Data Structures

Tuple

In [1]:
```
a=(1, 1.2, "taxi", False)
a
```

Out[1]:
```
(1, 1.2, 'taxi', False)
```

In [2]:
```
type(a)
```

Out[2]:
```
tuple
```

Count

In [3]:
```
a.count("taxi")
```

Out[3]:
```
1
```

In [4]:
```
a.count(True)#why true is counted as 1 although true is not described?
```

Out[4]:
```
1
```

In [5]:
```
a.count(1.2)
```

Out[5]:
```
1
```

In [6]:
```
a.count(True) #why true is counted as 1 although true is not described?
```

Out[6]:
```
1
```

In [7]:
```
a.count(False)
```

Out[7]:
```
1
```

In [8]:
```
a.count(True)
```

Out[8]:
```
1
```

In [9]:
```
b=(4,4.3,"Ali", True)
```

In [10]:
```
b
```

Out[10]:
```
(4, 4.3, 'Ali', True)
```

In [11]:
```
b.count(True)
```

Out[11]:
```
1
```

In [12]:
```
b.count(False)
```

Out[12]:
```
0
```

Index

In [13]:
```
b.index("Ali")
```

Out[13]:
```
2
```

In [14]:
```
c=(1,2.3,"Hussain", False) # there is no true in this tuple
```

In [15]:
```
c
```

Out[15]:
```
(1, 2.3, 'Hussain', False)
```

In [16]:
```
c.count(True) # still the answer is 1 rather than 0
```

Out[16]:
```
1
```

List

In [17]:
```
lin1=[1, 1.2, "Haseeb", True]
lin1
```

Out[17]:
```
[1, 1.2, 'Haseeb', True]
```

In [18]:
```
type(lin1)
```

Out[18]:
```
list
```

Append

In [19]:
```
lin1.append(False)
lin1
```

Out[19]:
```
[1, 1.2, 'Haseeb', True, False]
```

In [20]:
```
lin1.append(True) #for true it worked perfect let's try again
lin1
```

Out[20]:
```
[1, 1.2, 'Haseeb', True, False, True]
```

In [21]:
```
lin1.append(False)
lin1
```

Out[21]:
```
[1, 1.2, 'Haseeb', True, False, True, False]
```

In [22]:
```
lin2=[1,1.2, "Shaikh", True]
lin2
```

Out[22]:
```
[1, 1.2, 'Shaikh', True]
```

In [23]:
```
b=[3,3.5,"Haseeb", True]
b
```

Out[23]:
```
[3, 3.5, 'Haseeb', True]
```

In [24]:
```
b.append(False) #why false is added twice although I have appended once only?
b
```

Out[24]:
```
[3, 3.5, 'Haseeb', True, False]
```

Clear

In [25]:
```
lin1.clear()
lin1
```

Out[25]:
```
[]
```

In [26]:
```
lin2.clear("Shaikh") # it should be noticed that clear is not
#about clear any single variable in the list but the whole list at once
line2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\ABDULH~1\AppData\Local\Temp/ipykernel_11844/280165805.py in <module>
----> 1 lin2.clear("Shaikh") # it should be noticed that clear is not
      2 #about clear any single variable in the list but the whole list at once
      3 line2

TypeError: list.clear() takes no arguments (1 given)
```

Copy

In [27]:
```
lin2.copy()
lin2
```

Out[27]:
```
[1, 1.2, 'Shaikh', True]
```

In [28]:
```
lin2.copy("Shaikh")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\ABDULH~1\AppData\Local\Temp/ipykernel_11844/1325560725.py in <module>
----> 1 lin2.copy("Shaikh")

TypeError: list.copy() takes no arguments (1 given)
```

In [29]:
```
lin2.copy() # What does copy do exactly? Not sure :(
lin2
```

Out[29]:
```
[1, 1.2, 'Shaikh', True]
```

In [30]:
```
lin1
```

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

In [31]:
```
lin1.copy()
```

Out[31]:
```
[]
```

In [32]:
```
lin2=[1,3, "Laptop", True]
lin2
```

Out[32]:
```
[1, 3, 'Laptop', True]
```

In [33]:
```
lin2
```

Out[33]:
```
[1, 3, 'Laptop', True]
```

In [34]:
```
lin2+lin2
```

Out[34]:
```
[1, 3, 'Laptop', True, 1, 3, 'Laptop', True]
```

Count

In [35]:
```
lin2.count(True) # why 2 through true is once in lin2 List?
```

Out[35]:
```
2
```

In [36]:
```
lin2.count("Laptop")
```

Out[36]:
```
1
```

In [37]:
```
lis3=[True, False]
lis3.count(True)
```

Out[37]:
```
1
```

In [38]:
```
lis3.count(True)
```

Out[38]:
```
1
```

Extend

In [39]:
```
lin2.extend(lis3)
```

In [40]:
```
lin2 # I just assumed its function but it really does same it is nearly same as c
```

Out[40]:
```
[1, 3, 'Laptop', True, True, False]
```

In [41]:
```
lin2.extend(lis3)
lin2
```

Out[41]:
```
[1, 3, 'Laptop', True, True, False, True, False]
```

Index

In [42]:
```
lin2.index(False) #because 0 is also counted so False has 5th index
```

Out[42]:
```
5
```

In [43]:
```
lin2[1:5]
```

Out[43]:
```
[3, 'Laptop', True, True]
```

In [44]:
```
lin2[5:8]
```

Out[44]:
```
[False, True, False]
```

In [45]:
```
lin2[-8:-6]
```

Out[45]:
```
[1, 3]
```

In [46]:
```
lin2.insert(4, "Raheem")
```

In [47]:
```
lin2
```

Out[47]:
```
[1, 3, 'Laptop', True, 'Raheem', True, False, True, False]
```

Remove

In [48]:
```
lin2.remove("Raheem") # it should be noted remove option removes one by one
```

In [49]:
```
lin2
```

Out[49]:
```
[1, 3, 'Laptop', True, True, False, True, False]
```

In [50]:
```
lin2.remove("Raheem")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\ABDULH~1\AppData\Local\Temp/ipykernel_11844/193019633.py in <module>
----> 1 lin2.remove("Raheem")

ValueError: list.remove(x): x not in list
```

In [51]:
```
lin2
```

Out[51]:
```
[1, 3, 'Laptop', True, True, False, True, False]
```

Pop

In [52]:
```
lin2.pop(3) # Pop seems remove the element of the particular index
```

Out[52]:
```
True
```

In [53]:
```
lin2
```

Out[53]:
```
[1, 3, 'Laptop', True, False, True, False]
```

In [54]:
```
lin2.pop(6)
```

Out[54]:
```
False
```

In [55]:
```
lin2("Laptop")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\ABDULH~1\AppData\Local\Temp/ipykernel_11844/2828234519.py in <module>
----> 1 lin2("Laptop")

TypeError: 'list' object is not callable
```

In [56]:
```
lin2
```

Out[56]:
```
[1, 3, 'Laptop', True, False, True]
```

In [57]:
```
lin2.pop(3)
```

Out[57]:
```
True
```

In [58]:
```
lin2
```

Out[58]:
```
[1, 3, 'Laptop', False, True]
```

In [59]:
```
lin2.pop(1)
```

Out[59]:
```
3
```

In [60]:
```
lin2
```

Out[60]:
```
[1, 'Laptop', False, True]
```

In [61]:
```
lin2.pop(0)
```

Out[61]:
```
1
```

In [62]:
```
lin2
```

Out[62]:
```
['Laptop', False, True]
```

In [63]:
```
lin2.pop(0)
```

Out[63]:
```
'Laptop'
```

In [64]:
```
lin2
```

Out[64]:
```
[False, True]
```

Reverse

In [65]:
```
lin4=["Hi",1,1.5,True]
```

In [66]:
```
lin4.reverse()
```

In [67]:
```
lin4
```

Out[67]:
```
[True, 1.5, 1, 'Hi']
```

Sort

In [68]:
```
lin4.sort
```

Out[68]:
```
<function list.sort(*, key=None, reverse=False)>
```

In [69]:
```
lin5=[1,3,1.5,0]
lin5
```

Out[69]:
```
[1, 3, 1.5, 0]
```

In [70]:
```
lin5.sort()
lin5
```

Out[70]: [0, 1, 1.5, 3]

Practise Pop Again

In [71]:
```
lin5
```

Out[71]: [0, 1, 1.5, 3]

In [72]:
```
lin5.pop(2) # 2th index is 1.5 so its given below now 1.5 should not be part of l
```

Out[72]: 1.5

In [73]:
```
lin5 # proved
```

Out[73]: [0, 1, 3]

In [74]:
```
lin5.pop(2) # now 2th index is 3 so 3 is printed
```

Out[74]: 3

In [75]:
```
lin5 # 3 is no more part if lin5-proved
```

Out[75]: [0, 1]

Sets

In [76]:
```
set1={1,1.2,"Ali", True}
```

In [77]:
```
set1 # so boolean is not allowed
```

Out[77]: {1, 1.2, 'Ali'}

In [78]:
```
set1.clear
```

Out[78]: <function set.clear>

In [79]:
```
set1 # have a look set is empty now and brackets are changed into tuple one ()
```

Out[79]: {1, 1.2, 'Ali'}

In [80]:
```
set1={1,1.2,"Ali", True}
```

In [81]:
```
set1
```

Out[81]: {1, 1.2, 'Ali'}

Clear

In [82]:
```
set1.clear()
```

In [83]:
```
set1 # it was expected as this {} but resulted in set()
```

Out[83]:
```
set()
```

In [84]:
```
set2={1,1.2,"Ali", True}
```

In [85]:
```
set2
```

Out[85]:
```
{1, 1.2, 'Ali'}
```

Copy

In [86]:
```
set1.copy(set2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\ABDULH~1\AppData\Local\Temp/ipykernel_11844/1387351528.py in <module>
----> 1 set1.copy(set2)

TypeError: set.copy() takes no arguments (1 given)
```

In [87]:
```
set1.copy()
```

Out[87]:
```
set()
```

In [88]:
```
set2.copy() # Still don't know what does copy do
```

Out[88]:
```
{1, 1.2, 'Ali'}
```

In [89]:
```
set3={2,3,5}
```

In [90]:
```
set4={10,11,12}
```

Difference

In [91]:
```
set3.difference(set4)
```

Out[91]:
```
{2, 3, 5}
```

In [92]:
```
set4
```

Out[92]:
```
{10, 11, 12}
```

In [93]:
```
set4.difference(set4-set3)
```

Out[93]:
```
set()
```

In [94]:
```
set4
```

Out[94]:
```
{10, 11, 12}
```

In [95]:
```
set5={1,2,3,4,4}
```

In [96]:
```
set5 # no repetition bro
```

Out[96]:
```
{1, 2, 3, 4}
```

In [97]:
```
# lets see what difference do now
```

In [98]:
```
difference(set 5-set4)
```

```
  File "C:\Users\ABDULH~1\AppData\Local\Temp/ipykernel_11844/1528959839.py", li
ne 1
    difference(set 5-set4)
                   ^
SyntaxError: invalid syntax
```

In [99]:
```
.difference(set5-set4)
```

```
  File "C:\Users\ABDULH~1\AppData\Local\Temp/ipykernel_11844/2159072251.py", li
ne 1
    .difference(set5-set4)
    ^
SyntaxError: invalid syntax
```

In [100]:
```
set4-set5
```

Out[100]:
```
{10, 11, 12}
```

In [101]:
```
# lets do it with concept of sets in mathematics
```

In [102]:
```
setA={1,2,3,4,5,6}
```

In [103]:
```
setB={4,5,6,7,8,9}
```

In [104]:
```
setA.difference(setB)
```

Out[104]:
```
{1, 2, 3}
```

In [105]:
```
setB.difference(setA)
```

Out[105]:
```
{7, 8, 9}
```

In [106]:
```
# X.difference(Y) = X-Y and Y.difference(X)= Y-X where X and Y are sets
```

In [107]:
```
setA # here it should be noted after difference the actual set remains still same
```

Out[107]:
```
{1, 2, 3, 4, 5, 6}
```

Difference Update

In [108]:
```
setA.difference_update(setB)
```

In [109]:
```
setA # differece-update firstly carries out difference then update the actual cas
```

Out[109]:
```
{1, 2, 3}
```

Intersection

In [110]:
```
setA.intersection(setA)
```

Out[110]:
```
{1, 2, 3}
```

In [111]:
```
setC={1,2,3,4}
```

In [112]:
```
setA.intersection(setC)
```

Out[112]:
```
{1, 2, 3}
```

Intersection Update

In [113]:
```
setC.intersection_update(setA)
```

In [114]:
```
setC # after intersection it is updated
```

Out[114]:
```
{1, 2, 3}
```

Disjoint Boolean

In [115]:
```
setC.isdisjoint(setA) #elements are same
```

Out[115]:
```
False
```

In [116]:
```
setC
```

Out[116]:
```
{1, 2, 3}
```

In [117]:
```
setD={4,5,6}
```

In [118]:
```
setD.isdisjoint(setC) # disjoint means no elements are same
```

Out[118]: True

In [119]:
```
setE={4,5}
```

In [120]:
```
) # when a set has same elements but less than or equal to other set than it is t
```

Out[120]: True

Subset Boolean

In [121]:
```
setF={4,5,6}
setF.issubset(setD)
```

Out[121]: True

In [122]:
```
setG={4,5,6,7}
```

Superset Boolean

In [123]:
```
setG.issuperset(setF) #setG has same elements of setF and more than it or setF is
```

Out[123]: True

Remove

In [124]:
```
setG.remove(6)
```

In [125]:
```
setG # 6 is removed
```

Out[125]: {4, 5, 7}

In [126]:
```
# pop is already tested
```

Symmetric Difference

In [127]:
```
setG.symmetric_difference(setF) # slightly different than difference just removes
```

Out[127]: {6, 7}

In [128]:
```
setG.symmetric_difference_update(setF)
setG # setG is updated also, cool!!
```

Out[128]: {6, 7}

Union

In [129]:
```
setH={1,2}
setH.union(setG) # simply union
```

Out[129]: {1, 2, 6, 7}

Update

In [130]:
```
setH.update(setA)
```

In [131]:
```
setH # setA is {1,2,3}
```

Out[131]: {1, 2, 3}

In [132]:
```
setG.update(setA)
```

In [133]:
```
setG # it works similar to union, the difference is union does not change the act
#set but update do union and change the actual set into the union result
```

Out[133]: {1, 2, 3, 6, 7}

In [134]:
```
setG
```

Out[134]: {1, 2, 3, 6, 7}

Dictionaries

In [135]:
```
Dic1={"Gamesa Turbines":20,"LG Solar Panels":200, "Hydrogren Batteries":10, "Game
Dic1 # it should be noted in the final item you can't get two items with
#same keys in aswer so the last one will appear here one with value 5
```

Out[135]: {'Gamesa Turbines': 5, 'LG Solar Panels': 200, 'Hydrogren Batteries': 10}

In [136]:
```
Dic1
```

Out[136]: {'Gamesa Turbines': 5, 'LG Solar Panels': 200, 'Hydrogren Batteries': 10}

Keys

In [137]:
```
Dic1.keys()
```

Out[137]:
```
dict_keys(['Gamesa Turbines', 'LG Solar Panels', 'Hydrogren Batteries'])
```

Values

In [138]:
```
Dic1.values()
```

Out[138]:
```
dict_values([5, 200, 10])
```

Get

In [139]:
```
Dic1.get("Hydrogren Batteries") # get gives value of partocular string
```

Out[139]:
```
10
```

In [140]:
```
Dic1
```

Out[140]:
```
{'Gamesa Turbines': 5, 'LG Solar Panels': 200, 'Hydrogren Batteries': 10}
```

In [141]:
```
Dic1
```

Out[141]:
```
{'Gamesa Turbines': 5, 'LG Solar Panels': 200, 'Hydrogren Batteries': 10}
```

Items

In [142]:
```
Dic1.items()
```

Out[142]:
```
dict_items([('Gamesa Turbines', 5), ('LG Solar Panels', 200), ('Hydrogren Batte
ries', 10)])
```

In [143]:
```
Dic2={"Tesla EV Cars":3, "Vestas":50}
Dic2.items()
```

Out[143]:
```
dict_items([('Tesla EV Cars', 3), ('Vestas', 50)])
```

In [144]:
```
Dic1={"Gamesa Turbines":20,"LG Solar Panels":200, "Hydrogren Batteries":10, "Game
Dic1
```

Out[144]:
```
{'Gamesa Turbines': 5, 'LG Solar Panels': 200, 'Hydrogren Batteries': 10}
```

Pop Item

In [145]:
```
Dic1.popitem()
```

Out[145]:
```
('Hydrogren Batteries', 10)
```

In [146]:
```
Dic1 # Pop Item removes the last item or item at right most following LIFO (last
# Pop remove the asked index (0th, 1th,....., nth) item only
```

Out[146]:
```
{'Gamesa Turbines': 5, 'LG Solar Panels': 200}
```

From Keys

In [147]:
```
Dic1.fromkeys(Dic2) # fromkeys is still unclear
```

Out[147]:
```
{'Tesla EV Cars': None, 'Vestas': None}
```

In [148]:
```
Dic3={"Gamesa Turbines":5, "Vestas":50}
```

In [149]:
```
Dic1.fromkeys(Dic3)
```

Out[149]:
```
{'Gamesa Turbines': None, 'Vestas': None}
```

In [150]:
```
Dic1
```

Out[150]:
```
{'Gamesa Turbines': 5, 'LG Solar Panels': 200}
```

In [151]:
```
Dic3
```

Out[151]:
```
{'Gamesa Turbines': 5, 'Vestas': 50}
```

In [152]:
```
Dic7=dict.fromkeys(Dic1,6) # implementation slightly different than other functio
```

In [153]:
```
Dic7
```

Out[153]:
```
{'Gamesa Turbines': 6, 'LG Solar Panels': 6}
```

In [154]:
```
Dic8=dict.fromkeys(Dic7,3)
```

In [155]:
```
Dic8 # so from keys simply forms the new dictionary of same keys with the new seq
# for each key of newly created dictionary
```

Out[155]:
```
{'Gamesa Turbines': 3, 'LG Solar Panels': 3}
```

Get (once more to compare with Set Default)

In [156]:
```
Dic3.get("Gamesa Turbines","Vestas") # get gives values of the first asked key
```

Out[156]:
```
5
```

In [157]:
```
Dic9={"a":1, "b":10, "c":5, "d":1.5}
```

In [158]:
```
Dic9.get("d", "b") # get gives value of first asked key independent of value
```

Out[158]:
```
1.5
```

Set Default

In [159]:
```
Dic9.setdefault ("b","c") # what is difference between setdefault and get, they s
```

Out[159]:
```
10
```

In [160]:
```
Dic9
```

Out[160]:
```
{'a': 1, 'b': 10, 'c': 5, 'd': 1.5}
```

In [161]:
```
Dic9.get("d")
```

Out[161]:
```
1.5
```

```
The End
```