# Lecture 3-1

# Lists

## Week 3 Monday

## Made possible by 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

# Lists

- A list stores items in a single variable.
- A list is a sequence of values.
- The values can be changed (modify/add/remove)
- The values can be reordered (e.g., sorted)
- In a list, values can be any type. It's data tyoe is list.
- The values can be duplicates too.
- The values in a list are called elements or sometimes items.

# List Creation

- Start simple
- Enclose a sequence with `[ ]` brackets

In [1]:
```
[8, 6, 7, 5]
```

Out[1]:

```
[8, 6, 7, 5]
```

In [2]:
```
["Baguette", "Brioche", "Boule"]
```

Out[2]:

```
['Baguette', 'Brioche', 'Boule']
```

## List Creation (cont'd)

- We can also use a list constructor (function) to initialize (assign values)
- Some people like to say "instantiate" a list object. Create works here too.

In [3]:

```
my_list = list(range(1, 11))
print(my_list)
my_list_too = list(("Baguette", 3.95, 4))
print(my_list_too)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
['Baguette', 3.95, 4]
```

In [4]:

```
type(my_list)
```

Out[4]:

```
list
```

## More properties

- We can also nest lists inside other lists to any depth.
- We can declare empty lists
- We can assign lists to variables (later)

In [5]:

```
favs = []
favs = ["Rose Donut", 4.5, ["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"]]
```

In [6]:

```
favs
```

Out[6]:

```
['Rose Donut',
 4.5,
 ['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']]
```

# Indexing lists

- referencing elements within a list
- index starts at 0 (hardest part to adapt for R users)
- use a series of square brackets to access nested lists
- use negative numbers to count from the end

In [7]:

```
        print(favs[0])
print(favs[3])
print(favs[6])
```

```
Rose Donut
Porto's
Bagel Broker
```

In [8]:

```
favs[2]
```

Out[8]:

```
['Sat 5:00 AM - 3:00 PM', 'Sun 5:00 AM - 3:00 PM']
```

In [9]:

```
favs[2][0]
```

Out[9]:

```
'Sat 5:00 AM - 3:00 PM'
```

In [10]:

```
favs[-1]
```

Out[10]:

```
['Sat 7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM']
```

## List Slicing

- obtaining a subset based on their indices
- Note that the slice will not include the item in the index **after** the colon.
- You can think of the 'slice' happening at the commas corresponding to the number.

In [11]:

```
favs[0:3]
```

Out[11]:

```
['Rose Donut', 4.5, ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00
AM - 3:00 PM']]
```

In [12]:

```
favs[1:4]
```

Out[12]:

```
[4.5, ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00 AM - 3:00 P
M'], "Porto's"]
```

In [13]:

```
favs[-4:-1]
```

Out[13]:

```
[['Sat 6:30 AM - 8:00 PM', 'Sun 6:30 AM - 8:00 PM'], 'Bagel Broker', 4.5]
```

In [14]:

```
favs[1:1]  # there is nothing between the first and first commas
```

Out[14]:

```
[]
```

In [15]:

```
favs[0:2]
```

Out[15]:

```
['Rose Donut', 4.5]
```

In [16]:

```
favs[3:]
```

Out[16]:

```
["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']]
```

In [17]:

```
favs[:3]
```

Out[17]:

```
['Rose Donut', 4.5, ['Sat 5:00 AM - 3:00 PM', 'Sun 5:00
AM - 3:00 PM']]
```

In [18]:

```
print(favs[-3:-1])
```

```
['Bagel Broker', 4.5]
```

In [19]:

```
favs[] # throws error
```

```
  File "/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/
T/ipykernel_77619/2163124824.py", line 1
    favs[] # throws error
         ^
SyntaxError: invalid syntax
```

# Shallow copy and assignment

In [20]:

```
favs[:]  # slice with no indices will create a (shallow) copy of
the list.
```

Out[20]:

```
['Rose Donut',
 4.5,
 ['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']]
```

In [21]:

```
favs2 = favs # assign favs to favs2
x = favs[:]  # assign a shallow copy of favs to x
```

In [22]:

```
favs[1] , favs[0] = favs[0], favs[1] # modify favs
favs
```

Out[22]:

```
[4.5,
 'Rose Donut',
 ['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']]
```

In [23]:

```
favs2  # compare
```

Out[23]:

```
[4.5,
 'Rose Donut',
 ['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']]
```

In [24]:

```
x # the shallow copy
```

Out[24]:

```
['Rose Donut',
 4.5,
 ['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']]
```

From their documentation

**https://docs.python.org/3/library/copy.html**

Assignment statements in Python do not copy objects, they create bindings *(associations)* between a target and an object. For collections that are mutable or contain mutable items, a copy is sometimes needed so one can change one copy without changing the other. This module provides generic shallow and deep copy operations (explained below). (example later)

The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):

- A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.
- A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

## Lists are mutable

- This means that methods change the lists themselves.
- If the list is assigned to another name, both names refer to the exact same object.
- recall we have favs, favs2 and x (a shallow copy)

In [25]:

```
print(favs2)
```

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

In [26]:

```
favs2[3], favs2[4] = favs2[4], favs2[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'], 'Bagel Broker', 4.5, ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM']]
```

In [27]:

```
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'], 'Bagel Broker', 4.5, ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM']]
```

In [28]:

```
print(x)
```

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

In [29]:

```
x[1] = 5.0 # modify the shallow copy
print(x)
```

```
['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 P
M', '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']]
```

In [30]:

```
print(favs) # no effect on the orignal
```

```
[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'], 'Bagel Broker', 4.5, ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM']]
```

In [31]:

```
print(favs2) # no effect here
```

```
[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'], 'Bagel Broker', 4.5, ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM']]
```

You can use list slicing with assignment to change values

In [32]:

```
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'], 'Bagel Broker', 4.5, ['Sat
7:00 AM - 2:00 PM', 'Sun 7:00 AM - 2:00 PM']]
```

In [33]:

```
favs2[6:8] = [4.5, "Bagel Broker"]
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']]
```

In [34]:

```
print(favs) # modified
```

```
[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']]
```

# 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 [35]:

```
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 [36]:

```
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 [37]:

```
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 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']
```

In [38]:

```
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 [39]:

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

In [40]:

```
favs
```

Out[40]:

```
[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 [41]:

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

Out[41]:

```
[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 [42]:

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

Out[42]:

```
[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 [43]:

```
favs # sanity check
```

Out[43]:

```
[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 [44]:

```
x # sanity check
```

Out[44]:

```
['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 [45]:

```
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 [46]:

```
        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 [47]:

```
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 [48]:

```
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 [49]:

```
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_77619/1622468637.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 [50]:

```
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, 'joe', 2.0]
```

In [51]:

```
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, 'dad', 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 [52]:

```
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 [53]:

```
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 [54]:

```
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 [55]:

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

Out[55]:

```
2
```

In [56]:

```
fam.index(3) # nothing valued 3
```

```
---------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykernel_77619/1632768634.py in <module>
----> 1 fam.index(3) # nothing valued 3

ValueError: 3 is not in list
```

In [57]:

```
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 [58]:

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

In [59]:

```
print(fam)
```

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

In [60]:

```
fam.sort()  # can't sort floats and string
```

```
---------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykernel_77619/565531861.py in <module>
----> 1 fam.sort()  # can't sort floats and string

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

In [61]:

```
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 [62]:

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

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

In [63]:

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

Out[63]:

```
float
```

In [64]:

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

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

In [65]:

```
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[65]:

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

In [66]:

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

Out[66]:

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

# Cleanup

- clear will empty a list but it is still defined

In [67]:
```
some_digits.clear()
print(some_digits)
```

```
[]
```

In [68]:
```
some_digits = [6, 5, 9, 2]
print(some_digits)
```

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

# Cleanup (continued)

- del will delete a list, no longer exists

In [69]:
```
del some_digits
```

In [70]:
```
print(some_digits)
```

```
---------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/var/folders/2z/y8vhcz612_5bbs23g3cpndqm0000gn/T/ipykernel_77619/4265411950.py in <module>
----> 1 print(some_digits)

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

# Extra - Reading a CSV using the standard library

In [71]:

```
        # 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
```

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

# The reveal

In [72]:

```
        # 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 [73]:

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

```
First few rows are:
```

Out[73]:

```
[['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 [74]:

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

Out[74]:

```
[['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 [75]:

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

Out[75]:

```
'5.1'
```

In [76]:

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

In [77]:

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

Out[77]:

```
5.843333333333334
```

In [78]:

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

Out[78]:

```
5.843333333333335
```