4.2.9. Anidamiento de estructuras

Ahora vamos a resolver otro problema. Vamos a hacer que el programa pida un número y nos muestre por pantalla los números primos entre 1 y el que hemos introducido. Mira este programa:

primos.py

```
1  limite = int(raw_input('Dame un número: '))
2
3  for num in range(1, limite+1):
4      creo_que_es_primo = True
5      for divisor in range(2, num):
6          if num % divisor == 0:
7              creo_que_es_primo = False
8              break
9      if creo_que_es_primo:
10         print num
```

No debería resultarte difícil entender el programa. Tiene *bucles anidados* (un **for-in** dentro de un **for-in**), pero está claro qué hace cada uno de ellos: el más exterior recorre con *num* todos los números comprendidos entre 1 y *limite* (ambos inclusive); el más interior forma parte del procedimiento que determina si el número que estamos estudiando en cada instante es o no es primo.

Dicho de otro modo: *num* va tomando valores entre 1 y *limite* y *para cada valor* de *num* se ejecuta el bloque de las líneas 4–10, así que, *para cada* valor de *num*, se comprueba si éste es primo o no. Sólo si el número resulta ser primo se muestra por pantalla.

Puede que te intrigue el **break** de la línea 8. ¿A qué bucle «rompe»? Sólo al más interior: una sentencia **break** siempre aborta la ejecución de un solo bucle y éste es el que lo contiene directamente.

Antes de acabar: existen procedimientos más eficientes para determinar si un número es primo o no, así como para listar los números primos en un intervalo. Hacer buenos programas no sólo pasa por conocer bien las reglas de escritura de programas en un lenguaje de programación: has de saber diseñar algoritmos y, muchas veces, buscar los mejores algoritmos conocidos en los libros.

EJERCICIOS

▶ 136 ¿Qué resultará de ejecutar estos programas?

a) ejercicio_for_7.py

```
1  for i in range(0, 5):
2      for j in range(0, 3):
3          print i, j
```

b) ejercicio_for_8.py

```
1  for i in range(0, 5):
2      for j in range(i, 5):
3          print i, j
```

c) ejercicio_for_9.py

```
1  for i in range(0, 5):
2      for j in range(0, i):
3          print i, j
```

Índice de bucle for-in: ¡prohibido asignar!

Hemos aprendido que el bucle **for-in** utiliza una variable índice a la que se van asignando los diferentes valores del rango. En muchos ejemplos se utiliza la variable *i*, pero sólo porque también en matemáticas los sumatorios y productorios suelen utilizar la letra *i* para indicar el nombre de su variable índice. Puedes usar cualquier nombre de variable válido.

Pero que el índice sea una variable cualquiera no te da libertad absoluta para hacer con ella lo que quieras. En un bucle, las variables de índice sólo deben usarse para consultar su valor, nunca para asignarles uno nuevo. Por ejemplo, este fragmento de programa es incorrecto:

```
for i in range(0, 5):
    i += 2
```

Y ahora que sabes que los bucles pueden anidarse, también has de tener mucho cuidado con sus índices. Un error frecuente entre primerizos de la programación es utilizar el mismo índice para dos bucles anidados. Por ejemplo, estos bucles anidados están mal:

```
for i in range(0, 5):
    for i in range(0, 3):
        print i
```

En el fondo, este problema es una variante del anterior, pues de algún modo se está asignando nuevos valores a la variable *i* en el bucle interior, pero *i* es la variable del bucle exterior y asignarle cualquier valor está prohibido.

Recuerda: *nunca debes asignar un valor a un índice de bucle ni usar la misma variable índice en bucles anidados.*

d) ejercicio_for_10.py — ejercicio_for.py

```
for i in range(0, 4):
    for j in range(0, 4):
        for k in range(0, 2):
            print i, j, k
```

e) ejercicio_for_11.py — ejercicio_for.py

```
for i in range(0, 4):
    for j in range(0, 4):
        for k in range(i, j):
            print i, j, k
```

f) ejercicio_for_12.py — ejercicio_for.py

```
for i in range(1, 5):
    for j in range(0, 10, i):
        print i, j
```