Cadenas Python

Las cadenas en Python o `strings` son un tipo inmutable que permite almacenar secuencias de caracteres. Para crear una, es necesario incluir el texto entre comillas dobles `"`. Puedes obtener más ayuda con el comando `help(str).`

```
s = "Esto es una cadena"
print(s)       #Esto es una cadena
print(type(s)) #<class 'str'>
```

También es valido declarar las cadenas con comillas simples simples `'`.

```
s = 'Esto es otra cadena'
print(s)        #Esto es otra cadena
print(type(s))  #<class 'str'>
```

Las cadenas no están limitadas en tamaño, por lo que el único límite es la memoria de tu ordenador. Una cadena puede estar también vacía.

```
s = ''
```

Una situación que muchas veces se puede dar, es cuando queremos introducir una comilla, bien sea simple `'` o doble `"` dentro de una cadena. Si lo hacemos de la siguiente forma tendríamos un error, ya que Python no sabe muy bien donde empieza y termina.

```
#s = "Esto es una comilla doble " de ejemplo" # Error!
```

Para resolver este problema debemos recurrir a las secuencias de escape. En Python hay varias, pero las analizaremos con más detalle en otro capítulo. Por ahora, la más importante es `\"`, que nos permite incrustar comillas dentro de una cadena.

```
s = "Esto es una comilla doble \" de ejemplo"
print(s) #Esto es una comilla doble " de ejemplo
```

También podemos incluir un salto de línea dentro de una cadena, lo que significa que lo que esté después del salto, se imprimirá en una nueva línea.

```
s = "Primer linea\nSegunda linea"
print(s)
#Primer linea
#Segunda linea
```

También podemos usar `\` acompañado de un número, lo que imprimirá el carácter asociado. En este caso imprimimos el carácter `110` que se corresponde con la `H`.

```
print("\110\110") #HH
```

Para saber más: Te recomendamos que busques información sobre ASCI y Unicode. Ambos son conceptos muy útiles a la hora de entender los strings.

Se puede definir una cadena que ocupe varias líneas usando triple `"""` comilla. Puede ser muy útil si tenemos textos muy largo que no queremos tener en una sola línea.

Existe también otra forma de declarar cadenas llamado `raw strings`. Usando como prefijo `r`, la cadena ignora todos las secuencias de escape, por lo que la salida es diferente a la anterior.

```
print(r"\110\110") #\110\110

print("""La siguiente
cadena ocupa
varias lineas""")
```

Formateo de cadenas

Tal vez queramos declarar una cadena que contenga variables en su interior, como números o incluso otras cadenas. Una forma de hacerlo sería concatenando la cadena que queremos con otra usando el operador `+`. Nótese que `str()` convierte en `string` lo que se pasa como parámetro.

```
x = 5
s = "El número es: " + str(x)
print(s) #El número es: 5
```

Otra forma es usando `%`. Por un lado tenemos `%s` que indica el tipo que se quiere imprimir, y por otro a la derecha del `%` tenemos la variable a imprimir. Para imprimir una cadena se usaría `%s` o `%f` para un valor en coma flotante.

Para saber más: En el siguiente enlace puedes encontrar más información sobre el uso de %.

```
x = 5
s = "El número es: %d" % x
print(s) #El número es: 5
```

Si tenemos más de una variable, también se puede hacer pasando los parámetros dentro de `()`. Si vienes de lenguajes como `C`, esta forma te resultará muy familiar. No obstante, esta no es la forma preferida de hacerlo, ahora que tenemos nuevas versiones de Python.

```
s = "Los números son %d y %d." % (5, 10)
print(s) #Los números son 5 y 10.
```

Una forma un poco más moderna de realizar lo mismo, es haciendo uso de `format()`.

```
s = "Los números son {} y {}".format(5, 10)
print(s) #Los números son {} y {}".format(5, 10)
```

Es posible también darle nombre a cada elemento, y `format()` se encargará de reemplazar todo.

```
s = "Los números son {a} y {b}".format(a=5, b=10)
print(s) #Los números son 5 y 10
```

Por si no fueran pocas ya, existe una tercera forma de hacerlo introducida en la versión `3.6` de Python. Reciben el nombre de cadenas literales o `f-strings`. Esta nueva característica, permite incrustar expresiones dentro de cadenas.

```
a = 5; b = 10
s = f"Los números son {a} y {b}"
print(s) #Los números son 5 y 10
```

Puedes incluso hacer operaciones dentro de la creación del `string`.

```
a = 5; b = 10
s = f"a + b = {a+b}"
print(s) #a + b = 15
```

Puedes incluso llamar a una función dentro.

```
def funcion():
    return 20
s = f"El resultado de la función es {funcion()}"
print(s) #El resultado de la funcion es 20
```

Ejemplos `string`

Para entender mejor la clase `string`, vamos a ver unos ejemplos de como se comportan. Podemos sumar dos strings con el operador `+`.

```
s1 = "Parte 1"
s2 = "Parte 2"
print(s1 + " " + s2) #Parte 1 Parte 2
```

Se puede multiplicar un `string` por un `int`. Su resultado es replicarlo tantas veces como el valor del entero.

```
s = "Hola "
print(s*3) #Hola Hola Hola
```

Podemos ver si una cadena esta contenida en otra con `in`.

```
print("mola" in "Python mola") #True
```

Con `chr()` and `ord()` podemos convertir entre carácter y su valor numérico que lo representa y viceversa. El segundo sólo función con caracteres, es decir, un `string` con un solo elemento.

```
print(chr(8364)) #€
print(ord("€"))  #110
```

La longitud de una cadena viene determinada por su número de caracteres, y se puede consultar con la función `len()`.

```
print(len("Esta es mi cadena"))
```

Como hemos visto al principio, se puede convertir a `string` otras clases, como `int` o `float`.

```
x = str(10.4)
print(x)       #10.4
print(type(x)) #<class 'str'>
```

También se pueden indexar las cadenas, como si de una lista se tratase.

```
x = "abcde"
print(x[0])  #a
print(x[-1]) #e
```

Del mismo modo, se pueden crear cadenas más pequeñas partiendo de una grande, usando indicando el primer elemento y el último que queremos tomar menos uno.

```
x = "abcde"
print(x[0:2])
```

Si no se indica ningún valor a la derecha de los `:` se llega hasta el final.

```
x = "abcde"
print(x[2:])
```

Es posible también crear subcadenas que contengan elementos salteados y no contiguos añadiendo un tercer elemento entre `[]`. Indica los elementos que se saltan. En el siguiente ejemplo se toman elementos del `0` al `5` de dos en dos.

```
x = "abcde"
print(x[0:5:2]) #ace
```

Tampoco es necesario saber el tamaño de la cadena, y el segundo valor se podría omitir. El siguiente ejemplo es igual al anterior.

```
x = "abcde"
print(x[0::2]) #ace
```

Métodos `string`

Algunos de los métodos de la clase `string`.

`capitalize()`

El método `capitalize()` se aplica sobre una cadena y la devuelve con su primera letra en mayúscula.

```
s = "mi cadena"
print(s.capitalize()) #Mi cadena
```

`lower()`

El método `lower()` convierte todos los caracteres alfabéticos en minúscula.

```
s = "MI CADENA"
print(s.lower()) #mi cadena
```

`swapcase()`

El método `swapcase()` convierte los caracteres alfabéticos con mayúsculas en minúsculas y viceversa.

```
s = "mI cAdEnA"
print(s.swapcase()) #Mi CaDeNa
```

`upper()`

El método `upper()` convierte todos los caracteres alfabéticos en mayúsculas.

```
s = "mi cadena"
print(s.upper())
```

`count(<sub>[, <start>[, <end>]])`

El método `count()` permite contar las veces que otra cadena se encuentra dentro de la primera. Permite también dos parámetros opcionales que indican donde empezar y acabar de buscar.

```
s = "el bello cuello "
print(s.count("llo")) #2
```

isalnum()

El método `isalnum()` devuelve `True` si la cadena esta formada únicamente por caracteres alfanuméricos, `False` de lo contrario. Caracteres como @ o & no son alfanumericos.

```
s = "correo@dominio.com"
print(s.isalnum())
```

isalpha()

El método `isalpha()` devuelve `True` si todos los caracteres son alfabéticos, `False` de lo contrario.

```
s = "abcdefg"
print(s.isalpha())
```

strip([<chars>])

El método `strip()` elimina a la izquierda y derecha el carácter que se le introduce. Si se llama sin parámetros elimina los espacios. Muy útil para limpiar cadenas.

```
s = "  abc  "
print(s.strip()) #abc
```

zfill(<width>)

El método `zfill()` rellena la cadena con ceros a la izquierda hasta llegar a la longitud pasada como parámetro.

```
s = "123"
print(s.zfill(5)) #00123
```

join(<iterable>)

El método `join()` devuelve la primera cadena unida a cada uno de los elementos de la lista que se le pasa como parámetro.

```
s = " y ".join(["1", "2", "3"])
print(s) #1 y 2 y 3
```

split(sep=None, maxsplit=-1)

El método `split()` divide una cadena en subcadenas y las devuelve almacenadas en una lista. La división es realizada de acuerdo a el primer parámetro, y el segundo parámetro indica el número máximo de divisiones a realizar.

```
s = "Python,Java,C"
print(s.split(",")) #['Python', 'Java', 'C']
```