CADENAS DE CARACTERES

Algoritmos y estructuras de datos I

CADENA DE CARACTERES

```
print("Esto es una cadena en 'python'")
print('Esto es una cadena en "python"')

print('''Esto es una cadena
en 'python' con
varias líneas''')

print("""Esto es una cadena
en "python" con
varias líneas""")

print("Esto es una cadena con comillas simples: \'
y comillas dobles: \" dentro de ella")

print("Esto es una cadena\ncon una nueva línea")
```

Las cadenas de caracteres (o *strings*) son un tipo de datos compuestos por secuencias de caracteres que representan texto. Estas cadenas de texto son de tipo *str* y se delimitan mediante el uso de comillas simples o dobles.

En el caso que queramos usar comillas (o un apóstrofo) dentro de una cadena tenemos distintas opciones. La más simple es encerrar nuestra cadena mediante un tipo de comillas (simples o dobles) y usar el otro tipo dentro de la cadena. Otra opción es usar en todo momento el mismo tipo de comillas, pero usando la barra invertida (\) como carácter de escape en las comillas del interior de la cadena para indicar que esos caracteres forman parte de la cadena.

CADENAS DE ESCAPE

Una cadena de escape es una secuencia de caracteres que comienza con una barra invertida y que representa un carácter especial en una cadena de texto. Estos caracteres especiales se utilizan para representar particularidades dentro de una cadena, como saltos de línea, tabulaciones y comillas dentro de una cadena de texto.

Por ejemplo, la cadena de escape \n representa un salto de línea, mientras que \t representa una tabulación. La cadena de escape \\ se utiliza para representar una barra invertida literal dentro de una cadena de texto, sin que sea confundida por una cadena de escape.

Secuencia de escape	Significado
\\	barra invertida (\)
\'	comilla simple (')
\"	comilla doble (")
\n	salto de línea
\t	tabulación

```
print("Números binários:\n\t0\n\t1")
```

```
Números binarios:
        0
        1
```

FORMATEO DE CADENAS

```
numero = 11
cadena = "El número es: " + str(numero) + "."
print(cadena) #El número es: 11
```

Si queremos declarar una cadena que contenga variables en su interior, como números o incluso otras cadenas, una forma de hacerlo sería concatenando la cadena que queremos con otra usando el operador +.

Por medio de la función *str()* podemos convertir al tipo de dato *string* lo que se pasa como parámetro.

INDEXACIÓN

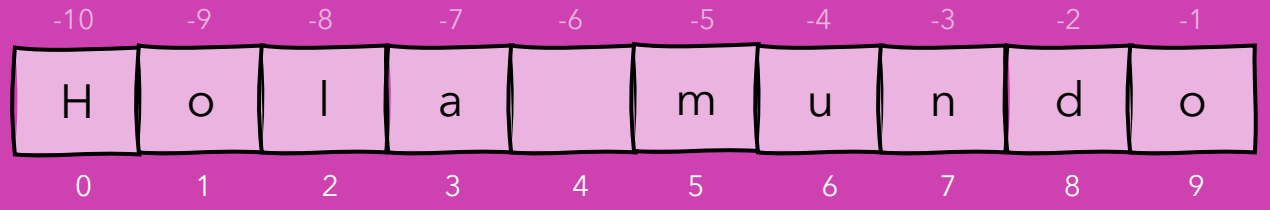
Cada uno de los caracteres de una cadena (incluidos los espacios) tiene asignado un índice. Este índice nos permite seleccionar su carácter asociado haciendo referencia a él entre corchetes ([]) en el nombre de la variable que almacena la cadena.

Si consideremos el orden de izquierda a derecha, el índice comienza en 0 para el primer carácter, etc. También se puede considerar el orden de derecha a izquierda, en cuyo caso al último carácter le corresponde el índice -1, al penúltimo -2 y así sucesivamente.

```
cadena = "Hola mundo"

print("Primer caracter:", cadena[0]) #H
print("Segundo caracter:", cadena[1]) #o
print("Tercer caracter:", cadena[2]) #l
print("Cuarto caracter:", cadena[3]) #a
print("Quinto caracter:", cadena[4]) #

print("Último caracter:", cadena[-1]) #o
print("Penúltimo caracter:", cadena[-2]) #d
print("Antepenúltimo caracter:", cadena[-3]) #n
```



INDEXACIÓN

Además, las cadenas presentan la propiedad de inmutabilidad. Esto significa que una vez han sido creadas no pueden modificarse. En efecto, si intentamos modificar una cadena el intérprete nos indica que a ésta no se pueden asignar elementos.

En caso de que quisiéramos realizar esta operación, obtendríamos el error *TypeError: 'str' object does not support item assignment*.

```
cadena = "Hola mundo"
cadena[5] = "M"
```

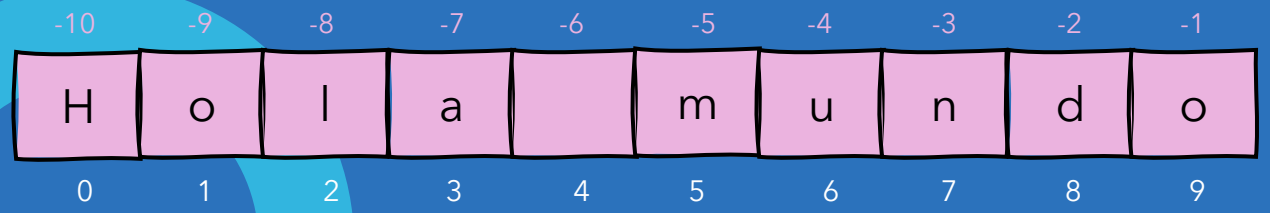
```
Traceback (most recent call last):
  File "caracteres.py", line 2, in
<module>
    cadena[5] = "M"
    ~~~~~~^^^
TypeError: 'str' object does not
support item assignment
```

REBANADO

```
cadena = "Hola mundo“

print("Desde 5 hasta 10:", cadena[5:10]) #mundo
print("Del 1 al 10 paso 2:", cadena[1:10:2]) #oamno
print("Inicio hasta 4:", cadena[:4]) #Hola
print("Desde 5 al final:", cadena[5:]) #mundo
print("Al revés:", cadena[::-1]) #odnum aloH

print("Desde -2 hasta -4:", cadena[-2:-4:-1]) #dn
print("Desde -3 hasta -5:", cadena[-3:-5:-1]) #nu
```



Otra operación que podemos realizar a una cadena es seleccionar solamente una parte de ella. Para ello se usa la notación *[inicio:fin:paso]* que hemos utilizado en su implementación en listas.

Recordemos que, si omitimos las posiciones de inicio o fin, nos referimos respectivamente a la primera y última posición de la cadena. Además, omitir el paso es equivalente a indicar que el paso es 1.

OPERACIONES

Una operación que podemos realizar es la concatenación que consiste en unir distintas cadenas mediante el uso del signo más (+):

```
cadena1 = "Hola"
cadena2 = "Mundo"
cadena3 = cadena1 + " " + cadena2
print(cadena3) #Hola Mundo
```

También podemos concatenar con el signo de multiplicación (*), que en este caso significa adjuntarle un determinado número de copias a la cadena:

```
cadena = "Hola mundo"
print(cadena*3) #Hola mundoHola mundoHola mundo
```

La longitud de una cadena viene determinada por su número de caracteres, y se puede consultar con la función *len()*:

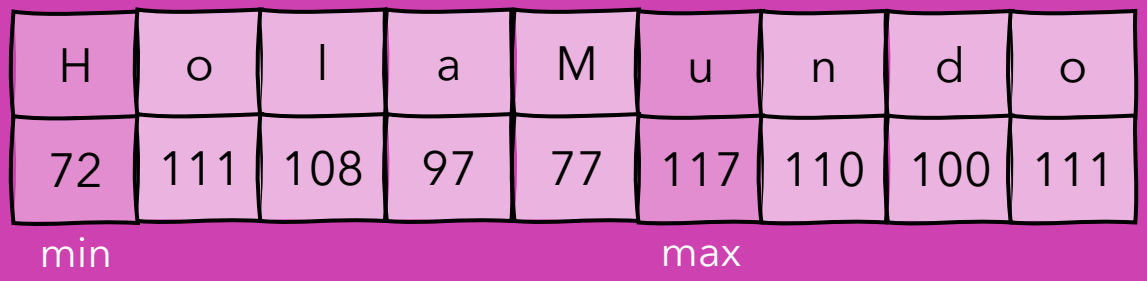
```
cadena = "Hola mundo"
print(len(cadena)) #10
```

El método *count()* permite contar las veces que otra cadena se encuentra dentro de la primera. Permite también dos parámetros opcionales que indican donde empezar y acabar de buscar.

```
cadena = "Hola Mundo"
print(cadena.count("o")) #2
```

FUNCIONES MIN Y MAX

La función *min()* en Python devuelve el carácter que precede a todos los demás en el orden lexicográfico de los caracteres Unicode presentes en la cadena. De manera similar, la función *max()* devuelve el carácter que sigue a todos los demás en dicho orden.




```
cadena = "HolaMundo"
print(min(cadena)) #H
print(max(cadena)) #u
```

En este ejemplo, la función *min()* devuelve el carácter 'H' porque en el orden lexicográfico de los caracteres Unicode, 'H' precede a 'a'. Mientras que *max()* devuelve 'u', que sigue a todos los demás caracteres en dicho orden.

FUNCIONES CHR Y ORD

Las funciones *chr()* y *ord()* se utilizan para convertir entre valores numéricos y caracteres *ASCII estándar*.

La función *chr()* toma un número entero como argumento y devuelve el carácter correspondiente a ese número. Por otro lado, la función *ord()* toma un carácter como argumento y devuelve el valor numérico correspondiente a ese carácter:

```
print(chr(65)) #A
print(ord('A')) #65
```

```
for i in range(65, 75):
    print(i, chr(i))
```

```
65 A
66 B
67 C
68 D
69 E
70 F
71 G
72 H
73 I
74 J
```

OPERADOR IN

Para determinar si una cadena está contenida dentro de otra, podemos utilizar el operador *in*. Este operador verifica si la subcadena se encuentra en la cadena principal. Si la está presente, el operador devuelve *True*; de lo contrario, devuelve *False*.

```
cadena = "Hola mundo"

if ("Hola" in cadena):
    print("Hola existe en la cadena")
else:
    print("Hola no existe en la cadena")
```

Es posible realizar una iteración por cada uno de los caracteres en una cadena utilizando un bucle *for*. Al hacerlo, se puede acceder y procesar individualmente cada carácter de la cadena.

```
cadena = "Hola mundo"

for caracter in cadena:
    print(caracter)
```

En este ejemplo, el bucle *for* recorre cada carácter de la cadena "Hola mundo" uno por uno. Durante cada iteración del bucle, la variable caracter toma el valor del siguiente carácter en la cadena, permitiendo así imprimir cada carácter por separado.

TRANSFORMACIÓN DE CARACTERES

Existen distintos métodos que nos permiten variar la capitalización de los caracteres que forman una cadena, remover espacios en blanco o separar palabras. A continuación, conoceremos algunos de ellos.

LOWER - Minúsculas

Retorna una copia de la cadena convirtiendo las letras mayúsculas a minúsculas.

```
print(cadena.lower()) #hola mundo
```

SWAPCASE - Inversión de mayúsculas y minúsculas

Devuelve los caracteres alfabéticos con mayúsculas en minúsculas y viceversa.

```
print(cadena.swapcase()) #hOLA mUNDO
```

UPPER - Mayúsculas

Retorna una copia de la cadena convirtiendo las letras minúsculas a mayúsculas. El hecho de que devuelva una copia significa que la variable original no se ve afectada por la operación.

```
cadena = "Hola mundo"
print(cadena.upper()) #HOLA MUNDO
```

TRANSFORMACIÓN DE CARACTERES

TITLE - Capitalizado

Devuelve una copia de la cadena usando donde la primera letra de cada palabra se pone en mayúscula.

```
print(cadena.title()) #Hola Mundo
```

REPLACE - Reemplazo de caracteres

Retorna una copia de la cadena a la cual se le ha cambiado la primera ocurrencia del carácter especificado en el primer parámetro por el especificado en el segundo.

```
print(cadena.replace("Hola", "Chau")) #Chau Mundo
```

JOIN – Unión de una lista

Se utiliza para unir una lista de cadenas de texto en una sola cadena, utilizando otro *string* como separador entre ellas.

```
nombres = ['Oliver', 'Irina', 'Gabriela']
cadenaUnida = ', '.join(nombres)
print(cadenaUnida) #Oliver, Irina, Gabriela
```

LSTRIP, RSTRIP Y STRIP - Quitado de espacios al comienzo o fin de la cadena

lstrip() devuelve una copia de la cadena a la cual se le han eliminado los espacios del principio Y *rstrip()* los espacios del final. El método *strip()* es la combinatoria de ambos.

```
cadena = "          Hola mundo          "
print(cadena.lstrip()) #'Hola mundo          '
print(cadena.rstrip()) #'          Hola mundo'
print(cadena.strip()) #'Hola mundo'
```

TRANSFORMACIÓN DE CARACTERES

ISALNUM()

El método devuelve *True* si la cadena está formada únicamente por caracteres alfanuméricos, es decir, si son letras (mayúsculas o minúsculas) o números; *False* en caso contrario:

```
usuario = "oliverperez87"
nombre = "Oliver Pérez"

print(usuario.isalnum()) #True
print(nombre.isalnum()) #False
```

SPLIT – División de cadena de caracteres

Devuelve una lista de las palabras de la cadena separadas acorde al parámetro que reciba. Si este parámetro no se especifica o es *None*, la cadena se separa teniendo en cuenta los espacios en blanco.

```
numeros = input("Ingresa números separados por espacios: ")
numeros = [int(numero) for numero in numeros.split()]

cadena = "Hola mundo; chau mundo"
print(cadena.split()) #['Hola', 'mundo;', 'chau', 'mundo']
print(cadena.split(";")) #['Hola mundo', ' chau mundo']
```

TRANSFORMACIÓN DE CARACTERES

ISDIGIT()

Nos permite determinar si una cadena de caracteres contiene solamente dígitos numéricos o no. Devuelve *True* si todos los caracteres de la cadena son dígitos numéricos, y *False* en caso contrario:

```
cadena1 = "12345"
cadena2 = "12a34"

print(cadena1.isdigit()) #True
print(cadena2.isdigit()) #False
```

ISALPHA()

La función *isalpha()* devuelve *True* si todos los caracteres de una cadena son letras (mayúsculas o minúsculas) y no contiene ningún otro tipo de caracteres, como números o signos de puntuación:

```
cadena1 = "Hola"
cadena2 = "Hola123"
cadena3 = "¡Hola!"

print(cadena1.isalpha()) #True
print(cadena2.isalpha()) #False
print(cadena3.isalpha()) #False
```

```
nombre = "Oliver"
edad = 6

print("Nombre: " + nombre)
print("Edad: " + str(edad))
print("Nombre y edad: " + nombre + ", " + str(edad))
```

OPERADOR + Y EL USO DE VARIABLES

Una de las formas más básicas de realizar este formateo es mediante el uso del operador +. Este operador, que normalmente se utiliza para la concatenación de cadenas, también puede ser empleado para combinar cadenas con variables.

Cuando se utilizan variables en una cadena de texto mediante el operador +, es importante recordar que las variables deben ser convertidas explícitamente a cadenas de texto si son de otro tipo, como números. Esto se logra utilizando la función *str()*, que convierte el valor de la variable en una representación de cadena.

FORMATEO

Una alternativa a la concatenación de cadenas es el uso del método *format()*. Este método devuelve una copia de la cadena a la que se le han sustituido las posiciones que contienen llaves ({}) por los argumentos del método. Esta sustitución se realiza por defecto en el mismo orden de los argumentos.

```
#Nombre: Oliver - Apellido: Pérez - DNI 59.111.333
print("Nombre: {} - Apellido: {} - DNI {}".format("Oliver", "Pérez", "59.111.333"))
```

También podemos referenciar los argumentos del método por su posición (siendo el índice del primero 0) o mediante un nombre.

```
#Nombre: Oliver - Apellido: Pérez - DNI 59.111.333
print("Nombre: {1} - Apellido: {2} - DNI {0}".format("59.111.333", "Oliver", "Pérez"))
```

También nos permite especificar el número de decimales que se muestran en números de coma flotante.

```
pi = 3.141592653589793
print("El numero pi es {}".format(pi)) #El numero pi es 3.141592653589793
print("El numero pi es {:.2f}".format(pi)) #El numero pi es 3.14
```

F-STRING

El método *format()* puede producir código un poco engorroso de leer cuando tratamos con *strings* largos que contienen múltiples parámetros.

Es por ello que a partir de Python 3.6 se añadieron los *f-strings*, que son cadenas de texto con una *f* al inicio y expresiones entre llaves que se sustituyen por sus valores, tal y como se muestra en el ejemplo.

También podríamos hacer operaciones dentro de la creación de la cadena. Incluso, de ser necesario, podríamos invocar a la función dentro de las llaves.

Para acortar el número de decimales en un valor numérico, se puede utilizar la sintaxis *:.nf*, donde n es el número de decimales que se desea mostrar

```
def cuadrado(numero):
    return numero ** 2


nombre = "Oliver"
apellido = "Pérez"

#Nombre: Oliver - Apellido: Pérez
print(f"Nombre: {nombre} - Apellido: {apellido}")

numero = 13

#Cuadrado de 13 = 169
print(f"Cuadrado de {numero} = {numero**2}")
print(f"Cuadrado de {numero} = {cuadrado(numero)}")
```

CAMPO DE FORMATO

Proporciona un conjunto de directrices para formatear valores de diferentes tipos, como números enteros, números de punto flotante, entre otros.

El campo de formato se compone de varios componentes que definen aspectos específicos del formato de presentación de un valor.

Los dos puntos *(:)* indican el inicio del campo de formato; seguido de las opciones que se aplican al valor, como la precisión decimal y el tipo de valor que se va a formatear, como *d* para enteros, *f* para números de punto flotante, entre otros.

```
pi = 3.141592653589793

print("Punto flotante: {:.2f}".format(pi)) #3.14
print("Notación científica: {:.2e}".format(pi)) #3.14e+00

print(f"Punto flotante: {pi:.2f}") #3.14
print(f"Notación científica: {pi:.2e}") #3.14e+00

numero = 11

print(f"Decimal: {numero:d}") #11
print(f"Binario: {numero:b}") #1011
print(f"Octal: {numero:o}") #13
print(f"Hexadecimal (minúsculas): {numero:x}") #b
print(f"Hexadecimal (mayúsculas): {numero:X}") #B
```

CAMPO DE FORMATO

Para números enteros

d: Formatea el número como un entero.
b: Formatea el número en binario.
o: Formatea el número en octal.
x: Formatea el número en hexadecimal (en minúsculas).
X: Formatea el número en hexadecimal (en mayúsculas).

Para números de punto flotante

f: Formatea el número como un número de punto flotante.
e: Formatea el número en notación científica (exponencial).
g: Formatea el número usando la notación más corta entre e y f.
%: Formatea el número como un porcentaje.

```
pi = 3.141592653589793

print("Punto flotante: {:.2f}".format(pi)) #3.14
print("Notación científica: {:.2e}".format(pi)) #3.14e+00

print(f"Punto flotante: {pi:.2f}") #3.14
print(f"Notación científica: {pi:.2e}") #3.14e+00

numero = 11

print(f"Decimal: {numero:d}") #11
print(f"Binario: {numero:b}") #1011
print(f"Octal: {numero:o}") #13
print(f"Hexadecimal (minúsculas): {numero:x}") #b
print(f"Hexadecimal (mayúsculas): {numero:X}") #B
```

FORMATEO %

El carácter modulo % es un operador integrado en Python y es conocido como el operador de interpolación. Se deberá ingresar el caracter % seguido por el tipo que se necesita formatear o convertir. El operador % entonces substituye la frase *«%tipodato»* con cero o más elementos del tipo de datos especificado. Con esta sintaxis hay que determinar el tipo del objeto:

%s → str, cadena de carácter.
%d → int, enteros.
%f → float, coma flotante.

```
texto = "La raíz cuadrada de 2"
resultado = 2**0.5

#La raíz cuadrada de 2 es 1.414214
print("%s es %f" % (texto, resultado))
```

También aquí se puede controlar el formato de salida. Por ejemplo, para obtener el valor con 2 dígitos después de la coma:

```
#La raíz cuadrada de 2 es 1. 41
print("%s es %.2f" % (texto, resultado))
```

DOCSTRINGS

En Python todos los objetos cuentan con una variable especial llamada *__doc__*, gracias a la cual puede describir para qué sirven los objetos y cómo se usan. Estas variables reciben el nombre de *docstrings*, o cadenas de documentación.

Python implementa un sistema muy sencillo para establecer el valor de las *docstrings* en las funciones, únicamente se debe crear un comentario en la primera línea después de la declaración de estas. Como se observa en el ejemplo, para consultar la documentación de la función *cubo()* se debe utilizar la función integrada *help()* y pasarle el argumento el nombre de la función.

```
def cubo(valor):
    """
    Calcula el cubo de un número.

    Parámetros:
    valor (int o float): El número al cual se le calculará el cubo.

    Retorna:
    int o float: El resultado de elevar el número al cubo.
    """
    return valor ** 3

help(cubo)
```

```
Help on function cubo in module __main__:

cubo(valor)
    Calcula el cubo de un número.
    …
```

Lic. Yaps David P.

AUTOEVALUACIÓN

La autoevaluación es crucial para identificar mejoras y fortalezas de forma objetiva, facilitando la fijación de metas y la toma de acciones para alcanzarlas.