4.2.7. Números primos

Vamos ahora con un ejemplo más. Nos proponemos construir un programa que nos diga si un número (entero) es o no es primo. Recuerda: un número primo es aquel que sólo es divisible por sí mismo y por 1.

¿Cómo empezar? Resolvamos un problema concreto, a ver qué estrategia seguiríamos normalmente. Supongamos que deseamos saber si 7 es primo. Podemos intentar dividirlo por cada uno de los números entre 2 y 6. Si alguna de las divisiones es exacta, entonces el número *no* es primo:

Dividendo	Divisor	Cociente	Resto
7	2	3	1
7	3	2	1
7	4	1	3
7	5	1	2
7	6	1	1

Ahora estamos seguros: ninguno de los restos dio 0, así que 7 es primo. Hagamos que el ordenador nos muestre esa misma tabla:

es_primo_10.py — es_primo.py

```
num = 7

for divisor in range(2, num):
    print '%d entre %d' % (num, divisor) ,
    print 'es %d con resto %d' % (num / divisor, num % divisor)
```

(Recuerda que *range*(2,*num*) comprende todos los números enteros entre 2 y *num* – 1.) Aquí tienes el resultado de ejecutar el programa:

```
7 entre 2 es 3 con resto 1
7 entre 3 es 2 con resto 1
7 entre 4 es 1 con resto 3
7 entre 5 es 1 con resto 2
7 entre 6 es 1 con resto 1
```

Está claro que probar todas las divisiones es fácil, pero, ¿cómo nos aseguramos de que *todos* los restos son distintos de cero? Una posibilidad es contarlos y comprobar que hay exactamente *num* – 2 restos no nulos:

es_primo_11.py — es_primo.py

```
num = 7

restos_no_nulos = 0
for divisor in range(2, num):
    if num % divisor != 0:
        restos_no_nulos += 1

if restos_no_nulos == num - 2:
    print 'El número', num, 'es primo'
else:
    print 'El número', num, 'no es primo'
```

Pero vamos a proponer un método distinto basado en una «idea feliz» y que, más adelante, nos permitirá acelerar notabilísimamente el cálculo. Vale la pena que la estudies bien: la utilizarás siempre que quieras probar que *toda una serie* de valores cumple una propiedad. En nuestro caso la propiedad que queremos demostrar que cumplen todos los números entre 2 y *num*–1 es «al dividir a *num*, da resto distinto de cero».

- Empieza siendo optimista: supón que la propiedad es cierta y asigna a una variable el valor «cierto».
- Recorre todos los números y cuando alguno de los elementos de la secuencia no satisfaga la propiedad, modifica la variable antes mencionada para que contenga el valor «falso».
- Al final del todo, mira qué vale la variable: si aún vale «cierto», es que nadie la puso a «falso», así que la propiedad se cumple para todos los elementos y el número es primo; y si vale «falso», entonces alguien la puso a «falso» y para eso es preciso que algún elemento no cumpliera la propiedad en cuestión, por lo que el número no puede ser primo.

Mira cómo plasmamos esa idea en un programa:

es_primo_12.py — es_primo.py

```
num = 7

```

```
3  creo_que_es_primo = True
4  for divisor in range(2, num):
5      if num % divisor == 0:
6          creo_que_es_primo = False
7
8  if creo_que_es_primo:
9      print 'El número', num, 'es primo'
10 else:
11     print 'El número', num, 'no es primo'
```

EJERCICIOS

▶ 126 Haz un traza del programa para los siguientes números:

a) 4 b) 13 c) 18 d) 2 (¡ojo con éste!)

True == True

Fíjate en la línea 8 de este programa:

es_primo.py

```
1  num = 7
2
3  creo_que_es_primo = True
4  for divisor in range(2, num):
5      if num % divisor == 0:
6          creo_que_es_primo = False
7
8  if creo_que_es_primo:
9      print 'El número', num, 'es primo'
10 else:
11     print 'El número', num, 'no es primo'
```

La condición del **if** es muy extraña, ¿no? No hay comparación alguna. ¿Qué condición es esa? Muchos estudiantes optan por esta fórmula alternativa para las líneas 8 y sucesivas

es_primo.py

```
⋮
8  if creo_que_es_primo == True:
9      print 'El número', num, 'es primo'
10 else:
11     print 'El número', num, 'no es primo'
```

Les parece más natural porque de ese modo se compara el valor de *creo_que_es_primo* con algo. Pero, si lo piensas bien, esa comparación es superflua: a fin de cuentas, el resultado de la comparación *creo_que_es_primo* == *True* es *True* y, directamente, *creo_que_es_primo* vale *True*.

No es que esté mal efectuar esa comparación extra, sino que no aporta nada y resta legibilidad. Evítala si puedes.

Después de todo, no es tan difícil. Aunque esta idea feliz la utilizarás muchas veces, es probable que cometas un error (al menos, muchos compañeros tuyos caen en él una y otra vez). Fíjate en este programa, que está mal:

es_primo_13.py — es_primo.py

```
num = 7

creo_que_es_primo = True
for divisor in range(2, num):
    if num % divisor == 0:
        creo_que_es_primo = False
    else:
        creo_que_es_primo = True

if creo_que_es_primo:
    print 'El número', num, 'es primo'
else:
    print 'El número', num, 'no es primo'
```

¡El programa sólo se acuerda de lo que pasó con el último valor del bucle! Haz la prueba: haz una traza sustituyendo la asignación de la línea 1 por la sentencia *num* = 4. El número *no* es primo, pero al no ser exacta la división entre 4 y 3 (el último valor de *divisor* en el bucle), el valor de *creo_que_es_primo* es *True*. El programa concluye, pues, que 4 es primo.

Vamos a refinar el programa. En primer lugar, haremos que trabaje con cualquier número que el usuario introduzca:

es_primo_14.py — es_primo.py

```
num = int(raw_input('Dame un número: '))

creo_que_es_primo = True
for divisor in range(2, num):
    if num % divisor == 0:
        creo_que_es_primo = False

if creo_que_es_primo:
    print 'El número', num, 'es primo'
else:
    print 'El número', num, 'no es primo'
```

Fácil. Ahora vamos a hacer que vaya más rápido. Observa qué ocurre cuando tratamos de ver si el número 1024 es primo o no. Empezamos dividiéndolo por 2 y vemos que el resto de la división es cero. Pues ya está: estamos seguros de que 1024 no es primo. Sin embargo, nuestro programa sigue haciendo cálculos: pasa a probar con el 3, y luego con el 4, y con el 5, y así hasta llegar al 1023. ¿Para qué, si ya sabemos que no es primo? Nuestro objetivo es que el bucle deje de ejecutarse tan pronto estemos seguros de que el número no es primo. Pero resulta que no podemos hacerlo con un bucle **for-in**, pues este tipo de bucles se basa en nuestro conocimiento *a priori* de cuántas iteraciones vamos a hacer. Como en este caso no lo sabemos, hemos de utilizar un bucle **while**. Escribamos primero un programa equivalente al anterior, pero usando un **while** en lugar de un **for-in**:

es_primo_15.py — es_primo.py

```
num = int(raw_input('Dame un número: '))

creo_que_es_primo = True
divisor = 2
while divisor < num:
    if num % divisor == 0:
        creo_que_es_primo = False
    divisor += 1

if creo_que_es_primo:
```

Se cumple para todos/se cumple para alguno

Muchos de los programas que diseñaremos necesitan verificar que cierta condición se cumple para algún elemento de un conjunto o para todos los elementos del conjunto. En ambos casos tendremos que recorrer todos los elementos, uno a uno, y comprobar si la condición es cierta o falsa para cada uno de ellos.

Cuando queramos comprobar que *todos* cumplen una condición, haremos lo siguiente:

1. Seremos *optimistas* y empezaremos suponiendo que la condición se cumple para todos.
2. Preguntaremos a cada uno de los elementos si cumple la condición.
3. Sólo cuando detectemos que uno de ellos *no la cumple*, cambiaremos de opinión y pasaremos a saber que la condición no se cumple para todos. *Nada nos podrá hacer cambiar de opinión.*

He aquí un esquema que usa la notación de Python:

```
creo_que_se_cumple_para_todos = True
for elemento in conjunto:
    if not condición:
        creo_que_se_cumple_para_todos = False

if creo_que_se_cumple_para_todos:
    print 'Se cumple para todos'
```

Cuando queramos comprobar que *alguno* cumple una condición, haremos lo siguiente:

1. Seremos *pesimistas* y empezaremos suponiendo que la condición no se cumple para ninguno.
2. Preguntaremos a cada uno de los elementos si se cumple la condición.
3. Sólo cuando detectemos que uno de ellos *sí la cumple*, cambiaremos de opinión y pasaremos a saber que la condición se cumple para alguno. *Nada nos podrá hacer cambiar de opinión.*

He aquí un esquema que usa la notación de Python:

```
creo_que_se_cumple_para_alguno = False
for elemento in conjunto:
    if condición:
        creo_que_se_cumple_para_alguno = True

if creo_que_se_cumple_para_alguno:
    print 'Se cumple para alguno'
```

```
    print 'El número', num, 'es primo'
else:
    print 'El número', num, 'no es primo'
```

EJERCICIOS

▶ 127 Haz una traza del último programa para el número 125.

Hemos sustituido el **for-in** por un **while**, pero no hemos resuelto el problema: con el 1024 seguimos haciendo todas las pruebas de divisibilidad. ¿Cómo hacer que el bucle acabe tan pronto se esté seguro de que el número no es primo? Pues complicando un poco la condición del **while**:

Error para alguno/error para todos

Ya te hemos dicho que muchos de los programas que diseñaremos necesitan verificar que cierta condición se cumple para algún elemento de un conjunto o para todos los elementos del conjunto. Y también te hemos dicho cómo abordar ambos problemas. Pero, aun así, es probable que cometas un error (muchos, muchos estudiantes lo hacen). Aquí tienes un ejemplo de programa erróneo al tratar de comprobar que una condición se cumple para todos los elementos de un conjunto:

```
creo_que_se_cumple_para_todos = True
for elemento in conjunto:
    if not condición:
        creo_que_se_cumple_para_todos = False
    else: # Esta línea y la siguiente sobran
        creo_que_se_cumple_para_todos = True

if creo_que_se_cumple_para_todos:
    print 'Se cumple para todos'
```

Y aquí tienes una versión errónea para el intento de comprobar que una condición se cumple para alguno:

```
creo_que_se_cumple_para_alguno = False
for elemento in conjunto:
    if condición:
        creo_que_se_cumple_para_alguno = True
    else: # Esta línea y la siguiente sobran
        creo_que_se_cumple_para_alguno = False

if creo_que_se_cumple_para_alguno:
    print 'Se cumple para alguno'
```

En ambos casos, sólo se está comprobando si *el último* elemento del conjunto cumple o no la condición.

es_primo_16.py — es_primo.py

```
num = int(raw_input('Dame un número: '))

creo_que_es_primo = True
divisor = 2
while divisor < num and creo_que_es_primo:
    if num % divisor == 0:
        creo_que_es_primo = False
    divisor += 1

if creo_que_es_primo:
    print 'El número', num, 'es primo'
else:
    print 'El número', num, 'no es primo'
```

Ahora sí.

EJERCICIOS

▶ 128 Haz una traza del último programa para el número 125.

▶ 129 Haz un programa que calcule el máximo común divisor (mcd) de dos enteros positivos. El mcd es el número más grande que divide exactamente a ambos números.

▶ 130 Haz un programa que calcule el máximo común divisor (mcd) de tres enteros