Cadenas de Caracteres

Tomado del Capítulo 6 de
"Aprendiendo a Programar usando Python como Herramienta",
de Wachenchauzer et al.

Una cadena es una secuencia de caracteres. Ya las hemos usado para mostrar mensajes, pero sus usos son mucho más amplios que sólo ése: los textos que manipulamos mediante los editores de texto, los textos de Internet que analizan los buscadores, los mensajes enviados mediante correo electrónico, son todos ejemplos de cadenas de caracteres. Pero para poder programar este tipo de aplicaciones debemos aprender a manipularlas. Comenzaremos a ver ahora cómo hacer cálculos con cadenas.

Operaciones con cadenas

- Sumar cadenas entre sí (y el resultado es la concatenación de todas las cadenas dadas):

```
>>> "Un divertido " + "programa " + "de " + "radio"
'Un divertido programa de radio'
```

- Multiplicar una cadena s por un número k (y el resultado es la concatenación de s consigo misma, k veces):

```
>>> 3 * "programas "
'programas programas programas '
>>> "programas " * 3
'programas programas programas '
```

- Se puede averiguar la longitud de una cadena utilizando la función len(). La cadena vacía no contiene ningún carácter, y por lo tanto, su longitud es cero.

```
>>> len("programas ")
10
>>> s = ""
>>> s
''
>>> len(s)
0
```

- Se puede recorrer todos los caracteres de una cadena de manera muy sencilla, usando directamente un ciclo definido:

```
>>> for c in "programas ":
…       print(c)
```

- Preguntar si una cadena contiene una subcadena, usando el operador in. **a in b** es una expresión que se evalúa True si la cadena **b** contiene la subcadena **a**.

```
>>> 'qué' in 'Hola, ¿qué tal?'
True
>>> '7' in '2468'
False
```

- Acceder a una posición de la cadena, con la notación con corchetes: escribiremos s[i] para hablar de la posición i-ésima de la cadena s.
- Podemos identificar segmentos de una cadena. La notación es similar a la vista para listas y tuplas: s[0:2] se refiere a la subcadena formada por los caracteres cuyos índices están en el

rango [0,2).

- Las cadenas son inmutables: si bien podemos acceder a una posición de la cadena, no podemos modificarla por asignación. La cadena no soporta la modificación de un carácter, si queremos corregir la ortografía de una cadena, debemos asignarle una nueva cadena.
- A partir de una cadena de caracteres, podemos obtener una lista con sus componentes usando la función split. Si queremos obtener las palabras (separadas entre sí por espacios) que componen la cadena padrones escribiremos simplemente padrones.split() :

```
>>> c = "Una cadena con espacios  "
>>> c.split()
['Una', 'cadena', 'con', 'espacios']
```

  En este caso split elimina todos los blancos de más, y devuelve sólo las palabras que conforman la cadena.

  Si en cambio el separador es otro carácter (por ejemplo la arroba, "@"), se lo debemos pasar como parámetro a la función split. En ese caso se considera una componente todo lo que se encuentra entre dos arrobas consecutivas. En el caso particular de que el texto contenga dos arrobas una a continuación de la otra, se devolverá una componente vacía:

```
>>> d="@@Una@@@cadena@@@con@@arrobas@"
>>> d.split("@")
['', '', 'Una', '', '', 'cadena', '', '', 'con', '', 'arrobas', '']
```

- La “casi”–inversa de split es una función join que tiene la siguiente sintaxis:

  <separador >.join(<lista de componentes a unir>)

  y que devuelve la cadena que resulta de unir todas las componentes separadas entre sí por medio del separador:

```
>>> xs = ['aaa', 'bbb', 'cccc']
>>> " ".join(xs)
'aaa bbb cccc'
>>> ", ".join(xs)
'aaa, bbb, cccc'
>>> "@@".join(xs)
'aaa@@bbb@@cccc'
```

Formato de cadenas

Muchas veces es necesario darle un formato determinado a las cadenas, o dicho de otro modo, procesar los datos de entrada para que las cadenas resultantes se vean de una manera en particular.

Por ejemplo, para incluir un número en medio de un mensaje, podemos usar la siguiente cadena de formato:

```
>>> no_leidos = 8
>>> "Hay {} mensajes sin leer".format(no_leidos)
'Hay 8 mensajes sin leer'
```

Equivalente a:

```
>>> "Hay " + str(no_leidos) + " mensajes sin leer"
'Hay 8 mensajes sin leer'
```

La función format devuelve una cadena en donde se reemplazan todas las marcas {} por los valores

indicados, que automáticamente se convierten a cadenas de texto. Al usar la cadenas de formato de esta manera, podemos ver claramente cuál es el contenido del mensaje, evitando errores con respecto a espacios de más o de menos, interrupciones en el texto que complican su lectura, etc.

Otro ejemplo, utilizando varios valores, en este caso cadenas:

```
>>> "Artista: {}. Disco: {}.".format(artista , disco)
```

Equivalente a:

```
>>> "Artista: " + artista + ". Disco: " + disco + "."
```

En el caso de los valores numéricos, es posible modificar la forma en la que el número es presentado. Por ejemplo, si se trata de un monto monetario, usualmente queremos mostrarlo con dos dígitos decimales, para ello utilizaremos la marca {:.2} para indicar dos dígitos luego del separador decimal.

```
>>> suma = 205.5
>>> 'Sin IVA: {:.2}. Con IVA: {:.2}'.format(suma, suma * 1.21)
'Sin IVA: 205.50. Con IVA: 248.66'
```

En otras situaciones, como el caso de un valor en un estudio médico, podemos querer mostrar el número en notación científica. En este caso utilizaremos la marca {:.1e} , indicando que queremos un dígito significativo luego del separador decimal.

```
>>> rojos = 4640000
>>> 'Glóbulos rojos: {:.1e}/uL'
'Glóbulos rojos: 4.6e+06/uL'
```

Las marcas de formato se indican con {}, y pueden incluir diversos parámetros de conversión.

A continuación se lista una referencia de algunos tipos de conversión utilizados frecuentemente:

Tipo	Significado	Ejemplo
{:d}	Valor entero en decimal	'{:d}'.format(10.5) → '10'
{:o}	Valor entero en octal	'{:o}'.format(8) → '10'
{:x}	Valor entero en hexadecimal	'{:x}'.format(16) → '10'
{:f}	Valor de punto flotante, en decimal	'{:f}'.format(0.1**5) → '0.000010'
{:.2f}	Punto flotante, con dos dígitos de precisión	'{:.2f}'.format(0.1**5) → '0.00'
{:e}	Punto flotante, en notación exponencial	'{:e}'.format(0.1**5) → '1.000000e-05'
{:g}	Punto flotante, lo que sea más corto	'{:g}'.format(0.1**5) → '0.00001'
{:.2s}	Cadena recortada en 2 caracteres	'{:.2s}'.format('Python') → 'Py'
{:<6s}	Cadena alineada a la izquierda, ocupando 6 caracteres	'{:<6s}'.format('Py') → 'Py '
{:^6s}	Cadena centrada	'{:^6s}'.format('Py') → ' Py '
{:>6s}	Cadena alineada a la derecha	'{:>6s}'.format('Py') → ' Py'
{{	Un {	
}}	Un }	