## Day Objectives

- Set
- Dictionary
- Functions in Python
- Packages and Modules
- Problem Set on All topics

## Set

- Set is collection of hetrogenious data and it is unOrdered
- Set doesn't allows indexing
- Set doesn't allows duplicate elements
- In set we cann't update the existing value but we can add new values
- set Declaration : variblename = {value1,valu2,.....}

In [1]:

```
#Creation of set
#here we are having two ways to create the set:
#  1.By using set() predefined method
#  2.By using variblename = {}
```

In [4]:

```
s = {"APSSDC",35,55.4,True,"Mastan","APSSDC"}
s
```

Out[4]:

```
{35, 55.4, 'APSSDC', 'Mastan', True}
```

In [5]:

```
dir(set)
```

Out[5]:

```
['__and__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__iand__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__isub__',
 '__iter__',
 '__ixor__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__rand__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__ror__',
 '__rsub__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__xor__',
 'add',
 'clear',
 'copy',
 'difference',
 'difference_update',
 'discard',
 'intersection',
 'intersection_update',
 'isdisjoint',
 'issubset',
 'issuperset',
 'pop',
 'remove',
 'symmetric_difference',
 'symmetric_difference_update',
 'union',
 'update']
```

In [6]:

```
s.add(90)
```

In [7]:

```
s
```

Out[7]:

```
{35, 55.4, 90, 'APSSDC', 'Mastan', True}
```

In [8]:

```
s1 = s.copy()
s1
```

Out[8]:

```
{35, 55.4, 90, 'APSSDC', 'Mastan', True}
```

In [9]:

```
a = {20,40,30,70,40}
b = {30,70,20,10,60}
a-b
```

Out[9]:

```
{40}
```

In [10]:

```
b-a
```

Out[10]:

```
{10, 60}
```

In [11]:

```
a.difference(b)
```

Out[11]:

```
{40}
```

In [12]:

```
a
```

Out[12]:

```
{20, 30, 40, 70}
```

In [18]:

```
a.difference_update(b)
```

In [19]:

```
a
```

Out[19]:

```
{40}
```

In [20]:

```
s
```

Out[20]:

```
{35, 55.4, 90, 'APSSDC', 'Mastan', True}
```

In [21]:

```
s.discard(90)
```

In [22]:

```
s
```

Out[22]:

```
{35, 55.4, 'APSSDC', 'Mastan', True}
```

In [23]:

```
s.discard(20)
```

In [24]:

```
s
```

Out[24]:

```
{35, 55.4, 'APSSDC', 'Mastan', True}
```

In [27]:

```
v = s.pop()
```

In [28]:

```
v
```

Out[28]:

```
True
```

In [29]:

```
s
```

Out[29]:

```
{35, 55.4, 'Mastan'}
```

In [30]:

```
s.add(v)
```

In [31]:

```
s
```

Out[31]:

```
{35, 55.4, 'Mastan', True}
```

In [33]:

```
s.remove(55.4)
```

In [34]:

```
s
```

Out[34]:

```
{35, 'Mastan', True}
```

In [36]:

```
s
```

Out[36]:

```
{35, 'Mastan', True}
```

In [37]:

```
s.remove(True)
```

In [38]:

```
s
```

Out[38]:

```
{35, 'Mastan'}
```

In [40]:

```
a.intersection(b)
```

Out[40]:

```
set()
```

In [41]:

```
k = {20,10,40,50,70}
n = {10,40,80,30,70}
```

In [42]:

```
k.intersection(n)
```

Out[42]:

```
{10, 40, 70}
```

In [43]:

```
s
```

Out[43]:

```
{35, 'Mastan'}
```

In [44]:

```
s.update({"a","IIIT"})
```

In [45]:

```
s
```

Out[45]:

```
{35, 'IIIT', 'Mastan', 'a'}
```

In [46]:

```
k,n
```

Out[46]:

```
({10, 20, 40, 50, 70}, {10, 30, 40, 70, 80})
```

In [47]:

```
k.isdisjoint(n)
```

Out[47]:

```
False
```

In [48]:

```
h = {1,2,3}
g = {4,5,6}

```

In [49]:

```
h.isdisjoint(g)
```

Out[49]:

```
True
```

In [50]:

```
k,n
```

Out[50]:

```
({10, 20, 40, 50, 70}, {10, 30, 40, 70, 80})
```

In [51]:

```
k.issubset(n)
```

Out[51]:

```
False
```

In [52]:

```
r = {1,2,3,4,5}
t = {1,2,3}
t.issubset(r)
```

Out[52]:

```
True
```

In [54]:

```
t.symmetric_difference(r)
```

Out[54]:

```
{4, 5}
```

In [55]:

```
r.symmetric_difference_update(t)
```

In [56]:

```
r
```

Out[56]:

```
{4, 5}
```

In [57]:

```
t
```

Out[57]:

```
{1, 2, 3}
```

In [58]:

```
r.union(t)
```

Out[58]:

```
{1, 2, 3, 4, 5}
```

In [59]:

```
li = [10,20,20,40,50,60,60]
li
```

Out[59]:

```
[10, 20, 20, 40, 50, 60, 60]
```

In [60]:

```
uniqueli = set(li)
```

In [62]:

```
len(uniqueli)
```

Out[62]:

```
5
```

## Dictionary

- It is a collection of hetrogenious data
- If you want store the data in dictionary compulsary you should maintain keys with out keys we cann't add the values in dictionary
- Dictionary will accept key - value based data
- Here keys are unique why beacuse through the keys only we are accessing the values from the dictionary
- Keys will be you can give any data type except boolean
- Decleration of dictionary : variablename = {"key1":value1,"key2":value2,........}

In [63]:

```
#Creation of dictionary
#If you want create the dictionary here we are having two ways
# 1.By using dict() method
# 2.variblename = {}

```

In [64]:

```
d = {"key1":20,55.4:40,6:9,"key2":70.4,"key3":True,"Key4":"APSSDC"}
d
```

Out[64]:

```
{'key1': 20, 55.4: 40, 6: 9, 'key2': 70.4, 'key3': True, 'Key4': 'APSSDC'}
```

In [84]:

```
myd = {"key1":30,"key2":60,"key3":70,"key4":80}
myd
```

Out[84]:

```
{'key1': 30, 'key2': 60, 'key3': 70, 'key4': 80}
```

In [ ]:

```

```

In [85]:

```
myd["key1"]
```

Out[85]:

```
30
```

In [80]:

```
myd["key4"]
```

Out[80]:

```
80
```

In [90]:

```
print(dir(dict))
```

```
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__do
c__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__
repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclassh
ook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popite
m', 'setdefault', 'update', 'values']
```

In [95]:

```
myd.keys()
```

Out[95]:

```
dict_keys(['key1', 'key2', 'key3', 'key4'])
```

In [96]:

```
myd.values()
```

Out[96]:

```
dict_values([30, 60, 70, 80])
```

In [97]:

```
myd.items()
```

Out[97]:

```
dict_items([('key1', 30), ('key2', 60), ('key3', 70), ('key4', 80)])
```

In [98]:

```
myd["key5"]="APSSDC"
```

In [99]:

```
myd
```

Out[99]:

```
{'key1': 30, 'key2': 60, 'key3': 70, 'key4': 80, 'key5': 'APSSDC'}
```

In [100]:

```
myd["key5"]=500
```

In [101]:

```
myd
```

Out[101]:

```
{'key1': 30, 'key2': 60, 'key3': 70, 'key4': 80, 'key5': 500}
```

In [105]:

```
myd.fromkeys({"key6":800})
```

Out[105]:

```
{'key6': None}
```

In [106]:

```
myd
```

Out[106]:

```
{'key1': 30, 'key2': 60, 'key3': 70, 'key4': 80, 'key5': 500}
```

In [110]:

```
myd.fromkeys(('k1','k2','k3'),(10,40,50))
```

Out[110]:

```
{'k1': (10, 40, 50), 'k2': (10, 40, 50), 'k3': (10, 40, 50)}
```

In [109]:

```
myd
```

Out[109]:

```
{'key1': 30, 'key2': 60, 'key3': 70, 'key4': 80, 'key5': 500}
```

In [111]:

```
myd
```

Out[111]:

```
{'key1': 30, 'key2': 60, 'key3': 70, 'key4': 80, 'key5': 500}
```

In [112]:

```
myd["mylist"] = [1,2,3,4,5,6]
```

In [113]:

```
myd
```

Out[113]:

```
{'key1': 30,
 'key2': 60,
 'key3': 70,
 'key4': 80,
 'key5': 500,
 'mylist': [1, 2, 3, 4, 5, 6]}
```

In [117]:

```
#create a static dictionary to store the 4 employee details
# empd = {"empid":[name,mobilno,emailid]}
empd = {"emp121":['mastan',1234567890,"mastanvali.p@apssdc.in"],
       "emp122":['vali',567346658,"vali.p@apssdc.in"]}
```

In [118]:

```
empd
```

Out[118]:

```
{'emp121': ['mastan', 1234567890, 'mastanvali.p@apssdc.in'],
 'emp122': ['vali', 567346658, 'vali.p@apssdc.in']}
```

In [122]:

```python

```

In [126]:

```python
d = {}
key = input("Enter key ")
value = input("Enter value ").split()
d[key]=value
```

```
Enter key 145
Enter value ravi 645634583 ravi@apssdc.in
```

In [127]:

```python
d
```

Out[127]:

```
{'145': ['ravi', '645634583', 'ravi@apssdc.in']}
```

In [129]:

```python
mycontacts = {}
```

In [133]:

```python
#Create a dictionary  to store the contacts of a person
# How may contacts you want to store
# 5
# mycontacts = {"name":[int==>mobileno,emailid]}
n = int(input("Enter how many contacts you want to store "))
for i in range(n):
    li = []
    name = input("Enter your name ")
    values = input("enter Mobileno and emailid with space ").split()
    mobileno = int(values[0])
    emailid = values[1]
    li.append(mobileno)
    li.append(emailid)
    mycontacts[name] = li
print(mycontacts)



```

```
Enter how many contacts you want to store 1
Enter your name mastan
enter Mobileno and emailid with space 5346735734 mastan@gmail.com
{'mastan': [5346735734, 'mastan@gmail.com']}
```

In [140]:

```
#Find the highest frequiency number in the above list

#d.items() will give a list of items
#d.keys() will give a list of keys
#d.values() will give a list of values
li = [10,10,20,40,30,10,50,30,50,20,30,30,40]
d = {}
for i in li:
    d[i] = li.count(i)
    #print(i,end = " ")
maximumvalue = max(d.values())
for item in d.items():
    if maximumvalue == item[1]:
        print("Highest frequency number is : ",item[0])
print(maximumvalue)
print(d)
```

```
Highest frequency number is :  30
4
{10: 3, 20: 2, 40: 2, 30: 4, 50: 2}
```

```
# Functions:

    - To perform a specific task
    - 2 types:
        - Predefine functions
            Ex:print,sum,len.max,min
        - Userdefine functions -> Created by user
    - Syntax:

        def functionname(arguments):
            "function document"
            statements
            return (None)
    - Userdefine functions has 4 types they are:

            - With returntype and with arguments
            - With returntype and without arguments
            - Without returntype and with arguments
            - Without returntype and without arguments
```

In [144]:

```
# With returntype and with arguments
# Reading - Main
# Logic - Function
# Printing - Main

def Sumof2numbers(a,b):
    '''Sample Example for Addition by using Function'''
    return a+b
```

```
3
5
8
```

In [148]:

```
n = int(input())
m = int(input())
print(Sumof2numbers(n,m))
```

```
4
5
9
```

In [143]:

```
Sumof2numbers(10,30)
```

Out[143]:

```
40
```

In [145]:

```
print(Sumof2numbers.__doc__)
```

```
Sample Example for Addition by using Function
```

In [146]:

```
print(print.__doc__)
```

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

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

In [147]:

```
print(len.__doc__)
```

```
Return the number of items in a container.
```

In [162]:

```
# With returntype and without arguments
# Reading - Function
# Logic - Function
# Printing - Main

def Sumof2numberswr():
    '''With return type'''
    n = int(input("Value of n is: "))
    m = int(input("Value of m is: "))
    print("Sum of {} and {} is: ".format(n,m),end="")
    return n+m

print(Sumof2numberswr())
print(Sumof2numberswr())
```

```
Value of n is: 2
Value of m is: 3
Sum of 2 and 3 is: 5
Value of n is: 5
Value of m is: 8
Sum of 5 and 8 is: 13
```

In [168]:

```
# print("a and b".format(3,4))
print("{2},{0} and {1}".format(3,5,7))
```

```
7,3 and 5
```

In [175]:

```
print("{2},{1} and {0}".format(60,40,80,70))
```

```
80,40 and 60
```

In [169]:

```
Sumof2numberswr()
```

```
Value of n is: 3
Value of m is: 4
Sum of 3 and 4 is:
```

Out[169]:

```
7
```

In [170]:

```
# Value of a is:3
# Value of b is:7
# Sum of 3 and 7 is: 10
```

In [177]:

```python
# Without returntype and with arguments
# Reading - Main
# Logic - Function
# Printing - Function

def Sumof2Numberswa(c,d):
    print("Sum of {} and {} is: {}".format(c,d,c+d))
    print("n value is: {}".format(n))
    return

n = int(input())
m = int(input())
Sumof2Numberswa(n,m)
```

```
5
6
Sum of 5 and 6 is: 11
n value is: 5
```

In [178]:

```python
print(n)
```

```
5
```

In [179]:

```python
# Without returntype and without arguments
# Reading - Function
# Printing - Function
# Logic - Function

def Sumof2wta():
    g = int(input())
    h = int(input())
    print("Sum of {} and {} is: {}".format(g,h,g+h))
    return

Sumof2wta()
```

```
4
7
Sum of 4 and 7 is: 11
```

```
    Function Arguments has 4 types they are:
        - Required argument
        - Keyword argument
        - Default argument
        - Value-length argument
```

In [181]:

```
# Required argument:

def Username(st):
    print("Username is: ",st)
    return

Username("raju")

```

```
Username is:  raju
```

In [185]:

```
# Keyword argument

def Username(name,msg):
    print("Hello {} Your msg is: {}".format(name,msg))
    return

Username(msg="hi welcome",name="Raju")
```

```
Hello Raju Your msg is: hi welcome
```

In [192]:

```
# Default argument

def Username(age,name="Latha"):
    print("Your name is:{} and age is:{}".format(name,age))
    return

Username(age=50,name="Rajesh")
Username(age=20)
```

```
Your name is:Rajesh and age is:50
Your name is:Latha and age is:20
```

In [194]:

```
# Value-Length argument

def Usernames(*n):
    print(type(n))
    print(n)
    return

Usernames('rajesh','raju','giri','kiran')
```

```
<class 'tuple'>
('rajesh', 'raju', 'giri', 'kiran')
```

In [199]:

```
def SumofNumbers(*h):
#     print(sum(h))
    su=0
    for i in h:
        if i%2==0:
            print(i,su)
            su+=i
    print(su)
    return

SumofNumbers(34,12,45,1,2,48,100,102,0)
```

```
6 0
8 6
12 14
26
```

In [196]:

```
t = (1,2,3)
print(sum(t))
```

```
6
```

In [ ]:

```

```