```
str.capitalize()
Return a copy of the string with its first character capitalized and the rest
    lowercased.

str.count(sub[, start[, end]])
Return the number of non-overlapping occurrences of substring sub in the range
     [start, end]. Optional arguments start and end are interpreted as in
    slice notation.


str.endswith(suffix[, start[, end]])
Return True if the string ends with the specified suffix, otherwise return
    False. suffix can also be a tuple of suffixes to look for. With optional
    start, test beginning at that position. With optional end, stop comparing
    at that position.

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

str.isalnum()
Return true if all characters in the string are alphanumeric and there is at
    least one character, false otherwise. A character c is alphanumeric if one
     of the following returns True: c.isalpha(), c.isdecimal(), c.isdigit(),
    or c.isnumeric().

str.isalpha()
Return true if all characters in the string are alphabetic and there is at
    least one character, false otherwise. Alphabetic characters are those
    characters defined in the Unicode character database as Letter, i.e
    ., those with general category property being one of Lm, Lt,
     Lu, Ll, or Lo. Note that this is different from the
     Alphabetic property defined in the Unicode Standard.

str.isdecimal()
Return true if all characters in the string are decimal characters and there
    is at least one character, false otherwise. Decimal characters are those
    from general category Nd. This category includes digit characters,
    and all characters that can be used to form decimal-radix numbers, e.g. U
    +0660, ARABIC-INDIC DIGIT ZERO.

str.isdigit()
Return true if all characters in the string are digits and there is at least
    one character, false otherwise. Digits include decimal characters and
    digits that need special handling, such as the compatibility superscript
    digits. Formally, a digit is a character that has the property value
    Numeric_Type=Digit or Numeric_Type=Decimal.

str.islower()
Return true if all cased characters [4] in the string are lowercase and there
    is at least one cased character, false otherwise.
```

```
str.isnumeric()
Return true if all characters in the string are numeric characters, and there
    is at least one character, false otherwise. Numeric characters include
    digit characters, and all characters that have the Unicode numeric value
    property, e.g. U+2155, VULGAR FRACTION ONE FIFTH. Formally, numeric
    characters are those with the property value Numeric_Type=Digit,
    Numeric_Type=Decimal or Numeric_Type=Numeric.

str.isspace()
Return true if there are only whitespace characters in the string and there is
     at least one character, false otherwise. Whitespace characters are those
    characters defined in the Unicode character database as   Other   or
     Separator   and those with bidirectional property being one of
     WS   ,   B   , or   S   .

str.isupper()
Return true if all cased characters [4] in the string are uppercase and there
    is at least one cased character, false otherwise.

str.join(iterable)
Return a string which is the concatenation of the strings in the iterable
    iterable. A TypeError will be raised if there are any non-string values in
     iterable, including bytes objects. The separator between elements is the
    string providing this method.

str.lower()
Return a copy of the string with all the cased characters [4] converted to
    lowercase.

str.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.

str.split(sep=None, maxsplit=-1)
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 or
    -1, then there is no limit on the number of splits (all possible splits
    are made).

If sep is given, 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
    : runs of consecutive whitespace are regarded as a single separator, and
    the result will contain no empty strings at the start or end if the string
     has leading or trailing whitespace. Consequently, splitting an empty
```

```
    string or a string consisting of just whitespace with a None separator
    returns [].

str.startswith(prefix[, start[, end]])
Return True if string starts with the prefix, otherwise return False. prefix
    can also be a tuple of prefixes to look for. With optional start, test
    string beginning at that position. With optional end, stop comparing
    string at that position.

str.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:

str.upper()
Return a copy of the string with all the cased characters [4] converted to
    uppercase. Note that str.upper().isupper() might be False if s contains
    uncased characters or if the Unicode category of the resulting character(s
    ) is not    Lu    (Letter, uppercase), but e.g.    Lt    (Letter,
    titlecase).
```

```
my_int = 42
my_str = "the answer to life the universe and everything"
my_float = 3.14
print("%d is %s, not %f" % (my_int, my_str, my_float))
```

`dict` methods

```
clear()
Remove all items from the dictionary.

copy()
Return a shallow copy of the dictionary.

get(key[, default])
Return the value for key if key is in the dictionary, else default. If default
     is not given, it defaults to None, so that this method never raises a
    KeyError.

items()
Return a new view of the dictionary's items ((key, value) pairs).

keys()
Return a new view of the dictionary's keys. .

pop(key[, default])
If key is in the dictionary, remove it and return its value, else return
    default. If default is not given and key is not in the dictionary, a
```

```
    KeyError is raised.

popitem()
Remove and return an arbitrary (key, value) pair from the dictionary.

popitem() is useful to destructively iterate over a dictionary, as often used
    in set algorithms. If the dictionary is empty, calling popitem() raises a
    KeyError.

setdefault(key[, default])
If key is in the dictionary, return its value. If not, insert key with a value
     of default and return default. default defaults to None.

update([other])
Update the dictionary with the key/value pairs from other, overwriting
    existing keys. Return None.

update() accepts either another dictionary object or an iterable of key/value
    pairs (as tuples or other iterables of length two). If keyword arguments
    are specified, the dictionary is then updated with those key/value pairs:
    d.update(red=1, blue=2).

values()
Return a new view of the dictionary's values.
```

x in s True if an item of s is equal to x, else False
x not in s False if an item of s is equal to x, else True
s + t the concatenation of s and t
s * n or n * s n shallow copies of s concatenated
s[i] ith item of s, origin 0
s[i:j] slice of s from i to j
s[i:j:k] slice of s from i to j with step k
len(s) length of s
min(s) smallest item of s
max(s) largest item of s
s.index(x[, i[, j]]) index of the first occurrence of x in s (at or after index i and before index j)
s.count(x) total number of occurrences of x in s
s[i] = x item i of s is replaced by x
s[i:j] = t slice of s from i to j is replaced by the contents of the iterable t
del s[i:j] same as s[i:j] = []
s[i:j:k] = t the elements of s[i:j:k] are replaced by those of t
del s[i:j:k] removes the elements of s[i:j:k] from the list
s.append(x) appends x to the end of the sequence (same as s[len(s):len(s)] = [x])
s.clear() removes all items from s (same as del s[:])
s.copy() creates a shallow copy of s (same as s[:])
s.extend(t) extends s with the contents of t (same as s[len(s):len(s)] = t)
s.insert(i, x) inserts x into s at the index given by i (same as s[i:i] = [x])
s.pop([i]) retrieves the item at i and also removes it from s
s.remove(x) remove the first item from s where s[i] == x
s.reverse() reverses the items of s in place

```
L = [3, 2, 10]
L.sort() #L is now [2, 3, 10]
L = [3, 2, 10]
L1 = sorted(L) #L1 is now [2, 3, 10], a new list
L = [3, 2, 10]
L2 = sorted(L, reverse=True) #L2 is now [10, 3, 2], a new list
```

The following examples all return a dictionary equal to `{"one": 1, "two": 2, "three": 3}`

```
{'one': 1, 'two': 2, 'three': 3}
dict([('two', 2), ('one', 1), ('three', 3)])
```

Also:

```
>> list({"one": 1, "two": 2, "three": 3}.items())
[('three', 3), ('two', 2), ('one', 1)]
```

$$1 + 2 + 3 + 4 + ... + n = \frac{n(n+1)}{2}$$

$$1 + r + r^2 + r^3 + ... + r^n = \frac{1 - r^{n+1}}{1 - r}$$

$$n = a^{log_a n}$$

```
The matrix is currently:
[[ 0.  0.  1.  0.  2.]
 [ 1.  0.  2.  3.  4.]
 [ 3.  0.  4.  2.  1.]
 [ 1.  0.  1.  1.  2.]]
Now looking at row 0
Swapping rows 0 and 1 so that entry 0 in the current row is non-zero
The matrix is currently:
[[ 1.  0.  2.  3.  4.]
 [ 0.  0.  1.  0.  2.]
 [ 3.  0.  4.  2.  1.]
 [ 1.  0.  1.  1.  2.]]
Adding row 0 to rows below it to eliminate coefficients in column 0
The matrix is currently:
[[  1.   0.   2.   3.   4.]
 [  0.   0.   1.   0.   2.]
 [  0.   0.  -2.  -7. -11.]
 [  0.   0.  -1.  -2.  -2.]]
================================================================================
Now looking at row 1
Swapping rows 1 and 1 so that entry 2 in the current row is non-zero
The matrix is currently:
[[  1.   0.   2.   3.   4.]
 [  0.   0.   1.   0.   2.]
 [  0.   0.  -2.  -7. -11.]
 [  0.   0.  -1.  -2.  -2.]]
Adding row 1 to rows below it to eliminate coefficients in column 2
The matrix is currently:
[[ 1.  0.  2.  3.  4.]
 [ 0.  0.  1.  0.  2.]
 [ 0.  0.  0. -7. -7.]
 [ 0.  0.  0. -2.  0.]]
================================================================================
Now looking at row 2
Swapping rows 2 and 2 so that entry 3 in the current row is non-zero
The matrix is currently:
[[ 1.  0.  2.  3.  4.]
 [ 0.  0.  1.  0.  2.]
 [ 0.  0.  0. -7. -7.]
 [ 0.  0.  0. -2.  0.]]
Adding row 2 to rows below it to eliminate coefficients in column 3
The matrix is currently:
[[ 1.  0.  2.  3.  4.]
 [ 0.  0.  1.  0.  2.]
 [ 0.  0.  0. -7. -7.]
 [ 0.  0.  0.  0.  2.]]
================================================================================
Now looking at row 3
Swapping rows 3 and 3 so that entry 4 in the current row is non-zero
The matrix is currently:
[[ 1.  0.  2.  3.  4.]
```

```
 [ 0.  0.  1.  0.  2.]
 [ 0.  0.  0. -7. -7.]
 [ 0.  0.  0.  0.  2.]]
Adding row 3 to rows below it to eliminate coefficients in column 4
The matrix is currently:
[[ 1.  0.  2.  3.  4.]
 [ 0.  0.  1.  0.  2.]
 [ 0.  0.  0. -7. -7.]
 [ 0.  0.  0.  0.  2.]]
================================================================================
Done with the forward step
The matrix is currently:
[[ 1.  0.  2.  3.  4.]
 [ 0.  0.  1.  0.  2.]
 [ 0.  0.  0. -7. -7.]
 [ 0.  0.  0.  0.  2.]]
```