Capítulo 4
Bucles, listas y entradas-salidas

1. Introducción

En este capítulo, presentamos los elementos siguientes:

- **Bucles y condiciones anidadas**, que permiten que aparezcan estructuras complejas en la cadena de tratamiento.
- **Listas**, que permiten almacenar varias piezas de información y eliminar o agregar información dinámicamente.
- **Administración de archivos**, que permite leer y escribir información en el disco.

2. Las condiciones anidadas

2.1 La noción de bloques anidados

La ramificación de una instrucción `if` y el cuerpo del bucle de una instrucción `for` representan una noción similar, que agrupamos bajo el término general de subbloque de código. Los subbloques, ya provengan de una ramificación o del cuerpo de un bucle, tienen la capacidad de contener otros subbloques. Se encajan naturalmente, como las muñecas rusas.

La diferencia entre las muñecas rusas y el código de un programa es que solo existe una inclusión por cada nivel: cada muñeca solo puede contener otra muñeca de menor tamaño, que a su vez también podrá contener una sola muñeca.

Muñecas rusas anidadas

En el lenguaje Python, un bloque corresponde a líneas sucesivas de código con el mismo nivel de indentación. Dentro de esas líneas puede haber otras líneas de código de nivel de indentación más bajo. Estas líneas describen los subbloques relativos al bloque actual. Las instrucciones: `print("hello")` o `a = b + 1` no generan la aparición de subbloques en las líneas del código. Para crear un anidamiento, tiene que usar un bucle `for` o una condición `if`. En los últimos niveles de inclusión, encontramos bloques de terminales que solo contienen instrucciones simples sin bucle y sin condiciones.

La indentación natural del lenguaje Python resalta el nivel de inclusión en el que se encuentra cada línea de código. Analicemos un ejemplo para mostrar la jerarquía entre los bloques:

```
1
2  for in range (5) :                Bloque A: principal - Niv 1
3      print ("-"*10)                Bloque B: Niv 2
4      for k in range(10) :
5          if i + k == 10 :          Bloque C: Niv 3
6              print('#',end="")     Bloque D: Niv 4
7          else:
8              print(".",end="")     Bloque E: Niv 4
9      print()
10 for in range (10)
11     print("$"*10) :               Bloque F: Niv 2
12     for k in range(10) :
13         print(k,end='')           Bloque G: Niv 3
14     print("foo")
15
16 print ("terminado")
17
18
```

El bloque principal contiene tres instrucciones, de las cuales dos son bucles `for` que crean dos subbloques B y F de nivel 2 y una llamada a la función `print()`. Los bloques terminales D, E y H no contienen ningún subbloque porque no contienen ninguna instrucción `if`/`for`. Podemos ver que se encuentran en niveles diferentes. Esto es normal; el nivel de un bloque terminal depende de los bloques anteriores. De esta manera, en un programa, los bloques terminales no están todos al mismo nivel. El bloque B contiene tres instrucciones, incluido un bucle `for` lo que crea un subbloque C. El bloque C contiene una instrucción `if`/`else` que genera dos subbloques suplementarios. El bloque F contiene tres instrucciones, incluyendo un bucle `for`, que es el origen del subbloque G. Como ha podido ver, un bloque incluye tantos subbloques como palabras clave `if`/ `else` / `for`.

Cuando ejecutamos un programa en modo paso a paso, nos centramos principalmente en el bloque actual. De hecho, en cuanto se encuentre dentro de un bloque, el tiempo parece detenerse para los bloques superiores: los índices de los bucles `for` superiores no cambian y las condiciones asociadas a las instrucciones superiores `if` han sido validadas. Ahora que sabemos cómo identificar esta jerarquía de bloques, vamos a examinar su dinámica.

Cuando se encuentra una instrucción `if`:

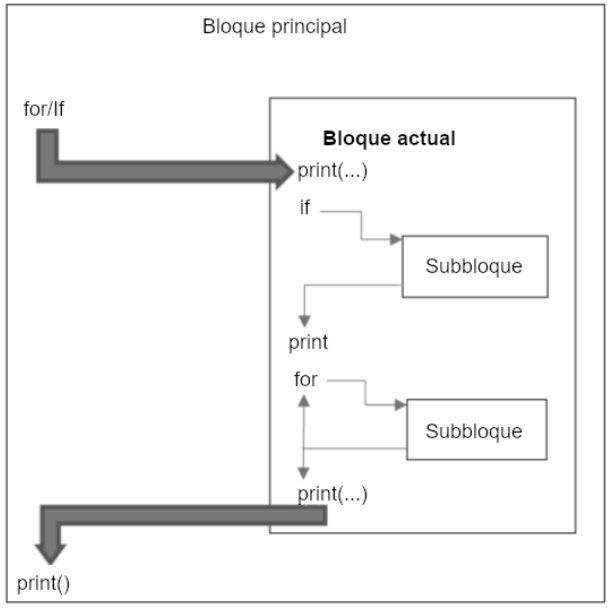
- Si su condición es verdadera:
  - entramos en el subbloque,
  - ejecutamos completamente este subbloque,
  - cuando se haya completado el subbloque, volveremos al nivel de la instrucción `if`.
- Si una instrucción se encuentra después de la instrucción `if` en el bloque actual:
  - pasamos a esta instrucción,
  - si no, salimos del bloque actual.

Cuando se encuentra una instrucción `for`:

- Mientras queden índices que recorrer en el rango de índices:
  - entramos en el subbloque,
  - ejecutamos este subbloque completamente con este valor de índice,

  - cuando se haya completado el subbloque, volveremos al nivel de la instrucción `for`.
- Si una instrucción se encuentra después de la instrucción `for` en el bloque actual:
  - pasamos a esta instrucción,
  - si no, salimos del bloque actual.

Observe que la interacción entre la instrucción `if` y la instrucción `for` con sus subbloques y con el bloque superior es casi idéntica. La única diferencia reside en que la condición `if` le permite entrar 0 o 1 veces en el subbloque, mientras que la instrucción `for` le permite entrar varias veces. He aquí hay un diagrama que resume la interacción del bloque actual con sus subbloques y el bloque superior. Las reglas presentadas le permiten recorrer por un programa en su totalidad en modo paso a paso. Son comunes a todos los demás lenguajes informáticos que utilizan palabras clave `if`/`for`:



2.2 Anidar condiciones

Le presentamos un ejemplo de condiciones anidadas `if`. Tenga en cuenta que la comparación entre dos cadenas de caracteres se hace con el operador ==.

```
01 if edad >= 18 :                    IF – empieza la ramificación 1
02    if sexo == "M" :                |     | subbloque del if
03         print("Hombre")            |     |
04    else:                           |     |
05         print("Mujer")             |     v
06    print("Fin de la subparte")     v
07 else :                            ELSE – empieza la ramificación 2
08    if edad < 12:                   |     | subbloque del else
09         print("Niño")              |     |
10    else:                           |     |
11         print("Adolescente")       v     v
12 print("Fin")
```

En la primera línea está la condición `if edad >= 18`. Supongamos que la variable `edad` tenga un valor de 20. En este caso, el resultado de la condición es verdadero. A continuación, el intérprete de Python introduce el subbloque después de la instrucción `if`. El subbloque entre las líneas 8 y 11 después de la instrucción `else` no se ejecuta en este caso. Pasamos así a la segunda línea, donde hay una segunda condición `if`: se trata del sexo de la persona. Supongamos que la variable `sexo` valga "F", por lo que el resultado de la condición `género == "M"` sería falsa. El segundo subbloque se elige luego para su ejecución. Por lo tanto, la línea `print("Hombre")` se ignorará en este caso. Entramos en el subsubbloque de la línea 5, que contiene la instrucción `print("Mujer")` para ejecutarla. Después de eso, no hay más instrucciones en este subbloque y volvemos a la instrucción `if` del bloque superior, que se encuentra en la línea 2. Como hay una instrucción del mismo nivel en la línea 6, la ejecución continúa en el subbloque actual y, por lo tanto, muestra el mensaje: "Fin de la subparte". No hay más instrucciones en este subbloque. El subbloque actual termina y volvemos a la instrucción `if` del bloque superior, a la línea 1. Tenemos una instrucción `print()` en la línea 12 que está al mismo nivel que la instrucción `if` en la línea 1, por lo que la ejecución se mueve a esa línea. Después de eso, no hay más instrucciones en el bloque principal, por lo que el programa está terminado.

Segundo escenario. Supongamos esta vez que la variable `edad` vale 16. La

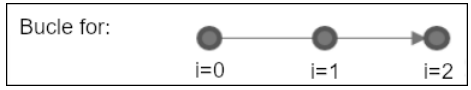
condición de la línea 1 será falsa. Como una instrucción `else` está asociada con esta instrucción `if`, ejecutaremos el subbloque después de la línea 7. Supongamos que la variable `edad` valga 10. Por lo tanto, la condición presente en la línea 8 sería verdadera y ejecutaríamos el subbloque presente en la línea 9. Después de ejecutar la instrucción `instrucción print("Niño")`, este subsubbloque se termina y volvemos a la instrucción `if` del bloque superior, a la línea 8. Como en el bloque actual no quedan instrucciones por ejecutar, volvemos a la instrucción `if` del bloque superior, a la línea 1. Tenemos una instrucción `print()` en la línea 12 que se encuentra en el mismo nivel que la instrucción `if` de la línea 1. Después, la ejecución se traslada a esa línea. Como ya no quedan más instrucciones en el bloque principal, el programa está terminado.

2.3 Anidar bucles y condiciones

La corrección de estos cuatro ejercicios está disponible para su descarga en el sitio web de la editorial con el nombre de archivo: Ejercicios con doble bucle. Un bucle `for` seguido de un comando `print()` que muestra su indice no plantea muchos problemas de comprensión porque su estructura es lineal, casi narrativa. Todo el mundo visualiza progresión que comienza desde un índice 0 y avanza paso a paso hasta alcanzar el último valor del índice. El uso de bucles de `for` anidados revela una estructura de dos dimensiones y esta configuración no es necesariamente familiar. Ahora veamos los ejemplos:

```
for i in range (0,3) :
 print(i)
```

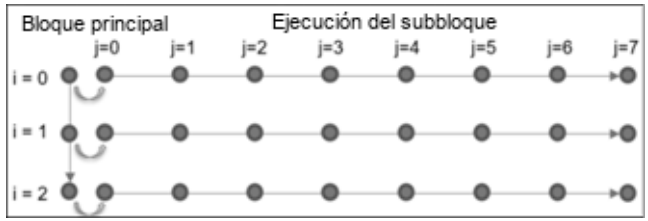
Representamos la progresión de este bucle a través de una línea y de nodos correspondientes a cada valor tomado por el índice.



Ahora veamos un doble bucle `for`:

```
for i in range(0,3) :
 print("i :",i, "--> j : ", end="") #sin salto de línea
 for j in range(8) :
   print("{0:2d}".format(j),end="")
 print()
```

Al saber cómo manejar las llamadas de subbloque, sabemos que para cada iteración del bucle `for i`, activaremos la ejecución de un subbloque que contenga un bucle `for j` que muestre los valores de este índice. Una vez que se complete el bucle `for j`, haremos un salto de línea usando la función `print()`, luego con el bloque terminado, volvemos al bloque de nivel superior. Así, volvemos a la instrucción `for i`, que volverá a ejecutar el subbloque `for j` con un nuevo valor para el índice `i`. Estas ejecuciones continuarán mientras el índice `i` tenga valores en el `range (0,3)`. Usando el diagrama anterior, obtenemos el siguiente recorrido en dos dimensiones:



He aquí el resultado obtenido al ejecutar el código anterior:

```
i : 0 --> j : 0 1 2 3 4 5 6 7
i : 1 --> j : 0 1 2 3 4 5 6 7
i : 2 --> j : 0 1 2 3 4 5 6 7
```

Ejercicio 1:

```
for i in range(4) :
 for j in range(6) :
   print("{0:3d}".format(i*10+j),end="")
 print()
```