Variables and GK

```
#comment
'''Multiple
Line
comments'''
INT_VAR=1
FLOAT_VAR=1.5
STRING_VAR="PYTHON" # non mutable, can not be changes once created
print(STRING_VAR)
type(STRING_VAR)
Temp=5
```

```
PYTHON
```

```
print("Before:",Temp)
del Temp
# print("After:",Temp) # Will generate error
```

```
Before: 5
```

Data Types

```
LIST_VAR=[] # Mutable , can be changed, Mixed Data type
TUPPLE_VAR=() # non-mutable, Can not be modified, Same data type
DICT_VAR={} # Key and value concept, Keys are unique
```

```
LIST_VAR.append(11)
LIST_VAR.append(1.5)
LIST_VAR.append("Hi")
LIST_VAR.append("Hello")#Add to last position
print(LIST_VAR)
LIST_VAR.insert(0,55)#add 55 at 0th location
print(LIST_VAR)
LIST_VAR.extend([1,2,4,11])# Add new list at the end
print(LIST_VAR)
LIST_VAR.reverse()# Reverse the list
print(LIST_VAR)
LIST_VAR.pop()# Remove last location element
print(LIST_VAR)
LIST_VAR.pop(0)# Remove 0th location element
print(LIST_VAR)
LIST_VAR.remove('Hello')# Remove given element
print(LIST_VAR)
print("Location of 11:",LIST_VAR.index(11))
print("Location of 11:",LIST_VAR.count(11))
```

```
[11, 1.5, 'Hi', 'Hello']
[55, 11, 1.5, 'Hi', 'Hello']
[55, 11, 1.5, 'Hi', 'Hello', 1, 2, 4, 11]
[11, 4, 2, 1, 'Hello', 'Hi', 1.5, 11, 55]
[11, 4, 2, 1, 'Hello', 'Hi', 1.5, 11]
[4, 2, 1, 'Hello', 'Hi', 1.5, 11]
[4, 2, 1, 'Hi', 1.5, 11]
Location of 11: 5
Location of 11: 1
```

```
LIST_VAR=[1,4,8,7,2,5,6,9,2]
print(sorted(LIST_VAR))#sort list
LIST_VAR.sort()#Sort in ascending order
```

```
print(LIST_VAR)
LIST_VAR.reverse()# Reverse the list
print(LIST_VAR)
print(sum(LIST_VAR))
print(max(LIST_VAR))
print(min(LIST_VAR))
```

```
[1, 2, 2, 4, 5, 6, 7, 8, 9]
[1, 2, 2, 4, 5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 2, 2, 1]
44
9
1
```

```
TUPPLE_VAR=tuple(LIST_VAR)
print(TUPPLE_VAR)
print(TUPPLE_VAR[0])
print(max(TUPPLE_VAR))
print(min(TUPPLE_VAR))
print(sum(TUPPLE_VAR))
sorted(TUPPLE_VAR)
print(TUPPLE_VAR.count(2))
print(TUPPLE_VAR.index(6))
#TUPPLE_VAR[0]=5 # Generare Erroe
```

```
(9, 8, 7, 6, 5, 4, 2, 2, 1)
9
9
1
44
2
3
```

```
TUPPLE_VAR=(1,2,3,4)
print(TUPPLE_VAR)
print(*TUPPLE_VAR)#Print all elements in line by space
print(TUPPLE_VAR[-1])
```

```
(1, 2, 3, 4)
1 2 3 4
4
```

```
DICT_VAR={'Mehul':'CE','Virag':'SE','Khevin':'ISE'}# Create dictionary
DICT_VAR['Utsav']='EE'# Add new element
print(DICT_VAR)
DICT_VAR['Mehul']={'BE':'EC','MS':'CE'}# Add new element with nested Dictionary
print(DICT_VAR)
print(DICT_VAR['Mehul'])
print(DICT_VAR['Mehul']['BE'])
print(DICT_VAR.keys())
print(DICT_VAR.values())
DICT_VAR.pop('Utsav')# Remove entry
print(DICT_VAR)
DICT_VAR.clear()
print(DICT_VAR)
```

```
{'Mehul': 'CE', 'Virag': 'SE', 'Khevin': 'ISE', 'Utsav': 'EE'}
{'Mehul': {'BE': 'EC', 'MS': 'CE'}, 'Virag': 'SE', 'Khevin': 'ISE', 'Utsav': 'EE'}
{'BE': 'EC', 'MS': 'CE'}
EC
dict_keys(['Mehul', 'Virag', 'Khevin', 'Utsav'])
dict_values([{'BE': 'EC', 'MS': 'CE'}, 'SE', 'ISE', 'EE'])
{'Mehul': {'BE': 'EC', 'MS': 'CE'}, 'Virag': 'SE', 'Khevin': 'ISE'}
{}
```

```
LIST_VAR=[1,2,4,4,2,1,5,6,5]
x = list(map(str,LIST_VAR)) # apply str function on each element of LIST_VAR
print("List of students: ", x)
```

```
x = [int(x) for x in input("Enter 3 values").split()]
print(x)
print(2 in x)#Check for members
```

```
List of students:  ['1', '2', '4', '4', '2', '1', '5', '6', '5']
Enter 3 values1 2 3
[1, 2, 3]
True
```

```
NAME="MEHUL"
print("".join(reversed(NAME)))# Reverse string
print(NAME[1:4])
print(NAME[1:-2])
```

```
LUHEM
EHU
EH
```

```
MULTI_DIM_LIST=[[1,2,3],[4,5,6]]
print(MULTI_DIM_LIST)
print(MULTI_DIM_LIST[0])
print(MULTI_DIM_LIST[0][0])
print("Len:",len(MULTI_DIM_LIST))#Row count
print("Len:",len(MULTI_DIM_LIST[0]))# Column count
```

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

```
SET_VAR=set(LIST_VAR)# Type cast
print(SET_VAR)
SET_VAR.add(7)# Add element
print(SET_VAR)
SET_VAR.add(1)# Already exits so will not add
print(SET_VAR)
SET_VAR.update([8,9])#Add list of 2+ elements with update
print(SET_VAR)
SET_VAR.update("AABBCC")# Add string with update
print(SET_VAR)
SET_VAR.update((10,10)) # Add tuple
print(SET_VAR)
SET_VAR.remove('B')# Remove element
print(SET_VAR)
SET_VAR.discard('C')# Remove element
print(SET_VAR)
SET_VAR.pop()#Remove 1st element
print(SET_VAR)
for i in SET_VAR:
  print(i,end=" ")
print("")
SET_VAR.clear()# Remove all elements
print(SET_VAR)
```

```
{1, 2, 4, 5, 6}
{1, 2, 4, 5, 6, 7}
{1, 2, 4, 5, 6, 7}
{1, 2, 4, 5, 6, 7, 8, 9}
{1, 2, 'B', 4, 5, 6, 7, 8, 9, 'C', 'A'}
{1, 2, 'B', 4, 5, 6, 7, 8, 9, 10, 'C', 'A'}
{1, 2, 4, 5, 6, 7, 8, 9, 10, 'C', 'A'}
{1, 2, 4, 5, 6, 7, 8, 9, 10, 'A'}
{2, 4, 5, 6, 7, 8, 9, 10, 'A'}
2 4 5 6 7 8 9 10 A
set()
```

Set methods

- update() Updates a set with the union of itself and others
- union() Returns the union of sets in a new set
- difference() Returns the difference of two or more sets as a new set
- difference_update() Removes all elements of another set from this set
- discard() Removes an element from set if it is a member. (Do nothing if the element is not in set)
- intersection() Returns the intersection of two sets as a new set
- intersection_update() Updates the set with the intersection of itself and another
- isdisjoint() Returns True if two sets have a null intersection
- issubset() Returns True if another set contains this set
- issuperset() Returns True if this set contains another set
- symmetric_difference() Returns the symmetric difference of two sets as a new set

IO

```
a=1
b=a+1
c=a
print(a is b-1)
print(a is c)
```

```
True
True
```

```
PI=3.14159265359
print("0 Value of Pi is:",PI)
print("1 Value of Pi is: %d"%(PI))
print("2 Value of Pi is: %2d"%(PI))
print("3 Value of Pi is: %3d"%(PI))
print("4 Value of Pi is: %f"%(PI))
print("5 Value of Pi is: %.2f"%(PI))
print("6 Value of Pi is: %.3f"%(PI))
print("7 Value of pi is {0:.1f}".format(PI))
```

```
0 Value of Pi is: 3.14159265359
1 Value of Pi is: 3
2 Value of Pi is:  3
3 Value of Pi is:   3
4 Value of Pi is: 3.141593
5 Value of Pi is: 3.14
6 Value of Pi is: 3.142
7 Value of pi is 3.1
```

```
INPUT_VAR=input("Enterv a number:")
print("The entered number is:",INPUT_VAR)
print("The entered number is: {}".format(INPUT_VAR))# Replace {} by variable value in string
print("Type of input is: ", type(INPUT_VAR))
INPUT_VAR=int(INPUT_VAR)
print("Type of input is: ", type(INPUT_VAR))
```

```
Enterv a number:1
The entered number is: 1
The entered number is: 1
Type of input is:  <class 'str'>
Type of input is:  <class 'int'>
```

```
INPUT_VAR=int(INPUT_VAR) # convert string/float to integer
FtoI=float(INT_VAR)
ItoF=int(FLOAT_VAR)
```

```
STR=str(ItoF)
print("FtoI is ", FtoI, " and ItoF is ", ItoF, ' and STRING is', STR)
```

```
FtoI is  1.0  and ItoF is  1  and STRING is 1
```

```
import time
for i in reversed(range(5)):
  print(i,end="-")
  time.sleep(1)#skip 1 second
print("Start")
```

```
4-3-2-1-0-Start
```

Conditional and Loop

```
if (INPUT_VAR >10):
    print("The number is greater then 10")
elif (INPUT_VAR < 10):
    print("The number is less then 10")
else:
    print("The number is 10")
```

```
The number is less then 10
```

```
if __name__ == '__main__':
    print("Entry point of code is original file")
else:
    print("Entry point of code is from IMPORT FILE.py")
```

```
Entry point of code is original file
```

```
for i in range(5):
    print(i,end=" ")
```

```
0 1 2 3 4
```

```
for i in range(5,10):
    print(i)
```

```
5
6
7
8
9
```

```
for i in LIST_VAR:
    print (i,type(i))
```

```
1 <class 'int'>
2 <class 'int'>
4 <class 'int'>
4 <class 'int'>
2 <class 'int'>
1 <class 'int'>
5 <class 'int'>
6 <class 'int'>
5 <class 'int'>
```

```
STRING_VAR="My name is Mehul"
for i in STRING_VAR:
  print(i)
```

```
M
y

n
a
m
e

i
s

M
e
h
u
l
```

```
if 'M' in STRING_VAR:
  print("M is in the string")
```

```
M is in the string
```

```
STRING_VAR=STRING_VAR.split(" ")
for i in STRING_VAR:
  print(i)
```

```
My
name
is
Mehul
```

```
if 'Mehul' in STRING_VAR:
  print("Mehul is in the string")
```

```
Mehul is in the string
```

```
Count=5
while (Count>0):
  print(Count)
  Count-=1
else:
  print("Process Done")
```

```
5
4
3
2
1
Process Done
```

```
for i in range(5):
  if i==2: continue
  if i==4: break
  print(i)
```

```
0
1
3
```

```
for i in range(5):
  pass # to do nothing in loop or to return
```

```
LIST_VAR1=['A','B','C']
LIST_VAR2=[1,2,3]
for X,Y in zip(LIST_VAR1,LIST_VAR2):# Two List in one loop
  print(X,":",Y)
```

```
A : 1
B : 2
C : 3
```

```
DICT_VAR={'Mehul':'CE','Virag':'SE','Khevin':'ISE'}
for Key,Value in DICT_VAR.items():
  print(Key,":",Value)
```

```
Mehul : CE
Virag : SE
Khevin : ISE
```

```
# Statements in multiple line can be defined using
# square brackets [], semi-colon (;)
A=1;B=2
Y=[A,
   B]
print(A,B,Y)
```

```
1 2 [1, 2]
```

```
A,B,C=input("Enter 3 values sepatared by space").split(' ',3)# split(separator, max seperator)
print(A,B,C)
```

```
Enter 3 values sepatared by space1 2 3
1 2 3
```

```
x,y= [int(x) for x in input("Enter multiple value: ").split()]
print(x,y)
```

```
Enter multiple value: 1 2
1 2
```

```
print("Value of A and B are",A,"and",B,sep=" ",end="\n")
```

```
Value of A and B are 1 and 2
```

```
#Smart methods
A=5
B= A if A>5 else 0
print(A,B)
```

```
5 0
```

Libraries and Keywords

```
import math as MATHEMATICS
print(MATHEMATICS.fabs(ItoF))
print(MATHEMATICS.factorial(5))
```

```
1.0
120
```

```
from math import factorial
print(factorial(5))
```

```
120
```

```
import operator
print(operator.add(1,2))
```

```
print(operator.gt(1,2))
print(operator.pow(2,3))
```

```
3
False
8
```

```
#Keywords are predefined words and can not be used as variables
"""
Logical: and or not
Loop: for in break continue while
Conditional: if elif else
Input / Output : print input
Data types: type True False global None
Object oriented programming: def class import return as
Other: assert del except finally from is lambda (???)
     : nonlocal pass raise try with yield (???)
"""
#yeild : used to replace return in function and return generator object
#       : next(function_name) can be used to get value inside Generator
#       : Yeild returns the value each time and continue function execution when returns only executed once in the functio
#with : used for exception handling and error management
#assert: is used to generate assertion error if condition is false
#      assert x == 0, "x should be non zero for division"
#del : delete any variable,class object or element in list
#    : del X, del X[0], del Car
#try and except : used to handle exception error handling
#     if error is generated in try the execute except block
#    :Some of the common Exception Errors
#     IOError: if the file can't be opened
#     KeyboardInterrupt: when an unrequired key is pressed by the user
#     ValueError: when built-in function receives a wrong argument
#     EOFError: if End-Of-File is hit without reading any data
#     ImportError: if it is unable to find the module
#raise : stop program execution and raise error
#     raise Exception("Display message")
#     raise TypeError("Display message")
#pass : Do nothing and used to add  atleast one line in if, class, def
#     if X>B:
#         pass
#nonlocal: nonlocal variables are used in nested functions
#       : its is nither global nor local
#       : nonlocal VAR1 does not define new local variable in child funstion but use the already defined variable in paren
#       : def Parent():
#       :     VAR1=0
#       :     def child():
#       :         nonlocal VAR1;
#       :         VAR1 = VAR1 * VAR1
#None : is NULL, X= None so X does not have any value
#lambda: The function without the name with multiple arguments and one expression
#      : Function_name = lambda arguments: expression --> how to define
#      : Function_name(argunets) --> how to call
#is : used to check two variables represent same obejct or not, not for same values or not
#     X is Y, CAR is VEHICLE returns True or False
# from and import: Import only specific section from the module
#      : from datetime import time
#
#
#
#
#
#

```

```
'\nLogical: and or not\nLoop: for in break continue while\nConditional: if elif else \nInput / Output : print input\nDa
ta types: type True False global None\nObject oriented programming: def class import return as\nOther: assert del excep
t finally from is lambda (???)\n     : nonlocal pass raise try with yield (???)\n'
```

File Handling

```
# Create a file if does not exist
File1=open('Hello.txt', 'x')
File1.close()

#read/open a file without with
#Read mode, Can not edit
File1_r=open("Hello.txt","r")
File1_r.close()

#Append mode
File1_a=open("Hello.txt","a")
File1_a.write("Opened in append mode")
File1_a.close()

# write mode, clear all content and start writing
File1_w=open("Hello.txt","w")
File1_w.write("Writen in Write mode")
File1_w.close()

# with try and finally
File1_w=open("Hello.txt","w")
try:
    File1_w.write("With try and finally")
finally:
    File1_w.close()
```

```
#file open using with
with open("Hello.txt","w") as File2_w:
    File2_w.write("Uisng with")
# no need to close the file - Difference
```

```
# Memory location of object File2_w
print(id(File2_w))
```

```
140425814359856
```

```
X=1
Y=1
#Y=0
try:
    print(X/Y)
except ZeroDivisionError:
    print("Value of Y is Zero, so division is not possible")
else:
    print("Division is successfull")
finally:
    print("Process is over")
```

```
1.0
Division is successfull
Process is over
```

Functions and Class

class: Attributes State, Attributes: Variables Behaviours: Methods, Functions Identity: Name of objects which consist class property

```
#class
class Car:
    Speed=0 # Attributes, Data
    def Acc(self): # Method
```

```
        self.Speed+=5
    def Break(self): # Method
        self.Speed-=5
SUV=Car()
print(SUV.Speed)
SUV.Acc()
print(SUV.Speed)
SUV.Acc()
print(SUV.Speed)
SUV.Break()
print(SUV.Speed)
```

```
0
5
10
5
```

```
def ADDITION (NUM_1=5, NUM_2=10):
  """Function to add two numbers"""
  Ans= NUM_1 + NUM_2
  return Ans
print(ADDITION())
print(ADDITION(ItoF,FtoI))
print(ADDITION.__doc__)# Print function info types in """ """
```

```
15
2.0
Function to add two numbers
```

```
X=10
Y=6
def SUB(A,B):
    return A-B
SUBTRACT = lambda A,B : A-B
print(SUB(X,Y))
print(SUBTRACT(X,Y))

# Function pass (Function can be considered a sobject)
new_sub=SUB
print(new_sub(5,5))

# Function can return function and set a once and B multiple time
def func_ret_func(a):
  def R_SUB(b):
    return a-b
  return R_SUB
new_func=func_ret_func(10)
print(new_func(5))

#Closure: After delete of func_ret_func, new_func is working
del func_ret_func
print(new_func(1))
```

```
4
4
0
5
9
```

```
#Decorators: used to change behaviour of already defined functions / class to perforrm Max - min always
def SUBRTACT(A,B):
  return A-B

print(SUBTRACT(5,1))
print(SUBTRACT(1,5))
```

```
4
-4
```

```
def FACTORIAL(NUM):
  Ans=1
  while NUM>1:
    Ans=Ans*NUM
    NUM-=1
  return Ans # Outside of loop, return only once

FACTORIAL(5)
```

```
120
```

```
def FACTORIAL(NUM):
  Ans=1
  while NUM>1:
    Ans=Ans*NUM
    NUM-=1
    yield Ans # inside the loop, returns multiple values before end of function execution

for i in FACTORIAL(5):
  print(i)
```

```
5
20
60
120
```

```
# global variable, Variable in global space
a=5
b=55

def add():
    b=10 # Variable in local space
    a=100
    def addition():
        nonlocal b # Use the same b as defined in main function inside the sub function
        global a
        c = a + b
        print(c) # Variable in sub-local space
    addition()

add()
```

```
15
```

```
# *args (Non-Keyword Arguments) & **kwargs (Keyword Arguments)
def PRINT(*MEM_LOC):# pass only one item, List/Tuple
  for i in MEM_LOC:
    print(i,end=" ")
PRINT(*LIST_VAR)
print("\n")

def PRINT_(**MEM_LOC):# pass arguments and values in pair for Dict, Map
  for Key,Value in MEM_LOC.items():
    print(Key,':',Value)
PRINT_(Name='Mehul',Surname='Kantaria')
PRINT_(**DICT_VAR)
```

```
9 8 7 6 5 4 2 2 1

Name : Mehul
Surname : Kantaria
Mehul : CE
Virag : SE
Khevin : ISE
```

```
def Function_ref(X):
  print("Old:",X)# Pass by refrence
  X=5 # New Location for X is assigned which is different and Local
  print("New:",X)

X=2
Function_ref(X)
print("Outside",X)
```

```
Old: 2
New: 5
Outside 2
```

Binary Operators

```
print(any([True,False,False]))# Any True??
print(any([False,False,False]))
print(all([True,True,True]))# All True??
print(all([True,True,False]))
```

```
True
False
True
False
```

```
#and Return the first false value. If not found return last
print(5 and 0)
print(0 and 5)
print(0 and 5 and 10 and 15)
print(15 and 10 and 5)
print(15 and 0 and 5)
print(15 and 10 and 50)
```

```
0
0
0
5
0
50
```

```
#or Return the first True value. If not found return first
print(5 or 0)
print(0 or 5)
print(0 or 5 or 10 or 15)
print(15 or 0 or 5)
print(0 or 0 or 5)
print(15 or 0 or 0)
```

```
5
5
5
15
5
15
```

```
print(not 15)
print(not 0)
print(not True)
```

```
False
True
False
```

```
1
```

```
print(' ' is ' ') # immutable so both ' ' have same memory location
print({} is {}) # Dictionary is mutable so both {} assigned different memory address
```

```
True
False
```

Shortcuts

```
EVEN_LIST= [x for x in range(0, 11) if x % 2 == 0]
EVEN_LIST
```

```
[0, 2, 4, 6, 8, 10]
```

```
a,b,c=1,5,9
print(a<b<c)
```

```
True
```

```
i=0
while (i<5): print(i);i+=1
```

```
0
1
2
3
4
```

OOP

```
# Constructors: Constructors are executed when class is defined. (Automatic process):
# Destructors: Destructors are called when an object gets destroyed
# Self is there in all constructors and methods
class Person():#or class Person:
  name="" # Class Variable print(Person.name) can be diffent then MRorMISS.name

  # Default constructor does not have any argiment (name), only self
  def __init__(self):
      self.language = 'Python' # Instance Variable

  # init method or constructor
  def __init__(self, n):
      self.name = n
      print("Constructor called")

  # Sample Method
  def say_hi(self):
      print('Hello, my name is', self.name)

  # Calling destructor at the end
  def __del__(self):
      print("Destructor called")

MRorMISS = Person('Mehul')
MRorMISS.say_hi()
print("EOP")
del MRorMISS
```

```
Constructor called
Hello, my name is Mehul
EOP
Destructor called
```

```
#Inheritance : Destructors are called when an object gets destroyed
#Multiple Inheritance: class INHERI(Person1, Person2):
class INHERI(Person):
  def IN_METHOD(self):
    print("How are you",self.name,"?")

new_MrorMiss=INHERI('GOOGLE')

new_MrorMiss.say_hi()

new_MrorMiss.IN_METHOD()
```

```
Constructor called
Hello, my name is GOOGLE
How are you GOOGLE ?
```

```
# Encapsulation puts restrictions on accessing variables and methods directly and can prevent the accidental modification
# To prevent accidental change, an object’s variable can only be changed by an object’s method
# Those types of variables are known as private variables.
# Example: Class
# self.__variable is private variable, which can not be used by derived class
```

```
# Polymorphism : means the same function name (but different signatures) being used for different types.
# Polymorphism in class meand some methods are in parent class but modified in derived class.

class Bird:
  def intro(self):
    print("There are many types of birds.")

  def flight(self):
    print("Most of the birds can fly but some cannot.")

class sparrow(Bird):
  def flight(self):# Modified
    print("Sparrows can fly.")

class ostrich(Bird):
  def flight(self):
    print("Ostriches cannot fly.")
```

```
# Class vs Ststic method
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # classmethod has cls as first argument
    def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)

    @staticmethod
    def isAdult(age):
        return age > 18
```

Exception handling and errors

```
a = 5
b = 0 #0 or 5
try:
  C=a//b
except: # execute if error
  print ("Divide by 0 error")
```

```
else: # execute if no exception
  print("Division is ",C)
finally: # Always executed
  print("END")
#  raise NameError("BHOOOOOOOOT") # To generate error forcefully
```

```
Divide by 0 error
END
```

Different types of errors NZEC (non zero exit code) as the name suggests occurs when your code is failed to return 0 OverflowError ZeroDivisionError FloatingPointError BufferError MemoryError AssertionError AttributeError EOFError

```

```

Regular Expression

```
import re
STRING="MY NAME IS MEHUL"
print(re.search(r"MEHUL",STRING)) # r for raw
print(re.search(r"MEHUL",STRING).start())
print(re.search(r"MEHUL",STRING).end())
```

```
<re.Match object; span=(11, 16), match='MEHUL'>
11
16
```

```
import re
from re import split

STRING="MY_NAME_ISSS$MEHUL^11*ABCD#22"
print("1.",re.search(r".",STRING)) #. is meta character, so wiill not work without \
print("2.",re.search(r"\.",STRING)) #consider . as regular

print("\n","3.",re.search(r"[A-C]",STRING)) #[A-C]=ABC
print("4.",re.search(r"[^A-Z]",STRING)) #[A-Z]=![A-Z] NOT

print("\n","5.",re.search(r"^MY",STRING)) # String start with MY
print("6.",re.search(r"^ABCD",STRING)) #

print("\n","7.",re.search(r"MY$",STRING)) #
print("8.",re.search(r"22$",STRING)) #String END  with MY

print("\n","9.",re.search(r"M.H.L",STRING)) # Find word with M_H_L pattern

print("\n","10.",re.search(r"ME|AB",STRING)) # ME or AB

print("\n","11.",re.search(r"M?_",STRING)) # M?_ or M_

print("\n","12.",re.search(r"IS*",STRING)) # Multiple S, SS, SSS, SSSS *-Zero or more, +-One or more occurance

print("\n","13.",re.findall(r"\d+",STRING)) # find digits

S=re.compile('[A-C]')
print("\n","14.",S.findall(STRING)) # find A,B,C all

print("\n","14.",split('\W+',STRING))# Split by non alphanumeric character
print("15.",split('\d+',STRING))# Split by numbers
print("16.",split('[A-B]+',STRING))# Split by A,B,C

regex = "([a-zA-Z]+) (\d+)"
print("\n","17.",regex)
```

```
1. <re.Match object; span=(0, 1), match='M'>
```

```
2. None

 3. <re.Match object; span=(4, 5), match='A'>
4. <re.Match object; span=(2, 3), match='_'>

 5. <re.Match object; span=(0, 2), match='MY'>
6. None

 7. None
8. <re.Match object; span=(27, 29), match='22'>

 9. <re.Match object; span=(13, 18), match='MEHUL'>

 10. <re.Match object; span=(5, 7), match='ME'>

 11. <re.Match object; span=(2, 3), match='_'>

 12. <re.Match object; span=(8, 12), match='ISSS'>

 13. ['11', '22']

 14. ['A', 'A', 'B', 'C']

 14. ['MY_NAME_ISSS', 'MEHUL', '11', 'ABCD', '22']
15. ['MY_NAME_ISSS$MEHUL^', '*ABCD#', '']
16. ['MY_N', 'ME_ISSS$MEHUL^11*', 'CD#22']

 17. ([a-zA-Z]+) (\d+)
```

Collections

```
# Heap data structure is mainly used to represent a priority queue
# The heap[0] element also returns the smallest element each time

import heapq

li = [5, 7, 9, 1, 3]

heapq.heapify(li)
print (li)

heapq.heappush(li,2)
print (li)

heapq.heappop(li)
print (li)

heapq.heappushpop(li,4)
print (li)

heapq.heapreplace(li,8) # pop + replace poped item
print (li)

print (heapq.nlargest(2,li))
print (heapq.nsmallest(2,li))
```

```
[1, 3, 9, 7, 5]
[1, 3, 2, 7, 5, 9]
[2, 3, 9, 7, 5]
[3, 4, 9, 7, 5]
[4, 5, 9, 7, 8]
[9, 8]
[4, 5]
```

```
# Counter is a sub-class of the dictionary
# It is used to keep the count of the elements in an iterable in the form of an unordered dictionary
# key represents the element in the iterable
# value represents the count of that element
```

```
from collections import Counter
C=Counter(['B','B','A','B','C','A','B','B','A','C'])
print(C)

C.update(['B',"B"])
print(C)

C.subtract(['B',"B"])
print(C)

```

```
Counter({'B': 5, 'A': 3, 'C': 2})
Counter({'B': 7, 'A': 3, 'C': 2})
Counter({'B': 5, 'A': 3, 'C': 2})
```

```
# OrderedDict is also a sub-class of dictionary
# it remembers the order in which the keys were inserted
from collections import OrderedDict

od = OrderedDict()
od['a'] = 1
od['d'] = 4
od['c'] = 3
od['b'] = 2

for key, value in od.items():
    print(key, value)

od.pop('a')
print("updated before")
for key, value in od.items():
    print(key, value)

od['a']=10
print("updated after")
for key, value in od.items():
    print(key, value)
```

```
a 1
d 4
c 3
b 2
updated before
d 4
c 3
b 2
updated after
d 4
c 3
b 2
a 10
```

```
# DefaultDict is also a sub-class to dictionary
# It is used to provide some default values for the key that does not exist and never raises a KeyError
from collections import defaultdict

# Defining the dict
dd = defaultdict(int)
print(dd['a'])

dd = defaultdict(float)
print(dd['a'])

dd = defaultdict(str)
print(dd['a'])

dd = defaultdict(lambda: "Not Present")
print(dd['a'])
```

```
0
0.0

Not Present
```

```
# ChainMap encapsulates many dictionaries into a single unit and returns a list of dictionaries.
from collections import ChainMap

d1 = {'a': 1, 'b': 2}
d2 = {'c': 3, 'd': 4}
d3 = {'e': 5, 'f': 6}

c = ChainMap(d1, d2)
print(c)

nc=c.new_child(d3)
print(nc)
```

```
ChainMap({'a': 1, 'b': 2}, {'c': 3, 'd': 4})
ChainMap({'e': 5, 'f': 6}, {'a': 1, 'b': 2}, {'c': 3, 'd': 4})
```

```
#NamedTuple returns a tuple object with names for each position which the ordinary tuples lack
from collections import namedtuple

Student = namedtuple('Student',['name','age','DOB'])

# Adding values
S = Student('MEHUL','35','11223333')

# Access using index and key
print (S[0])
print (S.name)
```

```
MEHUL
MEHUL
```

```
# Deque (Doubly Ended Queue) is the optimized list for quicker append and pop operations from both sides of the container
from collections import deque

dq = deque(['MEHUL','UTSAV'])
print(dq)

dq.append('SAJAL')
print(dq)

dq.appendleft('AMAN')
print(dq)

dq.pop()
print(dq)

dq.popleft()
print(dq)

print(dq.index('UTSAV'))

dq.insert(0,'MEHUL')
print(dq)

print(dq.count('MEHUL'))

dq.extend(['ACHAL','JAY'])
print(dq)

dq.extendleft(['ACHAL','JAY'])
print(dq)
```

```
31
32 dq.rotate(1)
33 print(dq)
34
35 dq.rotate(-3)
36 print(dq)
37
38 dq.reverse()
39 print(dq)
```

```
deque(['MEHUL', 'UTSAV'])
deque(['MEHUL', 'UTSAV', 'SAJAL'])
deque(['AMAN', 'MEHUL', 'UTSAV', 'SAJAL'])
deque(['AMAN', 'MEHUL', 'UTSAV'])
deque(['MEHUL', 'UTSAV'])
1
deque(['MEHUL', 'MEHUL', 'UTSAV'])
2
deque(['MEHUL', 'MEHUL', 'UTSAV', 'ACHAL', 'JAY'])
deque(['JAY', 'ACHAL', 'MEHUL', 'MEHUL', 'UTSAV', 'ACHAL', 'JAY'])
deque(['JAY', 'JAY', 'ACHAL', 'MEHUL', 'MEHUL', 'UTSAV', 'ACHAL'])
deque(['MEHUL', 'MEHUL', 'UTSAV', 'ACHAL', 'JAY', 'JAY', 'ACHAL'])
deque(['ACHAL', 'JAY', 'JAY', 'ACHAL', 'UTSAV', 'MEHUL', 'MEHUL'])
```

```
1 # UserDict, UserList and UserString are container
2 # It is used when someone wants to create their own Dict, List and strings with some modified or additional functionality
3 from collections import UserDict, UserList, UserString
4
5 class MyDict(UserDict):
6     def __del__(self):
7         raise RuntimeError("Deletion not allowed")
8     def pop(self, s = None):
9         raise RuntimeError("Deletion not allowed")
10
11 d = MyDict({'a':1, 'b': 2})
12 d['c']=3
13 print(d)
14 d.pop(1)
```

```
{'a': 1, 'b': 2, 'c': 3}
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-44-f89f1994ee64> in <module>
     12 d['c']=3
     13 print(d)
---> 14 d.pop(1)

<ipython-input-44-f89f1994ee64> in pop(self, s)
      7         raise RuntimeError("Deletion not allowed")
      8     def pop(self, s = None):
----> 9         raise RuntimeError("Deletion not allowed")
     10
     11 d = MyDict({'a':1, 'b': 2})

RuntimeError: Deletion not allowed

SEARCH STACK OVERFLOW
```

Advance

```
1 # working directory
2 import os
3 print(os.name)
4 CWD=os.getcwd()
5 print(CWD)
6 os.chdir('/content/sample_data')
```

```
NCWD=os.getcwd()
print(NCWD)
```

```
posix
/content/sample_data
/content/sample_data
```

```
# Operation with directory
import os

print(os.listdir('/content/'))
try:
  os.mkdir('/content/New')
except:
  print('Directory already exist')
print(os.listdir('/content/'))

try:
  os.rmdir('/content/New')# Directory
  os.remove('/content/sample_data/README.md')# File
except:
  print('Directory not available')
print(os.listdir('/content/'))

try:
  os.rename('/content/sample_data/anscombe.json','/content/sample_data/new_anscombe.json')
except:
  print('Directory not available')

print(os.listdir('/content/sample_data'))
print(os.path.exists('/content/sample_data'))# File available or not

print(os.path.getsize('/content/sample_data/mnist_test.csv'))
```

```
['.config', '.ipynb_checkpoints', 'sample_data']
['.config', 'New', '.ipynb_checkpoints', 'sample_data']
Directory not available
['.config', '.ipynb_checkpoints', 'sample_data']
Directory not available
['newanscombe.json', 'california_housing_train.csv', 'mnist_test.csv', 'mnist_train_small.csv', 'california_housing_test
True
18289443
```

```
#functional programming
def Functional_func(n):
  for i in range (0,n):
    print(i,end=" ")


#Recursion programming
def Recurtion_func(n):
  if n==1:
    return 1
  else:
    return n * Recurtion_func(n-1)

Functional_func(5)
print("")
print(Recurtion_func(5))
```

```
0 1 2 3 4
120
```

```
# Higher order functions
# map apply function on Data
```

```
def MUL_by_2(L):
  return L*2

Li=[1,2,3,4,5]
Ans=map(MUL_by_2,Li)
print("Answr is:",end="")
for i in Ans:
  print(i,end=" ")

# Filter elements from reference
def Filter_fun(value):
    reference = [1,2,3]
    if (value in reference):
        return True
    else:
        return False

sequence = [1,2,3,4,5]
filtered = filter(fun, sequence)
print("\nFiltered numbers are:",end="")
for s in filtered:
    print(s,end=" ")
```

```
Answr is:2 4 6 8 10
Filtered numbers are:1 2 3
```

```
# Abstract class (Used for multiple parallel implementation/team)
from abc import ABC, abstractmethod
class Polygon(ABC):
  def no_of_sides(self):
    print("There are multiple sides")

class Triangle(Polygon):
  def no_of_sides(self):
    print("3 sides")

P=Polygon()
P.no_of_sides()
T=Triangle()
T.no_of_sides()
```

```
There are multiple sides
3 sides
```