4.2. Sentencias iterativas

Aún vamos a presentar una última reflexión sobre el programa de los menús. Cuando el usuario no escoge correctamente una opción del menú el programa le avisa, pero finaliza inmediatamente. Lo ideal sería que cuando el usuario se equivocara, el programa le pidiera de nuevo una opción. Para eso sería necesario *repetir* la ejecución de las líneas 11–21. Una aproximación naïf consistiría, básicamente, en añadir al final una copia de esas líneas precedidas de un **if** que comprobara que el usuario se equivocó. Pero esa aproximación es muy mala: ¿qué pasaría si el usuario se equivocara una segunda vez? Cuando decimos que queremos *repetir* un fragmento del programa no nos referimos a *copiarlo* de nuevo, sino a *ejecutarlo* otra vez. Pero, ¿es posible expresar en este lenguaje que queremos que se repita la ejecución de un trozo del programa?

Python permite indicar que deseamos que se repita un trozo de programa de dos formas distintas: mediante la sentencia **while** y mediante la sentencia **for**. La primera de ellas es más general, por lo que la estudiaremos en primer lugar.

4.2.1. La sentencia while

En inglés, «while» significa «mientras». La sentencia **while** se usa así:

```
while condición:
    acción
    acción
    ...
    acción
```

y permite expresar en Python acciones cuyo significado es:

«*Mientras* se cumpla esta condición, repite estas acciones.»

Las sentencias que denotan repetición se denominan *bucles*.

Vamos a empezar estudiando un ejemplo y viendo qué ocurre paso a paso. Estudia detenidamente este programa:

contador_3.py — contador.py

```
1 i = 0
2 while i < 3:
3     print i
4     i += 1
5 print 'Hecho'
```

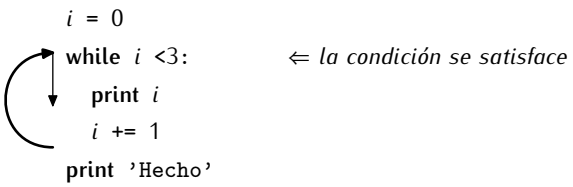
Observa que la línea 2 finaliza con dos puntos (`:`) y que la indentación indica que las líneas 3 y 4 dependen de la línea 2, pero no la línea 5. Podemos leer el programa así: primero, asigna a *i* el valor 0; a continuación, *mientras* *i* sea menor que 3, *repite estas acciones*: muestra por pantalla el valor de *i* e incrementa *i* en una unidad; finalmente, muestra por pantalla la palabra «Hecho».

Si ejecutamos el programa, por pantalla aparecerá el siguiente texto:

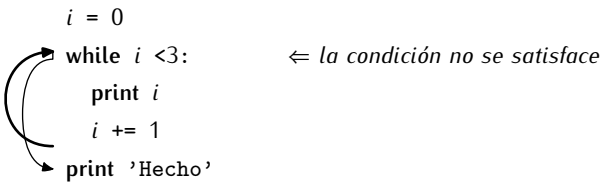
```
0
1
2
Hecho
```

Veamos qué ha ocurrido paso a paso con una traza.

- Se ha ejecutado la línea 1, con lo que *i* vale 0.
- Después, se ha ejecutado la línea 2, que dice «mientras *i* sea menor que 3, hacer...». Primero se ha evaluado la condición $i < 3$, que ha resultado ser cierta. Como la condición se satisface, deben ejecutarse las acciones supeditadas a esta línea (las líneas 3 y 4).
- Se ejecuta en primer lugar la línea 3, que muestra el valor de *i* por pantalla. Aparece, pues, un cero.
- Se ejecuta a continuación la línea 4, que incrementa el valor de *i*. Ahora *i* vale 1.
- ¡Ojo!, ahora *no* pasamos a la línea 5, sino que volvemos a la línea 2. Cada vez que finalizamos la ejecución de las acciones que dependen de un **while**, volvemos a la línea del **while**.



- Estamos nuevamente en la línea 2, así que comprobamos si *i* es menor que 3. Es así, por lo que toca ejecutar de nuevo las líneas 3 y 4.
- Volvemos a ejecutar la línea 3, así que aparece un 1 por pantalla.
- Volvemos a ejecutar la línea 4, con lo que *i* vuelve a incrementarse y pasa de valer 1 a valer 2.
- Nuevamente pasamos a la línea 2. *Siempre* que acaba de ejecutarse la última acción de un bucle **while**, volvemos a la línea que contiene la palabra ·while·. Como *i* sigue siendo menor que 3, deberemos repetir las acciones expresadas en las líneas 3 y 4.
- Así que ejecutamos otra vez la línea 3 y en pantalla aparece el número 2.
- Incrementamos de nuevo el valor de *i*, como indica la línea 4, así que *i* pasa de valer 2 a valer 3.
- Y de nuevo pasamos a la línea 2. Pero ahora ocurre algo especial: la condición no se satisface, pues *i* ya no es menor que 3. Como la condición ya no se satisface, no hay que ejecutar otra vez las líneas 3 y 4. Ahora hemos de ir a la línea 5, que es la primera línea que no está «dentro» del bucle.



- Se ejecuta la línea 5, que muestra por pantalla la palabra «Hecho» y finaliza el programa.

EJERCICIOS

▶ 94 Ejecuta el último programa paso a paso con el entorno de depuración de PythonG.

Pero, ¿por qué tanta complicación? Este otro programa muestra por pantalla lo mismo, se entiende más fácilmente y es más corto.

contador_simple.py

```
1 print 0
2 print 1
3 print 2
4 print 'Hecho'
```

Bueno, `contador.py` es un programa que sólo pretende ilustrar el concepto de bucle, así que ciertamente no hace nada demasiado útil, pero aun así nos permite vislumbrar la potencia del concepto de iteración o repetición. Piensa en qué ocurre si modificamos un sólo número del programa:

contador_4.py — contador.py

```
1 i = 0
2 while i < 1000:
3     print i
4     i += 1
5 print 'Hecho'
```

¿Puedes escribir fácilmente un programa que haga lo mismo y que no utilice bucles?

EJERCICIOS

▶ 95 Haz una traza de este programa:

ejercicio_bucle_9.py — ejercicio_bucle.py

```
1 i = 0
2 while i <= 3:
3     print i
4     i += 1
5 print 'Hecho'
```

▶ 96 Haz una traza de este programa:

ejercicio_bucle_10.py — ejercicio_bucle.py

```
1 i = 0
2 while i < 10:
3     print i
4     i += 2
5 print 'Hecho'
```

▶ 97 Haz una traza de este programa:

ejercicio_bucle_11.py — ejercicio_bucle.py

```
1 i = 3
2 while i < 10:
3     i += 2
4     print i
5 print 'Hecho'
```

▶ 98 Haz una traza de este programa:

ejercicio_bucle_12.py — ejercicio_bucle.py

```
1 i = 1
2 while i < 100:
3     i *= 2
4     print i
```

▶ 99 Haz una traza de este programa:

ejercicio_bucle_13.py — ejercicio_bucle.py

```
1 i = 10
2 while i < 2:
3     i *= 2
4     print i
```

▶ 100 Haz unas cuantas trazas de este programa para diferentes valores de i.

ejercicio_bucle_14.py — ejercicio_bucle.py

```
1 i = int(raw_input('Valor inicial: '))
2 while i < 10:
3     print i
4     i += 1
```

¿Qué ocurre si el valor de i es mayor o igual que 10? ¿Y si es negativo?

▶ 101 Haz unas cuantas trazas de este programa para diferentes valores de *i* y de *limite*.

ejercicio_bucle_15.py — ejercicio_bucle.py

```
i = int(raw_input('Valor inicial: '))
limite = int(raw_input('Límite: '))
while i < limite:
    print i
    i += 1
```

▶ 102 Haz unas cuantas trazas de este programa para diferentes valores de *i*, de *limite* y de *incremento*.

ejercicio_bucle_16.py — ejercicio_bucle.py

```
i = int(raw_input('Valor inicial: '))
limite = int(raw_input('Límite: '))
incremento = int(raw_input('Incremento: '))
while i < limite:
    print i
    i += incremento
```

▶ 103 Implementa un programa que muestre todos los múltiplos de 6 entre 6 y 150, ambos inclusive.

▶ 104 Implementa un programa que muestre todos los múltiplos de n entre n y $m \cdot n$, ambos inclusive, donde n y m son números introducidos por el usuario.

▶ 105 Implementa un programa que muestre todos los números potencia de 2 entre 2^0 y 2^{30}, ambos inclusive.

Bucles sin fin

Los bucles son muy útiles a la hora de confeccionar programas, pero también son peligrosos si no andas con cuidado: es posible que no finalicen nunca. Estudia este programa y verás qué queremos decir:

bucle_infinito.py

```
i = 0
while i < 10:
    print i
```

La condición del bucle siempre se satisface: dentro del bucle nunca se modifica el valor de *i*, y si *i* no se modifica, jamás llegará a valer 10 o más. El ordenador empieza a mostrar el número 0 una y otra vez, sin finalizar nunca. Es lo que denominamos un *bucle sin fin* o *bucle infinito*.

Cuando se ejecuta un bucle sin fin, el ordenador se queda como «colgado» y nunca nos devuelve el control. Si estás ejecutando un programa desde la línea de órdenes Unix, puedes abortarlo pulsando `C-c`. Si la ejecución tiene lugar en el entorno PythonG (o en el editor XEmacs) puedes abortar la ejecución del programa con `C-c C-c`.