Strings en `Python`

Variable string

String. Cadena ordenada de caracteres.

Una variable de tipo string es aquella que guarda un string. Cuando queremos que una variable se trate de una variable de tipo string, `str` en `Python`, a la hora de declararla, el contenido de la variable debe ir o bien entre comillas dobles `" "`, o bien entre comillas simples `' '`.

```
s1 = "Esto es un string entre comillas dobles"
type(s1)
```

```
str
```

```
s2 = 'Esto es un string entre comillas simples'
type(s2)
```

```
str
```

¡Cuidado! `Python` no trabaja bien con los acentos. Por tanto, aunque no esté del todo bien escrito, mejor evitarlos, pues nos ahorraremos muchos problemas.

String literals

El hecho de que el contenido de las variables de tipo `str` vaya entre comillas, ya sean simples o dobles, conlleva a que algunos caracteres deban ser tratados de forma especial.

Aquí entran en juego los string literals. Algunos de los más utilizados se muestran en la siguiente tabla:

Código	Significado
`\\`	Backslash, `\`
`\'`	Comilla simple, `'`
`\"`	Comilla doble, `"`
`\n`	Salto de línea
`\t`	Tabulación horizontal

Para más información acerca de los string literals ir a la documentación.

Ejemplo 1

Si queremos guardar en una variable el siguiente texto,

`Juan dijo: "me gusta el chocolate"`

lo tendremos que hacer del siguiente modo

```
s1 = "Juan dijo: \"me gusta el chocolate \""
s1
```

```
'Juan dijo: "me gusta el chocolate "'
```

```
s2 = 'Juan dijo: "me gusta el chocolate "'
s2
```

```
'Juan dijo: "me gusta el chocolate "'
```

Observación. Si usamos comillas dobles, para guardar la frase de este ejemplo necesitaremos usar string literals, ya que si no nos saltará error. Sin embargo, si usamos comillas simples, para guardar la frase de este ejemplo en una variable no hace falta que cambiemos nada.

Ejemplo 2

Si queremos guardar en una variable el siguiente texto,

`Ricardo dijo: 'me gusta la playa'`

lo tendremos que hacer del siguiente modo

```
s1 = "Ricardo dijo: 'me gusta la playa'"
s1
```

```
'Ricardo dijo: 'me gusta la playa''
```

```
s2 = 'Ricardo dijo: \'me gusta la playa\''
s2
```

```
'Ricardo dijo: 'me gusta la playa''
```

Observación. Si usamos comillas dobles, para guardar la frase de este ejemplo no necesitaremos usar string literals. Sin embargo, si usamos comillas simples, para guardar la frase de este ejemplo en una variable tendremos que usar string literals, porque si no nos saltará un error.

Ejemplo 3

Si queremos guardar en una variable el siguiente texto y que se conserve el salto de línea,

`Con diez cañones por banda,`

`viento en popa a toda vela`

lo tendremos que hacer del siguiente modo

```
s3 = "Con diez cañones por banda,\nviento en popa a toda vela"
s3
```

```
'Con diez cañones por banda,\nviento en popa a toda vela'
```

Observación. El resultado con el salto de línea aplicado lo veremos cuando hablemos de la función `print()`, cosa que haremos más adelante en esta sección.

Concatenación de strings

La concatenación es una operación que une dos o más strings en uno solo.

En `Python`, para concatenar dos variables de tipo string usamos la función `+`.

```
s1 = "Hola, "
s2 = "Juan"
s1 + s2
```

```
'Hola, Juan'
```

Observación. La concatenación viene a ser como pegar el final del primer string con el princio del segundo. Entonces, conviene poner un espacio al final de la primera variable a concatenar, o bien al principio de la segunda para que así, al realizar la concatenación, exista ese espacio entre las palabras.

De no añadir espacios adicionales, obtendríamos resultados como el mostrado en el siguiente chunk:

```
s1 = "Bienvenido"
s2 = "al curso."
s1 + s2
```

```
'Bienvenidoal curso.'
```

Si dejamos un espacio adicional al final del string `s1`, obtenemos

```
s1 = "Bienvenido "
s2 = "al curso."
s1 + s2
```

```
'Bienvenido al curso.'
```

Si dejamos un espacio adicional al principio del string `s2` obtenemos

```
s1 = "Bienvenido"
s2 = " al curso."
s1 + s2
```

```
'Bienvenido al curso.'
```

Si dejamos un espacio adicional tanto al final del string `s1` como al principio del string `s2` obtenemos

```
s1 = "Bienvenido "
s2 = " al curso."
s1 + s2
```

```
'Bienvenido  al curso.'
```

Si dejamos más de un espacio adicional ya sea al final del string `s1` como al principio del string `s2` obtenemos

```
s1 = "Bienvenido   " # Se han dejado 3 espacios adicionales
s2 = "al curso."
s1 + s2
```

```
'Bienvenido   al curso.'
```

Observación. El número de espacios añadidos se conserva. Más adelante en esta sección veremos como eliminar los posibles espacios en blanco sobrantes.

Repetición de strings

La repetición es una operación que repite la variable string tantas veces como indiquemos.

En `Python`, para repetir una variable de tipo string usamos la función `*`. El orden de los factores no altera el producto. Es decir, tanto da usar la sintaxis `num_repeticiones * variable_str` como `variable_str * num__repeticiones`.

```
s1 = "¿Falta mucho? "
```

```
s1 = "¿Falta mucho?
s1 * 5
```

```
'¿Falta mucho? ¿Falta mucho? ¿Falta mucho? ¿Falta mucho? ¿Falta mucho? '
```

```
s2 = " ¿Hemos llegado ya?"
5 * s2
```

```
' ¿Hemos llegado ya? ¿Hemos llegado ya? ¿Hemos llegado ya? ¿Hemos llegado ya? ¿He
mos llegado ya?'
```

Observación. Al igual que ocurría con la concatenación, hay que añadir manualmente uno o más espacios en blanco al principio o al final del string para que las repeticiones no estén pegadas las unas a las otras, tal y como ocurre en el siguiente chunk de código.

```
s3 = "Había una vez un barquito chiquitito"
s3 * 2
```

```
'Había una vez un barquito chiquititoHabía una vez un barquito chiquitito'
```

La función `print()`

Hasta ahora, cada vez que mostrábamos strings por pantalla, estos salían entre comillas simples.

La función `print()` nos sirve, entre otras muchas cosas, para mostrar strings por pantalla.

```
s = "Hello world"
s
```

```
'Hello world'
```

```
print(s)
```

```
Hello world
```

Observación. Como véis, una de las principales diferencias entre usar la función `print()` o no usarla es que a la hora de mostrar la cadena de caracteres por pantalla, ésta no va entre comillas simples y el formato en que se imprime también es diferente.

No solamente podemos imprimir strings, sino que podemos mostrar el resultado de cualquier variable (numérica o de tipo string)

```
x = "Vivo en una isla"
print(x)
```

```
Vivo en una isla
```

```
vivo en una isla
```

```
y = 2.0
print(y)
```

```
2.0
```

Al igual que podíamos concatenar strings con la función `+`, combinando ésta junto con la función `print()` podemos concatenar strings con variables que almacenan strings

```
name = "Don Pepito"
print("¡Buenos días, " + name + "!")
```

```
¡Buenos días, Don Pepito!
```

Observación. Recordad introducir un espacio adicional siempre que vayáis a concatenar cualquier cosa (strings con strings, strings con variables...), para que así el resultado quede legible.

Observación. Utilizando la función `print()`, el uso de acentos o de algunos caracteres especiales como `¿` o `¡` ya no dan problemas a la hora de mostrarse por pantalla.

Observación. Podemos obtener exactamente el mismo resultado utilizando comas (`,`) en vez de la función `+`. Eso sí, después de cada coma se nos añade automáticamente un espacio en blanco que no siempre buscamos, como ocurre a continuación después del resultado de la variable `name`.

```
name = "Don Pepito"
print("¡Buenos días,", name, "!")
```

```
¡Buenos días, Don Pepito !
```

Al igual que podíamos repetir un mismo string un número cualquiera de veces con la función `*`, combinando ésta junto con la función `print()` podemos multiplicar un string o variables que contengan strings

```
print("¿Falta mucho? " * 5)
```

```
¿Falta mucho? ¿Falta mucho? ¿Falta mucho? ¿Falta mucho? ¿Falta mucho?
```

```
pregunta = "¿Falta mucho? "
print(pregunta * 5)
```

```
¿Falta mucho? ¿Falta mucho? ¿Falta mucho? ¿Falta mucho? ¿Falta mucho?
```

La función `str()`

La función `str()`

Con la función `str()`, podemos concatenar strings y variables de cualquier tipo dentro de un `print()`:

```
nombre = "María"
edad = 22
print("Mi hermana se llama " + nombre + " y su edad es " + str(edad))
```

```
Mi hermana se llama María y su edad es 22
```

El método `.format()`

Existe otra forma de concatenar strings y variables de cualquier tipo dentro de un `print()` y es gracias al método `.format()`. Lo que hay que hacer es indicar con llaves, `{}`, donde queremos situar el resultado de las variables y luego, dentro de los paréntesis del método `.format()`, indicar las variables en su respectivo orden

```
nombre = "Ricardo"
numero_gatos = 3
print("Mi abuelo se llama {} y tiene {} gatos".format(nombre, numero_gatos))
```

```
Mi abuelo se llama Ricardo y tiene 3 gatos
```

Saltos de línea y tabulaciones

Si recordáis el ejemplo 3, teníamos la variable `s3`, que contenía un salto de línea

```
s3 = "Con diez cañones por banda,\nviento en popa a toda vela"
```

Con la función `print()`, seremos capaces de visualizar dicho salto de línea

```
print(s3)
```

```
Con diez cañones por banda,
viento en popa a toda vela
```

Y lo mismo ocurriría con la tabulación horizontal.

```
s4 = "La string literal \\t producía \t una tabulación horizontal"
s4
```

```
'La string literal \\t producía \t una tabulación horizontal'
```

```
print(s4)
```

```
print(s4)
```

```
La string literal \t producía    una tabulación horizontal
```

Substrings

Para acceder a un caracter de una variable string usamos la sintaxis de `[]`

```
s = "Soy fan de los videojuegos"
```

```
s[0] # Primer caracter
```

```
'S'
```

```
s[5] # Sexto caracter
```

```
'a'
```

¡Cuidado! En `Python`, los índices siempre empiezan en 0, al contrario de lo que ocurre con otros lenguajes de programación, como por ejemplo `R`.

Si precedemos el índice por un `-`, entonces empezamos desde el final

```
s[-1] # Último elemento
```

```
's'
```

```
s[-7] # Séptimo elemento empezando por el final
```

```
'o'
```

Si queremos acceder a varios caracteres seguidos, podemos utilizar la función `:`

```
s[4:7] # Del quinto al séptimo
```

```
'fan'
```

```
s[:7] # Del primero al séptimo
```

```
'Soy fan'
```

```
s[8:] # Del noveno al final
```

```
'de los videojuegos'
```

¡Cuidado! En `Python`, siempre que usemos la función `:`, el índice que se encuentra a la derecha nunca es incluido, tal y como hemos visto en los ejemplos anteriores.

Si precedemos por `-` al índice de la izquierda de `:` y no ponemos ninguno a su derecha, lo que hacemos es obtener los últimos elementos

```
s[-10:] # Diez últimos elementos
```

```
'ideojuegos'
```

Si al contrario, precedemos por `-` al índice de la derecha, sin poner ningún índice a la izquierda de `:`, obtendremos todos los elementos salvo el número de elementos indicados por el índice (recordemos que si precedíamos por `-`, los índices empezaban desde el final).

```
s[:-10]
```

```
'Soy fan de los v'
```

Métodos para trabajar con strings

El método `.lower()` nos transforma el string que indiquemos a minúsculas.

```
s = "Me ENCANTAN el chocolate y las galletas"
s.lower()
```

```
'me encantan el chocolate y las galletas'
```

El método `.upper()`, por el contrario, lo transforma a mayúsculas.

```
s.upper()
```

```
'ME ENCANTAN EL CHOCOLATE Y LAS GALLETAS'
```

El método `.count()` cuenta cuántas veces aparece una letra o un string dentro del string al cuál le aplicamos dicho método.

```
s.count("o")
```

```
2
```

```
s.count("la")
```

```
2
```

El método `.capitalize()` convierte a mayúscula el primer caracter de un string.

```
s = "me encanta aprender con udemy"
s.capitalize()
```

```
'Me encanta aprender con udemy'
```

El método `.title()` convierte a mayúscula el primer caracter de cada palabra de un string.

```
s.title()
```

```
'Me Encanta Aprender Con Udemy'
```

El método `.swapcase()` convierte a mayúscula las minúsculas y viceversa.

```
s = "Me ENCANTA aprender con Udemy"
s.swapcase()
```

```
'mE encanta APRENDER CON uDEMY'
```

El método `.replace()` reemplaza el caracter (o caracteres) que le indiquemos por el string que queramos.

```
s = "Los tomberis son buenos"
s.replace("buenos", "malos")
```

```
'Los tomberis son malos'
```

El método `.split()` rompe el string en el caracter que le indiquemos y elimina dicho caracter.

```
s = "El elefante tiene las orejas muy grandes"
s.split("e") # Rompemos por la letra e minúscula
```

```
['El ', 'l', 'fant', ' ti', 'n', ' las or', 'jas muy grand', 's']
```

```
s.split(" ") # Rompemos por los espacios
```

```
['El', 'elefante', 'tiene', 'las', 'orejas', 'muy', 'grandes']
```

```
s.split("tiene") # Rompemos por la palabra tiene
```

```
['El elefante ', ' las orejas muy grandes']
```

El método `.strip()` elimina los espacios sobrantes a principio y final del string.

```
s = "       El elefante tiene las orejas muy grandes        "
s.strip()
```

```
'El elefante tiene las orejas muy grandes'
```

El método `.rstrip()` elimina los espacios sobrantes al final del string.

```
s.rstrip()
```

```
'       El elefante tiene las orejas muy grandes'
```

El método .lstrip() elimina los espacios sobrantes al principio del string.

```
s.lstrip()
```

```
'El elefante tiene las orejas muy grandes        '
```

El método `.find()` busca el caracter que indiquemos y nos devuelve la primera posición en la que aparece.

```
s = "Este es un curso de Python para hacer en casa o en cualquier lado"
s.find("e")
```

```
3
```

Si le pedimos buscar un conjunto de caracteres, nos devuelve la posición del primer caracter de dicho conjunto

```
s.find("casa")
```

```
41
```

El método `.find()` tiene otros dos parámetros de uso opcional: `start` y `end`, que sirven para indicar donde queremos que empiece la búsqueda y donde queremos que acabe.

```
s.find("e", 10) # Solamente indicamos start
```

```
18
```

```
s.find("e", 30, 40) # Indicamos start y end
```

```
35
```

El método `.index()` busca el caracter que indiquemos y nos devuelve la primera posición en la que aparece.

```
s = "Este es un curso de Python para hacer en casa o en cualquier lado"
s.index("e")
```

```
3
```

Si le pedimos buscar un conjunto de caracteres, nos devuelve la posición del primer caracter de dicho conjunto

```
s.index("casa")
```

```
41
```

El método `.index()` tiene otros dos parámetros de uso opcional: `start` y `end`, que sirven para indicar donde queremos que empiece la búsqueda y donde queremos que acabe.

```
s.index("e", 10) # Solamente indicamos start
```

```
18
```

```
s.index("e", 30, 40) # Indicamos start y end
```

```
35
```

Observación. Observemos que los métodos `.index()` y `.find()` son casi idénticos. El único punto en que difieren es que si el caracter indicado no se encuentra en el string, el método `.index()` arroja error, mientras que `.find()` arroja el índice -1.

El método `.rindex()` busca el caracter que indiquemos y devuelve el último indice en el que fue encontrado.

```
s.rindex("e")
```

```
58
```

También consta de los dos parámetros de uso opcional: `start` y `end`, que sirven para indicar donde queremos que empiece la búsqueda y donde queremos que acabe.

Otras funciones a tener en cuenta

La función `len()` nos devuelve el número de caracteres del string.

```
s = "Tengo hambre"
len(s)
```

```
12
```

Observación. Los espacios en blanco también son caracteres, por lo que éstos también son incluidos al contar el número de caracteres de los que consta un string.

Si tenemos un número en formato string, por mucho que sea un número para nosotros, en realidad `Python` no lo ve así. El gran problema es cuando queremos operar con un número que se encuentra en formato string. Ahí es donde entran en juego las funciones `int()` y `float()`, que lo que hacen es convertir a formato integer o float, respectivamente.

```
numero = "5"
type(numero)
```

```
str
```

En este caso, pasamos a formato integer:

```
numero_int = int(numero)
numero_int
```

```
5
```

```
numero_int ** 2
```

```
25
```

```
type(numero_int)
```

```
int
```

En este otro caso, pasamos a formato float:

```
numero_float = float(numero)
numero_float
```

```
5.0
```

```
numero_float ** 2 - numero_float
```

```
20.0
```

```
type(numero_float)
```

```
float
```

La función `input()` sirve para que el usuario introduzca un string por consola:

```
print("Introduce tu nombre: ")
name = input("Nombre: ")
```

```
Introduce tu nombre:
Nombre: Juan Gabriel Gomila
```

```
name
```

```
'Juan Gabriel Gomila'
```

Aquí también nos serán útiles las funciones `int()` y `float()`, pues si en vez del nombre queremos que el usuario nos indique su edad o su altura, querremos tratar dichos valores como números. Entonces, haríamos lo siguiente

```
print("Introduce tu edad: ")
age = int(input("Edad: "))
```

```
Introduce tu edad:
Edad: 32
```

```
print("Introduce tu altura: ")
height = float(input("Altura (en m): "))
```

```
Introduce tu altura:
Altura (en m): 1.66
```

```
print("La edad de {} es {} y mide {}m".format(name, age, height))
```

```
La edad de Juan Gabriel Gomila es 32 y mide 1.66m
```