String Module - Common string operations

Templates provide simpler string substitutions as described in PEP 292. Instead of the normal "%"-based substitutions, Templates support "$"-based substitutions.

```
from string import Template
s = Template('$who likes $what')
s.substitute(who='tim', what='kung pao')   # 'tim likes kung pao'
d = dict(who='tim')
Template('Give $who $100').substitute(d)
# Traceback (most recent call last):
# [...]
# ValueError: Invalid placeholder in string: line 1, col 10
Template('$who likes $what').substitute(d)
# Traceback (most recent call last):
# [...]
# KeyError: 'what'
Template('$who likes $what').safe_substitute(d)
# 'tim likes $what'
```

capwords(s) - Split the argument into words using split(), capitalize each word using capitalize(), and join the capitalized words using join().

```
capwords("circus clowns take to the streets")
# Circus Clowns Take To The Streets
```

The following are methods of string objects.

capitalize()

```
"circus clowns".capitalize()   # "Circus clowns"
```

center(width[, fillchar]) - Center in a string of given length.

```
'juggler'.center(50)
'                    juggler                     '
'juggler'.center(50,'-')
'--------------------juggler----------------------'
```

count(sub[, start[, end]]) - Return the number of occurrences of substring sub in string S[start:end].

```
"abca".count('a')   # 2
```

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 of Python LibRef.

```
"abc".decode()                          # u'abc'
"abc".decode("utf-8")                   # u'abc'
print 'āb'.decode('ascii', 'ignore')    # b
print 'āb'.decode('ascii', 'replace')   # ??b
```

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.

```
u'abc'.encode()         # 'abc' [default is ascii]
u'abc'.encode("utf-8")  # 'abc'
```

endswith(suffix[, start[, end]]) - Return True if the string ends with the specified suffix, otherwise return False. startswith() is the same but from beginning of the string.

```
"abc".endswith("bc")   # True
"abc".endswith("zz")   # False
```

expandtabs([tabsize]) - Return a copy of the string where all tab characters are expanded using spaces.

```
'\tabc'.expandtabs(4)   # "    abc"
```

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].

```
"abcdef".find("b")   # 1
"abcdef".find("e")   # 4
"abcdef".find("z")   # -1
```

index(sub[, start[, end]]) - Like find(), but raise ValueError when the substring is not found.

```
"abcdef".index("z")   # [ValueError raised]
```

isalnum() - Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.

```
"abc4".isalnum()    # True
"1234".isalnum()    # True
";".isalnum()       # False
" ".isalnum()       # False
"".isalnum()        # False
```

isalpha() - Return true if all characters in the string are alphabetic and there is at least one character, false otherwise.

```
"abc".isalpha()     # True
"abc4".isalpha()    # False
" ".isalpha()       # False
"".isalpha()        # False
```

isdigit() - Return true if all characters in the string are digits and there is at least one character, false otherwise.

```
"123".isdigit()     # True
"abc".isdigit()     # False
"".isdigit()        # False
```

islower() - Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.

```
"abc".islower()   # True
"123".islower()   # False
"Abc".islower()   # False
```

isspace() - Return true if there are only whitespace characters in the string and there is at least one character, false otherwise.

```
"abc".isspace()     # False
" ".isspace()       # True
"\t \n".isspace()   # True
"".isspace()        # False
```

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.

```
"Abc".istitle()         # True
"abc".istitle()         # False
"Abc Abc".istitle()     # True
"Abc abc".istitle()     # False
```

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

```
"".isupper()        # False
"Abc".isupper()     # False
"ABC".isupper()     # True
```

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.

```
",".join("abc")   # "a,b,c"
```

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). rjust() is the same but justifies on the right side.

```
"abc".ljust(5)   # "abc  "
"abc".ljust(2)   # "abc"
```

lower() - Return a copy of the string converted to lowercase. upper() is the same but converts to upper case.

```
"ABcd".lower()   # "abcd"
```

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. rstrip() is the same but on the right side.

```
'   spacious   '.lstrip()          # 'spacious   '
'..,example..,'.lstrip('.,')       # 'example..,'
```

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.

```
"monster".replace("o", "ue")   # "muenster"
"baobab".replace("b", "t", 1)  # "taobab"
```

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() is the same but raises ValueError if substring is not found.

```
"juggler".rfind("le")   # 4
"juggler".rfind("ze")   # -1
```

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. If sep is not specified or None, any whitespace string is a separator. Except for splitting from the right, rsplit() behaves like split().

```
"a b c".split()                # ['a', 'b', 'c']
"a,b,c".split(",")             # ['a', 'b', 'c']
"".split()                     # []
"a b c d e f".split(" ", 2)    # ['a', 'b', 'c d e f']
"a b c d e f".rsplit(" ", 2)   # ['a b c d', 'e', 'f']
```

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.

```
'a\nb\nc\n'.splitlines()        # ['a', 'b', 'c']
'a\nb\nc\n'.splitlines(True)    # ['a\n', 'b\n', 'c\n']
```

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'
'..example,,'.strip('.,')         # 'example'
```

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

```
"tRaPeze".swapcase()   # "TrApEZE"
```

title() - Return a titlecased version of the string: words start with uppercase characters, all remaining cased characters are lowercase. Unlike string.capwords this function will lowercase the rest of each word.

```
'tigers jumping through hoops'.title()
# 'Tigers Jumping Through Hoops'
'tigErs jUmping througH hooPS'.title()
# 'Tigers Jumping Through Hoops'
```

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. For Unicode objects, the translate() method does not accept the optional deletechars argument. Instead, it returns a copy of the s where all characters have been mapped through the given translation table which must be a mapping of Unicode ordinals to Unicode ordinals, Unicode strings or None. Unmapped characters are left untouched. Characters mapped to None are deleted. Note, a more flexible approach is to create a custom character mapping codec using the codecs module (see encodings.cp1251 for an example).

```
tbl = maketrans("ab", "yz")
"about".translate(tbl)          # "yzout"
```

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).

```
"5".zfill(3) + ".txt"   # "005.txt"
"5:00".zfill(5)         # "05:00"
```