Lecture 3-2

Lists Part 2 and Strings

Week 3 Wednesday

with thanks to Miles Chen, PhD

Adapted from Chapter 6 of Think Python by Allen B Downey
List content adapted from "Whirlwind Tour of Python" by Jake VanderPlas

List Methods

- `list.copy()`
  - Return a shallow copy of the list. Equivalent to a[:]
- `list.append(x)`
  - Add an item to the end of the list. Equivalent to a[len(a):] = [x].

In [2]:

```
          favs.append(4.6)   # unlike R, you don't have to "capture" the result
of the function.
# the list itself is modified. You can only append one item.
print(favs)
```

```
[4.5, 'Rose Donut', ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00
AM - 3:00 PM'], 4.0, "Porto's", ['Sat 6:30 AM - 8:00 P
M', 'Sun 6:30 AM - 8:00 PM'], 4.5, 'Bagel Broker', ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM'], 4.6]
```

In [3]:

```
print(favs2)
```

```
[4.5, 'Rose Donut', ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00
AM - 3:00 PM'], 4.0, "Porto's", ['Sat 6:30 AM - 8:00 P
M', 'Sun 6:30 AM - 8:00 PM'], 4.5, 'Bagel Broker', ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM'], 4.6]
```

In [4]:

```
        favs = favs + ["La Azteca Tortilleria"]  # you can also append to a list with the addition `+` operator
# note that this output needs to be 'captured' and assigned back to fam
print(favs)
```

```
[4.5, 'Rose Donut', ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00
AM - 3:00 PM'], 4.0, "Porto's", ['Sat 6:30 AM - 8:00 P
M', 'Sun 6:30 AM - 8:00 PM'], 4.5, 'Bagel Broker', ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM'], 4.6, 'La A
zteca Tortilleria']
```

In [5]:

```
print(favs2) # and look what happens to our copy...
```

```
[4.5, 'Rose Donut', ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00
AM - 3:00 PM'], 4.0, "Porto's", ['Sat 6:30 AM - 8:00 P
M', 'Sun 6:30 AM - 8:00 PM'], 4.5, 'Bagel Broker', ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM'], 4.6]
```

```
In [6]:
        favs.append(['Sat 7:00 AM - 3:30 PM', 'Sun 7:00 AM - 3:30 PM'])
```

```
In [7]:
        favs
```

```
Out[7]:

[4.5,
 'Rose Donut',
 ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00 AM - 3:00 PM'],
 4.0,
 "Porto's",
 ['Sat 6:30 AM - 8:00 PM', 'Sun 6:30 AM - 8:00 PM'],
 4.5,
 'Bagel Broker',
 ['Sat 7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM'],
 4.6,
 'La Azteca Tortilleria',
 ['Sat 7:00 AM - 3:30 PM', 'Sun 7:00 AM - 3:30 PM']]
```

In [8]:

```
favs2 # reassignment broke their relationship
```

Out[8]:

```
[4.5,
 'Rose Donut',
 ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00 AM - 3:00 PM'],
 4.0,
 "Porto's",
 ['Sat 6:30 AM - 8:00 PM', 'Sun 6:30 AM - 8:00 PM'],
 4.5,
 'Bagel Broker',
 ['Sat 7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM'],
 4.6]
```

In [9]:

```
        favs2 +['La Azteca Tortilleria',['Sat 7:00 AM - 3:30 PM', 'Sun 7:00
AM - 3:30 PM']] # plus operator concatenates the lists
```

Out[9]:

```
[4.5,
 'Rose Donut',
 ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00 AM - 3:00 PM'],
 4.0,
 "Porto's",
 ['Sat 6:30 AM - 8:00 PM', 'Sun 6:30 AM - 8:00 PM'],
 4.5,
 'Bagel Broker',
 ['Sat 7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM'],
 4.6,
 'La Azteca Tortilleria',
 ['Sat 7:00 AM - 3:30 PM', 'Sun 7:00 AM - 3:30 PM']]
```

```
In [10]:
favs # sanity check
```

```
Out[10]:
[4.5,
 'Rose Donut',
 ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00 AM - 3:00 PM'],
 4.0,
 "Porto's",
 ['Sat 6:30 AM - 8:00 PM', 'Sun 6:30 AM - 8:00 PM'],
 4.5,
 'Bagel Broker',
 ['Sat 7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM'],
 4.6,
 'La Azteca Tortilleria',
 ['Sat 7:00 AM - 3:30 PM', 'Sun 7:00 AM - 3:30 PM']]
```

In [11]:

```
x # sanity check
```

Out[11]:

```
['Rose Donut',
 5.0,
 ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00 AM - 3:00 PM'],
 "Porto's",
 4.0,
 ['Sat 6:30 AM - 8:00 PM', 'Sun 6:30 AM - 8:00 PM'],
 'Bagel Broker',
 4.5,
 ['Sat 7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM']]
```

Copy vs. Deep Copy Example using the copy module

- `list.copy` and `list[:]` both create shallow copies.
- A shallow copy creates a copy of the list, but does not create new copies of any objects that the list references. It references the same location in memory.
- A deep copy will copy the list and create copies of objects that the list references. The copied objects are in a new location.

In [12]:

```
          a = ["a", 1, 2]
b = ["b", 3, 4]
c = [a, b]

import copy
d = c[:]  # d is a shallow copy of c
e = copy.deepcopy(c)  # e is a deep copy of c
f = c # assigned c to f

c.append("x")  # modify c
print(c) # c reflects the change
print(d) # d is a shallow copy and is not changed
print(e) # e is a deep copy and is not changed
print(f) # f is assigned c
```

```
[['a', 1, 2], ['b', 3, 4], 'x']
[['a', 1, 2], ['b', 3, 4]]
[['a', 1, 2], ['b', 3, 4]]
[['a', 1, 2], ['b', 3, 4], 'x']
```

In [13]:

```
          a.append("z")  # modify list a, an element in c
print(c) # c reflects change
print(d) # d shallow copy of c and reflects the change
print(e) # is a deep copy and is not affected by changes to underlying
elements
print(f) # f is assigned c and reflects changes in c
```

```
[['a', 1, 2, 'z'], ['b', 3, 4], 'x']
[['a', 1, 2, 'z'], ['b', 3, 4]]
[['a', 1, 2], ['b', 3, 4]]
[['a', 1, 2, 'z'], ['b', 3, 4], 'x']
```

insert and extend

- `list.insert(i, x)`
  - Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).
- `list.extend(iterable)`
  - Extend the list by appending all the items from the iterable. Equivalent to a[len(a):] = iterable.

In [14]:

```
          fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam.insert(4, "joe") # inserts joe at the location of the 4th comma between 
1.68 and mom
print(fam)
```

```
['liz', 1.73, 'emma', 1.68, 'joe', 'mom', 1.71, 'dad', 
1.89]
```

In [15]:

```
          fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam.insert(4, ["joe", 2.0])  # trying to insert multiple items by using a 
list inserts a list
print(fam)
```

```
['liz', 1.73, 'emma', 1.68, ['joe', 2.0], 'mom', 1.71, 
'dad', 1.89]
```

In [16]:

```
          fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam.insert(4, "joe", 2.0)  # like append, you can only insert one item
# trying to insert multiple items causes and error
print(fam)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykernel_54073/1622468637.py in <module>
      1 fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
----> 2 fam.insert(4, "joe", 2.0)  # like append, you can only insert one item
      3 # trying to insert multiple items causes and error
      4 print(fam)

TypeError: insert expected 2 arguments, got 3
```

In [17]:

```
            fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam.extend(["joe", 2.0]) # lets you add multiple items, but at the end
print(fam)
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'j
oe', 2.0]
```

In [18]:

```
            fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam[4:4] = ["joe", 2.0] # Use slice and assignment to insert multiple items
in a specific position
print(fam)
```

```
['liz', 1.73, 'emma', 1.68, 'joe', 2.0, 'mom', 1.71, 'da
d', 1.89]
```

Remove/pop/clear

- `list.remove(x)`
  - Remove the first item from the list whose value is x. It is an error if there is no such item.
- `list.pop([i])`
  - Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list.
- `list.clear()`
  - Remove all items from the list. Equivalent to del a[:].

In [19]:

```
          fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam.remove("liz")
print(fam)
```

```
[1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

In [20]:

```
          fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
j = fam.pop()  # if you don't specify an index, it pops the last item in the
list
# default behavior of pop() without any arguments is like a stack. last in
first out
print(j)
print(fam)
```

```
1.89
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad']
```

In [21]:

```
          fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
j = fam.pop(0)  # you can also specify an index.
# Using index 0 makes pop behave like a queue. first in first out
print(j)
print(fam)


fam.clear()
print(fam)
```

```
liz
[1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
[]
```

Index and count

- `list.index(x)`
  - Return zero-based index in the list of the first item whose value is x. Raises a ValueError if there is no such item.
- `list.count(x)`
  - Return the number of times x appears in the list.

```
In [22]:
          fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam.index("emma")
```

```
Out[22]:
2
```

```
In [23]:
          fam.index(3) # nothing valued 3
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (mos
t recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykern
el_54073/1632768634.py in <module>
----> 1 fam.index(3) # nothing valued 3

ValueError: 3 is not in list
```

In [24]:

```
          fam2 = [["liz", 1.73],
["emma", 1.68],
["mom", 1.71],
["dad", 1.89]]
print(fam2.count("emma"))  # the string by itself does not exist
print(fam2.count(["emma", 1.68]))
```

```
0
1
```

sort and reverse

- `list.sort(key=None, reverse=False)`
  - Sort the items of the list in place (the arguments can be used for sort customization, see sorted() for their explanation).
- `list.reverse()`
  - Reverse the elements of the list in place.

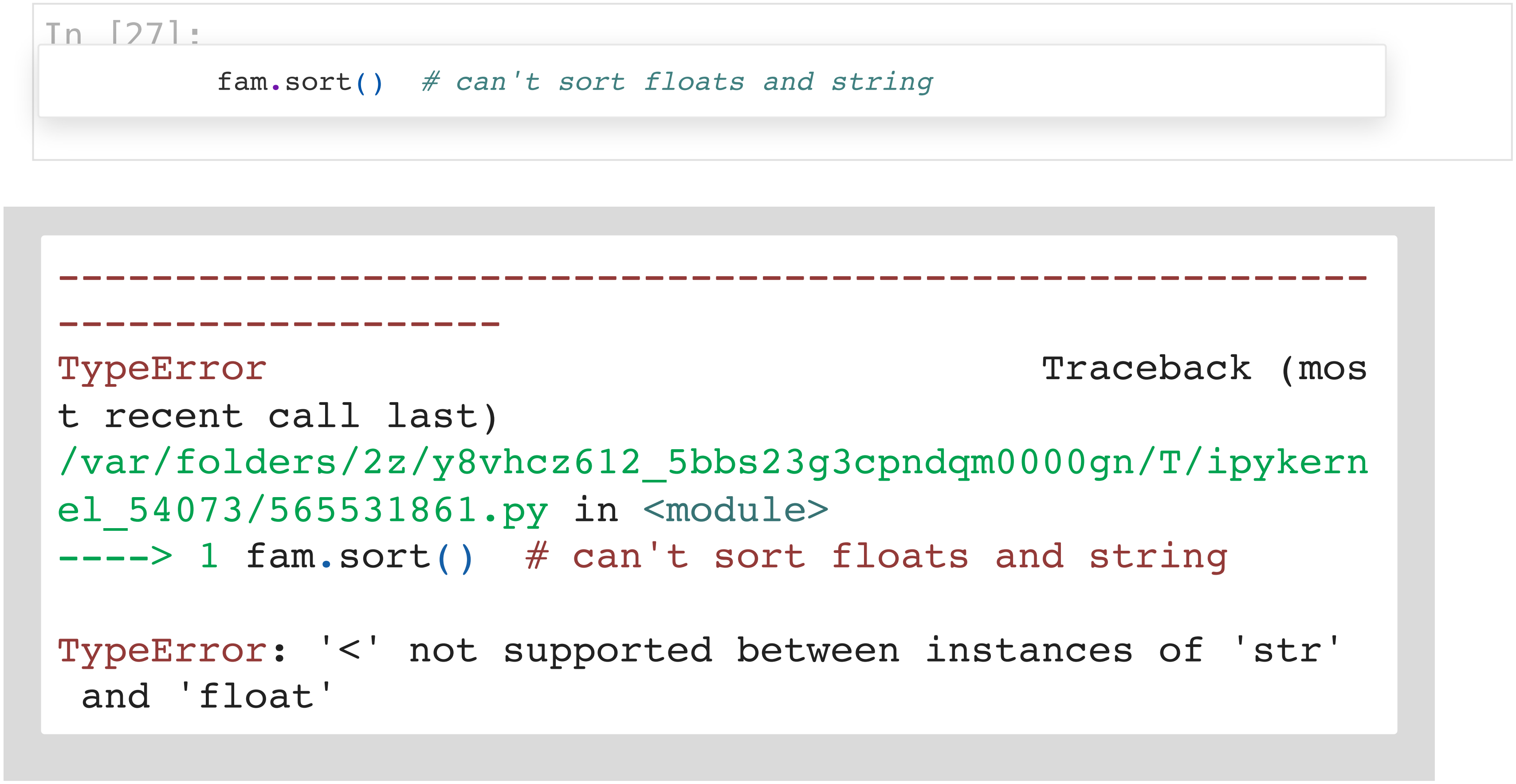
In [25]:

```
        fam.reverse()  # no output to 'capture', the list is changed in
place
```

In [26]:

```
        print(fam)
```

```
[1.89, 'dad', 1.71, 'mom', 1.68, 'emma', 1.73, 'liz']
```



```
In [27]:
            fam.sort()  # can't sort floats and string
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykernel_54073/565531861.py in <module>
----> 1 fam.sort()  # can't sort floats and string

TypeError: '<' not supported between instances of 'str' and 'float'
```

In [28]:

```
some_digits = [4, 2, 7, 9, 2, 5.1, 3]
some_digits.sort()  # the list is sorted in place. no need to resave the
output
```

In [29]:

```
print(some_digits)  # preserves numeric data types
```

```
[2, 2, 3, 4, 5.1, 7, 9]
```

In [30]:

```
type(some_digits[4])
```

Out[30]:

```
float
```

In [31]:

```
            some_digits.sort(reverse = True)
print(some_digits)
```

```
[9, 7, 5.1, 4, 3, 2, 2]
```

In [32]:

```
            some_digits = [4, 2, 7, 9, 2, 5.1, 3] # create a new list
sorted(some_digits)  # sorted will return a sorted copy of the list
```

Out[32]:

```
[2, 2, 3, 4, 5.1, 7, 9]
```

In [33]:

```
            some_digits  # the list is unaffected
```

Out[33]:

```
[4, 2, 7, 9, 2, 5.1, 3]
```

Cleanup

- clear will empty a list but it is still defined

In [34]:
```
            some_digits.clear()
print(some_digits)
```

```
[]
```

In [35]:
```
            some_digits = [6, 5, 9, 2]
print(some_digits)
```

```
[6, 5, 9, 2]
```

Cleanup (continued)

- del will delete a list, no longer exists

```
In [36]:
        del some_digits
```

```
In [37]:
        print(some_digits)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykernel_54073/4265411950.py in <module>
----> 1 print(some_digits)

NameError: name 'some_digits' is not defined
```

Extra - Reading a CSV using the standard library

```
In [38]:
          # import the csv module
import csv

# csv file name
filename = "iris.csv"

# create columns and rows list
columns = []
rows = []

# read csv file
with open(filename, 'r') as csvfile:
    # create a csv reader object
    iris = csv.reader(csvfile)

    # extract column names in the first row
    columns = next(iris)

    # extract each row
```

```python
    for row in iris:
        rows.append(row)
```

The reveal

```
In [39]:
            # reveal number of rows
nrows = len(rows)


print(f"Number of rows: {nrows}\n")


# columns
print(f'Columns are: {columns}\n')
```

```
Number of rows: 150

Columns are: ['sepal length (cm)', 'sepal width (cm)',
'petal length (cm)', 'petal width (cm)', 'target']
```

```
In [40]:
        # print some rows
print('First few rows are:\n')
rows[0:5]
```

```
First few rows are:

```

```
Out[40]:
[['5.1', '3.5', '1.4', '0.2', '0.0'],
 ['4.9', '3.0', '1.4', '0.2', '0.0'],
 ['4.7', '3.2', '1.3', '0.2', '0.0'],
 ['4.6', '3.1', '1.5', '0.2', '0.0'],
 ['5.0', '3.6', '1.4', '0.2', '0.0']]
```

In [41]:

```
rows[0:5]
```

Out[41]:

```
[['5.1', '3.5', '1.4', '0.2', '0.0'],
 ['4.9', '3.0', '1.4', '0.2', '0.0'],
 ['4.7', '3.2', '1.3', '0.2', '0.0'],
 ['4.6', '3.1', '1.5', '0.2', '0.0'],
 ['5.0', '3.6', '1.4', '0.2', '0.0']]
```

In [42]:

```
rows[0][0]
```

Out[42]:

```
'5.1'
```

In [43]:

```
sl = [float(sl[0]) for sl in rows]
```

In [44]:

```
import numpy as np
np.mean(sl)
```

Out[44]:

```
5.843333333333334
```

In [45]:

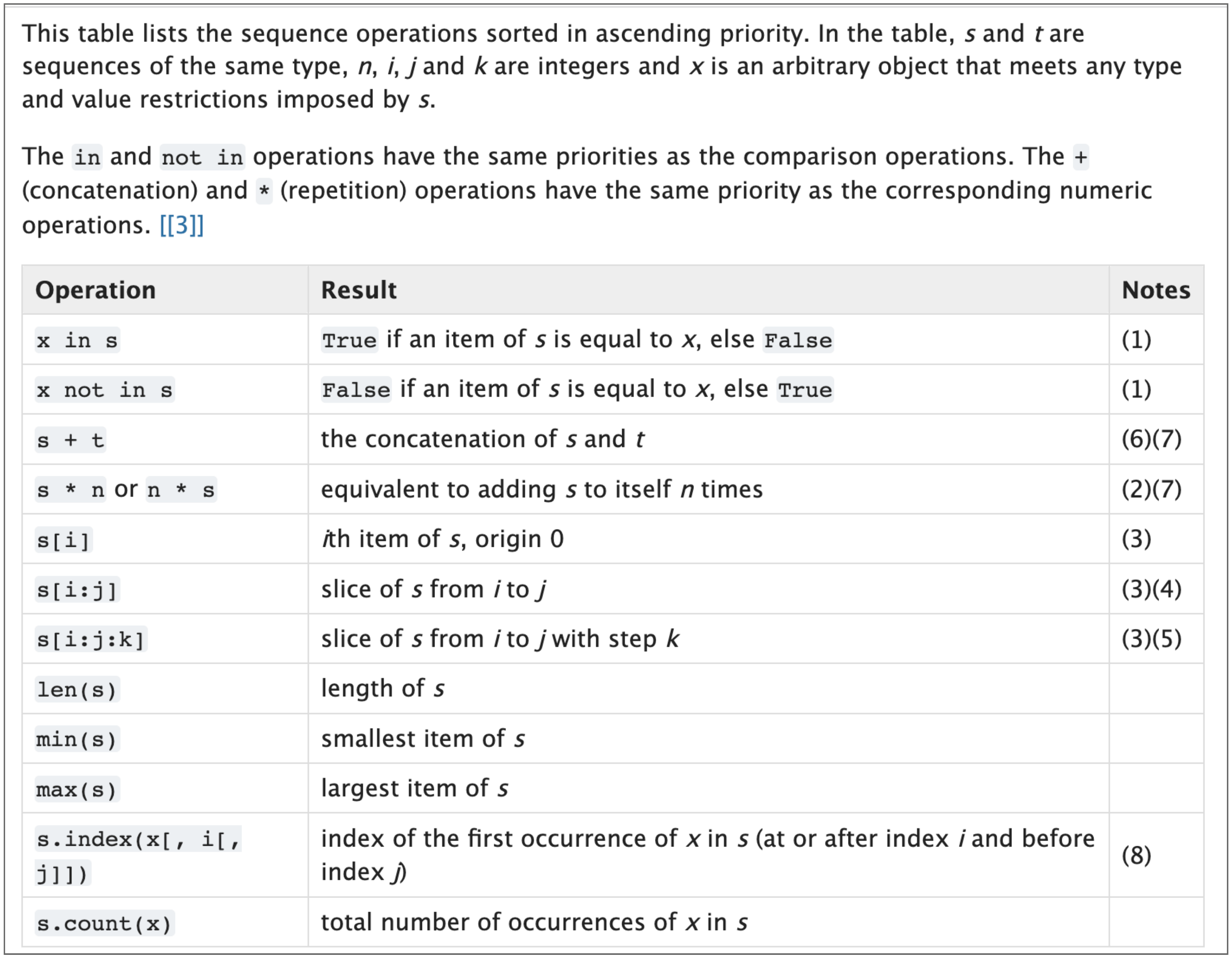
```
sum(sl)/len(sl)
```

Out[45]:

```
5.843333333333335
```

Common Sequence Operations

It might help to have this available
https://docs.python.org/3/library/stdtypes.html#typesseq-common



This table lists the sequence operations sorted in ascending priority. In the table, *s* and *t* are sequences of the same type, *n*, *i*, *j* and *k* are integers and *x* is an arbitrary object that meets any type and value restrictions imposed by *s*.

The `in` and `not in` operations have the same priorities as the comparison operations. The `+` (concatenation) and `*` (repetition) operations have the same priority as the corresponding numeric operations. [[3]]

Operation	**Result**	**Notes**
`x in s`	`True` if an item of *s* is equal to *x*, else `False`	(1)
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`	(1)
`s + t`	the concatenation of *s* and *t*	(6)(7)
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times	(2)(7)
`s[i]`	*i*th item of *s*, origin 0	(3)
`s[i:j]`	slice of *s* from *i* to *j*	(3)(4)
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*	(3)(5)
`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*)	(8)
`s.count(x)`	total number of occurrences of *x* in *s*	

Strings

From the documentation:

- Textual data in Python is handled with str objects, or strings. Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways:
    - Single quotes: 'allows embedded "double" quotes'
    - Double quotes: "allows embedded 'single' quotes"
    - Triple quoted: '''Three single quotes''', """Three double quotes"""

Triple quoted strings may span multiple lines - all associated whitespace will be included in the string literal.

Examples

```
In [46]:
fruit = "bananas"
```

```
In [47]:
fruit[0]  # Python is 0-indexed
```

```
Out[47]:
'b'
```

```
In [48]:
fruit[1]
```

```
Out[48]:
'a'
```

```
In [49]:
         fruit[-1] # last letter
```

```
Out[49]:
's'
```

```
In [50]:
         fruit[1.5]
```

```
---------------------------------------------------------
--------------------
TypeError                                  Traceback (mos
t recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykern
el_54073/1901970537.py in <module>
----> 1 fruit[1.5]

TypeError: string indices must be integers
```

`len()` tells you the length of a string

In [51]:

```
len(fruit) # that is "bananas"
```

Out[51]:

```
7
```

Subsetting Strings and strings as iterables

You can subset and slice a string much like you would a list or tuple:

```
In [52]:
s = 'abcdefghijklmnopqrstuvwxyz'
```

```
In [53]:
s[4:9]
```

```
Out[53]:
'efghi'
```

Subsetting Strings and strings as iterables

In [54]:

```
s[-6:]
```

Out[54]:

```
'uvwxyz'
```

In [55]:

```
for x in s[0:5]:
    print(x + '!')
```

```
a!
b!
c!
d!
e!
```

Strings are immutable

This means that when you use a method on a string, it does not modify the string itself and returns a new string object.

```
In [56]:
          # strings are immutable. You cannot modify a string that has been
created.
s[0] = 'b'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykernel_54073/543338217.py in <module>
      1 # strings are immutable. You cannot modify a string that has been created.
----> 2 s[0] = 'b'

TypeError: 'str' object does not support item assignment
```

Strings are immutable (cont'd)

```
In [57]:
            'b' + s[1:] # if i wanted the string where the first letter is now
b
```

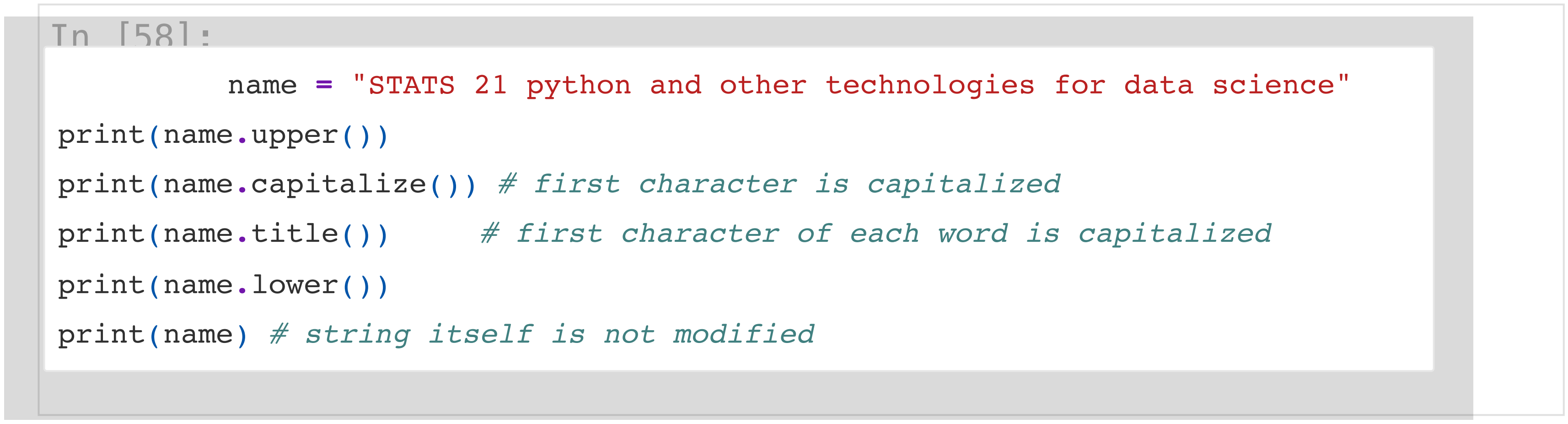
```
Out[57]:
'bbcdefghijklmnopqrstuvwxyz'
```

String Methods

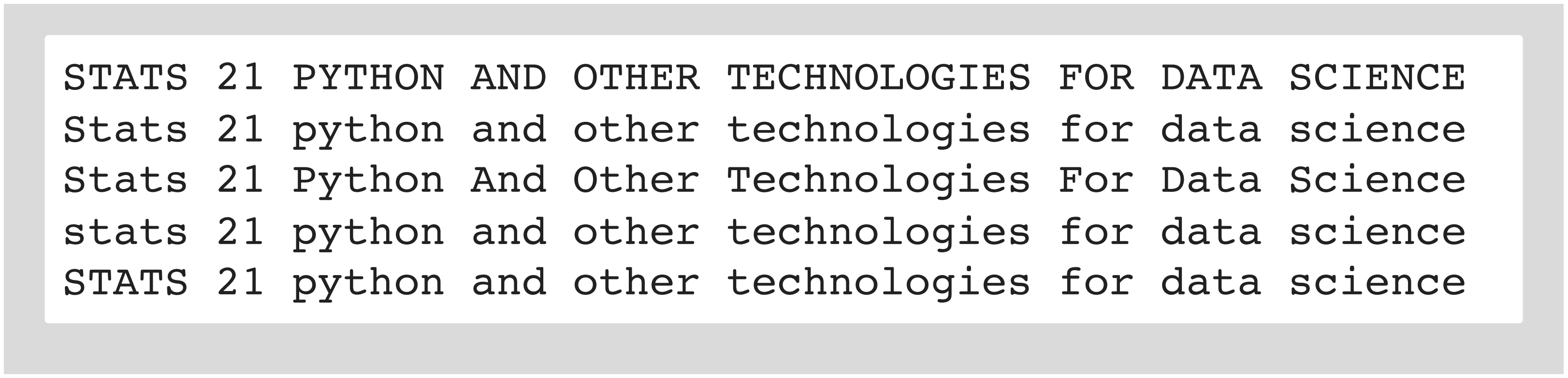
- Strings can make use of all of the common sequence operations (listed above)
- Strings also have many builtin methods

https://docs.python.org/3/library/stdtypes.html#string-methods-1

String Methods - examples



```
In [58]:
         name = "STATS 21 python and other technologies for data science"
print(name.upper())
print(name.capitalize()) # first character is capitalized
print(name.title())     # first character of each word is capitalized
print(name.lower())
print(name) # string itself is not modified
```



```
STATS 21 PYTHON AND OTHER TECHNOLOGIES FOR DATA SCIENCE
Stats 21 python and other technologies for data science
Stats 21 Python And Other Technologies For Data Science
stats 21 python and other technologies for data science
STATS 21 python and other technologies for data science
```

String Methods

They might be handy for a homework

```
In [59]:
        print(name.startswith("s"))
print(name.endswith("e"))
print(name.isalpha())
print(name.split(sep=" "))
name_split = name.split(sep=" ")
print(name_split[0])
```

```
False
True
False
['STATS', '21', 'python', 'and', 'other', 'technologie
s', 'for', 'data', 'science']
STATS
```

Count how many times a letter appears

In [60]:

```
            count = 0
for letter in name:
    if letter == "e":
        count = count + 1
print(count)
```

```
5
```

In [61]:

```
            # can be achieved with a simple method:
name.count("e")
```

Out[61]:

```
5
```

In [62]:

```
name.index('A') # index of the first instance
```

Out[62]:

```
2
```

In [63]:

```
name.endswith("k")
```

Out[63]:

```
False
```

In [64]:

```
name.endswith("e")
```

Out[64]:

```
True
```

In [65]:

```
        name.startswith("s")  # case sensitive
```

Out[65]:

```
False
```

In [66]:

```
        # create multi-line strings with triple quotes
name2 = '''    los angeles



'''
print(name2)
```

```
    los angeles



```

In [67]:

```python
name2.strip()  # removes extra whitespace
```

Out[67]:

```
'los angeles'
```

In [68]:

```python
name2 # remember strings are immutable, the original string still has the white space
```

Out[68]:

```
'   los angeles \n\n\n'
```

string.split()

```
In [69]:
            name2.split() # the result of split() is a list
```

```
Out[69]:

['los', 'angeles']
```

```
In [70]:
            num_string = "2,3,4,7,8"
print(num_string.split()) # defaults to splitting on space
print(num_string.split(','))
```

```
['2,3,4,7,8']
['2', '3', '4', '7', '8']
```

In [71]:

```
            # list comprehension (covered later) to convert the split strings
into int
[int(x) for x in num_string.split(',')]
```

Out[71]:

```
[2, 3, 4, 7, 8]
```

In [72]:

```
            # the list comprehension is a more concise version of the following
code
l = []
for x in num_string.split(','):
    l.append(int(x))
l
```

Out[72]:

```
[2, 3, 4, 7, 8]
```

In [73]:

```
          print(name)
print(name.isalpha()) # has spaces and digits, so it is not strictly alpha
name3 = "abbaAZ"
name3.isalpha()
```

```
STATS 21 python and other technologies for data science
False
```

Out[73]:

```
True
```

In [74]:

```
          name4 = "abbaAZ4"
name4.isalpha()
```

Out[74]:

```
False
```

In [75]:

```python
          # strings can span multiple lines with triple quotes
long_string = """Lyrics to the song Hurt
I hurt myself today
To see if I still feel
I focus on the pain
The only thing that's real
The needle tears a hole
The old familiar sting
Try to kill it all away
But I remember everything"""
shout = long_string.upper()
print(shout)
word_list = long_string.split() # separates at spaces
print(word_list)
```

```
LYRICS TO THE SONG HURT
I HURT MYSELF TODAY
TO SEE IF I STILL FEEL
I FOCUS ON THE PAIN
THE ONLY THING THAT'S REAL
THE NEEDLE TEARS A HOLE
THE OLD FAMILIAR STING
```

```
TRY TO KILL IT ALL AWAY
BUT I REMEMBER EVERYTHING
['Lyrics', 'to', 'the', 'song', 'Hurt', 'I', 'hurt', 'my
self', 'today', 'To', 'see', 'if', 'I', 'still', 'feel',
'I', 'focus', 'on', 'the', 'pain', 'The', 'only', 'thin
g', "that's", 'real', 'The', 'needle', 'tears', 'a', 'ho
le', 'The', 'old', 'familiar', 'sting', 'Try', 'to', 'ki
ll', 'it', 'all', 'away', 'But', 'I', 'remember', 'every
thing']
```

In [76]:

```
        long_string.splitlines() # separates at line ends
# you'll notice that python defaults to using single quotes, but if the
string contains an apostrophe,
# it will use double quotes
```

Out[76]:

```
['Lyrics to the song Hurt',
 'I hurt myself today',
 'To see if I still feel',
 'I focus on the pain',
 "The only thing that's real",
 'The needle tears a hole',
 'The old familiar sting',
 'Try to kill it all away',
 'But I remember everything']
```

In [77]:

```
        long_string.count("e")
```

Out[77]:

21

Searching for a letter

```
long_string = """Lyrics to the song Hurt
I hurt myself today
To see if I still feel
I focus on the pain
The only thing that's real
The needle tears a hole
The old familiar sting
Try to kill it all away
But I remember everything"""
```

In [78]:

```
          def myfind(string, letter):
    index = 0
    while index < len(string):
        if string[index] == letter:
            return index
        index = index + 1
    return -1
```

```
        myfind(long_string, "t")
```

Out[79]:

```
7
```

```
In [80]:
            # Python already has a find method built in
long_string.find("t") # index of the first instance of 't'
```

```
Out[80]:
7
```

```
In [81]:
            long_string.index('t') # string.index() and string.find() are
similar.
```

```
Out[81]:
7
```

```
In [82]:
        long_string.find('$') # string.find() returns a -1 if the character
doesn't exist in the string
```

```
Out[82]:
-1
```

```
In [83]:
        long_string.index('$')  # string.index() returns error if the
character doesn't exist in the string.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (mos
t recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykern
el_54073/1812285253.py in <module>
----> 1 long_string.index('$')  # string.index() returns
error if the character doesn't exist in the string.
```

```
ValueError: substring not found
```

`in` operator

returns a boolean value if the first string is a substring of the second string.

```
In [84]:
'a' in 'bananas'
```

```
Out[84]:
True
```

```
In [85]:
'nan' in 'bananas'
```

```
Out[85]:
True
```

In [86]:

```
'bad' in 'bananas'
```

Out[86]:

```
False
```

String comparisons

Use of `>` or `<` compares strings in alphabetical order.

In [87]:

```
'A' < 'B'
```

Out[87]:

```
True
```

In [88]:

```
'a' < 'b'
```

Out[88]:

```
True
```

In [89]:

```
'Z' < 'a'
```

Out[89]:

```
True
```

In [90]:

```
        # digits are less than capital letters
'1' < 'A'
```

Out[90]:

```
True
```

In [91]:

```
        '0' < '00'
```

Out[91]:

```
True
```

In [92]:

```
        # must treat digits like "letters" with alphabetical rules
'11' < '101'
```

Out[92]:

```
False
```

```
In [93]:
        '!' < '@' # the sorting of symbols feels very arbitrary
```

```
Out[93]:
True
```

```
In [94]:
        # sorted order
string = '!@#$%^&*()[]{}\|;:,.<>/?1234567890ABCXYZabcxyz'
x = sorted(string)
print(x)
```

```
['!', '#', '$', '%', '&', '(', ')', '*', ',', '.', '/',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':',
';', '<', '>', '?', '@', 'A', 'B', 'C', 'X', 'Y', 'Z',
'[', '\\', ']', '^', 'a', 'b', 'c', 'x', 'y', 'z', '{',
'|', '}']
```