menor o igual que 10
- Incrementa i en uno al final de cada iteración

Es importante señalar que la expresión `i++` es equivalente a lo que en Python escribiríamos como `i += 1`.

Una segunda forma de crear un bucle definido es iterando en una colección de objetos. Esta es la forma que Python utiliza:

```
for <variable> in <iterable>:
    <expresión>
```

El bucle for en Python

En la definición anterior debemos entender `<iterable>` como una colección de objetos; y la `<variable>` como el elemento específico que se está exponiendo mediante el bucle en cada iteración.

```
>>> frutas = ['manzana', 'pera', 'mango']
>>> for fruta in frutas:
        print(fruta)

manzana
pera
mango
```

En Python, un iterable es un objeto que se puede utilizar en un bucle definido. Si un objeto es iterable significa que se puede pasar como argumento a la función `iter`. El `iterable` que se pasa como parámetro a la función `iter`
regresa un `iterator`.

```
>>> iter('cadena') # cadena
>>> iter(['a', 'b', 'c']) # lista
>>> iter(('a', 'b', 'c')) # tupla
>>> iter({'a', 'b', 'c'}) # conjunto
>>> iter({'a': 1, 'b': 2, 'c': 3}) # diccionario
```

Todas las llamadas anteriores regresan un objeto de tipo `iterator`.

¿Qué pasa si le pasamos a la función `iter` un objeto que no es `iterable`? Obtendremos un `TypeError` que señala que el objeto no es un `iterable`. Esto es un ejemplo de programación defensiva en el que Python verifica el tipo del
objeto antes de proceder al cómputo. ¡Intentalo en tu consola!

Es importante señalar que estos no son los únicos tipos de objetos que pueden ser `iterable`. Existen gran cantidad de ejemplos en la librería estándar y, de hecho, casi cualquier objeto se puede convertir en un `iterable` (pero eso
ya lo veremos cuando hablemos de Python avanzado).

Iterators

Ahora que ya sabemos cómo obtener un `iterator`, ¿Qué podemos hacer con él? Un `iterator` es un objeto que regresa sucesivamente los valores asociados con el `iterable`.

```
>>> frutas = ['manzana', 'pera', 'mango']
>>> iterador = iter(frutas)
>>> next(iterador)
manzana
>>> next(iterador)
pera
>>> next(iterador)
mango
```

Como puedes ver, el `iterator` guarda el estado interno de la iteración, de tal manera que cada llamada sucesiva a `next` regresa el siguiente elemento. ¿Qué pasa una vez que ya no existan más elementos en el `iterable`? La llamada
a `next` arrojará un error de tipo `StopIteration`.

¿Cómo implementa Python los bucles definidos?

Ahora ya conocemos todos los elementos necesarios para entender que es lo que sucede en Python cuando ejecutamos un bucle `for`. Considera nuevamente el siguiente código:

```
>>> frutas = ['manzana', 'pera', 'mango']
>>> for fruta in frutas:
        print(fruta)
```

Este bucle se puede describir con los conceptos que explicamos previamente:

1. Python llama internamente la función `iter` para obtener un `iterator`
2. Una vez que tiene un `iterator` llama repetidamente la función `next` para tener acceso al siguiente elemento en el bucle.
3. Detiene el bucle una vez que se arroja el error `StopIteration`.

Bucles for con diccionarios

Para iterar a lo largo de un diccionario tenemos varias opciones:

- Ejecutar el bucle `for` directamente en el diccionario, lo cual nos permite iterar a lo largo de las llaves del diccionario.
- Ejecutar el bucle `for` en la llamada `keys` del diccionario, lo cual nos permite iterar a lo largo de las llaves del diccionario.
- Ejecutar el bucle `for` en la llamada `values` del diccionario, lo cual nos permite iterar a lo largo de los valores del diccionario.
- Ejecutar el bucle `for` en la llamada `items` del diccionario, lo cual nos permite iterar en una tupla de las llaves y los valores del diccionario.

```
estudiantes = {
    'mexico': 10,
    'colombia': 15,
    'puerto_rico': 4,
}

for pais in estudiantes:
    ...

for pais in estudiantes.keys():
    ...

for numero_de_estudiantes in estudiantes.values():
    ...

for pais, numero_de_estudiantes in estudiantes.items():
    ...
```

Modificación del comportamiento de un bucle for

Podemos modificar el comportamiento de un bucle `for` mediante los *keywords* `break` y `continue`.

`break` termina el bucle y permite continuar con el resto del flujo de nuestro programa.

`continue` termina la iteración en curso y continua con el siguiente ciclo de iteración.

Conclusiones

Como pudimos observar, Python implementa los bucles definidos mediante los bucles `for`. Esta implementación nos permite iterar a lo largo de cualquier objeto que sea iterable. Para iterar necesitamos un iterador que nos regresará
el siguiente valor en cada iteración. Todo esto, Python lo puede hacer por nosotros con el constructo `for ... in ...`.