

# INTRODUCCIÓN A LA PROGRAMACIÓN

CON  python

## CADENAS

### Cadenas. Formato

Antes de empezar a ver las estructuras de datos comenzando con cadenas, nos vamos a detener en el formato de la salida que le damos a nuestros datos.

Hay veces que nos interesa generar una salida por `print` en la que se mezclan diferentes tipos de variables

## Cadenas. Formato

---

A medida que ha ido evolucionando Python, han ido apareciendo diferentes formas, pero a día de hoy la más utilizada es la que se conoce como **f-string** (desde Python 3.6).

También llamado *formatted string literals* ó *formateo literal de cadenas*, f-Strings tiene una sintaxis simple y fluida que hará más sencillo el darle formato a cadenas de texto.

## ¿qué son las f-Strings?

---

En Python, una cadena de texto normalmente se escribe entre comillas dobles (""") o comillas simples ("). Para crear f-strings, solo tienes que agregar la letra f o F mayúscula antes de las comillas.

Por ejemplo, `"esto"` es una cadena de texto normal y `f"esto"` es una f-string.

Si quieres mostrar variables utilizando f-strings, solo tienes especificar el nombre de las variables entre llaves {}. Y al ejecutar tu código, todos los nombres de las variables serán remplazados con sus respectivos valores.

No es necesario que hagamos conversión de la variable a cadena.

# ¿qué son las f-Strings?

---

## Ejemplo

```
nombre = 'Pepe'  
edad = 20
```

```
print(f'{nombre} tiene {edad} años')
```

Dentro de las f-string se pueden incluir expresiones

```
print(f'{nombre} tiene {edad} años, naciste en {2021-edad}')
```

# ¿qué son las f-Strings?

---

## Más ejemplos

```
num1 = 83  
num2 = 9  
print(f"El producto de {num1} y {num2} es {num1 * num2}.")
```

Y la salida es:

```
El producto de 83 y 9 es 747.
```

## Cadenas. Formato

---

También podemos alinear decimales y números de la siguiente manera:

- Usaremos ":" después de la variable indicando el formato
- número de cifras totales (enteras + decimales) y tras un punto "." el número de decimales.
- Terminaremos la expresión con una "f"

```
valor1 = 1.45  
valor2 = 45.45  
print(f'{valor1:6.3f}')  
print(f'{valor2:6.3f}')
```

```
1.450  
45.450
```

## Cadenas. Formato

---

Más ejemplos de formatos

```
number = 0.123456789  
print(f'Formatear el valor con cuatro dígitos: {number:.4f}')
```

```
print(f'Imprimir el valor como un porcentaje: {number:.2%}')
```

```
print(f'Formato exponencial: {number:e}')
```

Formatear el valor con cuatro dígitos: 0.1235

Imprimir el valor como un porcentaje: 12.35%

Formato exponencial: 1.234568e-01

## Cadenas. Formato

---

Más ejemplos:

```
numbers = [1,10,100]

for number in numbers:
    print(f'El valor con ceros es: {number:04}')
```

El valor con ceros es: 0001

El valor con ceros es: 0010

El valor con ceros es: 0100

## Cadenas. Formato

---

Alienar las cadenas de texto

Otra opción bastante interesante es aligerar los mensajes a la izquierda, derecha o centro. Lo que se indica respectivamente con >, < y ^ seguido del número de caracteres que tiene la cadena texto. Algo que se puede ver cómo funciona en el siguiente ejemplo:

```
name = 'analytics'

print(f'12345678901234567890')
print(f'{name : >20}')
print(f'{name : <20}')
print(f'{name : ^20}')
```

12345678901234567890

analytics

analytics

analytics

## Cadenas. Formato

---

También puedes llamar funciones y métodos con f-strings.

## Cadenas. Resumen Formato

---

Por si los encuentras en el código, otros formatos más antiguos pero que dan el mismo resultado son:

```
print("{} tiene {} años".format(nombre,edad))
print("%s tiene %d años"%(nombre,edad)) # %s indica que nombre es cadena y %d
que edad es entero
```

### Vamos a usar esta forma

```
print(f'{nombre} tiene {edad} años')
```

# Estructuras de datos

---

Una **estructura de datos** es la **colección de datos** que se caracterizan por su **organización** y las **operaciones** que se definen en ella.

## Estructuras de datos

---

En general:

- **cadena**s
- listas
- tuplas
- diccionarios
- conjuntos

Nombre	Tipo	Ejemplo	Descripción
Lista	<code>list</code>	<code>[1, 2, 3]</code>	Secuencia heterogénea de datos mutable
Tupla	<code>tuple</code>	<code>1, 2, 3</code> o <code>(1, 2, 3)</code>	Secuencia heterogénea de datos inmutable
Rango	<code>range</code>	<code>range(1, 20, 2)</code>	Secuencia de enteros inmutable
Cadena de caracteres	<code>str</code>	<code>'Hola'</code>	Secuencia de caracteres inmutable
Diccionario	<code>dict</code>	<code>{'a':1, 'b':2, 'c':3}</code>	Tabla asociativa de valores únicos (clave, valor)
Conjunto	<code>set</code>	<code>{1, 2, 3}</code>	Colección sin orden de valores únicos

# Cadenas

---



# Cadenas

---

Una cadena es una secuencia **inmutable** de caracteres, delimitada por simples (') o con comillas dobles (")

La función **print()** muestra por pantalla el contenido de la cadena, pero no las comillas delimitadoras de las cadenas.

```
print("Hello")
print('Hello')
```



## Cadenas. Asignación a una variable

---

La asignación de una cadena a una variable se realiza con el nombre de la variable seguido de un signo igual y la cadena:

```
a = "Hello"  
print(a)
```

## Cadenas. Operaciones

---

Algunas de las operaciones con cadenas son:

- concatenar y multiplicar
- comparar
- pertenencia
- obtener un carácter
- obtener una porción
- obtener la longitud
- convertir a mayúsculas y a minúsculas
- reemplazar
- separar
- juntar
- encontrar
- recorrer

## Cadenas. Concatenación

Este término significa **juntar** cadenas de caracteres. El proceso de *concatenación* se realiza mediante el operador de suma (+). Ten en cuenta que debes marcar explícitamente dónde quieres los espacios en blanco y colocarlos entre comillas.

En este ejemplo, la cadena de caracteres "c" tiene el contenido "HelloWorld"

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

```
HelloWorld
-----
(program exited with code: 0)
```

## Cadenas. Multiplicación

Si quieres **varias copias** de una cadena de caracteres utiliza el operador de multiplicación (\*). En este ejemplo, la cadena de caracteres *mensaje2a* lleva el contenido "Hola" tres veces, mientras que la cadena de caracteres *mensaje2b* tiene el contenido "Mundo". Ordenemos imprimir las dos cadenas.

```
a = "Hello "
c = a*3
print(c)
```

```
Hello Hello Hello
-----
(program exited with code: 0)
```

## Cadenas. Comparación

Los operadores de comparación trabajan sobre cadenas. Para ver si dos cadenas son iguales:

```
fruta="manzana"
if fruta=="banana":
    print("Sí, tenemos bananas")
```

## Cadenas. Pertenencia

Se puede comprobar si un elemento (subcadena) pertenece o no a una cadena de caracteres con los operadores `in` y `not in`

```
cadena = "informática"
pertenece = "a" in cadena
print(pertenece)
```

```
C:\WINDOWS\SYSTEM32\cmd.exe
True
```

```
cadena = "informática"
pertenece = "a" not in cadena
print(pertenece)
```

```
C:\WINDOWS\SYSTEM32\cmd.exe
False
```

## Cadenas. Obtener un carácter

Obtiene el carácter en la posición 1 (recuerda que el primer carácter tiene la posición 0):

```
a = "Hello, World!"
print(a[1])
```

Hello

0 1 2 3 4  
-5 -4 -3 -2 -1

```
e
-----
(program exited with code: 0)
```

## Cadenas. Obtener un carácter

Los índices positivos comienzan en 0 pero también se pueden utilizar enteros negativos como índices. Por ejemplo, -1 se refiere al último elemento de la lista, -2 se refiere al anteúltimo, etc.

```
a="Hello"
print(a[-2])
```

Hello

0 1 2 3 4  
-5 -4 -3 -2 -1

```
l
-----
(program exited with code: 0)
Presione una tecla para continuar . . .
```

## Cadenas. Obtener una porción: slicing (rebanar)

---

Slice significa rebanada

Puede devolver un rango de caracteres. Especifique el índice inicial y el índice final, separados por dos puntos, para devolver una parte de la cadena.

Obtiene los caracteres de la posición 2 a la posición 5 (no incluido):

```
b = "Hello, World!"
print(b[2:5])
```

```
llo
-----
(program exited with code: 0)
```

## Cadenas. Obtener una porción: slicing (rebanar)

---

En forma general

<code>a[inicio:fin]</code>	# desde <b>inicio</b> hasta <b>fin-1</b>
<code>a[inicio:]</code>	# desde <b>inicio</b> hasta el <b>final de la cadena</b>
<code>a[:fin]</code>	# desde el <b>principio de la cadena</b> hasta <b>fin-1</b>
<code>a[:]</code>	# Todos los elementos
<code>a[inicio:fin:pasos]</code>	# desde <b>inicio</b> hasta <b>fin-1</b> , en los <b>pasos</b> indicados por pasos

## ¡Pruébalo!

---

```
>>> cadena = "informática"
>>> cadena[2:5]
>>> cadena[2:7:2]
>>> cadena[:5]
>>> cadena[5:]
>>> cadena[-1:-3]
>>> cadena[::-1]
```

## Cadenas. Longitud

---

La función `len()` devuelve la longitud de una cadena:

```
a = "Hello, World!"
print(len(a))
```

```
13
-----
(program exited with code: 0)
```

# Las cadenas son inmutables

---

Cuando creamos una variable de tipo cadena de caracteres, estamos creando un **objeto** de la **clase** `str`. Una clase específica determina las operaciones que pueden realizar los objetos de dicha clase.

Cada vez que creamos una variable de una determinada clase, creamos un objeto, que además de guardar información (en nuestro caso los caracteres de la cadena) puede realizar distintas operaciones que llamamos **métodos**.

No podemos cambiar los caracteres de una cadena de la siguiente forma:

```
cadena = "informática"
cadena[2]="g"
```

# Las cadenas son inmutables

---

Esto implica que al usar un método la cadena original no cambia, el método devuelve otra cadena modificada. Veamos un ejemplo:

```
>>> cadena = "informática"
>>> cadena.upper()
'INFORMÁTICA'
>>> cadena
'informática'
>>>
```

Si queremos cambiar la cadena debemos modificar su valor con, por ejemplo, el operador de asignación:

```
>>> cadena = cadena.upper()
>>> cadena
'INFORMÁTICA'
```

## Cadenas. Pasar a minúsculas

---

El método `lower()` devuelve la cadena en minúsculas:

```
a = "Hello, World!"  
print(a.lower())
```

```
hello, world!  
-----  
(program exited with code: 0)
```

## Cadenas. Pasar a mayúsculas

---

El método `upper()` devuelve la cadena en mayúsculas:

```
a = "Hello, World!"  
print(a.upper())
```

```
HELLO, WORLD!  
-----  
(program exited with code: 0)
```



## Cadenas. Reemplazar

El método `replace()` reemplaza una cadena con otra cadena:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

```
Jello, World!
-----
(program exited with code: 0)
```

En forma general

`cadena.replace(viejovalor, nuevovalor, cantidad)`

Cantidad es opcional e indica el número de apariciones que quieres reemplazar

## Cadenas. Separar

El método `split()` divide la cadena en subcadenas si encuentra instancias del separador.

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

```
['Hello', ' World!']
-----
(program exited with code: 0)
```

# ¡Pruébalo!

Podemos indicar el separador:

```
>>> gazpacho = 'tomate-pepino-pimiento-aceite-vinagre'
>>> gazpacho.split('-')
['tomate', 'pepino', 'pimiento', 'aceite', 'vinagre']
```

No tiene por qué limitarse a un único carácter:

```
>>> extensiones = '201abc202abc203'
>>> extensiones.split('abc')
['201', '202', '203']
```

Podemos precisar el **número de veces** que meteremos el cuchillo:

```
>>> python = '1 Python Programas Generador de claves aleatorias'
>>> python.split(' ', 3)
['1', 'Python', 'Programas', 'Generador de claves aleatorias']
```

Nota: se podría haber utilizado maxsplit

## Cadenas. Juntar

Planteemos ahora el problema inverso: dada una lista de cadenas, fusionarla en una única cadena empleando determinado carácter como separador. Las cadenas disponen de un método que logra precisamente eso: `join()`.

Percátate de una sutileza en la frase anterior: he dicho que el método `join()` es propio del tipo cadena y no del tipo lista. Por lo tanto no se puede aplicar sobre una lista sino sobre una cadena.

```
>>> estaciones = ['Primavera', 'Verano', 'Otoño', 'Invierno']
>>> ' '.join(estaciones)
'Primavera Verano Otoño Invierno'
```

## Cadenas. Juntar

El separador puede ser cualquier *cadena*, obviamente:

```
>>> ' <---> '.join(estaciones)
'Primavera <---> Verano <---> Otoño <---> Invierno'
```

## Cadenas. Encontrar

El método `find()` se utiliza para buscar una subcadena dentro de una cadena. Si encuentra la subcadena devuelve el índice por el que comienza la primera aparición de dicha subcadena, y -1 en caso contrario.

```
cadena = 'me gusta la mermelada'
i=cadena.find('la')
print(i)
i=cadena.find('lo')
print(i)
```

```
9
-1
-----
(program exited with code: 0)
Presione una tecla para continuar . . .
```

## Cadenas. Encontrar

El método `index()` devuelve el índice de una subcadena o carácter dentro de la cadena (si se encuentra). Si no se encuentra la subcadena o el carácter, genera un error: *ValueError: substring not found*.

```
cadena = "me gusta la mermelada"
i = cadena.index("la")
print(i)
i = cadena.index("lo")
```

Traceback (most recent call last):  
 File "C:\Users\gabri\Desktop\cadenas.py", line 4, in <module>  
 i = cadena.index("lo")  
ValueError: substring not found

## Cadenas. Recorrido

Usar un índice para recorrer un conjunto de valores es tan común que Python proporciona una sintaxis alternativa más simple — el bucle `for`:

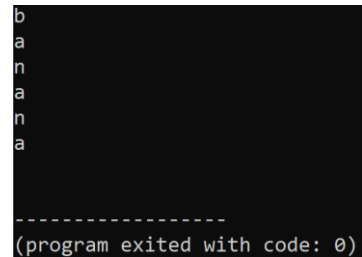
```
fruta="banana"
for caracter in fruta:
    print(caracter)
```

```
b
a
n
a
n
a
-----
(program exited with code: 0)
```

## Cadenas. Recorrido

Muchos cálculos implican procesar una cadena carácter por carácter. A menudo empiezan por el principio, seleccionan cada carácter por turno, hacen algo con él y siguen hasta el final. Este patrón de proceso se llama recorrido. Una forma de codificar un recorrido es con una sentencia while:

```
fruta="banana"
indice=0
while indice < len(fruta):
    letra = fruta[indice]
    print(letra)
    indice=indice+1
```



```
b
a
n
a
n
a
-----
(program exited with code: 0)
```

## Cadenas. capitalize() y swapcase()

- capitalize() nos permite devolver la cadena con el primer carácter en mayúsculas.

```
>>> cad = "hola, como estás?"
>>> print(cad.capitalize())
Hola, como estás?
```

- swapcase(): devuelve una cadena nueva con las minúsculas convertidas a mayúsculas y viceversa.

```
>>> cad = "Hola Mundo"
>>> print(cad.swapcase())
hOLA mUNDO
```

## Cadenas. `title()` y `count()`

---

- `title()`: Devuelve una cadena con los primeros caracteres en mayúsculas de cada palabra.

```
>>> cad = "hola mundo"
>>> print(cad.title())
Hola Mundo
```

- `count()`: Es un método al que indicamos como parámetro una subcadena y cuenta cuantas apariciones hay de esa subcadena en la cadena.

```
>>> cad = "bienvenido a mi aplicación"
>>> cad.count("a")
3
```

## Cadenas. `count()`

---

Además podemos indicar otro parámetro para indicar la posición desde la que queremos iniciar la búsqueda. Y otro parámetro optativo para indicar la posición final de búsqueda.

```
>>> cad = "bienvenido a mi aplicación"
```

```
>>> cad.count("a", 16)
2
>>> cad.count("a", 10, 16)
1
```

## Cadenas. `startswith()`

---

- `startswith()` nos indica con un valor lógico si la cadena empieza por la subcadena que hemos indicado como parámetro. Podemos indicar también con otro parámetro la posición donde tiene que buscar.

```
cad = "bienvenido a mi aplicación"
```

```
>>> cad.startswith("b")
True
>>> cad.startswith("m")
False
>>> cad.startswith("m",13)
True
```

## Cadenas. `endswith()`

---

- `endswith()` igual que la anterior pero indica si la cadena termina con la subcadena indicada. En este caso, se puede indicar la posición de inicio y final de búsqueda.

```
cad = "bienvenido a mi aplicación"
```

```
>>> cad.endswith("ción")
True
>>> cad.endswith("ción",0,10)
False
>>> cad.endswith("nido",0,10)
True
```

## Cadenas. `strip()`

- `strip()`: Devuelve una cadena donde se han quitado los espacios del principio y del final. Si indicamos una subcadena como parámetro quitará dicha subcadena del principio y del final.

```
>>> cadena = "   www.eugeniabahit.com   "
>>> print(cadena.strip())
www.eugeniabahit.com
>>> cadena="000000000123000000000"
>>> print(cadena.strip("0"))
123
```

## Cadenas. Letras

`ascii_letters` es una cadena preiniciada que se usa como constante de cadena.

`ascii_letters` es básicamente una concatenación de constantes de cadena `ascii_lowercase` y `ascii_uppercase`. Además, el valor generado no depende de la configuración regional, por lo tanto, no cambia.

```
>>> from string import ascii_letters
>>> from string import ascii_lowercase
>>> from string import ascii_uppercase
>>> ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
>>> ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```



## Cadenas. Dígitos

---

De forma similar podemos obtener una cadena con todos los dígitos

```
>>> from string import digits
>>> digits
'0123456789'
```

## Cadenas. Sorted

---

La función `sorted()` toma uno iterable y lo ordena. La función devuelve una **lista** de todos los elementos ordenados.

```
palabra = 'fresa'
palabra_ordenada = sorted(palabra)
print(palabra_ordenada)
```

La salida del código es `['a', 'e', 'f', 'r', 's']`

¿Cómo puedo hacer que el resultado sea una cadena? Pues con la función `join`

```
palabra = 'fresa'
palabra_ordenada = sorted(palabra)
palabra_ordenada = ''.join(palabra_ordenada)
print(palabra_ordenada)
```

La salida del código es `aefrs`