In [ ]:
```
recap
int float bool None complex
operator
conditional statements
Looping statements
```

In [ ]:
```
int, float, bytes, none---> scalar

collection
|----seq     list, str, tuple - ordered - index based access
|-----mapping dict, set unordered -key based access
```

In [14]:
```
s="python"
#  s | p| y |t |h |o |n
#      0  1   2  3 4   5
#                    -2  -1
print(s[3])
print(s[-3])
```

```
h
h
```

In [15]:
```
x=s[3]
print(x)
```

```
h
```

In [16]:
```
s[3]='x' # immutable- umodifyable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 1
----> 1 s[3]='x'

TypeError: 'str' object does not support item assignment
```

In [17]:
```
for i in s:
    print(i)
```

```
p
y
t
h
o
n
```

In [18]:
```
#slicing operation- sub string

#  stringname[n:m]    --> str starts fro index n to index() m-1)

print(s[2:4])
```

```
th
```

In [20]:
```
msg="The python programming "
print(msg[5:])
```

```
ython programming
```

In [21]:
```
print(msg[:10])
```

```
The python
```

In [23]:
```
print(s,s[0],s[1],s[2])
print(type(s),type(s[0]),type(s[1]))
```

```
python p y t
<class 'str'> <class 'str'> <class 'str'>
```

In [24]:
```
help(str)
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
```

In [25]:
```
s.upper()
```

Out[25]:
```
'PYTHON'
```

In [26]:
```
s.islower()
```

Out[26]:
```
True
```

In [27]:
```
s="abcd"
print(s[10])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[27], line 2
      1 s="abcd"
----> 2 print(s[10])

IndexError: string index out of range
```

In [28]:
```
s1="12"
s2="ab"

i=int(s1)+100
j=int(s2)
print(i,j)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[28], line 5
      2 s2="ab"
      4 i=int(s1)+100
----> 5 j=int(s2)
      6 print(i,j)

ValueError: invalid literal for int() with base 10: 'ab'
```

In [29]:
```
print(s3)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[29], line 1
----> 1 print(s3)

NameError: name 's3' is not defined
```

In [30]:
```
class ABC:
    pass

obj=ABC()

print(obj.s1)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[30], line 6
      2     pass
      4 obj=ABC()
----> 6 print(obj.s1)

AttributeError: 'ABC' object has no attribute 's1'
```

In [32]:
```
#List - collection of different datatype- mutable- index based- ordered  -  []
emp_name="Harish"
empId=1212
e_loginStatus=True
emp=[emp_name,empId,e_loginStatus]
```

In [33]:
```
print(emp)
```

```
['Harish', 1212, True]
```

In [34]:
```
print(type(emp))
```

```
<class 'list'>
```

In [35]:
```
print(emp[0],emp[1],emp[2])
```

```
Harish 1212 True
```

In [36]:
```
print(type(emp), type(emp[0]),type(emp[1]))
```

```
<class 'list'> <class 'str'> <class 'int'>
```

In [37]:
```
for i in emp:
    print(i)
```

```
Harish
1212
True
```

In [38]:
```
#membership operator--->  in  not in

'i' in 'python'
```

Out[38]:
```
False
```

```
In [39]: 'i' not in 'python'

Out[39]: True

In [52]: L=[]

In [41]: print(L,type(L))

         [] <class 'list'>

In [53]: # Listsname.append(item)   vs  L.insert()

         L.append("Data1")
         L.append("Data2")
         L.append("Data3")
         print(L)

         ['Data1', 'Data2', 'Data3']

In [47]: L.insert(2,"Data5")
         print(L)

         ['Data1', 'Data1', 'Data5', 'Data2', 'Data3']

In [50]: print(L[0])
         print(L[-1])

         Data1
         Data3

In [55]: print(L[:2])    #slicing
         print(L[1:])
         print(L[:])

         ['Data1', 'Data2']
         ['Data2', 'Data3']
         ['Data1', 'Data2', 'Data3']

 In [ ]: # len()  vs    "abc".format()

         #function  vs   method
         #help(), len(), type(),int(), float(),str(), complex()

In [56]: s="Python is an Interpreted language"
         len(s)

Out[56]: 33
```

In [58]:
```
print(L)
len(L)
```

```
['Data1', 'Data2', 'Data3']
```

Out[58]:
```
3
```

In [ ]:
```
# append(), insert()---> add item to list
# pop, remove
```

In [42]:
```
help(L)
```

```
Help on list object:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
```

In [59]:
```
print(L)
```

```
['Data1', 'Data2', 'Data3']
```

In [60]:
```
L.pop()  # removes last item
```

Out[60]:
```
'Data3'
```

In [61]:
```
print(L)
```

```
['Data1', 'Data2']
```

In [62]:
```
L.pop(0)
```

Out[62]:
```
'Data1'
```

In [63]:
```
print(L)
```

```
['Data2']
```

In [67]:
```
L=["data1","data2","data3","data4"]
L.remove("data3")
```

In [68]:
```
print(L)
```
```
['data1', 'data2', 'data4']
```

In [71]:
```
re=L.remove("data2")
print(type(re))
```
```
<class 'NoneType'>
```

In [ ]:
```
L.pop(index)---> item that was removed   vs  L.remove(Value)----> None
```

In [72]:
```
del(s)
```

In [73]:
```
print(s)
```
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[73], line 1
----> 1 print(s)

NameError: name 's' is not defined
```

In [74]:
```
del(L[0])
```

In [75]:
```
L
```
Out[75]:
```
['data4']
```

In [76]:
```
s="arun,sales,bengaluru,190000\n"
print(s)
```
```
arun,sales,bengaluru,190000

```

In [77]:
```
with open("E:\\emp.csv",'r') as robj:
   # robj.read()          #   entire file content as single str
     L=robj.readlines()   # each line as each item of list
```

In [78]:
```
L
```

Out[78]:
```
['eid,ename,edept,eplace,ecost\n',
 '101,raj,sales,pune,1000\n',
 '102,leo,prod,bglore,2000\n',
 '103,paul,HR,chennai,3000\n',
 '104,anu,hr,hyderabad,4000\n',
 '456,kumar,sales,bglore,3000\n',
 '105,zion,Hr,mumbai,5000\n',
 '106,bibu,sales,bglore,1450\n',
 '107,theeb,sales,noida,4590\n',
 '108,bibu,sales,bglore,5000']
```

In [79]:
```
with open("E:\\emp.csv",'r') as robj:
    s=robj.read()          #   entire file content as single str
    #L=robj.readlines()   # each line as each item of list
```

In [80]:
```
s
```

Out[80]:
```
'eid,ename,edept,eplace,ecost\n101,raj,sales,pune,1000\n102,leo,prod,bglore,2
000\n103,paul,HR,chennai,3000\n104,anu,hr,hyderabad,4000\n456,kumar,sales,bgl
ore,3000\n105,zion,Hr,mumbai,5000\n106,bibu,sales,bglore,1450\n107,theeb,sale
s,noida,4590\n108,bibu,sales,bglore,5000'
```

In [ ]:
```

with open(filename,mode)
                       |-------r  , w
open(filename)---> opens file for read
```

In [98]:
```
e1= '101,raj,sales,pune,1000\n'

e1.rstrip()
print(type(e1))
```

```
<class 'str'>
```

In [94]:
```
help(e1.strip)   #lstrip()   rstrip()
```

```
Help on built-in function strip:

strip(chars=None, /) method of builtins.str instance
    Return a copy of the string with leading and trailing whitespace removed.

    If chars is given and not None, remove characters in chars instead.
```

In [99]:
```
e1
```

Out[99]:
```
'101,raj,sales,pune,1000\n'
```

```
In [105]: e1= '101,raj,sales,pune,1000\n'
          e1.strip()

Out[105]: '101,raj,sales,pune,1000'
```

```
In [111]: e2=e1.strip()
          L2=e2.split(",")
          print(L2[1],L2[-1])

          raj 1000
```

```
In [112]: L2[1]

Out[112]: 'raj'
```

```
In [113]: L2[-1]

Out[113]: '1000'
```

In [116]:
```
'''
Task1
====
1. create an empty List L
2. Add server1, server2, server4, server5 as items to list
        2A. display the list
3. Insert server3  to the list L  at index 2
        3A. display the list
4. delete server5 from the list --> try with pop()
5. delete server1 from list  ----> try with remove()
6. delete server2 from list --> try with del()

'''


L=[]

L.append("server1")
L.append("server2")
L.append("server4")
L.append("server5")
print(L)

L.insert(2,"server3")
print(L)

L.pop(-1)
print(L)

L.remove("server1")
print(L)

del(L[0])
print(L)
```

```
['server1', 'server2', 'server4', 'server5']
['server1', 'server2', 'server3', 'server4', 'server5']
['server1', 'server2', 'server3', 'server4']
['server2', 'server3', 'server4']
['server3', 'server4']
```

In [119]:
```
'''
Task2
=====
Given string s1="Yamal,1322,True,3.4\n"

1. strip the given string
2. split the string and store it in variables like app_name, port_no, running_s
3. display the app details
'''
s1="Yamal,1322,True,3.4\n"
s2= s1.strip()
L=s2.split(",")
app_name=L[0]
port_no=L[1]
running_status=L[2]
version=L[3]

print(f'''Application Name = {app_name}
Port = {port_no}
App Running Status = {running_status}
Version = {version}''')
```

```
Application Name = Yamal
Port = 1322
App Running Status = True
Version = 3.4
```

In [123]:
```
s1="Yamal,1322,True,3.4\n"
s2= s1.strip()
L=s2.split(",")

app_name,port_no,running_status=L   # multiple assignment  - value error

print(f'''Application Name = {app_name}
Port = {port_no}
App Running Status = {running_status}
Version = {version}''')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[123], line 5
      2 s2= s1.strip()
      3 L=s2.split(",")
----> 5 app_name,port_no,running_status=L   # multiple assignment
      7 print(f'''Application Name = {app_name}
      8 Port = {port_no}
      9 App Running Status = {running_status}
     10 Version = {version}''')

ValueError: too many values to unpack (expected 3)
```

In [125]:
```
s1="Yamal,1322,True,3.4\n"
s2= s1.strip()
L=s2.split(",")

app_name,port_no,running_status,version=L   # multiple assignment

print(f'''Application Name = {app_name}
Port = {port_no}
App Running Status = {running_status}
Version = {version}''')
```

```
Application Name = Yamal
Port = 1322
App Running Status = True
Version = 3.4
```

In [127]:
```
s1="Yamal,1322,True,3.4\n"

app_name,port_no,running_status,version= s1.strip().split(",")

print(f'''Application Name = {app_name}
Port = {port_no}
App Running Status = {running_status}
Version = {version}''')
```

```
Application Name = Yamal
Port = 1322
App Running Status = True
Version = 3.4
```

In [128]:
```
'''
Task 3
=====
step1: create an empty list filenames
step2: use while loop and limit it to 5 times
step3: read an input filename from user and append it to the List filenames
step4: using for loop display the content of list filenames.

'''
filenames=[]

i=1
while(i<=5):
    file=input("Enter a Filename")
    filenames.append(file)
    i=i+1    #     i+=1

for i in filenames:
    print(i)
```

```
Enter a Filenamef1.java
Enter a Filenamef2.sh
Enter a Filenamef3.py
Enter a Filenamef4.pyo
Enter a Filenamef5.c
f1.java
f2.sh
f3.py
f4.pyo
f5.c
```

In [ ]:
```
Tuple--> collection of different datatype values - ordered - index based- immut
```

In [129]:
```
L
```

Out[129]:
```
['Yamal', '1322', 'True', '3.4']
```

In [130]:
```
L[0]= "Prometheus"    # mutable
print(L)
```

```
['Prometheus', '1322', 'True', '3.4']
```

In [131]:
```
var=1,2
print(type(var))
```

```
<class 'tuple'>
```

In [132]:
```
T=() # empty tuple
```

```
In [134]: T.append("data")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[134], line 1
----> 1 T.append("data")

AttributeError: 'tuple' object has no attribute 'append'
```

```
In [135]: help(T)
```

```
Help on tuple object:

class tuple(object)
 |  tuple(iterable=(), /)
 |
 |  Built-in immutable sequence.
 |
 |  If no argument is given, the constructor returns an empty tuple.
 |  If iterable is specified the tuple is initialized from iterable's items.
 |
 |  If the argument is a tuple, the return value is the same object.
 |
 |  Built-in subclasses:
 |      asyncgen_hooks
 |      UnraisableHookArgs
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |  
 |  __ne__(self, value, /)
 |      Return self!=value.
 |  
 |  __repr__(self, /)
 |      Return repr(self).
 |  
 |  __rmul__(self, value, /)
 |      Return value*self.
 |  
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |  
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |      
 |      Raises ValueError if the value is not present.
 |  
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |  
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585
 |  
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |  
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signatur
e.
```

In [136]:
```python
L1=list(T)
print(L1, type(L1))
```

```
[] <class 'list'>
```

In [137]:
```python
T=(10,20,30)
```

In [138]:
```python
L=list(T)
L.append(40)
print(L,type(L))
T=tuple(L)
print(T,type(T))
```

```
[10, 20, 30, 40] <class 'list'>
(10, 20, 30, 40) <class 'tuple'>
```

In [140]:
```python
T=(10,20,10,30,40)
T.count(10)
```

Out[140]:
```
2
```

```
In [ ]: # List- mutable- ordered- index based - [] - allows duplicates
        #tuple - immutable- ordered-index based- () - allows duplications
```

```
In [ ]: #dict- key:value pair - unorderd - key based access- allows duplicates only in
        # keys are unique
```

```
In [141]: 
          d={(1,2):"data"} # valid
```

```
In [142]: d={1:"data"} # valid
```

```
In [143]: d={"key1":"value1"}   # recommended
```

```
In [144]: #dictname[keyname] --> value                  Vs      dictname.setdefault(newke
          d["key1"]
Out[144]: 'value1'
```

```
In [145]: d={}
```

```
In [146]: d["key2"]="value"
```

```
In [147]: d
Out[147]: {'key2': 'value'}
```

```
In [148]: d.setdefault("key3","value3")
Out[148]: 'value3'
```

```
In [149]: d
Out[149]: {'key2': 'value', 'key3': 'value3'}
```

In [150]:
```
#help(d)   or help(dict)
help(d)
```

```
Help on dict object:

class dict(object)
 |  dict() -> new empty dictionary
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
 |  dict(iterable) -> new dictionary initialized as if via:
 |      d = {}
 |      for k, v in iterable:
 |          d[k] = v
 |  dict(**kwargs) -> new dictionary initialized with the name=value pairs
 |      in the keyword argument list.  For example:  dict(one=1, two=2)
 |
 |  Built-in subclasses:
 |      StgDict
 |
 |  Methods defined here:
 |
 |  __contains__(self, key, /)
```

In [151]:
```
d.get("key3")          #    dictname["key"]---> value
```

Out[151]:
```
'value3'
```

In [152]:
```
d.items()
```

Out[152]:
```
dict_items([('key2', 'value'), ('key3', 'value3')])
```

In [154]:
```
d.keys()
```

Out[154]:
```
dict_keys(['key2', 'key3'])
```

In [155]:
```
d.values()
```

Out[155]:
```
dict_values(['value', 'value3'])
```

In [ ]:
```
#dictname["oldkey"]= "newvalue"  vs dictname.setdefault(oldkey,newvalue)
```

In [157]:
```
del(d["key2"])
```

In [158]:
```
d
```

Out[158]:
```
{'key3': 'value3'}
```

In [ ]:
```
#dict- key:value pair - unorderd - key based access- allows duplicates only in
# keys are unique
```

In [162]:
```
d={}
hostname=input("Enter the hostname")
ipaddr=input("Enter the IP address")

d[hostname]=ipaddr  # dictname[key]=value
print(d,len(d))
```

```
Enter the hostnamehost01
Enter the IP address192.168.1.1
{'host01': '192.168.1.1'} 1
```

In [163]:
```

hostname=input("Enter the hostname")
ipaddr=input("Enter the IP address")

d.setdefault(hostname,ipaddr)  # dictname.setdefault(newkey,newvalue)
print(d,len(d))
```

```
Enter the hostnamehost02
Enter the IP address192.168.1.2
{'host01': '192.168.1.1', 'host02': '192.168.1.2'} 2
```

In [1]:
```
'''
2. Q1. Write  a python program

Given tuple Products=(“P1”,”P2”,”P3”,”P4”,”P5”)

display the list of products except P2 and P3

Note :use for loop statement

'''
Products=("P1","P2","P3","P4","P5")
for i in Products:
    if i=="P2" or i=="P3":    # if i in ("P1","P4","P5")
        continue
    print(i)
```

```
P1
P4
P5
```

In [3]:
```
'''
3.Write a python program
Step 1 : create an empty dict
Step 2 : use looping statements – 5times
                 i) Read a hostname from <STDIN>
                 ii) Read a IP-Address from <STDIN>
                 iii) Add a input details to existing dict
                 iv) with hostname as a key and IP address as it’s value

Step 4 : display Key/ value details to monitor
'''
d={}
i=1
while(i<=5):
    host=input("Enter a hostname")
    ip=input("Enter the Ip address")
    d.setdefault(host,ip)            # d[host]=ip
    i=i+1
print(d)
```

```
Enter a hostnamehost01
Enter the Ip address192.168.1.2
Enter a hostnamehost02
Enter the Ip address198.162.1.3
Enter a hostnamehost03
Enter the Ip address192.168.1.9
Enter a hostnamehost04
Enter the Ip address168.192.1.0
Enter a hostnamehost05
Enter the Ip address192.168.1.1
{'host01': '192.168.1.2', 'host02': '198.162.1.3', 'host03': '192.168.1.9',
'host04': '168.192.1.0', 'host05': '192.168.1.1'}
```

In [4]:
```
'''
4.
Step 1: Modify the above code

Step 2: Use membership operator to test whether the input hostname already exis

Step 3: if it’s exists already, display pop up message “Sorry your input hostna

'''
d={}
i=1
while(i<=5):
    host=input("Enter a hostname")
    if host in d.keys():
        print("Sorry your input hostname exist, Try again")
        continue
    ip=input("Enter the Ip address")                  #d.setdefault(old_hostnam
    d.setdefault(host,ip)            # d[host]=ip
    i=i+1
print(d)
```

```
Enter a hostnamehost01
Enter the Ip address192.189.1.2
Enter a hostnamehost01
Sorry your input hostname exist, Try again
Enter a hostnamehost02
Enter the Ip address1.2.3.4
Enter a hostnamehost03
Enter the Ip address1.92.169.1.4
Enter a hostnamehost04
Enter the Ip address1.2.3.4
Enter a hostnamehost05
Enter the Ip address2.3.4.5
{'host01': '192.189.1.2', 'host02': '1.2.3.4', 'host03': '1.92.169.1.4', 'hos
t04': '1.2.3.4', 'host05': '2.3.4.5'}
```

In [5]:
```
d
```

Out[5]:
```
{'host01': '192.189.1.2',
 'host02': '1.2.3.4',
 'host03': '1.92.169.1.4',
 'host04': '1.2.3.4',
 'host05': '2.3.4.5'}
```

In [6]:
```
L=[1,2,3,4]
for i in L:
    print(i)
```
```
1
2
3
4
```

In [7]:
```
d={"k1":"v1","k2":"v2","k3":"v3"}
```

In [9]:
```
for i in d:
    print(i)
```
```
k1
k2
k3
```

In [10]:
```
for i in d:
    print(d[i])
```
```
v1
v2
v3
```

In [11]:
```
for i in d:
    print(i,d[i])    # key value
```
```
k1 v1
k2 v2
k3 v3
```

In [14]:
```
for i in d.items():
    print(i)
```
```
('k1', 'v1')
('k2', 'v2')
('k3', 'v3')
```

In [15]:
```
for i in d.keys():
    print(i,d[i])
```
```
k1 v1
k2 v2
k3 v3
```

In [16]:
```
print(type(d.keys()))
```
```
<class 'dict_keys'>
```

In [17]:
```
for i in d.values():
    print(i)
```

```
v1
v2
v3
```

In [ ]:
```
Set     -    keybased - avoids duplicates --   key only structure  - {key}
===
```

In [18]:
```
s=set()  # oop concept- constructor
print(type(s))
```

```
<class 'set'>
```

In [19]:
```
help(set)
```

```
Help on class set in module builtins:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
```

In [20]:
```
s.add(10)
```

In [21]:
```
s.add(20)
```

In [22]:
```
print(s)
```

```
{10, 20}
```

In [23]:
```
s.update([10,20,30])
print(s)
```

```
{10, 20, 30}
```

In [24]:
```
s.add(30)
print(s)
```

```
{10, 20, 30}
```

In [26]:
```
s.add(40)
print(s)
```

```
{40, 10, 20, 30}
```

In [37]:
```
L=[10,20,30,40,50,1,2,3,4,5,12,3,4,5,1,2,3]
s=set(L)
print(s)
```

```
{1, 2, 3, 4, 5, 40, 10, 12, 50, 20, 30}
```

In [38]:
```

print(s)
s.remove(1)

```

```
{1, 2, 3, 4, 5, 40, 10, 12, 50, 20, 30}
```

In [39]:
```
print(s)
```

```
{2, 3, 4, 5, 40, 10, 12, 50, 20, 30}
```

In [40]:
```
s.discard(5)
print(s)
```

```
{2, 3, 4, 40, 10, 12, 50, 20, 30}
```

In [41]:
```
s.remove(100)
print()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[41], line 1
----> 1 s.remove(100)
      2 print()

KeyError: 100
```

In [42]:
```
s.discard(100)
```

In [43]:
```
A={"p1.c","p2.java","p3.sh","Demo"}
B={"p1.java","p2.java","p1.c","p3.sh","D1","Demo"}

print("Common files in A abd B ", A&B)
print("Common files in A abd B ", A.intersection(B))

print("Combining files in A and B ", A.union(B))
print("Combining files in A and B ", A|B)
```

```
Common files in A abd B  {'p1.c', 'p3.sh', 'Demo', 'p2.java'}
Common files in A abd B  {'p1.c', 'p3.sh', 'Demo', 'p2.java'}
Combining files in A and B  {'p2.java', 'D1', 'p3.sh', 'Demo', 'p1.java', 'p
1.c'}
Combining files in A and B  {'p2.java', 'D1', 'p3.sh', 'Demo', 'p1.java', 'p
1.c'}
```

In [ ]:
```
function--> reusuable -
|----------------------------> func defn
                                    syntax:-
                                                  def functionname(parameters):
                                                         statement
                    |--------> function call
                                                     functionname()
```

In [44]:
```
def display():   # simple function defn without input arg/parameter
    print("Im inside the display function")
```

In [45]:
```
display
```

Out[45]:
```
<function __main__.display()>
```

In [46]:
```
display()
```

```
Im inside the display function
```

In [47]:
```
type(display)
```

Out[47]:
```
function
```

In [50]:
```
def f1():
    print("Im inside F1")

def f2():
    print("Im inside F2")

f2()
f1()
```

```
Im inside F2
Im inside F1
```

In [52]:
```
def f1():
    print("Im inside F1")
    f2()
    print("Out of F1")

def f2():
    print("IM inside F2")
    print("Out of F2")

print("Start of Main Script")
f1()
print("End of Main")
```

```
Start of Main Script
Im inside F1
IM inside F2
Out of F2
Out of F1
End of Main
```

In [57]:
```
def display_var():
    var=1000
    print("End of function display", var)

print("main script started")
display_var()
print("Still in main script")
print(var)                                  # var is local to function
```

```
main script started
End of function display 1000
Still in main script

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[57], line 8
      6 display_var()
      7 print("Still in main script")
----> 8 print(var)

NameError: name 'var' is not defined
```

In [59]:
```
#1. return
def display_var():
    var=1000
    print("End of function display", var)
    return var

print("main script started")
var1=display_var()
print("Still in main script")
print(var1)
```

```
main script started
End of function display 1000
Still in main script
1000
```

In [60]:
```
#2. global

def display_var():
    global var                    # global var declaration
    var=1000
    print("End of function display", var)


print("main script started")
display_var()
print("Still in main script")
print(var)
```

```
main script started
End of function display 1000
Still in main script
1000
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```