class str(object)

| str(object='') -> str

| str(bytes\_or\_buffer[, encoding[, errors]]) -> str

|

| Create a new string object from the given object. If encoding or

| errors is specified, then the object must expose a data buffer

| that will be decoded using the given encoding and error handler.

| Otherwise, returns the result of object.\_\_str\_\_() (if defined)

| or repr(object).

| encoding defaults to sys.getdefaultencoding().

| errors defaults to 'strict'.

|

| Methods defined here:

|

| \_\_add\_\_(self, value, /)

| Return self+value.

|

| \_\_contains\_\_(self, key, /)

| Return key in self.

|

| \_\_eq\_\_(self, value, /)

| Return self==value.

|

| \_\_format\_\_(self, format\_spec, /)

| Return a formatted version of the string as described by format\_spec.

|

| \_\_ge\_\_(self, value, /)

| Return self>=value.

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_getitem\_\_(self, key, /)

| Return self[key].

|

| \_\_getnewargs\_\_(...)

|

| \_\_gt\_\_(self, value, /)

| Return self>value.

|

| \_\_hash\_\_(self, /)

| Return hash(self).

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_le\_\_(self, value, /)

| Return self<=value.

|

| \_\_len\_\_(self, /)

| Return len(self).

|

| \_\_lt\_\_(self, value, /)

| Return self<value.

|

| \_\_mod\_\_(self, value, /)

| Return self%value.

|

| \_\_mul\_\_(self, value, /)

| Return self\*value.

|

| \_\_ne\_\_(self, value, /)

| Return self!=value.

|

| \_\_repr\_\_(self, /)

| Return repr(self).

|

| \_\_rmod\_\_(self, value, /)

| Return value%self.

|

| \_\_rmul\_\_(self, value, /)

| Return value\*self.

|

| \_\_sizeof\_\_(self, /)

| Return the size of the string in memory, in bytes.

|

| \_\_str\_\_(self, /)

| Return str(self).

|

| capitalize(self, /)

| Return a capitalized version of the string.

|

| More specifically, make the first character have upper case and the rest lower

| case.

|

| casefold(self, /)

| Return a version of the string suitable for caseless comparisons.

|

| center(self, width, fillchar=' ', /)

| Return a centered string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| count(...)

| S.count(sub[, start[, end]]) -> int

|

| Return the number of non-overlapping occurrences of substring sub in

| string S[start:end]. Optional arguments start and end are

| interpreted as in slice notation.

|

| encode(self, /, encoding='utf-8', errors='strict')

| Encode the string using the codec registered for encoding.

|

| encoding

| The encoding in which to encode the string.

| errors

| The error handling scheme to use for encoding errors.

| The default is 'strict' meaning that encoding errors raise a

| UnicodeEncodeError. Other possible values are 'ignore', 'replace' and

| 'xmlcharrefreplace' as well as any other name registered with

| codecs.register\_error that can handle UnicodeEncodeErrors.

|

| endswith(...)

| S.endswith(suffix[, start[, end]]) -> bool

|

| Return True if S ends with the specified suffix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| suffix can also be a tuple of strings to try.

|

| expandtabs(self, /, tabsize=8)

| Return a copy where all tab characters are expanded using spaces.

|

| If tabsize is not given, a tab size of 8 characters is assumed.

|

| find(...)

| S.find(sub[, start[, end]]) -> int

|

| Return the lowest index in S 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.

|

| format(...)

| S.format(\*args, \*\*kwargs) -> str

|

| Return a formatted version of S, using substitutions from args and kwargs.

| The substitutions are identified by braces ('{' and '}').

|

| format\_map(...)

| S.format\_map(mapping) -> str

|

| Return a formatted version of S, using substitutions from mapping.

| The substitutions are identified by braces ('{' and '}').

|

| index(...)

| S.index(sub[, start[, end]]) -> int

|

| Return the lowest index in S 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.

|

| Raises ValueError when the substring is not found.

|

| isalnum(self, /)

| Return True if the string is an alpha-numeric string, False otherwise.

|

| A string is alpha-numeric if all characters in the string are alpha-numeric and

| there is at least one character in the string.

|

| isalpha(self, /)

| Return True if the string is an alphabetic string, False otherwise.

|

| A string is alphabetic if all characters in the string are alphabetic and there

| is at least one character in the string.

|

| isascii(self, /)

| Return True if all characters in the string are ASCII, False otherwise.

|

| ASCII characters have code points in the range U+0000-U+007F.

| Empty string is ASCII too.

|

| isdecimal(self, /)

| Return True if the string is a decimal string, False otherwise.

|

| A string is a decimal string if all characters in the string are decimal and

| there is at least one character in the string.

|

| isdigit(self, /)

| Return True if the string is a digit string, False otherwise.

|

| A string is a digit string if all characters in the string are digits and there

| is at least one character in the string.

|

| isidentifier(self, /)

| Return True if the string is a valid Python identifier, False otherwise.

|

| Use keyword.iskeyword() to test for reserved identifiers such as "def" and

| "class".

|

| islower(self, /)

| Return True if the string is a lowercase string, False otherwise.

|

| A string is lowercase if all cased characters in the string are lowercase and

| there is at least one cased character in the string.

|

| isnumeric(self, /)

| Return True if the string is a numeric string, False otherwise.

|

| A string is numeric if all characters in the string are numeric and there is at

| least one character in the string.

|

| isprintable(self, /)

| Return True if the string is printable, False otherwise.

|

| A string is printable if all of its characters are considered printable in

| repr() or if it is empty.

|

| isspace(self, /)

| Return True if the string is a whitespace string, False otherwise.

|

| A string is whitespace if all characters in the string are whitespace and there

| is at least one character in the string.

|

| istitle(self, /)

| Return True if the string is a title-cased string, False otherwise.

|

| In a title-cased string, upper- and title-case characters may only

| follow uncased characters and lowercase characters only cased ones.

|

| isupper(self, /)

| Return True if the string is an uppercase string, False otherwise.

|

| A string is uppercase if all cased characters in the string are uppercase and

| there is at least one cased character in the string.

|

| join(self, iterable, /)

| Concatenate any number of strings.

|

| The string whose method is called is inserted in between each given string.

| The result is returned as a new string.

|

| Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

|

| ljust(self, width, fillchar=' ', /)

| Return a left-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| lower(self, /)

| Return a copy of the string converted to lowercase.

|

| lstrip(self, chars=None, /)

| Return a copy of the string with leading whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| partition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string. If the separator is found,

| returns a 3-tuple containing the part before the separator, the separator

| itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing the original string

| and two empty strings.

|

| replace(self, old, new, count=-1, /)

| Return a copy with all occurrences of substring old replaced by new.

|

| count

| Maximum number of occurrences to replace.

| -1 (the default value) means replace all occurrences.

|

| If the optional argument count is given, only the first count occurrences are

| replaced.

|

| rfind(...)

| S.rfind(sub[, start[, end]]) -> int

|

| Return the highest index in S 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(...)

| S.rindex(sub[, start[, end]]) -> int

|

| Return the highest index in S 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.

|

| Raises ValueError when the substring is not found.

|

| rjust(self, width, fillchar=' ', /)

| Return a right-justified string of length width.

|

| Padding is done using the specified fill character (default is a space).

|

| rpartition(self, sep, /)

| Partition the string into three parts using the given separator.

|

| This will search for the separator in the string, starting at the end. If

| the separator is found, returns a 3-tuple containing the part before the

| separator, the separator itself, and the part after it.

|

| If the separator is not found, returns a 3-tuple containing two empty strings

| and the original string.

|

| rsplit(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| Splits are done starting at the end of the string and working to the front.

|

| rstrip(self, chars=None, /)

| Return a copy of the string with trailing whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| split(self, /, sep=None, maxsplit=-1)

| Return a list of the words in the string, using sep as the delimiter string.

|

| sep

| The delimiter according which to split the string.

| None (the default value) means split according to any whitespace,

| and discard empty strings from the result.

| maxsplit

| Maximum number of splits to do.

| -1 (the default value) means no limit.

|

| splitlines(self, /, keepends=False)

| 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(...)

| S.startswith(prefix[, start[, end]]) -> bool

|

| Return True if S starts with the specified prefix, False otherwise.

| With optional start, test S beginning at that position.

| With optional end, stop comparing S at that position.

| prefix can also be a tuple of strings to try.

|

| strip(self, chars=None, /)

| Return a copy of the string with leading and trailing whitespace removed.

|

| If chars is given and not None, remove characters in chars instead.

|

| swapcase(self, /)

| Convert uppercase characters to lowercase and lowercase characters to uppercase.

|

| title(self, /)

| Return a version of the string where each word is titlecased.

|

| More specifically, words start with uppercased characters and all remaining

| cased characters have lower case.

|

| translate(self, table, /)

| Replace each character in the string using the given translation table.

|

| table

| Translation table, which must be a mapping of Unicode ordinals to

| Unicode ordinals, strings, or None.

|

| The table must implement lookup/indexing via \_\_getitem\_\_, for instance a

| dictionary or list. If this operation raises LookupError, the character is

| left untouched. Characters mapped to None are deleted.

|

| upper(self, /)

| Return a copy of the string converted to uppercase.

|

| zfill(self, width, /)

| Pad a numeric string with zeros on the left, to fill a field of the given width.

|

| The string is never truncated.

|

| ----------------------------------------------------------------------

| Static methods defined here:

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| maketrans(x, y=None, z=None, /)

| Return a translation table usable for str.translate().

|

| If there is only one argument, it must be a dictionary mapping Unicode

| ordinals (integers) or characters to Unicode ordinals, strings or None.

| Character keys will be then converted to ordinals.

| If there are two arguments, they must be strings of equal length, and

| in the resulting dictionary, each character in x will be mapped to the

| character at the same position in y. If there is a third argument, it

| must be a string, whose characters will be mapped to None in the result.