Lists and Tuples

- List items can be accessed by *index* and *range*.

```
#initialize a list
hogwarts_houses = ['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff']
hogwarts_houses
['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff']

#Access by index
hogwarts_houses[1]
'Slytherin'

#Access by range
hogwarts_houses[0:2]
['Gryffindor', 'Slytherin']
hogwarts_houses[::-1]
['Hufflepuff', 'Ravenclaw', 'Slytherin', 'Gryffindor']
```

list functions

- append() -> add an item to end of list

```
hogwarts_houses.append('NewHouse')
hogwarts_houses
['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff', 'NewHouse']
```

- insert() -> add element at a particular index. Does not over-write. Elements are right-shifted.

```
hogwarts_houses.insert(0,'AnotherImaginaryHouse')
hogwarts_houses
['AnotherImaginaryHouse', 'Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff', 'NewHouse']
```

- extend() -> used to add *multiple* items to end of list. *Unlike append which adds just 1 item to end of list.*

```
house_list = ['NewHouse1','NewHouse2']
hogwarts_houses.extend(house_list)
hogwarts_houses
['AnotherImaginaryHouse', 'Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff', 'NewHouse', 'NewH

#Append will add the list as a list item and not as an element of the list
hogwarts_houses
['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff']
hogwarts_houses.append(house_list)
hogwarts_houses
['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff', ['NewHouse1', 'NewHouse2']]
```

- remove() - >removes an element by value

```
hogwarts_houses = ['AnotherImaginaryHouse', 'Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff',
hogwarts_houses.remove('AnotherImaginaryHouse')
hogwarts_houses
['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff', 'NewHouse', 'NewHouse1', 'NewHouse2']
```

- pop() -> removes element by index. By default index is -1 so it removes last element of list. Returns the removed element.

```
hogwarts_houses
['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff', 'NewHouse', 'NewHouse1']
hogwarts_houses.pop(4)
'NewHouse'
hogwarts_houses
['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff', 'NewHouse1']
```

- Reverse a list

```
hogwarts_houses
['Gryffindor', 'Slytherin', 'Ravenclaw', 'Hufflepuff']
hogwarts_houses.reverse()
hogwarts_houses
['Hufflepuff', 'Ravenclaw', 'Slytherin', 'Gryffindor']
```

- Sort a list in-place
  This will change the actual list. To change the sorting order to descending, pass an argument reverse=True

```
hogwarts_houses
['Hufflepuff', 'Ravenclaw', 'Slytherin', 'Gryffindor']
hogwarts_houses.sort()
hogwarts_houses
['Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin']

#Reverse sort
hogwarts_houses.sort(reverse=True)
hogwarts_houses
['Slytherin', 'Ravenclaw', 'Hufflepuff', 'Gryffindor']
```

- To get a copy of the sorted list without changing the original list use the sorted() function

```
hogwarts_houses = ['Slytherin', 'Ravenclaw', 'Hufflepuff', 'Gryffindor']
sorted_houses = sorted(hogwarts_houses)

#new list returned by the sorted function
sorted_houses
['Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin']

#Original list is unchanged
hogwarts_houses
['Slytherin', 'Ravenclaw', 'Hufflepuff', 'Gryffindor']
```

- max() and min() -> work on both string lists and numeric lists to find max and min values respectively

```
hogwarts_houses
['Slytherin', 'Ravenclaw', 'Hufflepuff', 'Gryffindor']

min(hogwarts_houses)
'Gryffindor'

max(hogwarts_houses)
'Slytherin'

max([1,2,10,4,5])
10

min([1,2,10,4,5])
1
```

- sum() works only on numeric lists

```
num_list = [1,2,3,4,5]
sum(num_list)
15

#throws error on a string list as expected
sum(hogwarts_houses)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- index() -> returns the first index if an item exists in the list

```
hogwarts_houses
['Slytherin', 'Ravenclaw', 'Hufflepuff', 'Gryffindor']

#index function on a list
hogwarts_houses.index('Gryffindor')
3

#index function on a string
test_string = "abracadabra"
test_string.index('c')
4
```

- Testing if element exists in a list with *in* operator

```
'Ravenclaw' in hogwarts_houses
True
'abc' in hogwarts_houses
False
```

- Looping through a list items with for loop

```
hogwarts_houses
['Slytherin', 'Ravenclaw', 'Hufflepuff', 'Gryffindor']

for house in hogwarts_houses:
    print(house)

Slytherin
Ravenclaw
Hufflepuff
Gryffindor
```

- Looping through list items and indexes with enumerate()
  enumerate function returns both index and item from a list

```
for index,house in enumerate(hogwarts_houses,start=1):
    print(index,house)

(1, 'Slytherin')
(2, 'Ravenclaw')
(3, 'Hufflepuff')
(4, 'Gryffindor')
```

- join() and split() -> join will convert a list to string, join list elements
  split() will split a string into a list of individual elements based on a separator.

```
hogwarts_houses
['Slytherin', 'Ravenclaw', 'Hufflepuff', 'Gryffindor']

hogwarts_house_string =  '*'.join(hogwarts_houses)
hogwarts_house_string
'Slytherin*Ravenclaw*Hufflepuff*Gryffindor'
```

```
hogwarts_subjects = 'Potions,DarkArts,Herbology,Charms'
hogwarts_subject_list = hogwarts_subjects.split(',')

hogwarts_subject_list
['Potions', 'DarkArts', 'Herbology', 'Charms']
```

- Tuples -> just like lists but they are *immutable*

```
#change an item in a list
hogwarts_subject_list
['Potions', 'DarkArts', 'Herbology', 'Charms']
hogwarts_subject_list[0] = 'ancientrunes'
hogwarts_subject_list
['ancientrunes', 'DarkArts', 'Herbology', 'Charms']

#initialize a tuple
tuple1 = ('snape','darkarts')

#try to change a tuple element, it will throw error
tuple1[1] = 'hagrid'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- Sets - > unordered collection of unique elemetnts. DOES NOT ALLOW DUPLICATES.

```
#Initialize a set with duplicate elements
set1 = {'ancientrunes', 'DarkArts', 'Herbology', 'Charms', 'DarkArts'}

#it will remove the duplicate elements
set1
set(['DarkArts', 'ancientrunes', 'Herbology', 'Charms'])
```

- Set Functions

```
#initialize two sets
set1 = {'A', 'B', 'C', 'D'}
set2 = {'A','B', 'X','Y','Z'}

#return all elements in both sets
set1.union(set2)
set(['A', 'C', 'B', 'D', 'Y', 'X', 'Z'])

#return set1-set2 , elements that exist inly in set1
set1.difference(set2)
set(['C', 'D'])

#return common elements in set1 and set2
set1.intersection(set2)
set(['A', 'B'])

#return elements that exist ONLY in set1 and elements that exist ONLY in set2
set1.symmetric_difference(set2)
set(['C', 'D', 'Y', 'X', 'Z'])
```

- Creating empty lists, tuples and sets - look for the gotcha with sets!

```
#create an empty list
list1 = []         //OK
list1 = list()     //OK

#create an empty tuple
t1 = ()            //OK
t1 = tuple()   //OK

set1 = set()    //OK
set1 = {}        //NOT OK!   This creates a dict type object
```

```
1
```