MÓDULOS: ESP - ECP

CADENAS DE CARACTERES EN PYTHON

José L. Berenguel

Tabla de Contenidos

1. Introducción a las cadenas de caracteres

- Las cadenas de caracteres (**str**) es un tipo secuencia (**sequence**) en Python. Contiene caracteres en un orden específico y se puede acceder a cada valor individual a través de su posición.
- Es un objeto **inmutable**. Esto significa que una vez creado, cualquier modificación del objeto supone la creación de un nuevo objeto.
- Se declara entre **comillas simples** o **comillas dobles**.
- Si se desea incluir una comilla o carácter especial es necesario escapar ese carácter con ****. Por ejemplo, un salto de línea se escribe con **\n**.

```
# Cadena con comillas dobles escapadas y saltos de línea
nombre = "IES \"Zaidín-Vergeles\"\nCalle Primavera\nGranada"
print(nombre)

# Otra opción es usar comillas diferentes a las usadas en el interior
nombre = 'IES "Zaidín-Vergeles"\nCalle Primavera\nGranada'
print(nombre)
```

1. Introducción a las cadenas de caracteres

- Con el caracter **** se puede indicar el código octal del carácter que se desea imprimir. **”\101\103”** representa los caracteres **AC**.
- Si queremos escribir una cadena tal cual la escribamos podemos usar **comillas triples**. Se conoce como **docstring**.
- Si usamos el carácter **r** como prefijo de la cadena se ignoran las secuencias de escape y la cadena se representa tal cual esté escrita. Se conoce como **raw string**. Por ejemplo **r”\101\103”** representa la cadena **\101\103**.

```
print("\101\103") # AC
print(r"\101\103")# \101\103
```

2. Formato de cadenas de caracteres

- Hay varios métodos para **dar formato a una cadena**.
  - Cadenas **f-string**. Se usa la letra **f** como prefijo de la cadena. El interior de la cadena contiene variables entre llaves {} (**placeholder**). Los valores de estas variables serán sustituidos en la cadena.

    f”Bienvenido/a {nombre} al curso de Python”
  - Método **.format()**. La opción usada antes de la versión 3.6 cuando aparecieron las f-string (ahora la forma preferida).

    ”Bienvenido/a {} al curso de {}”.format(nombre, lenguaje)
  - Operador **%**. Funciona de manera similar a la función **printf** de C. El operador **%** junto con un modificador que indica el tipo de la variable a sustituir.

    nombre = ”Maria”; ”Bienvenida %s al curso de Python” %nombre

https://www.w3schools.com/python/python_string_formatting.asp

https://realpython.com/python-string-formatting/

3. Cadenas f-string

- **Cadenas f-string** (formatted string literals). Añadidas en la versión 3.6.
  - Permiten agregar objetos y expresiones en el interior de una cadena. Los datos que contienen son interpolados a una representación de string.
  - La cadena incluye **placeholders {}** donde se incluye el nombre de la variable o expresión cuyo valor será sustituido en la cadena.

```
# f-string que incluye el dato en bruto
f"The number is {42}"

a = 5
b = 10
# f-string que contiene una expresión aritmética
f"{a} más {b} = {a + b}" # 5 más 10 = 15'
```

4. Format Mini-Language

- **Format Mini-Language** es una especificación de Python que permite formatear los valores de acuerdo con las características deseadas.

BNF Grammar

```
format_spec     ::=  [[fill]align][sign]["z"]["#"]["0"][width]
                     [grouping_option]["." precision][type]
fill            ::=  <any character>
align           ::=  "<" | ">" | "=" | "^"
sign            ::=  "+" | "-" | " "
width           ::=  digit+
grouping_option ::=  "_" | ","
precision       ::=  digit+
type            ::=  "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" |
                     "G" | "n" | "o" | "s" | "x" | "X" | "%"
```

https://realpython.com/python-format-mini-language/

4. Format Mini-Language

- **Format Mini-Language.**
  - Campo **width**. Indica el número de caracteres que se ocuparán.
  - Campo **align**. Indica la posición que ocupará la cadena en el espacio disponible.
  - Campo **fill**. Carácter de relleno con el que se ocupará el espacio sobrante.
- Valores para el campo **align**:

Value	Description
<	Aligns the interpolated value to the left within the available space. It's the default alignment for most objects.
>	Aligns the interpolated value to the right within the available space. It's the default alignment for numbers.
^	Aligns the interpolated value in the center of the available space.
=	Adds padding after the sign but before the digits in numeric values. It's the default for numbers when `0` immediately precedes the field width.

https://realpython.com/python-format-mini-language/

4. Format Mini-Language

- **Format Mini-Language.**

```
text = "Hello!"

# align="<" and width=30
f"{text:<30}"
#'Hello!                        '

# align="^" and width=30
f"{text:^30}"
#'            Hello!            '

# align=">" and width=30
f"{text:>30}"
#'                        Hello!'

# fill="=", align="^" and width=30
f"{text:=^30}"
#'============Hello!============'
```

https://realpython.com/python-format-mini-language/

4. Format Mini-Language

- **Format Mini-Language.**
  - Podemos representar valores enteros en diferentes formatos

Representation Type	Type	Description
b	Binary	Converts the number to base 2
c	Character	Converts the number to the corresponding Unicode character
d	Decimal Integer	Converts the number to base 10
o	Octal	Converts the number to base 8
x or X	Hexadecimal	Converts the number to base 16, using lowercase or uppercase letters for the digits above 9
n	Number	Works the same as d, except that it uses the current locale setting to insert the appropriate thousand separator characters
None	Decimal Integer	Works the same as d

https://realpython.com/python-format-mini-language/

4. Format Mini-Language

- **Format Mini-Language.**

```
number = 42

f"int: {number:d},  hex: {number:x},  oct: {number:o},  bin: {number:b}"
#'int: 42,  hex: 2a,  oct: 52,  bin: 101010'
```

https://realpython.com/python-format-mini-language/

4. Format Mini-Language

- **Format Mini-Language.**
  - Para valores en punto flotante:

Representation Type	Description
e or E	Scientific notation with the separator character in lowercase or uppercase, respectively
f or F	Fixed-point notation with `nan` and `inf` in lowercase or in uppercase, respectively
g or G	General format where small numbers are represented in fixed-point notation and larger numbers in scientific notation
n	General format (same as g), except that it uses a locale-aware character as a thousand separator

https://realpython.com/python-format-mini-language/

4. Format Mini-Language

- **Format Mini-Language.**

```
# Representación en notación científica
numero = 1234567890
f"{numero:e}"
#'1.234568e+09'

from math import pi

# Valor de la constante math.pi
# 3.141592653589793

# 4 posiciones decimales
f"{pi:.4f}"
#'3.1416'

# 8 posiciones decimales
f"{pi:.8f}"
#'3.14159265'
```

https://realpython.com/python-format-mini-language/

5. Formato con str.format()

- Método **str.format()**.
  - Es similar a f-string y también soporta el Format Mini-Language.
  - La llamada se realiza como un método de la cadena usada como plantilla.
  - La cadena incluye placeholders {} que pueden contener el nombre de la variable o no.

```
debito = 300.00
credito = 450.00

# Cadena que contiene los placeholders con formato
plantilla = "Débito: {0:.2f}€, Crédito: {1:.2f}€, Balance: {2:.2f}€"
# Se sustituyen los valores de las variables de forma posicional
plantilla.format(debito, credito, credito - debito)
'Débito: 300.00€, Crédito: 450.00€, Balance: 150.00€'
```

6. Formato con el operador %

- **Operador %**. Funciona de manera similar a las cadenas con formato de C.
  - El interior de la cadena incluye el % junto con unos modificadores de formato.
  - A continuación de la cadena se incluye el % junto a la variable que sustituirá al modificador.
  - En caso de que haya más de un modificador, las variables se separan por comas usando paréntesis.

```
# Operador % con una variable
nombre = "Juan"
"Hola, %s, bienvenido a Python" %nombre
# 'Hola, Juan, bienvenido a Python'

# Operador % con más de una variable
"%f %d" % (33.4, 44.4) #'33.400000 44'
```

6. Formato con el operador %

- **Operador %**. Modificadores de tipo

d	Signed integer decimal
i	Signed integer decimal
o	Signed octal value
x	Signed hexadecimal with lowercase prefix
X	Signed hexadecimal with uppercase prefix
e	Floating-point exponential format with lowercase e
E	Floating-point exponential format with uppercase E
f	Floating-point decimal format
F	Floating-point decimal format

6. Formato con el operador %

- **Operador %**. Modificadores de tipo

g	Floating-point format
G	Floating-point format
c	Single character (accepts integer or single character string)
r	String as per calling `repr()`
s	String as per calling `str()`
a	String as per calling `ascii()`
%	A percentage character (%) in the result if no argument is converted

6. Formato con el operador %

- **Operador %**. Ejemplo con diccionario y campos con nombre

```
# Variable diccionario
juan =  {"nombre": "Juan", "apellido": "Gómez"}

# Modificadores con nombre y sustitución con variable diccionario
"Nombre completo: %(nombre)s %(apellido)s" % juan
# 'Nombre completo: Juan Gómez'
```

6. Formato con el operador %

- **Operador %**. Ejemplo con alineado y tamaño que ocupará la representación en cadena.

```
# Tamaño mínimo de 20 caracteres
"%-20f" % 3.1416  # El signo - alinea a la izquierda
# '3.141600            '

# Tamaño de 20 caracteres alineado a la derecha
"%20f" % 3.1416
#'            3.141600'
```

6. Formato con el operador %

- **Operador %**. Ejemplo con tamaño dinámico.

```
# Ancho dinámico
ancho = 20
# El asterisco * permite incluir el ancho a través de una variable
"%-*f" % (ancho, 3.1416)
#'3.141600            '

"%-*f El ancho es %d" % (ancho, 3.1416, ancho)
#'3.141600             El ancho es 20'
```

7. Operaciones con cadenas

- Operaciones habituales con cadenas.
  - **Concatenar** cadenas con el operador **+**.
  - Obtener la **longitud** de la cadena con **len()**.
  - Convertir a cadena con la función **str()**.
  - Acceso a un carácter de la cadena a través de un **índice entre corchetes** y valores entre 0 y longitud-1. Si el índice es negativo, se comienza desde el carácter final.
  - Crear subcadenas o **slice** con **[inicio:fin-1]**. Si no se indica el fin, se obtiene el resto de la cadena. O también **[inicio:fin-1:incremento]**

```
ies = "Zaidin-Vergeles"
print(ies[0])   # Z
print(ies[-2])  # e
print(ies[0:6]) # Zaidin
print(ies[7:])   # Vergeles
print(ies[0:6:2]) # Zii
print(ies[0::2])  # Zii-egls
```

7. Operaciones con cadenas

- Métodos de la clase **str**.
  - Un método es una función accesible a través del objeto y que aplica la operación sobre el contenido del mismo.
  - Como se ha dicho anteriormente, **str** es inmutable, por lo que las operaciones no modifican el contenido del objeto, sino que devuelve un objeto con el nuevo valor. Por ejemplo, **”hola”.upper()**, devuelve la cadena ”HOLA”.
- Algunos métodos interesantes:
  - **capitalize()**. Convierte el primer carácter a mayúscula y el resto a minúsculas.
  - **lower()**. Obtiene la cadena con todos los caracteres en minúscula.
  - **swapcase()**. Convierte las minúsculas en mayúsculas y viceversa.
  - **isalnum()**. Devuelve True si la cadena contiene solo letras y números.
  - **isalpha()**. Devuelve True si la cadena contiene solo caracteres alfabéticos.
  - **join(<iterable>)**. Concatena la cadena con las cadenas del iterable pasado por parḿaetro.
  - **split()**. Devuelve una lista con las cadenas separadas según el carácter separador indicado por parámetro.

https://docs.python.org/3/library/stdtypes.html#textseq

7. Operaciones con cadenas

- El módulo **string** contiene algunas constantes interesantes.
  - **string.ascii_letters**.
  - **string.ascii_lowercase**.
  - **string.ascii_uppercase**.
  - **string.digits**.
  - **string.hexdigits**.
  - **string.punctuation**.
  - **string.printable**.
  - **string.whitespace**.

https://docs.python.org/3/library/string.html

Cadenas de caracteres en Python

FIN