Unidad 2.2

Cadenas de caracteres

1. Introducción

En este tema exploraremos uno de los tipos de datos más fundamentales y utilizados en la programación: las cadenas de caracteres. Las cadenas, también conocidas como strings, son secuencias de caracteres que nos permiten trabajar con texto.

En Python, las cadenas son inmutables, lo que significa que una vez creadas no se pueden modificar directamente, pero puedes crear nuevas cadenas a partir de las existentes. Este tipo de dato es muy versátil, y Python cuenta con una gran cantidad de herramientas y métodos para manipular y transformar las cadenas, desde buscar y reemplazar subcadenas, hasta formatear textos y convertir a diferentes tipos.

1.1. Creación

Las cadenas de texto se pueden crear de varias formas:

1. **Comillas Simples** ('...'):

Las cadenas definidas entre comillas simples son la forma más básica. Útil para cadenas cortas sin caracteres especiales como apóstrofes.

```
x = 'Hola, mundo'

print(x)
```

```
Hola, mundo
```

2. **Comillas Dobles** ("..."):

Funciona igual que las comillas simples y es útil si la cadena contiene comillas simples dentro.

```
x = "Hola, mundo"

print(x)
```

```
Hola, mundo
```

3. **Comillas Simples Triples** ('''...'''):

Se usan para cadenas multilínea o cadenas que incluyen tanto comillas simples como dobles. También permite escribir texto en varias líneas.

```
x = '''Esto es un texto
con una " doble y una ' simple
en tres lineas'''

print(x)
```

```
Esto es un texto
con una " doble y una ' simple
en tres lineas
```

4. **Comillas Dobles Triples** ("""..."""):

Funciona igual que las comillas simples triples y es otra opción para cadenas multilínea o cadenas largas.

```
x = """Otro ejemplo
de una cadena multilínea"""

print(x)
```

```
Otro ejemplo
de una cadena multilínea
```

5. **Cadenas sin Procesar o "Raw Strings"** (r'...' o r"..."):

Se usan cuando queremos que los caracteres especiales, como \n (nueva línea) o \t (tabulación), se interpreten literalmente, sin aplicar el escape.

```
x = r'C:\nueva\carpeta\archivo.txt'

print(x)
```

```
C:\nueva\carpeta\archivo.txt
```

6. **F-strings o Cadenas Formateadas** (f'...' o f"..."):

Introducidas en Python 3.6, permiten incluir expresiones y variables dentro de una cadena usando {}.

```
nombre = "Juan"
edad = 30
x = f'{nombre} tiene {edad} años'

print(x)
```

```
Juan tiene 30 años
```

7. **Cadenas con el Método str.format()**:

Otro método para formatear cadenas, útil para interpolar variables y expresiones en una cadena.

```
x = "El precio es de {} dólares".format(20)

print(x)
```

```
El precio es de 20 dólares
```

8. **Cadenas con el Constructor str()**:

Por otro lado, generalmente todos los tipos de datos nativos de python se pueden convertir a una cadena de texto aplicándole la función str().

1.2. Operaciones

Hay varias operaciones que podemos hacer con cadenas de caracteres:

1. Concatenación:

Utilizando el operador de suma (+) se pueden unir cadenas de caracteres.

```
x = "Uno,"
y = "Dos,"
z = "Tres."

print(x+y+z)
```

```
Uno,Dos,Tres.
```

2. Repetición:

Utilizando el operador de multiplicación (*) podemos repetir una cadena un número determinado de veces.

```
x = "Probando"

print(x*3)
```

```
ProbandoProbandoProbando
```

3. Indexación:

Se puede obtener un carácter concreto de una cadena accediendo por su index (posición en la cadena), teniendo en cuenta que la primera posición es la 0. Podemos acceder a las últimas posiciones utilizando números negativos (-1 último carácter, -2 penúltimo...)

```
x = "Probando"

print(x[0])
print(x[6])
print(x[-1])
```

```
P
d
o
```

4. Longitud:

Podemos saber la longitud de una cadena utilizando la función *len()*.

```
x = "12345"

print(len(x))
```

```
5
```

Hay que tener cuidado ya que esta función nos da el número exacto de caracteres, es decir, que si queremos indexar (teniendo en cuenta que se empieza a contar por 0) por ejemplo la última posición de una cadena, tenemos que acceder a la posición len(x)-1.

5. Recorrer cadena:

Las cadenas de caracteres se pueden recorrer elemento a elemento utilizando
"*for [nombre de variable] in [nombre de la cadena a recorrer]:*"

```
cadena = "12345"

for x in cadena:
    print(x + ".")
```

```
1.
2.
3.
4.
5.
```

```
cadena = "12345"

for caracter in cadena:
    print(caracter + ".")
```

```
1.
2.
3.
4.
5.
```

El nombre de variable puede ser cualquiera que queramos, será el que se usará dentro del for para referirse a cada caracter que recorramos.

6. Pertenencia:
Podemos comprobar si un caracter forma parte de una cadena utilizando *in*, esto nos devolverá un booleano indicando si la pertenencia es verdadera o falsa.

```
x = "Hola"
pertenenciaTrue = "a" in x
pertenenciaFalse = "c" in x
pertenenciaFalse2 = "a" not in x

print(pertenenciaTrue, pertenenciaFalse, pertenenciaFalse2)
```

```
True False False
```

7. Comparación:
Podemos comparar diferentes cadenas utilizando los operadores lógicos “>”, “<”, “=”. Python compara caracter a caracter entre las diferentes cadenas, siendo mayor las que van despúes en la tabla unicode. (Caracteres especiales < Números < Mayúsculas < Minúsculas).

Resumen de resultados y explicaciones

Expresión	Resultado	Explicación
"apple" == "apple"	True	Las cadenas son exactamente iguales.
"Apple" == "apple"	False	Distinción entre mayúsculas y minúsculas.
"banana" < "grape"	True	'b' viene antes que 'g' en el orden lexicográfico.
"orange" > "apple"	True	'o' viene después que 'a'.

8. Slice:
Podemos hacer “cortes” de la cadena de caracteres para crear subcadenas. Esto se puede hacer de varias formas:
-cadena[inicio:fin] # Desde la posición *inicio* hasta la posición *fin*.
-cadena[inicio:] # Toda la cadena empezando desde la posición *inicio*.
-cadena[:fin] # Toda la cadena hasta la posición *fin*.
-cadena[:] # Toda la cadena
-cadena[inicio:fin:salto] # Desde la posición *inicio* hasta la *fin* pero haciendo saltos de *salto* caracteres.

La posición *inicio* se incluye pero la posición *fin* no está incluida, aquí vemos varios ejemplos.

```
x = "Esto es una cadena de ejemplo"

print(x[12:18])
print(x[12:])
print(x[:18])
print(x[:])
print(x[12:18:2])
```

```
cadena
cadena de ejemplo
Esto es una cadena
Esto es una cadena de ejemplo
cdn
```

1.3. Métodos para cadenas

Hay diferentes métodos que podemos utilizar con las cadenas de texto, formateo, busqueda, validación… Hay que tener en cuenta que las cadenas de texto son inmutables, es decir, que cuando aplicamos métodos que "cambian" la cadena, realmente se está creando una cadena nueva. Si intentamos cambiar algún caracter de la cadena, Python arroja un error.

```
x = "Hola"
x[0] = "h"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 2
      1 x = "Hola"
----> 2 x[0] = "h"

TypeError: 'str' object does not support item assignment
```

Vamos a ver algunos de los métodos más utilizados pero hay muchos. Si queréis ver más podéis consultar la documentación de Python en la web.

1.3.1. Métodos de formateo

-.upper() y .lower(): Devuelve la cadena escrita en mayúsculas o minúsculas respectivamente.

-.capitalize(): devuelve la cadena con la primera letra en mayúsculas.

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

-title(): devuelve una cadena con la primera letra de cada palabra en mayúsculas.

1.3.2. Métodos de búsqueda

-.count(caracter,[inicio],[fin]): Pasándole un caracter, busca cuantas concurrencias de él hay en la cadena. Si además le pasamos una posición de inicio, buscará a partir de esa posición (incluida) y si le pasamos una de fin, parará de buscar en esa posición (sin incluirla).

-find(subcadena): devuelve la posición de la subcadena que hayamos pasado como parámetro. Si no se encuentra, devuelve -1

```
x = "Hola mundo"

print(x.find("mun"))
print(x.find("fin"))
```

```
5
-1
```

1.3.3. Métodos de validación

-.startswith(subcadena, [inicio], [fin]): nos devuelve un valor lógico indicando si la cadena empieza con la subcadena pasada como parámetro. Si además le pasamos un posición, nos dirá si la subcadena comienza en esa posición.

-.endswith(subcadena, [inicio],[fin]): nos devuelve un valor lógico indicando si la cadena termina con la subcadena pasada como parámetro. Además podemos usar los parámetros opcionales inicio y fin para comprobar si las posiciones aportadas acaban con la subcadena pasada como parámetro. Si pasamos solo un parámetro opcional se tomará como inicio.

-.isalpha(): devuelve un valor lógico indicando si todos los caracteres de la cadena son alfanuméricos o no.

1.3.4. Métodos de sustitución

-.replace(subcadena, remplazo): el método replace devuelve una nueva cadena en la que se cambia la subcadena indicada por el remplazo indicado. En caso de no existir la subcadena, la cadena quedaría igual.

```
x = "Hola mondo"

print(x.replace("mondo", "mundo"))
print(x.replace("Holo", "Nada"))
```

```
Hola mundo
Hola mondo
```

-.strip([subcadena]): devuelve una cadena de la que se ha quitado del principio y el final (si existiesen) los espacios en blanco. Se le puede pasar una subcadena como parámetro y se quitará esta del principio y final en lugar de los espacios en blanco. Si no existiese, la cadena quedaría igual.

1.3.5. Métodos de unión y división

-.split([subcadena]): devuelve una **lista** de los elementos separados por la subcadena indicada.

```
x = "1coma2coma3coma4"

print(x.split("coma"))
```

```
['1', '2', '3', '4']
```

-.splitlines(): separa las líneas de una cadena (separadas por “\n”) y devuelve una lista.