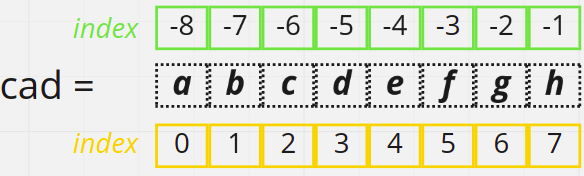
Índices de cadenas

Los índices nos permiten posicionarnos en un carácter específico de una cadena.

Representan un número [índice], que empezando por el 0 indica el carácter de la primera posición, y así sucesivamente, por ejemplo en la cadena= “**abcdefgh**“:



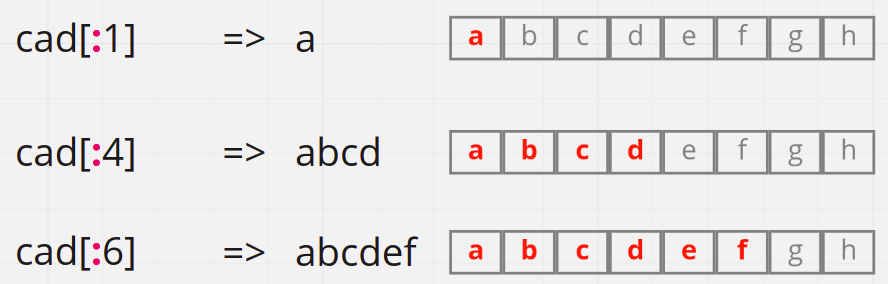
Nota: Las cadenas son inmutables

Para hacer slicing se usar el operador (dos puntos), ejemplo:

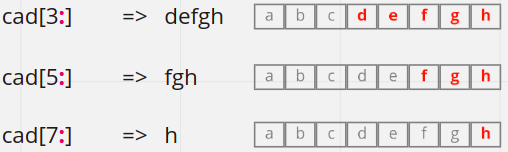
* [n:m] devuelve la parte de la cadena desde el “n-ésimo” carácter hasta el “m-ésimo”
* Si omite el primer índice (antes de los dos puntos), la porción comienza al principio de la cadena.
* Si omite el segundo índice, la porción llega al final de la cadena.

Pero vayamos al ejemplo en concreto:

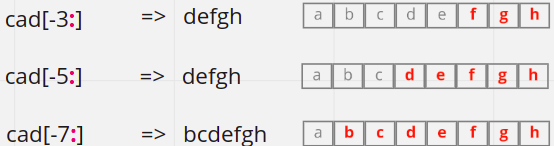
* Entonces para obtener **solo los primeros X** caracteres de la cadena – (Incluye):



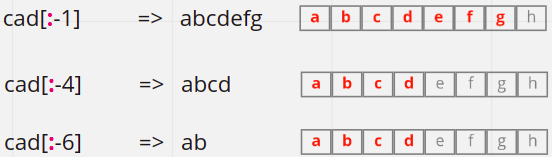
* Para **excluir los X primeros** elementos.



* Para **incluir solo los X últimos** caracteres (forma inclusiva)



* Para **excluir los X últimos** caracteres (forma excluyente)



**Métodos de cadenas**

El método count() retorna el número de veces que se repite un conjunto de caracteres especificado.

1. >>> s = "Hola mundo"
2. >>> s.count("Hola")
3. 1

Los métodos find() e index() retornan la ubicación (comenzando desde el cero) en la que se encuentra el argumento indicado.

1. >>> s.find("mundo")
2. 5
3. >>> s.index("mundo")
4. 5

Difieren en que esta última lanza ValueError cuando el argumento no es encontrado, mientras que aquélla retorna -1.

1. >>> s.find("world")
2. -1
3. >>> s.index("world")
4. Traceback (most recent call last):
5. File "<stdin>", line 1, in <module>
6. ValueError: substring not found

En ambos métodos la búsqueda ocurre de izquierda a derecha. Para buscar un conjunto de caracteres desde el final, utilícense del mismo modo rfind() y rindex().

1. >>> s = "C:/python36/python.exe"
2. >>> s.find("/") # Retorna la primera ocurrencia.
3. 2
4. >>> s.rfind("/") # Retorna la última.
5. 11

startswith() y endswith() indican si la cadena en cuestión comienza o termina con el conjunto de caracteres pasados como argumento, y retornan True o False en función de ello.

1. >>> s = "Hola mundo"
2. >>> s.startswith("Hola")
3. True
4. >>> s.endswith("mundo")
5. True
6. >>> s.endswith("world")
7. False

Ambos métodos son preferidos ante la opción de emplear *slicing*.

1. # Se prefiere startswith().
2. >>> s[:4] == "Hola"
3. True

Los métodos isdigit(), isnumeric() e isdecimal() determinan si todos los caracteres de la cadena son dígitos, números o números decimales.

1. >>> "1234".isnumeric()
2. True
3. >>> "1234".isdecimal()
4. True
5. >>> "abc123".isdigit()
6. False

Si bien estas definiciones resultam *a priori* similares, no lo son. La primera, isdigit(), considera caracteres que pueden formar números, incluidos aquellos correspondientes a lenguas orientales. isnumeric() es más amplia, pues incluye también caracteres de connotación numérica que no necesariamente son dígitos (por ejemplo, una fracción). La última, isdecimal(), es la más restrictiva al tener en cuenta únicamente números decimales; esto es, formados por dígitos del 0 al 9.

Las siguientes seis funciones de análisis son bastante explicativas por sí mismas.

1. # Determina si todos los caracteres son alfanuméricos.
2. >>> "abc123".isalnum()
3. True
4. # Determina si todos los caracteres son alfabéticos.
5. >>> "abcdef".isalpha()
6. True
7. >>> "abc123".isalpha()
8. False
9. # Determina si todas las letras son minúsculas.
10. >>> "abcdef".islower()
11. True
12. # Mayúsculas.
13. >>> "ABCDEF".isupper()
14. True
15. # Determina si la cadena contiene todos caracteres imprimibles.
16. >>> "Hola \t mundo!".isprintable()
17. False
18. # Determina si la cadena contiene solo espacios.
19. >>> "Hola mundo".isspace()
20. False
21. >>> " ".isspace()
22. True

**Métodos de transformación**

Recuérdese que las cadenas son inmutables; por ende, todos los métodos a continuación no actúan sobre el objeto original sino que retornan uno nuevo.

capitalize() retorna la cadena con su primera letra en mayúscula.

1. >>> "hola mundo".capitalize()
2. 'Hola mundo'

encode() codifica la cadena con el mapa de caracteres especificado y retorna una instancia del tipo bytes.

1. >>> "Hola mundo".encode("utf-8")
2. b'Hola mundo'

Los métodos center(), ljust() y rjust() alinean una cadena en el centro, la izquierda o la derecha respectivamente. Toman un argumento, la cantidad de caracteres respecto de la cual se producirá la alineación.

1. >>> "Hola".center(10)
2. ' Hola '
3. >>> "Hola".ljust(10)
4. 'Hola '
5. >>> "Hola".rjust(10)
6. ' Hola'

Estos métodos son especialmente útiles al imprimir en forma de tabla para que ésta se mantenga alineada. Un segundo argumento indica con qué carácter se deben llenar los espacios vacíos (por defecto un espacio en blanco).

1. >>> "Hola".center(10, "\*")
2. '\*\*\*Hola\*\*\*'

lower() y upper() retornan una copia de la cadena con todas sus letras en minúsculas o mayúsculas según corresponda.

1. >>> "Hola Mundo!".lower()
2. 'hola mundo!'
3. >>> "Hola Mundo!".upper()
4. 'HOLA MUNDO!'

swapcase(), por su parte, cambia las mayúsculas por minúsculas y viceversa.

1. >>> "Hola Mundo!".swapcase()
2. 'hOLA mUNDO!'

Las funciones strip(), lstrip() y rstrip() remueven los espacios en blanco que preceden y/o suceden a la cadena.

1. >>> s = " Hola mundo! "
2. >>> s.strip()
3. 'Hola mundo!'
4. # Remueve los de la derecha.
5. >>> s.rstrip()
6. ' Hola mundo!'
7. # Remueve los de la izquierda.
8. >>> s.lstrip()
9. 'Hola mundo! '

Por último, el método replace() ─ampliamente utilizado─ reemplaza una cadena por otra.

1. >>> s = "Hola mundo"
2. >>> s.replace("mundo", "world")
3. 'Hola world'

**Métodos de separación y unión**

El método de división de una cadena según un carácter separador más empleado es split(), cuyo separador por defecto son espacios en blanco y saltos de línea.

1. >>> "Hola mundo!\nHello world!".split()
2. ['Hola', 'mundo!', 'Hello', 'world!']

El separador puede indicarse como argumento.

1. >>> "Hola mundo!\nHello world!".split(" ")
2. ['Hola', 'mundo!\nHello', 'world!']

O bien, para separar únicamente según saltos de línea splitlines():

1. # Equivalente a split("\n").
2. >>> "Hola mundo!\n Hello world!".splitlines()
3. ['Hola mundo!', 'Hello world!']

Un segundo argumento en split() indica cuál es el máximo de divisiones que puede tener lugar (-1 por defecto para representar una cantidad ilimitada).

1. >>> "Hola mundo hello world".split(" ", 2)
2. ['Hola', 'mundo', 'hello world']

Un segundo método de separación es partition(), que retorna una tupla de tres elementos: el bloque de caracteres anterior a la primera ocurrencia del separador, el separador mismo, y el bloque posterior.

1. >>> s = "Hola mundo. Hello world!"
2. >>> s.partition(" ")
3. ('Hola', ' ', 'mundo. Hello world!')

rpartition() opera de forma similar, pero realizando la búsqueda de derecha a izquierda.

1. >>> s.rpartition(" ")
2. ('Hola mundo. Hello', ' ', 'world!')

Por último, el método join() ─sumamente útil─, que debe ser llamado desde una cadena que actúa como separador para unir dentro de una misma cadena resultante los elementos de una lista.

1. >>> " ".join(["Hola", "mundo"])
2. 'Hola mundo'
3. >>> ", ".join(["C", "C++", "Python", "Java"])
4. 'C, C++, Python, Java'

Como puede observarse, split() y join() son exactamente contrarios.

1. >>> sep = " "
2. >>> sep.join("Hola mundo!".split(sep))
3. 'Hola mundo!'