Métodos de las cadenas

upper()

Devuelve la cadena con todos sus caracteres a mayúscula:

```
"Hola Mundo".upper()
'HOLA MUNDO'
```

lower()

Devuelve la cadena con todos sus caracteres a minúscula:

```
"Hola Mundo".lower()
'hola mundo'
```

capitalize()

Devuelve la cadena con su primer carácter en mayúscula:

```
"hola mundo".capitalize()
'Hola mundo'
```

title()

Devuelve la cadena con el primer carácter de cada palabra en mayúscula:

```
"hola mundo".title()
'Hola Mundo'
```

count()

Devuelve una cuenta de las veces que aparece una subcadena en la cadena:

```
"Hola mundo".count('mundo')
1
```

find()

Devuelve el índice en el que aparece la subcadena (-1 si no aparece):

```
"Hola mundo".find('mundo')
5
"Hola mundo".find('mundoz')
-1
```

rfind()

Devuelve el índice en el que aparece la subcadena, empezando por el final:

```
"Hola mundo mundo mundo".rfind('mundo')
17
```

isdigit()

Devuelve True si la cadena es todo números (False en caso contrario):

```
c = "100"
c.isdigit()
True
```

isalnum()
Devuelve True si la cadena es todo números o carácteres alfabéticos:
```
c = "ABC10034po"
c.isalnum()
True
```

isalpha()
Devuelve True si la cadena es todo carácteres alfabéticos:
```
c = "ABC10034po"
c.isalpha()
False
"Holamundo".isalpha()
True
```

islower()
Devuelve True si la cadena es todo minúsculas:
```
"Hola mundo".islower()
False
```

isupper()
Devuelve True si la cadena es todo mayúsculas:
```
"Hola mundo".isupper()
False
```

istitle()
Devuelve True si la primera letra de cada palabra es mayúscula:
```
"Hola Mundo".istitle()
True
```

isspace()
Devuelve True si la cadena es todo espacios:
```
"  -  ".isspace()
False
```

startswith()
Devuelve True si la cadena empieza con una subcadena:
```
"Hola mundo".startswith("Mola")
False
```

endswith()
Devuelve True si la cadena acaba con una subcadena:
```
"Hola mundo".endswith('mundo')
True
```

split()

Separa la cadena en subcadenas a partir de sus espacios y devuelve una lista:

```
"Hola mundo mundo".split()[0]
 'Hola'
```

Podemos indicar el carácter a partir del que se separa:

```
"Hola,mundo,mundo,otra,palabra".split(',')
['Hola', 'mundo', 'mundo', 'otra', 'palabra']
```

join()

Une todos los caracteres de una cadena utilizando un caracter de unión:

```
",".join("Hola mundo")
'H,o,l,a, ,m,u,n,d,o'
" ".join("Hola")
'H o l a'
```

strip()

Borra todos los espacios por delante y detrás de una cadena y la devuelve:

```
"   Hola mundo     ".strip()
'Hola mundo'
```

Podemos indicar el carácter a borrar:

```
"-----Hola mundo---".strip('-')
'Hola mundo'
```

replace()

Reemplaza una subcadena de una cadena por otra y la devuelve:

```
"Hola mundo".replace('o','0')
'H0la mund0'
```

Podemos indicar un límite de veces a reemplazar:

```
"Hola mundo mundo mundo mundo mundo".replace(' mundo','',4)
'Hola mundo'
```

Métodos de las listas

append()

Añade un ítem al final de la lista:

```
lista = [1,2,3,4,5]
lista.append(6)
lista
[1, 2, 3, 4, 5, 6]
```

clear()

Vacía todos los ítems de una lista:

```
lista.clear()
lista
[]
```

extend()

Une una lista a otra:

```
l1 = [1,2,3]
l2 = [4,5,6]
l1.extend(l2)
l1
[1, 2, 3, 4, 5, 6]
```

count()

Cuenta el número de veces que aparece un ítem:

```
["Hola", "mundo", "mundo"].count("Hola")
1
```

index()

Devuelve el índice en el que aparece un ítem (error si no aparece):

```
["Hola", "mundo", "mundo"].index("mundo")
1
```

insert()

Agrega un ítem a la lista en un índice específico:

Primera posición (0):

```
l = [1,2,3]
l.insert(0,0)
l
[0, 1, 2, 3]
```

Penúltima posición (-1):

```
l = [5,10,15,25]
l.insert(-1,20)
l
[5, 10, 15, 20, 25]
```

Última posición en una lista con len():

```
l = [5,10,15,25]
```

```
n = len(l)
l.insert(n,30)
l
[5, 10, 15, 20, 25, 30]
```
Una posición fuera de rango añade el elemento al final de la lista (999):
```
l.insert(999, 35)
l
[5, 10, 15, 20, 25, 30, 35]
```

pop()

Extrae un ítem de la lista y lo borra:
```
l = [10,20,30,40,50]
print(l.pop())
print(l)
50
[10, 20, 30, 40]
```
Podemos indicarle un índice con el elemento a sacar (0 es el primer ítem):
```
print(l.pop(0))
print(l)
10
[20, 30, 40]
```

remove()

Borra el primer ítem de la lista cuyo valor concuerde con el que indicamos:
```
l = [20,30,30,30,40]
l.remove(30)
print(l)
[20, 30, 30, 40]
```

reverse()

Le da la vuelta a la lista actual:
```
l.reverse()
print(l)
[40, 30, 30, 20]
```
Las cadenas no tienen el método *.reverse()* pero podemos simularlo haciendo unas conversiones:
```
lista = list("Hola mundo")
lista.reverse()
cadena = "".join(lista)
cadena
'odnum aloH'
```

sort()

Ordena automáticamente los ítems de una lista por su valor de menor a mayor:
```
lista = [5,-10,35,0,-65,100]
lista.sort()
lista
[-65, -10, 0, 5, 35, 100]
```

Podemos utilizar el argumento reverse=True para indicar que la ordene del revés:

```
lista.sort(reverse=True)
lista
[100, 35, 5, 0, -10, -65]
```

Métodos de los diccionarios

```
colores = { "amarillo":"yellow", "azul":"blue", "verde":"green" }
```

get()

Busca un elemento a partir de su clave y si no lo encuentra devuelve un valor por defecto:

```
colores.get('negro','no se encuentra')
'no se encuentra'
```

keys()

Genera una lista en clave de los registros del diccionario:

```
colores.keys()
dict_keys(['amarillo', 'azul', 'verde'])
```

values()

Genera una lista en valor de los registros del diccionario:

```
colores.values()
dict_values(['yellow', 'blue', 'green'])
```

items()

Genera una lista en clave-valor de los registros del diccionario:

```
colores.items()
dict_items([('amarillo', 'yellow'), ('azul', 'blue'), ('verde',
'green')])
for clave, valor in colores.items():
    print(clave, valor)
amarillo yellow
azul blue
verde green
```

pop()

Extrae un registro de un diccionario a partir de su clave y lo borra, acepta valor por defecto:

```
colores.pop("amarillo", "no se ha encontrado")
'yellow'
colores.pop("negro","no se ha encontrado")
```

```
'no se ha encontrado'
```

clear()

Borra todos los registros de un diccionario:

```
colores.clear()
colores
{}
```

Métodos de los conjuntos

Métodos básicos

add()

Añade un ítem a un conjunto, si ya existe no lo añade:

```
c = set()
c.add(1)
c.add(2)
c.add(3)
c
{1, 2, 3}
```

discard()

Borra un ítem de un conjunto:

```
c.discard(1)
c
{2, 3}
```

copy()

Devuelva una copia de un conjunto, ya que éstos como la mayoría de colecciones se almacenan por referencia:

```
c1 = {1,2,3,4}
c2 = c.copy()
print(c1, c2)
c2.discard(4)
print(c1, c2)
{1, 2, 3, 4} {1, 2, 3, 4}
{1, 2, 3, 4} {1, 2, 3}
```

clear()

Borra todos los ítems de un conjunto:

```
c2.clear()
c2
set()
```

Comparación de conjuntos

```
c1 = {1,2,3}
c2 = {3,4,5}
c3 = {-1,99}
c4 = {1,2,3,4,5}
```

isdisjoint()

Comprueba si el conjunto es disjunto de otro conjunto, es decir, si no hay ningún elemento en común entre ellos:

```
c1.isdisjoint(c2)
False
```

issubset()

Comprueba si el conjunto es subconjunto de otro conjunto, es decir, si sus ítems se encuentran todos dentro de otro:

```
c3.issubset(c4)
False
```

issuperset()

Comprueba si el conjunto es contenedor de otro subconjunto, es decir, si contiene todos los ítems de otro:

```
c3.issuperset(c1)
False
```

Métodos avanzados

Se utilizan para realizar uniones, diferencias y otras operaciones avanzadas entre conjuntos.

Suelen tener dos formas, la normal que **devuelve** el resultado, y otra que hace lo mismo pero **actualiza** el propio resultado.

union()

Une un conjunto a otro y devuelve el resultado en un nuevo conjunto:

```
c1 = {1,2,3}
c2 = {3,4,5}
c3 = c1.union(c2)
print(c1, "+", c2, "=", c3)
{1, 2, 3} + {3, 4, 5} = {1, 2, 3, 4, 5}
```

update()

Une un conjunto a otro en el propio conjunto:

```
c1.update(c2)
c1
{1, 2, 3, 4, 5}
```

difference()
Encuentra los elementos no comunes entre dos conjuntos:
```
c1 = {1,2,3}
c2 = {3,4,5}

c1.difference(c2)
{1, 2}
```

difference_update()
Guarda en el conjunto los elementos no comunes entre dos conjuntos:
```
c1.difference_update(c2)
c1
{1, 2}
```

intersection()
Devuelve un conjunto con los elementos comunes en dos conjuntos:
```
c1 = {1,2,3}
c2 = {3,4,5}

c1.intersection(c2)
{3}
```

intersection_update()
Guarda en el conjunto los elementos comunes entre dos conjuntos:
```
c1.intersection_update(c2)
c1
{3}
```

symmetric_difference()
Devuelve los elementos simétricamente diferentes entre dos conjuntos, es decir, todos los elementos que no concuerdan entre los dos conjuntos:
```
c1 = {1,2,3}
c2 = {3,4,5}

c1.symmetric_difference(c2)
{1, 2, 4, 5}
```