Hour 1

Nested Functions:

- Function inside a function is called nested function
- It cannot be called directly by other functions as it is out of scope
- It can be called in the parent function of the nested function
- Below, test function call will only print Hello World!

```
def test():

    print("Hello World")

    def inner_test():
        print("Let's go to dinner!")

def main():
    test()

if __name__ == "__main__":
    main()
```

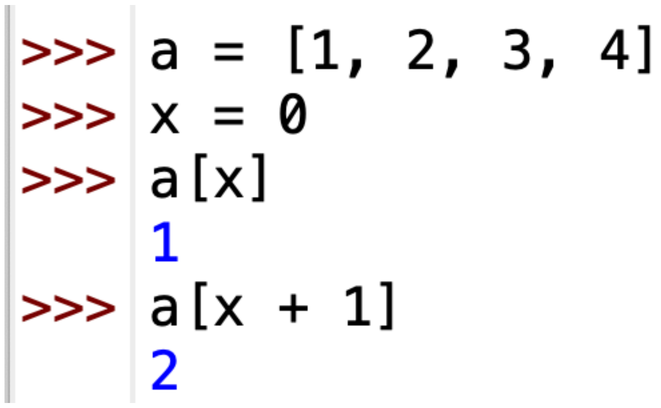
Lists

- Set of things aggregated together
- Named collection of data separated by commas.

```
list_1 = [10, 20, 30, 40]
list_2 = ["Eggs", "Milk", "Flour", "Tomatoes", "Bacon"]
```

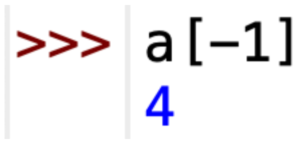
Ex: Shopping list

- Each list item is indexed meaning numbered. First one is always 0 and next is 1 and so on.
- To get an element, you can call it with list name and its index.
- Basic calculation can be done with the index calls.



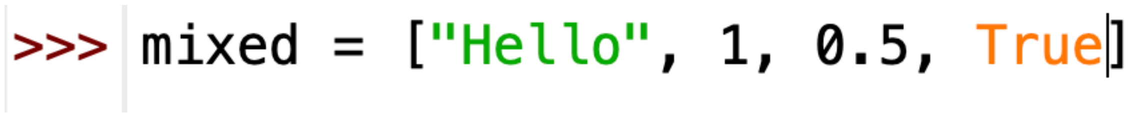
```
>>> a = [1, 2, 3, 4]
>>> x = 0
>>> a[x]
    1
>>> a[x + 1]
    2
```

Negative numbers can be used to start from the back



```
>>> a[-1]
    4
```

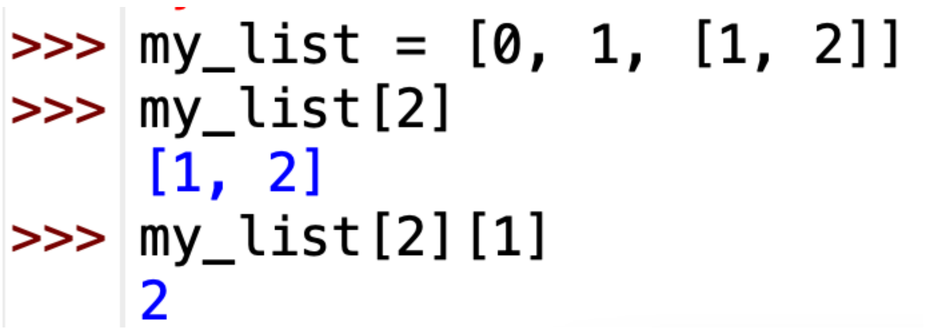
If numbers more than the last max index are used, out of bounds error is returned.



```
>>> mixed = ["Hello", 1, 0.5, True]
```

Hour 2

Lists can also have different data types inside mixed.

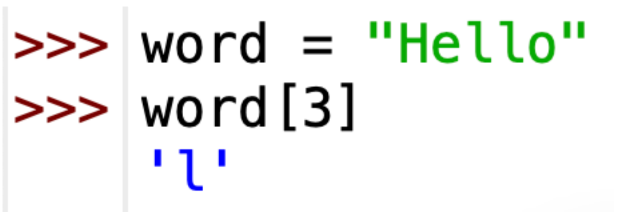


```
>>> my_list = [0, 1, [1, 2]]
>>> my_list[2]
    [1, 2]
>>> my_list[2][1]
    2
```

Lists can also store lists inside them. These are called 2 dimensional lists and they can be called with double bracket groups.

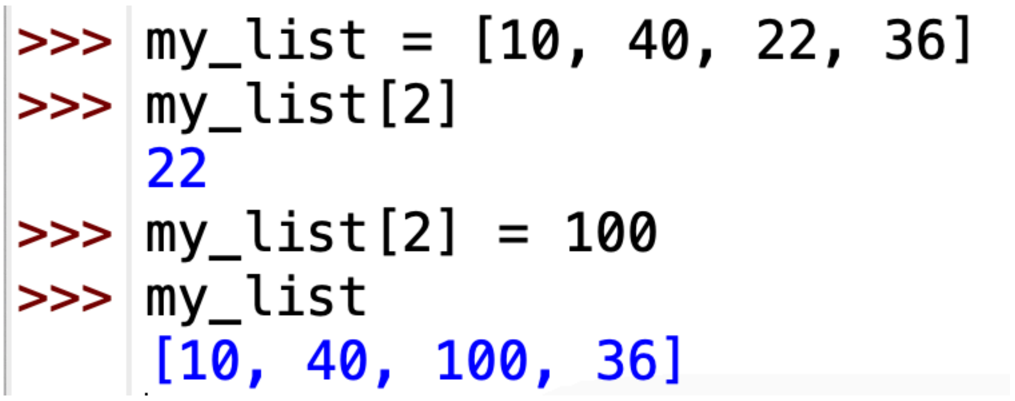
Strings are also considered lists of characters

They are immutable, meaning you cannot change them



```
>>> word = "Hello"
>>> word[3]
    'l'
```

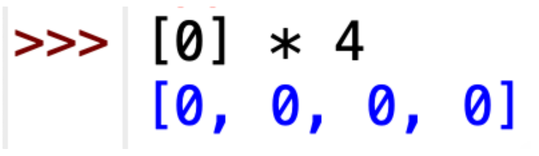
Lists are mutable, meaning you can assign / change its elements:



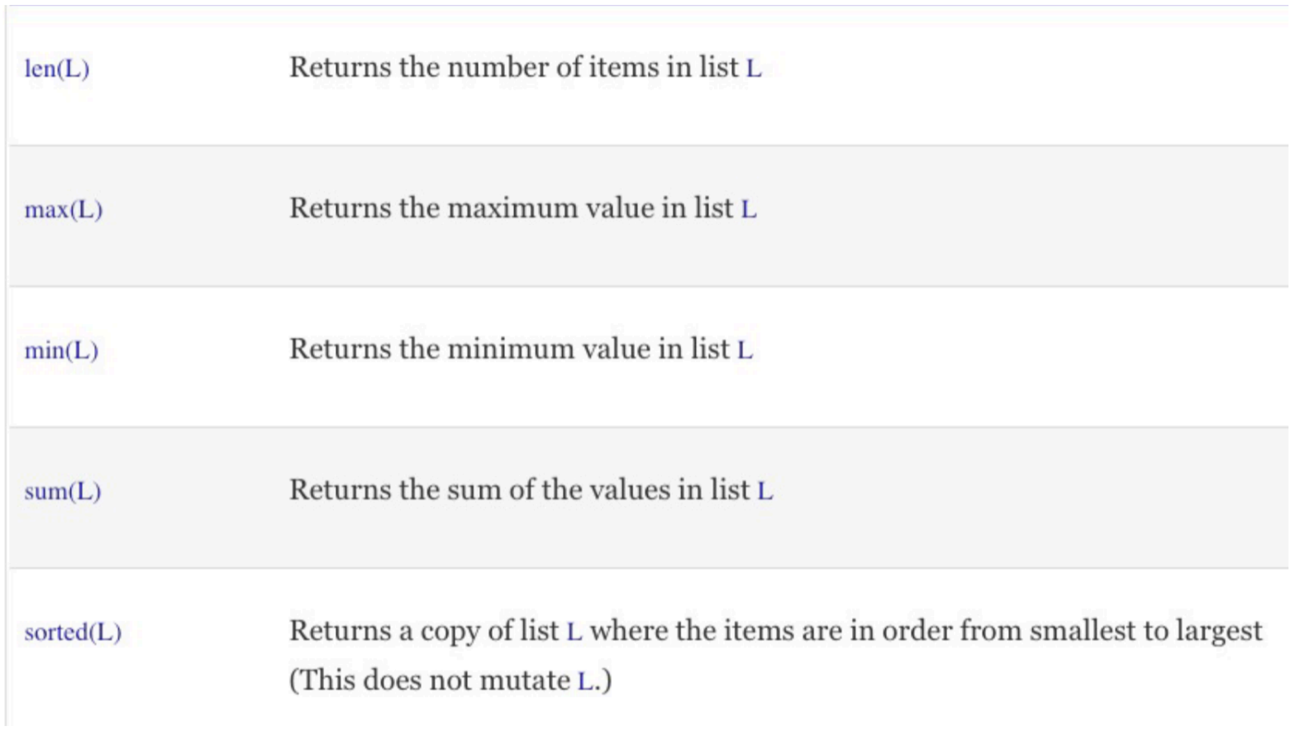
```
>>> my_list = [10, 40, 22, 36]
>>> my_list[2]
    22
>>> my_list[2] = 100
>>> my_list
    [10, 40, 100, 36]
```

List functions:

We can also use + / * operators on lists. + operator extends lists together. * operator repeats the given element/ list of elements in the list.



```
>>> [0] * 4
    [0, 0, 0, 0]
```

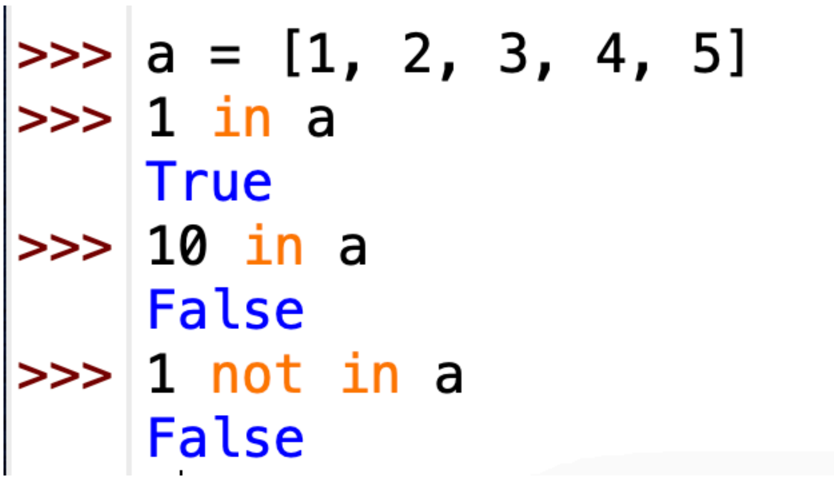


len(L)	Returns the number of items in list L
max(L)	Returns the maximum value in list L
min(L)	Returns the minimum value in list L
sum(L)	Returns the sum of the values in list L
sorted(L)	Returns a copy of list L where the items are in order from smallest to largest (This does not mutate L.)

Grade entry program with lists and list functions

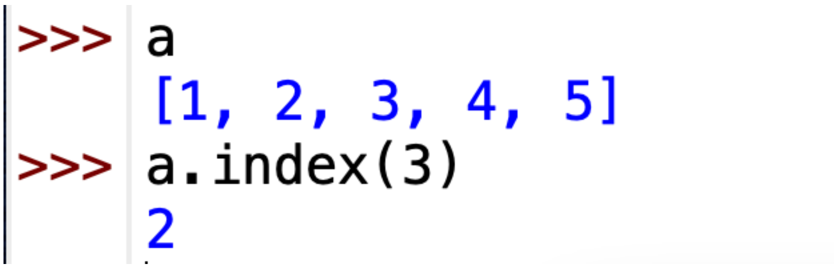
```python
"""
    Grade entry workshop together
"""

def main():
    grades = []

    my_grade = float(input("Enter a grade, negative to stop: "))
    while my_grade >= 0:
        grades.append(my_grade)
        my_grade = float(input("Enter another grade, negative to stop: "))

    average = sum(grades) / len(grades)

    print(f"Average is: {average}")


if __name__ == "__main__":
    main()
```

The **in** keyword can be used to check if a value exists inside a list



```
>>> a = [1, 2, 3, 4, 5]
>>> 1 in a
True
>>> 10 in a
False
>>> 1 not in a
False
```

To find the index of a particular element, index() function can be used

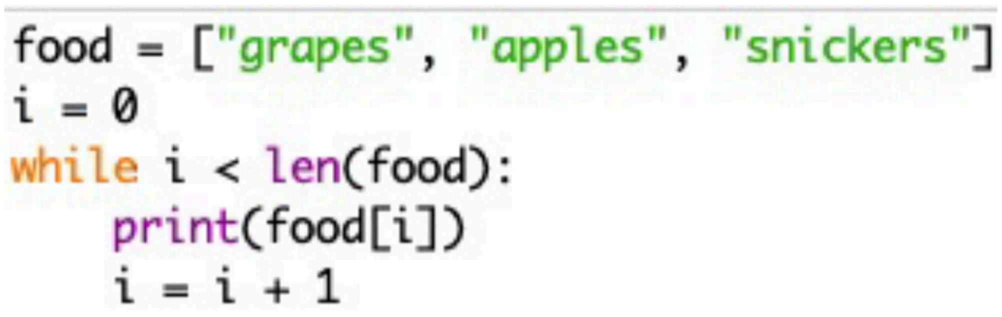


```
>>> a
[1, 2, 3, 4, 5]
>>> a.index(3)
2
```

Traversing Lists:

Using index:

Use a while loop to loop till your traversing variable reaches the end of the list.



```
food = ["grapes", "apples", "snickers"]
i = 0
while i < len(food):
    print(food[i])
    i = i + 1
```

Hour 3 and 3.5

Breakout Exercise: Shopping Lists

```python
def main():
    MY_LIST = ["apples", "grapes", "guava", "melons"]
    user_list = []

    food = input("Enter a food item that you need to buy: ")

    while food.lower() != "stop":
        user_list.append(food)
        food = input("Enter another food item that you need to buy: ")

    i = 0
    while i < len(user_list):
        if user_list[i] in MY_LIST:
            print(f"Hey we both are buying {user_list[i]}!")
        i += 1


if __name__ == "__main__":
    main()
```

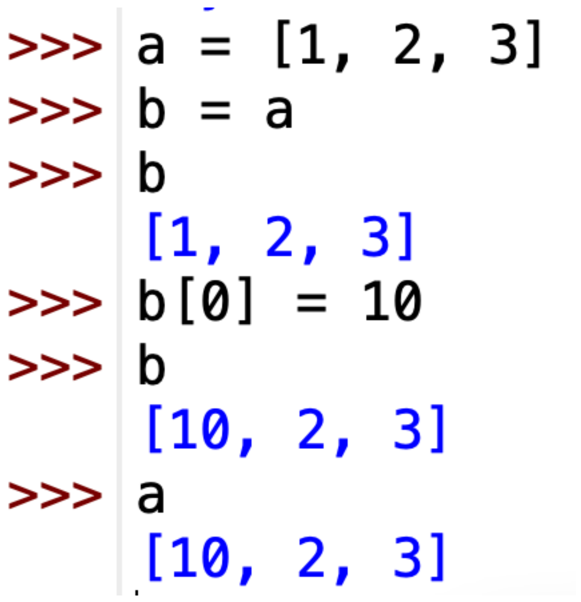
Here we can use the For loop to make our code more concise

```python
for each in user_list:
    if each.lower() == MY_LIST:
        print(f"Hey we both are buying {user_list[i]}!")
```

Range function

- range (starting value, stopping value, steps)
- Starting value is inclusive and stopping value is exclusive.
- Can be used in a for loop to specify how long the loop should run

Pass by Reference:

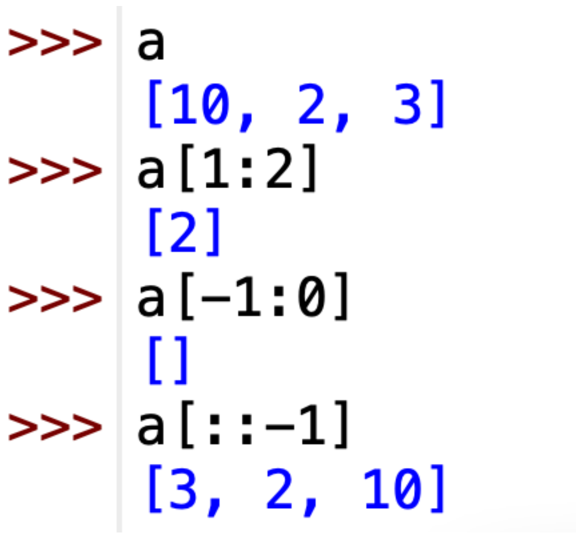


```
>>> a = [1, 2, 3]
>>> b = a
>>> b
[1, 2, 3]
>>> b[0] = 10
>>> b
[10, 2, 3]
>>> a
[10, 2, 3]
```

Here the b is pointing to the exact same space in memory where a is and so both are changed. To actually make a copy, use the copy() function.

Slices:

Slice is a way to get a part of a list or string. [start : stop : step]

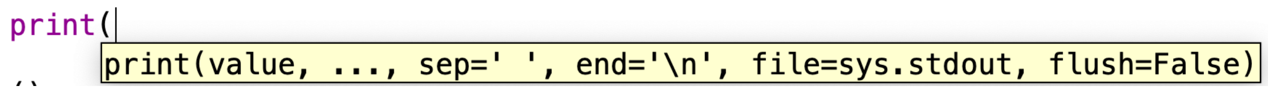


```
>>> a
[10, 2, 3]
>>> a[1:2]
[2]
>>> a[-1:0]
[]
>>> a[::-1]
[3, 2, 10]
```

Default values for a function

def function(x=0, y=1)

Here x and y have a default value specified. This means if they are not specified when calling the function, they resort to using the default values.



```
print(
      print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Here, print has only 1 value that needs an input and rest all parameters have default values.