```
>>> s = "hello!"
>>> s.upper()
'HELLO!'
>>> s
'hello!'
>>> s.count("l") # thats a lowercase L, not a 1
2
>>> s.count("h")
1
>>> greeting = "good morning"
>>> greeting.replace("good", "great")
'great morning'
>>> greeting.replace("g", "G")
'Good morninG'
>>> greeting.replace("g", "G").count("G")
```

New: calling methods

documented with the type of calling object, in this case str.

`str.upper()`

Produces a new string with all the letters in the string in uppercase.

`str.count(tofind)`

Takes a string tofind and produces the number of times tofind appears in this string.

`str.replace(toreplace, replacewith)`

Takes two strings toreplace and replacewith, and produces a new string with all instances of toreplace changed to replacewith

`str.startswith(prefix)`

Takes a string prefix and returns True if the string starts with prefix

calling object or **calling value** **method name** **argument(s)**

```
>>> s.count("h")
1
```

return value or **result** or **answer**

This entire **expression** is a **method call** or a **use of a method**.

The **arguments** can also be expressions.

The **calling object** can also be an expression.

```
>>> s = "hello!"
>>> # Write a single expression that uses both a
>>> # method call and a function call
>>>
```

```
# Write a function shout that takes a string and
# returns a new string with the original string in
# uppercase, with an exclamation point
# added at the end.
```

```
# Write a function is_hashtag that takes a string
# and returns true if the string is longer than
# 4 characters and starts with a # symbol.
```

New: assert statements

example call or **test call** **expected result**

```
assert shout("hi") == "HI!"
```

This is an **assert statement**. We put it in a **code file** after a **function definition**. It reports an error if the assertion doesn't evaluate to True, after which we can investigate what's wrong.

This is a form of **testing** or **example writing** to document and check our work.

```
== RESTART ...
>>> [42, 57, 3]
[42, 57, 3]
>>> nums = [5, 6, 7, 2]
>>> nums
[5, 6, 7, 2]
>>> strs = ["cse8a", "cse8b", "cse12"]
>>> strs
['cse8a', 'cse8b', 'cse12']
>>> strs[0]
'cse8a'
>>> nums[0]
5
>>> nums[1]


>>> strs[2]


>>> sentence = "Welcome to lists"
>>> sentence.split(" ")
['Welcome', 'to', 'lists']
```

New: Lists

We can create lists with **list expressions** or **list literals**:

`[42, 57, 3]`

Each position between commas can also be an expression.

Typically we make all the elements in the list have the same type (all numbers, all strings, etc)

```
# write a function average that takes a list of numbers
# and produces their average (mean)
```

`str.split(sep)`

Takes a string sep and returns a **list** of the strings in between instances of sep in this string.

`sum(lst)`

Takes a list of numbers and produces their sum

`len(lst)`

Takes a list and produces its length

```
== RESTART ...
>>> nums = [5, 6, -7, 2]
>>> words = ["the", "it", "their", "a", "whose"]
>>> list(map(square, nums))
[25, 36, 49, 4]
>>>
>>> list(filter(is_long_word, words))
['their', 'whose']
 >>> list(filter(is_pos, nums))
[5, 6, 2]
>>> # use map to create a list of shouted words
>>>


>>> # create a list of just the long words, shouted
>>>
```

```
def square(x): return x * x
def is_pos(n): return n > 0
def is_long_word(s): return len(s) > 4
def shout(s): return s.upper() + "!"




# Challenge: write a function that takes a string
# and returns a list of the hashtags in that string
```

```
list(map(square, [5, 6, -7, 2]))

      ⇒ [square(5), square(6), square(-7), square(4)]

        ⇒ [25, 36, 49, 4]
```

map calls a function on every element of a list, and makes a new list with the results

```
list(filter(is_pos, [     5,        6,        -7,        2])

          ⇒ [is_pos(5), is_pos(6), is_pos(-7), is_pos(2)]

          ⇒ [     True,      True,      False,      True]

          ⇒ [        5,         6,                     2]
```

filter calls a function on every element of a list, and makes a new list of just the elements where the function returned True