CADENAS DE TEXTO

No cabe duda de que, a parte de los números, las cadenas de texto (*strings*) son otro de los tipos de datos más utilizados en programación. El intérprete de Python integra este tipo de datos, además de una extensa serie de funciones para interactuar con diferentes cadenas de texto.

En algunos lenguajes de programación, como en C, las cadenas de texto no son un tipo integrado como tal en el lenguaje. Esto implica un poco de trabajo extra a la hora de realizar operaciones como la concatenación. Sin embargo, esto no ocurre en Python, lo que hace mucho más sencillo definir y operar con este tipo de dato.

Básicamente, una cadena de texto o *string* es un conjunto inmutable y ordenado de caracteres. Para su representación y definición se pueden utilizar tanto comillas dobles ("), como simples ('). Por ejemplo, en Python, la siguiente sentencia crearía una nueva variable de tipo *string*:


```
>>> cadena = "esto es una cadena de texto"
```

Si necesitamos declarar un string que contenga más de una línea, podemos hacerlo utilizando comillas triples en lugar de dobles o simples:

```
>>> cad_multiple = """Esta cadena de texto
... tiene más de una línea. En concreto, cuenta

... con tres líneas diferentes"""
```

Principales funciones y métodos

Para trabajar con strings Python pone a nuestra disposición una serie de funciones y métodos. Las primeras pueden ser invocadas directamente y reciben como argumento una cadena. Por otro lado, una vez que tenemos declarado el string, podemos invocar a diferentes métodos con los que cuenta este tipo de dato.

Una de las funciones más comunes que podemos utilizar sobre strings es el cálculo del número de caracteres que contiene. El siguiente ejemplo nos muestra cómo hacerlo:

```
>>> cad = "Cadena de texto de ejemplo"
>>> len(cad)
26
```

Otro ejemplo de función que puede ser invocada, sin necesidad de declarar una variable de tipo string, es *print()*. En el capítulo anterior mostramos cómo emplearla para imprimir una cadena de texto por la salida estándar.

Respecto a los métodos con los que cuentan los objetos de tipo string, Python incorpora varios de ellos para poder llevar a cabo funcionalidades básicas relacionadas con cadenas de texto. Entre ellas, contamos con métodos para buscar una subcadena dentro de otra, para reemplazar subcadenas, para borrar espacios en blanco, para pasar de mayúsculas a minúsculas, y viceversa.

La función *find()* devuelve el índice correspondiente al primer carácter de la cadena original que coincide con el buscado:

```
>>> cad ="xyza"
>>> cad.find("y")
1
```

Si el carácter buscado no existe en la cadena, *find()* devolverá -1.

Para reemplazar una serie de caracteres por otros, contamos con el método *replace()*. En el siguiente ejemplo, sustituiremos la subcadena "Hola" por "Adiós":

```
>>> cad = "Hola Mundo"
>>> cad.replace("Hola", "Adiós")
'Adiós Mundo'
```

Obsérvese que *replace()* no altera el valor de la variable sobre el que se ejecuta. Así pues, en nuestro ejemplo, el valor de la variable *cad* seguirá siendo *"Hola Mundo"*.

Los métodos *strip()*, *lstrip()* y *rstrip()* nos ayudarán a eliminar todos los espacios en blanco, solo los que aparecen a la izquierda y solo los que se encuentran a la derecha, respectivamente:

```
>>> cad = "  cadena con espacios en blanco "
>>> cad.strip()
"cadena con espacios en blanco"
>>> cad.lstrip()
"cadena con espacios en blanco "
>>> cad.rstrip()
"  cadena con espacios en blanco"
```

El método *upper()* convierte todos los caracteres de una cadena de texto a mayúsculas, mientras que *lower()* lo hace a minúsculas. Veamos un sencillo ejemplo:

```
>>> cad2 = cad.upper()
>>> print(cad2)
"CADENA CON ESPACIOS EN BLANCO"
>>> print(cad3.lower())
"  cadena con espacios en blanco "
```

Relacionados con *upper()* y *lower()* encontramos otro método llamado *capitalize()*, el cual solo convierte el primer carácter de un string a mayúsculas:

```
>>> cad = "un ejemplo"
>>> cad.capitalize()
'Un ejemplo'
```

En ocasiones puede ser muy útil dividir una cadena de texto basándonos en un carácter que aparece repetidamente en ella. Esta funcionalidad es la que nos ofrece *split()*. Supongamos que tenemos un cadena con varios valores separadas por *;* y que necesitamos una lista donde cada valor se corresponda con los que aparecen delimitados por el mencionado carácter. El siguiente ejemplo nos muestra cómo hacerlo:

```
>>> cad = "primer valor;segundo;tercer valor"
>>> cad.split(";")
['primer valor', 'segundo', 'tercer valor']
```

join() es un método que devuelve una cadena de texto donde los valores de la cadena original que llama al método aparecen separados por un carácter pasado como argumento:

```
>>> "abc".join(',')

'a,b,c'
```

Operaciones

El operador + nos permite concatenar dos strings, el resultado puede ser almacenado en una nueva variable:

```
>>> cad_concat = "Hola" + " Mundo!"
>>> print(cad_concat)
Hola Mundo!
```

También es posible concatenar una variable de tipo string con una cadena o concatenar directamente dos variables. Sin embargo, también podemos prescindir del operador + para concatenar strings. A veces, la expresión es más fácil de leer si empleamos el método *format()*. Este método admite emplear, dentro de la cadena de texto, los caracteres *{}*, entre los que irá, número o el nombre de una variable. Como argumentos del método pueden pasarse variables que serán sustituidas, en tiempo de ejecución, por los marcadores *{}*, indicados en la cadena de texto. Por ejemplo, veamos cómo las siguientes expresiones son equivalentes y devuelven el mismo resultado:

```
>>> "Hola " + cad2 + ". Otra " + cad3
>>> "Hola {0}. Otra {1}".format(cad2, cad3)
>>> "Hola {cad2}. Otra {cad3}".format(cad2=cad2,
cad3=cad3)
```

La concatenación entre strings y números también es posible, siendo para ello necesario el uso de funciones como *int()* y *str()*. El siguiente ejemplo es un caso sencillo de cómo utilizar la función *str()*:

```
>>> num = 3
>>> "Número: " + str(num)
```

Interesante resulta el uso del operador * aplicado a cadenas de texto, ya que nos permite repetir un string *n* veces. Supongamos que deseamos repetir la cadena "Hola Mundo" cuatro veces. Para ello, bastará con lanzar la siguiente sentencia:

```
>>> print("Hola Mundo" * 4)
```

Gracias al operador *in* podemos averiguar si un determinado carácter se encuentra o no en una cadena de texto. Al aplicar el operador, como resultado, obtendremos *True* o *False*, en función de si el valor se encuentra o no en la cadena. Comprobémoslo en el siguiente ejemplo:

```
>>> cad = "Nueva cadena de texto"
>>> "x" in cad
False
```

Un string es inmutable en Python, pero podemos acceder, a través de índices, a cada carácter que forma parte de la cadena:

```
>>> cad = "Cadenas"
>>> print(cad[2])
d
```

Los comentados índices también nos pueden ayudar a obtener subcadenas de texto basadas en la original. Utilizando la variable *cad* del ejemplo anterior podemos imprimir solo los tres primeros caracteres:

```
>>> print(cad[:3])
Cad
```

En el ejemplo anterior el operador *:* nos ha ayudado a nuestro propósito. Dado que delante del operador no hemos puesto ningún número, estamos indicando que vamos a utilizar el primer carácter de la cadena. Detrás del operador añadimos el número del índice de la cadena de texto que será utilizado como último valor. Así pues, obtendremos los tres primeros caracteres. Los índices negativos también funcionan, simplemente indican que se empieza contar desde el último carácter. La siguiente sentencia devolverá el valor *a*:

```
>>> cad[-2]
```

A continuación, veamos un ejemplo donde utilizamos un número después del mencionado operador:

```
>>> cad[3:]

'enas'
```