5.1.2. Escapes

Las cadenas que hemos estudiado hasta el momento consistían en sucesiones de caracteres «normales»: letras, dígitos, signos de puntuación, espacios en blanco... Es posible, no obstante, incluir ciertos caracteres especiales que no tienen una representación trivial.

Por ejemplo, los *saltos de línea* se muestran en pantalla como eso, saltos de línea, no como un carácter convencional. Si intentamos incluir un salto de línea en una cadena pulsando la tecla de retorno de carro, Python se queja:

```
>>> a = 'una ↲
  File "<string>", line 1
    'una
       ^
SyntaxError: invalid token
```

¿Ves? Al pulsar la tecla de retorno de carro, el intérprete de Python intenta ejecutar la sentencia inmediatamente y considera que la cadena está inacabada, así que notifica que ha detectado un error.

Observa esta otra asignación de una cadena a la variable *a* y mira qué ocurre cuando mostramos el contenido de *a*:

```
>>> a = 'una\ncadena' ↲
>>> print a ↲
una
cadena
```

Al mostrar la cadena se ha producido un salto de línea detrás de la palabra *una*. El salto de línea se ha codificado en la cadena con dos caracteres: la barra invertida \ y la letra *n*.

La barra invertida se denomina *carácter de escape* y es un carácter especial: indica que el siguiente carácter tiene un significado diferente del usual. Si el carácter que le sigue es la letra *n*, por ejemplo, se interpreta como un salto de línea (la *n* viene del término «new line», es decir, «nueva línea»). Ese par de caracteres forma una *secuencia de escape* y denota un único carácter. ¿Y un salto de línea es un único carácter? Sí. Ocupa el mismo espacio en memoria que cualquier otro carácter (un byte) y se codifica internamente con un valor numérico (código ASCII): el valor 10.

```
>>> ord('\n') ↲
10
```

Cuando una impresora o un terminal de pantalla tratan de representar el carácter de valor ASCII 10, saltan de línea. El carácter \n es un carácter *de control*, pues su función es permitirnos ejecutar una acción de control sobre ciertos dispositivos (como la impresora o el terminal).

Secuencia de escape para carácter de control	Resultado
\a	Carácter de «campana» (BEL)
\b	«Espacio atrás» (BS)
\f	Alimentación de formulario (FF)
\n	Salto de línea (LF)
\r	Retorno de carro (CR)
\t	Tabulador horizontal (TAB)
\v	Tabulador vertical (VT)
ooo	Carácter cuyo código ASCII en octal es *ooo*
\x*hh*	Carácter cuyo código ASCII en hexadecimal es *hh*

Tabla 5.1: Secuencias de escape para caracteres de control en cadenas Python.

Hay muchos caracteres de control (ver tabla 5.1), pero no te preocupes: nosotros utilizaremos fundamentalmente dos: \n y \t. Este último representa el carácter de tabulación horizontal o, simplemente, tabulador. El tabulador puede resultar útil para alinear en columnas datos mostrados por pantalla. Mira este ejemplo, en el que destacamos los espacios en blanco de la salida por pantalla para que puedas contarlos:

```
>>> print 'uno\tdos\ttres' ↲
uno␣␣␣␣␣dos␣␣␣␣␣tres
>>> print '1\t2\t3' ↲
1␣␣␣␣␣␣␣2␣␣␣␣␣␣␣3
>>> print '1\t12\t13\n21\t2\t33' ↲
1␣␣␣␣␣␣␣12␣␣␣␣␣␣13
21␣␣␣␣␣␣2␣␣␣␣␣␣␣33
```

Es como si hubiera unas marcas de alineación (los tabuladores) cada 8 columnas.

Alternativamente, puedes usar el código ASCII (en octal o hexadecimal) de un carácter de control para codificarlo en una cadena, como se muestra en las dos últimas filas de la tabla 5.1. El salto de línea tiene valor ASCII 10, que en octal se codifica con \012 y en hexadecimal con \x0a. Aquí te mostramos una cadena con tres saltos de línea codificados de diferente forma:

```
>>> print 'A\nB\012C\x0aD' ↲
A
B
C
D
```

Ciertos caracteres no se pueden representar directamente en una cadena. La barra invertida es uno de ellos. Para expresarla, debes usar dos barras invertidas seguidas.

```
>>> print 'a\\b' ↲
a\b
```

En una cadena delimitada con comillas simples no puedes usar una comilla simple: si Python trata de analizar una cadena mal formada como `'Munich'72'`, encuentra un error, pues cree que la cadena es `'Munich'` y no sabe cómo interpretar los caracteres `72'`. Una comilla simple en una cadena delimitada con comillas simples ha de ir precedida de la barra invertida. Lo mismo ocurre con la comilla doble en una cadena delimitada con comillas dobles (véase la tabla 5.2):

```
>>> print 'Munich\'72' ↲
Munich'72
>>> print "Una␣\"cosa\"␣rara." ↲
Una "cosa" rara.
```

Otras secuencias de escape	Resultado
\\	Carácter barra invertida (\)
\'	Comilla simple (')
\"	Comilla doble (")
\ y salto de línea	Se ignora (para expresar una cadena en varias líneas).

Tabla 5.2: Secuencias de escape para algunos caracteres especiales.

EJERCICIOS

▶ 152 ¿Qué se mostrará en pantalla al ejecutar estas sentencias?

```
>>> print '\\n' ↲
>>> print '\157\143\164\141\154' ↲
>>> print '\t\tuna\bo' ↲
```

(Te recomendamos que resuelvas este ejercicio a mano y compruebes la validez de tus respuestas con ayuda del ordenador.)

▶ 153 ¿Cómo crees que se pueden representar dos barras invertidas seguidas en una cadena?

▶ 154 La secuencia de escape `\a` emite un aviso sonoro (la «campana»). ¿Qué hace exactamente cuando se imprime en pantalla? Ejecuta **`print '\a'`** y lo averiguarás.

▶ 155 Averigua el código ASCII de los 10 primeros caracteres de la tabla 5.1.

Unix, Microsoft y Apple: condenados a no entenderse

Te hemos dicho que `\n` codifica el carácter de control «salto de línea». Es cierto, pero no es toda la verdad. En los antiquísimos sistemas de teletipo (básicamente, máquinas de escribir controladas por ordenador que se usaban antes de que existieran los monitores) se necesitaban dos caracteres para empezar a escribir al principio de la siguiente línea: un salto de línea (`\n`) y un retorno de carro (`\r`). Si sólo se enviaba el carácter `\n` el «carro» saltaba a la siguiente línea, sí, pero se quedaba en la misma columna. El carácter `\r` hacía que el carro retornase a la primera columna.

Con objeto de ahorrar memoria, los diseñadores de Unix decidieron que el final de línea en un fichero debería marcarse únicamente con `\n`. Al diseñar MS-DOS, Microsoft optó por utilizar dos caracteres: `\n\r`. Así pues, los ficheros de texto de Unix no son directamente compatibles con los de Microsoft. Si llevas un fichero de texto de un sistema Microsoft a Unix verás que cada línea acaba con un símbolo extraño (¡el retorno de carro!), y si llevas el fichero de Unix a un sistema Microsoft, parecerá que las líneas están mal alineadas.

Para poner peor las cosas, nos falta hablar de la decisión que adoptó Apple en los ordenadores Macintosh: usar sólo el retorno de carro (`\r`). ¡Tres sistemas operativos y tres formas distintas de decir lo mismo!

De todos modos, no te preocupes en exceso, editores de texto como XEmacs y PythonG son bastante «listos»: suelen detectar estas situaciones y las corrigen automáticamente.

Más sobre la codificación de las cadenas

Hemos visto que podemos codificar cadenas encerrando un texto entre comillas simples o entre comillas dobles. En tal caso, necesitamos usar secuencias de escape para acceder a ciertos caracteres. Python ofrece aún más posibilidades para codificar cadenas. Una de ellas hace que no se interpreten las secuencias de escape, es decir, que todos sus caracteres se interpreten literalmente. Estas cadenas «directas» (en inglés, «raw strings») preceden con la letra «r» a las comillas (simples o dobles) que la inician:

```
>>> print r'u\n' ↲
u\n
>>> print r"u\\n" ↲
u\\n
```

Cuando una cadena ocupa varias líneas, podemos usar la secuencia de escape `\n` para marcar cada salto de línea. O podemos usar una «cadena multilínea». Las cadenas multilínea empiezan con tres comillas simples (o dobles) y finalizan con tres comillas simples (o dobles):

```
>>> print '''Una ↲
... cadena ↲
... que ocupa ↲
... varias líneas''' ↲
Una
cadena
que ocupa
varias líneas
```