PYTHON

CADENA DE CARACTERES

LIC. MARIELA ASENSIO

¨Me encanta Python”

“Estoy dando mis primeros pasos en este lenguaje”

Son ejemplos de cadenas de caracteres en Python. Conocer cómo manipular cadenas de textos es fundamental a la hora de escribir programas ya que estas nos permiten interactuar con el usuario, guardar información en archivos y mostrar contenido legible en nuestros programas.

DECLARACIÓN DE UNA CADENA

Una cadena es una secuencia inmutable de caracteres que en Python se representa con el tipo de dato str.
Esta secuencia se puede delimitar con comillas simples (‘ ‘) o con comillas dobles (“ “). Aunque se pueden utilizar ambos tipos de comillas , las cadenas se deben cerrar con el mismo tipo de comillas usado en la apertura.

```
cadena = 'probando cadena con comilla simple’
print(cadena)
cadena_2 = "Esta es otra cadena con comillas dobles"
print(cadena_2)
```

También es posible delimitar las cadenas utilizando cadenas triples simples (‘ ‘ ‘) o dobles (“ “ “). Estos tipos de comillas se suelen emplear con cadenas largas que ocupan más de una línea.

```
cadena_multiple = '''Cadena de texto
con más de una línea'''
print(cadena_multiple)
```

Hasta ahora, en los ejemplos mostrados, no hemos encontrado ninguna cadena con una comilla como carácter. ¿Qué ocurre si uno de los caracteres de las cadenas de texto que queremos declarar es precisamente una comilla? ¿Cómo podemos escribirlo?
Tenemos dos opciones:

- Escribir una comilla simple en una cadena delimitada por comillas dobles y viceversa:

```
print('Ada Lovelace, la primera programadora de la historia, dijo: "la imaginación es la facultad del descubrimiento, pre eminentemente. Es lo que penetra en los mundos nunca vistos a nuestro alrededor, los mundos de la ciencia".')
```

- Utilizar los caracteres especiales \' o \" formado por el carácter de escape diagonal invertida (\), qué hace que Python no intérprete las comillas como delimitadores de cadena

```
print('Ada Lovelace, la primera programadora de la historia,
dijo: \'la imaginación es la facultad del descubrimiento,
pre eminentemente. Es lo que penetra en los mundos nunca
vistos a nuestro alrededor, los mundos de la ciencia\'.')
```

Las comillas simples y las comillas dobles no son los únicos caracteres especiales que necesitan el carácter de escape (\) para poder ser incluidos en una cadena de texto.

Descripción	Representación
Salto de línea	\n
Tabulador	\t
Comilla doble	\"
Comilla simple	\'
Barra diagonal invertida	\\

ACCESO A LOS ELEMENTOS DE UNA CADENA

Las cadenas de texto son un tipo de dato compuesto, es decir, están formadas por elementos más pequeños que tienen significado, los caracteres. Al tratarse de un tipo de dato compuesto podemos acceder a la cadena como un todo o a las partes que la componen.

Para acceder a la cadena completa basta con utilizar el nombre de la variable con el que la hemos declarado:

```
serie = 'Game of Thrones'
serie
'Game of Thrones'
```

Para acceder a los elementos de la cadena se utiliza la técnica denominada indexación, la cual consiste en acceder a los elementos a partir de su posición. Para ello, se utiliza el operador corchete [ ]. Recuerden que las posiciones empiezan a contar desde cero. Así, para acceder al primer elemento de la cadena usaremos la posición cero, para el segundo elemento la posición uno, etc.

```
serie = 'Game of Thrones'
>>> serie
'Game of Thrones'
>>> serie[0]
'G'
>>> serie[3]
'e'
```

También, es posible acceder a los caracteres de una cadena de texto utilizando índices negativos. Estos índices cuentan desde el final de la cadena, de manera que la posición -1 representa el último carácter; la posición -2 el penúltimo; la posición -3 el antepenúltimo, y así sucesivamente.

```
>>> serie = 'Game of Thrones'
>>> serie
'Game of Thrones'
>>> serie[-1]
's'
>>> serie[-2]
'e'
>>> serie[-3]
'n'
```

Por último, para acceder a una porción de una cadena, es decir, a una subcadena, hay que indicar la posición de inicio y la posición límite separadas por dos puntos. Al escribir la posición límite debemos tener cuidado, puesto que el operador [posición_inicio : posición_límite] devuelve la subcadena que va desde el carácter que ocupa la posición inicio hasta el carácter que ocupa la posición límite, sin incluirlos. Así, si queremos acceder a la subcadena 'Game' utilizaremos como posición inicial 0 Y como posición límite 4, ya que si usamos 3 como límite perderemos el carácter e.

```
>>> serie[0:4]
'Game'
```

Si se omite la posición de inicio, se considera que la porción a extraer comienza en el primer carácter de la cadena, mientras que, si se omite la posición límite, la porción se extrae hasta el final de la cadena.

```
>>> serie[ :4]
'Game'
>>> serie[8 : ]
'Thrones'
```

¿Qué ocurre si se omiten ambas posiciones?

¿Qué obtenemos si hacemos serie[: : -1]?

CONCEPTO DE INMUTABILIDAD

Las cadenas de texto en Python son inmutables, esto significa que no se puede modificar. Son un tipo de dato que, una vez definido, no puede ser modificada.

```
>>> nombre = 'manuel'
>>> nombre[0]= 'M'
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    nombre[0]= 'M'
TypeError: 'str' object does not support
item assignment
```

Si intentamos modificar la primera letra, Python nos informa de un error que indica que las cadenas de texto no soportan la asignación por posición, es decir, que no se pueden modificar sus elementos.

Si queremos modificar una cadena ya declarada, crearemos una nueva que sea una variación de la original.

```
>>> nombre_modificado = 'M' +
nombre[1: ]
>>> nombre_modificado
'Manuel'
```

```
>>> nombre = 'manuel'
>>> nombre = 'M' + nombre[1: ]
>>> nombre
'Manuel'
```

OPERADORES ESPECIALES

Existen cuatro paradores que nos permiten operar con cadenas de texto: el operador de suma +, el operador de asignación de suma +=, el operador de multiplicación * y el operador módulo % .

El operador suma (+) permite concatenar cadenas de caracteres, es decir, generar una cadena a partir de la unión de varias cadenas. Es importante incluir los espacios en blanco de la cadena para que ésta sea legible, puesto que el operador + no los establece.

```
>>> serie = 'Game of Thrones'
>>> temporada = 8
>>> estreno = '14 de abril de 2019'
>>> mensaje = 'La temporada' + temporada + 'de' + serie + 'se estrena
el' + estreno
Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    mensaje = 'La temporada' + temporada + 'de' + serie + 'se estrena
el' + estreno
TypeError: can only concatenate str (not "int") to str
mensaje = 'La temporada' + str(temporada) + 'de' + serie + 'se estrena
el' + estreno
>>> mensaje
'La temporada8deGame of Thronesse estrena el14 de abril de 2019’
>>> mensaje_mejorado = 'La temporada ' + str(temporada) + ' de ' +
serie + ' se estrena el ' + estreno
>>> mensaje_mejorado
'La temporada 8 de Game of Thrones se estrena el 14 de abril de 2019'
```

Si queremos añadir cadenas al final de una cadena de caracteres podemos usar el operador de asignación de suma (+=). En el ejemplo tenemos una lista, personajes, con los nombres de algunos de los personajes de la serie Game of Thrones. A partir de esta lista hemos generado la cadena personajes principales como resultado de unir los nombres de los personajes de la lista. Para ello, usamos la estructura de control for, del operador in, del operador += y del operador +.

```
>>> personajes = ['Daenerys Targaryen', 'Tyrion Lannister', 'Jon
Snow', 'Cersei Lannister', 'Arya Stark']
>>> personajes_principales = " "
>>> for nombre in personajes:
  personajes_principales += nombre + ", "

>>> personajes_principales
' Daenerys Targaryen, Tyrion Lannister, Jon Snow, Cersei Lannister,
Arya Stark, '
```

Para concatenar una cadena de caracteres varias veces podemos utilizar el operador multiplicación *. Este operador nos permite repetir la cadena especificada N veces, pudiendo ser N el multiplicando o el multiplicador.

```
>>> texto = 'Hace ' + 3* 'mucho ' + 'tiempo...'
>>> texto
'Hace mucho mucho mucho tiempo...'
```

Por último como el operador módulo % permite interpolar variables dentro de una cadena de caracteres.

MÉTODOS PARA LA MANIPULACIÓN DE CADENAS

Len(cad)

Devuelve la longitud de la cadena, es decir como el número de caracteres que la forma. Los espacios en blanco también cuentan como caracteres.

```
>>> cadena = 'Juego de tronos es una serie de televisión estadounidense de fantasía medieval'
>>> len(cadena)
78
```

str.count(sub[, inicio[, limite]])

Devuelve el número de veces que se repite la subcadena sub en la cadena. Los argumentos opcionales del inicio y límite permiten limitar la búsqueda a una porción de la cadena

```
>>> texto = 'Hace ' + 3* 'mucho ' + 'tiempo...'
>>> texto
'Hace mucho mucho mucho tiempo...'
>>> texto.count('mucho')
3
```

str.find(sub[, inicio[, limite]])

Devuelve la posición de la primera ocurrencia de la subcadena sub en la cadena. Es posible que una cadena se repita varias veces a lo largo de una cadena, por eso este método devuelve la posición de la primera ocurrencia encontrada. En caso de no encontrar ninguna ocurrencia de la cadena, devuelve el valor -1. Los argumentos opcionales de inicio y límite permiten limitar la búsqueda a una porción de la cadena, aunque el índice devuelto será siempre sobre la posición global de la subcadena sub en la cadena y no sobre la posición relativa a los valores de inicio y límites indicados.

```
>>> frase = 'Erase una vez una princesa llamada
Jasmine'
>>> frase.find('princesa')
18
>>> frase.find('principe')
-1
```

str.index(sub[, inicio[, limite]])

Realiza la misma función que el método find(), pero genera un error ErrorValue si no encuentra la subcadena.

```
>>> frase = 'Erase una vez una princesa llamada jazmine'
>>> frase.index('genio')
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    frase.index('genio')
ValueError: substring not found
```

str.rfind(sub[, inicio[, limite]])

Devuelve la posición de la última ocurrencia de la subcadena sub en la cadena. En caso de no encontrar ninguna ocurrencia de la subcadena, devuelve el valor -1. A diferencia del método find() este método realiza la búsqueda desde el final de la cadena, en lugar de desde el principio.

```
>>> frase.rfind('vez')
10
>>> frase.rfind('bruja')
-1
```

str.rindex(sub[, inicio[, limite]])

Realiza la misma función que el método rfind(), pero genera un ErrorValue error si no encuentra la subcadena.

```
>>> frase.rindex('princesa')
18
>>> frase.rindex('bruja')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    frase.rindex('bruja')
ValueError: substring not found
```

str.capitalize()

Devuelve una copia de la cadena convirtiendo el primer carácter a mayúscula y el resto a minúscula

```
>>> cadena = 'pYTHON'
>>> cadena.capitalize()
'Python'
```

str.lower(sub[, inicio[, limite]])

Devuelve una copia de la cadena convirtiendo todos los caracteres a minúscula.

```
>>> cadena = 'pYTHON'
>>> cadena.lower()
'python'
```

str.upper()

Devuelve una copia de la cadena convirtiendo todos los caracteres a mayúsculas .

```
>>> cadena = 'pyTHon'
>>> cadena.upper()
'PYTHON'
```

str.swapcase()

Devuelve una copia de la cadena convirtiendo los caracteres en minúscula a mayúscula y viceversa.

```
>>> cadena = 'pyTHon'
>>> cadena.swapcase()
'PYthON'
```

str.strip([caracteres])

Devuelve una copia de la cadena tras eliminar los caracteres iniciales y finales que se corresponden con los caracteres especificados en caracteres. El argumento caracteres debe ser una cadena que especifique el conjunto de caracteres a eliminar. Si no se especifica este argumento, por defecto se utiliza el conjunto de caracteres espacio en blanco (string.whitespace), es decir, se eliminan los caracteres espacios en blanco iniciales y finales de la cadena.

```
>>> cadena = 'Erase una vez una pricesa llamada Jazmine'
>>> cadena.strip("E.e")
'rase una vez una pricesa llamada Jazmin'
>>> cadena.strip("Erase.ne")
' una vez una pricesa llamada Jazmi'
```

str.lstrip([caracteres])

Realiza la misma función que el método strip(), pero eliminando solamente los caracteres iniciales que se corresponden con los caracteres especificados en caracteres. El argumento caracteres debe ser una cadena que especifique el conjunto de caracteres que se quiere eliminar. Si no se especifica este argumento, por defecto se utiliza el conjunto de caracteres espacio en blanco, es decir, se eliminan los caracteres espacio en blanco iniciales de la cadena.

```
>>> cadena = 'Erase una vez una pricesa llamada Jazmine'
>>> cadena.lstrip()
'Erase una vez una pricesa llamada Jazmine'
>>> cadena.lstrip("E")
'rase una vez una pricesa llamada Jazmine'
```

str.rstrip([caracteres])

Igual que la anterior, pero en esta ocasión eliminando solamente los caracteres finales, en lugar de los iniciales.

```
>>> cadena = 'Erase una vez una pricesa llamada Jazmine'
>>> cadena.rstrip("azmine")
'Erase una vez una prnicesa llamada J'
```

str.replace(sub_antigua,sub_nueva[,num_ocurrencias])

Devuelve una copia de la cadena tras reemplazar las ocurrencias de la subcadena sub_antigua por la subcadena sub_nueva. Si se proporciona el argumento num_ocurrencias, solo se reemplazarán las primeras ocurrencias. Si no se proporciona este argumento, se reemplazarán todas las ocurrencias.

```
>>> texto_a = 'me  gusta programar en java'
>>> texto_a.replace('java','python')
'me  gusta programar en python'
>>> texto_b ='Práctico número 1. Práctico número 2. Práctico número 3.'
>>> texto_b.replace('número','Nº')
'Práctico Nº 1. Práctico Nº 2. Práctico Nº 3.'
```

str.split(sep=None, maxsplit=-1)

Divide la cadena utilizando sep como carácter separador y devuelve una lista con las palabras resultantes de la división. Los argumentos sep y maxsplit son opcionales. Si no se especifica el argumento sep o si su valor es None (valor por defecto), se realizará la división utilizando como caracteres paradores el conjunto de caracteres espacio en blanco.

El segundo argumento especifica el número máximo de divisiones a realizar, por lo que la lista tendrá como máximo maxsplit+1 elementos si no se especifica este argumento o si su valor es -1 (valor por defecto), no se establecerá límite en el número de particiones a realizar por lo que se realizarán todas las particiones posibles.

```
>>> texto_b
'Práctico número 1. Práctico número 2. Práctico número 3.'
>>> texto_b.split('número')
['Práctico ', ' 1. Práctico ', ' 2. Práctico ', ' 3.']
>>> texto_b.split('.')
['Práctico número 1', ' Práctico número 2', ' Práctico número 3', '']
>>> texto_b.split('.', 2)
['Práctico número 1', ' Práctico número 2', ' Práctico número 3.']
```

str.join(iterable)

Devuelve una cadena resultante de unir las cadenas del iterable indicado como argumento, utilizando como separador la cadena desde la que se realiza la llamada. Si en el iterable hay valores que no son cadenas, se generará un error TypeError.

```
>>> texto = ['Práctico N°1', 'Práctico N°2', 'Práctico N°3']
>>> '-'.join(texto)
'Práctico N°1-Práctico N°2-Práctico N°3'
>>> ' - '.join(texto)
'Práctico N°1 - Práctico N°2 - Práctico N°3'
```

FORMATEO DE UNA CADENA

Operador %

Las cadenas de Python ofrecen el operador módulo (%), que permite interpolar variables dentro de una cadena de caracteres. Para ello, hay que indicar mediante caracteres de tipo la posición del texto en la que se insertarán los valores de las variables

```
>>> nombre = 'Mariela'
>>> '%s domina lenguajes como Java y C' % nombre
'Mariela domina lenguajes como Java y C'
>>> lenguaje = 'Python'
>>> '%s domina lenguajes como Java y C y quiere aprender a programar en %s'
%(nombre,lenguaje)
'Mariela domina lenguajes como Java y C y quiere aprender a programar en Python'
>>> año=2019
>>> '%s domina lenguajes como Java y C y quiere aprender a programar en %s antes que
finalice el año %d' %(nombre,lenguaje,año)
'Mariela domina lenguajes como Java y C y quiere aprender a programar en Python
antes que finalice el año 2019'
```

Función str.format()

Esta función permite realizar un formateo de la cadena desde la que se realiza la llamada. Los valores que se deben insertar en la misma se especifican mediante llaves { }. Existen dos formas de representar las variables entre llaves. La primera de ellas consiste en incluir entre las llaves un índice numérico para identificar a la variable y se conoce como formateo por posición. La segunda de ellas se basa en el uso de un nombre asociado a la variable a incluir y se denomina formateo por nombre.

```
>>> nombre = 'Mariela'
>>> lenguaje = 'Python'
>>> año=2019
>>> '{0} domina lenguajes como Java y C, y quiere aprender a
programar en {1} antes de que finalice el año {2}.’
.format(nombre,lenguaje,año)
'Mariela domina lenguajes como Java y C, y quiere aprender a
programar en Python antes de que finalice el año 2019.'
```

```
'{nom} domina lenguajes como Java y C, y quiere aprender a
programar en {len} antes de que finalice el año {a}.’
.format(nom=nombre,len=lenguaje,a=año)
'Mariela domina lenguajes como Java y C, y quiere aprender a
programar en Python antes de que finalice el año 2019.'
```

CARACTERES DE TIPO

Carácter de Tipo	Significado
s	Cadena de texto
d	Entero
f	Real
e	Real en formato exponencial
o	Octal
x	Hexadecimal

F-strings

Las cadenas literales, f – strings, se representan con una f al principio y contienen llaves con las variables o expresiones cuyos valores se insertarán en la cadena. Se trata de un formateo basado en la inserción directa de variables y expresiones.

```
>>> nombre = 'Mariela'
>>> lenguaje = 'Python'
>>> año=2019
>>> f'{nombre} domina lenguajes como Java y C, y quiere aprender a
programar en {lenguaje} antes de que finalice el año {año}.'
'Mariela domina lenguajes como Java y C, y quiere aprender a
programar en Python antes de que finalice el año 2019.'
```

MÉTODOS DE VALIDACIÓN

Saber si una cadena es alfanumérica
Método: isalnum()

Retorna: True o False

```
>>> cadena = "pepegrillo 75"
>>> print cadena.isalnum()
False
>>> cadena = "pepegrillo"
>>> print cadena.isalnum()
True
>>> cadena = "pepegrillo75"
>>> print cadena.isalnum()
True
```

Saber si una cadena es alfabética
Método: isalpha()

Retorna: True o False

```
>>> cadena = "pepegrillo 75"
>>> print cadena.isalpha()
False
>>> cadena = "pepegrillo"
>>> print cadena.isalpha()
True
>>> cadena = "pepegrillo75"
>>> print cadena.isalpha()
False
```

Saber si una cadena es numérica
Método: isdigit()

Retorna: True o False

```
>>> cadena = "pepegrillo 75"
>>> print cadena.isdigit()
False
>>> cadena = "7584"
>>> print cadena.isdigit()
True
>>> cadena = "75 84"
>>> print cadena.isdigit()
False
>>> cadena = "75.84"
>>> print cadena.isdigit()
False
```