


<https://pythonista.io>
[Entrar \(https://pythonista.io/login\)](https://pythonista.io/login)
[Menu](#)

Próximos eventos (<https://pythonista.io/events?mode=future>)

- Diplomado en línea: "Programador Python Jr."
(https://pythonista.io/events/python_jr)
09/03/2020 15:00 (2020-03-09T15:00:00-06:00)
(America/Mexico_City) — En línea
- Diplomado en línea: "Desarrollo de aplicaciones web con Django"
(<https://pythonista.io/events/django>)
23/03/2020 15:00 (2020-03-23T15:00:00-06:00)
(America/Mexico_City) — En línea con transmisión en vivo

 Eventos anteriores... (<https://pythonista.io/events?mode=past>)

 Eventos próximos... (<https://pythonista.io/events?mode=future>)

Palabras reservadas y nombres

 (<https://pythonista.io/cursos/py101/palabras-reservadas-y-espacios-de-nombres>)

Expresiones y declaraciones

 (<https://pythonista.io/cursos/py101/expresiones-y-declaraciones>)

Tipos de datos básicos y operadores

 (<https://pythonista.io/cursos/py101/tipos-de-datos-basicos-y-operadores>)

Orientación a objetos (<https://pythonista.io/cursos/py101/orientacion-a-objetos>)

Entrada y salida estándar (<https://pythonista.io/cursos/py101/entrada-y-salida-estandar>)

Bloques de código, comentarios y condicionales

 (<https://pythonista.io/cursos/py101/bloques-de-codigo-y-condicionales>)

Ciclos, iteraciones e interrupciones de flujo

 (<https://pythonista.io/cursos/py101/ciclos-con-while-e-interrupciones-de-flujo>)

Objetos tipo list y tipo tuple (<https://pythonista.io/cursos/py101/objetos-de-tipo-list-y-de-tipo-tuple>)

Objetos tipo dict (<https://pythonista.io/cursos/py101/objetos-tipo-dict>)

Objetos tipo str y bytes

}

Objetos de tipo set y frozenset

 (<https://pythonista.io/cursos/py101/objetos-de-tipo-set-y-frozenset>)








Funciones (<https://pythonista.io/cursos/py101/funciones>)

Gestión de excepciones (<https://pythonista.io/cursos/py101/gestion-de-excepciones>)

Iteradores y generadores (<https://pythonista.io/cursos/py101/iteradores-y-generadores>)

Completado de elementos

 (<https://pythonista.io/cursos/py101/completado-de-elementos>)

 Módulos y paquetes (https://pythonista.io/cursos/py101/modulos-y-paquetes)
 Escritura y lectura de archivos (https://pythonista.io/cursos/py101/escritura-y-lectura-de-archivos)
 Gestión de paquetes con pip (https://pythonista.io/cursos/py101/gestion-de-paquetes-con-pip)
 Descargas (https://pythonista.io/cursos/py101/descargas)
 paquete (https://pythonista.io/cursos/py101/paquete)
 modulo.py (https://pythonista.io/cursos/py101/modulo.py/view)
 modulo2.py (https://pythonista.io/cursos/py101/modulo2.py/view)

Objetos tipo str y bytes

Cadenas de texto en Python con objetos tipo str, bytes y unicode.

Objetos tipo *str*.

Python cuenta con varios tipos de objetos relacionados con cadenas de caracteres. El más común es el tipo *str*, el cual puede incluir cualquier tipo de caracter, incluyendo caracteres de escape.

Los objetos de tipo *str* son inmutables.

Los elementos contenidos en un objeto de tipo *str* no pueden ser modificados ni eliminados con *del*.

Ejemplos:

```
In [1]: saludo = "Hola, amigos."

In [2]: saludo[1]
Out[2]: 'o'

In [3]: saludo[1] = 'a'

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-3-799a9398b90a> in <module>()
----> 1 saludo[1] = 'a'

TypeError: 'str' object does not support item assignment

In [4]: del saludo[1]

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-4-283c8de2aa0c> in <module>()
----> 1 del saludo[1]

TypeError: 'str' object doesn't support item deletion
```

El tipo *str* cuenta con una enorme cantidad de métodos de diversas índoles que son de gran utilidad cuando se manipulan cadenas de texto.

Métodos de ajuste de mayúsculas y minúsculas.

lower()

Convierte la cadena de caracteres en minúsculas.

Ejemplo:

```
In [5]: 'Hola, amigos.'.lower()
Out[5]: 'hola, amigos.'
```

upper()

Convierte la cadena de caracteres en mayúsculas.

Ejemplo:

```
In [6]: 'Hola, amigos.'.upper()
```

```
Out[6]: 'HOLA, AMIGOS.'
```

capitalize()

Convierte al primer caracter del texto en mayúscula y el resto en minúsculas.

Ejemplo:

```
In [7]: 'hola, amigos.'.capitalize()
```

```
Out[7]: 'Hola, amigos.'
```

title()

Convierte al primer caracter de cada palabra en mayúsculas y los otros en minúsculas.

Ejemplo:

```
In [8]: 'hola, amigos.'.title()
```

```
Out[8]: 'Hola, Amigos.'
```

swapcase()

Transforma los caracteres que están en minúsculas a mayúsculas y los que están en mayúsculas a minúsculas.

Ejemplo:

```
In [9]: 'Hola, amigos.'.swapcase()
```

```
Out[9]: 'hOLA, AMIGOS.'
```

casefold()

Transforma el texto en un formato que permita compararlo con otro texto.

Ejemplo:

```
In [10]: 'HoLA, amiGoS.'.casefold()
```

```
Out[10]: 'hola, amigos.'
```

Métodos de ajuste del texto.***center()***

Centra el texto en una cadena del tamaño que se ingresa como argumento. Si se añade un caracter como segundo argumento, se sustituyen los espacios en blanco por dicho caracter.

Ejemplos:

```
In [11]: 'Hola, amigos.'.center(30)
```

```
Out[11]: '      Hola, amigos.      '
```

```
In [12]: 'Hola, amigos.'.center(30, 'x')
```

```
Out[12]: 'xxxxxxxxxHola, amigos.xxxxxxxxxx'
```

ljust()

Ajusta el texto a la izquierda dentro de una cadena del tamaño que se ingresa como argumento. Si se añade un caracter como segundo argumento, se sustituyen los espacios en blanco por dicho caracter.

Ejemplos:

```
In [13]: 'Hola, amigos.'.ljust(30)
```

```
Out[13]: 'Hola, amigos.          '
```

```
In [14]: 'Hola, amigos.'.ljust(30, 'x')
```

```
Out[14]: 'Hola, amigos.aaaaaaaaaaaaaaaaaaaa'
```

rjust()

Ajusta el texto a la derecha dentro de una cadena del tamaño que se ingresa como argumento. Si se añade un caracter como segundo argumento, se sustituyen los espacios en blanco por dicho caracter.

Ejemplos:

```
In [15]: 'Hola, amigos.'.rjust(30)
```

```
Out[15]: '                Hola, amigos.'
```

```
In [16]: 'Hola, amigos.'.rjust(30, 'x')
```

```
Out[16]: 'xxxxxxxxxxxxxxxxxxxxxHola, amigos.'
```

strip()

En el caso de que el objeto de tipo *str* contenga un texto que se encuentre entre cierto número de espacios vacíos, este método eliminaría dichos espacios. En caso de que se ingrese un caracter como argumento, se eliminarían los caracteres que encierran al texto.

Ejemplos:

```
In [17]: '        Hola, amigos.        '.strip()
```

```
Out[17]: 'Hola, amigos.'
```

```
In [18]: 'xxxxxxxxxxxxxxxxxxxxxHola, amigos.xxxxxxx'.strip('x')
```

```
Out[18]: 'Hola, amigos.'
```

lstrip()

Elimina los espacios vacíos que se encuentran a la izquierda de un texto. En caso de que se ingrese un caracter como argumento, se eliminarán los caracteres correspondientes a la izquierda.

Ejemplos:

```
In [19]: '        Hola, amigos.        '.lstrip()
```

```
Out[19]: 'Hola, amigos.        '
```

```
In [20]: 'xxxxxxxxxxxxxxxxxxxxxHola, amigos.xxxxxxx'.lstrip('x')
```

```
Out[20]: 'Hola, amigos.xxxxxxx'
```

rstrip()

Elimina los espacios vacíos que se encuentran a la derecha de un texto. En caso de que se ingrese un caracter como argumento, se eliminarán los caracteres correspondientes a la derecha.

Ejemplos:

```
In [21]: '        Hola, amigos.        '.rstrip()
```

```
Out[21]: '        Hola, amigos.'
```

```
In [22]: 'xxxxxxxxxxxxxxxxxxxxxHola, amigos.xxxxxxx'.rstrip('x')
```

```
Out[22]: 'xxxxxxxxxxxxxxxxxxxxxHola, amigos.'
```

expandtabs()

Sustituye los tabuladores con espacios en blanco. Por defecto se sustituye cada tabulador con 4 espacios en blanco, pero se puede definir el numero si se ingresa como argumento.

Ejemplos:

```
In [23]: 'Hola\tamigos'.expandtabs()
```

```
Out[23]: 'Hola    amigos'
```

```
In [24]: 'Hola\tamigos'.expandtabs(2)
```

```
Out[24]: 'Hola amigos'
```

zfill()

Genera una cadena de caracteres del tamaño indicado por el argumento y justifica el texto a la derecha, llenando el resto con ceros. **Ejemplos:**

```
In [25]: 'Hola'.zfill(6)
```

```
Out[25]: '000000'
```

```
In [26]: '504'.zfill(7)
```

```
Out[26]: '00000504'
```

format()

Este método permite crear un nuevo objeto de tipo *str* al que se le insertan elementos de forma similar a como lo hace la función *print()*. La sustitución se realiza mediante el uso de llaves (*{}*). Los elementos a sustituir se ingresan como argumentos separados por comas.

Ejemplo:

```
In [27]: 'Contrato de prestacion de servicios entre {} y {}'.format("Juan", "Pedro")
```

```
Out[27]: 'Contrato de prestacion de servicios entre Juan y Pedro.'
```

Si no se especifica, cada elemento será sustituido de forma sucesiva de izquierda a derecha. Pero también es posible indicar cual elemento sustituir por su índice. Los argumentos entonces son tratados como objetos de tipo *tuple* o de tipo *dict*.

Ejemplos:

```
In [28]: 'Contrato de prestacion de servicios entre {1} y {0}'.format("Juan", "Pedro")
```

```
Out[28]: 'Contrato de prestacion de servicios entre Pedro y Juan.'
```

```
In [29]: 'Contrato de servicios entre {comprador} y {vendedor}'.format(comprador="Juan",
                                                                    vendedor="Pedro")
```

```
Out[29]: 'Contrato de servicios entre Juan y Pedro.'
```

Este método permite formatear y alinear los elementos a los que sustituye de muchas maneras. En el sitio <https://pyformat.info/> (<https://pyformat.info/>) es posible explorar las distintas formas en las que se puede utilizar dicho método.

format_map()

Este método es similar a *format()* con la diferencia de que acepta exclusivamente objetos mapeables, tales como los objetos tipo *dict*.

Ejemplos:

```
In [30]: 'Contrato de servicios entre {comprador} y {vendedor}'.format_map({'comprador': "Juan",
                                                                    "vendedor": "Pedro"})
```

```
Out[30]: 'Contrato de servicios entre Juan y Pedro.'
```

```
In [31]: 'Contrato de servicios entre {comprador} y {vendedor}'.format_map(dict(comprador="Juan",
                                                                    vendedor="Pedro"))
```

```
Out[31]: 'Contrato de servicios entre Juan y Pedro.'
```

```
In [32]: 'Contrato de servicios entre {} y {}'.format_map(("Juan", "Pedro"))
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-32-e4bebbd87a78> in <module>()
----> 1 'Contrato de servicios entre {} y {}'.format_map(("Juan", "Pedro"))

ValueError: Format string contains positional fields
```

Métodos de separación y unión.

join()

Permite concatenar una secuencia de objetos de tipo *str* agrupados al ser ingresados como un objeto iterable, utilizando el objeto de tipo *str* como separador.

Ejemplos:

```
In [33]: ", ".join(["Hugo", "Paco", "Luis"])
```

```
Out[33]: 'Hugo, Paco, Luis'
```

```
In [34]: " + ".join(("1", "2", "3", "4"))
```

```
Out[34]: '1 + 2 + 3 + 4'
```

partition()

Este método busca de izquierda a derecha el patrón que se ingresa como parámetro dentro del objeto tipo *str* y al encontrar la primera coincidencia, divide en tres partes al texto. Regresa la parte de la izquierda, el patrón y la parte de la derecha dentro de un objeto tipo *tuple*. En caso de no encontrar coincidencia, el objeto tipo *tuple* se compondrá del texto original y dos cadenas de caracteres vacías.

Ejemplos:

```
In [35]: "Hugo, Paco, Luis".partition(", ")
```

```
Out[35]: ('Hugo', ', ', 'Paco, Luis')
```

```
In [36]: "Hugo, Paco, Luis".partition("+")
```

```
Out[36]: ('Hugo, Paco, Luis', '', '')
```

rpartition()

Este método busca de derecha a izquierda el patrón que se ingresa como parámetro dentro del objeto tipo *str* y al encontrar la primera coincidencia, divide en tres partes al texto. Regresa la parte de la izquierda, el patrón y la parte de la derecha dentro de un objeto tipo *tuple*. En caso de no encontrar coincidencia, el objeto tipo *tuple* se compondrá del texto dos cadenas de caracteres vacías y el texto original.

Ejemplos:

```
In [37]: "Hugo, Paco, Luis".rpartition(", ")
```

```
Out[37]: ('Hugo, Paco', ', ', 'Luis')
```

```
In [38]: "Hugo, Paco, Luis".rpartition("+")
```

```
Out[38]: ('', '', 'Hugo, Paco, Luis')
```

split()

Este método busca de izquierda a derecha el patrón que se ingresa como parámetro dentro del objeto tipo *str*, identifica a todas las coincidencias y regresa un objeto de tipo *list* con cada fragmento de texto delimitado por el patrón. En caso de no encontrar al patrón, regresará un objeto de tipo *list*, conteniendo al objeto *str* original.

Ejemplos:

```
In [39]: "Hugo, Paco, Luis".split(", ")
```

```
Out[39]: ['Hugo', 'Paco', 'Luis']
```

```
In [40]: "Hugo, Paco, Luis".split("+")
```

```
Out[40]: ['Hugo, Paco, Luis']
```

rsplit()

Este método busca de derecha a izquierda el patrón que se ingresa como parámetro dentro del objeto tipo *str*, identifica a todas las coincidencias y regresa un objeto de tipo *list* con cada fragmento de texto delimitado por el patrón. En caso de no encontrar al patrón, regresará un objeto de tipo *list*, conteniendo al texto original.

Ejemplos:

```
In [41]: "Hugo, Paco, Luis".rsplit(", ")
```

```
Out[41]: ['Hugo', 'Paco', 'Luis']
```

```
In [42]: "Hugo, Paco, Luis".rsplit("+")
```

```
Out[42]: ['Hugo, Paco, Luis']
```

splitlines()

Método que identifica retornos de línea "\n" y regresa un objeto de tipo *list* conteniendo cada línea identificada. en caso de no encontrar un retorno de línea, regresará un objeto de tipo *list* conteniendo a su vez un objeto de tipo *str* conteniendo al texto original.

Ejemplo:

```
In [43]: nombres = "Hugo.\nPaco.\nLuis."
         print(nombres)

Hugo.
Paco.
Luis.
```

```
In [44]: nombres.splitlines()
```

```
Out[44]: ['Hugo.', 'Paco.', 'Luis.']
```

Métodos de búsqueda y reemplazo.

find()

Método que busca de izquierda a derecha un patrón que es ingresado como argumento y regresa el índice del primero que encuentra. En caso de no encontrarlo regresará el valor -1.

Ejemplos:

```
In [45]: 'Luis, Hugo, Paco, Luis'.find('Paco')
```

```
Out[45]: 12
```

```
In [46]: 'Luis, Hugo, Paco, Luis'.find('Luis')
```

```
Out[46]: 0
```

```
In [47]: 'Luis, Hugo, Paco, Luis'.find('Juan')
```

```
Out[47]: -1
```

rfind()

Método que busca de derecha a izquierda un patrón que es ingresado como argumento y regresa el índice del primero que encuentra. En caso de no encontrarlo regresará el valor -1.

Ejemplos:

```
In [48]: 'Luis, Hugo, Paco, Luis'.rfind('Paco')
```

```
Out[48]: 12
```

```
In [49]: 'Luis, Hugo, Paco, Luis'.rfind('Luis')
```

```
Out[49]: 18
```

```
In [50]: 'Luis, Hugo, Paco, Luis'.rfind('Juan')
```

```
Out[50]: -1
```

index()

Método que busca de izquierda a derecha un patrón que es ingresado como argumento y regresa el índice del primero que encuentre. En caso de no encontrarlo se generará un mensaje de error.

Ejemplos:

```
In [51]: 'Luis, Hugo, Paco, Luis'.index('Paco')
```

```
Out[51]: 12
```

```
In [52]: 'Luis, Hugo, Paco, Luis'.index('Luis')
```

```
Out[52]: 0
```

```
In [53]: 'Luis, Hugo, Paco, Luis'.index('Juan')
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-53-44c3b0d23805> in <module>()
----> 1 'Luis, Hugo, Paco, Luis'.index('Juan')

ValueError: substring not found
```

rindex()

Método que busca derecha a izquierda un patrón que es ingresado como argumento y regresa el índice del primero que encuentre. En caso de no encontrarlo se generará un mensaje de error de tipo *ValueError*.

Ejemplos:

```
In [54]: 'Luis, Hugo, Paco, Luis'.rindex('Paco')
```

```
Out[54]: 12
```

```
In [55]: 'Luis, Hugo, Paco, Luis'.rindex('Luis')
```

```
Out[55]: 18
```

```
In [56]: 'Luis, Hugo, Paco, Luis'.rindex('Juan')
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-56-2e77757f6557> in <module>()
----> 1 'Luis, Hugo, Paco, Luis'.rindex('Juan')
```

ValueError: substring not found

replace()

Método al que se le ingresan un patrón y un texto de sustitución. Regresará un objeto tipo *str* sustituyendo cada coincidencia. En caso de no encontrar coincidencias, regresará un objeto tipo *str* con el texto original.

Ejemplos:

```
In [57]: 'Luis, Hugo, Paco, Luis'.replace('Luis', 'Juan Antonio')
```

```
Out[57]: 'Juan Antonio, Hugo, Paco, Juan Antonio'
```

```
In [58]: 'Luis, Hugo, Paco, Luis'.replace('Pedro', 'Juan')
```

```
Out[58]: 'Luis, Hugo, Paco, Luis'
```

Métodos de evaluación.

count()

Cuenta el número de veces que aparece un patrón ingresado como argumento dentro del texto.

Ejemplos:

```
In [59]: "Hola, amigos.".count("a")
```

```
Out[59]: 2
```

```
In [60]: "Hola, amigos.".count("la")
```

```
Out[60]: 1
```

```
In [61]: "Hola, amigos.".count("z")
```

```
Out[61]: 0
```

startswith()

Evalúa si el patrón ingresado como argumento corresponde al inicio del texto en el objeto de tipo *str*. Regresa *True* si hay correspondencia y de lo contrario, regresa *False*.

Ejemplos:

```
In [62]: "Hola, amigos.".startswith("la")
```

```
Out[62]: False
```

```
In [63]: "Hola, amigos.".startswith("Hola")
```

```
Out[63]: True
```

endswith()

Evalúa si el patrón ingresado como argumento corresponde al final del texto en el objeto de tipo *str*. Regresa *True* si hay correspondencia y de lo contrario, regresa *False*.

Ejemplos:


```
In [64]: "Hola, amigos.".endswith("Hola")
```

```
Out[64]: False
```

```
In [65]: "Hola, amigos.".endswith("gos.")
```

```
Out[65]: True
```

isalnum()

Evalúa si el texto del objeto tipo *str* contiene exclusivamente caracteres alfanuméricos.

Ejemplos:

```
In [66]: "@#%&".isalnum()
```

```
Out[66]: False
```

```
In [67]: 'Hola 12'.isalnum()
```

```
Out[67]: False
```

```
In [68]: "Hola12".isalnum()
```

```
Out[68]: True
```

isalpha()

Evalúa si el texto del objeto tipo *str* contiene exclusivamente caracteres alfabéticos (en cualquier alfabeto de Unicode).

Ejemplos:

```
In [69]: "@#%&".isalpha()
```

```
Out[69]: False
```

```
In [70]: 'Hola'.isalpha()
```

```
Out[70]: True
```

```
In [71]: 'Hola.'.isalpha()
```

```
Out[71]: False
```

```
In [72]: 'Ω'.isalpha()
```

```
Out[72]: True
```

isnumeric()

Evalúa si el texto del objeto tipo *str* contiene caracteres exclusivamente con representación numérica.

Ejemplos:

```
In [73]: '0x1a'.isnumeric()
```

```
Out[73]: False
```

```
In [74]: '-22.1'.isnumeric()
```

```
Out[74]: False
```

```
In [75]: '23'.isnumeric()
```

```
Out[75]: True
```

```
In [76]: '2²'.isnumeric()
```

```
Out[76]: True
```

```
In [77]: '½'.isnumeric()
```

```
Out[77]: True
```

isdecimal()

Evalúa si el texto del objeto tipo *str* contiene caracteres exclusivamente con representación numérica decimal.

Ejemplos:

```
In [78]: '0x1a'.isdecimal()
```

```
Out[78]: False
```

```
In [79]: '-22.1'.isdecimal()
```

```
Out[79]: False
```

```
In [80]: '23'.isdecimal()
```

```
Out[80]: True
```

```
In [81]: '2²'.isdecimal()
```

```
Out[81]: False
```

```
In [82]: '½'.isdecimal()
```

```
Out[82]: False
```

isdigit()

Evalúa si el texto del objeto tipo *str* contiene caracteres exclusivamente que representen dígitos.

Ejemplos:

```
In [83]: '0x1a'.isdigit()
```

```
Out[83]: False
```

```
In [84]: '-22.1'.isdigit()
```

```
Out[84]: False
```

```
In [85]: '23'.isdigit()
```

```
Out[85]: True
```

```
In [86]: '2²'.isdigit()
```

```
Out[86]: True
```

```
In [87]: '½'.isdigit()
```

```
Out[87]: False
```

islower()

Evalúa si los caracteres alfabéticos contenidos en el texto están solamente en minúsculas.

Ejemplos:

```
In [88]: '¡hola, amigos!'.islower()
```

```
Out[88]: True
```

```
In [89]: '¡Hola, amigos!'.islower()
```

```
Out[89]: False
```

isupper()

Evalúa si los caracteres alfabéticos contenidos en el texto están solamente en mayúsculas.

Ejemplos:

```
In [90]: '¡HOLA, AMIGOS!'.isupper()
```

```
Out[90]: True
```

```
In [91]: '¡Hola, amigos!'.isupper()
```

```
Out[91]: False
```

istitle()

Evalúa si cada palabra del texto contenido en el objeto tipo *str* comience con mayúsculas y el resto sean minúsculas.

Ejemplos:

```
In [92]: '¡HOLA, AMIGOS!'.istitle()
```

```
Out[92]: False
```

```
In [93]: '¡Hola, amigos!'.istitle()
```

```
Out[93]: False
```

```
In [94]: '¡Hola, Amigos!'.istitle()
```

```
Out[94]: True
```

```
In [95]: '¡HOLA, Amigos!'.istitle()
```

```
Out[95]: False
```

isprintable()

Evalúa si el texto contenido en el objeto tipo *str* incluye sólo caracteres imprimibles.

Ejemplos:

```
In [96]: nombres = "Hugo.\nPaco.\nLuis."
print(nombres)
```

```
Hugo.
Paco.
Luis.
```

```
In [97]: nombres.isprintable()
```

```
Out[97]: False
```

```
In [98]: nombres = "Hugo.\tPaco.\tLuis."
print(nombres)
```

```
Hugo.  Paco.  Luis.
```

```
In [99]: nombres.isprintable()
```

```
Out[99]: False
```

```
In [100]: 'Hugo, Paco, Luis.'.isprintable()
```

```
Out[100]: True
```

isspace()

Evalúa si el objeto tipo *str* está compuesto exclusivamente de espacios.

Ejemplos:

```
In [101]: ''.isspace()
```

```
Out[101]: False
```

```
In [102]: ' '.isspace()
```

```
Out[102]: True
```

```
In [103]: 'Hola, mundo.'.isspace()
```

```
Out[103]: False
```

isidentifier()

Evalúa si el objeto tipo *str* cumple con las características de un identificador.

Ejemplos:

```
In [104]: 'Nombre'.isidentifier()
```

```
Out[104]: True
```

```
In [105]: '!Nombre'.isidentifier()
```

```
Out[105]: False
```

```
In [106]: 'Hola, mundo.'.isidentifier()
```

Out[106]: False

Métodos de mapeo de caracteres.

Es posible generar tablas de caracteres, los cuales puedan sustituir a unos por otros. Este tipo de sustitución se hace generalmente cuando se debe interactuar con aplicaciones que no aceptan caracteres especiales.

maketrans()

Este método permite crear una tabla de caracteres, la cual consiste en un diccionario que relaciona a un par de caracteres por su número correspondiente en la codificación.

Ejemplo:

```
In [107]: trans = "".maketrans("áàâëèéìíîóôöüù", "aaaaeeiiiiooooo")
trans
```

```
Out[107]: {224: 97,
225: 97,
228: 97,
232: 101,
233: 101,
235: 101,
236: 105,
237: 105,
239: 105,
242: 111,
243: 111,
246: 111,
249: 117,
250: 117,
252: 117}
```

translate()

Este método sustituye los caracteres a partir de una tabla de caracteres.

Ejemplo:

```
In [108]: trans = ''.maketrans("áàâëèéìíîóôöüù", "aaaaeeiiiiooooo")
'Hóla amigòs. ¿Cómò éstán?'.translate(trans)
```

```
Out[108]: 'Hola amigos. ¿Como estan?'
```

```
In [109]: trans = ''.maketrans('oieastg', '0134579')
"Hola, amigos. ¿Como estan?".translate(trans)
```

```
Out[109]: 'H0l4, 4m1905. ¿C0m0 3574n?'
```

Métodos de codificación.

Como ya se ha mencionado, Python 3 utiliza la codificación de caracteres UTF-8, un subconjunto de la especificación de Unicode; mientras que Python 2 utiliza la codificación ASCII. Las codificaciones estándar soportadas por Python 3 pueden ser consultadas desde [esta liga](https://docs.python.org/3/library/codecs.html#standard-encodings) (<https://docs.python.org/3/library/codecs.html#standard-encodings>).

Es posible cambiar el tipo de codificación del texto contenido en un objeto tipo *str* a otra codificación.

encode()

Este método permite transformar el contenido de un objeto *str* de una codificación en otra. El resultado de la transformación es un objeto tipo *bytes*. El formato de codificación se define mediante la asignación del valor de dicha codificación al identificador *encoding*. El valor por defecto de *encoding* es "utf-8".

Es común que no todos los caracteres tengan una correspondencia de una codificación a otra, por lo que se pueden generar errores de conversión. Se pueden definir cuatro formas de gestionar estos errores ingresando el valor de la forma relacionado con el identificador *errors*.

- *'strict'*, el cual emite un error de codificación. Si no se indica, esta es la forma por defecto de gestionar los errores de codificación.
- *'ignore'*, el cual ignora el carácter en cuestión y prosigue con el siguiente.
- *'replace'*, el cual sustituye al carácter en cuestión por un signo de interrogación (?).
- *'xmlcharrefreplace'*, el cual sustituye el carácter por el valor correspondiente a su [valor de entidad de carácter de XML](https://es.wikipedia.org/wiki/Anexo:Entidades_de_caracteres_XML_y_HTML) (https://es.wikipedia.org/wiki/Anexo:Entidades_de_caracteres_XML_y_HTML).

Ejemplos:

```
In [110]: 'Hołłé'.encode(encoding="ascii", errors='ignore')
```

```
Out[110]: b'Hol'
```

```
In [111]: 'Holåé'.encode(encoding="ascii", errors='replace')
```

```
Out[111]: b'Hol??'
```

```
In [112]: 'Holåé'.encode(encoding="ascii", errors='xmlcharrefreplace')
```

```
Out[112]: b'Hol&#229;&#233;'
```

```
In [113]: 'Holåé'.encode(encoding="ascii", errors='strict')
```

```
-----
UnicodeEncodeError                                Traceback (most recent call last)
<ipython-input-113-9caa06c15fe5> in <module>()
----> 1 'Holåé'.encode(encoding="ascii", errors='strict')

UnicodeEncodeError: 'ascii' codec can't encode characters in position 3-4: ordinal not in range(128)
```

Objetos tipo bytes.

Además de los objetos de tipo *str* existen diversos tipos de datos que también pueden contener caracteres.

Son una sucesión ordenada de valores que representan a un carácter conforme a su número correspondiente en el [código ASCII](http://www.asciitable.com/) (<http://www.asciitable.com/>) y se definen anteponiendo la letra *b* a los apóstrofes o comillas.

```
b'<texto>'
b"<texto>"
```

Cuando se utiliza un objeto de tipo *bytes* en la función *print()*, no despliega un texto sino el valor del objeto.

Ejemplos:

```
In [114]: palabra = b"Hola"
```

```
In [115]: type(palabra)
```

```
Out[115]: bytes
```

```
In [116]: print(palabra)
```

```
b'Hola'
```

```
In [117]: palabra[0]
```

```
Out[117]: 72
```

```
In [118]: palabra[2:4]
```

```
Out[118]: b'la'
```

```
In [119]: palabra[1] = b'm'
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-119-b789d5a9a16e> in <module>()
----> 1 palabra[1] = b'm'

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

Transformaciones entre objetos *str* y *bytes*.

Para transformar un objeto de tipo *str* a *bytes* se utiliza la función *bytes()*, ingresando al objeto como primer argumento y posteriormente el valor al que corresponde la codificación que se aplicará. En el caso de querer transformar un objeto *bytes* a *str* se utiliza la función *str()* ingresando al objeto como primer argumento y posteriormente el valor al que corresponde la codificación que se aplicará.

```
In [120]: print(bytes('hola', "utf-8"))
```

```
b'hola'
```

```
In [121]: print(str(b'hola'[1:3], 'ascii'))
```

```
ol
```

Métodos de los objetos tipo *bytes*.

Los objetos de tipo *bytes* comparten los siguientes métodos con los de tipo *str*:

- *capitalize()*

- `center()`
- `count()`
- `endswith()`
- `expandtabs()`
- `find()`
- `index()`
- `isalnum()`
- `isalpha()`
- `isdigit()`
- `islower()`
- `isspace()`
- `istitle()`
- `isupper()`
- `join()`
- `ljust()`
- `lower()`
- `lstrip()`
- `maketrans()`
- `partition()`
- `replace()`
- `rfind()`
- `rindex()`
- `rjust()`
- `rpartition()`
- `rsplit()`
- `rstrip()`
- `split()`
- `splitlines()`
- `startswith()`
- `strip()`
- `swapcase()`
- `title()`
- `translate()`
- `upper()`
- `zfill()`

`decode()`

Este método funciona de manera inversa al método `code()` de los objetos tipo `str`. Se ingresan como argumentos el valor de codificación ligado al identificador `encoding` y el valor de manejo de errores de codificación ligado al identificador `errors`.

Ejemplo:

```
In [122]: b'HoLa, '.decode(encoding="utf-8")
```

```
Out[122]: 'HoLa, '
```

`hex()`

Regresa un objeto tipo `str` que contiene la representación en hexadecimal de cada elemento del objeto tipo `bytes`.

Ejemplo:

```
In [123]: b'HoLa'.hex()
```

```
Out[123]: '486f6c61'
```

`fromhex()`

Regresa un objeto tipo `bytes` a partir de un objeto `str` que contiene la representación en hexadecimal de uno o más caracteres ASCII.

Ejemplo:

```
In [124]: b''.fromhex('486f6c61')
```

```
Out[124]: b'HoLa'
```

Funciones útiles con los objetos tipo `str` y tipo `bytes`.

La función `len()`.

La función `len()` devuelve el número de caracteres contenidos en un objeto tipo `str` o tipo `bytes` mediante la siguiente sintaxis:

```
len(<objeto>)
```

Ejemplo:

```
In [125]: len(b'Hola')
```

```
Out[125]: 4
```

La función *max()*.

La función *max()* devuelve el elemento de mayor valor contenido en un objeto de tipo *bytes* o *str* mediante la siguiente sintaxis:

```
max(<objeto>)
```

Ejemplo:

```
In [126]: max('Hola')
```

```
Out[126]: 'o'
```

```
In [127]: max(b'Hola')
```

```
Out[127]: 111
```

La función *min()*.

La función *min()* devuelve el elemento de mayor valor contenido en un objeto de tipo *bytes* o *str* mediante la siguiente sintaxis:

```
min(<objeto>)
```

Ejemplos:

```
In [128]: min(b'Hola')
```

```
Out[128]: 72
```

```
In [129]: min('Hola')
```

```
Out[129]: 'H'
```

La función *sum()* sólo para tipo *bytes*.

La función *sum()* devuelve la suma de los valores contenido en un objeto de tipo *bytes* mediante la siguiente sintaxis:

```
sum(<objeto>)
```

Ejemplos:

```
In [130]: sum(b'Hola')
```

```
Out[130]: 388
```

Iteraciones con *for...in* para objetos tipo *str* y tipo *bytes*.

Tanto para los objetos tipo *str* como para los objetos tipo *bytes*, cada caracter se vuelve un elemento iterable.

Ejemplos:

```
In [131]: for letra in "Hola":  
          print(letra)
```

```
H  
o  
l  
a
```

```
In [132]: for caracter in b"Hola":  
          print(caracter)
```


```
72  
111  
108  
97
```

Objetos tipo *unicode* en Python 2.

Python 2 incluye un tipo *unicode*, el cual se identifica con una *u*, previa a los apóstrofes o comillas. Python 3 no incluye a este tipo de dato.






(<http://creativecommons.org/licenses/by/4.0/>)

Esta obra está bajo una  [Licencia Creative Commons Atribución 4.0 Internacional](http://creativecommons.org/licenses/by/4.0/) (<http://creativecommons.org/licenses/by/4.0/>).

© José Luis Chiquete Valdivieso. 2018.

« Anterior: Objetos tipo dict (<https://pythonista.io/cursos/py101/objetos-tipo-dict>)

Siguiente: Objetos de tipo set y frozenset » (<https://pythonista.io/cursos/py101/objetos-de-tipo-set-y-frozenset>)

El  Plone® CMS/WCM de Fuentes Abiertas (<http://plone.com>) es © ([Creative Commons](http://creativecommons.org/licenses/by/4.0/)) 2000-2020 por la  Fundación Plone (<http://plone.org/foundation>) y amigos. Distribuido bajo la  [Licencia GNU GPL](http://creativecommons.org/licenses/GPL/2.0/) (<http://creativecommons.org/licenses/GPL/2.0/>)

[Mapa del Sitio](https://pythonista.io/sitemap) (<https://pythonista.io/sitemap>)

[Accesibilidad](https://pythonista.io/accessibility-info) (<https://pythonista.io/accessibility-info>)

[Contacto](https://pythonista.io/contact-info) (<https://pythonista.io/contact-info>)



Powered by Plone & Python (<http://plone.com>)