The Python Standard Library

Methods & Functions

Nathan Warner

Computer Science
Northern Illinois University
November 9, 2023 United States

Contents

1 String Methods

Pertains to CHANGING upper/lowercase (6)

Upper Case:

- ☞ **capitalize()** Converts the first character to upper case ⟶ ***[Paramaters: Null] [Return: String]***
- ☞ **title()** Converts the first character of each word to upper case ⟶ ***[Null] [String]***
- ☞ **upper()** Converts a string into upper case ⟶ ***[Null] [String]***

Lower Case:

- ☞ **casefold()** Converts string into lower case ⟶ ***[Null] [String]***
- ☞ **lower()** Converts a string into lower case ⟶ ***[Null] [String]***

Both:

- ☞ **swapcase()** Swaps cases, lower case becomes upper case and vice versa ⟶ ***[Null] [String]***

"Is" Methods/Returns Bool (14)

- ☞ **isupper()** Returns True if all characters in the string are upper case ⟶ ***[Null] [bool]***
- ☞ **islower()** Returns True if all characters in the string are lower case ⟶ ***[Null] [bool]***
- ☞ **isalnum()** Returns True if all characters in the string are alphanumeric ⟶ ***[Null] [bool]***
- ☞ **isalpha()** Returns True if all characters in the string are in the alphabet ⟶ ***[Null] [bool]***
- ☞ **isascii()** Returns True if all characters in the string are ascii characters ⟶ ***[Null] [bool]***
- ☞ **isdecimal()** Returns True if all characters in the string are decimals ⟶ ***[Null] [bool]***
- ☞ **isdigit()** Returns True if all characters in the string are digits ⟶ ***[Null] [bool]***
- ☞ **isidentifier()** Returns True if the string is an identifier ⟶ ***[Null] [bool]***
- ☞ **isnumeric()** Returns True if all characters in the string are numeric ⟶ ***[Null] [bool]***
- ☞ **isprintable()** Returns True if all characters in the string are printable ⟶ ***[Null] [bool]***
- ☞ **isspace()** Returns True if all characters in the string are whitespaces ⟶ ***[Null] [bool]***
- ☞ **istitle()** Returns True if the string follows the rules of a title ⟶ ***[Null] [bool]***
- ☞ **endswith()** Returns true if the string ends with the specified value ⟶ ***[r:Value, o:Start, o:End] [bool]***
- ☞ **startswith()** Returns true if the string starts with the specified value ⟶ ***[r:Value, o:Start, o:End] [bool]***

Searching (4)

- ☞ **find()** Searches the string for a specified value and returns the position of where it was found
  [r:Value, o:Start, o:End] [Int: Pos of first occurence] [Int: -1 if not found]
- ☞ **rfind()** Searches the string for a specified value and returns the last position of where it was found
  [r:Value, o:Start, o:End] [Int: Pos of last occurence] [Int: -1 if not found]
- ☞ **index()** Searches the string for a specified value and returns the position of where it was found
  [r:Value, o:Start, o:End] [Int: Pos of first occurence] [Throws error if not found]
- ☞ **rindex()** Searches the string for a specified value and returns the last position of where it was found
  [r:Value, o:Start, o:End] [Int: Pos of last occurence] [Throws error if not found]
- ☞ **count()** Returns the number of times a specified value occurs in a string
  [r:Value, o:Start, o:End] [Int: Num of occurences]

Mutate String (11)

- ☞ **replace()** Returns a string where a specified value is replaced with a specified value
  [r:oldvalue, r:newvalue, o:count] [String]
- ☞ **center()** Returns a centered string, default character is " "
  [r:length, o:character] [string]
- ☞ **strip()** Returns a trimmed version of the string
  [o:Character]
- ☞ **lstrip()** Returns a left trim version of the string, default character is " "
  [o:Character] [String]
- ☞ **rstrip()** Returns a right trim version of the string, default character is " "
  [o:Character] [String]
- ☞ **rjust()** Returns a right justified version of the string, defualt character is " "
  [r:Length, o:Character] [String]
- ☞ **ljust()** Returns a left justified version of the string, default character is " "
  [r:Length, o:Character] [String]
- ☞ **zfill()** Fills the string with a specified number of 0 values at the beginning
  [r:length] [String]
- ☞ **maketrans()** Returns a translation table to be used in translations
  [r:String, r:String, o:String[characters to remove]] [Dict]
- ☞ **translate()** Returns a translated string
  [r:Table[dict]] [String]
- ☞ **encode()** Returns an encoded version of the string, If no encoding is specified, UTF-8 will be used.
  [o:Encoding, o:Errors[See Web]]

Returns a List/Tuple (5)

Returns List:

- ☞ **split()** Splits the string at the specified separator, and returns a list

  [o:Separator, o:Maxsplit] [List]

- ☞ **rsplit()** Splits the string at the specified separator, and returns a list

  [o:Separator, o:Maxsplit] [List]

- ☞ **splitlines()** Splits the string at line breaks and returns a list

  [o:keeplinebreaks=Bool] [List]

Returns Tuple:

- ☞ **partition()** Returns a tuple where the string is parted into three parts

  [r:Value] [Tuple]

- ☞ **rpartition()** Returns a tuple where the string is parted into three parts

  [r:Value] [Tuple]

Format String (2)

- ☞ **format()** Formats specified values in a string

  [r:values[comma seperated list] [String]]

- ☞ **format_map()** Formats specified values in a string

  [r:Dict], [String]

- ☞ **expandtabs()** Sets the tab size of the string

  [r:Tabsize[int]] [String]

Takes iterable and turns into string (1)

- ☞ **join()** Converts the elements of an iterable into a string, A string must be specified as the separator **before the method call**

  [r:Iterable[Must be strings]] [String]

Note:-

When using a dictionary as an iterable, the returned values are the keys, not the values.

2 List Methods

Add To List (2)

- ☞ **append()** Adds an element at the end of the list

  [r:Element] [Mutates List]

- ☞ **insert()** Adds an element at the specified position

  [r:Pos, r:Element] [Mutates List]

- ☞ **extend()** Add the elements of a list (or any iterable), to the end of the current list

  [r:Iterable] [Mutates List]

Remove From list (2)

- ☞ **remove()** Removes the first item with the specified value

  [r:Element] [Mutates List]

- ☞ **pop()** Removes the element at the specified position

  [r:Pos] [Mutates List]

- ☞ **clear()** Removes all the elements from the list

  [Null] [Mutates List]

Search (2)

- ☞ **index()** Returns the index of the first element with the specified value

  [r:Element] [Int]

- ☞ **count()** Returns the number of elements with the specified value

  [r:Value] [Int]

Other (3)

- ☞ **copy()** Returns a copy of the list

  [Null] [List]

- ☞ **reverse()** Reverses the order of the list

  [Null] [Mutates List]

- ☞ **sort()** Sorts the list

  [o:reverse=Bool, o:key=Func] [Mutates List]

Bonus - Tuple Methods (2)

- ☞ **count()** Returns the number of times a specified value occurs in a tuple

  [r:Value] [Int]

- ☞ **index()** Searches the tuple for a specified value and returns the position of where it was found

  [r:Element] [Int]

3 Dict Methods

Fetch (7)

- ☞ **fromkeys()** Returns a dictionary with the specified keys and value

  [r:Keys, o:Value] [Dict]

  Keys: Required. An iterable specifying the keys of the new dictionary

  Value: Optional. The value for all keys. Default value is None

- ☞ **get()** Returns the value of the specified key

  [r:Keyname, o:Value] [Type of retrieved item]

  Keyname: Required. The keyname of the item you want to return the value from

  Value: Optional. A value to return if the specified key does not exist. Default value None

- ☞ **items()** Returns a list containing a tuple for each key value pair

  [Null] [View Object: Tuple]

- ☞ **keys()** Returns a list containing the dictionary's keys

  [Null] [View Object]

- ☞ **values()** Returns a list of all the values in the dictionary

  [Null] [View Object]

- ☞ **setdefault()** Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

  [r:Keyname, r:Value] [Value/Mutates Dict]

- ☞ **Update** Insert an item to the dictionary

  [r:Dict] [Mutates Dict]

Remove Items (3)

- ☞ **pop()** Removes the element with the specified key

  [r:Keyname, o:ReturnValueIfDNE] [Mutates Dict/Specified Return Value]

- ☞ **popitem()** Removes the last inserted key-value pair

  [Null] [Mutates list and returns tuple of removed item]

- ☞ **clear()** Removes all the elements from the dictionary

  [Null], [Mutates Dict]

Other (1)

- ☞ **copy()** returns a copy of the specified dictionary.

  [Null] [Dict]

4 Set Methods

Add to set (3)

☞ **add()** Adds an element to the set

[r:Element] [Mutates Set]

☞ **update()** Update the set with another set, or any other iterable

[Set] [Mutates Set]

☞ **symmetric_difference_update()** inserts the symmetric differences from this set and another

[r:Set] [Mutates Set]

Remove from set (6)

☞ **pop()** Removes an element from the set

[Null] [Removed Element]

☞ **remove()** Removes the specified element

[r:Item] [Mutates Set]

☞ **clear()** Removes all the elements from the set

[Null] [Mutates Set]

☞ **discard()** Remove the specified item

[r:Value] [Mutates Set]

☞ **difference_update()** Removes the items in this set that are also included in another, specified set

[r:Set] [Mutates Set]

☞ **intersection_update()** Removes the items in this set that are not present in other, specified set(s)

[r:Set] [Mutates Set]

Note:-

the discard() method is different from the remove() method, because the remove() method will raise an error if the specified item does not exist, and the discard() method will not.

is **methods [Returns Bool] (3)**

☞ **isdisjoint()** Returns whether two sets have a intersection or not

[r:Set] [Bool]

☞ **issubset()** Returns whether another set contains this set or not

[r:Set] [Bool]

☞ **issuperset()** Returns whether this set contains another set or not

[r:Set] [Bool]

Returns a Set (5)

☞ **difference()** Returns a set containing the difference between two or more sets

[r:Set] [Set]

☞ **intersection()** Returns a set, that is the intersection of two or more sets

[r:Set] [Set]

☞ **symmetric_difference()** Returns a set with the symmetric differences of two sets

[r:Set] [Set]

☞ **union()** Return a set containing the union of sets

[r:Set] [Set]

☞ **copy()** Returns a copy of the set

[Null] [Set]

5 Builtin Functions

Types (15)

☞ **type() Returns the type of an object**

Parameters: (obj, bases, dict)

r:Object → Required. If only one parameter is specified, the type() function returns the type of this object

o:Bases → Optional. Specifies the base classes

o:Dict → Optional. Specifies the namespace with the definition for the class

☞ **int() Returns an integer number**

Parameters: (Value, Base)

r:Value → A number or a string that can be converted into an integer number

o:Base → base A number representing the number format. Default value: 10

☞ **str() Returns a string object**

Parameters: (obj, encoding=encoding, errors=errors)

r:Object → Any object. Specifies the object to convert into a string

o:Encoding → Any object. Specifies the object to convert into a string

o:Errors Specifies what to do if the decoding fails

☞ **float() Returns a floating point number**

Parameters: (Value)

r:Value → A number or a string that can be converted into a floating point number

☞ **bool() Returns the boolean value of the specified object**

Parameters: (Obj)

r:Object → Any object, like String, List, Number etc.

Note: The object will always return True, unless:
The object is empty, like [], (),
The object is False
The object is 0
The object is None

☞ **complex() Returns a complex number**

Parameters: (Real, Imaginary)

r:Real → Required. A number representing the real part of the complex number. Default 0. The real number can also be a String, like this '3+5j', when this is the case, the second parameter should be omitted.

o:Imaginary → Optional. A number representing the imaginary part of the complex number. Default 0.

☞ **bytes() Returns a bytes object**

Parameters: (x, encoding, error)

r:X → A source to use when creating the bytes object.
If it is an integer, an empty bytes object of the specified size will be created.
If it is a String, make sure you specify the encoding of the source.

o:Encoding → The encoding of the string

o:Error → Specifies what to do if the encoding fails

☞ **bytearray() Returns an array of bytes**

Parameters: (x, encoding, error)

r:X → A source to use when creating the bytes object.
If it is an integer, an empty bytes object of the specified size will be created.
If it is a String, make sure you specify the encoding of the source.

o:Encoding → The encoding of the string

o:Error → Specifies what to do if the encoding fails

☞ **bin() Returns the binary version of a number**

Parameters: (n)

r:N → An integer

Note: The result will always start with the prefix 0b.

☞ **list() Returns a list**

Parameters: (iterable)

o:Iterable → A sequence, collection or an iterator object

☞ **tuple() Returns a tuple**

Parameters: (iterable)

o:Iterable → A sequence, collection or an iterator object

☞ **dict() Returns a dictionary (Array)**

Parameters: (kwargs)

o:Kwargs → As many keyword arguments you like, separated by comma: key = value, key = value ...

☞ **set()Returns a new set object**

Parameters: (iterable)

r:Optional → A sequence, collection or an iterator object

☞ **frozenset() Returns a frozenset object**

Parameters: (iterable)

o:Iterable → An iterable object, like list, set, tuple etc.

Works on Iteratables (6)

☞ **all() Returns True if all items in an iterable object are true**

Parameters: (iterable)

r:Iterable → An iterable object (list, tuple, dictionary)

☞ **any() Returns True if any item in an iterable object is true**

Parameters: (iterable)

r:Iterable → An iterable object (list, tuple, dictionary)

☞ **filter() Use a filter function to exclude items in an iterable object**

Parameters: (function, iterable)

r:Function → A Function to be run for each item in the iterable

r:Iterable → A Function to be run for each item in the iterable

☞ **max() Returns the largest item in an iterable**

Parameters: (iterable)

r:Iterable → An iterable, with one or more items to compare

☞ **min() Returns the smallest item in an iterable**

Parameters: (iterable)

r:Iterable → An iterable, with one or more items to compare

☞ **next() Returns the next item in an iterable**

Parameters: (iterable, defualt)

r:Iterable → An iterable object.

o:Defualt → An default value to return if the iterable has reached to its end.

☞ **sorted() Returns a sorted list**

Parameters: (iterable, key=func, reverse=bool)

r:Iterable → sorted() Returns a sorted list

o:Key → A Function to execute to decide the order. Default is None

o:Reverse → A Boolean. False will sort ascending, True will sort descending. Default is False

Note: str's sort alphabetically, numbers sort numerically
You cannot sort a list that contains both str and numerical values

☞ **enumerate() Takes a collection (e.g. a tuple) and returns it as an enumerate object**

Parameters: (iterable, start)

r:Iterable → an iterable

o:Start → enumerate() Takes a collection (e.g. a tuple) and returns it as an enumerate object

Pertains to Iterators (5)

☞ **iter() Returns an iterator object**

Parameters: (object, sentienel)

r:Object → An iterable object

o:Sentienel → If the object is a callable object the iteration will stop when the returned value is the same as the sentinel

☞ **map() Returns the specified iterator with the specified function applied to each item**

Parameters: (function, iterable)

r:Function → The function to execute for each item

r:Iterable → A sequence, collection or an iterator object. You can send as many iterables as you like, just make sure the function has one parameter for each iterable.

☞ **reversed() Returns a reversed iterator**

Parameters: (sequence)

r:Sequence → any iterable object

☞ **sum() Sums the items of an iterator**

Parameters: (iterable, start)

r:Iterable → the sequence to sum

o:Start → A value that is added to the return value

☞ **zip() Returns an iterator, from two or more iterators**

Parameters: (iterator1, iterator2, iterator3....) [n iterators]

r:iterators → Iterator objects that will be joined together

Pertains to numbers (9)

☞ **abs() Returns the absolute value of a number**

Parameters: (n)

r:N → a number

☞ **round() Rounds a numbers**

Parameters: (number, digits)

r:Number → A number to be rounded

o:Digits → The number of decimals to use when rounding the number, default is 0

☞ **hex() Converts a number into a hexadecimal value**

Parameters: (Number)

r:Number → an Integer

Note: The returned string always starts with the prefix 0x.

☞ **oct() Converts a number into an octal**

Parameters: (number)

r:Number → an Integer

Note: Octal strings in Python are prefixed with 0o.

☞ **range() Returns a sequence of numbers, starting from 0 and increments by 1 (by default)**

Parameters: (start, stop, step)

o:Start → An integer number specifying at which position to start. Default is 0

r:Stop → An integer number specifying at which position to stop (not included).

o:Step → An integer number specifying the incrementation. Default is 1

☞ **pow() Returns the value of x to the power of y**

Parameters: (x, y, z)

r:X → the base

r:Y → the exponent

o:Z → the modulus

☞ **divmod() Returns the quotient and the remainder when argument1 is divided by argument2**

Parameters: (dividend, divisor)

r:Dividend → The number you want to divide

r:Divisor → The number you want to divide with

☞ **ord() Convert an integer representing the Unicode of the specified character**

Parameters: (character)

r:Character → any character

☞ **chr() Returns a character from the specified Unicode code.**

Parameters: (number)

r:Number → An integer representing a valid Unicode code point

Returns Bool (4)

☞ **callable() Returns True if the specified object is callable, otherwise False**

Parameters: (object)

r:Object → The object you want to test if it is callable or not.

☞ **hasattr() Returns True if the specified object has the specified attribute, property/method**

Parameters: (object, attribute)

r:Object → An object

r:Attribute → The name of the attribute you want to check if exists

☞ **isinstance() Returns True if a specified object is an instance of a specified object**

Parameters: (object, type)

r:Object → An object

r:Type → A type or a class, or a tuple of types and/or classes

☞ **issubclass() Returns True if a specified class is a subclass of a specified object**

Parameters: (object, subclass)

r:Object → An object

r:Subclass → A class object, or a tuple of class objects

Returns info (10)

☞ **dir() Returns a list of the specified object's properties and methods**

Parameters: (object)

r:Object → The object you want to see the valid attributes of

☞ **help() Executes the built-in help system**

Parameters: (object)

r:Object → Object you wish to recieve info on example: help(list)

☞ **id() Returns the id of an object**

Parameters: (object)

r:Object → The object you want to see the valid attributes of

Note: The id is the object's memory address, and will be different for each time you run the program. (except for some object that has a constant unique id, like integers from -5 to 256)

☞ **hash() Returns the hash value of a specified object**

Parameters: (object)

r:Object → object you want to get the hash value of

☞ **len() Returns the length of an object**

Parameters: (object)

r:Object → An object. Must be a sequence or a collection

☞ **memoryview() Returns a memory view object**

Parameters: (object)

r:Object → A Bytes object or a Bytearray object.

☞ **getattr() Returns the value of the specified attribute (property or method)**

Parameters: (object, attribute, default)

r:Object → An object

r:Attribute → The name of the attribute you want to get the value from

o:Default → The value to return if the attribute does not exist

☞ **repr() Returns a readable version of an object**

Parameters: (object)

r:Object → returns the canonical string representation of an object

☞ **property() Gets, sets, deletes a property**

Parameters: (fget, fset, fdel, doc)

☞ **locals() Returns an updated dictionary of the current local symbol table**

Parameters: (none)

☞ **globals() Returns the current global symbol table as a dictionary**

Parameters: (none)

Works with objects (4)

☞ **ascii() Returns a readable version of an object. Replaces none-ascii characters with escape character**

Parameters: (object)

r:Object → An object, like String, List, Tuple, Dictionary etc.

☞ **vars() Returns the ___dict___ property of an object**

Parameters: (object)

o:Object → Any object with a ___dict___attribute

Note: calling the vars() function without parameters will return a dictionary containing the local symbol table.
The ___dict___ attribute is a dictionary containing the object's changeable attributes.

☞ **setattr() Sets an attribute (property/method) of an object**

Parameters: (object, attribute, value)

r:Object → An object

r:Attribute → The name of the attribute you want to set

r:Value → The value you want to give the specified attribute

☞ **delattr() Deletes the specified attribute (property or method) from the specified object**

Parameters: (object, attribute)

r:Object → An object

r:Attribute → The name of the attribute you want to remove

All others that return an object (5)

☞ **super() Returns an object that represents the parent class**

Parameters: (None)

☞ **slice() Returns a slice object**

Parameters: (Start, end, step)

o:Start → An integer number specifying at which position to start the slicing. Default is 0

r:Stop → An integer number specifying at which position to end the slicing

o:Step → An integer number specifying the step of the slicing. Default is 1

☞ **open() Opens a file and returns a file object**

Parameters: (file, mode)

r:File → The path and name of the file

o:Mode → A string, define which mode you want to open the file in:

Modes:
"r" - Read - Default value. Opens a file for reading, error if the file does not exist
"a" - Append - Opens a file for appending, creates the file if it does not exist
"w" - Write - Opens a file for writing, creates the file if it does not exist
"x" - Create - Creates the specified file, returns an error if the file exist
In addition you can specify if the file should be handled as binary or text mode
"t" - Text - Default value. Text mode
"b" - Binary - Binary mode (e.g. images)

☞ **object() Returns a new object**

Parameters: (none)

Other (8)

☞ **classmethod() Converts a method into a class method**

Parameters: (function)

r:Function → returns a class method for the given function.

☞ **staticmethod() Converts a method into a static method**

Parameters: (function)

r:Function → The function to convert to static method

☞ **eval() Evaluates and executes an expression**

Parameters: (expression, globals, locals)

r:Expression → that will be evaluated as Python code

o:Globals → A dictionary containing global parameters

o:Locals → A dictionary containing local parameters

☞ **exec() Executes the specified code (or object)**

Parameters: (object, globals, locals)

r:Object → A String, or a code object

o:Globals → A dictionary containing global parameters

o:Locals → A dictionary containing local parameters

Note: The exec() function accepts large blocks of code, unlike the eval() function which only accepts a single expression

☞ **format() Formats a specified value**

Parameters: (value, format)

r:Value → A value of any format

o:Format → The format you want to format the value into.

☞ **input() Allowing user input**

Parameters: (Prompt)

r:Prompt → A String, representing a default message before the input.

☞ **print() Prints to the standard output device**

Parameters: (object(s), sep=*seperator*, end=*end*, file=*file*, flush=*flush*)

r:Objects → Any object, and as many as you like. Will be converted to string before printed

o:Sep → Specify how to separate the objects, if there is more than one. Default is ' '

o:End → Specify what to print at the end. Default is '
n' (line feed)

o:File → An object with a write method. Default is sys.stdout

o:Flush → A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False