List methods

list.append(*x*)
Add an item to the end of the list. Equivalent to a[len(a):] = [x].
list.extend(*iterable*)
Extend the list by appending all the items from the iterable. Equivalent to a[len(a):] = iterable.
list.insert(*i*, *x*)
Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).
list.remove(*x*)
Remove the first item from the list whose value is equal to *x*. It raises a ValueError if there is no such item.
list.pop([*i*])
Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)
list.clear()
Remove all items from the list. Equivalent to del a[:].
list.index(*x*[, *start*[, *end*]])
Return zero-based index in the list of the first item whose value is equal to *x*. Raises a ValueError if there is no such item.
The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.
list.count(*x*)
Return the number of times *x* appears in the list.
list.sort(*key=None*, *reverse=False*)
Sort the items of the list in place (the arguments can be used for sort customization, see sorted() for their explanation).
A simple ascending sort is very easy: just call the sorted() function. It returns a new sorted list:
sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]
list.reverse()
Reverse the elements of the list in place.
list.copy()
Return a shallow copy of the list. Equivalent to a[:].

String methods which both 8-bit strings and Unicode objects support:

capitalize( )
Return a copy of the string with only its first character capitalized.
For 8-bit strings, this method is locale-dependent.
center(width[, fillchar])
Return centered in a string of length width. Padding is done using the specified fillchar (default is a space). Changed in version 2.4: Support for the fillchar argument.
count(sub[, start[, end]])
Return the number of occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.
decode([encoding[, errors]])
Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding. errors may be given to set a different error handling scheme. The default is 'strict', meaning that encoding errors raise UnicodeError. Other possible values are 'ignore', 'replace' and any other name registered via codecs.register_error, see section 4.9.1. New in version 2.2. Changed in version 2.3: Support for other error handling schemes added.
encode([encoding[,errors]])
Return an encoded version of the string. Default encoding is the current default string encoding. errors may be given to set a different error handling scheme. The default for errors is 'strict', meaning that encoding errors raise a UnicodeError. Other possible values are 'ignore', 'replace', 'xmlcharrefreplace', 'backslashreplace' and any other name registered via codecs.register_error, see section 4.9.1. For a list of possible encodings, see section 4.9.2. New in version 2.0. Changed in version 2.3: Support for 'xmlcharrefreplace' and 'backslashreplace' and other error handling schemes added.

endswith(suffix[, start[, end]])

Return True if the string ends with the specified suffix, otherwise return False. With optional start, test beginning at that position. With optional end, stop comparing at that position.

expandtabs([tabsize])

Return a copy of the string where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]])

Return the lowest index in the string where substring sub is found, such that sub is contained in the range [start, end]. Optional arguments start and end are interpreted as in slice notation. Return -1 if sub is not found.

index(sub[, start[, end]])

Like find(), but raise ValueError when the substring is not found.

isalnum( )

Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.
For 8-bit strings, this method is locale-dependent.

isalpha()

Return true if all characters in the string are alphabetic and there is at least one character, false otherwise.
For 8-bit strings, this method is locale-dependent.

isdigit( )

Return true if all characters in the string are digits and there is at least one character, false otherwise.
For 8-bit strings, this method is locale-dependent.

islower( )

Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.
For 8-bit strings, this method is locale-dependent.

isspace()

Return true if there are only whitespace characters in the string and there is at least one character, false otherwise.
For 8-bit strings, this method is locale-dependent.

istitle( )

Return true if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return false otherwise.

isupper( )

Return true if all cased characters in the string are uppercase and there is at least one cased character, false otherwise.

join(seq)

Return a string which is the concatenation of the strings in the sequence seq. The separator between elements is the string providing this method.

ljust(width[, fillchar])

Return the string left justified in a string of length width. Padding is done using the specified fillchar (default is a space). The original string is returned if width is less than len(s). Changed in version 2.4: Support for the fillchar argument.

lower( )

Return a copy of the string converted to lowercase.

lstrip([chars])

Return a copy of the string with leading characters removed. The chars argument is a string specifying the set of characters to be removed. If omitted or None, the chars argument defaults to removing whitespace. The chars argument is not a prefix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.lstrip()
'spacious   '
>>> 'www.example.com'.lstrip('cmowz.')
'example.com'
```

replace(old, new[, count])

Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

rfind(sub [,start [,end]])

Return the highest index in the string where substring sub is found, such that sub is contained within s[start,end]. Optional arguments start and end are interpreted as in slice notation. Return -1 on failure.

rindex(sub[, start[, end]])

Like rfind() but raises ValueError when the substring sub is not found.

rjust(width[, fillchar])

Return the string right justified in a string of length width. Padding is done using the specified fillchar (default is a space). The original string is returned if width is less than len(s). Changed in version 2.4: Support for the fillchar argument.

rsplit([sep [,maxsplit]])

Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done, the rightmost ones. If sep is not specified or None, any whitespace string is a separator. Except for splitting from the right, rsplit() behaves like split() which is described in detail below. New in version 2.4.

rstrip([chars])

Return a copy of the string with trailing characters removed. The chars argument is a string specifying the set of characters to be removed. If omitted or None, the chars argument defaults to removing whitespace. The chars argument is not a suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

split([sep [,maxsplit]])

Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. (thus, the list will have at most maxsplit+1 elements). If maxsplit is not specified, then there is no limit on the number of splits (all possible splits are made). Consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, "'1,,2'.split(',')"returns "['1', '', '2']"). The sep argument may consist of multiple characters (for example, "'1, 2, 3'.split(', ')" returns "['1', '2', '3']"). Splitting an empty string with a specified separator returns "['']".

If sep is not specified or is None, a different splitting algorithm is applied. First, whitespace characters (spaces, tabs, newlines, returns, and formfeeds) are stripped from both ends. Then, words are separated by arbitrary length strings of whitespace characters. Consecutive whitespace delimiters are treated as a single delimiter ("'1 2 3'.split()" returns "['1', '2', '3']"). Splitting an empty string or a string consisting of just whitespace returns an empty list.

splitlines([keepends])

Return a list of the lines in the string, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.

startswith(prefix[, start[, end]])

Return True if string starts with the prefix, otherwise return False. With optional start, test string beginning at that position. With optional end, stop comparing string at that position.

strip([chars])

Return a copy of the string with the leading and trailing characters removed. The chars argument is a string specifying the set of characters to be removed. If omitted or None, the chars argument defaults to removing whitespace. The chars argument is not a prefix or suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

swapcase()

Return a copy of the string with uppercase characters converted to lowercase and vice versa.

title()

Return a titlecased version of the string: words start with uppercase characters, all remaining cased characters are lowercase.

translate(table[, deletechars])

Return a copy of the string where all characters occurring in the optional argument deletechars are removed, and the remaining characters have been mapped through the given translation table, which must be a string of length 256.

upper( )

Return a copy of the string converted to uppercase.

zfill(width)

Return the numeric string left filled with zeros in a string of length width. The original string is returned if width is less than len(s).

close----------

Description

The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.
Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.
Return Value

None
Syntax

```
fileObject.close();
```

get----------

Description

The method **get()** returns a value for the given key. If key is not available then returns default value None.

Syntax

Following is the syntax for **get()** method −

```
dict.get(key, default=None)
```

Parameters

- **key** -- This is the Key to be searched in the dictionary.
- **default** -- This is the Value to be returned in case key does not exist.

Return Value

This method return a value for the given key. If key is not available, then returns default value None.

len----------

Description
The method **len()** returns the number of elements in the *list*.
Syntax
Following is the syntax for **len()** method −

```
len(list)
```

Parameters

- **list** -- This is a list for which number of elements to be counted.

Return Value
This method returns the number of elements in the list.

maketrans----------

Description
The method **maketrans()** returns a translation table that maps each character in the *intabstring* into the character at the same position in the*outtab* string. Then this table is passed to the translate() function.
Note: Both intab and outtab must have the same length.
Syntax
Following is the syntax for **maketrans()** method −

```
str.maketrans(intab, outtab)
```

Parameters

- **intab** -- This is the string having actual characters.
- **outtab** -- This is the string having corresponding mapping character.

Return Value
This method returns a translate table to be used translate() function.

open----------

Description
Before you can read or write a file, you have to open it using Python's built-in*open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Return Value

This method returns a file object.

read----------

Description

The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if *count* is missing, then it tries to read as much as possible, maybe until the end of file.

Return Value

None

readline----------

Description

The method **readline()**reads one entire line from the file. A trailing newline character is kept in the string. If the *size* argument is present and non-negative, it is a maximum byte count including the trailing newline and an incomplete line may be returned.

An empty string is returned only when EOF is encountered immediately.

Syntax

Following is the syntax for **readline()** method –

```
fileObject.readline( size );
```

Parameters

- **size** -- This is the number of bytes to be read from the file.

Return Value

This method returns the line read from the file.

translate

Description

The method **translate()** returns a copy of the string in which all characters have been translated using *table* (constructed with the maketrans() function in the string module), optionally deleting all characters found in the string*deletechars*.

Syntax

Following is the syntax for **translate()** method –

```
str.translate(table[, deletechars]);
```

Parameters

- **table** -- You can use the maketrans() helper function in the string module to create a translation table.
- **deletechars** -- The list of characters to be removed from the source string.

Return Value

This method returns a translated copy of the string.

\

write----------

Description

The *write()* method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string –

Syntax

```
fileObject.write(string);
```

Return Value

None