5.1.6. Un ejemplo: un contador de palabras

Ahora que tenemos nuevas herramientas para la manipulación de cadenas, vamos a desarrollar un programa interesante: leerá cadenas de teclado y mostrará en pantalla el número de palabras que contienen.

Empecemos estudiando el problema con un ejemplo concreto. ¿Cuántas palabras hay en la cadena `'una␣dos␣tres'`? Tres palabras. ¿Cómo lo sabemos? Muy fácil: contando el número de espacios en blanco. Si hay *dos* espacios en blanco, entonces hay *tres* palabras, ya que cada espacio en blanco separa dos palabras. Hagamos, pues, que el programa cuente el número de espacios en blanco y muestre ese número más uno:

palabras_5.py — palabras.py

```
1  cadena = raw_input('Escribe una frase: ')
2  while cadena != '':
3      blancos = 0
4      for caracter in cadena:
5          if caracter == ' ':
6              blancos += 1
7      palabras = blancos + 1 # Hay una palabra más que blancos
8      print 'Palabras:', palabras
9
10     cadena = raw_input('Escribe una frase: ')
```

El programa finaliza la ejecución cuando teclamos una cadena vacía, es decir, si pulsamos retorno de carro directamente. Ejecutemos el programa:

```
Escribe una frase: una dos tres
Palabras: 3
Escribe una frase: mi ejemplo
Palabras: 2
Escribe una frase: ejemplo
Palabras: 1
Escribe una frase: otro  ejemplo
Palabras: 3
```

¡Eh! ¿Qué ha pasado con el último ejemplo? Hay dos palabras y el programa dice que hay tres. Está claro: entre las palabras «**otro**» y «**ejemplo**» de la cadena `'otro  ejemplo'` hay *dos* espacios en blanco, y no uno solo. Corrijamos el programa para que trate correctamente casos como éste. Desde luego, contar espacios en blanco, sin más, no es la clave para decidir cuántas palabras hay. Se nos ocurre una idea mejor: mientras recorremos la cadena, veamos cuántas veces pasamos de un carácter que no sea el espacio en blanco a un espacio en blanco. En la cadena `'una dos tres'` pasamos *dos* veces de letra a espacio en blanco (una vez pasamos de la «**a**» al blanco y otra de la «**s**» al blanco), y hay *tres* palabras; en la cadena problemática `'otro  ejemplo'` sólo pasamos *una* vez de la letra «**o**» a un espacio en blanco y, por tanto, hay *dos* palabras. Si contamos el número de transiciones, el número de palabras será ese mismo número más uno. ¿Y cómo hacemos para comparar un carácter y su vecino? El truco está en recordar siempre cuál era el carácter anterior usando una variable auxiliar:

palabras_6.py — palabras.py

```
1  cadena = raw_input('Escribe una frase: ')
2  while cadena != '':
3      cambios = 0
4      anterior = ''
5      for caracter in cadena:
6          if caracter == ' ' and anterior != ' ':
7              cambios += 1
8          anterior = caracter
9      palabras = cambios + 1 # Hay una palabra más que cambios de no blanco a blanco
10     print 'Palabras:', palabras
11
12     cadena = raw_input('Escribe una frase: ')
```

¿Por qué hemos dado un valor a *anterior* en la línea 4? Para inicializar la variable. De no hacerlo, tendríamos problemas al ejecutar la línea 6 por primera vez, ya que en ella se consulta el valor de *anterior*.

EJERCICIOS

▶ 162 Haz una traza del programa para la cadena `'a␣b'`. ¿Qué líneas se ejecutan y qué valores toman las variables *cambios*, *anterior* y *caracter* tras la ejecución de cada una de ellas?

▶ 163 Ídem para la cadena `'a␣␣b'`.

Probemos nuestra nueva versión:

```
Escribe una frase: una␣dos␣tres
Palabras: 3
Escribe una frase: mi␣ejemplo
Palabras: 2
Escribe una frase: ejemplo
Palabras: 1
Escribe una frase: otro␣␣ejemplo
Palabras: 2
Escribe una frase: ejemplo␣
Palabras: 2
```

¡No! ¡Otra vez mal! ¿Qué ha ocurrido ahora? Si nos fijamos bien veremos que la cadena del último ejemplo acaba en un espacio en blanco, así que hay una transición de «no blanco» a espacio en blanco y eso, para nuestro programa, significa que hay una nueva palabra. ¿Cómo podemos corregir ese problema? Analicémoslo: parece que sólo nos molestan los blancos *al final* de la cadena. ¿Y si descontamos una palabra cuando la cadena acaba en un espacio en blanco?

palabras_7.py — palabras.py

```
1  cadena = raw_input('Escribe␣una␣frase:␣')
2  while cadena != '':
3      cambios = 0
4      anterior = ''
5      for caracter in cadena:
6          if caracter == '␣' and anterior != '␣':
7              cambios += 1
8          anterior = caracter
9
10     if cadena[-1] == '␣':
11         cambios -= 1
12
13     palabras = cambios + 1
14     print 'Palabras:', palabras
15
16     cadena = raw_input('Escribe␣una␣frase:␣')
```

Probemos ahora esta nueva versión:

```
Escribe una frase: una␣dos␣tres
Palabras: 3
Escribe una frase: mi␣ejemplo
Palabras: 2
Escribe una frase: ejemplo
Palabras: 1
Escribe una frase: otro␣␣ejemplo
Palabras: 2
Escribe una frase: ejemplo␣
Palabras: 1
```

¡Perfecto! Ya está. ¿Seguro? Mmmm. Los espacios en blanco dieron problemas al final de la cadena. ¿Serán problemáticos también al principio de la cadena? Probemos:

```
Escribe una frase: ␣ejemplo␣
Palabras: 2
```

Sí, ¡qué horror! ¿Por qué falla ahora? El problema radica en la inicialización de *anterior* (línea 4). Hemos dado una cadena vacía como valor inicial y eso hace que, si la cadena empieza por un blanco, la condición de la línea 6 se evalúe a *cierto* para el primer carácter, incrementando así la variable *cambios* (línea 7) la primera vez que iteramos el bucle. Podríamos evitarlo modificando la inicialización de la línea 4: un espacio en blanco nos vendría mejor como valor inicial de *anterior*.

palabras_8.py — palabras.py

```
1  cadena = raw_input('Escribe␣una␣frase:␣')
2  while cadena != '':
3      cambios = 0
4      anterior = '␣'
5      for caracter in cadena:
6          if caracter == '␣' and anterior != '␣':
7              cambios += 1
8          anterior = caracter
9
10     if cadena[-1] == '␣':
11         cambios = cambios - 1
12
13     palabras = cambios + 1
14     print 'Palabras:', palabras
15
16     cadena = raw_input('Escribe␣una␣frase:␣')
```

Ahora sí:

```
Escribe una frase: una␣dos␣tres
Palabras: 3
Escribe una frase: mi␣ejemplo
Palabras: 2
Escribe una frase: ejemplo
Palabras: 1
Escribe una frase: otro␣␣ejemplo
Palabras: 2
Escribe una frase: ejemplo␣
Palabras: 1
Escribe una frase: ␣ejemplo␣
Palabras: 1
```

EJERCICIOS

▶ 164 ¿Funciona el programa cuando introducimos una cadena formada sólo por espacios en blanco? ¿Por qué? Si su comportamiento no te parece normal, corrígelo.

El ejemplo que hemos desarrollado tiene un doble objetivo didáctico. Por una parte, familiarizarte con las cadenas; por otra, que veas cómo se resuelve un problema poco a poco. Primero hemos analizado el problema en busca de una solución sencilla (contar espacios en blanco). Después hemos implementado nuestra primera solución y la hemos probado con varios ejemplos. Los ejemplos que nos hemos puesto no son sólo los más sencillos, sino aquellos que pueden hacer «cascar» el programa (en nuestro caso, poner dos o más espacios en blanco seguidos). Detectar ese error nos ha conducido a una

«mejora» del programa (en realidad, una corrección): no debíamos contar espacios en blanco, sino transiciones de «no blanco» a espacio en blanco. Nuevamente hemos puesto a prueba el programa y hemos encontrado casos para los que falla (espacios al final de la cadena). Un nuevo refinamiento ha permitido tratar el fallo y, otra vez, hemos encontrado un caso no contemplado (espacios al principio de la cadena) que nos ha llevado a un último cambio del programa. Fíjate en que cada vez que hemos hecho un cambio al programa hemos vuelto a introducir todos los casos que ya habíamos probado (al modificar un programa es posible que deje de funcionar para casos en los que ya iba bien) y hemos añadido uno nuevo que hemos sospechado que podía ser problemático. Así es como se llega a la solución final: siguiendo un proceso reiterado de análisis, prueba y error. Durante ese proceso el programador debe «jugar» en dos «equipos» distintos:

- a ratos juega en el equipo de los programadores y trata de encontrar la mejor solución al problema propuesto;
- y a ratos juega en el equipo de los usuarios y pone todo su empeño en buscar configuraciones especiales de los datos de entrada que provoquen fallos en el programa.

EJERCICIOS

▶ 165 Modifica el programa para que base el cómputo de palabras en el número de transiciones de blanco a no blanco en lugar de en el número de transiciones de no blanco a blanco. Comprueba si tu programa funciona en toda circunstancia.

▶ 166 Nuestro aprendiz aventajado propone esta otra solución al problema de contar palabras:

```
cadena = raw_input('Escribe una frase: ')
while cadena != '':
    cambios = 0
    for i in range(1, len(cadena)):
        if cadena[i] == ' ' and cadena[i-1] != ' ':
            cambios = cambios + 1

    if cadena[-1] == ' ':
        cambios = cambios - 1

    palabras = cambios + 1
    print 'Palabras:', palabras

    cadena = raw_input('Escribe una frase: ')
```

¿Es correcta?

▶ 167 Diseña un programa que lea una cadena y un número entero k y nos diga cuántas palabras tienen una longitud de k caracteres.

▶ 168 Diseña un programa que lea una cadena y un número entero k y nos diga si *alguna* de sus palabras tiene una longitud de k caracteres.

▶ 169 Diseña un programa que lea una cadena y un número entero k y nos diga si *todas* sus palabras tienen una longitud de k caracteres.

▶ 170 Escribe un programa que lea una cadena y un número entero k y muestre el mensaje «`Hay palabras largas`» si alguna de las palabras de la cadena es de longitud mayor o igual que k, y «`No hay palabras largas`» en caso contrario.

▶ 171 Escribe un programa que lea una cadena y un número entero k y muestre el mensaje «`Todas son cortas`» si todas las palabras de la cadena son de longitud estrictamente menor que k, y «`Hay alguna palabra larga`» en caso contrario.

▶ 172 Escribe un programa que lea una cadena y un número entero k y muestre el mensaje «`Todas las palabras son largas`» si todas las palabras de la cadena son de longitud mayor o igual que k, y «`Hay alguna palabra corta`» en caso contrario.

▶ 173 Diseña un programa que muestre la cantidad de dígitos que aparecen en una cadena introducida por teclado. La cadena `'un␣1␣y␣un␣20'`, por ejemplo, tiene 3 dígitos: un 1, un 2 y un 0.

▶ 174 Diseña un programa que muestre la cantidad de números que aparecen en una cadena leída de teclado. ¡Ojo! Con número no queremos decir dígito, sino número propiamente dicho, es decir, secuencia de dígitos. La cadena `'un␣1,␣un␣201␣y␣2␣unos'`, por ejemplo, tiene 3 números: el 1, el 201 y el 2.

▶ 175 Diseña un programa que indique si una cadena leída de teclado está bien formada como número entero. El programa escribirá «`Es entero`» en caso afirmativo y «`No es entero`» en caso contrario.

Por ejemplo, para `'12'` mostrará «`Es entero`», pero para `'1␣2'` o `'a'` mostrará «`No es entero`».

▶ 176 Diseña un programa que indique si una cadena introducida por el usuario está bien formada como identificador de variable. Si lo está, mostrará el texto «`Identificador válido`» y si no, «`Identificador inválido`».

▶ 177 Diseña un programa que indique si una cadena leída por teclado está bien formada como número flotante.

Prueba el programa con estas cadenas: `'3.1'`, `'3.'`, `'.1'`, `'1e+5'`, `'-10.2E3'`, `'3.1e-2'`, `'.1e01'`. En todos los casos deberá indicar que se trata de números flotantes correctamente formados.

▶ 178 Un texto está bien parentizado si por cada paréntesis abierto hay otro más adelante que lo cierra. Por ejemplo, la cadena

```
'Esto␣(es␣(un)␣(ejemplo␣(de)␣((cadena)␣bien))␣parentizada).'
```

está bien parentizada, pero no lo están estas otras:

```
'una␣cadena)'    '(una␣cadena'    '(una␣(cadena)'    ')una(␣cadena'
```

Diseña un programa que lea una cadena y nos diga si la cadena está bien o mal parentizada.

▶ 179 Implementa un programa que lea de teclado una cadena que representa un número binario. Si algún carácter de la cadena es distinto de `'0'` o `'1'`, el programa advertirá al usuario de que la cadena introducida no representa un número binario y pedirá de nuevo la lectura de la cadena.