Bucles. Bucles anidados

Bucles anidados

Se habla de bucles anidados cuando un **bucle** se encuentra en el **bloque de instrucciones** de otro bloque.

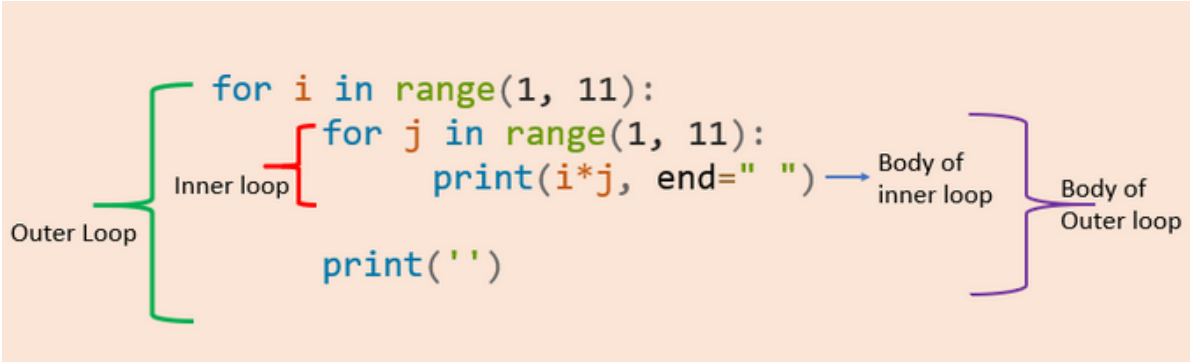
Al bucle que se encuentra dentro del otro se le puede denominar bucle interior o **bucle interno**. El otro bucle sería el bucle exterior o **bucle externo**.

Los bucles pueden tener cualquier nivel de anidamiento (un bucle dentro de otro bucle dentro de un tercero, etc.).

Aunque en Python no es necesario, se recomienda que los nombres de las variables de control de los bucles anidados no coincidan, para evitar ambigüedades.

Bucles anidados (variables independientes)

Se dice que las variables de los bucles son independientes cuando los valores que toma la **variable de control** del bucle **interno** no dependen del valor de la variable de control del bucle externo.



Por ejemplo:

Ejemplo de bucle anidado (variables independientes)

```
for i in range(3):
    for j in range(2):
        print(f"i vale {i} y j vale {j}")
```

Resultado:

```
i vale 0 y j vale 0
i vale 0 y j vale 1
i vale 1 y j vale 0
i vale 1 y j vale 1
i vale 2 y j vale 0
```

En el ejemplo anterior, el bucle externo (el controlado por `i`) se ejecuta 3 veces y el bucle interno (elcontrolado por `j`) se ejecuta 2 veces por cada valor de i.

Por ello la instrucción `print()` se ejecuta en total 6 veces (3 veces que se ejecuta el bucle externo x 2 veces que se ejecuta cada vez el bucle interno = 6 veces).

En general, el número de veces que se ejecuta el bloque de instrucciones del bucle interno es el **producto** de las veces que se ejecuta cada bucle.

Al escribir bucles anidados, hay que prestar atención al **sangrado** de las instrucciones, ya que ese sangrado indica a Python si una instrucción forma parte de un bloque u otro.

En los tres siguientes programas la única diferencia es el sangrado de la última instrucción:

Ejemplo de sangrado en bucle anidado (1)

```
for i in range(1,4):
    for j in range(11,13):
        print(j, end=" ")
        print(i, end=" ")
```

El resultado sería

```
11 1 12 1 11 2 12 2 11 3 12 3
```

En este caso, la última instrucción forma parte del cuerpo del bucle interno.

Por tanto el valor de `i` se escribe cada vez que se ejecuta el **bucle interno**.

Ejemplo de sangrado en bucle anidado (2)

```
for i in range(1, 4):
    for j in range(11, 13):
        print(j, end=" ")
    print(i, end=" ")
```

El resultado sería:

```
11 12 1 11 12 2 11 12 3
```

En este caso, la última instrucción forma parte del cuerpo del **bucle externo**, pero no del interno. Por tanto el valor de `i` se escribe cada vez que se ha terminado de ejecutar el bucle interno.

Ejemplo de sangrado en bucle anidado (3):

```
for i in range(1, 4)
    for j in range(11, 13):
        print(j, end=" ")
print(i, end=" ")
```

El resultado sería:

```
11 12 11 12 11 12 3
```

En este caso, la última instrucción no forma forma parte de ningún bucle. Por tanto el valor de `i` se escribe una sola vez, al terminarse de ejecutar el bucle externo.

La costumbre más extendida es utilizar la letra `i` como nombre de la variable de control del bucle externo y la letra `j` como nombre de la variable de control del bucle interno (o `k` si hay un tercer nivel de anidamiento), pero se puede utilizar cualquier otro nombre válido.

En Python se puede incluso utilizar la **misma variable** en los dos bucles anidados porque Python las trata como si fueran dos variables distintas (tienen ámbitos distintos).

Por ejemplo:

```
for i inrange(3):
    print(f"i (externa) vale {i}")
    for i in range(2):
        print(f"i (interna) vale {i}")
```

Daría como resultado:

```
i (externa) vale 0
i (interna) vale 0
i (interna) vale 1
i (externa) vale 1
i (interna) vale 0
i (interna) vale 1
i (externa) vale 2
i (interna) vale 0
i (interna) vale 1
```

De todas formas, este uso no se recomienda porque da lugar a programas difíciles de entender y además no es habitual en otros lenguajes de programación. Se aconseja utilizar siempre nombres de variables distintos

Bucles anidados (variables dependientes)

Se dice que las variables de los bucles son dependientes cuando los **valores** que toma la variable de control del bucle **interno dependen** del valor de la variable de control del bucle **externo**.

Por ejemplo:

Ejemplo de bucle anidado (variables dependientes) 1

```
for i in range(1, 4):
    for j in range(i):
        print(f"i vale {i} y j vale {j}")
```

Obtenemos:

```
i vale 1 y j vale 0
i vale 2 y j vale 0
i vale 2 y j vale 1
i vale 3 y j vale 0
i vale 3 y j vale 1
i vale 3 y j vale 2
```

En el ejemplo anterior, el bucle externo (el controlado por `i`) se ejecuta 3 veces y el bucle interno (el controlado por `j`) se ejecuta 1, 2 y 3 veces. Por ello la función `print()` se ejecuta en total 6 veces.

La variable `i` toma los valores de 1 a 3 y la variable `j` toma los valores de 0 a `i`, por lo que cada vez el bucle interno se ejecuta un número diferente de veces:

- Cuando `i` vale 1, `range(i)` devuelve la lista [0] y por tanto el bucle interno se ejecuta una sola vez y el programa escribe una sola línea en la que `i` vale 1 (y `j` vale 0)
- Cuando `i` vale 2, `range(i)` devuelve la lista `[0, 1]` y por tanto el bucle interno se ejecuta dos veces y el programa escribe dos líneas en la que `i` vale 2 (y `j` vale 0 o 1 en cada una de ellas)
- Cuando `i` vale 3, `range(i)` devuelve la lista `[0, 1, 2]` y por tanto el bucle interno se ejecuta tres veces y el programa escribe tres líneas en la que `i` vale 3 (y `j` vale 0, 1 o 2 en cada una de ellas).

Bucles anidados while

Los bucles anidados no tienen por qué ser bucle `for` también podemos usar la instrucción `while`

Ejemplo de bucle anidado con `while` y `for`

```
continuar = "s"
while continuar == "s":
    num = int(input("¿Cuantos números desea mostrar? "))
    for i in range(1, num):
        print(f"{i} ", end = "")
    continuar = input("\n¿Desea continuar?(s/n) ")
print("Programa terminado")
```

Referencias

Apuntes generados a partir del curso Introducción a la programación con Python que se distribuye bajo una Licencia Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional (CC BY-SA 4.0).

tags: `pro` `ut1` `python` `bucles` `for`