Lists, Tuples and Dictionaries

Lists

Any value or series of values enclosed in `[ ]` and elements seperated using a **comma** are under class List

In [1]:
```
# Creating an empty list

empty_list = []
print("Output of an empty list :"+str(empty_list))
# Creating and intializing Lists
a_list = [1,0.1,'Hello']
print("a_List :"+str(a_list))
```

```
Output of an empty list :[]
a_List :[1, 0.1, 'Hello']
```

Note:

- 1.Use help() to know all the list class methods.
- 2. Trailing commas and white space doesn't make any difference

```
# Can't have any null space between elements
l2 = ['a','foo','c'   ,   ,]
```

In [2]:
```
# Lists can haves lists in them. These can be known as Multi-dimensional l
a_list = [1,2,3,[4,5,6]]
a_list
```

Out[2]:
```
[1, 2, 3, [4, 5, 6]]
```

Some useful list methods

In [3]:
```
# append - Add an element at the end of the array
a_list = []
a_list.append(1) # Can use to append only one value at a time
print(a_list)
```

```
[1]
```

In [4]:
```
# Anything added to list will be appended as it is
a_list.append([3,4,5])
a_list
```

Out[4]:
```
[1, [3, 4, 5]]
```

In [5]:
```
# Extend funnction iterates through whole list and append them indivisuall
a_list.extend((6,7))
print(a_list)
```

```
[1, [3, 4, 5], 6, 7]
```

In [6]:
```
# The index of elements start with 0
for i,j in enumerate(a_list):
    print("index "+str(i)+" - "+str(j))
```

```
index 0 - 1
index 1 - [3, 4, 5]
index 2 - 6
index 3 - 7
```

In [7]:
```
# Accessing indvisual elements using index - Indexing
print("Element at index 1 :"+str(a_list[1]))
```

```
Element at index 1 :[3, 4, 5]
```

In [8]:
```
#Accessing elements using reverse index values

print("Element at last :"+str(a_list[-1]))
```

```
Element at last :7
```

Note:

The above process is called Indexing.

In [9]:
```
# Add elements at specific index using empty list. arguments - index,value
a_list.insert(1,2)
a_list
```

Out[9]:
```
[1, 2, [3, 4, 5], 6, 7]
```

In [10]:
```
# Remove and display an element present at last.
a_list.pop()
```

Out[10]:
```
7
```

In [11]:
```
# Remove element at certain index. Argument - Val
a_list.remove(6)
a_list
```

Out[11]:
```
[1, 2, [3, 4, 5]]
```

List Slicing

We can create a seperate list from another list(which practically is a subset) by mentioning range of indecies The slicing structure is `[start: stop(excluded): step]`

In [12]:
```
# For better understanding

print(slice.__doc__)
```

```
slice(stop)
slice(start, stop[, step])

Create a slice object.  This is used for extended slicing (e.g. a
[0:10:2]).
```

In [13]:
```
a_list = [i for i in range(1,10)]

# subset from index 1 - 8
s_list = a_list[1:9]
s_list
```

Out[13]:
```
[2, 3, 4, 5, 6, 7, 8, 9]
```

In [14]:
```
# Subset of only even index values
a_list[::2]
```

Out[14]:
```
[1, 3, 5, 7, 9]
```

Accessing elements from multi-dimentional / jagged lists

In [15]:
```
a_list = [1,[2,3,4],[5,6,[7,8]]]
print(a_list)
```

```
[1, [2, 3, 4], [5, 6, [7, 8]]]
```

In [16]:
```
a_list[1][0]
```

Out[16]:
```
2
```

In [17]:
```
a_list[2][2][1]
```

Out[17]:
```
8
```

Some list functions

In [18]:
```
a_list = [1.,2.,3.,4.]
```

In [19]:
```
# Summing off the elements
sum(a_list)
```

Out[19]:
```
10.0
```

In [20]:
```
max(a_list)
```

Out[20]:
```
4.0
```

In [21]:
```
min(a_list)
```

Out[21]:
```
1.0
```

In [30]:
```
len(a_list)
```

Out[30]:
```
10
```

In [32]:
```
# Creating a list from a string
groceries = "eggs, bread, butter, maggi"
grocery_list = groceries.split(", ")
print(grocery_list)
```

```
['eggs', 'bread', 'butter', 'maggi']
```

List Comprehention

Let's say we need a list of squares which start from 1 to 10

In [23]:
```
a_list = []
for i in range(1,11):
        a_list.append(i**2)

a_list
```

Out[23]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

We can reduce the same code into a one liner

In [24]:
```
a_list = [i**2 for i in range (1,11)]
a_list
```

Out[24]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

This is called List Comprehention. General structure is `[Variable for variable in iterator]` We can add some conditions into this

In [25]:
```
# Want only even numbers of squares
a_list = [i**2 for i in range (1,11) if i%2 == 0]
a_list
```

Out[25]:
```
[4, 16, 36, 64, 100]
```

This may look cool but the amount of time taken is same. For that we have lambda which run faster than List comprehentions.

In [26]:
```
a_list = list(map(lambda x: x**2, list(range(1,11))))
a_list
```

Out[26]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

You can loop through a list using the in keyword

In [79]:
```
for element in filtered_list:
    print(element)
```

```
1
9
25
49
81
```

Tuples

A tuple is essentially the same as a list, except it is immutable and is faster and more efficient. For example,

In [34]:
```
my_tuple = ("I", "remain", "the", "same")
print(my_tuple)
```

```
('I', 'remain', 'the', 'same')
```

Geographical coordinates are a good use case for tuples

In [36]:
```
coordinates = (4.21, 9.97)
print(coordinates)
```

```
(4.21, 9.97)
```

Tuples can be unpacked

In [65]:
```
x_coordinate, y_coordinate = coordinates
print(x_coordinate)
print(y_coordinate)
```

```
4.21
9.97
```

You can loop through a tuple using the in keyword

In [66]:
```
for coordinate in coordinates:
    print(coordinate)
```

```
4.21
9.97
```

Dictionaries

Dictionaries store data in key/value pairs and can be used to store relationships that give context.

In [70]:
```
basketball_player = {"team": "Raptors", "name": "Leonard", "jersey_number"
print(basketball_player)
```

```
{'team': 'Raptors', 'name': 'Leonard', 'jersey_number': 7}
```

Looking up values are much faster than traversing a list. Values are accessed like this

In [71]:
```
print(basketball_player["name"])
```

```
Leonard
```

You can add values to a list

In [72]:
```
basketball_player["num_championships"] = 2
print(basketball_player)
```

```
{'team': 'Raptors', 'name': 'Leonard', 'jersey_number': 7, 'num_champions
hips': 2}
```

You can remove a key/value pair

In [73]:
```
del(basketball_player["team"])
print(basketball_player)
```

```
{'name': 'Leonard', 'jersey_number': 7, 'num_championships': 2}
```

You can loop over the keys in a dictionary using the in keyword

In [74]:
```
for player_detail_key in basketball_player:
    print(player_detail_key, basketball_player[player_detail_key])
```

```
name Leonard
jersey_number 7
num_championships 2
```

The sorted() function will sort the keys alphabetically

In [75]:
```
for player_detail_key in sorted(basketball_player):
    print(player_detail_key, basketball_player[player_detail_key])
```

```
jersey_number 7
name Leonard
num_championships 2
```

Dictionaries can contain other dictionaries

In [76]:
```
basketball_player = {
    "team": "Raptors",
    "name": {"first_name": "Kawhi", "last_name": "Leonard"},
    "jersey_number": 7}
print(basketball_player)
```

```
{'team': 'Raptors', 'name': {'first_name': 'Kawhi', 'last_name': 'Leonard
'}, 'jersey_number': 7}
```

You can retrieve values from dictionaries within dictionaries

In [78]:
```
print(basketball_player["name"]["last_name"])
```

```
Leonard
```