# Introduction to

python

## for scientific computing

### - Lecture 4

```
In [ ]:
```

## Exercise recap!

- IMDb
- Blocket

```
!head -2 ../downloads/250.imdb
```

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
    126807|   8.5|1957|5280|https://images-na.ssl-images-amazon.com/images/M/M
V5BOTI5Nzc0OTMtYzBkMS00NjkxLThmM2UtNjM2ODgxN2M5NjNkXkEyXkFqcGdeQXVyNjQ2MjQ5NzM
@._V1_.jpg|Drama,War|Paths of Glory
```

In [10]:

```
movies_file = open("../downloads/250.imdb")

for line in movies_file:
    if not line[0] == "#":
        line_strip = line.strip()
        line_split = line_strip.split("|")
        genres = line_split[5]
        if "Adventure" in genres or "adventure" in genres:
            rating = float(line_split[1])
            if rating > 8.5:
                timing_seconds = int(line_split[3])
                timing_minutes = round(timing_seconds / 60)
                title = line_split[6]
                print("The movie " + title + " is " + str(timing_minutes) + " mi
nutes long.")
```

```
The movie Inception is 148 minutes long.
The movie The Lord of the Rings: The Two Towers is 179 minutes long.
The movie Star Wars: Episode V - The Empire Strikes Back is 124 minutes long.
The movie The Lord of the Rings: The Fellowship of the Ring is 178 minutes lon
g.
The movie The Lord of the Rings: The Return of the King is 201 minutes long.
The movie Seven Samurai is 158 minutes long.
The movie Star Wars: Episode IV - A New Hope is 121 minutes long.
The movie Spirited Away is 125 minutes long.
The movie Interstellar is 169 minutes long.
```

## More useful functions and methods

**What is the difference between a `function` and a `method` ?**

A `method` always belongs to an object of a specific class, a `function` does not have to. For example:

`print('a string')` and `print(42)` both works, even though one is a string and one is an integer

`'a string '.strip()` works, but `[1,2,3,4].strip()` does not work.
`strip()` is a method that only works on strings

## What does it matter to me?

For now, you mostly need to be aware of the difference, and know the different syntaxes:

**A function:**

```
functionName()
```

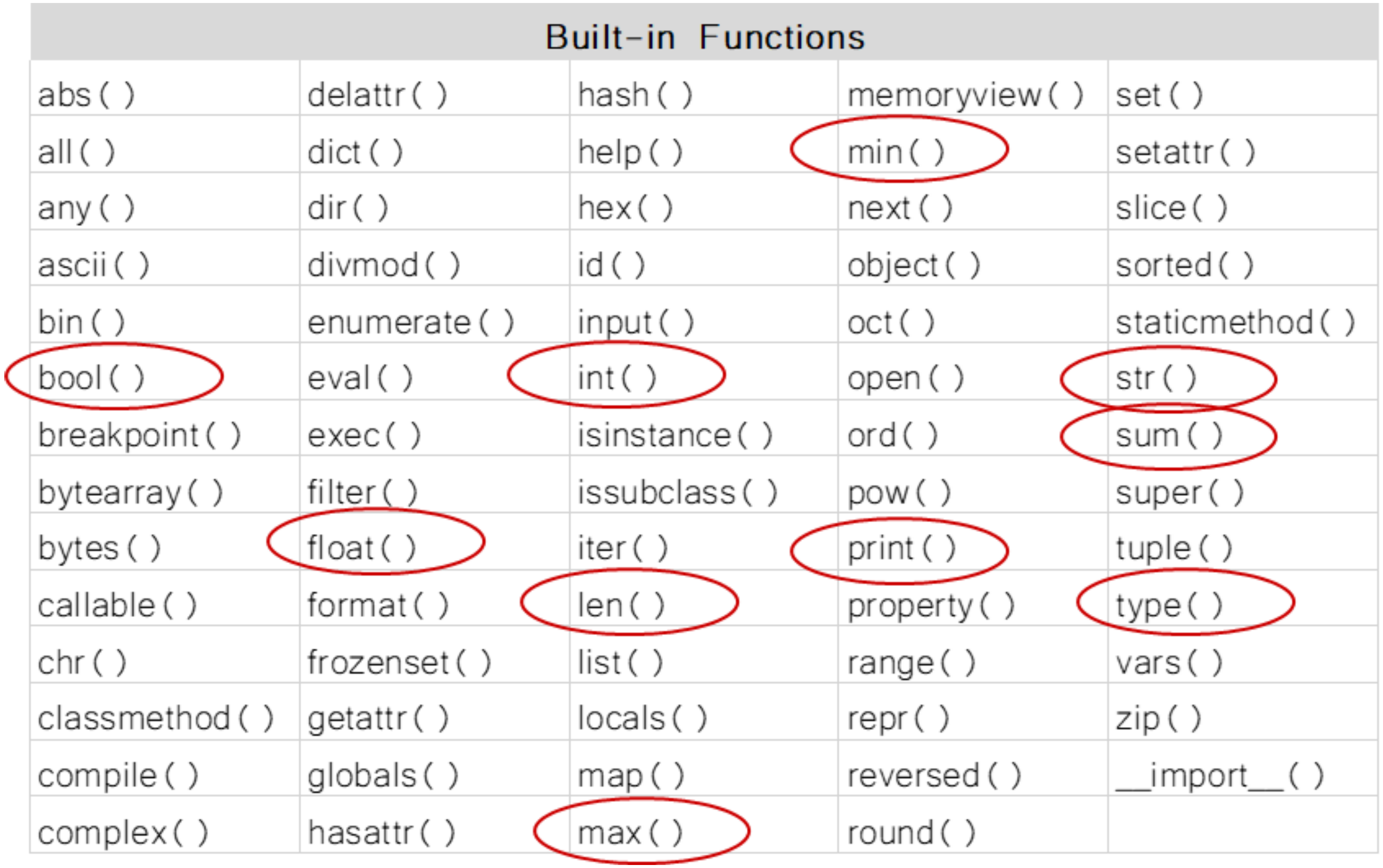
**A method:**

```
<object>.methodName()
```

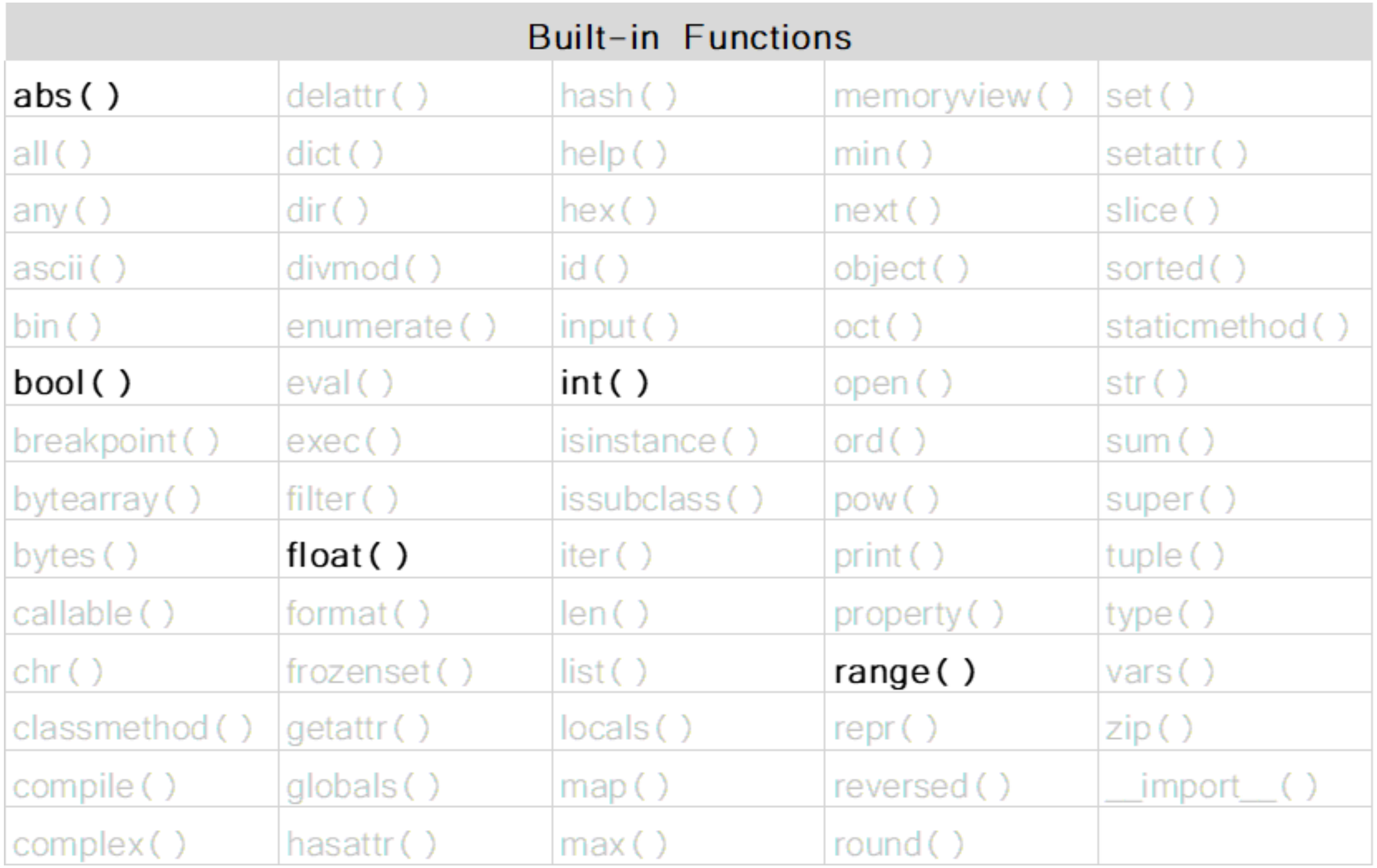
```python
#len([1,2,3])
#len('a string')
l = [1, 2, 3]
len(l)
#strip("a string")
#'a string  '.strip()
[1,2,3].strip()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [15], in <cell line: 7>()
      4 len(l)
      5 #strip("a string")
      6 #'a string  '.strip()
----> 7 [1,2,3].strip()

AttributeError: 'list' object has no attribute 'strip'
```

# Functions



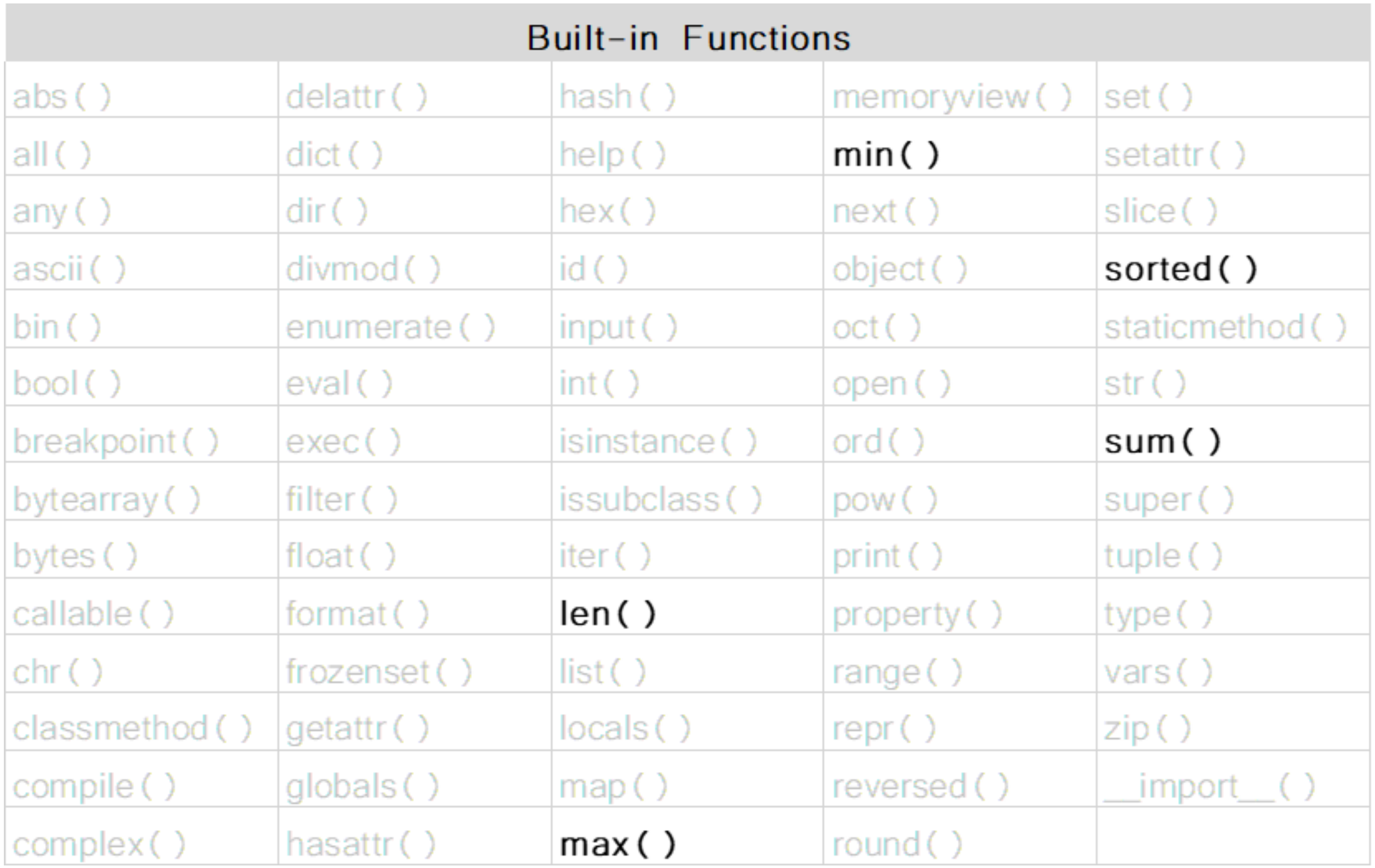
| Built-in Functions | | | | |
|---|---|---|---|---|
| abs( ) | delattr( ) | hash( ) | memoryview( ) | set( ) |
| all( ) | dict( ) | help( ) | [✓ min( )] | setattr( ) |
| any( ) | dir( ) | hex( ) | next( ) | slice( ) |
| ascii( ) | divmod( ) | id( ) | object( ) | sorted( ) |
| bin( ) | enumerate( ) | input( ) | oct( ) | staticmethod( ) |
| [✓ bool( )] | eval( ) | [✓ int( )] | open( ) | [✓ str( )] |
| breakpoint( ) | exec( ) | isinstance( ) | ord( ) | [✓ sum( )] |
| bytearray( ) | filter( ) | issubclass( ) | pow( ) | super( ) |
| bytes( ) | [✓ float( )] | iter( ) | [✓ print( )] | tuple( ) |
| callable( ) | format( ) | [✓ len( )] | property( ) | [✓ type( )] |
| chr( ) | frozenset( ) | list( ) | range( ) | vars( ) |
| classmethod( ) | getattr( ) | locals( ) | repr( ) | zip( ) |
| compile( ) | globals( ) | map( ) | reversed( ) | __import__( ) |
| complex( ) | hasattr( ) | [✓ max( )] | round( ) | |



| Built-in Functions | | | | |
|---|---|---|---|---|
| **abs( )** | delattr( ) | hash( ) | memoryview( ) | set( ) |
| all( ) | dict( ) | help( ) | min( ) | setattr( ) |
| any( ) | dir( ) | hex( ) | next( ) | slice( ) |
| ascii( ) | divmod( ) | id( ) | object( ) | sorted( ) |
| bin( ) | enumerate( ) | input( ) | oct( ) | staticmethod( ) |
| **bool( )** | eval( ) | **int( )** | open( ) | str( ) |
| breakpoint( ) | exec( ) | isinstance( ) | ord( ) | sum( ) |
| bytearray( ) | filter( ) | issubclass( ) | pow( ) | super( ) |
| bytes( ) | **float( )** | iter( ) | print( ) | tuple( ) |
| callable( ) | format( ) | len( ) | property( ) | type( ) |
| chr( ) | frozenset( ) | list( ) | **range( )** | vars( ) |
| classmethod( ) | getattr( ) | locals( ) | repr( ) | zip( ) |
| compile( ) | globals( ) | map( ) | reversed( ) | __import__( ) |
| complex( ) | hasattr( ) | max( ) | round( ) | |

```
In [26]: list(range(2,10,2))
```

```
Out[26]: [2, 4, 6, 8]
```

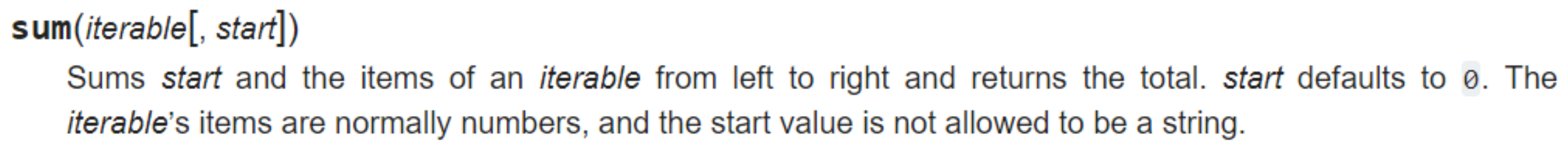


| Built-in Functions | | | | |
|---|---|---|---|---|
| abs( ) | delattr( ) | hash( ) | memoryview( ) | set( ) |
| all( ) | dict( ) | help( ) | **min( )** | setattr( ) |
| any( ) | dir( ) | hex( ) | next( ) | slice( ) |
| ascii( ) | divmod( ) | id( ) | object( ) | **sorted( )** |
| bin( ) | enumerate( ) | input( ) | oct( ) | staticmethod( ) |
| bool( ) | eval( ) | int( ) | open( ) | str( ) |
| breakpoint( ) | exec( ) | isinstance( ) | ord( ) | **sum( )** |
| bytearray( ) | filter( ) | issubclass( ) | pow( ) | super( ) |
| bytes( ) | float( ) | iter( ) | print( ) | tuple( ) |
| callable( ) | format( ) | **len( )** | property( ) | type( ) |
| chr( ) | frozenset( ) | list( ) | range( ) | vars( ) |
| classmethod( ) | getattr( ) | locals( ) | repr( ) | zip( ) |
| compile( ) | globals( ) | map( ) | reversed( ) | __import__( ) |
| complex( ) | hasattr( ) | **max( )** | round( ) | |

In [32]:
```
l = [1, 2, 3.3, 0, -1]
sorted(l)

help(sorted)
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending orde
r.

    A custom key function can be supplied to customize the sort order, and the
```

## From Python documentation

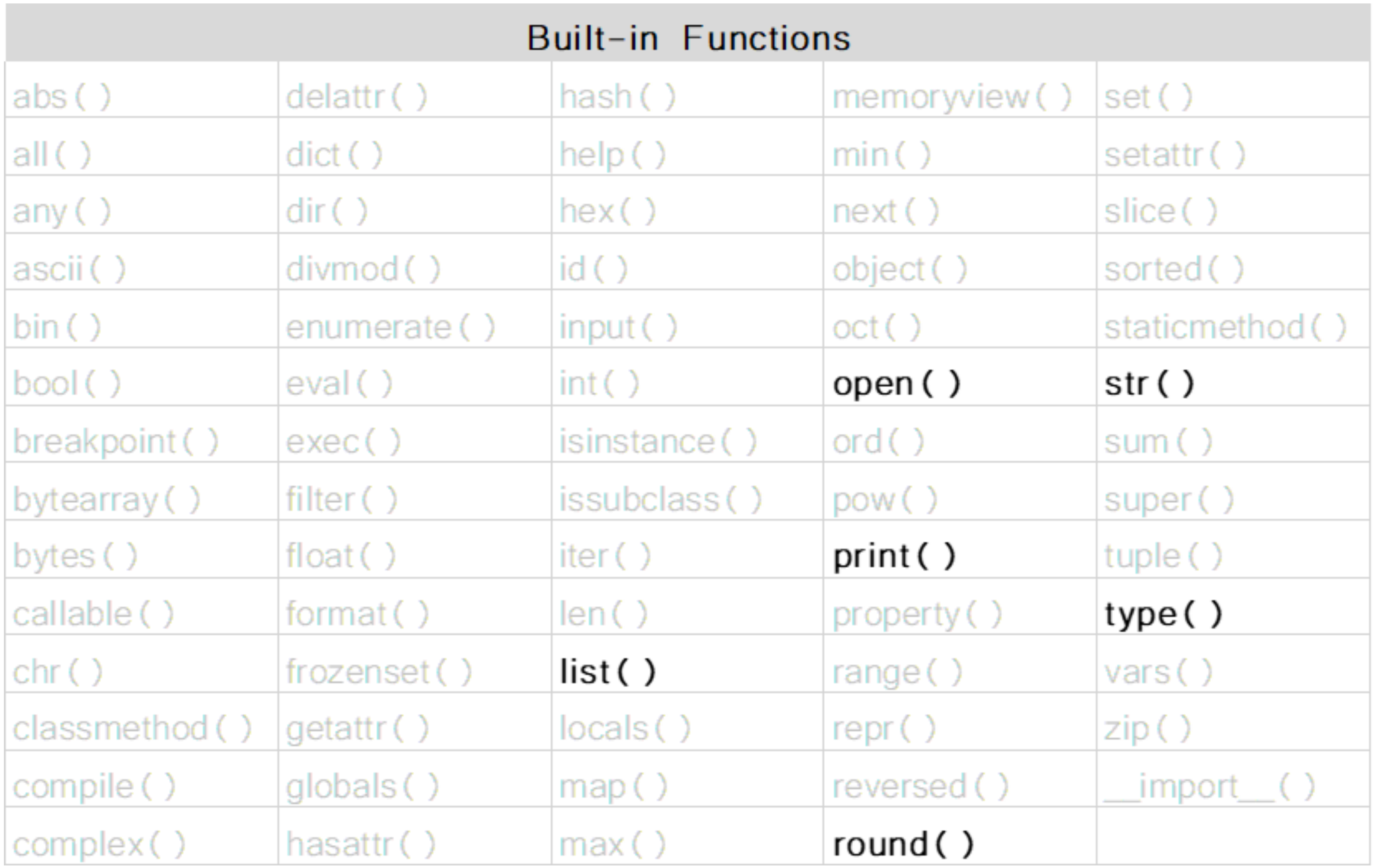


**sum**(*iterable*[, *start*])

Sums *start* and the items of an *iterable* from left to right and returns the total. *start* defaults to `0`. The *iterable*'s items are normally numbers, and the start value is not allowed to be a string.

```
In [35]: sum([1,2,3,4], 4)
         #help(sum)
```

```
Out[35]: 14
```



Built-in Functions

| | | | | |
|---|---|---|---|---|
| abs ( ) | delattr ( ) | hash ( ) | memoryview ( ) | set ( ) |
| all ( ) | dict ( ) | help ( ) | min ( ) | setattr ( ) |
| any ( ) | dir ( ) | hex ( ) | next ( ) | slice ( ) |
| ascii ( ) | divmod ( ) | id ( ) | object ( ) | sorted ( ) |
| bin ( ) | enumerate ( ) | input ( ) | oct ( ) | staticmethod ( ) |
| bool ( ) | eval ( ) | int ( ) | **open ( )** | **str ( )** |
| breakpoint ( ) | exec ( ) | isinstance ( ) | ord ( ) | sum ( ) |
| bytearray ( ) | filter ( ) | issubclass ( ) | pow ( ) | super ( ) |
| bytes ( ) | float ( ) | iter ( ) | **print ( )** | tuple ( ) |
| callable ( ) | format ( ) | len ( ) | property ( ) | **type ( )** |
| chr ( ) | frozenset ( ) | **list ( )** | range ( ) | vars ( ) |
| classmethod ( ) | getattr ( ) | locals ( ) | repr ( ) | zip ( ) |
| compile ( ) | globals ( ) | map ( ) | reversed ( ) | __import__ ( ) |
| complex ( ) | hasattr ( ) | max ( ) | **round ( )** | |

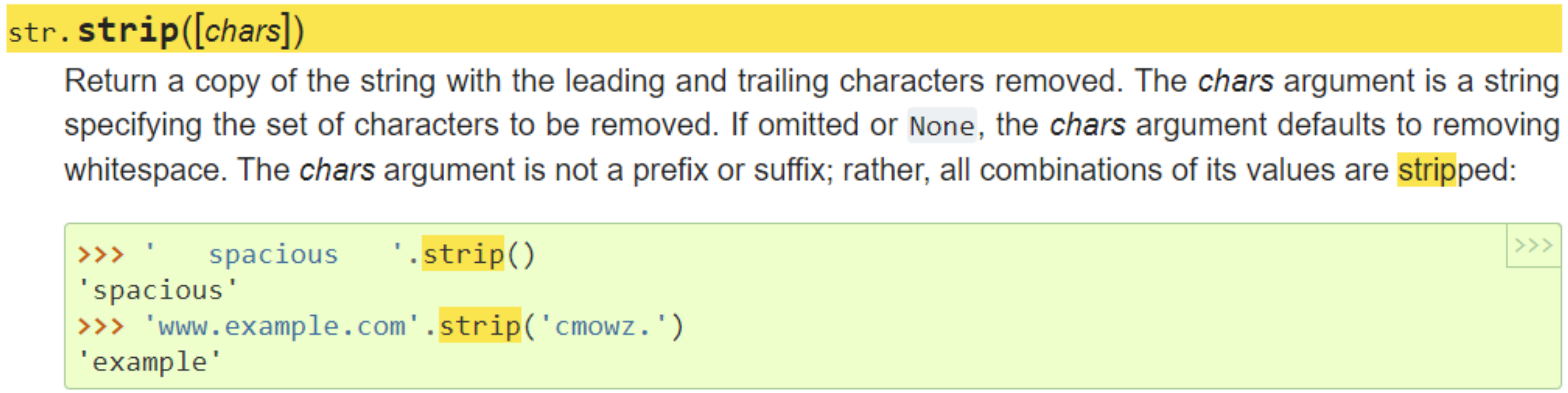
In [39]:
```
help(str.strip)
```

```
Help on method_descriptor:
```

## Methods

### Useful operations on strings

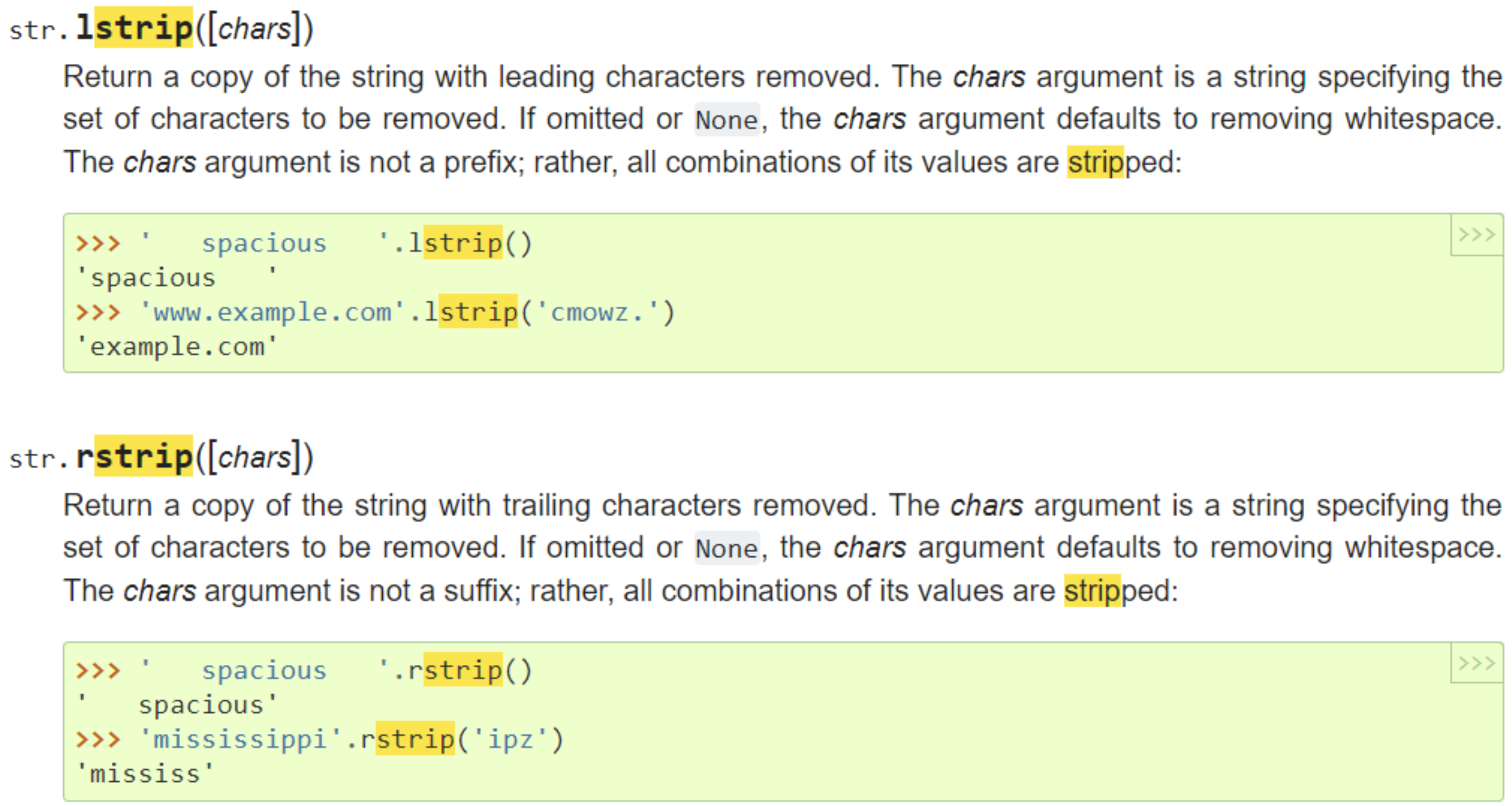
| String Methods | |
|---|---|
| str.strip( ) | str.startswith( ) |
| str.rstrip( ) | str.endswith( ) |
| str.lstrip( ) | str.upper( ) |
| str.split( ) | str.lower( ) |
| str.join( ) | |



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:

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



str.**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'
```

str.**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'
```

```
In [40]: '   spaciousWith5678.com'.strip('mco')

Out[40]: '   spaciousWith5678.'
```

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 `['']`.

For example:

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
```

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 `[]`.

For example:

```
>>> '1 2 3'.split()
['1', '2', '3']
>>> '1 2 3'.split(maxsplit=1)
['1', '2 3']
>>> '   1   2   3   '.split()
['1', '2', '3']
```

In [41]:
```
a = '  split a string into a list '
a.split(maxsplit=3)
```

Out[41]:
```
['split', 'a', 'string', 'into a list ']
```

`str.`**`join`**(*iterable*)

Return a string which is the concatenation of the strings in *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.

```
In [45]: #" ".join(["make", "this", "list", "into", "a", "string"])
         ",".join(["a", "b", "c", "d"])
```

```
Out[45]: 'a,b,c,d'
```

`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.`**`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.

In [49]:
```
"long string".startswith("ng")
element = "long string"
element.endswith("string")
```

Out[49]:
```
True
```

`str.`**`upper`**`()`

Return a copy of the string with all the cased characters [4] converted to uppercase. Note that `s.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).

`str.`**`lower`**`()`

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

```
In [51]: "LongRandomString".lower()
         "LongRandomString".upper()
```

```
Out[51]: 'LONGRANDOMSTRING'
```

## Useful operations on Mutable sequences

| Operation | Result |
|---|---|
| s.append(x) | appends *x* to the end of the sequence |
| s.insert(i, x) | *x* is inserted at pos *i* |
| s.pop([i]) | retrieves the item *i* from *s* and also removes it |
| s.remove(x) | retrieves the first item from *s* where s[i] == x |
| s.reverse() | reverses the items of *s* in place |

In [65]:

```
a = [1,2,3,4,5,5,5,5]
#a.append(6)
#last_el = a.pop()
#first_el = a.pop(0)
#print(first_el, last_el, a)
#b = []
#for element in a:
#    if element != 5:
#        b.append(element)
#a
#remove_out = a.remove(5)
[element for element in a if element != 5]
#print(remove_out)
#a.reverse()
#a
```

Out[65]:

```
[1, 2, 3, 4]
```

## Exercise

Calculate the average of the list `[1,2,3.5,5,6.2]` to one decimal, using Python

```
In [70]: l = [1,2,3.5,5,6.2]

         round(10/3, 5)
```

```
Out[70]: 3.33333
```

Take the list `['I','know','Python']` as input and output the string 'I KNOW PYTHON

## Exercise: IMDb (again)

Download the 250.imdb file from the course website

This format of this file is:

- Line by line
- Columns separated by the | character
- Header starting with #

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|    8.5|1957|5280|https://images-na.ssl-images....|Drama,War|Paths of Glory
    71379|    8.2|1925|4320|https://images-na.ssl-images....|Adventure,Comedy,Drama,Family|The Gold
```

# Votes | Rating | Year | Runtime | URL | Genres | Title

## Find the movie with the highest rating

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|    8.5|1957|5280|https://images-na.ssl-images....|Drama,War|Paths of Glory
    71379|    8.2|1925|4320|https://images-na.ssl-images....|Adventure,Comedy,Drama,Family|The Gold
```

In [ ]:

```
fh   = open("../downloads/250.imdb", "r", encoding = "utf-8")

...
```

# IMDb

## Find the number of unique genres

Watch out for the upper/lower cases!

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|    8.5|1957|5280|https://images-na.ssl-images....|Drama,War|Paths of Glory
    71379|    8.2|1925|4320|https://images-na.ssl-images....|Adventure,Comedy,Drama,Family|The Gold
```