```
#1st day
#Program to display "Employee-Details" using {} and format()
#Sample81.py

empno=1122;
ename="Sai";
job="Manager"
sal=6500;
deptno=10;

print("****Employee-Details****");
print()
print("{} with an Employee-No:{} is working as:{} with a Salary of:{} in Dept-No:{}".format(ename,empno,job,sal,deptno));


#Program to display proper-msgs with data using {} and format()
#Sample8.py

a=11
b=3
print("Arithmetic-Operations");
print("A =",a)
print("B =",b)
print()
print("Sum of {} and {} == {}".format(a,b,a+b))
print("Sub of {} and {} == {}".format(a,b,a-b))
print("Prod of {} and {} == {}".format(a,b,a*b))
print("Div of {} and {} == {}".format(a,b,a/b))
print("MOD of {} and {} == {}".format(a,b,a%b))


# 2nd day
##Program (Demo3.py)
##Program to demo Quotations with strings

name='Sai'
city="New Hyd"
addr='''
#101, St.No.9,
HimayathNagar,
Hyd-29
TS(India)
'''

print("Quotations in Python");
print(name)
print(city)
print(addr)


##Program (Demo1.py)
#Program to demo line-indentations

if False: #True/False
    print("True-Part");
    print("True-Block-of-Code");
else:
   print("False-Part");
   print("False-Block-of-Code");



##Program (Demo31.py)
#Program to demo strings & comments in quotes
'''
 Program : To display Welcome Msg on Console
 Author : Sai sir
 Company : TCS
 Location : New Hyd
 Country : India
 Technology : Python
 Platform : Windows 10 Pro
 Date of Creation : 26-JULY-2022
 Date of Modification : 28-JULY-2022
'''

print("Hello Students");
print("Welcome to Python");
print("***ALL THE BEST");

print()
a=11
b=3
print(a+b);     #addition
print(a-b);     #subtraction
print(a*b);     #Multiplication


##Program (Demo2.py)
#Prog to demo multi-line stmts using \(continuation-char)

a=10;
b=20;
c=30;
sum=a+b+c;
print(sum)

sum =  a+\
  b+\
  c;
print(sum);


#Sp-Case without \ continuation-char (Collections)
months = ['Jan', 'Feb', 'Mar',
    'Apr', 'May', 'Jun',
      'Jul', 'Aug','Sep',
                        'Oct','Nov','Dec'];
print(months);


##Program (Demo.py)
#Program with duplicate-variables(Identifiers)

a=10;
print(a)

a=100;
print(a)
```

```
a=10.5;
print(a)

a="Hello";
print(a);



#3rd day

##Program (DatatypeEx1.py)
#Program to demo int-dtype with number-system-conversions

#int-dtype
a=10
print(a)
print(type(a))
print(id(a))

a=-10
print(a)
print(type(a))
print(id(a))

a=0
print(a)
print(type(a))
print(id(a))


#number-system
print()
a=10
print(a)
a=0b1010
print(a)
a=0o123
print(a)
a=0x1ae
print(a)

print()
#number-system-conversions
a=10
print(a)
print(bin(a))

a=83
print(a)
print(oct(a))

a=430
print(a)
print(hex(a))




##Program (DatatypeEx2.py)
#Program to demo float,complex,bool and str datatypes

#float-dtype
a=1.5;
print(a)
print(type(a))

a=1.5E3;
print(a)
print(type(a))

a=1.5E-3;
print(a)
print(type(a))

print()
#complex-dtype
a = 5+6j
print(a)
print(type(a))

a = 5+6.5j
print(a)
print(type(a))

a = 5.5+6.5j
print(a)
print(type(a))


print()
#bool-dtype
a = True
print(a)
print(type(a))

a = False
print(a)
print(type(a))


print()
#str-dtype
a = 'Sai'
print(a)
print(type(a))

a = "Welcome to Python"
print(a)
print(type(a))

a = '''
#101,Str.No.9,
Hyd(TS)
India
'''
print(a)
print(type(a))
```

```
##Program (Demo4.py)
#Program to demo variable assignment with Student-Data

rollno=1001;  #integer-value
name="Sai";   #string-value
height=6.0;   #real-value (floating-value)

print("Student-Details");
print("Roll No :",rollno);
print("Name :",name);
print("Height :",height);


#Program (Demo51.py)
#Program to demo python-variables with type(), id() and print()

a=10
print(a)
print(type(a))
print(id(a))

print()
a=10.5
print(a)
print(type(a))
print(id(a))

print()
a="Sai"
print(a)
print(type(a))
print(id(a))

print()
a=True
print(a)
print(type(a))
print(id(a))

print()
a=5+6j
print(a)
print(type(a))
print(id(a))


#4th day
##Program (DelRefVar.py)
#Program to demo Reference Deletion

a=10
b=20
print(a,b)
print(id(a),id(b))

del a
del b
#print(a,b)     #NameError
c=10
d=20
print(c,d)
print(id(c),id(d))

del c,d;
#print(c,d)     #NameError


#Program (TypeCastingEx1.py)
#Program to demo Typecasting with python-functions

#int()
a=int(10)
a=int(10.8)     #complete deci-part is deleted
a=int("100")
#a=int("hi")       #ValueError
a=int(True)
#a=int(5+6j)    #complex is not-convertible
print(a)
print(type(a))


print()
#float()
a=float(10)     #deci-point(.) is added
a=float(10.8)
a=float("100")
#a=float("hi")       #ValueError
a=float(True)
#a=float(5+6j)    #complex is not-convertible
print(a)
print(type(a))


print()
#complex()
a=complex(10)
a=complex(10.8)
a=complex("100")
#a=complex("hi")       #ValueError
a=complex(True)
a=complex(5+6j)
a=complex("15+6j")
a=complex(5,6)
a=complex(5.5,6.6)
print(a)
print(type(a))


print()
#bool()
a=bool(1)
a=bool(0)
a=bool(10)      #non-zero is True & any zero is False
a=bool("hi")    #**sp-case any-string is T & empty-str is F
a=bool("False")
a=bool("")      #w.o any chars is empty-string
print(a)
print(type(a))
```

```
print()
#str()
a=str(10)
a=str(10.5)
a=str("hi")
a=str(True)
a=str(5+6j)
print(a)
print(type(a))
#sp-case
print(10+10)
#print(str(10)+10)       #"10"+10 (str+int) Error
print(str(10)+str(10))      #"10"+"10" ---> 1010



##Program (PythonVarMemory.py)
#Program to demo Python Variable Memory allocation & de-allocation

#Same-var diff-values(latest-value)
a=10
print(a)
a=10.5
print(a)
a="Hello"
print(a)

print()
#Diff-vars Same-value
a=10
b=10
print(a,b)
print(id(a),id(b))

print()
#Diff-vars diff-values
a=10
b=20
print(a,b)
print(id(a),id(b))



##Program (ImmutableValues.py)
#Program to demo Immutable Values

a=10
print(a)
print(id(a))
a=a+10
print(a)
print(id(a))

print()
b=10        #re-used
print(b)
print(id(b))


#5th day
##Program (EscSeqChars.py)
#Program to work with Escape-Sequence-Characters

print("Sai\nRam\nKumar");
print("Sai\tRam\tKumar\tAjay");
#tab-space-count is from begining of line

print()
print("Hello World\b\bks");
print("Jack\rB");

#sp-cases ',",\ inside string
print()
print("Hello\'World");
print("Hello\"World");
print("Hello\\World");



##Program (RangeDatatypeEx1.py)
#Program to work with range-coll-datatype

r1 = range(10);     #0 to 9
print(r1);
print(type(r1));
#access
print(r1[0],r1[-1])
print(r1[1],r1[-2])
print(r1[2],r1[-3])
print(r1[3],r1[-4])
print(r1[4],r1[-5])
print(r1[5],r1[-6])
print(r1[6],r1[-7])
print(r1[7],r1[-8])
print(r1[8],r1[-9])
print(r1[9],r1[-10])
print()
#using-loops-access
for i in r1:
 print(i);


print();
r1 = range(10,20);    #10 to 19
for i in r1:
 print(i);

print()
r1 = range(10,20,2);     #10,12,14,16,18 (Step-value=2)
for i in r1: print(i);

print()
r1 = range(10,20,3);
for i in r1: print(i);

print()
r = range(20,10,-2);
for i in r: print(i);
```

```
print()
r1 = range(1,20,-2);     #range not generated
for i in r1: print(i);
#1,20(forward-range) but -2(Backward-step)


print()
r = range(20,1,2);     #range not generated
for i in r: print(i);
#20,1(backward-range) but +2(Farward-step)


#range with indexes
print();
r = range(1,11);    #1 to 10 values
print(r[0]);
print(r[1]);
print(r[-1]);
print(r[-2]);
#print(r[20]); #IndexError

print()
#range-coll in immutable
#r[0]=100;  #TypeError (range values are immutable-obj)


##Program (ArithmeticOperators1.py)
##Program to demo Arithmetic-Operators using integers)

a=11;
b=3;
print(a+b);
print(a-b);
print(a*b);
print(a/b);  #Quot, till Rem is Zero(approx)
print(a%b);  #Rem
print(a//b); #Quot
print(a**b);


print()
#String-Additions(+) & Repititions(*)
print("Hello"+"World");
#print("Hello"+100);  #Error (str+int)
print("Hello"+"100");
#print(100+"Hello");  #Error (int+str)
print("100"+"Hello");

print()
####
print("Hello"*5);
print(2*"Hello");
#print(1.5*"Hello");  #Error(float*Str)
#print("Hello"*1.5);  #Error(str*Float)
print("Hello"*"Hello");     #Error(str*str)


##Program (RelationalOperators1.py)
#Program to demo Relational-Operators

a=11
b=3
print(a<b)
print(a>b)
print(a<=b)     #< or == (any one of them)
print(a>=b)     #> or ==
print(a==b)
print(a!=b)

print()
a=10
b=10
print(a==b)
print(a!=b)



##Program (NoneTypeEx1.py)
#Program to work with none-datatype

a=10;
print(a)
print(id(a))
print(type(a))

print()
a=None
print(a)
print(id(a))
print(type(a))



#6th day
##Program (AssignmentOperator1.py)
#Program to demo Assignment-Operator

a=10;
b=20;
sum=a+b;    #30
print(a,b,sum);

print()
####multi-var-assignment
a=b=c=10;   #multi-vars with same-value
x,y,z=11,22,33;     #multi-vars with diff-vals
print(a);
print(b);
print(c);
print(x);
print(y);
print(z);

print()
####sp-case
#Compound Assignments
a=11;
b=3;
a+=b; #a=a+b;
print(a);
```

```
print(b);

a=11;
b=3;
a-=b;
print(a);
print(b);

a=11;
b=3;
a*=b;
print(a);
print(b);

a=11;
b=3;
a/=b;
print(a);
print(b);

a=11;
b=3;
a%=b;
print(a);
print(b);

a=11;
b=3;
a//=b;
print(a);
print(b);

a=11;
b=3;
a**=b;
print(a);
print(b);


##Program (TernaryOperator1.py)
#(Program to demo Ternary-Operator)

a,b = 10,20;
result = "A is min" if (a<b) else "B is min";
print(result);

###
a,b=11,22;
result = "A&B are SAME" if (a==b) else "A&B are NOT-SAME";
print(result);


##Program (LogicalOperators1.py)
#Program to work with Logical Operators


a=10;
b=20;
#logical-and
print(a>5 and b>5) #T and T (True)
print(a>5 and b<5) #T and F (False)
print(a<5 and b>5) #F and T (False)
print(a<5 and b<5) #F and F (False)

print()
###
#logical-or
print(a>5 or b==20); #T or T (True)
print(a>5 or b!=20); #T or F (True)
print(a<5 or b==20); #F or T (True)
print(a<5 or b!=20); #F or F (False)

print()
###
#logical-not
print(not (a==b)); #not F(True)
print(not (a!=b)); #not T(False)

##Program (IdentityMembershipOper1.py)
##Program to work with Identity-Operator & Membership-Operator


#is, is not
a=10;
b=10;
print(a is b);
###
a=10;
b=20;
print(a is not b);
###
a=True;
b=False;
print(a is b);
print(a is not b);
###
a="Sai";
b="Sai";
print(id(a));
print(id(b));
print(a is b);
print(a is not b);



#in, not in
#using strings
ss = "Hello and Welcome";
print('H' in ss);
print('z' in ss);
print('x' not in ss);
print('and' in ss);
print('or' in ss);
print('HW' in ss);
print()
#using lists
a=[10,20,30,"Sai",6.0,True];
print(10 in a);
print(100 in a);
print("Sai" in a);
```

```
print("Ram" not in a);
print(True not in a);


##Program (BitwiseOperators1.py)
#Program to demo Bitwise-Operators

print(10&18);
print(10|18);
print(10^18);
print(~10);
print(10<<2);
print()


'''
a=10  1010(binary)
b=18  10010(binary)

Ex:-(applying bitwise operators on int-type values)
(10&18)
   128 64 32 16 8  4  2  1
 10  0   0  0  0  1  0  1  0
 18  0   0  0  1  0  0  1  0
 ---------------------------------
 &  0   0  0  0  0  0  1  0  (2)
 ---------------------------------

(10|18)
   128 64 32 16 8  4  2  1
 10  0   0  0  0  1  0  1  0
 18  0   0  0  1  0  0  1  0
 ---------------------------------
 |  0   0  0  1  1  0  1  0  (26)
 ---------------------------------


(10^18)
   128 64 32 16 8  4  2  1
 10  0   0  0  0  1  0  1  0
 18  0   0  0  1  0  0  1  0
 ---------------------------------
 ^  0   0  0  1  1  0  0  0  (24)
 ---------------------------------


~10
   128 64 32 16 8  4  2  1
 10  0   0  0  0  1  0  1  0
 ---------------------------------
 ~  1   1  1  1  0  1  0  1  (245)[-11]
 ---------------------------------

(10<<2)
   128 64 32 16 8  4  2  1
 10  0   0  0  0  1  0  1  0
 ---------------------------------
 <<  0 0 0   0  1  0  1  0  0  0
   0   0  1  0  1  0  0  0  (40)
 ---------------------------------
(Bits moving out-of-order ignore & empty bits replace with 0)

(18>>2)
   128 64 32 16 8  4  2  1
 18  0   0  0  1  0  0  1  0
 ---------------------------------
 >>      0   0  0  0  0  1  0  0  1  0
   0   0  0  0  0  1  0  0  (4)
 ---------------------------------

'''

#7th day
##Program (DynamicInput.py)
#Program to accept dynamic input from keyboard

#sum of 2-no's
a = input("Enter A-value = ");
b = input("Enter B-value = ");
a = int(a)
b = int(b)
sum=a+b;
print(sum)


print()
a = int(input("Enter A-value = "));
b = int(input("Enter B-value = "));
sum=a+b;
print(sum)

print()
print(int(input("Enter A-value = ")) + int(input("Enter B-value = ")))


#other-dtype-input-values
a = int(input("Enter int-value = "));
print(a)
print(type(a))

print()
a = float(input("Enter float-value = "));
print(a)
print(type(a))

print()
a = complex(input("Enter complex-value = "));
print(a)
print(type(a))

print()
a = bool(input("Enter bool-value = "));
print(a)
print(type(a))



##Program (CmdLineArgs.py)
#Program to demo Cmd-Line-Args
```

```
import sys;     #sys.py module-py-file
print(sys.argv); #all values are strings in list-of-values

print()
#using indexes
print(sys.argv[0]);
print(sys.argv[1]);
print(sys.argv[2]);
print(sys.argv[3]);

print()
print(type(sys.argv));
print("No.of Args : ",len(sys.argv));       #len() func

print()
#using loop
for x in sys.argv:  #[4-values]
 print(x);



##Assignment on Cmd-Line-Args
##WAP to accept to input-values as Cmd-Line-Args(11,3) & perform Arithmetic-Opers
a = int(sys.argv[1])
b = int(sys.argv[2])

##WAP to accept to input-values as Cmd-Line-Args(11.5,2.5) & perform Arithmetic-Opers
x = float(sys.argv[1])
y = float(sys.argv[2])



##Program (PrintSepEnd.py)
##Program to work with print() sep="" and end=""

#sep=""
a=11
b=3
sum=a+b;
print(a,b,sum)
print()
print(a,b,sum,sep=":")
print(a,b,sum,sep=",")
print(a,b,sum,sep="->")


print()
#end=""
a=11
b=3
sum=a+b;
print(a)
print(b)
print(sum)
print()
print(a,end="+")
print(b,end="=")
print(sum)



##Program (EvalFunction.py)
#Program to work with eval()


#evaluate-mathematical-expression
expr = input("Enter any mathematical-expression :: ");
result = eval(expr)
print(result)
print(type(result))


print()
#provide any type of data
a = eval(input("Enter int-value = "));
print(a)
print(type(a))

print()
a = eval(input("Enter float-value = "));
print(a)
print(type(a))

print()
a = eval(input("Enter complex-value = "));
print(a)
print(type(a))

print()
a = eval(input("Enter bool-value = "));
print(a)
print(type(a))


print()
#another-sp-case(derived-types) [],(),{} as input-coll
a = eval(input("Enter any-data = "));
print(a)
print(type(a))



'''
***(Assignments on eval() with diff-vars)
##WAP to accept different mathematical-formulas as input & calulate below operations
a) Area of Sqaure
b) Area of Rectangle
c) Area of Circle
d) Volume of Cube
e) Volume of Cuboid
f) Volume of Sphere
g) Surface-area of Cube
h) Surface-area of Cuboid
i) Surface-area of Sphere
j) Volume of Cone
k) Volume of Cylinder
l) Surface-area of Cone
m) Surface-area of Cylinder
'''
```

```
#8th day 8.02-08-22
"""
===>>> Control Structures or Flow Controls in Python:-
= Control means <<Condition>>: Ex:- (a<b)
= Structure means indented-block-of-code
Def:-
 Executing an indented-block-of-code, based on <<condition>> is known as Control-Structure
==Diagram==
Ex:-
 <<condition>>: (T/F)
  stmt1;
  stmt2;
  stmt3;
  .....


= In Python, we dont have blocks with {}
= Here we have indented-block-of-code
(single/multiple stmts with equal spaces from left)

***= Flow-control means the order in which statements are executed in program

=> CS(Flow-Controls) in python are classified into 3-types,
==Diagram==
 A) Conditional stmts (Branching-Stmts)
 (Executes stmts 0 or 1 time only)
  = if
  = multiple-if
  = if-else
  = nested-if
  = if-elif-else
 B) Iterative stmts (Looping-Stmts/Loops)
 (Executes stmts 0 or more time only)
  = for
  = while
  = nested-loops
 C) Transfer stmts (Jumps-Stmts)
  = break
  = continue
  = pass



***A) Conditional stmts:-
= They are also called as Branching stmts
= They execute indented-block-of-code 0 or 1 time only

i) if statement:-
Syntax:-
 if (condition):  #colon: is mandatory but not ()
  stmt1;
  stmt2;
  stmt3;
 .....

= Here if condition is True then "statements" are executed o.w not-executed for False (and comes to next-line)

"""
##Program (IfStmt1.py)
#Program to demo if-stmt


#if-stmt with single-stmt
name = input("Enter any name : ");
if name=="Sai":
 print("Hello :",name,"Good Morning");
print("Take care n Bye!!");



#with multiple-stmts
name = input("Enter any name : ");
if name=="Sai":
 print("Hello :",name,"Good Morning");
 print("Have a great day!!");
print("Take care n Bye!!");


#sp-case
#Header & Suite in single-line
#single-suite-stmts (CS/FS in single-line)
a = int(input("Enter any +ve integer :: "));
if (a>0): print("Given Number is +ve Number");
print("Take care n Bye!!");


"""
ii) Multiple-if Statements::-
= Here we use, multiple-if Statements one after the another
Ex:-
 if <<condition1>>:
  stmts;
  stmts;
 if <<condition2>>:
  stmts;
  stmts;
 if <<condition3>>:
  stmts;
  stmts;
 next-stmt;
 ......
 ......
= Here, whichever condition is True, its indented-statements are executed & comes to next-stmt o.w smts not-executed for False

"""
#Program (MultipleifStmt1.py)
#Program to demo Multiple-if stmts with +ve,-ve,0


num = int(input("Enter any integer-value :: "));
if num>0:
    print("Given Num is +ve");
if (num<0):
    print("Given Num is -ve");
if (num==0):
    print("Given Num is ZERO");
```

```
print("End of the Program");

"""
"Assignments"
#Program (MultipleifStmt2.py)
#(Program to demo Multiple-if stmts with Even,Odd,Neither of them)
Hint:-
 Even (num%2==0)
 Odd (num%2!==0)
 Neither Even Nor Odd (num==0)




iii) if-else statement:-
(Here we have 2nd option also)
Syntax:-
if condition:
 True-Stmts;
 .....
else:     #here : is mandatory
 False-Stmts;
 .....
.....
next-stmt
.....

=*** Here if condition is true then True-Stmts are executed,
if condition is False then False-Stmts are executed
(and comes to next-stmt)
"""

##Program (IfElseStmt1.py)
#Program to demo if-else stmt


#single-stmt
name = input("Enter any name : ");
if name=="Sai":
 print("Hello :",name);
else:
 print("Hello : New User!!");

print("End of Program!!");
'''

'''
#multi-stmts
name = input("Enter any name : ");
if name=="Sai":
 print("Hello :",name);
 print("Have a nice day!!");
else:
 print("Hello : New User!!");
 print("Register your Name...");

print("Take care n Bye!!");


#sp-case
#single-suite-stmt (Header & Suite in single-line)
num = int(input("Enter any Integer-Number : "));
if num==0: print("Given Number is 0");
else: print("Given Number is NON-Zero(+ve/-ve)");



# "Assignment"
#WAP to accept age of a person as input & display "Eligible for Voting or not"
##Program(IfElseStmt2.py)
#Program to demo if-else stmt (IfElseStmt1.py)


age = int(input("Enter your Age : "));
if (age>=18):
 print("Your are Eligible for Voting");
else:
 print("You are NOT-Eligible for Voting");

print("End of Program..!!");


"""
"Assignment"
#WAP to check for for eligibility for marraige
(maleage>=21 and femaleage>=21) [T and T  ---> T]
#WAP to accept age of candidate & print eligibility for employement
(personage >= 15 and personage <= 65) [T and T  ---> T]



iv)
=> Special-Case:-
==> Nested if's:-
= We can use "if-stmt" inside another if-stmt or another else-stmt as follows,
Ex:-
if condition:(T)
 stmts;
 if condition:(T)
  stmts;
  .....
else:
 stmts;
 if condition:
  stmts;
  .....

=** here if both conditions are True then its idented stmts are executed o.w else with if-stmts are executed

"""
##Program (NestedIf1.py)
#Program to check biggest of 3 numbers using nested-if)

print("Enter 3-diff values :");
a=int(input("Enter A value :: "));
b=int(input("Enter B value :: "));
c=int(input("Enter C value :: "));

if (a>b):
    if (a>c):
```

```
        print("A is Biggest");
    else:
        print("C is Biggest");
else:
    if (b>c):
        print("B is Biggest");
    else:
        print("C is Biggest");


print("End of the Program");



"Assignments"
#WAP to print smallest of 3-diff-numbers using nested-if stmts


"Assignment"
##Program (NestedIfs2.py)
#Program to demo Nested-If statements to check given number is +ve,-ve or 0

num = int(input("Enter any int-value : "));

if (num==0):
 print("Given Number is Zero");
else:
 if (num>0):
  print("Given Number is +VE");
 else:
  print("Given Number is -VE");
print("End of the Program");


#"Assignment"(nested-if)
##Program (NestedIfs3.py)
#WAP to display given number is even, odd or neither using nested-if

"""
**V) if-elif-else:-
Syntax:-
if condition1:
 stmts;
elif condtion2:
 stmts;
elif condtion3:
 stmts;
....
....
else:
 stmts;
next-stmt;(comes-out-of-CS)
......
= Here linearly one after the other conditions are checked
= Whichever condition is true, corresponding stmts are executed and remaining conditions are skipped till end
= If no-condition is true then last else part is executed
"""
##Program (IfElifElse.py)
#Program to accept 5 subject marks of a student and print the grade Distinction,1st-class,2nd-class,3rd-class or Fail
#Program to demo If-elif-else

print("Enter 5 Subject Marks :");
s1 = int(input("Sub1 : "));
s2 = int(input("Sub2 : "));
s3 = int(input("Sub3 : "));
s4 = int(input("Sub4 : "));
s5 = int(input("Sub5 : "));

total=s1+s2+s3+s4+s5;
avg = (total)/5;

print("Total-Marks : ",total);
print("Average-Marks : ",avg);

if (avg>=75):
 print("Distinction");
elif (avg>=65):
 print("1st-Class");
elif (avg>=50):
 print("2nd-Class");
elif (avg>=35):
 print("3rd-Class");
else:
 print("Failed");

print("End of Program");


#"Assignment"(if-elif-else)
#WAP to accept age of person and print following msgs
# (Baby, Kid, Teenage, Adult, Oldage)
"""
if personage >=65
elif personage>=18
elif personage>=12
elif personage>=5
else "Baby"
"""
#Assignment (if-elif-else)
#WAP to display given num is +ve, -ve, 0 (if-elif-else)
#WAP to display given num is even, odd, neither(if-elif-else)

"""
NOTE:-
= if-elif-else stmt is very efficient & fast when compared to multiple-if-stmts (or) nested-if-stmts becasuse here other-conditions are not-checked un-necessarily
"""



# 9th day 9.03-08-22

"""
***B) Iterative Statements:- (Looping-stmts)
= They are used to execute group-of-statements(indented block-of-code) 0 or more times based on condition/values
= We have 2-types of Iterative-Stmts,
 i) for loop
 ii) while loop
(do-while or foreach-loop are not in python)
```

```
i) for loop:-
= This loop is based on collection-of-values
Ex:-
Strings, List, Tuple, Set, Dictionary, Arrays, Range etc

Syntax:-
for x in collection-of-values:  [11,22,33]
 stmts;
 .....

= Here group-of-stmts are executed for each-value in collection
(each time coll-value is assigned to loop-var(x))
"""
##Program (ForLoopEx1.py)
# (Program to deme Python For Loop)
##Program (ForLoopEx1.py)
#Program to demo for-loop


#using str
sl = "Hello";  #coll.of.chars
for x in sl:
 print(x);
print("End of For Loop")
'''

'''
#using str with indexes
sl = "Hello";       #Hello(0,1,2,3,4)
i=0;
for x in sl:
 print("index",i,"char is",x);
 i=i+1;
print("End of For Loop")
'''

'''
#range()
for x in range(10):     #0 to 9
 print("Sai");

print()
for x in range(10):     #0 to 9
 print(x+1);
'''

'''
#sum of N-natural no's
sum=0;
for x in range(1,11):   #1 to 10
 sum=sum+x;
print("Sum :",sum);
'''

'''
#for loop single suite stmt (header & suite in single-line)
for x in range(1,11): print(x*x);
print()
for x in range(1,11): print(x*x*x);


#using list-coll
for x in [11,22,33,44,55]:
    print(x)

"""
NOTE:-
= For is purely based on coll.of.values


**Sp-case**
==>> for-loop with else-block:-
= In python, we can have else-block with for-loop
= else-block stmt is executed when for-loop values are finished (only 1-time in last)
Ex:-
 for x in coll:
  ....
 else:
  ....
"""
##Program (ForLoopElseEx1.py)
#Program to demo for-loop with else-block

vals = [11,22,33,44,55];
for x in vals:
 print(x);
else:    #else is executed only 1-time
 print("End of the Vals");

print("End of For Loop");


"""
**ii) while loop:-
= This loop-stmts is executed based on a condition (till it is false)
(here we use loop-var initial-value, condition-check, incre/decre)
Syntax:-
 inital-value;
 while condition(T/F):
  stmts;
  stmts;
  .....
  incre/decre;
"""

##Program (WhileLoopEx1.py)
#Program to print N-Natural numbers using while-loop

i=1;
while i<=10:
 print(i);
 i=i+1;
print("End of While Loop");
'''

'''
#Printing SUM of N-Natural Numbers
sum=0;
n = int(input("Enter any num : "));
i=1;
```

```
while i<=n:
 sum=sum+i;
 i=i+1;
print("Sum : ",sum);
print("Sum of :",n,"Natural numbers is :",sum);


#while loop single suite stmt
n = int(input("Enter N-value :: "))
i=1;
while i<=n: print(i*i); i+=1;
print()
i=1;
while i<=n: print(i*i*i); i+=1;


"""
**Sp-case**
**==> Infinite While-Loop:-
= A loop which is always true is called Infinite Loop
Ex:-
 while True:
  ...
  ...
  ...
"""
##Program (InfiniteLoop.py)
#(Program to demo Infinite-Loop while)

i=1
while True:
    print(i)
    i=i+1;

print("End of While Loop");
"""
NOTE:-
= (Ctrl+c) is used to come out of infinite loop
(ctrl+Pause-break-key)


**sp-case**
==> else-block with while-loop:-
= In python, we can have else-block with while-loop
= else stmt is executed when condition in while-loop is False
(only 1-time in last)
Ex:-
 while <<cond>>:
  ...
 else:
  ...
"""
##Program (WhileLoopElseEx1.py)
#Program to demo while-loop with else-block

i=1;
while i<=10:
 print(i);
 i=i+1;
else:  #here else-block is executed only once
 print("End of Natural-Nums");

print("\nEnd of While Loop")


# **sp-case**
# ==> Nested Loops:-
# = Using a particular loop inside another loop is called Nested-Loop
# (loop with-in loop)
# Ex:-
for i in range(3):  #outerloop(i=0,1,2) ---> rows
 for j in range(3): #innerlloop(j=0,1,2) ---> cols
  print(i,",",j,end="\t");
 print();


"""
NOTE:-
= Nested-Loops are mainly used to represent data in the form of rows and columns, table-data, matrices data etc
=** Here outer-loop represents no.of rows and inner-loop represents no.of columns
"""
##Program (NestedLoopEx1.py)
#Program to demo Nested-Loops

for i in range(3):   #outer-loop(0,1,2) => rows(i)
 for j in range(3):  #inner-loop(0,1,2) => cols(j)
  print(i,",",j,sep="",end="\t");
 print();

print("End of Program")


"""
***C) (Transfer Stmts) in Control Statements:-
(Jump Stmts)
(jumping from one-line of program to another-line)
a) break
b) continue
c) pass


A) break:-
(mainly used in loops(for/while))
= It is used to come out of looping-stmts using some condition
(Termination/Exit of Loop)

Syntax:-
loop:
 if cond:
  break;

=** it skips remaining stmts & also remaining iterations(cycle)
"""
##Program (BreakStmt.py)
##Program to demo break-stmt in Loops)


#using while loop
i=1
while i<=100:
```

```
    if i>=50:
        break;
    print(i)        #odd-nums
    i=i+2;
'''

'''
#using infinite while loop
i=2
while True:
    if i==102:
        break;
    print(i)        #even-nums
    i=i+2;


#using for-loop
for x in range(0,1001,3):
    if x>500:
        break;
    print(x)

"""
NOTE:-
= The best-way to come out of Infinite-Loop is with break-stmt(using cond..)


b) continue stmt:-
(mainly used in loops(for/while))
= It is used to skip current-iteration-stmts and continue with next-iteration(cycle)
Ex:-
 loop:
  ...
  if cond:
   continue;
  ....
"""
##Program (ContinueStmt.py)
##Program to demo continue-stmt in Loops

''
#using while-loop
i=1
while(i<=100):
    if i%2==0:  #even
        print(i)
    else:
        i=i+1;
        continue;
    i=i+1


#using for loop
for x in range(1,101,1):
    if x%2!=0:  #odd
        print(x)
    else:
        continue;

"""
c) pass statement:-
= It is used in program when some indented-block-of-code is not required for header-stmt
= pass takes control of execution to next-stmt in program
=** pass is like empty-stmts

Ex:-
 if (True):
  pass;
Ex:-
 def m1():
  pass;
 class A:
  pass;
"""
##Program (PassStmt.py)
#Program to demo pass-stmt in program with Loops

#inside loops
for x in range(1,101):
    if x%3!=0:
        pass;
    else:
        print(x)

print()
i=1;
while i<=100:
    if i%5!=0:
        pass;
    else:
        print(i)
    i=i+1







# 10th day 10.04-08-22

"""
==>> Numbers in Python:-
= Numbers mean numeric-values
= Python supports 3-types of numeric-values,
 a) int-type (10)
 b) float-type (10.5)
 c) complex-type (5+6j)

= For number dtypes, we have conversion funcs,
a) int()
b) float()
c) complex(a) ---> a+0j
d) complex(a,b) ---> a+bj

NOTE:-
= Python provides diff.funcs, using which we can perform mathematical-oper


==> Mathematical Functions in Py-Numbers:-
```

```
= Python provides "math.py", pre-defined module(lib-file)
= Here we hae diff. mathematical-funcs
Ex:-
 square-root
 logarithm
 trignometric
 factorial
 etc...

"""
##Program (MathFuncs.py)
##(Program to demo different Mathematical-Functions)
"""
NOTE:-
(using a module)
step1:-
 import math; module in program
step2:-
 use respective funcs or vars (.dot operator)
 math.sqrt(100)
 math.pi


1) abs(x):-
= It gives absolute (positive) value of given number "x"(Numeric-Expr)

2) ceil(x):-
= It gives least-integer-value which is greater-than or equals to given value
Ex:- 10--10.5--11(11 is ceil of 10.5)
= It is in math module (import it)
Ex:-
from math import ceil;
import math;

3) factorial()
= it gives factorial of given number
 math.factorial(5)

4) exp(x):-
= It gives exponential of x (e power of x)
= It is in math module

5) fabs(x):-
= It gives absolute value(+ve) of x
= it is mainly used on float-value(also works on int-values)
= It is in math-module

6) floor(x):-
= It gives next/greatest-integer-value which is greater-than or equals to given-value
= It is in math module (import it)

7) log(x):-
= It give natural logarithm value of x (log to the base-e)
= and x>0 (compulsory)
= It is in math-module

8) log10(x):-
= It give logarithm value of x (base-10)
= and x>0 (compulsory)
= It is in math-module

9) max(x,y,z,...):-
= It gives maximum value from given args

10) min(x,y,z,...):-
= It gives minimum value from given args

11) modf(x):- (it works on float-values)
= It gives fractional-part and integer-part of given value
= It is given as tuple-value
= It is in math-module
(both values are given as decimal-point values)

12)
pow(x,y):-
= Gives x to the power of y
pow(x,y,z):- (Not-Working)
= Gives pow(x,y)%z
= It is in math-module

13) round(x,n):-
= it gives rounded value of to n-digits after decimal-point
= If next-digit is >=5 then it rounds to next-digit
= If n is -ve then it rounds n-digits before decimal-point
(Here it is done based on 1's, 10's, 100's, 1000's places)
= If n value is not given then it is rounded to decimal-point itself

14) sqrt(x):-
= It gives square-root of given number (x>0)
= it is in math-module
"""


##Program (MathFuncs.py)
##(Program to demo different Mathematical-Functions)

import math;

#abs() function
print(abs(-9));
print(abs(9));
print(abs(0));
print(abs(-0));
print(abs(-9.5));
print(abs(9.5));
print(abs(0.0));
print(abs(-0.0));
'''

'''
#ceil() function (upper/next int-value)
from math import ceil;  #.dot is not-req
print(ceil(9.8));
print(ceil(9.2));
print(ceil(9.0));
print(ceil(9));
print(ceil(-9));
'''

'''
```

```
import math;
print(math.ceil(-9.8));
print(math.ceil(-9.2));
print(math.ceil(-9.0));
print(math.pi);
print(math.ceil(math.pi));
'''

'''
#exp() function (e--->Euler's number)
import math;
print(math.exp(3));
print(math.exp(-3));
print(math.exp(3.5));
print(math.exp(-3.5));
print(math.exp(0));
print(math.exp(1));
#We get exact e-value(2.718281828459045)
'''

'''
#fabs() function
import math;
print(math.fabs(3));
print(math.fabs(-3));
print(math.fabs(3.5));
print(math.fabs(-3.5));
print(math.fabs(0));
print(math.fabs(-0));
'''

'''
#floor() function (lower/prev int-value)
from math import floor;
print(floor(9.8));
print(floor(9.2));
print(floor(9.0));
print(floor(9));
print(floor(-9));
'''
'''
import math;
print(math.floor(-9.8));
print(math.floor(-9.2));
print(math.floor(-9.0));
print(math.pi);
print(math.floor(math.pi));
'''

'''
#log() function (base-e)
import math;
print(math.exp(1));  #We get exact e-value
print(math.log(math.exp(1))); #e power 0 is 1
print(math.log(10));
#print(math.log(-10)); #Error
print(math.log(10.5));
'''

'''
#log10() function (log value to base-10)
import math;
print(math.log10(10));
#print(math.log10(-10)); #ValueError
print(math.log10(10.5));
print(math.log10(1));
'''

'''
#max() function
print(max(10,20,30));
print(max(-10,-20,-30));
print(max(10.5,20.5,30.5));
print(max(-10.5,-20.5,-30.5));
'''

'''
#min() function
print(min(10,20,30));
print(min(-10,-20,-30));
print(min(10.5,20.5,30.5));
print(min(-10.5,-20.5,-30.5));
'''

'''
#modf() function(tuple())
import math;
print(math.modf(10.56));
print(math.modf(-10.56));
print(math.modf(10));
print(math.modf(-10));
print(math.modf(0));
print(math.modf(0.0));
'''

'''
#pow() function
import math;
print(math.pow(10,3));
print(math.pow(10,-3));
print(math.pow(100,0.5)); #it is 10 power 0.5(1/2) i.e. sqrt(100)
'''

'''
#round() function
print(round(10.12345,3));
print(round(10.12345,4));
print(round(12345.12345,-3));
print(round(12345.12345,-2));
print(round(12345.12345,-1));
print(round(12545.12345,-3));
print(round(10.123));
print(round(10.789));
'''

'''
#sqrt() function
import math;
print(math.sqrt(100));
print(math.sqrt(10.56));
```

```
print(math.sqrt(-10));  #ValueError


#factorial() function
import math
print(math.factorial(5))
print(math.factorial(6))



"""
==> Trigonometric Functions:-
= they are available in math-module (import math)
Ex:-
 sine, cosine, tangent etc (angles & radians)

"""
##Program (TrigonometricFuncs.py)
#Program to demo Trigonometric-Functions)

"""
1) sin(x):-
= Give sine of x (x in radians)
= here x radians should be converted to angles
= Angle => x*pi/180
= It is in math-module


2) cos(x):-
= Give cosine of x (x in radians)
= Angle => pi/180
= It is in math-module

3) tan(x):-
= Give tangent of x (x in radians)
= Angle => pi/180
= It is in math-module


4) hypot(x,y):-
= Gives Hypothesis value of Rt.angled triangle
i.e sqrt(x*x + y*y)
= It is in math-module

5) degrees(x):-
= Converts x radians to degrees
= It is in math-module

6) radians(x):-
= Converts x degrees to radians
= It is in math-module

==> Mathematical Constants:-
1) pi
2) e
= Both are in math-module
Ex:-
print(math.pi);
print(math.e);
"""

##Program (TrigonometricFuncs.py)
##(Program to demo Trigonometric-Functions)

##Program (TrigonometricFuncs.py)
#Program to demo Trigonometric-Functions)


#sin()
import math;
print(math.sin(30*math.pi/180));
print(math.sin(0*math.pi/180));
print(math.sin(90*math.pi/180));
'''

'''
#cos()
import math;
print(math.cos(60*math.pi/180));
print(math.cos(0*math.pi/180));
print(math.cos(180*math.pi/180));
'''


'''
##tan()
import math;
print(math.tan(0*math.pi/180));
print(math.tan(90*math.pi/180));
print(math.tan(45*math.pi/180));
'''

'''
#hypot()
import math;
print(math.hypot(2,2));
print(math.hypot(2,3));
'''

'''
#degrees()
import math;
print(math.degrees(0));
print(math.degrees(math.pi));
print(math.degrees(math.pi/2));
print(math.degrees(math.pi/4));
'''

'''
##radians()
import math;
print(math.radians(0));
print(math.radians(180));
print(math.radians(90));
print(math.radians(45));


#math-variables
import math;
print(math.pi);
```

```
print(math.e);
print(math.inf)



"""
-------------------------------------------------
==> Random Number Functions:-
= they are available in random-module (import random)
= they are used to generate Random-Numbers


##Program (RandomFuncs.py)
(Program to work with Random-Functions)


1) choice(sequence):-
= It gives random number/item from list,tuple,string
= It is in random-module
(import random;)


2) randrange(start,stop,step):-
= It gives random value for given range of values
= It is in random-module
= Here stop value is not-included in range
= step-value generates range of values

3) random():-
= It generates a random float number b/w 0 to 1
= It is in random-module

4) seed(x):-
= It sets the starting integer value before generating any random number using random() function

5) shuffle(list):-
= It will shuffle the values in a list
= it is in random-module
= It shuffles the values randomly

6) uniform(x,y):-
= It gives a random float-value b/w x and y
= It is in random-module
= last value is not-included(y)
"""
##Program (RandomFuncs.py)
#Program to work with Random-Functions


#choice() (works only on index-coll)
import random;
list1 = [11,22,33,44,55];
print(random.choice(list1));
s1 = "SaiRamKumar";
print(random.choice(s1));
tup1 = (10,20,30,40,50);
print(random.choice(tup1));
#set1 = {1,2,3,4,5};
#print(random.choice(set1)); #set does not have indexes
'''

'''
##randrange(start,end,step)
import random;
print(random.randrange(10,100,2));
print(random.randrange(3,100,3));
print(random.randrange(-20,-1));
#print(random.randrange(-1,-100));  #Empty-range
'''

'''
#random() #0 to 1(not-included)
import random;
print(random.random());
print(random.random());
'''

'''
#shuffle()
import random;
list1 = [10, 20, 30, 40,50];
random.shuffle(list1)
print(list1);
random.shuffle(list1)
print(list1);



#uniform()
import random
print(random.uniform(1,5));
print(random.uniform(11,15));



#11th day 11.05-08-22

"""
***===>>> Strings in Python:-
= String is collection-of-chars represented in quotes
i.e, single-quotes'...'
or double-quotes"..."
or triple-quotes'''....'''
Ex:-
s1 = 'Sai';
s2 = "Sai Ram"; (space is also 1-char)
s3 = '''
#101, St.No.9
HimayathNagar,
Hyd(TS)
India
'''

NOTE:-
1)
= Use single-quotes for single-word
= Use double-quotes for multi-words(1-line)
= Use triple-quotes for multi-line-text
2)**
= Python does not support char-dtype. Even single-char is also taken as string-type
```

```
Ex:-
ch = 'A'; or "A"
print(type(ch));
"""
##Program (StringsEx1.py)
##Program to demo strings in python
##Program (StringsEx1.py)
##Program to demo strings in python

#str-type
ch = 'A';
print(type(ch));
print(ch);

ch = "A";
print(type(ch));
print(ch);

ss='Sai Ram Kumar';
print(type(ss));
print(ss);

ss="Sai Ram Kumar";
print(type(ss));
print(ss);

print()
addr = """Sai Ram Kumar,
HimayathNagar,
Hyderabad.
""";
print(addr);
addr = """
    Sai Ram Kumar,

    HimayathNagar,

    Hyderabad.
""";
print(addr);

print()
#quotes with in quotes
#ss = 'Hi Hello's Welcome'; #Error
#ss = "Hi Hello's Welcome";
#ss = 'Hi Hello\'s Welcome';
#ss = 'Hi Hello"s Welcome';
#ss = "Hi Hello\"s Welcome"; #Error
#ss = 'Hi Iam "Sai Ram Kumar"';
#ss = 'Hi Iam "Sai Ram Kumar" and "Python" Trainer';
#ss = 'Hi Iam "Sai Ram Kumar" and \'Python\' Trainer';
#ss = '''#Hi Iam "Sai Ram Kumar" and 'Python' Trainer''';
ss = """Hi Iam "Sai Ram Kumar" and 'Python' Trainer""";
print(ss);

"""
NOTE:-
1) in triple-quotes string, no need to use \t or \n chars. they are taken auto. wrt given-text
2) to represent quotes with-in quotes, use one-type-quotes in another(but not same type of quotes)
3) However we can use same-type-of quotes inside same-type of quotes using \(ESC-SEQ-char)


==> Getting or Accessing chars from string:-
= It is done in 2-ways,
a) using indexes
b) using slice operator

a) using indexes:-
==Diagran==
Ex:-
 "hello"

= Python supports both +ve or -ve indexes
= +ve indexes are First to Last (L->R) (Forward) (0 to n-1)
= -ve indexes are Last to First (R->L) (Backward) (-1 to -n)
Ex:- (we use index with [] subscript-operator/index-oper)
ss = "Hello";
print(ss[0]); 1,2,3,4
print(ss[-1]); -2,-3,-4,-5
print(s[10]); #Error (string index out of range)

##Program (StringEx2.py)
#Program to accept string and print its chars with indexes

NOTE:-
= the best way to access elements of any collection is with loops(For-loop)


b) Using slice-operator:-
Syntax:-
 str[beginIndex : endIndex : stepValue]

= This acts like sub-string
= beginIndex is starting-index
= endIndex is (last-index - 1)
= stepValue is increment-value

NOTE:-
= If beginIndex is not given then it will take from 0-index
= If endIndex is not given then it will take till last-index
= Default stepValue is 1


==> Strings with Mathematical Operators:-
= + operator is used for string concatenation
= * operator is used for string repeatition

==> Length of a String:-
= len() gives length of a string
Ex:-
ss = "Sai Ram Kumar";
print(len(ss));
"""

##Program (StringEx2.py)
#Program to accept string and print its chars with indexes & slice-operator


#using index and [] oper
ss = "Hello";
```

```
print(ss[0]);
print(ss[1]);
print(ss[2]);
print(ss[3]);
print(ss[4]);
print(ss[-1]);
print(ss[-2]);
print(ss[-3]);
print(ss[-4]);
print(ss[-5]);
#print(ss[10]); #IndexError
'''

'''
#string access with loop
ss = input("Enter any String : ");
i=0;
for x in ss:
 print(i,"=====>",x)
 i=i+1;
'''

'''
#using slicing operator
ss = "Welcome to Python Session";
print(ss[1:9:1]);
print(ss[1:9]);
print(ss[1:9:2]);
print(ss[:9]);
print(ss[9:]);
print(ss[::]);
print(ss[:]);
print(ss[::-1]);
print(ss[::-2]);
print(ss[-1:-9:-1]);
print(ss[-1:-9:-2]);
print(ss[-9:-1:-1]);
print(ss[1:0:2]);
'''

'''
#str addition(+) & repetition(*)
s1 = "Sai Ram";
s2 = " Kumar";
print(s1+s2);
print(s1*5);
print(s1*-5);
print(s1*0);


#len() func
ss = "Sai Ram Kumar";
ss="Hello"
print(len(ss));

"""
(****)
==> Checking or Finding or Searching String:-
= Check whether given string or char is present in original-string or not
= It is done with "in" or "not in" operator
(membership-operators)
"""
#Program (StringEx3.py)
#(Prog to work with strings)
##Program (StringEx3.py)
#Program to work with strings(check/find/search)


#case-1
ss ="Hello Students, Welcome to Python Class";
print("to" in ss);
print("," in ss);
print("hi" in ss);
print("Python" not in ss);


#case-2
s1 = input("Enter any main or org. string : ");
s2 = input("Enter any searching string : ");
if s2 in s1:
 print(s2,": is found in org-string");
else:
 print(s2,": is NOT found in org-string");

"""
(*****)
==> String Comparision:-
= Comparision operators are used for string-comparison
i.e, <,<=,>,>=,==,!= Relational Operators
= Comparision is done based on Character Ascii-Codes
A-Z (65-90)
a-z (97-122)
0-9 (48-57)
space (32)
$ (36)
@ (64)
+ (43)
etc...
==Diagram==(Example)


##Program (StringEx4.py)
(Prog to perform with string-comparisons)

NOTE:-
= Internally difference of ASCII Codes are taken for comparison
"""
##Program...(StringEx4.py)
#Prog to perform with string-comparisons

s1 = input("Enter 1st string : ");
s2 = input("Enter 2nd string : ");
if s1 == s2:
 print("Both String are SAME");
elif s1<s2:
 print("1st String is Less than 2nd String");
else:
 print("1st String is Greater than 2nd String");

print("End of the Program") ;
```

```
# 12th day 12.06-08-22

"""
==> Removing spaces from string:-
= We can remove or del or truncate extra-spaces from a string from left/right/both sides
= For this we have 3 functions,
a) rstrip() :- removes extra spaces from right-side
b) lstrip() :- removes extra spaces from left-side
c) strip() :- removes extra spaces from both-sides
"""
##Program (StringEx5.py)
##Prog to perform with string-stripping
##Program (StringEx5.py)
#Prog to perform with string-stripping/deleting/truncate extra spaces

ss = input("Enter your city-name : ");
#sscity = ss.strip();
#sscity = ss.lstrip();
sscity = ss.rstrip();
print(sscity,"is your city!!!");

"""
(***)
==> Finding Sub-strings:-
= To find any sub-string in main-string is done with 4-methods,
a) find() //both forward-direction
b) index()
-----------------
c) rfind() //both backward-direction
d) rindex()
(in all cases index is given from beginging 0 to n-1)
"""
##Program (StringEx6.py)
##Program to work with string-functions or methods

"""
==> Counting all sub-strings in main-string:-
= It is done with count() method/function
= it gives count of sub-string in org-string
Ex:-
ss.count(subss); //from begin(0) to last(len)
ss.count(subss,beginIndex,endIndex);


==> Replacing a String with another string:-
= It is done with below method,
Syn:-
 ss.replace(oldstr, newstr);
= it replaces oldstr with newstr in org-str but we get new-string

NOTE:-
= String value(object) are immutable
(org-str cannot be modified but can be re-assigned)
= Any changes done to org-str, new str-obj is created
= Hence in replace(), we got new-str after replace
"""

##Program (StringEx6.py)
##Program to work with string-functions or methods

#find() & rfind()
ss = "Hello Students, Welcome to Python Session, Hello by Sai sir";
#print(ss.find("Hello"));    #gives only 1st occurance
#print(ss.find("Java"));  #-1 on un-successful search
#print(ss.rfind("Hello"));
#print(ss.rfind("Java"));
##find(), rfind() we get -1 for un-successful search


#find(string,beg-ind,end-ind) & rfind(string,beg-ind,end-ind)
ss = "Hello Students, Welcome to Python, Hello all";
print(ss.find("Hello",6)); #From 6th-index to last-index
print(ss.find("Hello",6,20));
print(ss.rfind("Hello",0,30));
print(ss.rfind("Hello",6,len(ss)));
#print(ss.rfind("Hello",-1,-10)); #Here indexes cant be -ve
'''

'''
#index() and rindex() #here we get ValueError for un-successful search
ss = input("Enter Main String : ");
subss = input("Enter Sub String : ");
#print(ss.index(subss));
#print(ss.index(subss,9,len(ss)));
#print(ss.rindex(subss));  #0(begin) & len(end)
print(ss.rindex(subss,0,17));
'''

'''
#count()
#Counting all sub-strings in main-string
ss = "Hello Welcome to Python Hello users";
#print(ss.count("Hello"));
#print(ss.count("Hello",6));
#print(ss.count("Hello",6,len(ss)));
#print(ss.count("Welcome",6));
#print(ss.count("Welcome",8));
'''

'''
#replace()
#Replacing old-str with new-str
ss = "Hello, Welcome to Python, Hello users";
#newss = ss.replace('e','*');
newss = ss.replace('Hello','Hi');
print(ss);
print(newss);


#Immutable str-obj
ss = "Hello Welcome";
newss = ss.replace('e','*');
print(ss,"===>",id(ss));
print(newss,"===>",id(newss));

"""

(***)
```

```
===> Splitting of Strings:-
= We can split given string into sub-strings
(we get list of sub-string values i.e, ['','','','',''])
= splitting is done based on separator
= Default separator is space
Ex:-
 sublist  = ss.split('separator');
= Return-type is List dtype(list of string-values)
"""
##Program (StringEx7.py)
##(Program to work with string functions)

"""
==> Joining of Strings:-
= it is used to join coll.of.strs to single string using a seperator
Ex:-
 newss = separator.join(group of string);
Ex:-
 newss = " ".join(list1);
  "_"
  ":"
  ";"
  ","
"""
##Program (StringEx7.py)
# (Program to work with string functions)

"""
==> Changing case of a string:-
= Using below methods, we can change string-cases
a) upper()   #converts to upper-case
b) lower()   #converts to lower-case
c) swapcase()  #converts to upper to lower & lower to upper
d) title()   #converts each word lst letter to Upper-case
e) capitalize() #only lst-char will be converted to upper-case
"""
##Program (StringEx7.py)
#(Program to work with string functions)

"""
==> Checking Starting and Ending part of a string:-
= These methods returns True or False
Ex:-
 ss.startswith(str);
 ss.endswith(str);
"""
##Program (StringEx7.py)
##Program (StringEx7.py)
#Program to work with string functions(operations)


#split()
ss = "Hello Students, Welcome to, Python Session";
#listss = ss.split(' ');
#listss = ss.split();
listss = ss.split(",");
print(listss);
'''

'''
#join()
listss = ['Hello', 'Students', 'Welcome', 'to', 'Python', 'Session'];
#Joining of strings
ss = " ".join(listss);
print(ss);
ss = "-".join(listss);
print(ss);
ss = "".join(listss);       #empty-string
print(ss);
ss = "-".join(["Sai","Ram","Ali"]);
print(ss);
listss = ["Hyd","Mum","Che","Delhi"];
ss = ":".join(listss);
print(ss);
'''

'''
##Changing-cases in string
ss = "Hello welcome to Python session";
print(ss.upper());
print(ss.lower());
print(ss.swapcase());
print(ss.title());
print(ss.capitalize());


#Starting & Ending strings
ss = "Hello Students, Welcome to Python Session";
print(ss.startswith("Hello"));
print(ss.endswith("Session"));
print(ss.startswith("Hi"));
print(ss.endswith("Bye"));


"""
==>> Checking Type-of-Characters in string:-
= For this we use following methods,
a) isalnum() #checks for alphabets or digits (a-z,A-Z,0-9)
b) isalpha() #checks for alphabets only (a-z,A-Z)
c) isdigit() #checks for digits only (0-9)
d) islower() #checks for lower-case alphabets only (a-z)
e) isupper() #checks for upper-case alphabets only (A-Z)
f) istitle() #checks for title-case string
g) isspace() #checks for ONLY space chars only

=** All functions returns True or False
"""
##Program (StringEx8.py)
#(Program to work with string functions)

##Program (StringEx8.py)
#Program to work with string functions (Type of Characters)

ss="SaiRam123";
print(ss.isalnum());
print(ss.isalpha());

print()
ss="SaiRam";
print(ss.isalpha());
```

```
print(ss.isdigit());

print()
ss="123123";
print(ss.isdigit());

print()
ss="sairam";
print(ss.islower());
print(ss.isupper());

print()
ss="sairam123";
print(ss.islower()); #**sp-case(T)
print()
ss="SAI123";
print(ss.isupper());    ##**

print()
ss="Hello Students Welcome To Python";
print(ss.istitle());

print()
ss="Hello students Welcome to Python";
print(ss.istitle());

print()
ss=" ";
print(ss.isspace());
ss="\t";
print(ss.isspace());
ss="\n";
print(ss.isspace());
ss="\b";  #**sp-case
print(ss.isspace());
ss="\r";
print(ss.isspace());

"""
==>> Formatting the string using print():-
({} replacement-operator and format() func)
= In print(), while printing string-data(quotes), we can use replacement-operator and format() func
Ex:-
 print("Sum of {} and {} == {}".format(a,b,sum))
 = here a,b,sum are replaced inside string at {} replacement-operator

##Program (StringEx9.py)
(Program to work with string formats for print() )

NOTE:-
= We can use {} replacement-operator in 3-cases
 a) without indexes
 b) with indexes
 c) with vars (order can be changed in format())
"""
##Program (StringEx9.py)
#Program to work with string formats with print() using {} & format()


#case-1
rno=1001;
name="Sai";
height=6.0;
print("RollNo : {}\nName: {}\nHeight: {}".format(rno,name,height));


#case-2
print();
rno=1001;
name="Sai";
height=6.0;
#using-indexes
print("RollNo : {0}\nName:{1}\nHeight: {2}".format(rno,name,height));

#case-3
#using-vars (we can change the order)
rno=1001;
name="Sai";
height=6.0;
print("RollNo : {x}\nName: {y}\nHeight: {z}".format(x=rno,y=name,z=height));
print("RollNo : {x}\nName: {y}\nHeight: {z}".format(y=name,z=height,x=rno));



# 13th day 13.08-08-22

"""
==> Collections in Python::-
(Data-Structures)
= Collections means group of values(Same-type/Diff-type)
= Data-Structures means org. of data in proper-way
=** Advantages of DS,
 a) Operations on Data are Easy
 Ex:-
  Add-data
  Del-Data
  Update-Data
  Search-Data
  Sort-Data
 b) Access-data is also Easy

==*** Python supports 5-types of Collections or DS
 A) List
 B) Tuple
 C) Set
 D) FrozenSet
 E) Dictionary


A)*** List Collection::-
def:-
 = It is collection of diff-objects as single-unit
=> Tech-Points???
(ref-notes)
= In list(Features),
 = Order is preserved
 = Duplicate objs are allowed
 = Heterogeneous/Homogeneous objects are allowed (diff-type or same-type)
 = Size is dynamic
```

```
 (incre/decre as per adding/deleting)
 = Values/Objs are represented in [....] with commas(,) separator
 = It provides indexes for values/objs (0 to n-1) or (-n to -1)
 = Both +ve and -ve indexes are supported
 = +ve indexes (L->R or First->Last) Forward and -ve indexes (R->L or Last->First) Backward-direction
Ex:-
list1 = [10,20,30,40,50];
   0, 1, 2, 3, 4  (Forward)
  -5,-4,-3,-2,-1  (Backward)
==Diagram==
NOTE:-
=** List-DS elements are "Mutable"
(org-data can be modified...)


==> Creation of List Objs:-
(WORKING WITH LIST OBJECT)
Ex:- (Empty-List)
list1 = [];  #empty []
print(list1);
print(type(list1));

##Program (ListEx1.py)
(Program to work with List DS)

Ex:- (List with Elements)
list1 = [10,20,30,40,50];


Ex:- (List with Dynamic-Elements)
list1 = [];
list1 = eval(input("Enter some list elements : ")); #[]11,22,33,44,55
print(list1);
print(type(list1));

=** Give Elements in list format only i.e, [11,22,33,44,55]
=** eval() converts input-list-data to list-type-values & stores in list1-variable

Ex:- (using list() and range() functions)
list1 = list(range(0,20,2));
print(list1);
print(type(list1));

=** range() is used to generate a range of values
i.e, range(beginValue,endValue,stepValue);
= stepValue can be +ve or -ve
=** list() conversion-function converts coll-of-values to list-type

==> other-extra-oprations on list:-
Ex:- (using a string with list())
ss = "Sai Ram Kumar";
list1 = list(ss);
print(list1);

Ex:- (using a string with split())
=**split() splits our string into sub-strings
(such sub-strings are kept in list)
ss = "Sai Ram Kumar";
list1 = ss.split();
print(list1);
print(type(list1));

Ex:- (Nested-List)
= It is list inside another-list
list1 = [11,22,33,[99,88]]; #here [99,88] is sub-list
print(list1);
print(list1[3]);
"""

##Program (ListEx1.py)
#Program to work with List DS (ListEx1.py)


#Empty-List (empty-[] brackets)
list1 = [];
print(list1);
print(type(list1));
'''

'''
#List with Elements
list1 = [10,20,30,40,50];
print(list1);
print(type(list1));
'''

'''
#List with Dynamic-Elements #given in [] sqr-brackets
list1 = [];
list1 = eval(input("Enter some list elements : "));
print(list1);
print(type(list1));
'''

'''
#using list() conversion-function and range() functions
list1 = list(range(0,20,2));
print(list1);
print(type(list1));

list1 = list(range(0,20,-2));
print(list1); #empty-list
print(type(list1));

list1 = list(range(20,0,-2));
print(list1);
print(type(list1));
'''

'''
#using a string with list()
ss = "Sai Kumar";
list1 = list(ss); #each char is converted to list-values
print(list1);
'''

'''
#using a string with split()
ss = "Sai Ram Kumar have a nice day";
list1 = ss.split(" ");  #by default splits data wrt space
```

```
print(list1);
print(type(list1));


#Nested-List
list1 = [11,22,33,[99,88]];
print(list1);

"""
-------------------------------------------------------------
(Advanced Operations on List-DS)
==> Accessing List Elements:-
= List Elements can be accessed using index or slice operator (:)
1) Using index:-
= Index are both +ve and -ve
= +ve indexes are (0 to n-1) (F->L)
= -ve indexes are (-1 to -n) (L->F)
Ex:-
list1 = [54,64,74,84,94];
print(list1[0]);
print(list1[1]);
....
print(list1[-1]);
print(list1[-2]);
....
print(list1[10]); #IndexError: list index out of range

//Program (ListEx2.py)
(Program to  work with list DS)

2) using Slice operator:-
= Slice Operator is : (colon) used with [](subscript-operator)
Syntax:-
 list2 = list1[startIndex:endIndex:stepValue]
= startIndex, its default value is 0
= endIndex, its default value is (len of list) (it is not included)
= stepValue, its default value is 1
(it work for both +ve,-ve indexes and also for stepValue)
Ex:-
list1 = [11,22,33,44,55,66,77,88,99,110];
print(list1[2:7:2]);
print(list1[4::2]);
print(list1[3:7]);
print(list1[8:2:-2]);
print(list1[4:100]);

NOTe:-
list1[::]
list1[:]

=> List v/s Mutability:-
= List object values can be modified i.e, Mutable
Ex:-
list1 = [11,22,33,44,55];
print(list1);
list1[1]=2222;
print(list1);


==> Traversing List Elements:- (linear/sequential access of list)
= Best-way is using a loop,
1) using while loop:-
list1 = [11,22,33,44,55,66,77,88,99,110];
i=0;
while i<len(list1):
 print(list1[i]);
 i=i+1;

2) using for-loop:-
list1 = [11,22,33,44,55,66,77,88,99,110];
for x in list1:
 print(x);

3) display only even numbers:-
list1 = [11,22,33,44,55,66,77,88,99,110];
for x in list1:
 if x%2==0:
  print(x);

4) display elements index-wise:-
Ex:-
list1 = [11,22,33,44,55];
x = len(list1);
for i in range(x):
 print(list[i] :"index is",i,"and -ve index is":(i-x));

"""

##Program...(ListEx2.py)
#Program to work with List DS (ListEx2.py)

#Using indexes (0 to n-1) or (-1 to -n)
list1 = [54,64,74,84,94];
print(list1[0]);
print(list1[1]);
print(list1[2]);
print(list1[3]);
print(list1[4]);
print()
print(list1[-1]);
print(list1[-2]);
print(list1[-3]);
print(list1[-4]);
print(list1[-5]);
#print(list1[10]); #IndexError: list index out of range
'''

'''
#Using slice-operator(Slicing)
#list1[startindex:endindex:stepindex]
list1 = [11,22,33,44,55,66,77,88,99,110];
print(list1);
print(list1[2:7:2]);
print(list1[4::2]);
print(list1[3:7]);
print(list1[8:2:-2]);
print(list1[4:100]); #No-Error(for 100) but takes till last-index
'''

'''
```

```
#List is Mutable (org-data can be modified)
listl = [11,22,33,44,55];
print(listl);
listl[1]=222;
print(listl);
'''

'''
#using for-loop:-
listl = [11,22,33,44,55,66,77,88];
for x in listl:
 print(x);


#using while loop:-
listl = [11,22,33,44,55,66,77,88]; #len=8
i=0;
while i<len(listl):
 print(listl[i]);
 i=i+1;

"""
-----------------------------------------------------------
==> Different Functions of List DS:-
1) len():
= it is common for all-collections
= gives no of elements/values/objs in a list DS
Ex:-
listl = [11,22,33,44,55];
print(len(listl));

//Program (ListEx3.py)
(Program to  work with list DS)

2) count():-
= Gives element occurred how many times in a list
listl = [11,22,33,11,22,33,44,55,55];
print(listl.count(11));
print(listl.count(22));
print(listl.count(33));
print(listl.count(44));
print(listl.count(55));


3) index():-
= It gives lst index position of given element in a list
Ex:-
listl = [11,11,22,22,33,33,44,44,55];
print(listl.index(11));
print(listl.index(22));
print(listl.index(33));
print(listl.index(44));
print(listl.index(55));
#print(listl.index(66)); #ValueError

=** If element is not available then we get "ValueError"
= Hence, we can check for element using "in" operator
Ex:-
print(66 in listl);


==> Manipulation Functions in list DS:-
1) append():-
= Adds item/element/value/obj at the end of list
Ex:-
listl=[];
listl.append("A");
listl.append("B");
listl.append("C");
listl.append("D");
listl.append("E");
print(listl);

Ex:- (Add elements in list divisible by 5)
listl=[];
for x in range(51):
 if (x%5==0):
  listl.append(x);
print(listl);


2) insert():-
= adds element at specified index position
Ex:-
listl = [11,22,33,44,55];
listl.insert(1,999);
print(listl);

NOTE:-
= If specified index is greater than max-index then element is added at last
= If specified index is lesser than min-index then element is added at begin

3) extend():-
= Adds elements of one-list into another-list
Ex:-
listl.extend(list2);

listl = [11,22,33];
list2 = ["Hi","Hello","Welcome"];
listl.extend(list2);
print(listl);

Ex:-
listl = [11,22,33];
listl.extend("World");
print(listl);

=** Here string is added/appended as individual chars to lst list


4) remove():-
= removes specified element from the list
= If element is more than once then only lst occurance is removed
Ex:-
listl = [11,22,11,22,33,11];
listl.remove(11);
print(listl);
listl.remove(22);
print(listl);
listl.remove(44); #ValueError
```

```
=** If element is not available then we get "ValueError"


5) pop():-
= it removes and gives/returns last element of list DS
list1 = [11,22,33,44,55];
print(list1.pop());
print(list1.pop());
print(list1);

list1=[];
print(list1.pop());  #IndexError
=** If list is empty then we get "IndexError"

NOTE:-
= pop() and append() functions can be used to implement stack DS operations (LIFO approach)

= pop() with indexes,
Ex:-
pop(index); #removes and returns specified indexed element
Ex:-
list1 = [11,22,33,44,55];
print(list1.pop());
print(list1.pop(1));
print(list1);
print(list1.pop(10)); #IndexError

==> Difference between remove() and pop():-
1)
remove() removes given element from list (1st occurance)
pop() removes only last-element from list

2)
remove() just removes element and doest not return any value
pop() removes last-element and return that value (or using index)

3)
If element is not there we get "ValueError"
If element is Empty or index is not there then we get "IndexError"


NOTE:-
= List size is dynamic (incre/decre as per elements)
= append(), insert(), extend() increases the size
= remove(), pop() decreases the size
"""
##Program (ListEx3.py)
#Program to  work with list DS (ListEx3.py)


#len()
list1 = [11,22,33,44,55];
print(len(list1));
list1 = [11,22,33];
print(len(list1));
list1 = [];
print(len(list1));
'''

'''
#count()
list1 = [11,22,33,11,22,33,44,55,55];
print(list1.count(11));
print(list1.count(22));
print(list1.count(33));
print(list1.count(44));
print(list1.count(55));
'''

'''
#index() gives 1st-index-postion(Search)
list1 = [11,11,22,22,33,33,44,44,55];
print(list1.index(11));
print(list1.index(22));
print(list1.index(33));
print(list1.index(44));
print(list1.index(55));
#print(list1.index(66)); #ValueError
##print(66 in list1);
'''

'''
#append() adds from last
list1=[];
print(list1);
list1.append("A");
list1.append("B");
list1.append("C");
print(list1);
list1.append("D");
list1.append("E");
print(list1);
'''

'''
#insert() -> inserts element in b/w(other moves to aside)
list1 = [11,22,33,44,55];
print(list1);
list1.insert(1,999);
print(list1);
list1.insert(10,987); #inserted at last
print(list1);
list1.insert(-10,789); #inserted at begin
print(list1);
'''

'''
#extend() -> adds other-coll to our list-coll
list1 = [11,22,33];
list2 = ["Hi","Hello","Welcome"];
list1.extend(list2);
print(list1);
print()
list1 = [11,22,33];
list1.extend("World"); #here each-char is added as list-element
print(list1);
'''

'''
```

```
#remove() -> dels lst-occurance element
list1 = [11,22,11,22,33,11];
print(list1)
list1.remove(11);
print(list1);
list1.remove(22);
print(list1);
list1.remove(44); #ValueError
'''

'''
#pop() removes last or random element o.w Error for empty
list1 = [11,22,33,44,55];
print(list1)
print(list1.pop());
print(list1.pop());
print(list1);
#list1=[];
#print(list1.pop());  #IndexError



#pop(index) remmoves elements using indexes
list1 = [11,22,33,44,55];
print(list1)
print(list1.pop());
print(list1.pop(0));
print(list1);
#print(list1.pop(10)); #IndexError (out of range)

"""
-------------------------------------------------------------
(List Ordering)
==> Ordering the elements of list DS:-
1) reverse():-
= Gives reverse-order of elements in a list
Ex:-
list1 = [11,22,33,44,55];
list1.reverse();
print(list1);

//Program (ListEx4.py)

2) sort():-
sort(reverse=true/false);
= Default order is preserved
= For Numbers, it is ASC order (false)
= For Strings, it is Alphabetical order (false)
Ex:-
list1 = [44,11,55,22,33];
list1.sort(); #default sort() order is ASC-order (reverse=False)
print(list1);

Ex:-
list1 = ["hyd","delhi","chennai","apple","ball"];
list1.sort();
print(list1);

NOTE:-
= For sort(), compulsory use same-dtype elements o.w "TypeError"
list1 = ["B",11,"C",22,"A"];
list1.sort();
print(list1);

=** However it is supported in Python2(1st Nums and 2nd Strs) not in Python3

=** For DESC order or nums or strs, we use
Ex:-
list1 = [44,11,55,22,33];
list1.sort(reverse=True);
print(list1);
list1.sort(reverse=False);
print(list1);

Ex:-
list1 = ["hyd","delhi","chennai","apple","ball"];
list1.sort(reverse=True);
print(list1);
list1.sort(reverse=False);
print(list1);



==> Alias and Clone of list DS:-
= Alias means alternate-name
(same data in list but 2 or more names)
= It is done by giving reference to other list-variable
Ex:-
list1 = [11,22,33,44,55];
list2 = list1;
print(id(list1));
print(id(list2));
==Diagram==

=** Here changes done with any variable is update to other variable also

Ex:-
list2[0]=1111;
print(id(list1));
print(id(list2));

NOTE:-
= To avoid above problem, we go for Cloning
= It is done using slice-operator or copy() function
(here we get new-list-ds in memory)

Ex:- (slice-operator)
list1 = [11,22,33,44,55];
list2 = list1[:];
list2[0]=1111;
print(id(list1));
print(id(list2));

Ex:- (copy())
list1 = [11,22,33,44,55];
list2 = list1.copy();
list2[0]=1111;
print(id(list1));
print(id(list2));
```

```
NOTE:-
"=" operator means Aliasing
copy() or slicing means Cloning


==> Mathematical Operators on List DS:-
= For this we use + and * operators

1) Concatenation using (+):-
= It will combine 2 lists into single one
Ex:-
list1 = [11,22,33];
list2 = [44,55,66];
list3 = list1+list2;
print(list3);

Ex:-
list3 = list1+77; #TypeError
list3 = list1+[77]; #Valid


2) Repetition using (*):-
= It is used to repeat a list given no.of times
Ex:-
list1 = [11,22,33];
list2 = list1*3;
print(list2);

=> Comparing List values:-
= It is done with ==, != operators
Ex:-
list1 = [11,22,33];
list2 = [11,22,33];
list1 = [44,22,33];
print(list1==list2);
print(list1==list3);
print(list2!=list3);

NOTE:-
= Using comparison operators(==,!=) then make sure,
 = No.of Elements
 = Order of Elements
 = Content of Elements (Case-Sensitive)
Ex:-
list1 = ["hi","hello","welcome"];
list2 = ["hi","hello","welcome"];
list3 = ["HI","HELLO","WELCOME"];
print(list1==list2);
print(list1==list3);
print(list2!=list3);

NOTE:-
= Using comparison operator (<,<=,>,>=) then only 1st Element comparison is done
Ex:-
list1 = [11,22,33];
list2 = [44,55,66];
print(list1<list2);
print(list1<=list2);
print(list1>list2);
print(list1>=list2);

Ex:-
list1 = ["Sai","Ram","Ali"];
list2 = ["Ram","Ali","Sai"];
print(list1<list2);
print(list1<=list2);
print(list1>list2);
print(list1>=list2);



==> Membership Operator:-
= Check whether elements are there in list or not
= It is done using "in", "not in" operators

Ex:-
list1 = [11,22,33,44,55];
print(11 in list1);
print(11 not in list1);
print(111 in list1);
print(111 not in list1);

==> clear():-
= It removes all the elements of a list
list1 = [11,22,33,44,55];
print(list1);
list1.clear();
print(list1);


=> Nested-List:-
= One list inside another list is called as Nested-List
Ex:-
list1 = [11,22,33,[44,55]];
print(list1[0]);
print(list1[1]);
print(list1[2]);
print(list1[3]);
print(list1[3][0]);
print(list1[3][1]);
==Diagram(with indexes)==


"Assignment"
==> Nested-List as Matrix:- (Representation)
= Nested-List can be used to represent a matrix (rows & cols)
= Each inner-list/sub-list/nested-list represents 1-row
= inner-list/sub-list/nested-list values represents column-values
Ex:-
A = [[11,22,33],[44,55,66],[77,88,99]];
print(A)
print("Row-wise Elements :");
for rw in A:
 print(rw);
print("Matrix-Style Elements :");
for i in range(len(A)):
 for j in range(len(A[i])):
  print(A[i][j],end=" ");
 print();
"""
```

```
##Program (ListEx4.py)
#Program to work with list DS


#list ordering (ASC/DESC)
#reverse()
#list1 = [11,22,33,44,55];
list1 = ["hi","hello","welcome"]
print(list1)
list1.reverse();
print(list1);
'''

'''
#sort() -> sorts elemnts ASC/DESC
list1 = [44,11,55,22,33];
print(list1);
list1.sort(); #sort() default-order is ASC(False)
print(list1);
print()
list1 = ["hyd","delhi","chennai","apple","ball"];
list1.sort(); #sort() default-order is ASC(False)
print(list1);
'''

#list1 = ["B",11,"C",22,"A"];
#list1.sort(); #Not-possible with diff-type of data
#print(list1);

'''
#sort(reverse=True) #DESC-order
list1 = [44,11,55,22,33];
list1.sort(reverse=True);
print(list1); #DESC
list1.sort(reverse=False);
print(list1); #ASC
'''

'''
list1 = ["hyd","delhi","chennai","apple","ball"];
list1.sort(reverse=True);
print(list1);
list1.sort(reverse=False);
print(list1);
'''


'''
#Alias(alternate-name)
list1 = [11,22,33,44,55];       #org-list(list1)
list2 = list1; #in memeory, list1 & list2 both points to same-location-values
print(id(list1));
print(id(list2));
print(list1);
print(list2);
#operation on alias-var-name
list2[0]=1111; #mutable-object
print(list1);
print(list2);
'''

'''
#list-cloning(we get separate data for memory)
#dup-obj or separate-obj
list1 = [11,22,33,44,55];
list2 = list1[:]; #slicing-operator
print(id(list1));
print(id(list2));
list2[0]=1111;
print(list1);
print(list2);
'''
'''
#copy() for list-ds(we get new memory location)
#it is also cloning (new dup-obj or separate-obj)
list1 = [11,22,33,44,55];
list2 = list1.copy();
print(id(list1));
print(id(list2));
list2[0]=1111;
print(list1);
print(list2);
'''

'''
#list-concatenation(+)
#using concatenation-operator(+)
list1 = [11,22,33];
list2 = [44,55,66];
list3 = list1+list2;
print(list3);
print()
list1 = [11,22,33];
#list3 = list1+77; #TypeError
list3 = list1+[77]; #Valid
print(list3);
list3 = list1+[77,88]; #Valid
print(list3);
'''

'''
#List Repeatition(*)
list1 = [11,22,33];
list2 = list1*3;
print(list2);
list2 = 5*list1;
print(list2);
'''

'''
#Listcomparisons (relational-oper) <,>,<=,>=,==,!=
list1 = [11,22,33];
list2 = [11,22,33];
list3 = [44,22,33];
print(list1==list2);
print(list1==list3);
print(list2!=list3);
print()
```

```
list1 = ["hi","hello","welcome"];
list2 = ["hi","hello","welcome"];
list3 = ["HI","HELLO","WELCOME"];
print(list1==list2);
print(list1==list3);
print(list2!=list3);
'''

'''
#membership operator (in, not in)
list1 = [11,22,33,44,55];
print(11 in list1);
print(11 not in list1);
print(111 in list1);
print(111 not in list1);
'''

'''
#clear() -> removes all elements at a time
list1 = [11,22,33,44,55];
print(list1);
list1.clear();
print(list1); #empty-list


#Nested-List (sub-list)
#list with-in list
list1 = [11,22,33,[44,55]];
print(list1[0]);
print(list1[1]);
print(list1[2]);
print(list1[3]); #[44,55]
print(list1[3][0]);  #sub-index
print(list1[3][1]);




# 14thday 14.09-08-22

"""
==>> Tuple Collection (or) Data-Structure in Python:-
= Tuple is exactly same as List DS but it is Immutable
= Once Tuple is created we cannot perform any changes or modifications(insert/update/del) to its values

=> Tuple-Tech-Points:-
 = Tuple is coll of objects (....)
 = Tuple is Read-only version of List
 = Here Order is Preserved
 = It allows Duplicate Objects
 = It allows same-type/different-type of Objects
 = Indexes are provided to access the objects
 = It supports both +ve (F->L) [0to(n-1)] and -ve (L->F) index [-1 to -n] (both Farward/Backward)
 = Its values are represented in () with , (commas)
 = () are not compulsory (optional) but it is recommended

Ex:- (Creating Tuple)
tup1 = 11,22,33,44,55;  #w.o ()
tup1 = (11,22,33,44,55); #with ()
print(tup1);
print(type(tup1));

##Program (TupleEx1.py)
(Program to work with Tuple DS)

Ex:- (Empty-Tuple)
tup1=();
print(type(tup1));

Ex:- (Single-Value-Tuple taken as int)
tup1=(10);  #Single-Value Tuple taken as int(respective-dtype)
print(tup1);
print(type(tup1));

Ex:- (Single-Value Tuple with , comma)
tup1=(10,);  #Single-Value Tuple with , comma #10,
print(tup1);
print(type(tup1));

=> Other Tuple Examples:-
(different ways of creating a tuple)
Ex:-
tup1=();
print(tup1);
print(type(tup1));

tup1=11,22,33;
print(tup1);
print(type(tup1));

tup1=10; #int type
print(tup1);
print(type(tup1));

tup1=10,;
print(tup1);
print(type(tup1));

tup1=(10); #int type
print(tup1);
print(type(tup1));

tup1=(10,);
print(tup1);
print(type(tup1));

tup1=(10,20,30);
print(tup1);
print(type(tup1));

NOTE:-
= Tuple-Creations,
1) tup1=(); #Empty-Tuple
2) tup1=(11,); #Tuple-with Single-Value
3) tup1=11,;
4) tup1=(10,20,30); #Tuple-with Multiple-Value
5) tup1=10,20,30; #Tuple-with Multiple-Value


=> Creating Tuple with tuple():-
```

```
= here tuple() is a conversion function
= it converts other coll.of.values into tuple-type values
Ex:-
list1 = [11,22,33];
tup1 = tuple(list1);
print(tup1);

tup1 = tuple(range(0,20,2));
print(tup1);


==> Accessing Elements of a Tuple:-
= It is done with indexes or slice-operator

1) By using index:-
+ve indexes (0 to n-1) First->Last (Forward)
-ve indexes (-1 to -n) Last->First (Backward)
Ex:-
tup1 = (11,22,33,44,55);
print(tup1[0]);
print(tup1[1]);
print(tup1[2]);
print(tup1[3]);
print(tup1[4]);
print(tup1[-1]);
print(tup1[-2]);
print(tup1[-3]);
print(tup1[-4]);
print(tup1[-5]);
#print(tup1[100]); #IndexError
#print(tup1[-100]);

2) Using slice-operator:-
= it gives sub-tuple values based on indexes and stepvalue
Syn:-
 tup1[startindex:endindex:stepvalue]
= startindex/endindex/stepvalue can be +ve or -ve
Ex:-
tup1 = (11,22,33,44,55);
print(tup1[0]);
print(tup1[1:5]);
print(tup1[0::2]);
print(tup1[0:100:2]);
print(tup1[-1:-5:2]);



==> Tuple and Immutability:-
= Immutable means org-data cannot be modified
= Once tuple is created it's values cannot be changed (Immutable)
Ex:-
tup1 = (11,22,33,44,55);
print(tup1);
tup1[0]=111;


==> Mathematical Operations on Tuple:-
= Here we can apply + and * operators

1) Concatenation Operator (+):-
Ex:-
tup1 = (11,22,33);
tup2 = (44,55,66);
tup3 = tup1+tup2;
print(tup3);

2) Repetition Operator (*):-
Ex:-
tup1 = (11,22,33);
tup2 = tup1*3;
print(tup2);


==> Commonly used Functions in Tuple:-
1) len():-
= Gives No.of Elements in a Tuple
Ex:-
tup1 = (11,22,33);
print(len(tup1));

2) count():-
= Gives a particular element is repeated how many times
Ex:-
tup1 = (11,22,33,44,55,11,22,11);
print(count(11));
print(count(22));
print(count(55));

3) index():-
= Gives index-position of given element (1st occurance)
= If not there then we get "ValueError"
Ex:-
tup1 = (11,22,33,44,55,11,22,11);
print(tup1.index(11));
print(tup1.index(22));
print(tup1.index(222)); #ValueError

4) sorted()
= Sorts or Orders the element of tuple (default is ASC)
Ex:-
tup1 = (11,22,33,44,55,11,22,11);
tup2 = sorted(tup1) #default ASC order
print(tup1);
print(tuple(tup2));
=** After sorting, tuple is converted to list
(becuase tuple is immutable)

Ex:- (DESC)
tup1 = (11,22,33,44,55,11,22,11);
tup2 = sorted(tup1,reverse=True);
print(tup1);
print(tup2);
=** After sorting, tuple is converted to list

5) min() and max():-
= It gives minimum and maximum values of a list
Ex:-
tup1 = (11,22,33,44,55,11,22,11);
print(min(tup1));
print(max(tup1));
```

```
6) cmp() #outdated from py-3
= Compares elements of both tuples and gives
0 (Equal)
-1 (1st tup < 2nd tup)
+1 (1st tup > 2nd tup)
Ex:-
tup1 = (11,22,33);
tup2 = (44,55,66);
tup3 = (11,22,33);
print(cmp(t1,t2));
print(cmp(t1,t3));
print(cmp(t2,t3));

=** cmp() is in Python2 but not from Python3


==> Tuple Packing and UnPacking:-
= Tuple is created by packing group of variables
= Packing means creating a tuple with 2 or more variables
= UnPacking means getting tuple-values into 2 or more variables
Ex:-
a=11;
b=22;
c=33;
d=44;
e=55;
tup1 = a,b,c,d,e;
print(tup1);

= Tuple unpacking is reverse of packing,
(UnPacking and assign its values to different variables)
Ex:-
tup1=(11,22,33,44,55);
a,b,c,d,e=tup1;
print(a);
print(b);
print(c);
print(d);
print(e);

=** While unpacking, no.of vars and values in tuple should be same o.w we get "ValueError"
Ex:-
tup1=(11,22,33,44,55);
a,b,c=tup1; #ValueError
"""

###Program... (TupleEx1.py)
#Program to work with Tuple DS


#creating a tuple(with & w.o ()brackets)
#tup1 = 11,22,33,44,55;  #() are not-complusory(optional)
tup1 = (111,222,333,444,555);
print(tup1);
print(type(tup1));
'''

'''
#empty-tuple
tup1=();
print(tup1);
print(type(tup1));
'''

'''
#tuple with single-value(give , compulsory)
tup1=(10); #Single-Value Tuple taken as int-value data-type
print(tup1);
print(type(tup1));
#sp-case
tup1=10,;   #(10,); #Single-Value Tuple with , comma
print(tup1);
print(type(tup1));
'''

#Other Tuple Examples
'''
tup1=();    #empty-tuple
print(tup1);
print(type(tup1));
'''

'''
tup1=11,22,33; #tuple w.o ()
print(tup1);
print(type(tup1));
'''

'''
tup1=10;        #single-value(respective-dtype)
print(tup1);
print(type(tup1));
print()
tup1=10,;  #give ,(compulsory)
print(tup1);
print(type(tup1));
'''
'''
#single-value
tup1=(10);
print(tup1);
print(type(tup1));
print()
tup1=(10,);
print(tup1);
print(type(tup1));
'''
'''
tup1=(10,20,30);
print(tup1);
print(type(tup1));
'''

'''
#Creating Tuple with tuple()
list1 = [11,22,33];
tup1 = tuple(list1); #converts list to tuple
print(tup1);
tup1 = tuple(range(0,20,2)); #converts range-values to tuple
```

```
print(tup1);
tup1 = tuple("hello") #str to tuple
print(tup1)
'''

'''
#Access Elements with indexes
tup1 = (11,22,33,44,55);
print(tup1[0]);
print(tup1[1]);
print(tup1[2]);
print(tup1[3]);
print(tup1[4]);
print(tup1[-1]);
print(tup1[-2]);
print(tup1[-3]);
print(tup1[-4]);
print(tup1[-5]);
print(tup1[100]); #IndexError (out-of-index-range)
print(tup1[-100]);
'''

'''
#Using slice-operator [startindex:endindex:stepvalue]
tup1 = (11,22,33,44,55);
print(tup1[0]);
print(tup1[1:5]);
print(tup1[0::2]);
print(tup1[0:100:2]);
print(tup1[-1:-5:-2]);
'''

'''
#Tuple is Immutability
tup1 = (11,22,33,44,55);
print(tup1);
tup1[0]=111; #TypeError
print(tup1)
'''

'''
#Concatenation (using +)
tup1 = (11,22,33);
tup2 = (44,55,66);
tup3 = tup1+tup2;
print(tup3);
#Repetition (using *)
tup1 = (11,22,33);
tup2 = tup1*3; #3*tup1
print(tup2);
'''

#Tuple-Functions
'''
#len()
#tup1 = (11,22,33);
tup1 = tuple("Welcome")
print(len(tup1));
'''

'''
#count()
tup1 = (11,22,33,44,55,11,22,11);
print(tup1.count(11));
print(tup1.count(22));
print(tup1.count(55));
print(tup1.count(99));
'''

'''
#index()
tup1 = (11,22,33,44,55,11,22,11);
print(tup1.index(11));
print(tup1.index(22));
print(tup1.index(222)); #ValueError
'''

'''
#sorting (sorted())
tup1 = (11,22,33,44,55,11,22,11);
tup2 = sorted(tup1) #after sorting we get list
print(tup1);
print(tup2); #Here we get list
print(tuple(tup2)); #list to tuple()
'''

'''
#sorted(tup1, revserve=True/Fase)
tup1 = (11,22,33,44,55,11,22,11);
tup2 = sorted(tup1,reverse=True); #True means DESC-order (False(ASC))
print(tup1);
print(tuple(tup2)); #Here we get list #use tuple() to coverted it
'''

'''
#min() & max()
tup1 = (11,22,33,44,55,11,22,11);
print(min(tup1));
print(max(tup1));
'''

'''
#Tuple-packing (creating a tuple)
a=11;
b=22;
c=33;
d=44;
e=55;
tup1 = a,b,c,d,e;
print(tup1);
'''

'''
#Tuple-unpacking
tup1=(11,22,33,44,55);
a,b,c,d,e=tup1;
print(a);
print(b);
print(c);
print(d);
```

```
print(e);
'''

'''
tup1=(11,22,33,44,55);
a,b,c=tup1; #ValueError (5-values cannot be un-packed to 3-variables)



"""
==> Difference between List and Tuple:-
1)
= List values are Mutable (modified)
= Tuple values are Immutable (no-modified)

2)
= List values are given in [] (compulsory)
= Tuple values are given in () (Optional)

3)
= Go for List if group of values are need to be changed
= Go for tuple if group of values are fixed need not be changed

-----------------------------------------------------------
===>>> Set Collection or Data-Structure in Python:-
=> Tech-Points??
(Refer-notes)
 = It is coll.of Unique-values as single-unit
 = Duplicates are not allowed
 = Insertion Order is not preserved(**)
 = Sorting is possible (ASC/DESC)
 = Allows Homo/Heterogeneous Values (coll.of diff.data-type values)
 = Its values are Mutable (can be modified)
 = Represented in {......} with commas(,)
 = On Set, we can apply Union, Intersection, Difference etc mathematical operations
 = Set does not have indexes (slicing not-possible)

##Program (SetEx1.py)
(Program to work with Set DS)

=> Creating a Set:-
Ex:-
set1 = {11,22,33,44,55};
print(set1);
print(type(set1));
"""
# //Program (SetEx1.py)
# (Program to work with Set DS)

# Ex:- (Using set())
list1 = [11,22,33,44,55];
set1 = set(list1);
print(set1);
print(type(set1));

# Ex:- (Using range())
set1 = set(range(10));
print(set1);
print(type(set1));

set1 = set(range(10,20));
print(set1);
print(type(set1));

set1 = set(range(20,100,5));
print(set1);
print(type(set1));

"""
Ex:- (Empty-Set)
= It is created compulsory with set() function only but not with {}
set1 = {};
print(set1);
print(type(set1));
=** It becomes Dictionary type

set1 = set();
print(set1);
print(type(set1));
=* Now it is treated as Empty-Set


==> Commonly used Set functions:-
1) add(x):-
= Adds x to set
Ex:-
set1 = {11,22,33}
set1.add(55);
print(set1);

2) update(x,y,z):-
= Adds multiple items to set (from another-coll.of.data)
= x,y,z are not single-items but Iterable/Group of values like List,Range etc
Ex:-
set1 = {11,22,33}
list1 = [10,20,30];
set1.update(list1);
print(set1);

NOTE:-
= add() adds single-items to set
= update() adds multiple-items to set

= add() takes only 1-arg
= update() takes only multiple-arg

Ex:-
set1.add(10);  #Valid
set1.add(10,20,30); #TypeError
set1.update(10); #TypeError
set1.update(list1,range(1,10)); #Valid


3) copy():-
= It returns a duplicate-copy of the Set
= It is cloned object of Set
Ex:-
set1 = {11,22,33};
set2 = set1.copy();
print(set2);
```

```
4) pop():-
= It removes and returns any random element from Set
(Generally it remove from begin)
set1 = {11,22,33,44};
print(set1);
print(set1.pop());
print(set1);

5) remove(x):-
= It removes specified element from Set
(o.w we get "KeyError")
Ex:-
set1 = {11,22,33,44,55};
set1.remove(33);
print(set1);
set1.remove(33);

6) discard(x):-
= It removes specified element from Set
(o.w we get NO-Error)
Ex:-
set1 = {11,22,33,44,55};
set1.discard(33);
print(set1);
set1.discard(33);
print(set1);


7) clear():-
= Removes all elements in a Set
Ex:-
set1 = {11,22,33,44,55};
print(set1);
set1.clear();
print(set1);



==> Mathematical Operations on Set:-
(union/intersection/difference/symmetric_difference)
1) union():-
= set1.union(set2)
(or)
= set1 | set2
= It gives all the elements of set1 and set2 without duplicates
Ex:-
set1 = {11,22,33,44};
set2 = {33,44,55,66};
print(set1.union(set2));
print(set1|set2);

2) intersection():-
= It gives only common values from given sets without duplicates
= set1.intersection(set2)
(or)
= set1 & set2
Ex:-
set1 = {11,22,33,44};
set2 = {33,44,55,66};
print(set1.intersection(set2));
print(set1&set2);


3) difference():-
= It gives only 1st-set values which are not in 2nd-set
Ex:-
= set1.difference(set2)
(or)
= set1-set2
Ex:-
set1 = {11,22,33,44};
set2 = {33,44,55,66};
print(set1.difference(set2));
print(set1-set2);

4) symmetric_difference():-
= It gives elements from both the sets without common-elements
Syntax:-
= set1.symmetric_difference(set2)
(or)
= set1^set2
Ex:-
set1 = {11,22,33,44};
set2 = {33,44,55,66};
print(set1.symmetric_difference(set2));
print(set1^set2);


==> Membership Operators on Sets:-
(in, not in)
= in checks for data in set
= not in does not check for data in set
(True/False)
Ex:-
set1 = set("Hello Welcome");
print(set1);
print("H" in set1);
print("2" in set1);
print("2" not in set1);
"""


##Program.... (SetEx1.py)
#Program to work with Set-DS & its Operation


#creating a set using {}
set1 = {11,22,33,44,55,11,22,33};
print(set1);
print(type(set1));
'''


'''
#Using set() conversion-function
list1 = [11,22,33,44,55,11,22,33];
print(list1);
```

```
set1 = set(list1);
print(set1);
print(type(set1));
'''

'''
#set() with range()
set1 = set(range(10));
print(set1);
print(type(set1));
print()
set1 = set(range(10,20));
print(set1);
print(type(set1));
print()
set1 = set(range(20,100,5));
print(set1);
print(type(set1));
'''

'''
#Empty-Set (Sp-case)
set1 = {}; #takes as dict-obj
print(set1);
print(type(set1));
'''
'''
#sp-case
set1 = set(); #use empty-set() function
print(set1);
print(type(set1));
'''

'''
#Set-Functions
#add()
set1 = {11,22,33}
print(set1);
set1.add(55);
print(set1);
'''

'''
#update()->modify with another collection
set1 = {11,22,33}
list1 = [10,20,30];
set1.update(list1);
print(set1);
'''

'''
#update() multi-colls
set1 = {11,22,33}
list1 = [10,20,30];
set1.update(list1,range(100,110,2));
print(set1);
'''

'''
#copy() -> dup-set(cloning) sep-obj & sep-addr
set1 = {11,22,33};
set2 = set1.copy();
print(set1,id(set1))
print(set2,id(set2)); #Order is Not Preserved
set2.add(44)
print(set1)
print(set2)
'''

'''
#pop() -> del's random/first element
set1 = {11,22,33,44,55};
print(set1);
print(set1.pop());
print(set1.pop());
print(set1);
'''

'''
#remove(x) -> based on given value removed
set1 = {11,22,33,44,55};
print(set1);
set1.remove(33);
set1.remove(22);
print(set1);
#set1.remove(99); #Error is value is not-found
'''

'''
#discard(x) -> removes given value but No-Error
set1 = {11,22,33,44,55};
print(set1);
set1.discard(22);
print(set1);
set1.discard(22); #No-Error
print(set1);
'''

'''
#clear()
set1 = {11,22,33,44,55};
print(set1);
set1.clear();
print(set1); #empty-set
'''

'''
#set-mathematical-operations
#Union on Sets (union(),|)
set1 = {11,22,33,44};
set2 = {33,44,55,66};
print(set1)
print(set2)
print(set1.union(set2));
print(set1|set2);
'''

'''
#Intersection on Sets (common-elements W.O dup)
#intersection() or & operator
```

```
set1 = {11,22,33,44};
set2 = {33,44,55,66};
print(set1.intersection(set2));
print(set1&set2);
'''

'''
#Difference on Sets
#difference() or - operator
set1 = {11,22,33,44};
set2 = {33,44,55,66};
print(set1.difference(set2));
print(set2.difference(set1));
print(set1-set2);
print(set2-set1);
'''

'''
#Symmetric-Difference on Sets (un-common elements)
#symmetric_difference() or ^ operator
set1 = {11,22,33,44};
set2 = {33,44,55,66};
print(set1.symmetric_difference(set2));
print(set1^set2);


#Membership operators (in, not in)
set1 = set("Hello Welcome");
print(set1);
print("H" in set1);
print("Z" in set1);
print("Z" not in set1);
print(" " in set1);


"""
-----------------------------------------------------------
==> Frozenset Collection (or) Data-Structure(datatype):-
(Refer notes)
= It is same as that of set but it is Immutable
(Original-data cannot be modified)
= Here, we cannot use add() or remove() or pop() functions
**= Here we use frozenset() conversion-function to create a Frozen-Set

Ex:-
set1 = {10,20,30,40,50,"Sai",6.0,True};
fset1 = frozenset(set1);
print(fset1);
print(type(fset1));

##Program (FrozensetEx1.py)
(#Program to work with datatypes)

=> We can use loop to access display frozenset values
Ex:-
for x in fset1 : print(x);

=> Size is non-dynamic(FIXED) & Immutable (Org.data CANNOT be modified)
Ex:-
fset1.add(60);  #Error
fset1.remove(44); #Error
"""


##Program (FrozensetEx1.py)
#Program to work with FrozenSet Data-Structure(datatype)

#create FrozenSet using frozenset()
set1 = {10,20,30,40,50,"Sai",6.0,True,10,20};
fset1 = frozenset(set1);
print(fset1);
print(type(fset1));

#fset1.add(80);  #it is immutable (Error)
#fset1.remove(40);
#fset1.pop();


fset1 = frozenset({11,22,33,44,55,11});
print(fset1);
print(type(fset1))
#loops
for x in fset1:
 print(x);


"Assignment"
##Program (FrozensetEx2.py)
##WAP to perform all operations of set same on frozenset
# Hint:-
# use SetEx1.py (replace set() with frozenset())
# (set1 replace fset1 var)




# 15th day 15.10-08-22


"""
===> Dictionary Collectioion (or) Data-Structure in Python:-
= List, Tuple, Set, FrozenSet collections have single-single values
= However, in Dict-coll, we have group of (Key,Value) pairs
Ex:-
 "rollno":1001  Key:Value
 "name":"Sai"
 "height":6.0
 etc...
 "addr":"Hyd"
 "phno":9988776655

=> Dict-Imp-Points:-
(Refer-notes)
 = Keys are unique, cannot be duplicate (Unique)
 = Values can be duplicated
 = Allows Homo/Heterogeneous (Keys,Values) objs (same-data or diff-data)
 = Order is not Preserved
 = Dictionary is Mutable-coll (Org-data is Modifiable)
 = NO-Indexes and Slicing is Not applicable
```

```
##Program (DictionaryEx1.py)
(Program to work with Dictionary DS)

==> How to create Dictionary?
= To represent dictionary DS we use {} curly-brackets
Ex:- {Key1:Value1, Key2:Value2, Key3:Value3,.......}

Ex:- (Empty-Dictionary)
dict1={};
dict2=dict();

Ex:- (Adding Entries)
Syntax:-
 dict1[Key]=Value;

dict1[1001]="Sai";
dict1[1002]="Ram";
dict1[1003]="Ali";
print(dict1);

//Program (DictionaryEx1.py)
(Program to work with Dictionary DS)

Ex:- (Dictionary with K,V pairs)
Syn:-
 dict1 = {K:V, K:V, K:V,....};
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};

Ex:- (Accessing-Data)
Syn:-
 dict1[key] ---> we get corresponding value
dict1[1001]
dict1[1002]
dict1[1003]
#print(dict1[1004]);  #KeyError

= If specified Key is not-found then we get "KeyError"

Ex:- (has_key())
= It checks whether Key is there or not in Dictionary
= It gives 1(Found) o.w 0(Not-Found)
= It is in Python-2 but not in Python-3
= Alternate is "in" operator
if 1004 in dict1:
 print(dict[1004]);
else
 print("Key Not Found");
"""

##Program (DictionaryEx1.py)
#Program to work with Dictionary DS & its Operations


#Empty-Dictionary
dict1={};       #Empty {} curly brackets
print(dict1);
print(type(dict1));
dict2=dict();   #Empty dict() func
print(dict2);
print(type(dict2));
'''

'''
#Adding Entries dict1[key]=value; (use [] key-oper)
dict1 = {}
dict1[1001]="Sai";
dict1[1002]="Ram";
dict1[1003]="Ali";
print(dict1);
print()
dict2 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict2);
'''

'''
#Accessing-Dict-Data(using [] key-oper)
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1[1001]);
print(dict1[1002]);
print(dict1[1003]);
#print(dict1[1004]);  #KeyError


#dict with membership-oper(in, not in)
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
#if 1004 in dict1:
if 1002 in dict1:
 print(dict1[1002]);
else:
 print("Key(1004) Not Found");

#dict with loops
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali",1004:"John"};
for key in dict1:     #assigns only keys
    print(dict1[key])

"""
NOTE:-
= The best-way to access dict-data is with loops is the best-way
Ex:-
for var in dict1: --->each-key is assigned to loop-var
 dict1[var]---->value
##Program (DictionaryEx1.py)



"Assignment"
#WAP to enter student-name(key) and Percentage-Marks(value) & insert in dict1
//Program (DictionaryEx2.py)
(Program to Enter Student-Name and Percentage-Marks for Display)

#Program to Enter Student-Name and Percentage-Marks for Display in Dict-obj
'''
records = {}; #empty-dict
n = int(input("Enter No.of Students : "));
i=1;
while i<=n:
 name=input("Enter Student Name : ");
 percentage=input("Enter Marks % : ");
```

```
 records[name]=percentage;
 i=i+1;

print("Student-Name","\t\t","% of Marks");
for x in records: #here dict-key is assigned to x
 print("\t",x,"\t\t",records[x]);

print(records);
'''




"Operations on dict-coll"
==> Dictionary Updates:-
(Updating data in a dictionary)

##Program (DictionaryEx3.py)
(Program to work with Dictionary DS)

Ex:-
 dict1[Key]=Value or new-value;
= If Key is not-available then New-Entry is Added
= If Key is available then Old-Value is replaced by New-Value
Ex:-
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
dict1[1004]="Tom";  #add-new K:V
print(dict1);
dict1[1001]="Baba";  #replace
print(dict1);

//Program (DictionaryEx3.py)
(Program to work with Dictionary DS)

=> Deleting Elements:-
= del keyword is used here

1)del dict1[Key]
= It deletes corresponding (K,V) pair o.w "KeyError"(Not-Found)
Ex:-
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
del dict1[1003];
print(dict1);
del dict1[1003];

2)dict1.clear()
= Removes all entries from dictionary
Ex:-
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
dict1.clear();
print(dict1);
dict1.clear(); #No-Error

3)del dict1
= deletes total dictionary object including its reference
Ex:-
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
del dict1;
print(dict1); #NameError
"""

##Program (DictionaryEx3.py)
##Program (DictionaryEx3.py)
#Program to work with Dictionary DS operations


#dict-Updates (updating existing value)
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
#key[] oper
dict1[1004]="Tom"; #add new K:V if not-there
print(dict1);
dict1[1001]="Baba"; #updated/replaced "Sai" with "Baba"
print(dict1);
'''

'''
#Deletes (single-entry)
#del stmt (with []key-oper)
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
del dict1[1003];
print(dict1);
#del dict1[1003]; #KeyError if not-there
'''

'''
#clear()
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
dict1.clear();
print(dict1);       #empty-dict
dict1.clear();     #No-Error
'''

'''
#delete complete dict (del stmt)
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
del dict1;
#print(dict1); #NameError(dict-obj-ref is deleted from memory)
'''

'''
#Dictionary-Functions
#dict() conversion-func
dict1 = dict();
print(dict1);
dict2 = dict({1001:"Sai",1002:"Ram",1003:"Ali"});
print(dict2);
dict3 = dict([(1001,"Sai"),(1002,"Ram"),(1003,"Ali")]); #list(k,v)->dict (list of tuples(k,v)) [(k,v),(k,v),(k,v)]
print(dict3);
'''

'''
#len()
```

```
dict1 = dict();
dict2 = dict({1001:"Sai",1002:"Ram"});
dict3 = dict([(1001,"Sai"),(1002,"Ram"),(1003,"Ali")]);
print(len(dict1));
print(len(dict2));
print(len(dict3));
'''


'''
#using get() we get values w.o KeyError
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1.get(1001));
print(dict1.get(1002));
print(dict1.get(1003));
print(dict1.get(1004)); #None but no-error
print(dict1.get(1004,"NOT-THERE"));
'''
'''
#using [] subscript-operator
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1[1001]);
print(dict1[1002]);
print(dict1[1004]);  #KeyError
'''

'''
#pop() -> deletes given key-value pair
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
print(dict1.pop(1001));
print(dict1.pop(1003));
print(dict1);
print(dict1.pop(1001));  #KeyError
'''

'''
#popitem() -> dels random key-value pair
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
print(dict1.popitem());
print(dict1);
print(dict1.popitem());
print(dict1);
print(dict1.popitem());
print(dict1);
print(dict1.popitem());  #KeyError (Empty-Dictionary)
'''

'''
#keys() -> we get list of keys[]
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1.keys());
for k in dict1.keys():
 print(k);
'''

'''
#values() -> we get list of values[]
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1.values());
for v in dict1.values():
 print(v);
'''

'''
#(***)
#items() -> we get list-of-key,value pairs as ()tuples
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1.items());
#**for loop with 2-vars(k,v)
for k,v in dict1.items():
 print(k,"--->",v);
'''

'''
#copy() cloning(dup-dict-coll, sep-coll with sep-addr)
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
dict2 = dict1.copy();
print(dict1,id(dict1));
print(dict2,id(dict2));
'''

'''
#setdefault()->adds new (k,v) o.w gives existing value
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
print(dict1.setdefault(1004,"Tom")); #adds to dict
print(dict1);
print(dict1.setdefault(1004,"Tommy")); #gives the value
print(dict1);


#update() #if entry is not-there then it will add o.w update
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1);
dict1.update({1004:"Tom"}); #not-there it adds
print(dict1);
dict1.update({1004:"Tommy",1005:"Sai"}); #1004(updated)
print(dict1);       #1005 is newly-added



"""
==> Functions in Dictionary DS:-
1) dict()
= It creates a Empty-Dictionary/Dictionary with Values also
Ex:-
dict1 = dict();  #Creates Empty-Dictionary
dict2 = dict({1001:"Sai",1002:"Ram",1003:"Ali"});
    #Creates Dictionary with Specified-Elements
dict3 = dict([(1001,"Sai"),(1002,"Ram"),(1003,"Ali")]);
  #Creates Dictionary with Specified-Tuple-Values

2) len():-
= Gives length of dictionary
Ex:-
len(dict1);
```

```
3) clear():-
= Removes all elements in dictionary (empty)
Ex:-
dict1.clear();

4) get(key):-
= Gives value for particular key o.w None(Not-Found)
Ex:-
dict1.get(1001);

=>get(key,defaultValue):-
= Gives value for particular key o.w defaultValue(Not-Found)
Ex:-
dict1.get(1004,"NOT-THERE");

Ex:- (Other)
print(dict1[1001]);
#print(dict1[1004]);  #KeyError
print(dict1.get(1001));
print(dict1.get(1004));  #None
print(dict1.get(1001,"NOT-THERE"));
print(dict1.get(1004,"NOT-THERE"));


5) pop():-
Ex:-
dict1.pop(Key);
= It removes corresponding (K,V) pair and returns its value o.w "KeyError"

dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1.pop(1001));
print(dict1);
print(dict1.pop(1001));

6) popitem():-
= Removes any random (K,V) pair from dictionary and returns it
Ex:-
dict1 = {1001:"Sai",1002:"Ram",1003:"Ali"};
print(dict1.popitem());
print(dict1);
print(dict1.popitem());
print(dict1);
print(dict1.popitem());
print(dict1);
print(dict1.popitem());  #KeyError (Empty-Dictionary)


==> Other functions:-
7) keys():-
= Returns all keys at a time
Ex:-
dict1.keys()

8) values():-
= Returns all values at a time
Ex:-
dict1.values()

9) items():-
= It returns list-of-tuples representing (K,V) pairs
Ex:- [(k,v), (k,v), (k,v)]
print(dict1.items());

10) copy()
= Creates a duplicate copy(cloned copy)
Ex:-
dict2 = dict1.copy();

11) setdefault()
Ex:-
 dict1.setdefault(k,v);
= If key is already there then it returns corresponding value
= If key is not there then new (K,V) entry is added to dictionary
print(dict1.setdefault(1004,"Tom"));

12) update():-
Ex:-
dict1.update(x); #x is {k:v,k:v,....}
= x will be updated in dictionary o.w new-entry is added
"""

"Assignment"
##Program (DictionaryEx4.py)
#Program to accept Dictionary vals(nums) and print its SUM
# Input: {"sub1":100,"sub2":92,"sub3":65,"sub4":75,"sub5":68}

##Program to accept Dictionary values(K:V) and print its SUM

dict1 = eval(input("Enter Dictionary :"));
sum1 = sum(dict1.values());
print("SUM :",sum1);


'''
-----------------------------------------------------------
==> Functions in Python:-
= A Function is indented-block-of-code, which is used to perform specific-task in a program (task/work)
= Advantage is write-once & call or use any no.of times (Code-Reusability)


= Basically, Functions are divided into 2-types,
 a) Built-in Functions
 b) User-defined Functions
 ==Diagram==

1) Built-in Functions:-
= They are available in Python-Library-Modules
(Modules means .py python-files only)
Ex:-
 id()
 type()
 input()
 eval()
 etc...

2) User-defined Functions:-
= These functions are developed by programmer/developer as per user-requirements in program

Syntax:-(***) [def is the keyword]
-----------------------------------------------
```

```
 def function_name(parameters,....): #input-values/args
  """doc-string/comment"""
  ....
  stmts; #(Body/Indented-block-of-code)
  ....
  return value;
-------------------------------------------------
(5-things in function::)
function_name/parameters/comments/body/return-value
-------------------------------------------------

NOTE:-
= def, return are 2-keywords used in function definition
= function-body is intended-block-of-stmts in multiple-lines till return value
'''
# Ex:-
#Program (FunctionEx1.py)
#(Program to define a function and call it)

#func with no-args(input) & no-return-value
def f1():
 """f1() def"""
 print("Hello User");
 print("Welcome to Python");
 print("All the best");

#use or call the function (any no.of.times) re-usability
f1();
print()
f1();
print()
f1()
print()
for i in range(10):
    f1();


# NOTE:-
# = After def. function, use or call the function

# 16th day 16.11-08-22

"""
==> Parameters to Function:-
(Function with Paramters)
(Function with Paramters)
= Parameters means input-values given to a function
= Such input-values are passed to function while calling a function

Ex:- (FunctionEx1.py)
(Program to define a function with params and call it)
def f2(uname):
 print("Hello User :",uname+", Welcome to Python...");
f2("Sai");
f2("Ram");
f2("Ali");
"""
# //Program (FunctionEx1.py)
# (Program to define a function with param as number and print is square)
def f3(num):
 print("Square of given :",num+" is :"+num*num);
f3(4);
f3(5);
f3(6);

#Program (FunctionEx1.py)
#(Program to define a function and call it)


#func with no-args(input) & no-return-value
def f1():
 # ''''''f1() def''''''
 print("Hello User");
 print("Welcome to Python");
 print("All the best");

#use or call the function (any no.of.times) re-usability
f1();
print()
f1();
print()
f1()
print()
for i in range(10):
    f1();
'''

'''
#func with input-para(args)
def f2(uname):  #uname='Sai'....
 # ''''''f2() func def.''''''
 print("Hello User :",uname);
 print("Welcome to Python...");

f2("Sai");  #based on input-value output also changes
f2("Ram");
f2("Ali");


#func with input-para
def f3(num): #num=4,5,6
 #'''f2() func def'''
 print("Square of given :",num," is :",num*num);

f3(4); #input-values can be used inside func-body
f3(5);
f3(6);

"""
-------------------------------------------------------
==> return statement in Functions:-
(Func with return-stmt)
= Function takes input-values, executes-the-logic and can provide return value/statement
Ex:-
 return sum;

##Program (FunctionEx2.py)
##(Program to define a function with return value)
= Accept 2 nums and return its sum-value
```

```
NOTE:-
= To call a function, we can pass values or variables
= We can call function in another function as parameter (return value of 1-function is input-para to another function)

Ex:- (without return stmt)
def f1():
 print("Hello User");
f1();
print(f1());

=** Here since f1() does not have return-value, it prints "None" in 2nd case


//Program (FunctionEx2.py)
(Program to check given num is Even or Odd)


==> return multiple-values:-
= This is special-feature(added) in Python-Functions, not available in C,C++,Java,.net languages

//Program (FunctionEx2.py)
(Program to return multiple-values in func)

//Program (FunctionEx2.py)
(Program to return multiple-values in func using arithmetic-oper)
"""
##Program (FunctionEx2.py)
#Program to define a function with return value
##Program (FunctionEx2.py)
##(Program to define a function with return value)


#Accept 2 nums as input and return its sum-value
def sum(a,b): #a=11,b=22
 # ''''''sum() func def''''''
 c=a+b
 return c;

#case1
x=11;
y=22;
result = sum(x,y); ##c=33
print("SUM1 :",result);
result = sum(111,222);
print("SUM2 :",result);
print("SUM3 :",sum(1111,2222)); #sp-case
#using or calling a func inside print() as para
'''

'''
#Without return-stmt
#func by default returns None
def f1():    #no-input-para
 print("Hello User");
 #no-return-value
f1();
print()
print(f1()); #prints None
print("End of Program");


#sp-case
#func with multiple-return-values(tuple)
#Multiple-return values(Arithmetic-Opertions)
def calci(a,b):
 sum=a+b;
 sub=a-b;
 prod=a*b;
 div=a/b;
 mod=a%b;
 return (sum,sub,prod,div,mod);
tup1 = calci(11,3);
print(tup1)
print(type(tup1))
#use loop
print("Result :");
for x in tup1:
 print(x);

"""
-----------------------------------------------------------
==> Types of Function-Arguments:- (paramters) i/p-values
Ex:-
def f1(a,b): #a,b -> Formal-Args/Para/input-values
 ...
 stmts;
 ...
f1(x,y); #x,y -> Actual-Args/Para/input-values

= Here x,y are Actual-Args and a,b are Formal-Args

= Pythons supports 4-types of Actual-Args, they are:-
1) Positional Args
2) Keyword Args
3) Default Args
4) Variable-Length Args


1) Positional Args:-
= Arguments passed to a function in proper-order or particular-order is called as Positional-Args
Ex:-
def sub(a,b):
 print(a-b);
x=11;
y=22;
sub(x,y);
sub(y,x);

= Here order/position/number of args will give appropriate result

##Program (FunctionEx3.py)
(Program to demo Types of Func-Args)

2) Keyword-Arguments:-
= We can pass args using keyword i.e, paramter name (formal-args)
Ex:-
def f1(uname,msg):
 print("Hello :",uname,msg);
f1(uname="Sai",msg="Thank You!!");
f1(msg="All the Best",uname="Ram");
```

```
=** Here we pass actual-args using formal-args name & value in any order separated with ,(comma)

//Program (FunctionEx3.py)
(Program to demo Types of Func-Args)

Ex:- (both positional and keyword args)
= Here 1st positional-args and then followed by keyword-args is mandatory
def f1(uname,msg):
 print("Hello :",uname,msg);
f1("Sai","Thank You!!");  #Valid
f1("Ram",msg="Thank You!!"); #Valid
f1(name="Ram","All the Best"); #InValid (SyntaxError)



3) Default-Arguments:-
= We can have default values to formal-args
= Such values are used when we do not pass actual-args
= It is used for positional-args
Ex:-
def sum(a=11,b=3):
 print("SUM :",(a+b));
sum(10,20);
sum();

//Program (FunctionEx3.py)
(Program to demo Types of Func-Args)

NOTE:-
= After default-args, we should not take non-default args in func-definition
Ex:-
def sum(a=11,b=3): #Valid
def sum(a=11,b): #In-Valid (SyntaxError)
def sum(a,b=3):  #Valid



4) Variable-Length Arguments:-
= We can pass any no.of args to call a function
i.e, 0 or 1 or more actual-args to call a function
**= Such functions formal-args are declared using * symbol
Ex:-
def f1(*num):
 total=0;
 for x in num:
  total=total+x;
 print("SUM :",total);
f1();
f1(11);
f1(11,22);
f1(11,22,33);
= Such values are represented as Tuple internally

//Program (FunctionEx3.py)
(Program to demo Types of Func-Args)


**NOTE:-
= We can declare Keyword Variable-Length-Args as follows,
= For this we use ** symbol
Ex:-
def f1(**nums):
= To call such function, we pass any no.of keyword-args
= Internally it is treated as Dictionary (K,V)
Ex:-
def f1(**nums):
 for k,v in nums.items():
  print(k,"=",v);
f1(a=10,b=20,c=30);
f1(rollno=1001,name="Sai",course="CSE");
"""

##Program (FunctionEx3.py)
#Program to demo Types of Func-Args
##Program (FunctionEx3.py)
#(Program to demo Types of Func-Args)


#Positional Args (position of input-values)
def sub(a,b):
 print(a-b);
x=11;
y=22;
sub(x,y); #-11
sub(y,x); #11 #based on postion of args output also changes
sub(x,x); #0
sub(y,y); #0
'''

'''
#Keyword-Arguments (input-para-names is used to pass args)
#here input-para-names are keyword-names
def f1(uname,msg):
 print("Hello :",uname,msg);
f1(uname="Sai",msg="Thank You!!");
f1(msg="All the Best",uname="Ram");
#here we can change order of input-para-names(keyword-names)
'''

'''
#Default-Arguments
#default-values to input-para
def sum(a=1,b=2,c=3): #here a,b,c have default values
 print("SUM :",(a+b+c));

sum(10,20,30);
sum(10,20);  #c(3rd-para) is missing
sum(10)  #b,c(2nd,3rd-para) is missing
sum();  #a,b,c(1st,2nd,3rd all are missing)
'''

'''
#Variable-Length Arguments
#sp-arg given with *varname(tuple)
def f1(*num): #here *num(tuple) can accept 0/1/more values
 print(num,type(num))
 print("SUM :",sum(num));

f1(11,22,33); #3-values(or more)
```

```
fl(11,22);
fl(11);
fl();
#list1 = [11,22,33,44,55]; #TypeError
#fl(list1);


#sp-case for var-len-args
#Keywords with Variable-Length-Args (**varname) dict{}
def fl(**nums):
 print(nums)
 print(type(nums))
 print();

fl(a=10,b=20,c=30,d=40); #keyword-args(our-own)
fl(rollno=1001,name="Sai",course="CSE");

"""
-----------------------------------------------------------------
**==> Functions Types of Variables:-
= Python supports 2 types of variables wrt functions
1) Global variables
2) Local variables

1) Global variables:-
= Variables declared outside a function
= They are accessible in all functions of that Module(.py file)
Ex:-
a=10;
def f1():
 print(a);
def f2():
 print(a);

f1();
f2();
"""
##Program (FunctionEx4.py)
##(Program to work with Function Variables)
#Program to work with Function Variables


#Global-Variables (def directly in prog outside func)
a=10; #global-var-a
def f1():
 print(a); #using inside f1()
def f2():
 print(a); #using inside f2()
f1();
f2();
'''

'''
#Local-Variables (Def. inside particluar func
def f1():
 a=11;  #local-var-a (local-access)
 print(a);
 #print(b)
def f2():
 b=22  #local-var-b (local-access)
 print(b);
 print(a)


f1();
f2();
#print(a,b)  #NameError
'''


'''
#Both Global & Local-vars with same name
a=100; #global-var-a(100)
def f1():
 a=11; #local-var-a(11) 1st preference to local-var
 print(a);
def f2():
 print(a); #It takes Global-Var reference

f1();
f2();
'''

'''
#global-Keyword (Local-var and Global-Var with same-name)
#global-Keyword inside a func for modifications
a=100;
def f1():
 global a;
 a=1000;  #It takes Global-Var reference
 print(a);
def f2():
 print(a);  #It takes Global-Var reference

#case1
#f1();
#f2();

#case2
f2();
f1();
f2()



#globals() -> gives all global-vars as dict{}
#Global-Var & Local-var with same-name and access both at a time
a=100; #global-vars
b=200
c=300
def f1():
 a=10; #local-vars with same-name as global-vars
 b=20
 c=30
 print(a,b,c);
 dict1 = globals()
 print(dict1["a"],dict1["b"],dict1["c"]);

f1();
```

```
"""
2) Local-Variables:-
= They are declared inside the body of a function
(indented block-of-code)
= They are accessible only inside that function-body itself but not outside the function
Ex:-
def f1():
 a=11;
 print(a);
#def f2():
# print(a);

f1();
#f2();

# //Program (FunctionEx4.py)
# (Program to work with Function Variables)

# Ex:- (Both Global & Local-vars with same name)
a=100;
def f1():
 a=11;
 print(a);
def f2():
 print(a);  #It takes Global-Var reference

f1();
f2();


==> Global-Keyword:-
= It is used in 2-ways
1) To declare global-var inside a function
2) To make global-var available to a function for modification
(provided local-var also have same-name)

Ex:- (Local-var and Global-Var with same-name)
a=100;
def f1():
 a=11;
 print(a);
def f2():
 print(a);  #It takes Global-Var reference

f1();
f2();

//Program (FunctionEx4.py)
(Program to work with Function Variables)

Ex:- (with global-keyword & modifications)
"""
a=100;
def f1():
 a=11;
 print(a);
 global a;
 a=1000;  #It takes Global-Var reference
def f2():
 print(a);  #It takes Global-Var reference

f1();
f2();

# Ex:- (global-Keyword inside func for global-var-declaration)
def f1():
 global a;
 a=100;  #It takes Global-Var reference
 print(a);
def f2():
 print(a);  #It takes Global-Var reference

f1();
f2();


# Ex:- (Global-Var & Local-var with same-name and access both at a time)
a=100;
def f1():
 a=10;
 print(a);
 print(globals()['a']);
f1();

# =** globals() is used here to access global-var inside a function with same name along with local-var



# 17th day 17.12-08-22

"""
==>> Lambda Functions::-
= they are anonymous funcs (func w.o names)
= they are defined with lambda keyword
= they take input-values(args/para) & directly return a value
=*** def, return keywords are not required
=* they are specially used for instantly used funcs
(use then & there only)
Ex:-
==Diagram==
regular-func v/s lambda-func
=** Advantage is,
 = func code-lines is reduced
NOTE:-
= in python, everything is object, even func is also one-object with func.ref.var(f1,f2,sqr,ss)
"""

##Program...(FunctionEx5.py)
#Program for Lambda-functions(Anonymous-Funcs)


#Prog for square of num using Lamda-Func
ss = lambda n: n*n;
print("Square of 5 :",ss(5));
print("Square of 9 :",ss(9));
'''

'''
#Prog for sum of 2 nums using Lamda-Func
```

```
ss = lambda a,b : a+b;
print("Sum of 10,20 :",ss(10,20));
print("Sum of 11,22 :",ss(11,22));
'''

'''
#Prog for bigger of 2 nums using Lamda-Func
ss = lambda a,b : a if a>b else b;
print("Bigger of 10,20 :",ss(10,20));
print("Bigger of 11,22 :",ss(111,99));
'''

#sp-cases of lambda-func
'''
#Prog to filter only even-nums from List using filter()
#filters the coll.of.data Ex:- filter(func,coll)->T(accept)/F(reject)
#With Lambda-Func
list1 = [11,22,33,44,55,66];
list2 = list(filter(lambda x : x%2==0,list1));
print(list2);
list2 = list(filter(lambda x : x%2!=0,list1));
print(list2);
'''

'''
#Prog to generate sqr-nums from List using map()
#map(func,coll)
#With Lambda-Func
list1 = [1,2,3,4,5];
list2 = list(map(lambda x : x*x,list1));
print(list2);

#map() with multiple-lists
list1=[1,2,3,4,10];
list2=[1,2,3,4,10];
list3 = list(map(lambda x,y:x*y,list1,list2));
print(list3);
'''

'''
#reduce(func,coll) ---> functools.py module(import it)
from functools import *;
list1 = [11,22,33,44,55];
result = reduce(lambda x,y:x+y, list1);
print(result);
result = reduce(lambda x,y:x*y, list1);
print(result);



#Sum of N-Nums
from functools import *;
result = reduce(lambda x,y:x+y, range(1,101));
print(result);


filter() #---> we get less no.of.values
map() #---> we get equal no.of.values
reduce() #---> we get single values
# (all takes lambda-func)


"""
==> Lamda functions are commonly used with filter(), map() and reduce() functions:-
= These funcs take function as argument/para
(here we pass lamda function as input-para to given functions)

1) filter():-
= It is used to filter-values from given-sequence of values based on some condition
Syn:-
 filter(function,sequence)
= function as arg, performs conditional-check
= sequence is List/Tuple/String/Rangeofvalues

//Program (FunctionEx5.py)
(Prog to filter only even-nums from List using filter())


2) Map():-
= For every-element in given sequence, apply some functionality and generate new-element (with some modifications)
Ex:-
= For every-element in (1 to 10), generate squares
 map(function,sequence)

//Program (FunctionEx5.py)
(Prog to generate sqr-nums from List using map())

NOTE:-
= map() can be applied on multiple-lists also but both should have same-length
Ex:-
list1=[1,2,3,4];
list2=[1,2,3,4];
list3 = list(map(lambda x,y:x*y,list1,list2));
print(list3);

=** x is taken from list1 and y is taken from list2

3) reduce():-
= It reduces sequence of elements into single-element by applying given function as arg
= Syn:
 reduce(function,sequence)

= This function is in "functools" module, hence import it.
Ex:-
from functools import *;
list1 = [11,22,33,44,55,66];
result = reduce(lambda x,y:x+y, list1);
print(result);

//Program (FunctionEx5.py)

Ex:-
result = reduce(lambda x,y:x*y, list1);
print(result);

Ex:-
from functools import *;
result = reduce(lambda x,y:x+y, range(1,101));
print(result);
```

```
NOTE:-
= In import *, * indicates all-functions


===> Function Aliasing(Referencing)::-
==> Function as Object:-
= In python, everything is an object
= Even functions are also objects
= Every function-name has unique reference/address

Ex:-
def f1():
 print("Hello World");
print(f1);
print(id(f1));

//Program (FunctionEx6.py)
(Every function is an object)


=> Function Aliasing:-
= For existing function, we can give another name (alias)
= alias means alternate-name or another-name
Ex:-
def f1(name):
 print("Hello :",name);

f2=f1;
f1("Sai");
f2("Ali")
print(f1);
print(f2);
print(id(f1));
print(id(f2));

=** Here we have only 1-function, it can be called with f1 or f2 names
= If we delete 1-name still we can access that func with alias name

Ex:- (Function Ref-deletion)
def f1(name):
 print("Hello :",name);

f2=f1;
f1("Sai");
f2("Ali")
del f1;
f1("Sai");
f2("Ali")



==> Nested Functions:-
= Defining one-func inside another func is called Nested-Functions
Ex:-
"""
def f1():
 print("f1() is Outer-Func");
 def f2():
  print("f2() is Inner");
 print("f1 is calling f2-Func");
 f2();
f1();
#f2(); #NameError, f2 is not-defined

##Program (FunctionEx6.py)
# (Nested-Functions)


##Program (FunctionEx6.py)
##Program to work with functions


#Every-function is an object
def f1():
 print("Hello World");
f1();
print(f1);
print(id(f1));
'''
'''
#function-aliasing(another-name)
#same-fun-def and diff-names
def f1(uname):
 print("Hello :",uname);

f1("Sai");
f2=f1;  #f1----->(def.....)<-------f2
f2("Ali")
print(f1);
print(f2);
print(id(f1));
print(id(f2));
'''

'''
#sp-case
#function-ref.var-deletion
def f1(uname):
 print("Hello :",uname);

f2=f1;  #f1----->[def....]<-----f2
f1("Sai");
f2("Ali")
del f1;  #      [def....]<-----f2
#f1("Sai");
f2("Ali")


#Nested-Functions (func with-in func) local-func(local-access)
def f1():
 print("f1() is Outer-Func");
 def f2():
  print("f2() is Inner-Func");
 print("f1 is calling f2");
 f2();

f1();
#f2(); #NameError
```

```
"""
============================================================
==> Date & Time in Python:-
= Python provides different ways to handle Date and Time
= Python provides "time" and "calendar" and "datetime" modules (.py files)
(we have to use this modules and work with date&time)
Ex:-
 import time;
 import calendar;
 import datetime;

(BASICS)
=> What is Tick?
(representing date&time in seconds/ticks)
= Date and Time are taken as float numbers (in seconds)
= Particular Date & Time is taken in seconds since "Jan 1st, 1970 00:00:00 AM"

==> Current System Date & Time?
= time module with time() gives current system date&time (in Ticks)
Ex:-
import time;
sysdatetime = time.time();
print(sysdatetime);

##Program (DateTimeEx1.py)
#Program to work with Date & Time


==> Getting Current-Time:-
= time-module
= For this we use localtime()
Ex:-
import time;
currenttime = time.localtime(time.time());
print(currenttime);

=** for localtime(), pass Tick-time as parameter
= Tick-time seconds is converted to TimeTuple(9 values) Ex:-struct-time()

=> Getting Formatted-Time:-
(Proper data & time in understandable format)
= asctime() gives Date and Time in simple format
Ex:-
import time;
formattedtime = time.asctime(time.localtime(time.time()));
print(formattedtime);

NOTE:-
= 3 methods to work with date & time
time()--->localtime()---->asctime()



=> Getting Calendar of Month:-
= "calendar" module gives monthly and yearly calendars
"month()"
Ex:-
import calendar;
cal = calendar.month(2020,12);
print(cal);

=* 1st parameter is Year
=* 2nd parameter is Month(1-12)



==> time module:-
= time module provides different functions and attributes(variables) to work with time-representations

1) time.altzone:-
= It is attribute, which gives Local DST timezone (offset) in seconds
Ex:-
import time;
print(time.altzone);

2) time.asctime():-
= This method gives date&time as "Thu Nov 14 20:00:09 2019" format
= It converts TimeTuple or struct_time to Local-Time
Ex:-
import time;
datetime = time.localtime();
print(time.asctime(datetime));

3) time.clock():- (NA)
= It gives seconds elapsed since 1st time call of function
Ex:-
import time;
print(time.clock());
time.sleep(5.5); #sleeps for 5.5 seconds
print(time.clock());

4) time.ctime():-
= It gives Local-Time by converting seconds elapsed from Jan 1,1970 00:00:00
Ex:-
import time;
datetime = time.ctime();
print(datetime);
=** no-need to use time(), localtime(), asctime()

5) time.gmtime(sec) (not-req)
= It gives struct_time values for given seconds
= If no para is given then current sysdatetime is taken
Ex:-
import time;
print(time.gmtime());
print(time.gmtime(300));

6) time.localtime(sec)
= It gives TimeTuple values for given seconds
= If no para is given then current sysdatetime is taken
Ex:-
import time;
print(time.localtime());
print(time.gmtime(300));


(Other-Methods)
7) time.mktime()
= It gives seconds elapsed since Jan 1st, 1970 00:00:00 for given localtime or TimeTuple
```

```
Ex:-
"""
import time;
timetup = (2009, 2, 17, 17, 3, 38, 1, 48, 0);
secs = time.mktime(timetup );
print(secs);
print(time.asctime(time.localtime(secs)))
"""
8) time.sleep(sec)
= It makes the program execution sleep for given no.of seconds(int/float)
Ex:-
import time;
print(time.ctime());
time.sleep(5);
print(time.ctime());

9) strftime() -> (NReq)
= Provides date&time in required formatted string (using format-specifiers %char)

10) strptime() -> (NReq)
= Takes formatted-string date&time using format-specifiers and gives struct_time

11) time.time()
= Gives date&time as seconds elapsed from Jan 1st, 1970 00:00:00
Ex:-
import time;
print(time.time())
print(time.asctime(time.localtime(time.time())));
-----------------------------------------------------
12) time.timezone  (time zone offset in seconds)
13) time.tzname  (local time zone name)


==> calendar module:-
= calendar module provides different functions to work with calendars
= By default it takes Monday as 1st day of week (Mon-Sun as 0-6)

Ex:-
calendar(YEAR,width b/w date,lines b/w weeks,charspaces b/w cal)
Ex:-
"""
import calendar
print(calendar.calendar(2021,2,1,6))

print(calendar.firstweekday()); #Mon-0
print(calendar.isleap(2020)); #True or False

print(calendar.leapdays(2020,2030)); #Leap days b/w 2 years

# print(calendar.calendar(Year,Month,))#width b/w dates,lines b/w weeks)
print(calendar.month(2020,11,2,1));

# print(calendar.monthcalendar(Year,Month));
print(calendar.monthcalendar(2020,11)); #Nested Lists with weeks

print(calendar.monthrange(2020,11)); #1st date weekday & No.ofdays

calendar.prcal(2020,2,1,6); #Prints Year Calendar
calendar.prmonth(2020,11,2,1); #Prints Year Calendar
calendar.setfirstweekday(6); #Mon-Sun(0-6)

print(calendar.weekday(2020,11,14)); #gives weekday-code(0-6 mon-sun)


"""
==>#datetime module
= In this module, we have datetime-class
= In that class, we have now() method/function
= now() method gives current system date&time
Ex:-
import datetime;
date1 = datetime.datetime.now();
print(date1);
"""


##Program (DateTimeEx1.py)
#Program to work with Date & Time


#time()-->time-module
import time;
sysdatetime = time.time();
print(sysdatetime);
'''

'''
#localtime()
import time;
currenttime = time.localtime(time.time());
print(currenttime);
#it gives struct_time as time-tuple with 9-values
'''

'''
#asctime()
import time;
formattedtime = time.asctime(time.localtime(time.time()));
print(formattedtime);
'''

'''
#ctime()
import time;
formattedtime = time.ctime()
print(formattedtime);
'''

'''
#time module (altzone-var)
import time;
print(time.altzone);
'''

'''
#sleep() with ctime()
import time;
print(time.ctime());
time.sleep(5); #sleeps for 5 seconds
print(time.ctime());
```

```
'''

'''
#digital-clock
import time;
while True:
 ct = time.localtime(time.time())
 print(ct[3],":",ct[4],":",ct[5],end="\t\r")
 time.sleep(1)
'''

'''
#time-module(vars)
import time;
print(time.timezone);
print(time.tzname);


#calendar module
import calendar;
#print(calendar.calendar(2022,2,1,6));
#print(calendar.calendar(2022,2,0,6));
#print(calendar.firstweekday());
#print(calendar.isleap(2020));
#print(calendar.isleap(2022));
#print(calendar.leapdays(2020,2032));
#print(calendar.month(2022,8,2,1));
#print(calendar.monthcalendar(2022,12));
#print(calendar.monthrange(2022,8));
#print(calendar.monthrange(2022,9));
#calendar.prcal(2022,2,1,6);
#calendar.prmonth(2022,8,2,1);
#calendar.setfirstweekday(6);
##calendar.prmonth(2022,8,2,1);
#print(calendar.weekday(2022,8,12));
#gives weekday-code(0-6 mon-sun)


#datetime module (classes & objects)
import datetime;
date1 = datetime.datetime.now();
print(date1);



# 18th day 18.13-08-22

"""
==> Modules in Python:-
(module means python .py file)
= A Module contains Classes, Functions, Variables etc saved in a single-file (.py as extension)
= Every Python .py file is a Module
Ex:- math.py,
 random.py,
 time.py
 (pre-defined modules)

=> we have 2 types of modules,
 a) Pre-defined modules
 b) User-defined Modules (our own modules)

= Now, we create our own modules in python (user-defined modules)
##Program (SaiMath.py)
#Program to create our own user-defined module
Ex:-
"""
# (SaiMath.py)
a=100;
b=200;
def add(x,y):
 print(x+y);
def sub(x,y):
 print(x-y);
def prod(x,y):
 print(x*y);
def div(x,y):
 print(x/y);
def mod(x,y):
 print(x%y);

##Program (SaiMath.py)
# (Program to demo a Module with Functions and Variables)

# NOTE:-
# = "SaiMath" is UD-Module, It contains 2-Vars and 5-Funcs

##Program (SaiMath.py)
#Program to create our own user-defined module

#2-vars
a=100;
b=200;

#5-funcs
def add(x,y):
 print(x+y);
def sub(x,y):
 print(x-y);
def prod(x,y):
 print(x*y);
def div(x,y):
 print(x/y);
def mod(x,y):
 print(x%y);
"""
==> How to use Module in Python-prog??
= It is used with "import-keyword"
= To use such module in our prog, we have to import it using import-keyword
Ex:
 import <<ModuleName>>;
 import SaiMath;

(***)
= After importing module in our program, we can access module vars,funcs,classes using ModuleName (.dot operator)
Ex:-
 SaiMath.variable (SaiMath.a)
 SaiMath.function() (SaiMath.add(11,3))

##**main-program**
##Program(Demo1.py)
```

```
##(Prog to use SaiMath module and its content)


NOTE:-
= Here just run main-program(Demo1.py) & NO need to run module-program
"""
# //Program...
##**main-program**
##Program(Demo1.py)
##(Prog to use SaiMath module and its content)


import SaiMath;
print("Using SaiMath.py module in main-program");

print(SaiMath.a);
print(SaiMath.b);

SaiMath.add(11,3);
SaiMath.sub(11,3);
SaiMath.prod(11,3);
SaiMath.div(11,3);
SaiMath.mod(11,3);
'''

'''
#Module Aliasing(alternate-name) with as keyword
import SaiMath as SM; #use only alias-name o.w NameError
print(SM.a);
print(SM.b);
SM.add(11,3);
SM.sub(11,3);
SM.prod(11,3);
SM.div(11,3);
SM.mod(11,3);
#SaiMath.add(11,3) #dont use org-name
'''

'''
#specific-content import
#from...import
from SaiMath import a,add,mod;
print(a);  #use directly mod-name not-req
add(11,3);
mod(11,3);
print(b)
div(11,3);  #NameError
'''

'''
#import all content at a time(*)
from SaiMath import *; #* means import-all
print(a);  #no-need to use module-name
print(b);
add(11,3);
sub(11,3);
prod(11,3);
div(11,3);  #NO-Error
mod(11,3);


#Member-Aliasing(content-alias) with as-keyword
from SaiMath import a as x, add as sum;
print(x);  #module-name not-req
sum(11,3);
#print(a);  #Error
#add(11,3);  #Error



"""
==> Module Renaming while usage:-
= It is like alias-name for module
Ex:-
import SaiMath as SM;
= SaiMath is Org-ModuleName
= SM is Alias-name

Ex:- (Demo1.py)
(Prog to use SaiMath module and its content with Alias-name)



==> from...import in Module:-
= We can import only particular members from a module as follows,
= Advantage is we can use them directly without ModuleName
Ex:-
from SaiMath import a,add,mod;
print(a);
add(11,3);
mod(11,3);
#div(11,3);  #NameError

Ex:-
from SaiMath import *;  #Here * means all the members
print(a);
add(11,3);
mod(11,3);
div(11,3);  #NO-Error

Ex:- (Demo1.py)

==> Possible ways for Module-import:-
1) import ModuleName;    #Single-Module
Ex:-
 import SaiMath;

2) import Module1,Module2,....;  #Multiple-Modules
Ex:-
 import SaiMath1, SaiMath2;
=** Here SaiMath1/SaiMath2 should be defined in our working-directory

3) import Module1 as M;    #Using Alias-name
Ex:-
 import SaiMath as SM;

4) import Module1 as M1,Module2 as M2,....;
(#Multiple-Modules with alias-name)
Ex:-
```

```
 import SaiMath1 as SM1, SaiMath2 as SM2,.....;

5) from ModuleName import MemberName; #Only Members Import
Ex:-
 from SaiMath import a;

6) from ModuleName import Member1, Member2,...;
      #Multiple Members Import
Ex:-
 from SaiMath import a,add,sub;

7) from ModuleName import Member as M;
      #Members Import with Alias
Ex:-
 from SaiMath import add as plus;
 print(plus(11,3));

8) from ModuleName import Member as M1, Member as m2,...;
      #Multiple-Members Import with Alias
Ex:-
 from SaiMath import add as plus, sub as minus,...;
 print(plus(11,3));
 print(minus(11,3));

9) from ModuleName import *;
      # * indicates all members import directly
Ex:-
 from SaiMath import *;
 print(a);
 print(add(11,3));

NOTE:-
= from...import provides direct access to Members without ModuleName



==> Module-Members Aliasing:-
(for module-members we give alternate-name)
Ex:-
from SaiMath import a as x, add as sum;
print(x);
sum(11,3);

NOTE:-
= Once alias-name is given we have to use only that alias-name but not original names o.w NameError
Ex:-
from SaiMath import a as x, add as sum;
print(a);
add(11,3);

Ex:- (Demo1.py)

-----------------------------------------------------
==> Reloading a Module:-
= In main-prog, when a module is imported, it is loaded only once, even if we are import it multiple-times in a program
Ex:-
#Program(MyModule1.py)
print("Hello from MyModule1");

Ex:-(main-program)
"""
#Main-Program(Demo2.py)
import MyModule1;
import MyModule1;
import MyModule1;
print("End of the Program");
"""
# NOTE:-
(Refer-notes)
= MyModule1.py is loaded only once (for multiple-imports)
= After loading a module, and if it is updated from outside then updated new-changes is not available in our main-program

==> reload() function
= However, we can reload a module mulitple-times in our main-program using reload() function of "importlib"(new) module
Ex:- (Demo2.py)
"""
#Main-Program(Demo2.py)

import time;
import MyModule1;
import MyModule1;
import MyModule1;

print()
time.sleep(20)
import importlib;
importlib.reload(MyModule1)

print()
time.sleep(20)
from importlib import reload;
reload(MyModule1)

print("End of the main-Program");


# Ex:- (#MyModule1.py)
#Program(MyModule1.py)

print("Hello from MyModule1");

print("1st changes done in MyModule1");
#do changes in sleep-time

print("2nd changes done in MyModule1");

"""
-----------------------------------------------------
NOTE:-
from importlib import reload;
= importlib is new-module and imp is old-module
= reload() reloads the module & its contents multiple-times



==> __name__ sp-variable::-
(refer-notes)
= __name__ is a sp-var in python-program
= it gives name of currently executing program
```

```
= It gives module-name for module-program
= It gives __main__ for main-program

Ex:-(Program)
##MyModule2.py (module-program)

print("Hello from MyModule2");
print(__name__)

-------------------------------------
##Demo3.py (main-program)

print("Hello from Demo3.py main-program")
print(__name__)

import MyModule2;



=====================================================
***dir() and help()***
(Refer-notes)
==> Getting all members of Module using dir() function:-
= dir() lists all the members of a current-module
(variables, functions, classes etc) [.......]
Ex:-
dir() #lists all members of current-module(current-program or .py file)
dir(ModuleName) #Lists all members of specified module

= help() gives complete desc of module including comments

Ex:- (Demo4.py)
a=100;
b=200;
def f1():
 print("Hello World");
print(dir());

Ex:-(with module-name)
import math;
print(dir(math)); #Pre-defined Module

import SaiMath;
print(dir(SaiMath)); #User-defined Module

help(math);  #Provides complete description of math-module
"""

# //Program...
#Demo4.py (listings all the members of module)

a=100;
b=200;
c=300
def f1():
 print("Hello World");
def f2():
 print("Welcome to Python");
print(dir()); #here it gives a list of all members of current-program
'''

'''
import math;
print(dir(math));
'''
'''
print();
import SaiMath;
print(dir(SaiMath));
'''

'''
import math;
help(math);



# 19th day 19.15-08-22

"""
==> Packages in Python:-
= Package is a Folder/Directory
(Coll.of Modules i.e, .py files)

***def:-
 = Packages is collection/grouping of related modules as single-unit

NOTE:-
= Python Package folder contains (__init__.py) empty-file. It indicated that special-folder is a Package-folder
= A package may have sub-packages also

Ex1:-
==Diagram==
D:\SAISIR\Python\Packages> (normal-regular-folder)
 => Pack1 (pkg-folder)
  = __init__.py
  => Pack11 (sub-pkg-folder)
   = __init__.py
   = Demo1.py  (module)
   = Demo2.py (module)

-----------------------------------------------------------
=> Advantages:-
1) Naming conflicts can be removed
(we can have same-name .py file in multiple pkgs)
2) Project Components can be identified uniquely
(pkgname.filename)
Pack1.Demo
Pack2.Demo
3) Improves Modularity of Application
(it means grouping togather related .py files as single-pkg)


==> Creating packages & using them in program:-
##Program (Packages\Pack1 folder)
**(Module1.py)
def f1():
 print("Hello from Module1.py of Pack1 package");
```

```
##Program (Packages> folder) main-prog
**(Demo1.py)
import pack1.Module1;
Pack1.Module1.f1();
(or)
(Demo1.py)
from pack1.Module1 import f1;
f1();

NOTE:-
= to use pkgs we use import keyword only
Ex:-
 import PkgName.ModuleName;
 from PkgName.ModuleName import var,func;
"""


#Packages\pack1\Module1.py

def f1():
 print("Hello from Module1.py of Pack1 package");

# //Program...
#Packages\Demo1.py

import Pack1.Module1;
Pack1.Module1.f1();
'''

'''
#other-way
from Pack1.Module1 import f1;
f1();
'''

'''
#Package\Pack2 sub-pkg
from Pack2.Module1 import f1;
f1();
'''
'''
#sub-pkg(pack2\pack22 folder)
from Pack2.Pack22.Module2 import f2;
f2();


"""
=> Creating Sub-Pkgs & using them in our program:-
=** creating a folder(pkg) inside another folder is called sub-pkg

##Program (Packages\Pack1\Pack11 i.e Sub-Pkg/sub-folder)
(Module11.py)
def f11():
 print("Hello from Module11.py of Pack11 package");


##main-program in same
(Demo1.py - Same add below-code)
==add-prog-code for sub-pkg access===


"Assignment" (Demo1.py main-prog only)
-----------------
1) WAP to work with Packages import with alias name
2) WAP to work with specific content import of a package with alias-name
(both cases use "as" keyword)
Ex:-
 import Pack1.Module1 as PM;
 PM.f1()
 (or)
 import Pack1.Pack11.Module11 as PPM;
 PPM.f11()
 (or)
 from Pack1.Module1 import f1 as hello;
 hello()
 (or)
 from Pack1.Pack11.Module11 import f11 as hi;
 hi()



##Program (Packages\Pack1 pkg-folder)
##(Module1.py)

def f1():
 print("Hello from Module1.py of Pack1 package");
"""
# ------------------------------------------------------
##Program (Packages> folder) main-prog
##(Demo1.py)


import Pack1.Module1;
Pack1.Module1.f1();
'''

'''
#other-way
from Pack1.Module1 import f1;
f1();


#access sub-pkg (PkgName.SubPkgName.ModName)
import Pack1.Pack11.Module11;
Pack1.Pack11.Module11.f11()

#other-way(using from)
from Pack1.Pack11.Module11 import f11;
f11()

# ------------------------------------------------
##Program (Packages\Pack1\Pack11 i.e Sub-Pkg/sub-folder)
#(Module11.py)

def f11():
 print("Hello from Module11.py of Pack11 sub-package");


"""
================================================
```

```
"Advanced Python Concepts"
---------------------------------
I) OOP:-
(Object Oriented Programming)
= it is not new-programming
= it is not separate-progrmming
(it is available in C++,Java,.net,Python also)
(it is a concept or subject)
def:-
 it is a programming based on real-life situtaions(objects) as real-time computer programs
Ex:-
 Online shopping
 Online banking
 Online Tickets

==> OOPS principles:- (Based on real-life)
==Diagram==
 a) Abstraction
 b) Encapsulation
 c) Inheritance
 d) Polymorphism


==> Abstraction::-
(Develop a Product in Company)
def:-
 = Hiding unnecessary things in a Product & exposing necessary things outside a product
 (desiging a product-model as dummy-product)
Ex:-
 Marker-Pen
  - Hidden (Ink,Refill)
  - Exposed (Body,Cap,Nib)

=** In python-program, we do abstraction in 2-ways,
 i) Abstract-classes
 ii) Interfaces


==> Encapsulation::-
= Encapsulation says that every product has 2-things,
 i) Data as Input
 ii) Functionality as Usage
Ex:-
 Marker-Pen
  - Input (Ink-color, Ink-Qty)
  - Use (Writing on Whiteboard)

=** In python-program, we do Encapsulation with,
 i) classes & objects


==> Inheritance::-
(G.Parents----->Parents----->Children----->G.Childrens)
def:-
 Getting properties & functionalities from existing-product to new-product
Ex:-
 iPhone11 (2G,3G,4G)
  |
  |
  |
 iPhone12 (2G,3G,4G + 5G**)

=** In python-program, we do Inheritance with
 i) Parent-class
 ii) Child-class
(Advantage of Inheritance is Re-usability)


==> Polymorphism::-
= Poly means Many (2 or more)
= Morphism means Forms (behaviour or functionality)
def:-
 Single Product in multiple-forms based on some condition
Ex:-
==Diagram==
 H2O(3-Forms) ----> (temp. is condition)
  = Solid(ICE)
  = Liquid(Water)
  = Gas(Vapour)

=** In python-program, we do Polymorphism as follows,
 i) Overloading
 ii) Overriding
 ii) Ducktyping



NOTE:-
= Python supports all principles of OOP. Hence it is called as OOP.Lang

"""



# 20th day 20.16-08-22

"""
****Encapsuation***
-----------------------------
==> Classes & Object:- (in OOP)
-----------------------------
= This concept is related to Encapsulation
(OOP-Principle)

**= In Python, everything is an object and to create an object we require a Class(model)

=> Def:-
 = A class is a Model, which represents properties(Vars) and behaviour(functions)
=** Properties ---> Variables
=** Behaviour means Functions/Methods

[Class = data-Variables + Functions/Methods]

=** Binding togather data-properties & functionalities into a single-unit
(class is collection of vars & methods(Funcs))

NOTE:-
= A class is a Model/Plan/Design/Arch/Blue-print before getting an actual Product (i.e, Object)
Ex:-
 House(Blue-print) ---> Class(model) [Civil-Engg]
```

```
 Constructed-House(Object/End-Product)
=** Object is End-Product used by End-User(Customer/Client/Humans)
=** For 1-class(model/blue-print) we can have multiple-objects(products)

=> How to define class?
= It is defined with "class" keyword
Syntax:-
 class Classname:
  '''Doc-String or Comments'''
  data-variables;  (instance/static/local)
  methods;   (instance/static/class)

= Indentation is Compulsory for class-body o.w Error...
(inside a class function is called as METHOD)

NOTE:-
'''Doc-String''' (Description/Comment of Class and it Optional)
= We can get it as follows,
Ex:-
help(Classname); #complete-desc/def of a class i.e, help()
print(Student.__doc__);


##Program (Student.py)
#(Prog to define Student-class)

class Student:
 '''Student class definition'''

help(Student);
print(Student.__doc__);

NOTE:-
= inside class, vars & methods are optional(not-compulsory)


===> Class with Vars & Methods:-
Ex:- (Student-class)
 sno/sname/height ---> (Variables)
 def display(): ---> (Methods/funcs)
 def show():

NOTE:-
= We define-vars(declare & initialize) using sp-method(Func)
Ex:-
 def __init__(self):

=** self means current object of class
"""
##Program (Student1.py)
#Program to work with Student1-class with vars & methods(funcs)
##Program (Student1.py)
#Program to work with Student1-class with vars & methods(funcs)

class Student1:
 '''Student1-class def...'''
 def __init__(self):
  self.sno=1001  #(.dot operator)
  self.sname='Sai'
  self.height=6.2
 def display(self):
  print(self.sno)
  print(self.sname)
  print(self.height)


print(Student1.__doc__)
print()
help(Student1)
print()


# NOTE:-
# = Student1-class has (3-vars & 2-methods)
"""
**Working with object**
==> Object:-
 = Object is instance of a class, which represents state(Vars) and behavior(func) of a class

(instance ----> 1-individual-unit)

=> how to create object??
Syntax:-
 objrefvar = Classname();
Ex:-
 s1 = Student1();
(when class-obj is created, __init__() is called auto.)
==Diagram==
(obj.ref.var--------->[sno/sname/height])

=> using object??
= to work or use any class object we use (Dot .) operator
Ex:-
 objref.Variables
 objref.methods()
Ex:-
 s1.sno
 s1.display()
"""
# [contd..]
##Program (Student1.py)
##Program (Student1.py)
#Program to work with Student1-class with vars & methods(funcs)

class Student1:
 '''Student1-class def...'''
 def __init__(self):
  self.sno=1001  #(.dot operator)
  self.sname='Sai'
  self.height=6.2
 def display(self):
  print("Roll-No =",self.sno)
  print("Name =",self.sname)
  print("Height =",self.height)



#case1
print(Student1.__doc__)
print()
```

```
help(Student1)
print()


#case2 (working with objs)
s1 = Student1();
print("s1-obj Student-details")
s1.display();

print()
s2 = Student1();
print("s2-obj Student-details")
s2.display();

# NOTE:-
# = Here s1 and s2 both objs have same-data(vars) b'coz in __init__() we have static-values


# [contd..]
##Program (Student1.py)
##Program (Student1.py)
#Program to work with Student1-class with vars & methods(funcs)

class Student1:
 '''Student1-class def...'''
 def __init__(self):
  self.sno=1001  #(.dot operator)
  self.sname='Sai'
  self.height=6.2
 def display(self):
  print("Roll-No =",self.sno)
  print("Name =",self.sname)
  print("Height =",self.height)



#case1
print(Student1.__doc__)
print()
help(Student1)
print()


#case2 (working with objs)
s1 = Student1();
print("s1-obj Student-details")
s1.display();

print()
s2 = Student1();
s2.sno=1002  #modify obj-data(Vars) outside class
s2.sname='Ram'
s2.height=5.9
print("s2-obj Student-details")
s2.display();

"""
NOTE:-
= Here we can modify s2(obj-data-vars) outside class
Ex:-
 s2.sno=1002
 s2.sname='Ram'
 s2.height=5.9
(this techique is not advisable)

***= However, we can modify object data using dynamic-values with input() inside __init__()
Ex:-
 self.rno = int(input("Enter Roll-No :: "))
"""
##Program (Student2.py)
#Program to work with Student2-class with vars & methods(funcs) with dynamic-values
##Program (Student2.py)
#Program to work with Student2-class with vars & methods(funcs) with dynamic-values


class Student2:
 '''Student2-class def...'''
 def __init__(self):
  self.sno=int(input("Enter Roll-No :: "))
  self.sname=input("Enter Name :: ")
  self.height=float(input("Enter Height :: "))
 def display(self):
  print("Roll-No =",self.sno)
  print("Name =",self.sname)
  print("Height =",self.height)


#working with objs
s1 = Student2(); #__init__() called auto for obj.create
print("s1-obj Student-details")
s1.display();

print()
s2 = Student2();
print("s2-obj Student-details")
s2.display();

# NOTE:-
# = Whenever we create obj of a class, __init__() is called auto with self-var(current-obj is passed auto)

# ***Other-Examples***
##Program (Employee.py)
##Program to work with Employee-class with vars & methods(funcs) with dynamic-values
##Program (Employee.py)
##Program to work with Employee-class with vars & methods(funcs) with dynamic-values

class Employee:
 '''Employee-class def...'''
 def __init__(self):
  self.empno=int(input("Enter Emp-No :: "))
  self.ename=input("Enter Emp-Name :: ")
  self.sal=float(input("Enter Salary :: "))
 def show(self):
  print("Emp-No =",self.empno)
  print("Name =",self.ename)
  print("Salary =",self.sal)


#working with Employee-class-objs
e1 = Employee(); #__init__() called auto for obj.create
```

```
print("e1-obj Employee-details")
e1.show();

print()
e2 = Employee(); #__init__() called auto for obj.create
print("e2-obj Employee-details")
e2.show();


# **"Assignment"**
##Program (Employee1.py)
##Program to work with Employee1-class with vars & methods(funcs) with dynamic-values
# (empno/ename/job/sal/comm) --> vars inside __init__()
# [cal. totalsal = e1.sal+e1.comm] --> inside main-prog
# [cal. annulsal = totalsal*12]


"Assignment"
##Program (Customer.py)
##WAP to work with Customer-class with vars & methods (dynamic-values)
# (cacno,cname,cbranch,cbal)


# 21st day 21.17-08-22

"""
==> Constructor inside class:- (__init__())
(Refer-notes)
= It is special-method in a class
= Its name is __init__(self)
= It is executed automatically when object is created
(s1 = Student())
#Student() -> Student-class constructor
= Its main purpose is declare and initialize instance-variables(obj-vars)
(rollno/name/height)
= It is executed only once per object creation (auto)
= It takes atleast 1-parameter
i.e, self(current-obj Ex:- s1/s2)
= It is optional in a class, If it is not given then python provides default-constructor
Ex:-
 def __init__(self):
  pass; #body-w.o-stmts
"""
##Program (ConstructorEx1.py)
##Program to define constructor in python-class
##Program (ConstructorEx1.py)
##Program to define constructor in python-class

class A:
 '''class-A definition'''
 def __init__(self):
  print("Class-A constructor is called");
  self.x=int(input("Enter X value :: "));
  self.y=int(input("Enter Y value :: "));
 def show(self):
  print(self.x,self.y)


#obj-creations
obj1 = A()  #__init__() constructor is call auto
obj2 = A()
obj3 = A()

print()
print("Obj1-data");
obj1.show();

print()
print("Obj2-data");
obj2.show();

print()
print("Obj3-data");
obj3.show();

"""
-----------------------------------------------------
==> Constructor with input-parameters::-
-----------------------------------------------------
(refer-notes)
= constructor is sp.method of a class (__init__())
= it is called automatically, when object is created
Ex:-
 s1 = Student(); #Student-class const-call
= constructor is used to declare & initialize instance-variables(obj-vars)
= for constructor, "self"(current-obj) is 1st-para
= for this constructor, we can pass our own  extra-input-parameters
Ex:-
 def __init__(self,r,n,h):
  self.rollno=r;
  self.sname=n;
  self.height=h;
=*** we can use these extra-input-parameters, to initialize our instance-variables(obj-vars)
"""

##Program (ConstructorEx2.py)
##Program to define constructor with para(args) in python-class
##Program (ConstructorEx2.py)
##Program to define constructor with para(args) in python-class

class Student:
 '''Student-class definition'''
 def __init__(self,r,n,h):
  self.rollno=r;
  self.sname=n;
  self.height=h;
 def display(self):
  print(self.rollno,self.sname,self.height)


s1 = Student(1001,"Sai",6.0); #i/p-para to const.
s2 = Student(1002,"Ram",5.9);

print("s1-details");
s1.display();
print()
print("s2-details");
s2.display();

"""
```

```
NOTE:-
= for __init__(self,....) constructor, current-obj(s1/s2) is passed auto to self(1st-para)

NOTE:-
= We can take input-parameters to constructors same-names as that of instance-variables(obj-vars)
Ex:-
 def __init__(self,rollno,name,height):
  self.rollno=rollno;
  self.name=name;
  self.height=height;
(obj-vars are assigned with local-vars)
"""


"Assignment"
#Employee2.py
#Program to demo Employee-class with constructor input-parameters, instance-vars, instance-methods & self-var using objects
# (empno,ename,job,sal)
e1 = Employee(1122,"Sai Kumar","Manager",6500.50)
e2 = Employee(1133,"Ram Kumar","HR",5500.50)


"Assignment"
#Customer2.py
#Program to demo Customer-class with constructor input-parameters, instance-vars, instance-methods & self-var using objects
# (custid,custacno,custname,custbal)
c1 = Customer(123456,9876543210,"Sai",50000)
c2 = Customer(112233,9988776655,"Ram",60000)


"""
=================================================
==> self variable in Methods:-
= it is sp-var of a class
= is used as input-parameter to constructor & methods
Ex:-
 def __init__(self):
  .....
 def display(self):
  .....
= self means current-obj under execution
===Diagram===
s1 = Student()
s2 = Student()
s3 = Student()
s1.display()
s2.display()
s3.display()

NOTE:-
= it is sp-obj-ref-var of a class
= it is used only inside a class
"""
##Program (Student11.py)
#Program to demo self-var in Methods of a class

#class-def
class Student11:
 "Student11 class definition"
 def __init__(self,rollno,name,height):
  self.rollno=rollno;
  self.name=name;
  self.height=height;

 def display(self):
  print("RollNo : {}\nName : {}\nHeight : {}".format(self.rollno,self.name,self.height) );

s1 = Student11(1001,"Sai",6.0);
s2 = Student11(1002,"Ram",5.9);
s3 = Student11(1003,"Ali",5.6);
print()
s1.display()
print()
s2.display();
print()
s3.display()

"""
==> Difference between Constructors & Methods:-
(refer-notes)
1)
= Constructor name is fixed-name __init__()
= Method-name can be any user-defined name (display()/accept()/show())
2)
= Constructor is called & executed automatically when object is created (s1 = Student())
= Method is called manually & executed (s1.display())
3)
= Constructor is called & executed only once per object
= Method can be called & executed any no.of times per object
4)
= Constructor is used to declare and initialize Instance-Variables(obj-var)
= Method is used to write Buss.Logic in a class


------------------------------------------------------------
**==> Types of Variables in a Python-class:-
= In python-class, we have 3-types of variables,
1) Instance-Variables (Object-Vars) #separate for every-obj
2) Static-Variables (Common-Vars) #common for all-objs
3) Local-Variables (Method-Vars) #define inside-a-method
-------------------------------------
4) Class-vars (sp-vars inside class-methods)



(Working with Instance-Variables)
=>1) Instance-Variables:- (Object-Level Vars)
(Refer-notes)
= Instance means Object
(Hence called as Object-vars)
= They are separate for every object of a class (separate-copy)
==Diagram== (rollno,name,height)
s1 = Student();  s1 -----> [(rollno,name,height)]
s2 = Student();  s2 -----> [(rollno,name,height)]
s3 = Student();  s3 -----> [(rollno,name,height)]

=> How to work(use) with Instance-Vars??
i) Inside class with self-var
ii) Outside class with obj.ref.var.name
"""
```

```
##Program (InstanceVarsEx1.py)
##Program to define & work with instance-vars in python-program(class)
##Program (InstanceVarsEx1.py)
##Program to define & work with instance-vars in python-program(class) 3-cases

class Student:
 def __init__(self):
  self.rollno=1001;
  self.sname="Sai";
  self.height=6.2;
 def display(self):
  print(self.rollno,self.sname,self.height);


#case-1(inside class)
s1 = Student()
print("s1-obj details ::");
s1.display()

#case-2(outside class) #it is not advisable
print()
s2 = Student()
s2.rollno=1002
s2.sname="Ram"
s2.height=5.9
print("s2-obj details ::");
print(s2.rollno,s2.sname,s2.height);



"Assignment"
##Program (InstanceVarsEx2.py)
####Program to define instance-vars in python-program(class) 3-cases using Employee-class
# (empno,ename,job,sal)

"Assignment"
##Program (InstanceVarsEx3.py)
####Program to define instance-vars in python-program(class) 3-cases using Customer-class
# (cacno,cname,bal,caddr)

"""
=====================================================
==>2) Static-Variables (Common Vars):-
**= These Variables are common for all the objects of a class
***= Such vars are declared directly in a class
(basically w.o constructor or w.o method)
= For all objs only 1-copy of memory is allocated and shared with all objs of that class

=> How to access(use)??
 (2-ways)
 =** Static-Variables are accessed with
  Classname or
  Objname
 (Classname is preferable)
 (Both inside/outside class)

==Diagram==
[s1:-rollno/name/height]--->[course,college]<-----[s2:-rollno/name/height]
[s3],[s4]
#here course/college etc are static-vars of a class
#here rollno/name/height are instance-vars
"""
# //Program
##Program (StaticVarsEx1.py)
#Program to demo Static-Vars in a python-class

class StaticVars:
 '''Static-Variables in a python-class'''
 c=11; #c is static-var(common-var for all objs)
 def __init__(self):
  self.x=10;
  self.y=20;
 def display(self):
  print(self.x);
  print(self.y);
  print(StaticVars.c); #self.c


obj1 = StaticVars();  #obj1---->[x:10,y:20]<----(C:11)
print("For obj1 ::");
obj1.display();

print();
obj2 = StaticVars();
#obj2---->[x:10,y:20]<----(C:11)
obj2.x=100;
obj2.y=200;
#obj2---->[x:100,y:200]<----(C:11)
print("For obj2 ::");
obj2.display();

print()
print("Using Classname or Obj.name ::");
print(StaticVars.c);
print(obj1.c);
print(obj2.c);


print()
#sp-case1
#modify static-var using classname
StaticVars.c=22
print("After modify class-var using classname");
print(StaticVars.c);
print(obj1.c);
print(obj2.c);

print()
#sp-case2
print("Modify instance-var using obj1");
obj1.x=1000
obj1.y=2000
obj1.display();
obj2.display();



# NOTE:-
# = Instance-Vars are separate-copy for every-object
```

```
# = Static-Vars are common-copy(sharable) for all-objects
# = Changes(Modify) done to Instance-Vars are updated to that obj only
# = Changes(Modify) done to Static-Vars are updated to  every-obj



"""
=====================================================
***3) Local-Variables:-
= Variables declared/defined inside particular-method of a class is called Local-Variables (Method-Vars)
= Such vars are created when method is executed and destroyed once method execution is completed
= Local-Variables have local-access and direct-access in that method-only but not outside the method
(input-para/args are also local-vars)
(local-access & direct-access)

Ex:-
"""
##Program (LocalVariables.py)
#Prog to demo Local-Variables inside method of a class

class LocalVariables:
 '''LocalVariables class-definition'''
 def m1(self):
  print("Inside m1() of class")
  x=10;  #x is local-var to m1()
  print(x);
  #print(y)
 def m2(self):
  print("Inside m2() of class")
  y=20;  #y is local-var to m2()
  print(y);
  #print(x); #NameError... (x is not-defined)

obj = LocalVariables();
obj.m1();
obj.m2();



# NOTE:-
# = Instance-Vars use obj-name(self)
# = Class-vars use Classname
# = Local-vars use direct-name (No obj-name, No classname)



# 22nd day 22.18-08-22

"""
***==> Types of Methods in a Python-class:-
= In class, we have 3-types of methods,
1) Instance-Methods (obj-methods)
2) Static-Methods (common-methods)
3) Class-Methods  (sp-methods with class-var)


1) Instance-Methods:-
(Instance means object)
= Methods whose Functionality(work) is diff. for every object are called Instance-Methods
= For Instance-Methods, we pass "self" variable (as 1st-para)
(self is current-obj)
Ex:-
def m1(self): #self means current-obj under execution
def display(self):
= In these methods, we access instance-vars using "self" var
s1[self.rollno/name/height]
s2[self.rollno/name/height]
....
= Accessing them,
 a) Inside class using self-var (self.m1())
 b) Outside class using Obj.Ref.Var.Name (s1.display())

Ex:-
"""
##Program (InstanceMethods1.py)
# (Prog to define Instance-Methods in a class)
#Prog to define Instance-Methods in a class

class Student:
 def __init__(self,rollno,name,avg):
  self.rollno=rollno;
  self.name=name;
  self.avg=avg;
 def display(self):
  print(self.rollno);
  print(self.name);
  print(self.avg);
 def grade(self):
  if(avg>=75):
   print("Distinction");
  elif(avg>=65):
   print("1st Class");
  elif(avg>=50):
   print("2nd Class");
  elif(avg>=35):
   print("3rd Class");
  else:
   print("FAILED");

n = int(input("Enter No.of Students :"));
for i in range(n):
 rollno=int(input("Enter-Rollno :"));
 name=input("Enter-Name :");
 avg=float(input("Enter-Avg-Marks :"));
 studObj = Student(rollno,name,avg);
 studObj.display();
 studObj.grade();
 print();

# NOTE:-
# = For Instance-Methods, compulsory we have self(input-parameter)


"""
===>>
2) Static-Methods:-
= They are general utility methods
(common func. for all objs of a class)
= Such methods are declared with "@staticmethod" decorator
```

```
= Here we do-not pass "self" as input-parameter
= In this method, we DO-NOT use instance-vars
=> Accessing:-
 a) Using Classname Ex:- Classname.staticmethod()
 b) Using Obj.Ref.Var Ex:- obj.staticmethod()
 =** preferable is class-name

Ex:-
"""
##Program (StaticMethods.py)
#Prog to demo Static-Methods in a class

class StaticMethods:
 @staticmethod
 def sum(x,y):
  print(x+y);
 @staticmethod
 def sub(x,y):
  print(x-y);
 @staticmethod
 def prod(x,y):
  print(x*y);

#using classname(w.o creating obj -> preferable)
print("Using Classname");
StaticMethods.sum(11,3);
StaticMethods.sub(11,3);
StaticMethods.prod(11,3);

print()
#using obj-ref-name
print("Using Obj.Name");
obj = StaticMethods();
obj.sum(11,33);
obj.sub(11,33);
obj.prod(11,33);


"""
==>>
3) Class Methods:- (with class-var(special))
= In this method, we mainly use/access Static-Vars/Methods of class or create object of a class
= It is declared using "@classmethod" decorator (annotation)
= For such methods, we provide sp.class-variables as parameter
(sp-var as parameter known as class-method class-ref-var)
(it is lst parameter, mainly used to access static-vars/methods of class)
(class-ref-var acts like classname)
(it refers to class-def in memory)
= it can take parameters (after class-method-class-var)
=> Accessing,
 a) Using classname Ex:- Classname.classmethod()
 b) Using Obj.Ref.Var Ex:- obj.classmethod()
 (preferable is Classname)
Ex:-
"""
##Program (ClassMethods.py)
#Prog to demo Class-Methods in a class

class ClassMethods:
 c=11; #static-var(common for all objs of class)
 def __init__(self):
  self.x=10; #instance-vars
  self.y=20;
 @staticmethod
 def m1():
  print("Static-Method m1()")
 def m2(self):
  print("Instance-Method m2()",self.x,self.y)
 @classmethod
 def m3(clsvar):
  print("Static-Var :",ClassMethods.c,clsvar.c);
  ClassMethods.m1()
  clsvar.m1()
  obj1 = ClassMethods();
  obj1.m2()
  obj2 = clsvar()
  obj2.m2()


#using classname
ClassMethods.m3();
print()
#obj-ref-var
obj = ClassMethods()
obj.m3()


"""
NOTE:-
= Instance-Methods & Static-Methods are mainly used in a class
= Class-Methods are rarely used in a class(Sp-case with class-var)

=======================================================
==> Passing Object of One-class to Another-class:-
(Passing class-object as parameter to method)
= We can pass object of one-class to another-class and access its members in another-class using obj as parameter to method
= Adv, is Reusability of code
(i.e, re-use one class members in another-class)
==Diagram==

Ex:-
"""
##Program (ObjectAsPara.py)
#Prog to demo passing object as parameter

#1st-class
class Student:
 def __init__(self,rollno,name,height):
  self.rollno=rollno;
  self.name=name;
  self.height=height;
 def show(self):
  print(self.rollno);
  print(self.name);
  print(self.height);

#2nd-class(here we use 1st-class)
class Demos:
 @staticmethod
 def update(studobj): #studobj=s1(alias)
```

```
  studobj.height=studobj.height+0.2;
  studobj.show();


#main-prog
s1 = Student(1001,"Sai",6.0);
s1.show();
print()
Demos.update(s1);
print()
s1.show();

"""
NOTE:-
= Here any changes done with formal-parameter object-ref(studobj) are updated/reflected to actual-argument object-ref(s1)


==================================================
==>> Inner-Classes in Python:-
= Defining one-class inside another-class is called Inner-Class
(class-with-in-class)
Ex:-
class Car: #outer-class
 ....
 ....
 class Engine: #inner-class
  ....
  ....
= Here without "class Car", there is no-chance of "class Engine"
NOTE:-
=** Hence, Inner-class are part of Outer-class
= Hence Inner-Class obj is created using Outer-class obj
Syntax:
 outObj = OuterClass(); #1st create obj of outer-class
 inObj = outObj.InnerClass(); #2nd using outer-class
       #object create inner-class obj

Ex:-
"""
##Program (InnerClass.py)
#Program to demo Inner-Class

class Outer:
 def __init__(self):
  print("Outer-class-Constructor obj created");
 class Inner:
  def __init__(self):
   print("Inner-class-Constructor obj created");
  def m1(self):
   print("Inner-Class m1() method");


#case-1
out = Outer();  #step1
inn = out.Inner(); #step2
inn.m1();
'''

'''
#case-2(w.o using Outer-class obj.ref.name)
inn2 = Outer().Inner();
inn2.m1();


#case-3(w.o both Outer-class & Inner-class obj.ref.var)
Outer().Inner().m1();



# NOTE:- (prev-program)
# = We can access Inner-Class members in different ways,
# 1) (with obj.ref's)
out = Outer();
inn = out.Inner();
inn.m1();
# 2) (with only inner-class obj-ref)
inn = Outer().Inner();
inn.m1();
# 3) (w.o any obj-refs)
Outer().Inner().m1();




# 23rd day 23.19-08-22

"""
==> Garbage Collection:-
= It means deleting unnecessary objects from program during execution and saving the memory for performance of prog (speedy-execution)
= In traditional P.Langs like C/C++, programmer does GC manually
(Hence, we had OutOfMemory issues)

= In python, it is done automatically in background using GC-Assistant (destroys useless objs)
= If any obj does not have any reference-var then such obj is eligible for GC
Ex:-
 s1[] ---> [rollno/name/height]
 del s1;
 s1[]    [rollno/name/height] is ready for GC

=> How to enable/disable GC in Python-prog?
= By default GC is enabled in program, however we can disable it also
= "gc module" is used for this and below methods,
1) gc.isenabled() #gives True if it is enabled
2) gc.disable()  #disables GC explicitly
3) gc.enable()  #enables GC explicitly

Ex:-
"""
##Program (GCEx1.py)
# (Prog to demo GC)
#Prog to demo GC

import gc;
import time;

print(gc.isenabled());
time.sleep(5);

gc.disable();
print("GC is disabled in program");
```

```
time.sleep(5);
print(gc.isenabled());

gc.enable();
print("GC is Enabled in program");
time.sleep(5);
print(gc.isenabled());

"""
=====================================================
==> Destructor in Python-Class:-
= It is special method to remove/close unnecessary resources/objs in a program
= It is,
Ex:
 __del__(self):
  ...
  ...
  ...
= Just before destroying any object in program by GC, GC always calls destructor-method to perform cleaning-activity
Ex:- (closing DB-conn, Network-conn, close-files etc)
= Once destructor-method execution is done, GC automatically destroys that unnecessary-object

NOTE:-
= destructor-method does only cleaning-activity but not actual destroying of object

[cleaning-activity(closing-objs) ------> destroying-objs]
(it is called automatically in program)

Ex:-
"""
##Program (DestructorEx1.py)
# (Prog to demo destructor-method)
#Prog to demo destructor-method

import time;
class Demo:
 def __init__(self):
  print("Object Created & Initialized");
 def __del__(self):
  print("Object Resource Cleaning is getting done...");

obj1 = Demo();
obj2 = Demo();
time.sleep(5);
time.sleep(5);
print("End of Prog");

"""
NOTE:-
= destructor-method (__del__) is automatically called at end-of the program & then finally GC is done automatically in memory


=====================================================
***==> Using members of one-class inside Another-class:-
(class-members =vars + methods)
Ex:-
 class A (x,y,m1())
 class B (p,q,m2())
(How to use class-A members(x,y,m1) in class-B)

=** It is done in 2-ways,
1) By Composition (has-a-relationship)
2) By Inheritance (is-a-relationship/kind-of-relationship)


Now,
***1) By Composition:::-
(Has-a-relationship):-
(Having object of another class in our class)
= By using Classname or by creating object of another class in your class
= Advantage is, "Code-Reusability"
"""
###Program (CompositionEx1.py)
# (Prog to demo composition)

###Program (CompositionEx1.py)
#Prog to demo composition in python-classes (using object/classname)

class A:
 def __init__(self):
  self.x=10;
  self.y=20;
 def m1(self):
  print("Class-A m1() method",self.x,self.y);

class B:
 def __init__(self):
  aobj1=A();  #inside const create obj
  aobj1.m1();
 def m2(self):
  aobj2=A();  #inside method create obj
  aobj2.m1();
 @staticmethod
 def m3(aobj3):  #obj as para(aobj3=aobj)
  aobj3.m1()

#case-1
bobj = B();

print()
#case-2
bobj.m2()

print()
#case-3
aobj= A();
B.m3(aobj)

"""
=====================================================
2) By using Inheritance (is-a-relationship):-
(child-class is-a kind-of base-class)
Def:-
 = Getting data-props(vars), methods and constructors from Parent-class to Child-class is called Inheritance
= Hence Parent-class members can be Re-used in child-class
= Child-class extends Parent-class Functionality
= Child-class is more powerful than Parent-class functionality

==Diagram==
 class A  (Parent) x,y,m1()  #Base/Super
```

```
  |
  |
 class B(A) (Child) p,q,m2() + x,y,m1() #Derived/Sub

Syntax:-
 class Childclass(Parentclass):

=** Just create child-class object and access both Parent-class members & Child-class members
= Advantage, is Re-usability of code w.o PC object

Ex:-
##Program (InheritanceEx1.py)
(Prog to demo Inheritance)
=** W.O Base-class object, only with child-class object, we can access B.C members in S.C using its object (Re-usability)


NOTE:-
= All the methods of Parent-class are available to Child-class
= Hence using Child-class obj.ref.var we can access both Parent-class methods and Child-class methods
Ex:-
//Program (InheritanceEx1.py)
(Prog to demo Inheritance)

NOTE:-
= Like methods, from Parent-class vars (data-props) are available to Child-class
Ex:-
"""
# //Program (InheritanceEx1.py)
# (Prog to demo Inheritance)
##Program (InheritanceEx1.py)
#Prog to demo Inheritance in python-coding


#case-1 (PC all mem in Child-class)
class Pclass:
 a=10;
 def __init__(self):
  self.b=20;
 def m1(self):
  print("Parent-class Instance m1() Method");
 @classmethod
 def m2(self):
  print("Parent-class Class m2() Method");
 @staticmethod
 def m3():
  print("Parent-class Static m3() Method");

class Cclass(Pclass):
 pass;  #indicates body W.O any stmts
 #in child-class (a,b,m1,m2,m3 are inherited)

cobj = Cclass();
print(cobj.a);
print(cobj.b);
cobj.m1();
cobj.m2();
cobj.m3();
'''


'''
#case-2
class Pclass:
 def m1(self):
  print("Parent-class m1() instance-method");
class Cclass(Pclass):
 def m2(self):
  print("Child-class m2() instance-method");

obj = Cclass();
obj.m1();
obj.m2();


#case-3(***)
class Pclass:
 a=10;
 def __init__(self):
  self.b=20;

class Cclass(Pclass):
 c=30;
 def __init__(self):
  super().__init__(); #calls BC/PC constructor
  self.d=40;

obj = Cclass();
print(obj.a);
print(obj.b);
print(obj.c);
print(obj.d);


# NOTE:-
# = using super() function, we can call parent-class constructors/methods with same-name in child-class from child-class
# = If we comment the line "super().__init__();" then "var b" is not available to Child-class


"""
==========================================
===> Parent-Child Constructors:-
(sp-cases in Inheritance)
= Whenever we are creating Child-class objs then child-class constructor is executed
==Diagram===
= If there is no child-class constructor then parent-class constructor is executed auto.
(but parent-class obj is not created)
==Diagram===
= From child-class constructor, we can reuse parent-class constructor using super() function
==Diagram===
"""

# Ex:-
##Program (ParentChildConstructor.py)
#Prog to demo Parent and Child-class constructors

class A:
 def __init__(self):
  print("parent-class constructor");
  print(id(self));
  print(type(self));
```

```
class B(A):
 def __init__(self):
  super().__init__()
  print("child-class constructor");
  print(id(self));
  print(type(self));
 #pass;  #body W.O any stmts


obj = B();
print()
print(id(obj));
print(type(obj));



# NOTE:-
# = pass stmt indicates body without any stmts


# 24th day 24.20-08-22

"""
==> Types of Inheritance:-
(PC----->CC)
= Python has 6-types of Inheritance. They are,
1) Single Inheritance
2) Multi-Level Inheritance
3) Hierarchical Inheritance
4) Multiple Inheritance
5) Hybrid Inheritance
--------------------------
6) Cyclic Inheritance (Not-supported)


**********
1) Single Inheritance:-
= In this, single parent-class gives derivation to single child-class
==Diagram==
 class A: #1-PC(x,y,m1())
  |
  |
 class B(A): #only 1-CC (p,q,m2() + x,y,m1())

obj = B()
obj.x
obj.y
obj.m1()
-----------
obj.p
obj.q
obj.m2()
"""

# Ex:-
##Program (SingleInheritance.py)
#Prog to demo Single-Inheritance (Single PC --> Single CC)

class A:
 def __init__(self):
  self.x=10;
  self.y=20;
 def m1(self):
  print("Parent-class-A m1() Method",self.x,self.y);
class B(A):
    #x,y,m1,__init__()
 def __init__(self):
  super().__init__()
  self.p=100;
  self.q=200;
 def m2(self): #m1() is inherited from class-A
  print("Child-class-B m2() Method",self.p,self.q);

obj = B();
obj.m1();
obj.m2();



"""
2) Multi-Level Inheritance:-
= In this, it is extension of Single-Inheritance, in which Child-class acts a Parent-class for next-derivation(next-child-class)
==Diagram==
 class A: #PC (level-0) x,y,m1()
  |
  |
 class B(A): #CC (level-1) p,q,m2() + x,y,m1()
  |
  |
 class C(B): #CC (level-2) m,n,m3()+p,q,m2() + x,y,m1()

obj = C();
obj.x,y,m1()
obj.p,q,m2();
obj.m,n,m3();


Ex:-
"""
##Program (MultiLevelInheritance.py)
#Prog to demo Multi-Level-Inheritance

class A:
 def __init__(self):
  self.x=10;
  self.y=20;
 def m1(self):
  print("Parent-class A m1() Method",self.x,self.y);
class B(A):
 def __init__(self):
  super().__init__()
  self.p=100;
  self.q=200;
 def m2(self): #m1()
  print("Child-class B m2() Method",self.p,self.q);
class C(B):
 def __init__(self):
  super().__init__()
  self.m=1000;
```

```
  self.n=2000;
 def m3(self): #m1(), m2()
  print("Child-class C m3() Method",self.m,self.n);

obj = C();
obj.m1();
obj.m2();
obj.m3();


"""
3) Hierarchical Inheritance:-
= In this, single Parent-class gives derivation to multiple Child-classes
==Diagram==
   class A: (x,y,m1)
    |
    |
  ---------------------
  |     |
 class B(A):     class C(A):
 p,q,m2()/x,y,m1() m,n,m3()/x,y,m1()

obj1=B() ---> obj.x,y,m1,p,q,m2
obj2=C() ---> obj.x,y,m1,m,n,m3

"""
# Ex:-
#Program (HierarchicalInheritance.py)
#Prog to demo Hierarchical-Inheritance


class A:
 def __init__(self):
  self.x=10;
  self.y=20;
 def m1(self):
  print("Parent-class A m1() Method",self.x,self.y);
class B(A):
 def __init__(self):
  super().__init__() #x,y
  self.p=100;
  self.q=200;
 def m2(self): #m1()
  print("Child-class B m2() Method",self.p,self.q);
class C(A):
 def __init__(self):
  super().__init__() #x,y
  self.m=1000;
  self.n=2000;
 def m3(self): #m1()
  print("Child-class C m3() Method",self.m,self.n);

print("Using class-B obj1 :");
obj1 = B();
obj1.m1();
obj1.m2();
print()
print("Using class-C obj2 :");
obj2 = C();
obj2.m1();
obj2.m3();

"""
4) Multiple Inheritance:-
= In this, multiple Parent-classes gives derivation to single Child-class
==Diagram==
 class A: m1() class B: m2()
  \    /
   \     /
  class C(A,B): m3()[m1,m2]

# Ex:-
//Program (MultipleInheritance.py)
(Prog to demo Multiple-Inheritance)

***NOTE:-
= If same method-name/__init__() is inherited from both Parent-classes then 1st-parent-class method in order is considered or taken in Child-class

Ex:-
"""
#Prog to demo Multiple-Inheritance
#Program (MultipleInheritance.py)


#basic-case
class A:
 def __init__(self):
  self.x=10;
  self.y=20;
 def m1(self):
  print("Parent-class A m1() Method",self.x,self.y);
class B:
 def __init__(self):
  self.p=11;
  self.q=22;
 def m2(self):
  print("Parent-class B m2() Method",self.p,self.q);

class C(A,B): #A is 1st Parent-class & B is 2nd Parent-cls
 def __init__(self):
  super().__init__() #x,y but not p,q
  self.m=100;
  self.n=200;

 def m3(self):
  print("Child-class C m3() Method",self.m,self.n);


class D(B,A): #B is 1st Parent-class & A is 2nd Parent-cls
 def __init__(self):
  super().__init__() #p,q but not x,y
  self.r=111;
  self.s=222;
 def m4(self):
  print("Child-class D m4() Method",self.r,self.s);

print("Using class-C obj1");
obj1 = C();
obj1.m1();
#obj1.m2();
```

```
obj1.m3();

print()
print("Using class-D obj2");
obj2 = D();
#obj2.m1();
obj2.m2();
obj2.m4();




#sp-case in multiple-inheritance
#Prog to demo Multiple-Inheritance with same-method-names(m1()) in multiple-Parent-Classes(A,B)

class A:
 def m1(self):
  print("Parent-class A m1() Method");
class B:
 def m1(self):
  print("Parent-class B m1() Method");
class C(A,B):
 def m3(self): #1st-Parent-class(A) m1() only
  print("Child-class C m3() Method");
class D(B,A):
 def m4(self): #1st-Parent-class(B) m1() only
  print("Child-class D m4() Method");

obj1 = C();
obj1.m1();
obj1.m3();
print()
obj2 = D();
obj2.m1();
obj2.m4();



"""
NOTE:-
= In multiple-inheritance, we can access only 1st parent-class constructor at a time from child-class constructor
i.e,
 super().__init__()
(alternatively take accept1() & accept2() method in Parent-Classes to define instance-vars)

=*** whenever we have same-method-name in multiple parent-classes, in child-class it takes only 1st-parent method but not 2nd-parent-method


5) Hybrid Inheritance:-
= In this, it is combination of 2 or more single, multi-level, hierarchical, multiple inheritances
==Diagram==
    class A: [m1]
     |
    class B(A): [m2]m1
    /    \
   class C(B):   class D(B):
   m3[m1,m2]  m4[m1,m2]

A--->B (Single-Inheritance)
B--->C,D (Hierarchical-Inheritance)

Ex:-
"""
# //Program (HybridInheritance.py)
# (Prog to demo Hybrid-Inheritance)
#Prog to demo Hybrid-Inheritance
#Program (HybridInheritance.py)

class A:
 def m1(self):
  print("Class A m1() Method");
class B(A):
 def m2(self): #m1
  print("Class B m2() Method");
class C(B):
 def m3(self): #m1,m2
  print("Class C m3() Method");
class D(B):
 def m4(self): #m1,m2
  print("Class D m4() Method");

print("Using Class-C obj1 :");
obj1 = C();
obj1.m1();
obj1.m2();
obj1.m3();
print("Using Class-D obj2 :");
obj2 = D();
obj2.m1();
obj2.m2();
obj2.m4();

"""
6) Cyclic-Inheritance:-
= In this, inheritance is done in cyclic way
= It is not supported in Python (as it is not required)
Ex:-
class A(A):  #Here we get Name-Error ('A' is not defined)
 pass;
= inheritance to itself

Ex:- (A->B and vice-versa)
==Diagram==
class A(B):  #Here we get Name-Error ('B' is not defined)
 pass;
class B(A):
 pass;
= inheritance again-back to itself



(****sp-cases*****)
**==> super() function:-
(it is used in inheritance concept)
(Parent-class & Child-class)

=** It is pre-defined method, using which we can call/access parent-class constructors, vars and methods from child-class
(provided Parent-class & Child-class have same member-names)

=> CASE1::- (PC & CC const.&method same-names)
```

```
Ex1:-
#Program (SuperEx1.py)
NOTE:-
= Here using super(), we are accessing parent-class constructor and display() provided they have same-name
"""
#Program to demo super()
#Program (SuperEx1.py)

class A:  #1-const, x,y prop, 1-display(self)
 def __init__(self,x,y):
  self.x=x;
  self.y=y;
 def display(self):
  print(self.x);
  print(self.y);

class B(A):
 def __init__(self,x,y,p,q):
  super().__init__(x,y);
  self.p=p;
  self.q=q;
 def display(self):
  super().display();
  print(self.p);
  print(self.q);

bobj = B(10,20,100,200);
bobj.display();


"""
**CASE-2**
Ex2:-
#Program (SuperEx2.py)
NOTE:-
= Here super(), is used to call various members of Parent-class (instance/static/class)
"""
#Program (SuperEx2.py)
#Program to demo super() with P.C any members

class A:
 x=10;
 def __init__(self):
  self.y=20;
  print(self.y);
 def m1(self):
  print("Pclass-A m1() instance-method");
 @classmethod
 def m2(clsvar):
  print("Pclass-A m2() class-method");
 @staticmethod
 def m3():
  print("Pclass-A m3() static-method");

class B(A):
 def __init__(self):
  print(super().x);
  super().__init__();
  super().m1();
  super().m2();
  super().m3();

bobj = B();



"""
=> Case-3
=>Ex3:-
==> How to call method of a particular Parent-class:-
= In multiple-parent-classes, how to access particular parent-class members with same-name using super()
==Diagram===
 A m1()
 |
 |
 B m1()
 |
 |
 C m1()
 |
 |
 D m1()
 |
 |
 |
 E** m1(), m1(),m1(),m1(),m1()


= We use below cases,
1) super(D,self).m1();
= It calls m1() of Parent-class of class-D
2) A.m1(self);
= It calls m1() of Parent-class-A
=** in both the case self-var is compulsory
"""
# Ex:-
#Program (SuperEx3.py)
#Program (SuperEx3.py)

class A:
 def m1(self):
  print("Class-A m1()");
class B(A):
 def m1(self):
  print("Class-B m1()");
class C(B):
 def m1(self):
  print("Class-C m1()");
class D(C):
 def m1(self):
  print("Class-D m1()");
class E(D):
 def m1(self):
  print("Class-E m1()");
  A.m1(self);
  B.m1(self);
  C.m1(self);
  D.m1(self);
  print()
  super(C,self).m1();  #***
```

```
eobj = E();
eobj.m1();



# 25th day 25.22-08-22

"""
25.22-08-22
==>> Polymorphism in Python:-
(it is principle/feature of OOP)
= Poly means Many (2 or more)
= Morphism means Forms (behavior ---> methods/functions)

**Definition:-
 = Existence of single-object in multiple-forms based on a condition is called Polymorphism

Ex1: (H2O)
==Diagram==
  "H2O"
 => Liquid (water)
    => Solid  (ice)
 => Gaseous (Vapour/Stream)
(based on temperature(degrees))

Ex2: + operator can be used in 2-ways
==Diagram== (Concatenation & Numeric-Add)
10+20 ===> 30
"Hello"+"World" ===> "HelloWorld"

Ex3: * operator can be used in 2-ways
==Diagram== (Multiplication & Repetition)
10*20 ===> 200
"Hello"*3 ===> "HelloHelloHello"

NOTE:-
**=> Polymorphism in Python can be done in 3-ways:-
1) Duck-Typing Philosophy
2) Overloading
 a) Operator Overloading
 b) Method Overloading
 c) Constructor Overloading
3) Overriding
 a) Method Overriding
 b) Constructor Overriding



1) Duck-Typing Philosophy:-
Def:-
 = deciding method input-parameter data-type at runtime is called duck-typing

= In Python variables, we cannot specify the datatype explicitly
Ex:-
 a=10;
 a=5.6;
 a="Hello"
= Based on provided value at Runtime, dtype is taken into consideration
= Hence Python is dynamically typed Prog.Lang
Ex:-
(variables in methods as paramters)
 def f1(obj):
  obj.display();
(here which ever class object is passed as paramter, its coressponding display() is executed)

**= Here data-type of "obj" in f1(obj) is decided at Runtime, when we pass any type of class-object. This is called Duck-Typing
Ex:-
//Program (DuckTypingEx1.py)

NOTE:-
= Here we have a problem, if any object-Class does not contain talk() then we get "AttributeError"
Ex:-
//Program (DuckTypingEx1.py)
"""
##Program (DuckTypingEx1.py)
#Program to demo Duck-Typing

class Student:
 def display(self):
  print("Student-Details");
class Employee:
 def display(self):
  print("Employee-Details");
class Customer:
 def display(self):
  print("Customer-Details");

#single-func with diff-input-para(duck-typing)
def f1(obj):
 obj.display();

s1 = Student();
e1 = Employee();
c1 = Customer();

f1(s1);
f1(e1);
f1(c1);



"""

2) Overloading in Polymorphism:-
= It is done with 3-cases
 a) Operator Overloading
 b) Method Overloading
 c) Constructor Overloading

Ex1:- (+ Operator used in 2-ways)
print(10+20);   #Numeric-Addition
print("Sai"+"Ram");  #String-Concatenation

Ex2:- (* Operator used in 2-ways)
print(11*3);   #Numeric-Multiplication
print("Sai"*5);   #String-Repetition

Ex3:- (Bank deposit() )
```

```
deposit(cash);
deposit(cheque);
deposit(dd);

=> Overloading is done in 3-ways,
i) Operator-Overloading
ii) Method-Overloading
iii) Constructor-Overloading


i) Operator-Overloading:-
= We can use same operator for multiple-purposes
Ex1:- (+ Operator used in 2-ways)
print(10+20);   #Numeric-Addition
print("Sai"+"Ram");  #String-Concatenation

Ex2:- (* Operator used in 2-ways)
print(11*3);   #Numeric-Multiplication
print("Sai"*5);   #String-Repetition

Ex:-
##Program (OperatorOverloading1.py) #with objects

NOTE:-
=** Here we have to overload +operator to work with Book-objs (class-objs)
= For every operator, python directly supports "Magic-Methods"
(Magic-Methods can be used on class-objs Ex:- b1+b2 directly)
= For operator-overloading, we have to override(redefine) Magic-Methods in our class
= For +operator on class-objs, magic-method is __add__()

Ex:- (redefining __add__())
"""
# //Program (OperatorOverloading1.py)
#Program (OperatorOverloading1.py)

class Book:
 def __init__(self,pages):
  self.pages=pages;
 def display(self):
  print(self.pages);

b1 = Book(100);
b1.display();
b2 = Book(200);
b2.display();
#print(b1+b2); #TypeError
b3 = Book(b1.pages+b2.pages);
print(b3.pages);


#redefining __add__()
class Book:
 def __init__(self,pages):
  self.pages=pages;
 def display(self):
  print(self.pages);
 def __add__(self,other):
  return (self.pages+other.pages);
b1 = Book(100);
b1.display();
b2 = Book(200);
b2.display();
print(b1+b2); #NO-error (adding b1,b2 objs & auto __add__() executed)

"""
NOTE:-
= Below are different Magic-Methods for corresponding operators
Ex:- (self is 1st-object & other is 2nd-object)
1) + __add__(self,other);
2) - __sub__(self,other);
3) * __mul__(self,other);
4) / __div__(self,other);
5) % __mod__(self,other);
6) // __floordiv__(self,other);
7) ** __pow__(self,other);
-----------------------------------------------
8) +=  __iadd__(self,other); Ex: a+=b; (a=a+b)
9) -=  __isub__(self,other);
10) *=  __imul__(self,other);
11) /=  __idiv__(self,other);
12) %=  __imod__(self,other);
13) //=  __ifloordiv__(self,other);
14) **=  __ipow__(self,other);
-----------------------------------------------
15) <  __lt__(self,other);
16) <=  __le__(self,other);
17) >  __gt__(self,other);
18) >=  __ge__(self,other);
19) ==  __eq__(self,other);
20) !=  __ne__(self,other);


Ex:- (Overloading > and <= for Student-class objs)
//Program (OperatorOverloading2.py)

Ex:- (Overloading Multiplication Operator on Student-objs)
//Program (OperatorOverloading2.py)
#Program (OperatorOverloading2.py)
#OO to demo magic-methods speciality
"""

class Student:
 def __init__(self,name,marks):
  self.name= name;
  self.marks=marks;
 def __gt__(self,other):
  return (self.marks > other.marks);
 def __le__(self,other):
  return (self.marks <= other.marks);


s1 = Student("Sai",96);
s2 = Student("Ram",86);
print(s1>s2);       ##> <=
print(s1<=s2);

print()
#magic-combinations are automatic
print(s1<s2);
print(s1>=s2);
```

```
print()
print(s1==s2);
print(s1!=s2);


# NOTE:-
# = These sp-methods are called magic-methods b'coz in prev.example, we have defined only >, <= and other combinations are taken auto. w.o defining


"Assignment"
#Program (OperatorOverloading3.py)
#Overloading Multiplication Operator on Employee-objs
class Employee:
 def __init__(self,name,sal):
  self.name=name;
  self.sal=sal;
 def __mul__(self,other):
  return (self.sal * other.days);

class TimeSheet:
 def __init__(self,name,days):
  self.name=name;
  self.days=days;

e1 = Employee("Sai",1000);
t1 = TimeSheet("Sai",26);
print(e1.name,"Month Salary : ",e1*t1);

"""

(b) Method-Overloading:-
def:
 = 2 or more methods in same-class/prog with same-name but atleast 1-difference in method-signature
  =** No-of-args,
  =** Order-of-args,
  =** Dtype-of-args
Ex:-
 m1(int x):
 m1(float a):
 m1(int, float):
 m1(float,int):

NOTE:-(***)
**= But in Python, Method-Overloading is NOT-POSSIBLE
= Trying to declare multiple-methods with same-name & diff. in method-signature then Python-takes only last-method into considerations (Like variables)
Ex:-
a=10; (int)
a=5.6; (float)
a="Sai"; (str)
a=True; (bool) --> only a=True is considered

Ex:-
"""
# //Program (MethodOverloading1.py)
#Program (MethodOverloading1.py)
#Program to perform method-overloading indirectly


#general-case
class Demo:
 def m1(self):
  print("0-Args m1()");
 def m1(self,a):
  print("1-Args m1()");
 def m1(self,a,b): #lastest m1() is considered
  print("2-Args m1()");

obj = Demo();
#obj.m1();  #0-args
#obj.m1(11); #1-arg
obj.m1(11,22); #2-args
'''

'''
#case-1
#Method with Default-Args
class Demo:
 def sum(self,a=1,b=2,c=3):
   print("SUM(of 3 numbers) : ",(a+b+c));

obj = Demo();
obj.sum(1000,2000,3000);  #3-args is passed
obj.sum(100,200);    #2-args
obj.sum(10);     #1-arg
obj.sum();       #0-args


#case-2
#Method with Variable-len(No)-of-Args
class Demo:
 def sum(self,*nums): #here nums is tuple
  print("SUM : ",sum(nums));

obj = Demo();
obj.sum(100,200,300);  #3-args
obj.sum(10,20);    #2-args
obj.sum(1);     #1-arg
obj.sum();     #0-args


"""
NOTE:-
=> How to handle Method-Overloading in Python?
= For this we use Method with Default-Args (or) Method with Variable Number of Args

=> Method with Default-Args:-
=** Default-Args means while declaring the method, we give default values to input-parameters/args
Ex:-
 def sum(a=10,b=20,c=30):
= whenever method is called with less no.of.para or no-para then default-values of args are taken into consideration

Ex:- (Method with Default-Args)
//Program (MethodOverloading1.py)

=> Var-len-Args to Method:-
(Variable-Length-Arguments)
= For any method we can pass last-para as variable-args
= It means for that last-para, we can pass 0 or more args/values to call such method
= It is done with *varName (it can accept 0 to more args/values as input)
= compulsory it should be last-para in method o.w error
```

```
Ex:- (Method with Variable-No-of-Args)
//Program (MethodOverloading1.py)



C)
==> Constructor-Overloading:-
= It is not possible in Python
= Here also, if we define multiple-constructors with same-name and atleast 1-difference in constructor-signature
 (No-of-args,
 Order-of-args,
 Dtype-of-args)
Ex:-
 def __init__(self):  #0-args
 def __init__(self,a): #1-arg
 def __init__(self,a,b): #2-args

=> def:-
 multiple-constructors with same-name in same-class & atleast 1-diff in constructor-signature

= In such case, last constructor is taken into consideration like methods/variable
Ex:-
Ex:-
a=10; (int)
a=5.6; (float)
a="Sai"; (str)
a=True; (bool)

Ex:-
"""
# //Program (ConstructorOverloading1.py)
#Program (ConstructorOverloading1.py)
#Program to demo Const-Over indirectly in 2-cases


#no Const-Over
class Demo:
 def __init__(self):
  print("0-Args Constructor");
 def __init__(self,x):
  print("1-Args Constructor");
 def __init__(self,x,y): #only lastest-const is taken
  print("2-Args Constructor");

#obj1 = Demo();         #0-args-const
#obj2 = Demo(11);        #1-args-const
obj3 = Demo(111,222);  #2-args-const
'''


'''
#case-1
print()
#Constructor with Default-Args
class Demo:
 def __init__(self,a=1,b=2,c=3):
  print("sum :: ",a+b+c)

obj1 = Demo();  #0-args
obj2 = Demo(11); #1-arg
obj3 = Demo(111,222); #2-args
obj4 = Demo(1111,2222,3333); #3-args



#case-2
print()
#Constructor with Variable-No-of-Args
class Demo:
 def __init__(self,*nums): #nums is tuple(dyc-size)
  print("Sum ::",sum(nums))

obj1 = Demo();   #0-args
obj2 = Demo(11);  #1-arg
obj3 = Demo(111,222); #2-args
obj4 = Demo(1111,2222,3333); #3-args



"""
NOTE:-
= However, we can make constructor-overloading possible using Constructor with Default-Args & Constructor with Variable-No-of-Args

=> Constructor with Default-Args:-
=** Default-Args means while declaring a constructor, we give default values to input-parameters/args
= when such constructor is called with less no.of.para or no-para then default-values of args are taken into consideration
Ex:-
def __init__(self,a=10,b=20,c=30):

Ex:- (Constructor with Default-Args)
//Program (ConstructorOverloading1.py)

=> Var-len-Args to Constructor:-
= For any constructor, we can pass last-para as variable-args
= It means for that last-para, we can pass 0 or more args/values to call such method
= It is done with *varName (it can accept 0 to more args/values as input)
= compulsory it should be last-para in constructor o.w error

Ex:- (Constructor with Variable-No-of-Args)
//Program (ConstructorOverloading1.py)

"""



# 26th day 26.23-08-22

"""
3) Overriding in Python-Polymorphism:-
(related to Inheritance i.e, PC & CC)
= It is done in 2-ways,
i) Method-Overriding
ii) Constructor-Overriding


i) Method-Overriding:- (Re-placing/Re-Writing)
= This concept is related to Inheritance (BC & SC)
= Parent-class members are available to Child-class
```

```
Def:-
 Re-defining Parent-class methods in Child-class with same-name & same-signature is called Method-Overriding
(here no-of-args, dtype-of-args, order-of-args SAME)
= Hence, always Child-class methods are more powerful than Parent-class methods

Ex:-
"""
# //Program (MethodOverriding1.py)
#Program (MethodOverriding1.py)
#Program to demo Method-Overriding in Inheritance

class Pclass:
 def m1(self): #0-args
  print("Parent-class m1() method");

class Cclass(Pclass):
 def m1(self): #0-args redefined
  print("Child-class powerful m1() method");
  super().m1();

obj = Cclass();
obj.m1();


# NOTE:-
# = From Child-class Overriding methods, we can call Parent-class Overridden methods using super() method

"""
ii) Constructor-Overriding:- (re-placing/re-writing in SC)
= This concept is related to Inheritance (BC & SC)
= Parent-class members are available to Child-class

Def:-
 Re-defining Parent-class constructor in Child-class with same-name & same-signature is called Constructor-Overriding
(here no-of-args, dtype-of-args, order-of-args SAME)

= Here we redefine powerful-constructor in sub-class

Ex:-
//Program (ConstructorOverriding1.py)
#Program (ConstructorOverriding1.py)

class Pclass:
 def __init__(self):     #0-args
  print("Parent-class Constructor");

class Cclass(Pclass):
 pass;
 #def __init__(self):     #0-args
 # print("Child-class Constructor");
 # super().__init__();

obj = Cclass();


NOTE:-
= If Child-class does not have constructor then Parent-class constructor is executed
= Also from Child-class constructor, we can call Parent-class constructor using super()


=====================================================
***Abstraction***
------------------------
def:-
 Hiding unncecessary things in an object & exposing only necessary things outside the object

= In python, abstraction is done in 2-ways,
 1) abstract-classes (with abstract-methods)
 2) interface (with abstract-methods)

1) Abstract-Methods in Python-Programming:-
(These methods are used inside a class)
= Method with a body or implementation is called as Complete or Concrete-Method
Ex:-
 def m1(self): #complete-method
  ....
  ....
=** Methods without a body(pass) or implementation is called as Abstract-Methods
Ex:-
 def m2(self): #in-complete-method
  pass;
= Abstract-Methods have only-declaration but no-definition or implementation or body
= Such methods are decorated with "@abstractmethod"
Ex:-
 @abstractmethod --> it is decorator ("abc" module)
 def m2(self):
  pass;
(here pass means body W.O definition)

= @abstractmethod decorator is in "abc" module
(import it)
(abstract base class)

Ex:-
from abc import *;
class Demo:
 @abstractmethod
 def m1(self):
  pass;
(here @abstractmethod decorator & pass for abstract-method are compulsory)


==> Abstract-Classes:-
= Class which is not complete is called as Abstract-Class (In-concrete-class)
= Abstract-Classes are inherited from "ABC" class of abc-module

Def:-
 Abstract-Classes are classes with atleast-1 absract-method
(Abstract-Classes object cannot be created because they are in-complete-classes)
Ex:-
from abc import *;
class Demo(ABC):
 @absractmethod
 def m1(self):
  pass;
obj = Demo(); #error

(Abstract-Classes should be inherited from "ABC" & they should have atleast 1 abstract-method)
```

```
=** Final-NOTE:-
1)
If a class contains "atleast 1 abstract-method" and is "inherited from ABC" then only it is Abstract-Class (Which cannot be instantiated i.e, object cannot be created for abstract-clas


=>>** How to make use of Abstract-Classes??
***= To use Abstract-Class, we have go for child-class(Inheritance)
= in child-class we have to re-define(method-overriding) all the abstract-methods of Parent-class
(then only child-class becomes complete-class & its object can be created)

***= Parent-class abstract-methods should be implemented or re-defined in Child-class o.w Child-class also becomes abstract-class & its object cannot be created

==Diagarm
 Incomplete-class(Abstract-Class) (obj-NOT-ok)
  |
  |
 <<inheritance>>
  |
  |
 Complete-Child-class (obj->ok)


=** Abstract-class(abc-module,ABC(pc),@absractmethod) & Complete-Child-class(Compulsory) -> redefine all abs-methods

Ex:-
"""
# //Program (AbstractClassEx1.py)
#Program (AbstractClassEx1.py)
#Program to work with abstract-classes & abstract-methods

from abc import *;
class A(ABC):
 @abstractmethod
 def m1(self):
  pass;
 def m2(self):
  print("m2() complete-method of abstract-class")

#child-class(inheritance)
class B(A):  #complete-class
    #m1(),m2()
 def m1(self):   #redefine or overriding of m1()
  print("m1() Redefined in Child-class B");
 def m3(self):
  print("m3() own-method in Child-class B");

#child-class(inheritance)
class C(A):  #complete-class
    #m1(),m2()
 def m1(self):   #redefine or overriding of m1()
  print("m1() Redefined in Child-class C");
 def m4(self):
  print("m4() own-method in Child-class C");


#abstract-class object
#obj = A();


#object of complete-child-class-B
obj1 = B();   #Complete-class
obj1.m1();
obj1.m2();
obj1.m3()


print()
#object of complete-child-class-C
obj2 = C();   #Complete-class
obj2.m1();
obj2.m2();
obj2.m4();

# NOTE:-
# = Abstract-Class may contain complete-methods but atleast 1-abstract-method

"""
=====================================================
3) Interfaces in Python:-(**)
= Complete-class contains all complete-methods (Obj can be created)
= Abstract-Class contains atleast-1 abstract-method (but Obj cannot be created)
= Interface(Pure-Abstract-Class) contains all methods as abstract-methods (Obj cannot be created)

def:-
 Inheritance is a abstract-class with all absract-methods in it compulsory

==** here also we use "ABC" and @abstractmethod decorator form abc-module
= Interfaces are used with child-classes same like abstract-classes
(in child-class re-define all abstract-methods & child-class becomes complete-class & finally its object can be created)
"""
# Ex:-
#Program (InterfaceEx1.py)
#Program (InterfaceEx1.py)

from abc import *;
class A(ABC):
 @abstractmethod
 def m1(self):
  pass;
 @abstractmethod
 def m2(self):
  pass;

class B(A):
 def m1(self):
  print("m1() Redefined in Child-class B");
 def m2(self):
  print("m2() Redefined in Child-class B");
 def m3(self):
  print("m3() own-method in Child-class B");

class C(A):
 def m1(self):
  print("m1() Redefined in Child-class C");
 def m2(self):
  print("m2() Redefined in Child-class C");
 def m4(self):
  print("m4() own-method in Child-class C");
```

```
#interface-obj
#obj = A();

#obj1 of complete child-class-B
obj1 = B();
obj1.m1();
obj1.m2();
obj1.m3();
print()
#obj2 of complete child-class-C
obj2 = C();
obj2.m1();
obj2.m2();
obj2.m4();

"""
=> Real-time Usage of Abstraction***
(Project-Design phase)
  Interfaces
   |
   |
  Abstract-Classes
   |
   |
   |
  Complete-class(child)
 (objs are created... and used in real-time)



==> Interface v/s Abstract-Class v/s Complete-class:-
1) If we dont know anything about functions & its implementations then we go for Interfaces (Only Specs are Available)
2) If we require only Partial Implementation of functions in class then we go for Abstract-Classes
3) If we require complete-implementation of class with objects then we go for Complete-classes
"""




# 27th day 27.24-08-22

"""
******(Access-Specifier in Python)**********
(Access-Modifiers)
(used inside a class for its members(vars+methods))
(Scope of access/level of access)
= they are 3-types,
 a) public  (a,m1())    #no-underscore
 b) protected (_a,_m1())  #single-underscore(begin)
 c) private (__a,__m1())  #double-underscore(begin)

= Access-Specifier means where we can access class-members in python program
(class-members => data-prop(vars) & methods)

= For this, we have 4-levels-of-access,
i.e, Same-class-access
  Sub-class-access
  Other-class-access
  Outside the class-access(main-prog)
  ----------------------------
  Outside the module-access(**)
==Diagram==



==> Public, Protected, Private Attributes:-
(these attributes are internally defined in python)
a) Public:-
= By default all attributes(data-prop or vars or methods) are Public in python-class
= Public attributes(data-prop or vars or methods) of python-class can be accessed from anywhere (inside or outside a class) i.e, Full-access
i.e, Same-class/Sub-class/Other-class/Outside the class
Ex:-
name="Sai";  #name-var and m1() are public by default
def m1(self):
 pass;

b) Protected:-
= Protected attributes(data-prop or vars or methods) of a class can be accessed inside Same-class (or Same-File) and only its Child-classes (Other classes of Same-File)
i.e, Same-class/Sub-class/Other-class/Outside the class
= It is prefixed with _ symbol
Ex:-
_name="Sai";
def _m1(self):
 pass;
(It is just notation but such protected-attributes does not exists in python class)

c) Private:-
= Private attributes(data-prop or vars or methods) of a python-class can be accessed only inside Same-class and not outside the class
i.e, Same-class access
= It is prefixed with two __ symbol (double-underscore)
Ex:-
__name="Sai";
def __m1(self):
 pass;

Ex:-
#Program (PubProPriEx1.py)
(Program to demo Public, Protected, Private Attributes)

NOTE:-
= Basically we have only 2-access-modifers (public/private)
(protected is just a notation)
"""
# //Program...
#Program to demo Public, Protected, Private Attributes)
#Program to demo Public, Protected, Private Attributes)
#Program (PubProPriEx1.py)

class Demo:
 a=10;  #public-var
 _b=20;  #protected-var
 __c=30;  #private-var
 def m1(self): #Same-class all are accessible
  print("Inside Same-class");
  print(Demo.a);
  print(Demo._b);
  print(Demo.__c);
```

```
#same-class-access
obj = Demo();
obj.m1();
'''

'''
print()
#in sub-class-access
class Demo1(Demo):
 def m2(self):
  print("Inside Sub-class-access");
  print(Demo.a);
  print(Demo._b);
  #print(Demo.__c);

obj1 = Demo1();
obj1.m2();
'''

'''
print()
#in other-class (same as out-side class access)
class Demo2:
 def m3(self):
  print("Inside Other-class-access");
  print(Demo.a);
  print(Demo._b);
  #print(Demo.__c);

obj2 = Demo2();
obj2.m3();
'''

'''
print()
#out-side the class-access(main-program __main__())
print("Outside-class-acess in main-program");
print(Demo.a);
print(Demo._b);
#print(Demo.__c); #private-mem


"""
==> Assignment::-
=**WAP to demo access-modifers(pub/pro/pri) of a class, accessing from other-modules(.py files)
(use import statement)
#PubProPriEx2.py
-------------------
from PubProPriEx1 import Demo;
#main-prog
print("Other-Module Access)
print(Demo.a);
print(Demo._b);
#print(Demo.__c); #private-mem



(=**sp-case**)
==> How to access Private-Vars outside the class:-
= Private-Vars cannot be accessed directly outside of a class
= It is accessed indirectly as follows,

Syntax:-
---------
objrefvar._classname__privatevarname/privatemethodname()
"""

# Ex:-
#Program (PubProPriEx3.py)
#Program (PubProPriEx3.py)
#Program to demo private-member access outside the class


class Demo3:
 __c=11;
 def __init__(self):
  self.__x=10;
 def __m3(self):
  print("Private __m3() method");


obj3 = Demo3();
#print(obj3.__x);  #error

print("Private-member outside access");
print(obj3._Demo3__x);
print(obj3._Demo3__c);
obj3._Demo3__m3();


"""
==> Assignment::-
=**WAP to demo access-modifers(pri with sp-case) of a class, accessing from other-modules(.py files)
(use import statement)
#PubProPriEx4.py
-------------------
from PubProPriEx3 import Demo;
obj4 = Demo3()
print("Private-member outside module-access");
print(obj4._Demo3__x);
print(obj4._Demo3__c);
obj4._Demo3__m3();


-----------------------------------------------
==> __str__() sp-method of class:-
(pre-defined magic-method)
= Whenever we print any Obj.Ref, internally __str__() is called
Ex:-
 print(s1)
 print(obj1)
= It returns or gives string-value
Ex:
<__main__.classname object at 0x1234AB0> #it prints object address

= This string-value is little bit confusion
= For easy understanding format, we override this method in our class
(re-define the same method in our-class with own-definition)
```

```
"""
# Ex:-
#Program (StrMethod.py)
#Program (StrMethod.py)
#Program to demo __str__() method in our class


class Student:
 def __init__(self,rollno,name):
  self.rollno=rollno;
  self.name=name;
 def __str__(self): #re-define in our class
  ss="Student-Data"+"\t"+str(self.rollno)+"\t"+self.name;
  return ss;

s1 = Student(1001,"Sai");
s2 = Student(1002,"Ram");
print(s1);
print(s2);
print(id(s1))
print(id(s2))


# NOTE:-
# = help(modulename/classname) gives complete description of that class or module



"""
============================================================
***==> Exception-Handling in Python:-
(Introduction)
= This concept is related to OOP(real-life-situations)
def:-
 Exception means runtime-error
 (Error which occurs during execution of a program)

***Runtime-Errors occur when end-user is using our-app or s/w.
(they occur b'coz end-userby mistake gives wrong input)

Ex:-
 "division program"
case-1:-
 a=10,b=2  #correct-input
 c=a/b;
 print(c) ##10/2-->5 (proper-output)
case-2:-
 a=10,b=0  #wrong-input by mistake
 c=a/b;   #10/0
 print(c) ##runtime-error (no-proper-output)

==** When Runtime-Errors occur, our program execution-stops there only(Abnormal-Termination of Prog)


=** Basically we have 2-types of Errors in python-program,
a) Syntax-Errors
b) Runtime-Errors

a) Syntax-Errors:- (devlopment-time errors))
= These errors occur due to Invalid-syntax or Wrong syntaxes in program
Ex1:-
x=10;
if x==10  #Syntax-Error : is missing
 print("Hello");

Ex2:-
print "Hello"; #Syntax-Error () Missing Parenthesis

NOTE:-
= Once program does not have any syntax-error(dev-time-errors) then program executes completely o.w program does not execute completely
= Generally, we get syntax errors during development-time


2) Runtime-Errors:-
= These errors occur while executing the program due to improper input given by user (or logics or memory-problems)
= Also called as Exceptions
Ex:-
print(10/0); #ZeroDivisionError
print(10/"ten"); #TypeError (int/string)
a = int(input("Enter any Number : "));
#if input is "ten" then we get "ValueError"
("ten" cannot be converted to int)

NOTE:-
= Syntax-Errors can be fixed at development-time
only
= But Runtime-Errors(Exception) occurs & we get Abnormal-Termination of Program
(break-down of the program)
(program stops executing at that particular line & remaining lines of program are not executed)



=> About Exceptions:-
= It is any unexpected-error during execution time of program and leads to Abnormal-Termination of Prog
Ex: (Some Pre-defined Exceptions)
 = ZeroDivisionError
 = TypeError
 = ValueError
 = FileNotFoundError
 = EOFError
 = SleepingError
 = InsufficientFundsError (User-Defined)

NOTE:-
= If we handle Runtime-Errors(Exceptions) then we get Proper & Complete Execution of Program
(Normal-Execution of Program)

=> DEFINITION:-
=** Exception-Handling is a mechanism of detecting runtime errors and providing with proper alternate solution
(Detect Runtime-Errors -------> Provide-Solution)

Ex:-
= Accessing Gmail from India-Server and if india-server is down then immediately Google will provide gmail-service from US-Backup-Server
(Always Online and NO-Breakdown)
"""
```

```
# 28th day 28.26-08-22

"""
***How to Handle Exceptions??***
==> How to do Exception-Handling in Program?
= Every exception is an object in Python with some corresponding class
= Exception-obj is created automatically by PVM when it occurs
= PVM searches for Exception-Handling code in prog and if not there then Interpreter terminated prog-exec-abnormally
= PVM also prints corresponding Exception-Info also
= Finally, rest of program is Not-Executed (Abnormal-Termination)

Ex:-
//program..."""
#(ExceptionEx1.py)
#Program to generate exception (ExceptionEx1.py)
#Abnormal-Termination of Program

print("Division Program in Python");
a = int(input("Enter A : "));
b = int(input("Enter B : "));
c = a/b;
print(c);
print("End of Program");
"""
NOTE:-
= In python, Every exception is an object, it has 3-properties,
a) Exception-Type (Exception-Classname)
b) Exception-Message (division by zero)
c) Exception-State (line-no,prog-name,module-name)
(Except-object is automatically created by PVM***)


(Pre-defined Exception-classes)
==> Inbuilt-Exception classes in Python:-
(Hierarchy --> B.C & its S.C)
==Diagram==(refer-notes)

=> BaseException
 = Exception
  - AttributeError
   * ZeroDivisionError
   * FloatingPointError
   * OverFlowError
  - ArithmeticError
  - EOFError
  - NameError
  - LookupError
   * IndexError
   * KeyError
  - OSError
   * FileNotFoundError
   * InterruptedError
   * PermissionError
   * TimeOutError
  - TypeError
  - ValueError
 = SystemExit
 = GeneratorExit
 = KeyboardInterrupt
NOTE:-
= Every exception is a class
= "BaseException" is top-level Parent-class (Root) and remaining all are Child-classes
= "Exception" class and it child-class are important for Exception-Handling mechanism



==> Steps for Exception-Handling:-
=> How to Handle Exception in Program?
= It is done with try-except (indented block-of-code)

Step-1)
 = Monitor or Detect the exception in prog
 = It is done with try-block(header & suite)
Syntax:-
 try:
  ......
  .code.
  ......

Step2)
 = When exception-occurs, raise it (alert or warning)
 = it is done with raise-stmt
Syntax:-
 try:
  ...
  c=a/b;  #raise ZeroDivisionError();
  ....
*** All pre-defined exceptions are automatically created(obj) & raised by PVM

Step3:-
 = Now accept the exception occured in try-block
 = it is done with except-block(header & Suite)
Syntax:-
 try:
  ...
  c=a/b;  #raise ZeroDivisionError();
  ....
 except ZeroDivisionError:
  ........
  ..body..
  ........

**= for except-block, pass Exception-Classname(type)
(use except-Block immediately after try-block)

Step4:-
 = Provide proper-solution in body of except-Block


NOTE:-
= Hence Exception-Handling mechanism is done successfully & NO Abnormal-Termination of Prog(normal-execution)

"""
# Ex:-
#Program (ExceptionEx2.py)
# (Program to generate exception & handle exception)
#Program to generate exception & handle exception
#ExceptionEx2.py
```

```
print("Division Program in Python");
a = int(input("Enter A : "));
b = int(input("Enter B : "));

try:
 c = a/b;
 print(c);
except ZeroDivisionError:
 print("Division by 0 NOT OK");
print("End of Program");

"""
NOTE:-
1) Where there is no-exception in try-block then except-block is not executed and remaining stmts after except-block are executed for normal-execution
2) Where there is exception in try-block then except-block is executed and remaining stmts after except-block are executed for normal-execution
3) When exception occurs in try-block, it skips remaining stmts in try-block and control goes to immediate except-block
4) When exception-occurs in try-block and there is no matching except-block then it leads to Abnormal-Termination of Prog
5) If exception-occurs in except-block or remaining stmts after except-block then it is abnormal-termination of prog


==> Printing Pre-defined Exception Information:-
= It means exception-object 3-properties
(exception-type/msg/state)

Ex:-"""
#Program (ExceptionEx3.py)
# (Program to display exception-info)
#Program to display exception-info
#Program (ExceptionEx3.py)

print("Division Program in Python");
a = int(input("Enter A : "));
b = int(input("Enter B : "));
try:
 c = a/b;
 print(c);
except ZeroDivisionError as msg:
 print("Division by 0 NOT OK");
 print("Exception-Message :",msg);

print("End of Program");


"""
==> try with multiple-except blocks:-
= For single try-block, we can  have multiple except-blocks
= Depending on exception-type in try-block, corresponding except-block is executed
Ex:-
try:
 ...
 ...
 ...
except Type1:
 ...
except Type2:
 ...
except Type3:
 ...
............
............
"""
#Program (ExceptionEx4.py)
# (Program to demo multiple-except-blocks)
#Program to demo multiple-except-blocks

print("Division Program in Python");
try:
 a = int(input("Enter A : "));
 b = int(input("Enter B : "));
 c = a/b;
 print(c);
except ZeroDivisionError:
 print("Division by 0 NOT OK");
except ValueError:
 print("Provide Proper int-value");
print("End of Program");

"""
NOTE:-
= In multiple-except-blocks, 1st exception raised from try-block is executed by corresponding except-block
= And also only 1 exception is processed at a time (i.e, 1st raised exception) [1st come, 1st server]
= If in multiple-except-blocks matching except-block is not present then it leads to Abnormal-Termination of Prog


==> Single except-block handling multiple-exceptions:-
= Single except-block can handle multiple-exceptions as follows,
Ex:-
except(Exception1,Exception2,...):
(or)
except(Exception1,Exception2,...) as msg:

=* parenthesis() are mandatory
= group of exceptions are taken as tuple
"""
# Ex:-
#Program (ExceptionEx5.py)
# (Program to demo single-except-block with multi-exceptions)
#Program to demo single-except-block with multi-exceptions
#ExceptionEx5.py

print("Division Program in Python");
try:
 a = int(input("Enter A : "));
 b = int(input("Enter B : "));
 c = a/b;
 print(c);
except (ZeroDivisionError, ValueError) as msg:
 print("Exception : ",msg);
print("End of Program");

"""

==> Default except-Block:-
= It is used to handle any type of exception
= Mainly used when we dont know type of exception in try-block and displaying general-exception-msgs

Syntax:-
except:  #do not give any Classname
 stmts;
```

```
=** Use it at the end of all the except-blocks (o.w syntax-error)

Ex:-"""
#Program (ExceptionEx6.py)
# (Program to demo default-except-block)
#Program to demo default-except-block

print("Division Program in Python");
try:
 a = int(input("Enter A : "));
 b = int(input("Enter B : "));
 c = a/b;
 print(c);
#except:
 #print("Default-Except-Block : Unknown Exception occurred");
except ZeroDivisionError:
 print("Division by 0 NOT-OK");
except:
 print("Default-Except-Block : Unknown Exception occurred");
print("End of Program");

"""
NOTE:-
= Possible except-blocks definitions,
1) except ZeroDivisionError:
2) except ZeroDivisionError as msg:
3) except (ZeroDivisionError,ValueError):
4) except (ZeroDivisionError,ValueError) as msg:
5) except:



==>finally block:-
= finally-block is used at the end of all the except-blocks
= This block is executed whether exception is raised or not in try-block
**= Its main purpose is cleaning up resources in program at the end of all the except-blocks
(Resource Deallocating or Releasing Codes)
Syntax:-
try:
 ....
except:
 ....
except:
 ....
...
...
finally:
 ....
=* It is executed whether exception occurs or not (OR) exception is handled or not

Ex:-"""
#Program (ExceptionEx7.py)
# (Program to demo finally-block)
#Program to demo finally-block

print("Division Program in Python");
try:
 a = int(input("Enter A : "));
 b = int(input("Enter B : "));
 c = a/b;
 print(c);
#except ZeroDivisionError:
except NameError:
 print("Division by 0 NOT OK");
except ValueError:
 print("Provide Proper int-value");
finally:
 print("Finally Block is executed");

print("End of Program");

"""
NOTE:-
=** it is executed if exception is handled or not(abnormal-termination also)

NOTE:-(Sp.case of finally-block)
= Only 1 situation where finally block is not-executed
i.e, os._exit(0) function
(0 is normal-termination of Py-Prog)
(1 is abnormal-termination of Py-Prog)
= This function shuts down PVM by OS
(_exit(0) is in os-module)
"""
# Ex:-
#Program (ExceptionEx8.py)
#Program (ExceptionEx8.py)
#Program to demo finally-block not executed sp-case

import os;
try:
 print("try-block");
 os._exit(0);
except ValueError:
 print("ValueError");
except:
 print("Unknown-Exception");
finally:
 print("Finally-block executed");

print("End of the Program");



# 29th day 29.27-08-22

"""
==> Nested try-except-finally block:-
(Nested-Exceptions)
(Nested try-blocks)
= We can have try-except-finally inside any other try/except/finally blocks
Ex:-
try:
 ...
 try-except-finally
 ...
except:
 ...
 try-except-finally
 ...
finally:
```

```
 ...
 try-except-finally
 ...

NOTE:-
***Advantage is,
 = Inner try-block exceptions are handled by inner-except-Block o.w they can be handled by outer-except-Block also
"""

# Ex:-
#Program (ExceptionEx9.py)
# (Program to demo nested try-blocks)
#Program to demo nested try-blocks
#Program (ExceptionEx9.py)

print("Division Program in Python");
try:
 print("Outer try-block");
 a = int(input("Enter A : "));
 b = int(input("Enter B : "));
 try:
  print("Inner try-block");
  c = a/b;
  print(c);
 except NameError:
  print("Inner except-block");
  print("NameError occurred");
 finally:
  print("Inner finally-block");
except ZeroDivisionError:
 print("Outer except-block");
 print("Division by 0 NOT OK");
except ValueError:
 print("Outer except-block");
 print("Provide Proper int-value");
finally:
 print("Outer Finally Block is executed");

print("End of Program");
"""


NOTE:-
= If control enter try-block (inner/outer) its corresponding finally-block is executed (whether exception is raised or not)

NOTE:-
= Exception occurred in finally-block is Abnormal-Termination of Program



==> else Block with try-except-finally blocks:-
(else block with exceptions)
= We can use else block with try-except-finally blocks
= It is executed only if there is NO-exception inside try-block
Ex:-
try:
 ...
except:
 ... (executed if exception occurs in try-block)
else:
 ... (executed if no-exception occurs in try-block)
finally:
 ... (executed if exception is there or not in try-block or handled or not in except-block)

=**else-block is used before finally-block
"""

# Ex:-
#Program (ExceptionEx10.py)
#Program (ExceptionEx10.py)
#Program to demo else-block in exceptions
try:
 print("try-block");
 a = int(input("Enter A : "));
 b = int(input("Enter B : "));
 c = a/b;
 print(c);
except:
 print("Div by 0 NOT-OK or Any other Exception");
else:
 print("Else-Block executed for NO-Exception");
finally:
 print("Finally-Block");


"""

(Our-own Exceptions)
==> Types of Exception:-
= There are 2-types of exceptions,
1) Pre-defined Exceptions
2) User-defined Exceptions(**)

1) Pre-defined Exceptions:-
= Also called Inbuilt-Exceptions
= They are automatically raised by PVM when occurs in prog
Ex1:-
= When 10/0 is performed in prog then ZeroDivisionError is automatically raised by PVM
Ex2:-
= When int("ten") is done in prog then ValueError is automatically raised by PVM

2) User-Defined Exceptions:-
= Also called Customized Exceptions or Programatic Exceptions
= These exceptions are defined & raised by programmer as per requirement in program/coding
= "raise" keyword is used in try-block to raise UD-Exceptions
Ex:-
raise InsufficientFundsException
raise InvalidPinException
etc

==> How to define and work with UD-Exceptions?
Step1:-
= Inherit our UD-Exception class from pre-defined Exception base-class
(Child-class is a kind of Parent-class)

Syntax:-
class UDExceptionClassname(Predefined-Exception-Class):
 def __init__(self,arg):
  self.msg=arg;
Ex:-
```

```
class InsufficientFundsException(Exception):
 def __init__(self,arg):
  self.msg=arg;
  #here self.msg becomes our UD-Exception Message
(Now our class can participate in Exception-Handling-Mechanism)

Step2:-
(use raise-stmt to raise an exception, when it occurs)
=** When UD exception occurs, raise it as follows,
Ex:-
 raise InsufficientFundsException("any-message");

=**here our exception-class-obj is created & its constructor is exec.automatically(__init__())


Step3:-
=** accept your exception in except-Block & give proper-solution
"""



# Ex:-
#Program (ExceptionEx11.py)
#Program to demo UD Exception

import time;

class InsufficientFundsException(Exception):
 def __init__(self,msg):
  self.msg=msg; #this is our UD-Exception Message

#main-prog
print("ATM Transaction");
print("*********************************");
acbal = 5000;
print("Initial-Balance ::",acbal);
wamt=int(input("Enter Withdraw Amount : "));  #6000
time.sleep(5)
try:
 if wamt > acbal:
  raise InsufficientFundsException("Less Funds in Account...");
 else:
  acbal = acbal - wamt;
 print("Balance after Successful-Transaction : ",acbal);
except InsufficientFundsException as msg:
 print(msg)
 print("Transaction NOT possible!!!");


time.sleep(5)
print("*********************************");
print("Account Balance : ",acbal);
print("End of the ATM Transaction");

"""

NOTE:-
= For UD-Exceptions also, objects created in raise stmt
= For this object also we have 3-properties
(Exception-class-type/Message/Type)
"""


"Assignment"
##WAP to perform ATM transaction using "InvalidPinException"
#Program (ExceptionEx12.py)




# 30th day 30.29-08-22

"""
==>>> Files-Handling in Python:-
def:-
 = A computer-file is a collection of data, which is stored permanently on a disk using filename
Ex:
 abc.txt (text-files)
 Resume.doc (Doc-files)
 myoffice.ppt
 etc...

NOTE:-
**= The data which is processed in program is temporary
Ex:-
 a=10;
 b=20;
 print(a+b);
= Once execution is done it is deleted from memory
=** To make it permanent for future references, we use files
(program-data is stored in files and used for next-executions)
Ex:-
 Student-data
 Employee-data
 Aadhar-data
 etc..

=> Types of Files:-
= There are 2-types of files used for python-programming
1) Text-Files
= Stores data in the form of characters
Ex:- "Hello, Welcome, bye"
2) Binary-Files
= Stores data in the form of binary-format (0,1) like images,audio,video files, setup-files etc



==> Steps(3)::-
1) Open the file
2) Perform Operation on File (R/W/A)
3) Close the file


==> Opening a file in Program:-
= For this we use open() function
= While opening a file, we have to specify file-opening-mode
Syntax:-
 fl = open(filename,filemode);
Ex:-
 fl = open("abc.txt","r");
```

```
==Diagram==
 [fl]------->[Org-file]"abc.txt"


==> File-Modes::-(7-types)
=**Commonly used file-modes are,
1) r
= Opens a file for read-mode (Reading-Purpose)
= It is default mode
= Inside the file, file-pointer(cursor) is placed at beginning
= If file is not exists then we get "FileNotFoundError" (exception)

2) w
= Opens a file for write-mode (Writing-Purpose)
= If file already there then it will over-write the file
= If file Not-there then it will create a new file

3) a
= Opens an existing file for append-mode (adding data from end-of-file)
= Inside the file, file-pointer(cursor) is placed at end for appending
= If file is Not-there then it will create a new file

4) r+
= Opens a file for read & write-data (Both-Purpose)
= Previous data in file is not deleted or not over-written
= File-Pointer (cursor) is placed at beginning of file

5) w+
= Opens a file for write & read-data (Both-Purpose)
= Previous data in file is over-written
= File-Pointer (cursor) is placed at beginning of file

6) a+
= Opens a file for append & read-data (Both-Purpose)
= Previous data in file is NOT over-written
= File-Pointer (cursor) is placed at end of file

7) x (exclusive-write-mode)
= Opens a file in exclusion creation mode for write-operation
= If file already there then we get FileExistsError (exception) no over-writing

NOTE:-
= All modes are applicable for text-files
= Add or suffix "b" at end then it become binary-files modes
Ex:-
 rb, wb, ab, r+b, w+b, a+b, xb

Ex:-
fl = open("aaa.txt","w");
= Opens a file "aaa.txt" for write-mode
= If no file then creates a new-file
= If file exists then over-writes the data in file

==> Closing a file:-
= After completing file-operations, we close the file using close() function
Ex:-
 fl.close();

NOTE:-
fl is file obj.ref.var
==Diagram==
[fl]------>[aaa.txt]



==> File object properties:-
(in python everything is object, including files)
= Once file is opened in program, we have to work with file for read/write/append purpose
= To work with file operations, we have diff properties(variable)
1) name --> name of the opened-file Ex:- fl.name
2) mode --> Mode in which file is opened
3) closed --> checks for file open or closed (True/False)
4) readable() --> checks for file is readable or not (True/False)
5) writable() --> checks for file is writable or not (True/False)
"""
# Ex:-
#Program (FileEx1.py)
# (Program to open a file and check diff properties)
#Program to open a file and check diff properties

fname = input("Enter any filename : ");  #aa.txt
#fl = open(fname,"w");
fl = open(fname,"r");
print("File-name :",fl.name);
print("File-Mode :",fl.mode);
print("File-is Readable :",fl.readable());
print("File-is Writable :",fl.writable());
print("File-is Closed :",fl.closed);
fl.close();
print("File-is Closed :",fl.closed);


"""
==> Writing data to text-files:-
= It is done with 2-methods,
 1) write(str) #writes given str-data on file
 2) writelines(list of lines)
  #writes given list-data as multi-lines on file
"""
# Ex:-
#Program (FileEx2.py)
#Program to open a file and write data on it

fname = input("Enter any filename : ");  #bb.txt
#fl = open(fname,"w");  #write-mode(over-write***)
fl = open(fname,"a");  #append-mode(add-from-end)
print("File-is opened for Writing data");

#fl.write("Sai Kumar\n");
#fl.write("India Country\n");
#fl.write("Hyderabad City\n");
fl.write("Ram\n");
fl.write("India\n");
fl.write("Secbad\n");

fl.close();
print("File-Data Writing is done");


"""
```

```
NOTE:-
= If program is executed multiple-times using "w" then data is over-written with new-data
= for this we can use "a" mode (append-mode)
Ex:-
fl = open(fname,"a");
"""
# Ex:-
#Program (FileEx3.py)
#Program to open a file and write multi-line data on it writelines()

fname = input("Enter any filename : ");  #cc.txt
#fl = open(fname,"w");
fl = open(fname,"a");

print("File-is opened for Writing data");

listl = ["Sai\n","Ram\n","Ali\n","Tom\n","Pop\n"];
fl.writelines(listl);
fl.close();
print("File-Data Writing is done");



# NOTE:-
# While using write() or writelines(), "\n" separator is compulsory for multiple lines of data o.w we get single-line data


"""
==> Reading Char data from Text-Files:-
= To read char-data from text-files, we use below methods,
1) read() =>  Reads total-data at a time from a file
2) read(n)  => Reads n-chars from a file
3) readLine() => Reads 1-line at a time
4) readLines() => Reads All-lines at a time into a List DS
"""
# Ex1:-
#Program (FileEx4.py)
# (Program to read data from a file with diff read-methods)
#Program to read total data from a file
#Program (FileEx4.py)


#read()   #bb.txt
fname = input("Enter any filename : ");
fl = open(fname,"r");
data = fl.read();
print(data);
fl.close();
'''

'''
#read(n)
fname = input("Enter any filename : ");
fl = open(fname,"r");
data = fl.read(10);
print(data);
fl.close();
'''


'''
#readline() #use loop to read all lines one by one till last
fname = input("Enter any filename : ");
fl = open(fname,"r");  #bb.txt
line1 = fl.readline();
print(line1,end="");
line2 = fl.readline();
print(line2,end="");
line3 = fl.readline();
print(line3,end="");
fl.close();



#readLines()
fname = input("Enter any filename : ");
fl = open(fname,"r");
listlines = fl.readlines();
print(listlines);
for line in listlines:
 print(line,end="");
fl.close();



"""
**sp-case of open()**
==> with statement to open a file:-
(it is header-stmt to open a file)
(it provides suite to perform diff operations on a file)
Ex:-
 with open(fname,"w") as fl:
   ...
   ...
   ...
   ...

= we can also use with statement to open a file
= with stmt can be used to group file-operations as a block of code
= Advantage is it closes the file after all operations are done, even if exception occurs not need to close explicitly
"""
# Ex:-
#Program (FileEx5.py)
#Program to demo with-stmt to open a file

fname = input("Enter any filename : ");
with open(fname,"w") as fl:  #dd.txt
 fl.write("Sai\n");
 fl.write("India\n");
 fl.write("Hyderabad\n");
 print("Is File closed(inside with body) :",fl.closed);

#fl.close()  #not-required
print("Is File closed(outside with body) :",fl.closed);


"""
==> seek() and tell() functions:-
1) tell():-
= It gives current position of a cursor(file-pointer) inside a file from beginning
```

```
= index-position of 1st-char in a file is 0
(same like string-indexes)
Ex:-
 fl.tell()
#Program (FileEx6.py)
(Program to demo tell() & seek() functions)

2) seek():-
= It is used to move the cursor(file-pointer) to specified location
Syntax:-
 fl.seek(offset,fromwhere);
 offset -> no.of positions
 fromwhere -> 0(default is begin), 1(current-position), 2(from end)
Ex:-
 fl.seek(10)  #+ve forward-move
 ##fl.seek(-10) #-ve backward-move(not-supports)

NOTE:-
= Python-2 supports 0,1,2 values but Python-3 supports only 0
"""
# Ex:-
#Program (FileEx6.py)
# (Program to demo tell() & seek() functions)
#Program to demo tell() & seek() function (FileEx6.py)


#tell()
fname = input("Enter any filename : "); #bb.txt
fl = open(fname,"r");
print(fl.tell());
print(fl.readline());
print(fl.tell());
print(fl.readline());
print(fl.tell());
print(fl.readline());
print(fl.tell());
fl.close();



#seek()
data = "Hello Students, Welcome to Python";
fname = input("Enter any filename : ");
fl = open(fname,"w");  #ee.txt
fl.write(data);
fl.close();

with open(fname,"r+") as fl:
 text = fl.read();
 print(text);
 print("Cursor is at :",fl.tell());
 fl.seek(16);
 print("Cursor is at :",fl.tell());
 fl.write("Weldone");
 fl.seek(0);
 text = fl.read();
 print(text);
 fl.close();


"""
==> Checking for file exists or not?
= For this we can use OS library to get info. about files in our computer
= "os" module  has "path" sub-module, it contains isFile(), using which we can check file exists or not
Ex:-
 os.path.isFile(filename);
Ex:-
"""
#Program (FileEx7.py)
# (Program to check whether file exists or not and display its data)
#Program to check whether file exists or not and display its data
#(FileEx7.py)

import os,sys;
fname = input("Enter File-name : ");

if os.path.isfile(fname):
 print("File is there :",fname);
 fl=open(fname,"r");
 print("File Contents :");
 data=fl.read();
 print(data);
else:
 print("File doest NOT exists :",fname);
 sys.exit(0);

print("End of the Program");

# NOTE:-
# = sys.exit(0); #exits program execution process there only, done by PVM
# = 0 indicates Normal-Termination
# = 1 is Abnormal-Termination



# 31st day 31.30-08-22

"""
(Working with Binary-Files)
==> Handling Binary-Files data:-
(Images,Audio,Video,etc file)
= Here we store data in the form of binary-bits(0,1)
Ex:-
 images
 audio
 video
 etc...
= For this we use "b" with filemodes
Example,
 (rb, wb, ab, r+b, w+b, a+b, xb) -->binary-modes
=** here we use same reading & writing methods
"""
# Ex:-
#Program (FileEx8.py)
# (Program to read Image-File and write to New-Image-File)
#Program to read Image-File and write to New-Image-File (FileEx8.py)

fname1 = input("Enter Image File-name : ");
fname2 = input("Enter Copy-Image File-name : ");
fl = open(fname1,"rb");
```

```
f2 = open(fname2,"wb");
bytesdata = f1.read();
f2.write(bytesdata);
f1.close();
f2.close();
print("Image Copying is Done");

"""
"Assignment" (Ex9 and Ex10)
#WAP to perfrom cut & paste operation in python program
(1st copy org-file)  #read()
(2nd paste dup-file) #write()
(3rd del org-file)  #??
#WAP to perfrom multiple files copy & paste in single python program
(use list of file-names with for-loop)


==> Handling CSV file:-
= CSV means comma separated values
= table-data as text-file
= to work with .csv files, we have "csv-module"
==Diagram==
Ex:-
#Program (FileEx9.py)
(Prog to work with CSV-file)

step1:-
 f1 = open("students.csv","w");
 w1 = csv.writer(f1) gives csv-writer-object-ref
step2:-
=** write data using csv-writer-object-ref,
 w1.writerow(); takes list of values as input-para
step3:-
=** close csv-writer-object(w1) & also f1-file-obj

"""
#Prog to work with CSV-file writing mode
#(FileEx9.py)

import csv;
#with open("student.csv","w",newline='') as f1:
with open("student.csv","w") as f1:
 w1 = csv.writer(f1);
 w1.writerow(["SNo","SName","Height","Address"]);
 n = int(input("Enter No.of Students : "));
 for i in range(n):
  print("Enter Students Data:");
  sno=input("Roll-No : ");
  sname=input("Name : ");
  height=input("Height : ");
  addr=input("Address : ");
  w1.writerow([sno,sname,height,addr]);

print("All Student data successfully written to student.csv file");


"""
NOTE:-
= W.O newline='' attribute parameter, in csv-file we get blank-lines between data
= With newline='' attribute parameter from Python-3 is required
= Python-2, we use "wb" and newline='' attribute is not required
=** Internally DB-tables & Excel-files are .csv files only(text-data)



==> Reading data from .csv file:-
= Opens csv file and read the data from the file

==> Reading data from .csv file:-
= Opens csv file and read the data from the file
Ex:-
#Program (FileEx10.py)
(Prog to work with CSV-file)
(open .csv file(f1),attach to csv-reader-obj(r1),read-data)

Step1:-
 =** r1 = csv.reader(f1) to get csv-reader-obj on .csv file
Step2:-
 =** r1.list(csv-reader-obj) we get complete-data from .csv file as nested-list
"""
# Ex:-
#Program (FileEx10.py)
#Prog to work with reading-CSV-file (FileEx10.py)

import csv;

f1 = open("student.csv","r");
r1 = csv.reader(f1); #gives csv-file reader obj
csvdata = list(r1);  #nested-list
print(csvdata);

#access data with loops

for row in csvdata:
 for col in row:
  print(col,"\t",end='');
 print();


for row in csvdata:
 print(row)

print("End of the Program") ;


"""
==> Zipping & UnZipping files:-
= Zipping means compressing the files
= UnZipping means un-compressing the files
= Advantage,
 - Less Memory Space
 - Easy File-sharing
 - Improves Performance

(***)
= zipfile-module is used to work with this
= It contains "ZipFile" class

=> How to create .zip file:-
= Step1:-
```

```
= Create an object of ZipFile-class
(use zip-filename, mode, ZIP_DEFLATED constant)
Ex:-
fl = ZipFile("files.zip","w",ZIP_DEFLATED);
#ZIP_DEFLATED means creating a new-zipfile

= Step2:-
= Once file is created, add files to it using write() method
fl.write("aa.txt");
fl.write("bb.txt");
fl.write("cc.txt");
"""
# Ex:-
#Program (FileEx11.py)
# (Program to work with .zip files)
#Program to work with .zip files
#FileEx11.py

from zipfile import *;
fl = ZipFile("files.zip","w",ZIP_DEFLATED);
fl.write("aa.txt");
fl.write("bb.txt");
fl.write("cc.txt");
fl.close();
print("files.zip file is created");


"""
=> How to unzip a file?
= For this create ZipFile obj as follows,
Ex:-
 fl = ZipFile("files.zip","r",ZIP_STORED);
= ZIP_STORED constant represents unzip operation
(its a default-value)
=** Once object is created, we get all file-names using namelist() method
Ex:-
 filenames = fl.namelist();
 fl.printdir()
 fl.extractall()
"""


# Ex:-
#Program (FileEx12.py)
# (Program to work with .zip files)
#Program to work with .zip files

import time;

from zipfile import *;
fl = ZipFile("files.zip","r",ZIP_STORED);
filenames = fl.namelist();
print(filenames);

time.sleep(5);
fl.printdir();

time.sleep(5);
fl.extractall();

print("End of the Program");

"""
==> Working with Directories?
= Some common operations on directories are, (folders)
a) Current Working Directory
b) Create New Directory
c) Remove Existing Directory
d) Rename a Directory
e) List contents of Directory
etc
=** For this we use "os" module
(to perform above operations)
=> Current Working Directory?
= getcwd()
Ex:
import os;
cwd = os.getcwd();
print(cwd);

#Program (FileEx13.py)
(Program to work with Directories)


=> Creating Sub-Directory?
= mkdir("sub-dir");
Ex:-
import os;
os.mkdir("subfiles");
print("subfiles sub-dir is created");

=> Creating sub-dir in another directory?
Ex:-
import os;
os.mkdir("subfiles/subsubfiles");
print("subsubfiles dir is created inside subfiles directory");

=> Creating multiple directories with sub-directories at a time?
= makedirs()
Ex:-
import os;
os.makedirs("sub1/sub2/sub3");
print("multiple directories with sub-directories created");


=> Remove a Directory?
= rmdir() #1-dir at a time
Ex:-
import os;
os.rmdir("subfiles/subsubfiles");
print("subsubfiles directory is removed");

=> Removing multiple-directories in path?
= removedirs() #multiple-dirs at a time
Ex:-
import os;
os.removedirs("sub1/sub2/sub3");
print("All sub1/sub2/sub3 directories are removed");

=> Rename Directory:-
```

```
= rename()
Ex:-
import os;
os.rename("subfiles","newsubfiles");
print("Directory is Renamed");

=> Directory Contents:-
= listdir()
Ex:-
import os;
print(os.listdir("."));  #. means current-directory
= It is displayed as list of data
"""


#Program
#Program to work with Directories (FileEx13.py)


#getcwd()
#Current Working Directory
import os;
cwd = os.getcwd();
print(cwd);
'''

'''
#mkdir()
#Creating Directory & Sub-Directory
import os;
os.mkdir("subfiles");
print("subfiles sub-dir is created");
'''
'''
import os;
os.mkdir("subfiles/subsubfiles");
print("subsubfiles dir is created inside subfiles directory");
'''

'''
#makedirs()
#creating multiple sub-directories at a time
import os;
os.makedirs("sub1/sub2/sub3");
print("multiple directories with sub-directories created");
'''

'''
#rmdir()
#Removing Directory
import os;
os.rmdir("subfiles/subsubfiles");
print("subsubfiles directory is removed");
'''

'''
#removedirs()
#Removing multiple sub-directories at a time
import os;
os.removedirs("sub1/sub2/sub3");
print("All sub1/sub2/sub3 directories are removed");
'''

'''
#rename()
#Renaming a Directory
import os;
os.rename("subfiles","newsubfiles");
print("Directory is Renamed");
'''

'''
#listdir()
#Listing all the contents of directory
import os;
print(os.listdir("."));  #. means current-directory


"""
(******)
----------
==> Pickling and UnPickling of Objects:-
= It is Serialization and De-Serialization in Java
= It is used to write complete state of object to file and read complete state of object from a file
= Pickling means writing state of an object to a file
= UnPickling means reading state of an object from a file
Ex:- Student-obj or Employee-obj etc

=** To perform this we use pickle-module
= For Pickling we use dump() function
Ex:-
 pickle.dump(object, filerefvar);
= For UnPickling we use load() function
Ex:-
 obj = pickle.load(filerefvar);
==Diagram==
(s1,s2,studentfile.txt)

NOTE:-
= Pickling & UnPickling is done as binary-data
(0's and 1's) ---> hence use "wb" and "rb" modes
"""
# Ex:-
#Program (PickUnpick.py)
# (Program to perform Pickling and UnPickling)
#Program to perform Pickling and UnPickling
#PickUnpick.py

import pickle;
class Student:
 def __init__(self,sno,sname,height):
  self.sno=sno;
  self.sname=sname;
  self.height=height;
 def display(self):
  print(self.sno,"\t",self.sname,"\t",self.height);


#pickling
with open("studentfile.txt","wb") as f1:
```

```
 s1 = Student(1001,"Sai",6.0);
 pickle.dump(s1,f1);
 print("Student-Data is Pickled :");


#unpickling
with open("studentfile.txt","rb") as f1:
 s2 = pickle.load(f1);
 print("Student-Data(after UnPickling) :");
 s2.display();



# 32nd day 32.01-09-22

"""
==>>> MultiThreading in Python:-
= Multi means many (2 or more)
= Threading means small-logics(functions)

Def:-
= Executing 2 or more small-logics parallelly in a same program is called Multi-Threading


NOTE:-
= Multi-Threading  are widely used here,
1) Multimedia Graphics
2) Animations
3) Video Games
4) Web-servers & Application-servers
etc



==> Single-Threaded App v/s Multi-Threaded App:-
= In Single-Threaded App, multiple-tasks are executed one-by-one linearly
==Diagram==
 main-function(.py file) __main__()
 m1();
 m2();
 m3();
(m1,m2,m3 are executed linearly one-by-one)

= In Multi-Threaded App, multiple-tasks are executed parallelly
==Diagram==
 main-function(.py file) __main__()
 m1(); or m2(); or m3();
(m1,m2,m3 are executed parallelly)
(PVM executed multi-threads(small-logics/funcs) parallelly)

NOTE:-
= "threading" module in Python provides multi-threading in program
= thread means light-weight-process (doing some sp.task in same-prog)
= Every python-prog by default contains 1-thread known as "Main-Thread"
(__main__())

Ex:-
"""
#Program (ThreadEx1.py)
# (Printing name of current executing thread)
import threading;
print("Current-Executing-Thread : ",threading.current_thread().getName());
"""
NOTE:-
= current_thread() is in threading-module, it gives currently executing thread obj.ref
= On this obj.ref, we call getName(), which gives current executing thread-name

=** Hence Thread is also 1-object in python prog



==> Different ways to create a Thread in Python:-
(3-ways)
1) Creating a thread directly in main-prog
2) Creating a thread extending Thread-class (inheritance sub-class)
3) Creating a thread w.o extending Thread-class
(our own-class)

(****)
1) Creating a thread directly in our Program:-
Step::-
i)
 = Write your own methodname/function
 Ex:-
  def display():
ii)
= create object of Thread-class from threading-module
Ex:-
 t1 = Thread(target=methodname/funcname);
 t1 = Thread(target=display);

iii)
= use start() of Thread-class to run our method parallely
 t1.start();

Ex:-"""
#Program (ThreadEx2.py)
#Program (ThreadEx2.py)
#Creating a Thread directly in program

from threading import *;

#step-1
def display():
 for i in range(1,11):
  print("Child-Thread");
#step-2
t1 = Thread(target=display);
#step-3
t1.start();


#main-thread
for i in range(1,11):
 print("\t\tMain-Thread");


"""
NOTE:-
= Here we get output for Main-Thread & Child-Thread 10 times each with parallelly execution (Random-Execution)
```

```
= The pattern differs from Run to run and execution to execution
** Here PVM calls main-thread(program) & we call display() thread



NOTE:-
==> Using Thread-class(Inheritance):-
= It is a pre-defined class present in threading-module
= Using this class we can create our own threads
= To Thread-class constructor, we pass the "target=funcname" attribute
= Such function becomes small-logic of the thread for execution
= To start or executed the method(logic) of thread-obj, we use start() method


2) Creating a Thread by extending Thread-class:- (Inheritance)
=i) Create our Child-class inheriting from Thread-class
=ii) Override run() from Parent-class for multi-threading logic
(run() is available in Thread-class, here we perform overriding)
=iii) Create and object of our thread-class
=iv) When we call start(), automatically run() is called for parallel execution

4-steps::
-------
i)
 define our own-class inherited from Thread-class
ii)
 redefine run() method --> overriding
 (it is responsible for parallel-exec)
iii)
 create our own-class object
iv)
 call start() method on our class-object

Ex:-
"""
#Program (ThreadEx3.py)
#Program (ThreadEx3.py)
#Program to multi-threading inheriting from thread-class

from threading import *;

#step-1
class MyThread(Thread):
 def run(self):      #step-2
  for i in range(1,11):
   print("Child-Thread");

#step-3
t1 = MyThread();
#here target=run method automatically(no-need to pass as para)

#step-4
t1.start();


#main-thread-logic
for i in range(1,11):
 print("\t\tMain-Thread");



"""
3) Creating a Thread without extending Thread-class:-
4-Steps::-
i)
= Here also we create our own class with our own method for multi-threading logic (no run())
ii)
= Create an object of our thread class
iii)
= Create an object of Thread class and pass "target=obj.methodname" attribute as parameter to constructor
iv)
= Call start() for parallel execution of given method in our class (using Thread-class obj)

Ex:-
"""
#Program (ThreadEx4.py)
#Program (ThreadEx4.py)
#Program to demo Multi-thread w.o inheriting Thread-class

from threading import *;

#step-1
class Demo:
 def display(self):
  for i in range(1,11):
   print("Child-Thread with display()");

#step-2
obj=Demo();
#Step-3
t1 = Thread(target=obj.display);
#step-4
t1.start();


#main-thread-logic
for i in range(1,11):
 print("\t\tMain-Thread");


"""
==> Working with Thread-Names:-
= We can give our own names to the threads
= It is done by below methods,
1)t1.getName() => gives name of a thread
2)t1.setName("newname") => sets name of a thread

=** Every thread-obj has implicit-var "name", which represents name of a thread
Ex:- t1.name

NOTE:-
= Default thread-names given to threads are Thread-1, Thread-2, ... so-on

Ex:-
"""
#Program (ThreadEx5.py)
#Program (ThreadEx5.py)
#(Program to work with thread-names)

from threading import *;
```

```
#case-1(main-thread)
print(current_thread().name);
#current_thread().name="Hello-Thread";
print(current_thread().name);




#case-2
def display():
 for i in range(1,11):
  #print(current_thread().getName());
  print(current_thread().name);
def show():
 for i in range(1,11):
  #print("\t\t",current_thread().getName());
  print("\t\t",current_thread().name);

t1 = Thread(target=display);
t2 = Thread(target=show);
#t1.setName("my-display-thread");
#t2.setName("your-show-thread");
t1.name="my-display-thread";
t2.name="your-show-thread";
t1.start();
t2.start();


"""
==> Thread Identification Number(ident):-
= "ident" is implicit-var to access unique thread identification number
= this identification-no is a unique no. given by PVM during execution
= It identifies each-thread uniquely in program
(int-number)
Ex:-
 t1.ident (from Thread-class)
 t2.ident
"""
# Ex:-
#Program (ThreadEx6.py)
#Program (ThreadEx6.py)
#Prog to demo ident implicit-var

from threading import *;

def test():
 print("Child-Thread");

t1 = Thread(target=test);
t2 = Thread(target=test);
t1.start();
t2.start();

print("Main-Thread ident :",current_thread().ident);
print("Child-Thread ident(t1) :",t1.ident);
print("Child-Thread ident(t2) :",t2.ident);



#(Working with thread-methods):-
# ==> active_count():-
# = It givens no.of active-thread currently running or active in python program

# Ex:-
#Program (ThreadEx7.py)
#Program (ThreadEx7.py)
#active_count()

from threading import *;
import time;

def display(): #it is our thread-run()
 print(current_thread().name,"...started");
 time.sleep(2);
 print(current_thread().name,"...ended");

print("No.of active-threads :",active_count());

t1=Thread(target=display,name="ChildThread1");
t2=Thread(target=display,name="ChildThread2");
t3=Thread(target=display,name="ChildThread3");
#t1.name="ChildThread1"

t1.start();
t2.start();
t3.start();
print("No.of active-threads :",active_count());

#main-thread
time.sleep(10);
print("No.of active-threads :",active_count());

print("End of the Main-Thread");

"""
NOTE:-
t1=Thread(target=display,name="ChildThread1");
= we can give names to the threads while creating object of thread-class using name="" parameter



==> enumerate():-
= It returns list[...] of all active-threads(obj.ref) currently running in program
Ex:-
"""
#Program (ThreadEx8.py)
#Program (ThreadEx8.py)
#enumerate()

from threading import *;
import time;

def display(): #it is our thread-run()
 print(current_thread().name,"...started");
 time.sleep(3);
 print(current_thread().name,"...ended");

t1=Thread(target=display,name="ChildThread1");
```

```
t2=Thread(target=display,name="ChildThread2");
t3=Thread(target=display,name="ChildThread3");
t1.start();
t2.start();
t3.start();

list1 = enumerate();
print(list1)
for tt in list1:
 print("Thread-Name :",tt.name);

print();
time.sleep(15);
list1 = enumerate();
for tt in list1:
 print("Thread-Name :",tt.name);



# ==> is_alive() Method:-
# = Checks whether particular thread is still executing or done its job (True/False)
# Ex:-
#Program (ThreadEx9.py)
#Program (ThreadEx9.py)
#is_alive()

from threading import *;
import time;

def display(): #it is our run()
 print(current_thread().name,"...started");
 time.sleep(2);
 print(current_thread().name,"...ended");

t1=Thread(target=display,name="ChildThread1");
t2=Thread(target=display,name="ChildThread2");
t3=Thread(target=display,name="ChildThread3");
t1.start();
t2.start();
t3.start();

#main-thread
print(t1.name,"is Alive :",t1.is_alive());
print(t2.name,"is Alive :",t2.is_alive());
print(t3.name,"is Alive :",t3.is_alive());

print();
time.sleep(15);
print(t1.name,"is Alive :",t1.is_alive());
print(t2.name,"is Alive :",t2.is_alive());
print(t3.name,"is Alive :",t3.is_alive());

print("End of the Main-Thread");


# NOTE:-
# = isAlive() is deprecated(outdated/old)
# = Use is_alive() instead of that



# 33rd day 33.02-09-22

"""
***Other-Methods in Thread-class***
-----------------------------------------
==> join():-
= if a thread wants to wait for other-threads to complete their job then we have to use join()
(once all the remaining threads completes their jobs then all thread go for dead-state/go for other-jobs)
Ex:-
#Program (ThreadEx10.py)

NOTE:-
= We can also use join() with time also
Ex:-
 t1.join(5);
= In this case, main-thread waits for 5-seconds
 t1.join()
"""
#Program (ThreadEx10.py)
#join() method

from threading import *;
import time;

def display(): #it is our run()
 for i in range(10):
  print("Hello-Thread");
  time.sleep(1);

def show():  #it is also our run()
 for i in range(20):
  print("\t\tWelcome-Thread");
  time.sleep(1);

t1 = Thread(target=display);
t2 = Thread(target=show);
t1.start();
t1.join(5);
#t1.join();
t2.start();
#t2.join();


#main-thread
for i in range(10):
 print("\t\t\t\tMain-Thread");
 time.sleep(1);


"""
==> Daemon Thread:-
(least priority threads)
= Threads which are running in the background are called Daemon-Thread
= They provide support for other non-daemon-threads (main-thread)
Ex:- Garbage-Collector
(When main-thread runs out-of-memory, PVM runs GC thread for to destroy useless objects and provide free-memory, hence main-thread can continue its execution without any memory-problem

= t1.isDaemon() checks whether a thread is daemon-thread or not
```

```
(they are low priority-threads)
= t1.daemon property(variable) checks for same-above

= t1.setDaemon(boolean-value), changes its property
(used before starting a thread o.w RuntimeException)
Ex:-
#Program (ThreadEx11.py)

==> Daemon Thread:-
(least priority threads)
= less chance for execution in group of threads
=> methods,
= t1.isDaemon()  #True/False
= t1.daemon variable #True/False
= t1.setDaemon(boolean)
(use before start())
"""
# Ex:-
#Program (ThreadEx11.py)
#Program (ThreadEx11.py)

from threading import *;


#case-1(main-thread)
#current_thread().setDaemon(True);
print(current_thread().isDaemon());
print(current_thread().daemon);


#case-2
def display(): #it is our run()
 while True:
  print("Child-Thread");

t1 = Thread(target=display);
print(t1.daemon);
t1.daemon=True;
t1.start();

#main-thread
while True:
 print("\t\tMain-thread");


"""
NOTE:-
= Main-Thread is always non-daemon-thread, its Nature can't be changed because it is already started at beginning only by PVM


==> Synchronization in Multi-Threading:-
(automatic ITC -> Inter-thread-communication)
= If multiple-threads are executing parallelly and accessing same common-sharable-data then there are chances of Data-Inconsistency (Wrong-data updates)
Ex:-
==Diagram==
(Online-Ticket Booking)
(multiple-users)---->website/app---->select-cinema---->theater----->date----->show-time---->seats--->booking/billing/checkout

NOTE:-
= To avoid this, we perform "synchronization"(auto-ITC)
(Proper communication b/w multiple-threads accessing common-sharable-data to avoid Data-Inconsistency/Wrong-data/Wrong-Tranx)
(Critical-Resource)
Ex:-
"""
#Program (ThreadEx12.py)
# (Without Synchronization)
#Program (ThreadEx12.py)
#(Without Synchronization)

from threading import *;
import time;

#common-transaction/func
def wishes(name):  #name is common-sharable-data
 for i in range(10):
  print("Good Morning :",end='');
  time.sleep(1);
  print(name);

t1 = Thread(target=wishes,args=("Sai",));
t2 = Thread(target=wishes,args=("Ram",));
t1.start();
t2.start();


"""
=** Here, we get Irregular-Output, coz both t1 and t2 are execute same common sharable method(wishes()) and output is In-consistent(Wrong)
= To overcome this we perform "Synchronization"
= Here PVM allow only 1-thread to perform operation common-sharable-data or functions and avoids Data-Inconsistency(wrong)
(one-by-one execution)


NOTE:-
= In python, synchronization (auto.ITC) is done in 3-ways,
 1) Lock mechanism
 2) RLock mechanism
 3) Semaphore mechanism

1) Synchronization using Lock concept:-
= Lock is fundamental synchronization mechanism in threading module
= It is created as follow,
Ex:-
 l1 = Lock(); #Lock-class-obj
= Lock-obj can be hold by only 1-thread at a time
(if other thread requires same lock then it will wait till before thread release lock)
Ex:-
Common Telephone Booth, Common Washrooms
= Thread gets lock as follows,
Ex:-
 l1.acquire();
= Thread releases lock as follows,
Ex:-
 l1.release();
(for releasing lock, that thread should be owner of lock o.w we get RuntimeError)

=** Above Both methods are called inside multi-threading logics
(run() or user-defined run() method)

Ex:-
#Program (ThreadEx13.py)
```

```
(Lock Mechanism)

NOTE:-
=** Here which ever thread lst goes to wishes() executes it without and Data-Inconsistency followed by other thread one-by-one
(Hence Synchronization is achieved by Lock()/acquire()/release())
=** Always create and keep Lock-obj ready at begin of Main-Thread
=** use acquire() at beginning of MT-logic
=** use release() at end of MT-logic



NOTE:- (Simple-Lock problem in Lock() mechanism)
= If any thread tries to acquire same lock then it is blocked even though same-thread tries to lock again
(same thread mulitple locks)
Ex:-
#Program (ThreadEx13.py)
(Simple-Lock Problem)

NOTE:-
= To kill Blocked-Thread, use Ctrl+PauseBreak (Ctrl+C)
= Threads calling Recursive-functions or Nested-Access or using Loops then it may acquire same-lock again and gets blocked for itself
(Hence it is not suitable for all-situations)
= To overcome such problem, we go for "RLock"



2)
==> Reentrant Lock:- (RLock())
= It means a thread can hold same-lock again and again
= But if lock is held by other threads then only it is blocked
= this chance is given only for owner-thread
Ex:-
#Program (ThreadEx13.py)
(RLock Problem)

NOTE:-
= For every acquire() of RLock make-sure to release() it
EX:-
rl = Rlock();
rl.acquire();
rl.acquire();
rl.release();
rl.release();
= After 2-release only RLock is released(o.w not)
NOTE:-
= Only same-owner can have RLock multiple-times
= No of acquire() and release() should be matched



==>(RLock Synchronization):-
Ex:-
#Program (ThreadEx13.py)
(RLock Synchronization)
NOTE:-
= Instead of RLock(), if we use Lock() then thread will be blocked for itself

==> Difference b/w Lock() and RLock():-
i)
= Lock() can be acquired by only 1 thread at a time including owner thread
= RLock() can be acquired by only 1 thread at a time but owner thread can acquire multiple-times
ii)
= Not suitable for Recursive or Looping logics
= suitable for Recursive or Looping logics
iii)
= Lock obj takes care of only Locked or Unlocked information
= RLock obj takes cares of Locked, Unlocked and also owner information, no.of times lock acquire and release
"""

#Program (ThreadEx13.py)
#Synchronization-Lock Mechanism using Lock()

#using Lock()
from threading import *;
import time;



#Lock-class-obj mechanism
#step-1
ll=Lock();
def wishes(name): #wishes() is run() method
 ll.acquire();
 for i in range(10):
  print("Good Morning :",end='');
  time.sleep(1);
  print(name);
 ll.release();

t1 = Thread(target=wishes,args=("Sai",));
t2 = Thread(target=wishes,args=("Ram",));
t3 = Thread(target=wishes,args=("Ali",));
t1.start();
t2.start();
t3.start();
'''

'''
#Simple-Lock mechanism small-Problem
from threading import *;
ll=Lock();
print("Main-Thread acquiring Lock");
ll.acquire();
print("Main-Thread again acquiring same-lock");
ll.acquire();
'''


'''
#RLock() mechanism with multiple-locks
from threading import *;
import time;
#step-1
rl=RLock();
print("Main-Thread acquiring RLock");
rl.acquire();
print("Main-Thread again acquiring same-Rlock");
rl.acquire();
time.sleep(3)
print("Main-Thread is not Blocked");
```

```
print("End of the Program");



#Semaphore-Mechanism
from threading import *;
import time;
sp=Semaphore(2);
def wishes(name):
 sp.acquire();
 for i in range(10):
  print("Good Morning :",end='');
  time.sleep(1);
  print(name);
 sp.release();

t1 = Thread(target=wishes,args=("Sai",));
t2 = Thread(target=wishes,args=("Ram",));
t3 = Thread(target=wishes,args=("Ali",));
t4 = Thread(target=wishes,args=("Tom",));
t1.start();
t2.start();
t3.start();
t4.start();


"""
3) Synchronization using Semaphore:-
= Here it allows to access common-sharable-data or resource by multiple-threads at a time
= It allows limited no.of sharings of data or resource b/w multiple-threads
= It is advanced Synchronization Mechanism
= Its object is created as follows,
Ex:-
sp = Semaphore(counter);
= counter is max.no.of threads for parallel access of data
= default is 1

= When thread acquires a lock then counter is decremented by 1
= When thread releases a lock then counter is incremented by 1

Case1:-
sp1 = Semaphore();
= Here it allows only 1-thread at a time to lock the data/resource
(same as Lock())

Case2:-
sp2 = Semaphore(5);
= Here it allows 5-threads at a time to lock the data/resource
(remaining threads wait)

Ex:-
#Program (ThreadEx14.py)
(Semaphore-Mechanism)
NOTE:-
= Here only 2-threads are allowed to access wishes() and remaining will be waiting



(2-types of Semaphore)
==> Regular & Bounded Semaphore:-
= Regular Semaphore is unlimited wrt to release() method, to increment counter
= Sometimes counter may exceed no.of acquire() also
Ex:-
#Program (ThreadEx13.py)
(Regular Semaphore)
sp = Semaphore(2);

= However, in Bounded-semaphore, no.of release() should not exceed no.of acquire() calls o.w we get "ValueError"
Ex:-
#Program (ThreadEx13.py)
(Bounded-Semaphore)
sp = BoundedSemaphore(2);

NOTE:-
= Recommended to use BoundedSemaphore() than Semaphore()

==> Difference b/w Lock() and Semaphore():-
1)
= Lock-obj can be obtained by only 1-thread
= Semaphore-obj can be obtained by multiple fixed no of threads using a counter parallelly

NOTE:-
= Advantage is Synchronization is to avoid Data-Inconsistency problems
= Dis-Advantage is increases waiting-time and performance issues
"""



# 34th day 34.1.03-09-22

"""
==>>> Database-Programming in Python:-
(Python Database Connectivity)
= Data which is processed in program in temporary
i.e, once execution is done it is erased from memory and for next-execution again we have to give new-input
= Files stores program data permanently and is less secure and less efficient
= For this we have Database storage system for Secure-Access and Efficient-Access of data

=> Basically we have 2-types of Storage Areas in any Programming:-
1) Temporary Storage
2) Permanent Storage

1) Temporary Storage:-
= Here program-data is stored temporarily till program execution process only
Ex:-
Python Variables, List, Tuple, Set, Frozen-Set, Dictionary etc

2) Permanent Storage:-
= Here program-data is stored permanently
= Also called as Persistent Storage
Ex:-
Computer-Files, DataBases, Data Warehouses, Big Data etc



==> File Systems:-
= They are Local Computer OS file-system
= Suitable for storing less amount of data
```

```
Limitations:-
 = Cannont store Huge Amount of data
 = All Operations are Manual (Insert/Update/Delete)
 = No Security to data (Anyone can open and access file-data)
 = Cannot prevent duplicate data
 = Have Inconsistency of data (Wrong-data)
= To overcome above limitations, we go for DataBases

==> DataBase System:-
= Here we can store Huge amount of data (Tables)
= Provides Query Language for DB Operations
= Data-Security is provided with Username & Password
= Provides constraints(conditions) on Table-Data (Duplicate data can be avoided)
Limitations:-
 = Cannot hold Tera bytes of data
 = Supports only Structured Table data
 = No support for Semi-Structured data (like XML)
 = No support for Unstructured data (Images/Audio/Video files)
= To overcome above limitations, we can go for Data ware Houses, Big-Data etc




==> Python DB Programming:-
= It allows us to communicate with DB & its tables to perform DB Operations (Create tables, insert, update, delete, selecting-data)
= Using Python we send SQL-commands to DB for operations
= We can communicate with diff DB's like Oracle, MySql, Sql-Server, Ingress, Postgre, GadFly, Sqlite3(django), MongoDB, DB2 etc
= Python provides separate module for each DB

***Python DB Drivers***
-------------------------
Ex:-
 mysql.connector module (MY-SQL DB)***
 cx_Oracle module (Oracle DB)###

 pymssql module (MS-SQL Server)***

NOTE:-
= These are 3rd-party lib-modules for DB-prog
= Such lib-modules, should be installed & used in prog
= It is done as follows
Ex:-
cmd> pip install mysql-connector-python
(mysql_connector_python-8.0.30)
cmd> pip install cx_Oracle
(internet connection is compulsory)

= Verify this
(in Py.Interactive-Mode)
>>>help("modules")
(it provides complete modules available in our py-software)


==> MySQL Software installation::-
(MySQL 5.0 or later) latest is MySQL 8.0
(www.mysql.com/downloads/installer/)
NOTE:-
 1)install MySQL workbench 8.0.30
 2)install MySQL server 8.0.30
 (portno:3306)
 (uname:root; pwd:root; repwd:root)
 (service-name: MYSQL80)
 (Execute-button)
"""
=> How to verify/open mysql8.0.30 cmd-line-utility::-
1)click-windows-start-btn -----> goto "MYSQL" folder ----> select "MySql command line client"

2) give pwd:root (for root-user)
3)
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql> use mysql;
Database changed

mysql> show tables;
+-----------------------------------------------------+
| Tables_in_mysql                                     |
+-----------------------------------------------------+
| columns_priv                                        |
| component                                           |
| db                                                  |
| default_roles                                       |
| engine_cost                                         |
| func                                                |
| general_log                                         |
| global_grants                                       |
| gtid_executed                                       |
| help_category                                       |
| help_keyword                                        |
| help_relation                                       |
| help_topic                                          |
| innodb_index_stats                                  |
| innodb_table_stats                                  |
| password_history                                    |
| plugin                                              |
| procs_priv                                          |
| proxies_priv                                        |
| replication_asynchronous_connection_failover        |
| replication_asynchronous_connection_failover_managed |
| replication_group_configuration_version             |
| replication_group_member_actions                    |
| role_edges                                          |
| server_cost                                         |
| servers                                             |
| slave_master_info                                   |
| slave_relay_log_info                                |
| slave_worker_info                                   |
| slow_log                                            |
| tables_priv                                         |
| time_zone                                           |
| time_zone_leap_second                               |
```

```
| time_zone_name                                          |
| time_zone_transition                                    |
| time_zone_transition_type                               |
| user                                                    |
+---------------------------------------------------------+
37 rows in set (0.00 sec)


mysql> create database sampledb;
Query OK, 1 row affected (0.00 sec)
(Query means command)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sampledb           |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> use sampledb;
Database changed

mysql> show tables;
Empty set (0.00 sec)

mysql> exit
(closes mysql-cmd-line client-app)
(but in background mysql-server-8.0.30 is still running, and we can connect from python-prog)



NOW,(*****)
==> Steps for Pythons DB Programming:-
Step1:-
= Import DB specific DBase module
Ex:-
 import mysql.connector;
 import cx_Oracle;



Step2:-
= Make or Establish connection b/w Python-Program & DB
= It is done with connect() function
Ex:-
conn = mysql.connector.connect(host="localhost",user="root", password="root",port="3306",database="mydb");

(or)
conn = cx_Oracle.connect("scott/tiger@localhost/xe");/orcl



Step3:-
= For executing SQL-cmds and hold result, we use special-object of "Cursor" class
= For this cursor() function is used
Ex:-
 mycursor = conn.cursor();


Step4:-
= Now, execute SQL-queries using execute() of Cursor-class
Syntax:-
 execute(sqlquery); #Single-Query
 executescript(sqlqueries); #String of SQL-Queries with ;
 executemany(); #Executes Parameterized Query (with-vars)
Ex:-
 mycursor.execute("select * from employees");

=*** Once the command is executed from python-prog, cursor-object gets & stores(holds) the result


Step5:-
= Commit or rollback the transaction based on requirement (DML operations) i.e, insert/update/delete
Ex:-
 conn.commit() #Saves changes to DB
 conn.rollback() #Undo the changes to DB

Step6:-
= Fetch/Getting the result-data using cursor object (for select-queries)
Ex:-
 fetchone() #for single-row
 fetchall() #for multiple-rows(list of rows)
 fetchmany(n) #for first n-rows only
(Each-row is tuple)

Ex1:-
 data = mycursor.fetchone();
 print(data);
Ex2:-
 data = mycursor.fetchall();
 for row in data:
  print(row);
Ex3:-
 data = mycursor.fetchmany(5);
 for row in data:
  print(row);

Step7:-
= After DB-operations, it is recommended to close the resources used in program in reverse order of opening
Ex:-
 mycursor.close();
 conn.close();

Step8::-
= Finally, verify output of the DB program


NOTE:-
= Some important methods used in Python-DB Programming
Ex:-
 connect()
 cursor()
 ------------
 execute()
```

```
 executescript()
 executemany()
 -------------
 commit()
 rollback()
 ------------
 fetchone()
 fetchall()
 fetchmany(n)
 fetch()
 ------------
 close()
=** These methods are common for different Databases



(just refer for understanding)
**********************************************************
==> Working with Oracle/MYSQL Database:-
= For making communication b/w Python-Programming and Databases, we required some translator
(translates Python-Program calls to DB Specific calls & vice-versa)
= Technically it is called "Driver/Connector"
==Diagram==
Python(prog-lang) ------<driver>------ Database(storage-s/w)


Ex:-
= For Oracle, we require "cx_Oracle" driver
= "cx_Oracle" is Python-Extension-Module
= It allows us to access Oracle DB
= Used in Python2 and Python3
= It works with all versions or Oracle (8,9,10,11,12,18,19 etc)

=> Installing cx_Oracle:-
= Open Windows Command-prompt (Not Python command-prompt)
= Use below commands,
Ex:-
cmd> pip install cx_Oracle
cmd> pip install cx_Oracle --upgrade
....
....
(Latest Version is : cx-Oracle-8.2.1)

=> Verify Installation:-
= From Python console(prompt), use this command
>>> help("modules");
(Provides list of all pre-defined modules installed in the system)

=> MYSQL Driver/Module:-
cmd> pip install mysql-connector-python
(8.0.27 is Last version)
cmd> pip install mysql-connector-python --upgrade
(8.0.27 is Last version)

(****)
MYSQL 5.5 or 8.0.30
==> Downloading & Installing MySQL softare::-
https://dev.mysql.com/downloads/installer/


C:\Program Files (x86)\MySQL\MySQL Server 5.5\bin>mysql -u root -p
Enter password: ****

NOTE:-
= Open oracle and mysql command-prompts


==> Working with MySQL DB directly::-
(DB-SQL-commands)
= to work with any DB directly, we use SQL commands

==> Open MYSQL Command Line::-
click(Start-button) ----> select MySQL from Programs ---> select MYSQL 5.5/8.0.30 ----> select "MYSQL command-line"
(enter pwd:root)

==> Some basic commands::-
mysql> show databases;
(4-default-DBs)

mysql> use test;
Database changed

mysql> show tables;
Empty set

mysql> exit
(closes mysql-cmd-line)
**********************************************************


"""
"""
==>> Start with Programming::-
=*** For mysql use port=3306/3308 for connection
//Programs on PDBC with MySQL/Oracle(32-bit)
Ex1:-
#Program (DBEx1.py)
(Program to connect with MYSQL and print its version)

NOTE:-
= MySQLConnection is a class, using which we store connection obj. to MYSQL DB

#Program (DBEx1.py)
#Program to connect with MySQL and print its version/connection)


#MySQL
import mysql.connector; #module for mysql
conn = mysql.connector.connect(host="localhost",user="root", password="root",port=3306);

print(conn);
print(type(conn))

conn.close();




# 35th day 35.05.09.22
##Programs in PDBC to MySQL::-
```

```
# ------------------------------------

#Program(MySQLDBCreate.py)
# (Program to create mysql new database "mydb" & verify it)

#Program(MySQLDBCreate.py)
#(Program to create mysql new database "mydb" & verify it)

#create database
import mysql.connector;

conn = mysql.connector.connect( host="localhost", user="root",password="root", port=3306);
mycursor = conn.cursor();

#drop mydb-database
mycursor.execute("drop DATABASE mydb");
print("Database dropped successfully");

#mycursor.execute("CREATE DATABASE mydb;");
#print("Database Created successfully");

#verify databases
mycursor.execute("SHOW DATABASES;");
print(mycursor);    #iterable-obj(used with loops) list-of-tuples
for x in mycursor:
   print(x)


# NOTE:-
# = MySQL command
mysql> show databases;
mysql> create database mydb;
mysql> show databases;
# -------------------------------
mysql> use mydb;
mysql> show tables;
# -------------------------------
mysql> drop database mydb;
mysql> show databases;

# Ex:-
#Program (DBEx2.py)
# (Program to create Employees-table in Oracle-DB/MYSQL)


***Create mydb-database & keep it ready****
(and after that create tables under mydb-database)
(inside database, we create tables)
NOTE:- (MYSQL commands for table-creation)
mysql> show databases;
mysql> use mydb;
mysql> show tables;
mysql> create table employees
    (
    eno int,
    ename varchar(10),
    esal int,
    eaddr varchar(20)
    );
mysql> show tables;
mysql> describe employees;
mysql> drop table employees;
mysql> show tables;


#Program (DBEx2.py)
#Program to create Employees-table Oracle-DB/MYSQL

#mysql
import mysql.connector;
try:
 conn = mysql.connector.connect(host="localhost",user="root",password="root",database="mydb",port=3306);
 print(conn);
 print();
 mycursor=conn.cursor();
 #mycursor.execute("create table employees(eno int,ename varchar(10), esal int, eaddr varchar(10))");
 #print(mycursor);
 #print()
 #for x in mycursor:
 # print(x)
 #print();
 #print("Table Created Successfully");
 #print()
 mycursor.execute("show tables;");
 for x in mycursor:
  print(x)
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();



# ------------------------------------------------------------
# Ex2a:-
#Program (DBEx2a.py)
# (Program to drop Employees-table in Oracle-DB/MYSQL)
# -------------------------------
# **connect to mysql-database**

mysql> show databases
mysql> use mydb
mysql> show tables
mysql> drop table Employees
mysql> show tables
mysql>create table employees
    (
    eno int,
    ename varchar(10),
    esal int,
    eaddr varchar(10)
    );
mysql> show tables
# -------------------------------
```

```
#Program (DBEx2a.py)
#Program to drop Employees-table from Oracle-DB/MySQL

#mysql
import mysql.connector;
try:
 conn = mysql.connector.connect(host="localhost",user="root",password="root",database="mydb",port=3306);
 print(conn);
 print()
 mycursor=conn.cursor();
 mycursor.execute("drop table employees");
 print(mycursor);
 print()
 print("Table Dropped Successfully");
 print()
 mycursor.execute("show tables;");
 for x in mycursor:
  print(x)
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();




# ---------------------------------------------------------------------
# Ex3:- (create table once-again and insert the records)
#Program (DBEx3.py)
# (Program to insert record into Employees-table of Oracle-DB/MYSQL)
# ---------------------------------------

(MY-SQL commands)
mysql> show tables;
mysql> create table employees
         (
         eno int,
         ename varchar(10),
         esal int,
         eaddr varchar(10)
         );
mysql> show tables;
mysql> describe employees;
mysql> select * from employees;
mysql> insert into employees values(1001,'Sai',5600,'Hyd');
mysql> select * from employees;
mysql> delete from employees where eno=1001;
mysql> select * from employees;
(dont use below-cmds)
#mysql> drop table employees;
#mysql> show tables;


#Program (DBEx3.py)
#(Program to create & insert record into Employees-table of Oracle-DB/MYSQL)

#mysql
import mysql.connector;
try:
 conn = mysql.connector.connect(host="localhost",user="root",password="root",database="mydb",port=3306);
 mycursor=conn.cursor();
 mycursor.execute("insert into employees values(1001,'Sai',5600,'Hyd')");
 print()
 for x in mycursor:
  print(x)
 conn.commit();
 print("Record Inserted Successfully");
 mycursor.execute("select * from employees;")
 print()
 for x in mycursor:
  print(x)
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();

"""
-------------------------------------------------
Ex4:-
Program (DBEx4.py)
(Program to insert multiple-records into Employees-table of Oracle-DB/MYSQL using executemany())
%s (string)  #preferable is %s(auto. converted to respective dtype on DB)
%d (integer)
%f (float)
==** %char means unknown value in sql-command in execute() function


NOTE:-
1)
= For inserting 1-record, mysql-command is,
mysql> insert into employees values(1001,'Sai',5600,'Hyd'); //static-values
****cursorObj.execute("insert into employees values(1001,'Sai',5600,'Hyd')");

2)
= For inserting multiple-records at a time from Python-Program,
sqlcmd = "insert into employees values(%s,%s,%s,%s)"
(here, we use executemany() with 2-input-para)
Ex:-
 cursorObj.executemany(sqlcmd,listoftuples)
records = [(1002,'Ram',3600,'Secbad'),
    (1003,'Ali',4600,'HiTech'),
    (1004,'Tom',2600,'KPHB')];

"""
#Program (DBEx4.py)
#(Program to insert multiple-records into Employees-table of Oracle-DB /MYSQL using executemany())

#mysql
import mysql.connector;
```

```
try:
 conn = mysql.connector.connect(host="localhost",user="root",password="root",database="mydb",port=3306);
 mycursor=conn.cursor();
 sqlcmd = "insert into employees(eno,ename,esal,eaddr) values(%s,%s,%s,%s)";
 #%s means unknown values
 #list of tuples
 listofrecords = [
  (1002,'Ram',3600,'Secbad'),
  (1003,'Ali',4600,'HiTech'),
  (1004,'Tom',2600,'KPHB')
            ];
 mycursor.executemany(sqlcmd,listofrecords);
 conn.commit();
 print(mycursor.rowcount)
 print("Record(s) Inserted Successfully");
 print()
 mycursor.execute("select * from employees");
 for x in mycursor:
  print(x)
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();


# NOTE:-
# = mycursor.rowcount var is available for insert/update/del commands...


# ------------------------------------------------------
# ***(Assignment)*** (DBEx41.py)
# = For inserting 1-record with dynamic-input,
print("Enter Employee-Data ::");
empno = input("Enter emp-no ::");
ename = input("Enter emp-name ::");
esal = input("Enter Salary ::");
eaddr = input("Enter Address ::");
# ****sqlcmd="insert into employees(eno,ename,esal,eaddr) values("+empno+",'"+ename+"',"+esal+",'"+eaddr+"')";
    # cursorObj.execute(sqlcmd);

# ------------------------------------------------------



-- 36th day 35.06-09-22
****PDBC other programs***
-----------------------------------------------
Ex5:-
#Program (DBEx5.py)
(Program to insert multiple-records into Employees-table of Oracle-DB/MYSQL using execute() with dynamic-input) - using loop

NOTE:-
(***)
sqlcmd = "insert into employees values(%s,%s,%s,%s)";
cursorObj.execute(sqlcmd,(eno,ename,esal,eaddr)); #tuple-of-values

=*** Always use conn.commit() in try-block & conn.rollback() in except-block


#Program (DBEx5.py)
#(Program to insert multiple-records into Employees-table of Oracle-DB/MYSQL using execute() with dynamic-input using loop)


#mysql
import mysql.connector;
try:
 conn = mysql.connector.connect(host="localhost",user="root",password="root",database="mydb",port=3306);
 mycursor=conn.cursor();
 while True:
  print("Enter Employee-Data :");
  eno=int(input("EmpNo : "));
  ename=input("Emp-Name : ");
  esal=float(input("Emp-Salary : "));
  eaddr=input("Emp-Addr : ");
  sqlcmd = "insert into employees values(%s,%s,%s,%s)";
  mycursor.execute(sqlcmd,(eno,ename,esal,eaddr)); #tuple-of-values
  print("Record Inserted Successfully");
  option=input("Do you want to insert one more record(Yes|No)? ");
  if option=="No":
   conn.commit();
   break;
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();


-----------------------------------------------------------------
Ex6:- (Update-Operation on Table-data)
#Program (DBEx6.py)
(Program to update Employees-Sal with Increment-Sal of Oracle-DB/MYSQL using Dynamic-Input)

NOTE:-
sqlcmd="update employees set esal=esal+%s where esal<%s";
cursorObj.execute(sqlcmd,(increment,salrange)); #tuple-of-values
print("Records Updated Successfully ::",cursorObj.rowcount);

#Program (DBEx6.py)
#(Program to update Employees-Sal with Increment-Sal of Oracle-DB/MYSQL using Dynamic-Input)


#MYSQL
#Increment-Sal by 400 whose sal is less than 4000
import mysql.connector;
try:
 conn = mysql.connector.connect(host="localhost",user="root",password="root",database="mydb",port=3306);
 mycursor=conn.cursor();
 increment = float(input("Enter Increment Salary : "));
 salrange = float(input("Enter Salary Range : "));
 sqlcmd="update employees set esal=esal+%s where esal<%s";
```

```
 mycursor.execute(sqlcmd,(increment,salrange)); #tuple-of-values
 print("Records Updated Successfully ::",mycursor.rowcount);
 conn.commit();
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();

--------------------------------------------------------------------
==> Programs on PDBC(MySQL)
(Deleting the records)
##Programs on Python DBC (MySQL)
Ex7:-
#Program (DBEx7.py)
(Program to delete Employees-Record whose sal is greater than provided Sal using Dynamic-Input)
#delete all emps whose sal is > than 5000
(%s-->string) #unknown values (pass with execute(sqlcmd,unknown))
(%d-->integer)
(%f-->floating)
Format-specifiers(unknown) used for dynamic-input-values in SQL-query

NOTE:-
sqlcmd="delete from employees where esal > %f"; #%s

cursorObj.execute(sqlcmd%cutoffsal);
#for single-value use SQLCmd%single-value

print(cursorObj.rowcount,"Records Deleted Successfully");


#Program (DBEx7.py)
#Program to delete Employees-Record whose sal is greater than provided Sal using Dynamic-Input
#delete all emps whose sal is > than 5000


#MySQL
import mysql.connector;
try:
 conn=mysql.connector.connect(host="localhost",port=3306,user="root",password="root",database="mydb"); #port=3306
 mycursor=conn.cursor();
 cutoffsal = float(input("Enter Cutoff Salary : "));
 sqlcmd="delete from employees where esal > %f";     #%s
 mycursor.execute(sqlcmd%cutoffsal);
    #for single-value use SQLCmd%value
 print(mycursor.rowcount,"Records Deleted Successfully");
 conn.commit();
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();




--------------------------------------------------------------------
=> (select command with fetchone())
Ex8:-
#Program (DBEx8.py)
#Program to select all Emps-Info using fetchone() method
NOTE:-
empno=input("Enter Employee-No :: ");
cursorObj.execute("select * from employees where eno="+empno);
row=cursorObj.fetchone();
if row is not None:
 print(row);
else:
 print("NO Such Record is there");


#Program (DBEx8.py)
#Program to select all Emps-Info using fetchone() method


#MySQLDB
import mysql.connector;
try:
 conn=mysql.connector.connect(host="localhost",user="root",password="root",database="mydb", port=3306);
 mycursor=conn.cursor();
 mycursor.execute("select * from employees where eno='1004'");
 row=mycursor.fetchone();
 print(row);
 print()
 for x in row:
  print(x);
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();


NOTe:-
= Here commit() is not-required
= it is used only for insert/update/delete


--------------------------------------------------------------------
=> (select command with fetchall())
//Programs
Ex9:-
#Program (DBEx9.py)
#Program to select all Emps-Info using fetchall() method
NOTE:-
cursorObj.execute("select * from employees");
data=cursorObj.fetchall();
```

```
print(data);  #list of tuples
#[(),(),(),(),()]
for row in data:
 print("Emp-No :",row[0]);
 print("Emp-Name :",row[1]);
 print("Emp-Sal :",row[2]);
 print("Emp-Addr :",row[3]);
 print("\n");

=*** insert/update/delete directly on DB-command prompt are auto-committed
=** insert/update/delete are manaully committed done from Python program

#Program (DBEx9.py)
#Program to select all Emps-Info using fetchall() method

#MySQLDB
import mysql.connector;
try:
 conn=mysql.connector.connect(host="localhost",user="root",password="root",database="mydb",port=3306);
 mycursor=conn.cursor();
 mycursor.execute("select * from employees");
 data=mycursor.fetchall();
 print(data);  #list of tuples
 for row in data:
  print("Emp-No :",row[0]);
  print("Emp-Name :",row[1]);
  print("Emp-Sal :",row[2]);
  print("Emp-Addr :",row[3]);
  print("\n");
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();



-----------------------------------------------------------------------
=> (select command with fetchmany())
Ex10:-
#Program (DBEx10.py)
#Program to select all Emps-Info using fetchmany() method & provide required no.of rows as dynamic-input
NOTE:-
cursorObj.execute("select * from employees");
n=int(input("Enter No.of Required Rows :"));
data=cursorObj.fetchmany(n);
print(data); #list of tuples [(row1),(row2),(row3),....]
for row in data:
 print(row);
cursorObj.close();


#Program (DBEx10.py)
#Program to select all Emps-Info using fetchmany() method & provide required no.of rows as dynamic-input


#MySQLDB
import mysql.connector;
try:
 conn=mysql.connector.connect(host="localhost",user="root",password="root",database="mydb",port=3306);
 mycursor=conn.cursor();
 mycursor.execute("select * from employees");
 n=int(input("Enter No.of Required Rows :"));
 data=mycursor.fetchmany(n);
 print(data); #list of tuples [(row1),(row2),(row3),....]
 print()
 for row in data:
  print(row);
 #mycursor.close();
except mysql.connector.DatabaseError as e:
 print("Error in executing SQL-cmd :",e);
 if conn:
  conn.rollback();
finally:
 if mycursor:
  mycursor.close();
 if conn:
  conn.close();


================================================================
==> Other MySQL sql-commands::- (For Practice)
mysql> insert into employees(eno,ename,eaddr)
    -> values(1007,'Hari','Amrpt');
Query OK, 1 row affected (0.00 sec)

mysql> select * from employees;
+------+---------+------+--------+
| eno  | ename   | esal | eaddr  |
+------+---------+------+--------+
| 1002 | Ram     | 4000 | Secbad |
| 1003 | Ali     | 4600 | HiTech |
| 1004 | Tom     | 3000 | KPHB   |
| 1005 | Anup    | 4500 | DSNR   |
| 1006 | Krishna | 4950 | KPHP   |
| 1007 | Hari    | NULL | Amrpt  |
+------+---------+------+--------+
6 rows in set (0.00 sec)

-------------------------------------------------------------
mysql> insert into employees
    -> values(1008,'Ravi',NULL,NULL);
Query OK, 1 row affected (0.00 sec)

mysql> select * from employees;
+------+---------+------+--------+
| eno  | ename   | esal | eaddr  |
+------+---------+------+--------+
| 1002 | Ram     | 4000 | Secbad |
| 1003 | Ali     | 4600 | HiTech |
| 1004 | Tom     | 3000 | KPHB   |
| 1005 | Anup    | 4500 | DSNR   |
| 1006 | Krishna | 4950 | KPHP   |
| 1007 | Hari    | NULL | Amrpt  |
| 1008 | Ravi    | NULL | NULL   |
+------+---------+------+--------+
```

```
7 rows in set (0.00 sec)

------------------------------------------------------------
mysql> update employees
    -> set esal=5000
    -> where eno=1007;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from employees;
+------+---------+------+--------+
| eno  | ename   | esal | eaddr  |
+------+---------+------+--------+
| 1002 | Ram     | 4000 | Secbad |
| 1003 | Ali     | 4600 | HiTech |
| 1004 | Tom     | 3000 | KPHB   |
| 1005 | Anup    | 4500 | DSNR   |
| 1006 | Krishna | 4950 | KPHP   |
| 1007 | Hari    | 5000 | Amrpt  |
| 1008 | Ravi    | NULL | NULL   |
+------+---------+------+--------+
7 rows in set (0.00 sec)

------------------------------------------------------------
mysql> update employees
    -> set esal=5600,eaddr='KOTI'
    -> where eno=1008;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from employees;
+------+---------+------+--------+
| eno  | ename   | esal | eaddr  |
+------+---------+------+--------+
| 1002 | Ram     | 4000 | Secbad |
| 1003 | Ali     | 4600 | HiTech |
| 1004 | Tom     | 3000 | KPHB   |
| 1005 | Anup    | 4500 | DSNR   |
| 1006 | Krishna | 4950 | KPHP   |
| 1007 | Hari    | 5000 | Amrpt  |
| 1008 | Ravi    | 5600 | KOTI   |
+------+---------+------+--------+
7 rows in set (0.00 sec)

------------------------------------------------------------
mysql> delete from employees
    -> where eaddr='KPHB';
Query OK, 1 row affected (0.00 sec)

mysql> select * from employees;
+------+---------+------+--------+
| eno  | ename   | esal | eaddr  |
+------+---------+------+--------+
| 1002 | Ram     | 4000 | Secbad |
| 1003 | Ali     | 4600 | HiTech |
| 1005 | Anup    | 4500 | DSNR   |
| 1006 | Krishna | 4950 | KPHP   |
| 1007 | Hari    | 5000 | Amrpt  |
| 1008 | Ravi    | 5600 | KOTI   |
+------+---------+------+--------+
6 rows in set (0.00 sec)
------------------------------------------------------------

mysql> delete from employees
    -> where eaddr='KPHP' and ename='Krishna';
Query OK, 1 row affected (0.00 sec)

mysql> select * from employees;
+------+-------+------+--------+
| eno  | ename | esal | eaddr  |
+------+-------+------+--------+
| 1002 | Ram   | 4000 | Secbad |
| 1003 | Ali   | 4600 | HiTech |
| 1005 | Anup  | 4500 | DSNR   |
| 1007 | Hari  | 5000 | Amrpt  |
| 1008 | Ravi  | 5600 | KOTI   |
+------+-------+------+--------+
5 rows in set (0.00 sec)
------------------------------------------------------------

mysql> update employees
    -> set ename='Ravindra'
    -> where eno=1008;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from employees;
+------+----------+------+--------+
| eno  | ename    | esal | eaddr  |
+------+----------+------+--------+
| 1002 | Ram      | 4000 | Secbad |
| 1003 | Ali      | 4600 | HiTech |
| 1005 | Anup     | 4500 | DSNR   |
| 1007 | Hari     | 5000 | Amrpt  |
| 1008 | Ravindra | 5600 | KOTI   |
+------+----------+------+--------+
5 rows in set (0.00 sec)

------------------------------------------------------------
mysql> select eno,ename from employees
    -> where esal>4500;
+------+----------+
| eno  | ename    |
+------+----------+
| 1003 | Ali      |
| 1007 | Hari     |
| 1008 | Ravindra |
+------+----------+
3 rows in set (0.00 sec)

------------------------------------------------------------
mysql> select eno,ename from employees
    -> where esal<=4500 and eaddr='DSNR';
+------+-------+
| eno  | ename |
+------+-------+
| 1005 | Anup  |
+------+-------+
1 row in set (0.00 sec)
```

```
mysql>
-----------------------------------------------------------------


=================================================================
==>>> Regular Expressions in Python:-
= Representation of strings in particular-format or particular-pattern is done with Regular-Expressions(proper-required-format)
=> tech-def:-
 = It is a declarative mechanism to represent Strings in particular-format or particular-pattern or proper-format
Ex:-
1) RegEx is used to represent Mobile-No's
(10-digits with +91)
Ex:- +91XXXXXXXXX
Ex:- +91-9XXXXXXXXX (8/7/6)
2) Used to represent Email-Id pattern
Ex:-
 userid@domain.xxx
 userid@domain.xx.xx

=> RegEx are used in below apps:-
1) Validation-Frameworks or Validation-Logics
2) Pattern Matching Apps (Ex:- Ctrl+F in Windows, grep in Unix)
3) Developing Translators like Compilers/Interpreters etc
4) Developing Digital Circuits
5) Developing Comm. Protocols like TCP/IP, UDP etc


==> Working with Regular-Expressions:-
NOTE:-
= "re" module in Python is used to work with RegEx
= this module provides different functions to work with RegEx

1) compile()
= Gives RegexObject i.e, converts pattern to RegexObject
Ex:-
 pattern = re.compile("ab");

2) pattern.finditer("org-string") #finditer()
= Gives and Iterator(Looping like) object
i.e, Match-object for every Match

3)
= On Match-object, we use below methods,
 i) start()  -> Gives start-index of the match
 ii) end() -> Gives end+1 index of the match
 iii) group()-> Gives the matched string

Ex:-
#Program (RegEx1.py)
#Program (RegEx1.py)
#Regex with compile() & finditer()

'''
#compile() and finditer()
import re;
#step1
pattern=re.compile("ab");
print(pattern)
print(type(pattern))  #re.Pattern-object

#search pattern
#step-2

matcher=pattern.finditer("abaababaaab"); #Match-obj is callable-iterator-obj
print(matcher)
print(type(matcher))

#step3
count=0;
for mm in matcher:
 count+=1;
 print(mm.start(),"\t",mm.end(),"\t",mm.group())

print("The No.of Occurrence :",count);
'''


#direct-case w.o compile()
#passing pattern directly to finditer()
import re;
count=0;
matcher=re.finditer("ba","abaababaaaba"); #Match-obj(call-iterator-obj)
for mm in matcher:
 count+=1;
 print(mm.start(),"\t",mm.end(),"\t",mm.group());

print("The No.of Occurrence :",count);


NOTE:-
= We can also pass pattern-string directly to finditer() function
Ex:- re.finditer("pattern","org-string")




-- 37th day 37.07-09-22

==>> Type of chars in Regex::-
= Characters are of 5-types
==Diagram==
(a-z, A-Z, 0-9, spaces, sp-chars)


***(General-topic)***
==> How to make Regex-Pattern-string???
(Every RegEx pattern is a string)
= We can make pattern-string in 3-ways
i) Character-classes
ii) Meta-Chars(Esc-Seq-chars)
iii) Quantifiers


Ist-one)
==> Character class:- "[...]" (pattern-string)
= Character classes can be used to search group of characters
1) [abc] -> either a OR b OR c
2) [^abc] -> Except a and b and c
3) [a-z] -> Any lower-case alphabet (either of any char)
4) [A-Z] -> Any upper-case alphabet (either of any char)
```

5) [a-zA-Z] -> Any lower/upper-case alphabet (either of any char)
6) [0-9] -> Any digit from 0 to 9 (either of any digit)
7) [a-zA-Z0-9] -> Any Alpha-Numeric char (either of them)
8) [^a-zA-Z0-9] -> Except Alpha-Numeric chars (none of them)
= Here ^ means "not"

Ex:-
```
#Program (RegEx2.py)
#using character classes
#Program (RegEx2.py)

#using character classes [....]
import re;
x="[abc]";
x="[^abc]";
x="[a-z]";
x="[0-9]";
x="[a-zA-Z0-9]";
x="[^a-zA-Z0-9]";

matcher=re.finditer(x,"a7bc@k9z p1");
for mm in matcher:
 print(mm.start(),"\t",mm.group());
```

**(Meta-Chars) for pattern-string::-
==> Predefined Character classes with ESC.SEQ.CHAR:-
(\ back-slash) Ex::- "\s"
1) \s -> Space-char
2) \S -> Except Space any other char is matched
3) \d -> Any digit from 0 to 9
4) \D -> Any char except digit
5) \w -> Any word char [a-zA-Z0-9] including digits
6) \W -> Any Sp.char except word char (Sp.chars or Spaces-char)
7) . -> Any char including Sp.chars/Spaces

Ex:-
```
#Program (RegEx3.py)
#using pre-defined character classes
#Program (RegEx3.py)
#using pre-defined character classes

import re;
x="\s";
x="\S";
x="\d";
x="\D";
x="\w";
x="\W";
x=".";
matcher=re.finditer(x,"a7b k@9z");
for mm in matcher:
 print(mm.start(),"\t",mm.group());
```

==> Quantifiers in RegEx:- "qty" (pattern-string)
= Quantifiers gives no.of occurrences to given match
1) a -> Exactly one single 'a' is matched
2) a+ -> Atleast one 'a' (1 or more) Ex:- a,aa,aaa,...
3) a* -> Any no.of a's including Zero 'a' (0 or more)
4) a? -> Atmost one 'a' (0 or 1 only)
5) a{m} -> Exactly m-no.of a's
6) a{m,n} -> Min m-no.of a's & Max n-no.of a's

Ex:-
```
#Program (RegEx4.py)
#using quantifiers
#Program (RegEx4.py)

#using quantifiers(no.of.times)
import re;
x="a";
x="a+";
x="a*";
x="a?";
x="a{1}";
x="a{2}";
x="a{3}";
x="a{2,3}";

matcher=re.finditer(x,"abaabaaab");
for mm in matcher:
 print(mm.start(),"\t",mm.group());
```

NOTE:-
1) ^x -> checks whether target string starts with x or not
2) x$ -> checks whether target string ends with x or not
(x is 1 or more chars of above cases)

==> Functions in "re" module:-
1) match()
2) fullmatch()
3) search()
4) findall()
5) finditer()
6) sub()
7) subn()
8) split()
9) compile()

```
#Program (RegEx5.py)
```

1) match()
= Checks given pattern at beginning of original-string
= If it matches then we get Match-object(start(),end(),group) o.w None
Ex:-
```
#Program (RegEx5.py)
#using match()
```

2) fullmatch():-
= Here complete pattern is matched in Original-String
= If it matches then we get Match-object(start(),end(),group) o.w None
Ex:-

```
#Program (RegEx5.py)
#using fullmatch()


3) search():-
= It searches for 1st occurence of pattern in original-string
= If it matches then we get Match-object o.w None
Ex:-
#Program (RegEx5.py)
#using search()

4) findall():-
= Finds all occurrences of pattern-string in original-string
= It returns list-object with all occurrences
Ex:-
#Program (RegEx5.py)
#using findall()


5) finditer():-
= Gives iterator(loop) like Match-object for each successful match
= On Match-object, we can use start(),end(),group() functions
Ex:-
#Program (RegEx5.py)
#using finditer()


6) sub():-
= It means substitution or replacement
= re.sub(regex,replacement,targetstring)
= In target-string,every matching-pattern(regex) is replaced with replacement-string
Ex:-
#Program (RegEx5.py)
#using sub()


7) subn():-
= It is same as sub() but it also returns no.of replacements
= It returns/gives tuple
i.e, (newstring,no.of replacements)
Ex:-
#Program (RegEx5.py)
#using subn()


8) split():-
= Splits the original-string using given pattern-string
= It returns list of all tokens
Ex:-
#Program(RegEx5.py)
#using split()


9) ^ symbol with search():-
= It checks whether our original-string starts with our pattern-string or not
Ex:-
 result = re.search("^Welcome",org-string);
= If org-string starts with 'Welcome' then it returns Match-object o.w None
Ex:-
#Program(RegEx5.py)
#using ^ symbol


10) $ symbol with search():-
= It checks whether our original-string ends-with our pattern-string or not
Ex:-
 result = re.search("Python$",org-string);
= If org-string ends-with 'Python' then it returns Match-object o.w None
Ex:-
#Program(RegEx5.py)
#using $ symbol

NOTE:-
= If we wish to ignore the case then pass "re.IGNORECASE" as 3rd-arg to search()
Ex:-
result = re.search(pss,ss,re.IGNORECASE);



#Program (RegEx5.py)
#re module functions


#using match()
import re;
ss =input("Enter Pattern-string to Check : ");
mm=re.match(ss,"abcabdefg");
if mm!=None:
 print("Matching is done at the beginning...");
 print("Start-Index :",mm.start(),"\tEnd-Index",mm.end()-1);
else:
 print("Match is NOT-available at the beginning...");
'''


'''
#using fullmatch()
import re;
ss =input("Enter Pattern to Check : ");
#mm=re.fullmatch(ss,"ababab");
mm=re.fullmatch(ss,"Welcome to Python Session")
if mm!=None:
 print("Full-Matching is done...");
 print("Start-Index :",mm.start(),"\tEnd-Index",mm.end()-1);
else:
 print("Full-Match is NOT-available...");
'''

'''
#using search()
import re;
ss =input("Enter search-Pattern to Check : ");
mm=re.search(ss,"ababab");
#mm=re.search(ss,"Welcome to Python Session");
if mm!=None:
 print("Search-Matching is done...");
 print("Start-Index :",mm.start(),"\tEnd-Index",mm.end()-1);
else:
 print("Search-Matching is NOT-available...");
'''
```

```
'''
#using findall()
import re;
ss=input("Enter finding-Pattern to Check : "); #[0-9],[a-e],\s
listl=re.findall(ss,"abababab");
#listl=re.findall(ss,"Welcome to Python Session");
print(listl);
'''


'''
#using finditer()
import re;
ss=input("Enter finding-Pattern : ");
iter=re.finditer(ss,"Welcome to Python Session");
for mm in iter:
 print(mm.start(),"\t",mm.end()-1,"\t",mm.group());
'''


'''
#using sub()
import re;
ss=input("Enter Matching-Pattern : ");
ress=input("Enter Replacing-String : ");
newss=re.sub(ss,ress,"Welcome to Python Session");
print(newss);
'''


'''
#using subn()
import re;
ss=input("Enter Matching-Pattern : ");
ress=input("Enter Replacing-String : ");
tupl=re.subn(ss,ress,"Welcome to Python Session");
print(tupl);
print("New-String :",tupl[0]);
print("No.of Relacements :",tupl[1]);
'''


'''
#using split()
import re;
ss=input("Enter Original-string : ");
pss=input("Enter pattern-String : ");
listl=re.split(pss,ss);
print(listl);
for ll in listl:
 print(ll);
'''


'''
#using ^ symbol
import re;
ss=input("Enter Original-string : ");
pss=input("Enter ^ pattern-String : ");
result = re.search(pss,ss);
if result!=None:
 print("Original-String starts with Pattern-String");
else:
 print("Original-String does not starts with Pattern-String");




#using $ symbol
import re;
ss=input("Enter Original-string : ");
pss=input("Enter $ pattern-String : ");
result = re.search(pss,ss);
#result = re.search(pss,ss,re.IGNORECASE);
if result!=None:
 print("Original-String ends-with Pattern-String");
else:
 print("Original-String does not end-with Pattern-String");


-----------------------------------------------------------------------

==> Regex-Case-Studies:- (Real-time Examples)

#Program (RegEx6.py)
-------------------------------------------
1) WAP for RegEx to represent all 10-digits Mobile Number
Rules:
a) Mobile-No should be exactly 10-digit
b) 1st-digit should be 7 or 8 or 9
Ex:-
[6-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]
(or)
[6-9][0-9]{9}
(or)
[6-9]\d{9}


NOTE:-
= We can use (91-), (0091) for group of chars in RegEx-Patterns
= (91|0091|[+]91|[+]91-) in regex grouping, we can use | for either of the selection inside group

NOTE:-
= (0091|91|[+]91) indicated either 0091 or 91 or +91
Here () means grouping of pattern/chars
= | indicates or
= [+] indicates 1-char is + before 91 o.w + (atleast one time)


2) WAP for email-id validation
3) WAP Vehicle reg-no validation


#Program (RegEx6.py)
#Regex real-time-case-studies


#Valid 10-digit Mobile-No (1st-digit[6-9]& remaining 9-dig[0-9])
```

```
import re;
num=input("Enter Mobile-No : ");
mm = re.fullmatch("[6-9]\d{9}",num);
if mm!=None:
 print(num,"is a Valid-Mobile-No");
else:
 print(num,"is NOT a Valid-Mobile-No");
'''


'''
#Check whether given Mobile-No is Valid or NOT in India-code(+91)
import re;
ss = input("Enter Mobile-No : ");
mm = re.fullmatch("(0091|91|[+]91|[+]91-)?[6-9][0-9]{9}",ss);
if mm!=None:
 print("Valid Mobile-No");
else:
 print("Invalid Mobile-No");
'''


'''
#Check whether given Email-Id is Valid or NOT
import re;
ss = input("Enter Email-ID : ");
mm = re.fullmatch("\w[a-zA-Z0-9_.]*@gmail[.]com",ss);
if mm!=None:
 print("Valid Email-Id");
else:
 print("Invalid Email-Id");



#Check whether given Vehicle Reg-No is Valid or NOT in TS
import re;
ss = input("Enter Vehicle Reg-No : ");
mm = re.fullmatch("(TS|AP)[0123][0-9][A-Z]{1,2}\d{4}",ss);
if mm!=None:
 print("Valid Registration-No");
else:
 print("Invalid Registration-No");



# -- 38th day 38.08-09-22
"""
==>>> Decorator Functions:-
= It is function, which takes another-function as argument and extend its functionality and returns/gives modified-function
==Diagram==
Ex1:-
I/P Function (wish()--->"@Decorator"---> new(add some functionality) using inner()
Ex2:-
I/P Function (wish()--->"Decorator-Function"---> Output-Function with Extended Functionality

[our-func+@decorator/decorator-function ---> extra-func-our-func]


=> Advantage,
= Decorator-Function extends the functionality of existing functions without modifying that function
Ex1:-
#Program (DecoratorEx1.py)

NOTE:-
=* Here wishes(), prints Same message for all user-names
= However to modify messages for different user-names, we can do this without touching wishes() & it is possible with @decorator/decorator-function



=> Using @decorator/decorator-function in program
Ex2:-
#Program (DecoratorEx1.py)
@decorator is passed or used before our function-def

NOTE:=
= Here for every wishes() function call, decor(func) is executed automatically
= For decorator-function, our function-ref is passed as input-para
= Inside decorator-function, inner() provides extra-functionality
= inner() takes same input-para as that of our function
= Finally, decorator-function returns inner() as return-value

==> How to call Same-Function with Decorator and without Decorator:-
(with @decor and W.O @decor)
Ex3:-
#Program (DecoratorEx1.py)

NOTE:-
= It is done with func-ref-var to decorator-function with input-para as org-function



==> Decorator Chaining:-
= We can define multiple-decorators for same-function and all these decorators will form Decorator-Chaining
Ex:-
@decor1
@decor
def num():
 ....
=** Here for num(), we are applying 2 decorator functions
(1st inner-decorator works & then 2nd outer-decorator works)
Ex6:-
"""
#Program (DecoratorEx1.py)



#Program (DecoratorEx1.py)
#Program to work with diff-decorators


#case-1
#same-msg w.o decorators
def wishes(name):
 print("Hello :",name,"Good Morning");

wishes("Sai");
wishes("Ram");
wishes("Ali");
'''
```

```
'''
#case-2
#diff-msgs using with decorator
def decor(func): #wishes() is passed here [func=wishes]
 def inner(name): #inner() gives extra-func. to wishes()
  if name=="Ram":
   print("Hello :",name,"Good Afternoon");
  elif name=="Ali":
   print("Hello :",name,"Good Evening");
  else:
   func(name);
 return inner;

@decor
def wishes(name):
 print("Hello :",name,"Good Morning");

wishes("Sai");
wishes("Ram");
wishes("Ali");


#sp-case (using decorator-func but w.o @decorator (manually)
print();
decorfunction = decor(wishes);
#func-ref-var to decor() with wishes()-i/p
decorfunction("Sai"); #decorator func is executed
decorfunction("Ram");
decorfunction("Ali");



#chaining of decorators
#Decorator-Chaining (using multiple-@decorators)
def decor1(func):
 def inner():
  x=func();
  return x*x*x;
 return inner;

def decor(func):
 def inner():
  x=func();
  return x*x;
 return inner;

@decor1  #2nd-exec outer-dec
@decor  #1st-exec inner-dec
def num():
 return 5;

print(num());


"""
-----------------------------------------------------------------
==>>> Generator Functions:-
= It is function responsible for generating sequence of values
Ex:-
 r1 = range(10) ----> 0 to 9 (pre-defined generator)
= We can write this function, just like regular-functions but here we use "yield-keyword" to return values
Ex:-
 yield 1;  #1st-value
 yield "Sai"; #2st-value
 yield 2;  #3rd-value
 etc....
==Diagram==
yield <---- "Generator-Function" ----> Sequence-of-values

Ex1:-
#Program (GeneratorEx1.py)
NOTE:-
= next(gen-var) is used to get/access values from generator-function
"""

Ex2:-(Generator with Loops)
#Program (GeneratorEx1.py)

Ex3:-
#Program (GeneratorEx1.py)
NOTE:-
= We can convert generator into list as follows,
Ex:-
listl = list(values);
print(listl);

"""
=> Advantages:-
1) Compared to class-level iterators, generators are easy to use
2) Improves Memory Utilization and Performance
3) Best suitable for reading-data from Large No.of Files
4) Works best for web-scraping and crawling
"""

#Program (GeneratorEx1.py)
#Program to work with generators

#generator-func
def mygenf1():
 yield 'A';
 yield 'B';
 yield 'C';

g1 = mygenf1();
print(g1);
print(type(g1));
print()
#get values from gererator-func using next()
print(next(g1));
print(next(g1));
print(next(g1));
#print(next(g1));
'''


'''
#another-example with Loops
def countdown(num):
```

```
 print("Start Countdown...");
 while(num>0):
  yield num;
  num=num-1;

values=countdown(10);
#values=countdown(20);
print(values);
for x in values:
 print(x);



#with lists using list()
def firstnnums(num):
 i=1;
 while(i<=num):
  yield i;
  i=i+1;

values=firstnnums(10);
listl = list(values);
print(listl);




# 39th day 10.09.22
# ------------------------------------------
"""
****Data-Science Modules in Python*****
= Data-Science means working with Big-data in program
Ex:-
 Getting-data
 Formatting-data
 Searching-data
 Sorting-data
 Cleaning-data
 Ananlysing-data
 etc...
= To work with data-science in python, we have 4-modules,
 a) NumPy
 b) Pandas
 c) SciPy or SciKit
 d) MatPlotLib

------------------------------------------
===>>> NumPy Arrays in Python:-
=> What is NumPy?
= NumPy means Numerical-Python
= NumPy is a python-library, which is used for working with Arrays in Python
= It also provides functions to work with Linear-Algebra, Matrices in Mathematics etc
= It was created by Travis Oliphant(2005)
= It is an open-source project and we can use it freely
(NumPy stands for Numerical-Python)

=> Why NumPy?
= In Python we have Lists & array-Module for Arrays concept, but they are slow to processing data
= NumPy provides an array-object (it is 50-times faster than Python-lists & Python-arrays)
= The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
= Arrays are used in Data-Science

NOTE:-
= NumPy are Faster than Lists because they are stored at one continuous place in memory unlike lists
= It is optimized to work with latest CPU architectures
= It is a Python library written in Python partially & in C or C++ for faster computations


==> Installation of NumPy:- (External-Library)
= Python and PIP should be already installed
= Next install NumPy
= Command goes like this,
cmd> pip install numpy
(latest-version is numpy-1.23.3) -> Python-3.10.7
(internet-conn is compulsory)

NOTE:-(How to use NumPy in program?)
import numpy;

//Program... (NumpyEx1.py)
#Program to work with NumPy module
import numpy;
arr = numpy.array([1, 2, 3, 4, 5]);
print(arr);

NOTe::-
= NumPy-Arrays are represented in [] brackets w.o ,(commas)

NOTE:-
= How to get numpy-version,
//Program... (NumpyEx1.py)
import numpy as np;
print(np.__version__);

==>How to Create a NumPy ndarray:-
= NumPy is used to work with arrays
= In NumPy, it is called ndarray
(it is done by using the array() function)
Ex:-
//Program... (NumpyEx1.py)
import numpy as np;
arr = np.array([11, 22, 33, 44, 55]);
print(arr);
print(type(arr));

NOTE:-
= To create an ndarray, we can pass a list[], tuple(),set{} or any array-like object into the array() method and it is converted into "ndarray"

Ex:- (NumpyEx1.py)
import numpy as np;
arr = np.array((11,12,13,14,15)); //[list],(tuple),{set}
print(arr);
"""

##Program... (NumpyEx1.py)
#Program to work with NumPy module (NumpyEx1.py)


import numpy;
```

```
arr = numpy.array([10, 20, 30, 40, 50]);
print(arr);
print(type(arr));
'''

'''
import numpy as np;
print(np.__version__);



import numpy as np;
arr = np.array((11,12,13,14,15)); #[list],(tuple),{set}
print(arr);
print(type(arr));



"""
==> NumPy Array Dimensions:- (0D/1D/2D/3D etc)
1) 0-D Arrays:-
= 0-D Arrays(Scalars), are the elements in an array
= Each value in an array is a 0-D array

Ex:- (NumpyEx2.py)
#Program to Create a Dimensional-Array (0D/1D/2D/3D etc)
#0D Array
import numpy as np;
arr = np.array(11);
print(arr);
print(type(arr));

2) 1-D Arrays:-
= An array which has 0-D arrays as its elements is called Single/Uni-dimensional or 1-D array

Ex:- (NumpyEx2.py)
#1D Array
import numpy as np;
arr = np.array([15, 25, 35, 45, 55]);
print(arr);
print(type(arr));


3) 2-D Arrays:-
= An array that has 1-D arrays as its elements is called a 2-D array
(Mainly used to Tables(rows&cols), Matrices etc)

Ex:- (NumpyEx2.py)
#2D Array
import numpy as np;
arr = np.array([[1, 2, 3], [4, 5, 6], [7,8,9]]);
print(arr);
print(type(arr));

=** Each array-represents 1-row & values as cols-elements

4) 3-D arrays:-
= An array that has 2-D arrays (matrices) as its elements is called 3-D array

Ex:- (NumpyEx2.py)
#3D Array
import numpy as np;
arr = np.array([ [[11,22,33],[44,55,66]], [[10,20,30],[40,50,60]] ]);
print(arr);
print(type(arr));

NOTE:-
=> How to Check Array-Dimensions?
= "ndim" attribute gives array-dimension (numpy-module)
Ex:- (NumpyEx2.py)
#ndim Array
"""
import numpy as np;
arr1 = np.array(11);
arr2 = np.array([15, 25, 35, 45, 55]);
arr3 = np.array([[1, 2, 3], [4, 5, 6], [7,8,9]]);
arr4 = np.array([ [[11,22,33],[44,55,66]], [[10,20,30],[40,50,60]] ]);
print(arr1.ndim);
print(arr2.ndim);
print(arr3.ndim);
print(arr4.ndim);


#Program to Create a Dimensional-Array (0D/1D/2D/3D etc)
#NumPyEx2.py


#0D Array
import numpy as np;
arr = np.array(11);
print(arr);
print(type(arr));
'''

'''
#1D Array
import numpy as np;
arr = np.array([15, 25, 35, 45, 55]);
print(arr);
print(type(arr));
'''

'''
#2D Array (rows & cols)
import numpy as np;
arr = np.array([[1, 2, 3], [4, 5, 6], [7,8,9]]); #nested-lists
print(arr);
print(type(arr));
'''

'''
#3D Array
import numpy as np;
arr = np.array([ [[11,22,33],[44,55,66]], [[10,20,30],[40,50,60]] ]);
print(arr);
print(type(arr));
'''

'''
#ndim-var Array
```

```
import numpy as np;
arr1 = np.array(11);
arr2 = np.array([15, 25, 35, 45, 55]);
arr3 = np.array([[1, 2, 3], [4, 5, 6], [7,8,9]]);
arr4 = np.array([ [[11,22,33],[44,55,66]], [[10,20,30],[40,50,60]] ]);
print(arr1.ndim);
print(arr2.ndim);
print(arr3.ndim);
print(arr4.ndim);


#n-Dimension
import numpy as np;
arr = np.array([1, 2, 3, 4], ndmin=5);
print(arr);
print(arr.ndim);



******
==> Accessing NumPy Array Elements:-
(Using-Indexes)
= NumPy Array-indexing is same as any array
i.e, 0 to (n-1) (F->L) Forward-direction
-1 to -n (L->F) Backward-direction

Ex:- (NumpyEx3.py)
#NumPy Array Accessing-Elements(with Indexes)
import numpy as np;
arr = np.array([11, 22, 33, 44, 55]);
print(arr[0]);
print(arr[1]);
print(arr[2]);
print(arr[3]);
print(arr[4]);
print(arr[-1]);
print(arr[-2]);
print(arr[-3]);
print(arr[-4]);
print(arr[-5]);

=** The best-way to access the elements of array is Loops
(for-Loop)
Ex:-
#loops access
for x in arr:
 print(x);
i=0;
while i<len(arr):
 print(arr[i]);
 i=i+1;

=* len(arr) function gives length of an NumPy-array


=> Accessing 2-D Arrays with Indexes:-
= It is done with comma separated integers representing the dimension and the index of the element
Ex:-
 arr[0,0] or a[0][0]
 arr[1,1] or a[1][1]
 arr[2,2] or a[2][2]

Ex:- (NumpyEx3.py)
#Access 2D Arrays with indexes arr[i,j] or arr[i][j]
import numpy as np;
arr = np.array([ [1,2,3], [4,5,6], [7,8,9] ]);
print(arr);
print(arr[0,0],"\t",arr[0,1],"\t",arr[0][2]); #1st-row
print(arr[1,0],"\t",arr[1,1],"\t",arr[1][2]); #2nd-row
print(arr[2,0],"\t",arr[2,1],"\t",arr[2][2]); #3rd-row

#with Nested-Loops
i=0;
while i<3:
 j=0;
 while j<3:
  print(arr[i][j],end="\t");  #arr[i,j]
  j=j+1;
 print();
 i=i+1;

NOTE:-
= Similarly we can work with 3-D and n-D arrays
Ex:-
 arr[0,0,0] or arr[0][0][0]
= Also we can use -ve indexes for accessing 2D or 3D or n-D arrays
Ex:- (NumpyEx3.py)
#-ve indexes
import numpy as np;
arr = np.array([[1,2,3],[4,5,6]])
print(arr[1,-1]); #2nd-row, last-element
print(arr[-1][-3]); #2nd-row, 1st-element


#NumPy Array Accessing-Elements(with Indexes)
#(NumpyEx3.py)

'''
import numpy as np;
arr = np.array([11, 22, 33, 44, 55]);
print(arr[0]);
print(arr[1]);
print(arr[2]);
print(arr[3]);
print(arr[4]);
print(arr[-1]);
print(arr[-2]);
print(arr[-3]);
print(arr[-4]);
print(arr[-5]);

print()
#loops access
for x in arr:
 print(x);

print();
i=0;
while i<len(arr):
 print(arr[i]);
```

```
 i=i+1;
'''

'''
#Access 2D Arrays with indexes arr[i,j] or arr[i][j]
import numpy as np;
arr = np.array([ [1,2,3], [4,5,6], [7,8,9] ]);
print(arr);
print(arr[0,0],"\t",arr[0,1],"\t",arr[0][2]); #1st-row
print(arr[1,0],"\t",arr[1,1],"\t",arr[1][2]); #2nd-row
print(arr[2,0],"\t",arr[2,1],"\t",arr[2][2]); #3rd-row

print()
#with Nested-Loops
i=0;
while i<3:
 j=0;
 while j<3:
  print(arr[i][j],end="\t");  #arr[i,j]
  j=j+1;
 print();
 i=i+1;
'''


#-ve indexes
import numpy as np;
arr = np.array([[1,2,3],[4,5,6]])
print(arr[1,-1]); #2nd-row, last-element
print(arr[-1][-3]); #2nd-row, 1st-element



***==> NumPy Array Slicing:-
= Slicing arrays means taking elements from one given index to another given index
= It is done as follow [start:end] #end-index-value not included
= It can have step-value also, Ex:- [start:end:step]
= default-start-value(0)
= default-end-value(length-of-array)
= default-step-value (+1)

Ex:- (NumpyEx4.py)
#NumPy-array Slicing
import numpy as np;
arr = np.array([11,22,33,44,55,66,77,88]);
print(arr[1:5]);
print(arr[4:]);
print(arr[:4]);
#-ve slicing
print(arr[-3:-1]);
#step-value
print(arr[1:5:2]);
print(arr[::2]);


=> 2D-Array Slicings:-
= Here we use arr[b:e:s][b:e:s] or arr[b:e:s, b:e:s]
Ex:-(NumpyEx4.py)
#2D Array-Slicings
import numpy as np;
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]]);
print(arr[1, 0:2]);
print(arr[0:2,2]);
print(arr[0:2, 1:4]);


#Program to work with NumPy-array Slicings
#NumpyEx4.py


'''
#NumPy-array Slicing
import numpy as np;
arr = np.array([11,22,33,44,55,66,77,88]);
print(arr[1:5]);
print(arr[4:]);
print(arr[:4]);
#-ve slicing
print(arr[-3:-1]);
#step-value
print(arr[1:5:2]);
print(arr[::2]);
'''

#2D Array-Slicings
import numpy as np;
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]]);
print(arr[1, 0:2]);
print(arr[0:2,2]);
print(arr[0:2, 1:4]);



==========================================================================
==>> Pandas in Python:-  (Tables & its columns-data)
= Pandas is a Python library, it is mainly used to "ANALYZE-Data"
(Data-Analysis)
= It is a Module (pandas)
= It is used as follows,
Ex:-
 import pandas as pd;
 pd.methodName()....;

NOTE::-
==> What?
= Pandas is a Python library, which is used to work with data-sets (records or table-data)
= It provides functions for,
 data-analysis,
 data-cleaning,
 data-exploring,
 data-manipulations

= "Panel-Data" + "Python Data Analysis" ==> Pandas
= It was developed by "Wes McKinney"(2008)

=> Advantage,
 = Analyze big data and make conclusions (statistical theories)
 = Cleans confusion in data, and make them easy to work
 = Such data is use in data-science
```

Example::-
= Pandas provide correlation between 2 or more columns in a table-data
(i.e, Average/Max/Min value etc)
= Pandas can delete rows in table that are of no-use, or has wrong value (empty or NULLs) --> "Pandas-Data-Cleaning"

==> How to install Pandas??
= Make-sure to have python & pip already installed in your system & give the below,
cmd> pip install pandas
pandas-1.4.4

**(py -m pip install --upgrade pip)

Ex:-
//Program (PythonPandasEx1.py)
(Program to demo Pandas usage in python)
```
#Program to demo Pandas usage in python
#(PythonPandasEx1.py)

'''
#data as dict-type using DataFrame() method
import pandas;
data1 = {
  'names': ["Sai", "Ram", "Ali"],
  'ages': [23, 27, 20],
  "heights" :[6.2, 6.4, 5.9]
};
newpandasdata1 = pandas.DataFrame(data1);
print(newpandasdata1);
'''

'''
print();
#import alias
import pandas as pd;
data1 = {
  'names': ["Sai", "Ram", "Ali","Krishna"],
  'ages': [23, 27, 20,24],
  "heights" :[6.2, 6.4, 5.9,5.5]
};
newpandasdata1 = pd.DataFrame(data1);
print(newpandasdata1);
'''


print();
#pandas __version__
import pandas as pd;
print(pd.__version__);
```


==>>> Pandas Series::- (table-column-data)
= Pandas Series is like a data in a table-columns
= It is coll.of.data like 1D-array (any type)
= In "pandas" module we have "Series()" method, to form a Pandas-series
Ex:-
//Program (PythonPandasEx2.py)
(Program to demo Pandas series using Series() method)
```
##Program (PythonPandasEx2.py)
#Program to demo Pandas series using Series() method

'''
#series from a list
import pandas as pd;
list1 = [11,88,55,22,99];
x = pd.Series(list1);
print(x);

#labels(indexes)
print(x[0]);
print(x[1]);
print(x[2]);
print(x[3]);
print(x[4]);
'''

#own labels (row-names)
import pandas as pd;
list1 = [11,88,55,22,99];
x = pd.Series(list1,index=["a","b","c","d","e"]);
print(x);
print(x["a"]);
print(x["b"]);
print(x["c"]);
print(x["d"]);
print(x["e"]);
```


NOTE:-
=> Pandas-Series "Labels":-
= The values of Pandas-Series are labeled with index number (0 to n-1)
Ex:- 0,1,2,...etc
= Label can be used to access Pandas-Series values
Ex:-
 x[0], x[1],....

=> Create labels?
= It is done with index=[....] argument to Series() method
Ex:-
x = pd.Series(list1, index=["a","b",....]);
(here we can have our own indexes to Pandas-Series values)
= Access with our own labels,
Ex:-
 x["a"], x["b"],....

==> Pandas Series with (Key:Value) pairs::-
= We can have, Key:Value pair Objects as Pandas-Series
(for this, we can use dictionary type)
Ex:-
```
students_data = {1001:"Sai", 1002:"Ram", 1003:"Ali"};

//Program (PythonPandasEx3.py)
(Program to demo Pandas series with (Key:Value) pairs of dict-type)
#Program (PythonPandasEx3.py)
```

```
#Program to demo Pandas series with (Key:Value) pairs of dict-type

'''
import pandas as pd;
students_data = {1001:"Sai", 1002:"Ram", 1003:"Ali"};
x = pd.Series(students_data);
print(x);

print();
#keys as labels(indexes)
print(x[1001]);
print(x[1002]);
print(x[1003]);
#print(x[1004]); #KeyError
'''

print();
#Series with specific elements of dictionary
import pandas as pd;
students_data = {1001:"Sai", 1002:"Ram", 1003:"Ali"};
x = pd.Series(students_data, index=[1001,1003]);
print(x);



NOTE:-
= Here dict-type "keys" are taken as labels(indexes) for Pandas-Series values
= such keys can be used to access individual values of Pandas-Series
Ex:-
 x[1001], x[1002],...

**=> Here we can include only specific elements of a dictionary as Pandas-Series values using index=[...] argument to Series() method
Ex:-
x = pd.Series(students_data, index=[1001,1003]);



*** (Pandas DataFrames)
===>> Pandas-Series DataFrames::-
= Data sets in Pandas are multi-dimensional tables, known as DataFrames
= Series is like a column & DataFrame is like a whole table
=** For this, we use DataFrame() method
Ex:-
students_data={
 "rollno":[1001,1002,1003],
 "names":["Sai","Ram","Ali"]
};
#pd.DataFrame(students_data);

//Program (PythonPandasEx4.py)
(Program to demo Pandas series with DataFrames)
#Program to demo Pandas series with DataFrames
#PythonPandasEx4.py

'''
#Series with DataFrames
import pandas as pd;
students_data={
 "rollno":[1001,1002,1003],
 "names":["Sai","Ram","Ali"]
};
x = pd.Series(students_data);
print(x);
print();
x = pd.DataFrame(students_data);
print(x);


#Locating Row loc[index]
print();
df = pd.DataFrame(students_data);
print(df.loc[0]);
print();
print(df.loc[1]);
print();
print(df.loc[2]);
#print(df.loc[3]);


print();
#with list of indexes(sub-list)
print(df.loc[[0,1,2]]);
print(df.loc[[0,1]]);
print(df.loc[[0]]);
#print(df.loc[[0,1,2,3]]);
'''


print();
#Named-indexes
import pandas as pd;
students_data={
 "rollno":[1001,1002,1003],
 "names":["Sai","Ram","Ali"]
};
df = pd.DataFrame(students_data, index=["Stud1", "Stud2", "Stud3"]);
print(df);

print();
print(df.loc["Stud1"]);
print(df.loc["Stud2"]);
print(df.loc["Stud3"]);
#print(df.loc["Stud4"]);


NOTE:-
= Pandas DataFrame is a 2D data structure, same-like a 2D-array, or a table with rows and columns

=> Pandas DataFrame Locate-Rows:-
(working with rows)
= To get rows from Pandas DataFrame, we use "loc" attribute
(it gives specified rows with given index(0 to n-1))
Ex:-
df = pd.DataFrame(students_data);
 df.loc[0];
 df.loc[1];
 df.loc[2];
 #df.loc[3];  #ValueError & KeyError

//Program (PythonPandasEx4.py) ==SAME==
```

(Program to demo Pandas series with DataFrames)

Ex:- (list of indexes)
df.loc[[0,1,2]];
=** Here result is Pandas DataFrame (using [list-of-indexes])

=> Named indexes::-
= using index="" attribute for DataFrame() method, we can give our own indexes to Pandas-DataFrame
Ex:-
df = pd.DataFrame(students_data, index = ["Stud1", "Stud2", "Stud3"]);
print(df);

=** instead of indexes(0,1,2,...), we get (Stud1,Stud2,Stud3)

=> Locate-Row with Named Indexes::-
= Use "loc" attribute to get the specified row with attribute-name
Ex::-
print(df.loc["Stud1"]);
print(df.loc["Stud2"]);
print(df.loc["Stud3"]);
#print(df.loc["Stud4"]); #KeyError

==>> Pandas with CSV files:-
(Reading data from CSV files)
= Pandas provide functions using with we can work with Big Data-Sets like CSV files (comma separated values)
Ex:-
RollNo,Name,Age,Height
1001,Sai,21,6.2
1002,Ram,22,5.9
1003,Ali,23,5.5
1004,John,20,6
1005,Krishna,24,5.1
1006,Tom,21,5.2
1007,Anup,19,5.8
1008,Sita,18,5.6
1009,Kishore,24,5.4
1010,Pavan,20,5.5

= CSV files contains plain-text with data-sets
Ex:-
 student_data.csv
 employee_data.csv

=** to read data from CSV files, we use read_csv() function from pandas-module

Ex:-
import pandas as pd;
df = pd.read_csv("student_data.csv")
print(df);
print(df.to_string());

#Program (PythonPandasEx5.py)
#Program to demo Pandas series with CSV files

import pandas as pd;
df = pd.read_csv("student_data.csv")
print(df);
print(type(df));
print();
print(df.to_string());
print(type(df.to_string()));

print();

==>> Pandas with JSON files:-
(Reading data from JSON files)
(Javascript Object-notation i.e, dict-data)
= Storing Big-data in the form of Key:Value pairs is done with JSON
Ex:-
"student_data.json"
{
 "rollno" : {"0":1001,"1":1002,"2":1003},
 "sname" : {"0":"Sai","1":"Ram","2":"Ali"},
 "age" : {"0":21,"1":"23","2":"20"},
 "height" : {"0":6.2,"1":"5.5","2":"5.9"},
 "address" : {"0":"hyderabad","1":"Secbad","2":"Hitech"}
}

= JSON is a plain-text data-set, it is javascript object {....}
= It is used as a Data-transfer from server to client Ui-Apps
(partial page updates)

=** To read data from JSON files, we use read_json() function from pandas-module
Ex:-
import pandas as pd;
df = pd.read_json("student_data.json");
print(df);
print(type(df));
print(df.to_string());

//Program (PythonPandasEx6.py)
(Program to demo Pandas series with JSON files)

=> If .json file is not available, then we can use python "dict" variable in program to read json-data
(for this we use DataFrame() function)
Ex:-
student_data = {
 "rollno" : {"0":1001,"1":1002,"2":1003},
 "sname" : {"0":"Sai","1":"Ram","2":"Ali"},
 "age" : {"0":21,"1":"23","2":"20"},
 "height" : {"0":6.2,"1":"5.5","2":"5.9"},
 "address" : {"0":"hyderabad","1":"Secbad","2":"Hitech"}
};
df = pd.DataFrame(student_data);

#Program to demo Pandas series with JSON files
#(PythonPandasEx6.py)

'''
import pandas as pd;
df = pd.read_json("student_data.json");

```
print(df);
print(type(df));
print();
print(df.to_string());
'''


print();
#dict-data as json-data
import pandas as pd;
student_data = {
 "rollno" : {"0":1001,"1":1002,"2":1003},
 "sname" : {"0":"Sai","1":"Ram","2":"Ali"},
 "age" : {"0":21,"1":23,"2":20},
 "height" : {"0":6.2,"1":5.5,"2":5.9},
 "address" : {"0":"hyderabad","1":"Secbad","2":"HitechCity"}
};
df = pd.DataFrame(student_data);
print(df);



#40th day 09-12-22

==>> SciPy(SciKit) Arrays in Python:-
(Data-Science Module)
= SciPy stands for Scientific Python
= SciPy is a scientific calculation library that internally uses NumPy only


==> SciPy Introduction::-
= SciPy is a scientific calculations library, which uses NumPy only
= SciPy stands for Scientific Python (SciKit)
= SciPy is open-source module (python-lib)
=**
 SciPy was invented by "Travis Olliphant" (same for NumPy)

=> Why SciPy?
= It gives more optimized functions than NumPy library
= SciPy is written in Python only


==>> How to install SciPy module??
(pre-requisites)
= Makesure Python & Pip are already installed in system
cmd> pip install scipy

#Program (PythonSciPyEx1.py)
(Program to demo SciPy module)
#Program to demo SciPy module (PythonSciPyEx1.py)

from scipy import constants;
print(constants.liter);  #1-liter as cubic-meters (mathematical-constants)

import scipy;
print(scipy.__version__);



==> Working with SciPy Constant Units::-
= SciPy provides built-in scientific constants
Ex::-
 pi
Ex:-
from scipy import constants;
print(constants.pi);

#Program (PythonSciPyEx2.py)
(Program to demo SciPy module with Constants)

=> listing all constants units,
Ex:-
dir(constants);
print(constants.year);
print(constants.week);
etc...

NOTE:-
=> Unit Categories (for all Constants)
(Metric,Binary,Mass,Angle,Time,Length,Pressure,Area,Volume,Speed,Temperature,Energy,Power,Force)


#Program to demo SciPy module with Constants
#Program (PythonSciPyEx2.py)

from scipy import constants;
print(constants.pi);

print();
print(dir(constants));

print();
print(constants.e);
print(constants.kilo);
print(constants.gram);

print(constants.hour);
print(constants.minute);

print(constants.mile);
print(constants.day);

print(constants.carat);
print(constants.week);

print(constants.speed_of_sound);
print(constants.speed_of_light);

print(constants.sigma);



==> Working with SciPy Optimizers::-
= Optimizers are set of procedures defined in SciPy
Ex:-   finding the min-value of a function
  getting root of mathematical equations
  etc..

=> Root of Equation::-
= In NumPy, we can get root for polynomials or linear equations
```

```
(but it does not have root for non linear equations)
Ex:-
 x + sin(x)

=** For this, we can use SciPy "optimze.root" function
Ex:-
 root(equation-function,x)

equation-function (represents an non-linear equations as return-value)
x -> initial guess for the root-value of equation

NOTE::-
= For SciPy Optimizers, we use "scipy.optimize" module

#Program (PythonSciPyEx3.py)
(Program to demo SciPy module with Optimizers)
#Program (PythonSciPyEx3.py)
#Program to demo SciPy module with Optimizers

from scipy.optimize import root;
from math import cos;

def equation(x):
 return x + cos(x);

rootresult = root(equation,0);
print(rootresult.x);
print(rootresult)


============================================================
==>> MatPlotLib in Python:- (Mathematical-Graphs)

=> What is Matplotlib?
= Matplotlib is graphs-plotting library in python
= It provides visualization of data
= Matplotlib was created by John D. Hunter
(open-source lib)
= It is mojorly written in python and minorly written in C, Objective-C and Javascript

=> Matplotlib Codebase?
= Source-Code for Matplotlib is in github repository https://github.com/matplotlib/matplotlib


==> Installation of Matplotlib:-
= Make sure Python and PIP are already installed in your system
= Next, install Matplotlib

[CMD> pip install matplotlib]

NOTE:-
= Alternatively, use python-distributions like Anaconda, Spyder etc, that already has Matplotlib in-built


==> Working with MatPlotLib in Programs::-
= for this, we use import statement,
Ex:-
 import Matplotlib;

##Program (MatPlotLibEx1.py)
import matplotlib
print(matplotlib.__version__);



==> Pyplot sub-module::-
= Matplotlib utilities are given under the pyplot sub-module
= It is usaully imported as "plt" alias
Ex:-
 import matplotlib.pyplot as plt;

##Program (MatPlotLibEx2.py)
#Draw a line in a diagram from position (0,0) to position (5,200)

import matplotlib.pyplot as plt;
import numpy as np;

xpoints = np.array([0, 5]);
ypoints = np.array([0, 200]);

plt.plot(xpoints, ypoints);
plt.show();


=> Plotting x and y points on graph::-
= For this plot() function is used to draw points in a graph
= By default, the plot() function draws a line from point to point
Syntax::-
 plt.plot(xpoints, ypoints);
(xpoints is an array with x-axis points)
(ypoints is an array with y-axis points)
Ex:-
 For plottin a line from (1, 1) to (10, 10), pass two arrays [1, 10] and [1, 10]

##Program (MatPlotLibEx2.py) ==SAME-Prog==
#Draw a line in a diagram from position (1,1) to position (10,10)

import matplotlib.pyplot as plt;
import numpy as np;

xpoints = np.array([1,10]);
ypoints = np.array([1,10]);

plt.plot(xpoints, ypoints)
plt.show();


==> Plotting Without Line::-
= To plot only markers, we use string-notation as parameter 'o', it means 'rings'

##Program (MatPlotLibEx2.py) ==SAME-Prog==
#Draw two points in the diagram, one at position (1,1) and one in position (8,8)

import matplotlib.pyplot as plt;
import numpy as np;

xpoints = np.array([1,8]);
ypoints = np.array([1,8]);

plt.plot(xpoints, ypoints, 'o');
```

```
plt.show();


==> Plotting with Multiple Points::-
= We can plot as many points as required, but make sure we have same number of points in both the axis

##Program (MatPlotLibEx2.py) ==SAME-Prog==
#Draw a line in a diagram from position (1, 1) to (3,6) then to (5,1) and finally (8,8)

import matplotlib.pyplot as plt;
import numpy as np;

xpoints = np.array([1,3,5,8]);
ypoints = np.array([1,6,1,8]);

plt.plot(xpoints,ypoints);
plt.show();


==> Default X-Points::-
= If we do not specify the x-axis points, then it will take default values as 0,1,2,3,4,.... etc. depending on the length of the y-axis-points
(The x-points are [0, 1, 2, 3, 4, 5])

##Program (MatPlotLibEx2.py) ==SAME-Prog==
#Plotting without x-axis-points

import matplotlib.pyplot as plt;
import numpy as np;

ypoints = np.array([2, 5, 1, 8, 3, 4]);

plt.plot(ypoints);
plt.show();


(***)
==>>> Matplotlib Markers::-
= We can use the argument/para "marker='o'", to mark each point in a graph with a specified marker

##Program (MatPlotLibEx3.py)
#Mark each point with a circle:

import matplotlib.pyplot as plt
import numpy as np;

ypoints = np.array([2, 5, 1, 8, 3, 4]);

plt.plot(ypoints, marker = 'o');
plt.show();


NOTE:-
**= Some Pre-defined Marker References:-
Marker Description
--------------------
'o'  Circle
'*'  Star
'.'  Point
','  Pixel
'x'  X
'X'  X (filled)
'+'  Plus
'P'  Plus (filled)
's'  Square
'D'  Diamond
'd'  Diamond (thin)
'p'  Pentagon
'H'  Hexagon
'h'  Hexagon
'v'  Triangle Down
'^'  Triangle Up
'<'  Triangle Left
'>'  Triangle Right
'1'  Tri Down
'2'  Tri Up
'3'  Tri Left
'4'  Tri Right
'|'  Vline
'_'  Hline



==> Format Strings fmt:-
= We can also use the shortcut string notation parameter to specify the marker
= This parameter is also called fmt, it is written as follows,
"marker|line|color"

##Program (MatPlotLibEx3.py)
#Mark each point with a circle:(using fmt)

import matplotlib.pyplot as plt;
import numpy as np;

ypoints = np.array([2, 5, 1, 8, 3, 4]);

plt.plot(ypoints, 'o:r');
plt.show();


NOTE:-
= Some pre-defined Line References are,
Line-Syntax  Description
--------------------------------------
'-'    Solid-line
':'    Dotted-line
'--'   Dashed-line
'-.'   Dashed/dotted-line
--------------------------------------

= Some Color References are,
Color-Syntax Description
--------------------------------------
'r'    Red
'g'    Green
'b'    Blue
'c'    Cyan
'm'    Magenta
'y'    Yellow
'k'    Black
'w'    White
```

==> Marker Size::-
= Here we use the argument/para markersize(ms) to set the size of the markers

```
##Program (MatPlotLibEx3.py)
#Set the size of the markers to 15:

import matplotlib.pyplot as plt;
import numpy as np;

ypoints = np.array([2, 5, 1, 8, 3, 4]);

plt.plot(ypoints, marker = 'o', ms = 15);
plt.show();
```

==>Marker Color::-
= Here we can use the argument/para markeredgecolor(mec) to set the color for edge of the markers

```
##Program (MatPlotLibEx3.py)
#Set the EDGE color to red

import matplotlib.pyplot as plt;
import numpy as np;

ypoints = np.array([2, 5, 1, 8, 3, 4]);

plt.plot(ypoints, marker = 'o', ms = 15, mec = 'r');
plt.show();
```

NOTE:-
= We can also use the argument/para markerfacecolor(mfc) to set the color inside the edge of the markers

```
##Program (MatPlotLibEx3.py)
#Set the FACE color to red

import matplotlib.pyplot as plt;
import numpy as np;

ypoints = np.array([2, 5, 1, 8, 3, 4]);

plt.plot(ypoints, marker='o', ms=20, mfc='r');
plt.show();
```

NOTE:-
= We can use both mec and mfc arguments/para to color of the entire marker

```
##Program (MatPlotLibEx3.py)
#Set the color of both the edge and the face to red

import matplotlib.pyplot as plt;
import numpy as np;

ypoints = np.array([2, 5, 1, 8, 3, 4]);

plt.plot(ypoints, marker='o', ms=15, mec='r', mfc='r');
plt.show();
```

NOTE:-
= We can also use Hexadecimal-color-values or color-names,
Ex:-
plt.plot(ypoints, marker='o', ms=15, mec='#4CAF50', mfc='coral');
= Other hexa-values & color-names are,(140)
AliceBlue
#F0F8FF
AntiqueWhite
#FAEBD7
Aqua
#00FFFF
Aquamarine
#7FFFD4
Azure
#F0FFFF
Beige
#F5F5DC
Bisque
#FFE4C4
Black
#000000
BlanchedAlmond
#FFEBCD
Blue
#0000FF
BlueViolet
#8A2BE2
Brown
#A52A2A
BurlyWood
#DEB887
CadetBlue
#5F9EA0
Chartreuse
#7FFF00
Chocolate
#D2691E
Coral
#FF7F50
CornflowerBlue
#6495ED
Cornsilk
#FFF8DC
Crimson
#DC143C
Cyan
#00FFFF
DarkBlue
#00008B
DarkCyan
#008B8B
DarkGoldenRod
#B8860B
DarkGray
#A9A9A9
DarkGrey
#A9A9A9

DarkGreen
#006400
DarkKhaki
#BDB76B
DarkMagenta
#8B008B
DarkOliveGreen
#556B2F
DarkOrange
#FF8C00
DarkOrchid
#9932CC
DarkRed
#8B0000
DarkSalmon
#E9967A
DarkSeaGreen
#8FBC8F
DarkSlateBlue
#483D8B
DarkSlateGray
#2F4F4F
DarkSlateGrey
#2F4F4F
DarkTurquoise
#00CED1
DarkViolet
#9400D3
DeepPink
#FF1493
DeepSkyBlue
#00BFFF
DimGray
#696969
DimGrey
#696969
DodgerBlue
#1E90FF
FireBrick
#B22222
FloralWhite
#FFFAF0
ForestGreen
#228B22
Fuchsia
#FF00FF
Gainsboro
#DCDCDC
GhostWhite
#F8F8FF
Gold
#FFD700
GoldenRod
#DAA520
Gray
#808080
Grey
#808080
Green
#008000
GreenYellow
#ADFF2F
HoneyDew
#F0FFF0
HotPink
#FF69B4
IndianRed
#CD5C5C
Indigo
#4B0082
Ivory
#FFFFF0
Khaki
#F0E68C
Lavender
#E6E6FA
LavenderBlush
#FFF0F5
LawnGreen
#7CFC00
LemonChiffon
#FFFACD
LightBlue
#ADD8E6
LightCoral
#F08080
LightCyan
#E0FFFF
LightGoldenRodYellow
#FAFAD2
LightGray
#D3D3D3
LightGrey
#D3D3D3
LightGreen
#90EE90
LightPink
#FFB6C1
LightSalmon
#FFA07A
LightSeaGreen
#20B2AA
LightSkyBlue
#87CEFA
LightSlateGray
#778899
LightSlateGrey
#778899
LightSteelBlue
#B0C4DE
LightYellow
#FFFFE0
Lime
#00FF00
LimeGreen
#32CD32
Linen
#FAF0E6
Magenta
#FF00FF

Maroon
#800000
MediumAquaMarine
#66CDAA
MediumBlue
#0000CD
MediumOrchid
#BA55D3
MediumPurple
#9370DB
MediumSeaGreen
#3CB371
MediumSlateBlue
#7B68EE
MediumSpringGreen
#00FA9A
MediumTurquoise
#48D1CC
MediumVioletRed
#C71585
MidnightBlue
#191970
MintCream
#F5FFFA
MistyRose
#FFE4E1
Moccasin
#FFE4B5
NavajoWhite
#FFDEAD
Navy
#000080
OldLace
#FDF5E6
Olive
#808000
OliveDrab
#6B8E23
Orange
#FFA500
OrangeRed
#FF4500
Orchid
#DA70D6
PaleGoldenRod
#EEE8AA
PaleGreen
#98FB98
PaleTurquoise
#AFEEEE
PaleVioletRed
#DB7093
PapayaWhip
#FFEFD5
PeachPuff
#FFDAB9
Peru
#CD853F
Pink
#FFC0CB
Plum
#DDA0DD
PowderBlue
#B0E0E6
Purple
#800080
RebeccaPurple
#663399
Red
#FF0000
RosyBrown
#BC8F8F
RoyalBlue
#4169E1
SaddleBrown
#8B4513
Salmon
#FA8072
SandyBrown
#F4A460
SeaGreen
#2E8B57
SeaShell
#FFF5EE
Sienna
#A0522D
Silver
#C0C0C0
SkyBlue
#87CEEB
SlateBlue
#6A5ACD
SlateGray
#708090
SlateGrey
#708090
Snow
#FFFAFA
SpringGreen
#00FF7F
SteelBlue
#4682B4
Tan
#D2B48C
Teal
#008080
Thistle
#D8BFD8
Tomato
#FF6347
Turquoise
#40E0D0
Violet
#EE82EE
Wheat
#F5DEB3
White
#FFFFFF
WhiteSmoke
#F5F5F5

```
Yellow
#FFFF00
YellowGreen
#9ACD32



==>> Matplotlib Lines::-
==> Linestyle:-
= We can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line

##Program (MatPlotLibEx4.py)

#Use a dotted line
import matplotlib.pyplot as plt;
import numpy as np;

ypoints = np.array([2, 8, 3, 6, 1, 9]);

plt.plot(ypoints, linestyle='dotted');
plt.show();


NOTE:-
=> Using a dashed-line,
Ex:-
 plt.plot(ypoints, linestyle = 'dashed')
##Program (MatPlotLibEx4.py)

=> Short-way Syntax,
Syntax:-
 linestyle as ls
 dotted as :
 dashed as --
Ex:-
 plt.plot(ypoints, ls=':');
##Program (MatPlotLibEx4.py)

=> Other Line Styles,
Ex:-
'solid' (default) '-'
'dotted'   ':'
'dashed'   '--'
'dashdot'   '-.'
'None'    '' or ' '
##Program (MatPlotLibEx4.py)


=> Line Color:-
= We can use the argument color or short-way (c) to set the color of line
Ex:-

##Program (MatPlotLibEx4.py)
#Set the line color to red

import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([2, 8, 3, 6, 1, 9]);

plt.plot(ypoints, color = 'r')
plt.show()


NOTE:-
= We can also use hexadecimal-values
Ex:-
 plt.plot(ypoints, c='#4CAF50');


==> Line-width::-
= We can use argument linewidth or (lw) to change the width of the line
= Its value is floating number, in points

##Program (MatPlotLibEx4.py)
##Line Width
import matplotlib.pyplot as plt;
import numpy as np;

ypoints = np.array([2, 8, 3, 6, 1, 9]);

plt.plot(ypoints, linewidth = '5.4');
plt.show();


==> Multiple Lines Plotting::-
= We can plot as multiple-lines as per requirement by adding multiple plt.plot() functions in code
Ex:-
##Program (MatPlotLibEx4.py)
#Draw two lines by specifying a plt.plot() function for each line
import matplotlib.pyplot as plt;
import numpy as np;

y1 = np.array([2, 8, 3, 6, 1, 9]);
y2 = np.array([1, 6, 2, 5, 1, 8]);

plt.plot(y1, c='r');
plt.plot(y2, c='g');

plt.show();


NOTE:-
= Here we only specified the points on the y-axis, meaning that the points on the x-axis got the the default values (0, 1, 2, 3)
= We can use both x-axis-points and y-axis-points for multiple-line plotting

Ex:-
##Program (MatPlotLibEx4.py)
#plotting with both x-axis-points & y-axis-points
import matplotlib.pyplot as plt;
import numpy as np;

x1 = np.array([0, 1, 2, 3, 4, 5]);
y1 = np.array([2, 8, 3, 6, 1, 9]);

x1 = np.array([0, 1, 2, 3, 4, 5]);
y2 = np.array([1, 6, 2, 5, 1, 8]);

plt.plot(x1, y1, x2, y2);
plt.show();
```

==>> Matplotlib Labels and Title::-
= For this, we use the xlabel() and ylabel() functions to set a label for the x-axis and y-axis

```
##Program (MatPlotLibEx5.py)
#Adding labels for x-axis and y-axis

import numpy as np;
import matplotlib.pyplot as plt;

x = np.array([1001,1002,1003,1004,1005]);
y = np.array([23, 25, 22, 20, 21]);

plt.plot(x,y);

plt.xlabel("Roll-Numbers");
plt.ylabel("Student-Ages");

plt.show();
```

==> Create a Title for a Graph(Plot)::-
= For this, we use title() function to set a title for the plot/graph

```
##Program (MatPlotLibEx5.py)
#Add a graph/plot title and labels for x-axis and y-axis

import numpy as np;
import matplotlib.pyplot as plt;

x = np.array([1001,1002,1003,1004,1005]);
y = np.array([23, 25, 22, 20, 21]);

plt.plot(x,y);

plt.title("Student-Details/Report-Card");
plt.xlabel("Roll-Numbers");
plt.ylabel("Student-Ages");

plt.show();
```

==> Set Font Properties for Title and Labels::-
= For this, we use the fontdict="" parameter in xlabel(), ylabel(), and title() functions to set font properties for the title and labels

```
##Program (MatPlotLibEx5.py)
#Set font properties for the title and labels

import numpy as np
import matplotlib.pyplot as plt

x = np.array([1001,1002,1003,1004,1005]);
y = np.array([23, 25, 22, 20, 21]);

font1 = {'family':'Courier','color':'Red','size':23};
font2 = {'family':'Courier','color':'Maroon','size':17};

plt.plot(x,y);

plt.title("Student-Details/Report-Card",fontdict=font1);
plt.xlabel("Roll-Numbers",fontdict=font2);
plt.ylabel("Student-Ages",fontdict=font2);

plt.show();
```

==> Position the Title::- (left/right/center)
= Here we can use the loc parameter in title() function to position the title
= Accepted-values are: left/right/center
= Default value is "center"

```
##Program (MatPlotLibEx5.py)
#Position the title to the left:

import numpy as np
import matplotlib.pyplot as plt

x = np.array([1001,1002,1003,1004,1005]);
y = np.array([23, 25, 22, 20, 21]);

plt.title("Student-Details/Report-Card",loc="left");
plt.xlabel("Roll-Numbers");
plt.ylabel("Student-Ages");

plt.plot(x,y);
plt.show();
```

(***)
==>> MatPlotLib Grids:::-
= For this, we use grid() function to add grid lines to the plot/graph

```
##Program (MatPlotLibEx6.py)
#Add grid lines to the plot

import numpy as np;
import matplotlib.pyplot as plt;

x = np.array([1001,1002,1003,1004,1005]);
y = np.array([23, 25, 22, 20, 21]);

plt.title("Student-Details/Report-Card");
plt.xlabel("Roll-Numbers");
plt.ylabel("Student-Ages");

plt.plot(x, y)
plt.grid()
plt.show()
```

==> Specify Which Grid Lines to Display::-
= For this, we use the axis="" parameter in the grid() function to specify which grid lines to display
= Accepted-values are 'x','y' and 'both'
(Default value is 'both')

```
##Program (MatPlotLibEx6.py)
#Display only grid lines for the x-axis

import numpy as np;
import matplotlib.pyplot as plt;

x = np.array([1001,1002,1003,1004,1005]);
y = np.array([23, 25, 22, 20, 21]);

plt.title("Student-Details/Report-Card");
plt.xlabel("Roll-Numbers");
plt.ylabel("Student-Ages");

plt.plot(x, y);
plt.grid(axis='x');
plt.grid(axis='y');
plt.show();
```

```
==> Setting Line-Properties for the Grid::-
You can also set the line properties of the grid, like this: grid(color = 'color', linestyle = 'linestyle', linewidth = number).

##Program (MatPlotLibEx6.py)
#Setting line-properties of the grid

import numpy as np;
import matplotlib.pyplot as plt;

x = np.array([1001,1002,1003,1004,1005]);
y = np.array([23, 25, 22, 20, 21]);

plt.title("Student-Details/Report-Card");
plt.xlabel("Roll-Numbers");
plt.ylabel("Student-Ages");

plt.plot(x, y);
plt.grid(color='orange', linestyle='--', linewidth=1.0);
plt.show();
```

```
(***)
==> Matplotlib Subplots::-
(Displaying Multiple Plots/Graphs)
= With the subplots() function you can draw multiple plots in one figure

##Program (MatPlotLibEx7.py)
#Drawing 2 plots at a time

import matplotlib.pyplot as plt;
import numpy as np;

#graph1
x = np.array([0, 1, 2, 3]);
y = np.array([0, 5, 2, 8]);

plt.subplot(1, 2, 1);
plt.plot(x,y);

#graph2
x = np.array([0, 1, 2, 3]);
y = np.array([0, 1, 2, 3]);

plt.subplot(1, 2, 2);
plt.plot(x,y);
plt.show();
```

```
**NOTE::-
=> subplots(arg1,arg2,arg3) Function
= It takes 3-args which describes layout of the figure
(layout is organized in rows and columns, which are represented by the 1st and 2nd argument)
(3rd argument represents the index of the current plot)


##Program (MatPlotLibEx7.py)
#Draw 2 plots on top of each other (2-rows & 1-col)

import matplotlib.pyplot as plt;
import numpy as np;

#graph1
x = np.array([0, 1, 2, 3]);
y = np.array([0, 5, 2, 8]);

plt.subplot(2, 1, 1);
plt.plot(x,y);

#graph2
x = np.array([0, 1, 2, 3]);
y = np.array([0, 1, 2, 3]);

plt.subplot(2, 1, 2);
plt.plot(x,y);
plt.show();
```

```
==> Multiple Plots ::-
= We can draw as many plots as required in one figure
= For this give no.of.rows, no.of.cols, and the index of the plot


##Program (MatPlotLibEx7.py)
#Draw 6 plots (2-rows & 3-cols)

import matplotlib.pyplot as plt;
import numpy as np;

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)
```

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)
plt.show()
```

==> Titles for Subplots::-
= You can add a title to each plot with the title() function

```
##Program (MatPlotLibEx7.py)
#2 sub-plots, with titles

import matplotlib.pyplot as plt;
import numpy as np;

#graph1
x = np.array([0, 1, 2, 3]);
y = np.array([0, 5, 2, 8]);

plt.subplot(1, 2, 1);
plt.plot(x,y);
plt.title("Students");

#graph2:
x = np.array([0, 1, 2, 3]);
y = np.array([0, 1, 2, 3]);

plt.subplot(1, 2, 2);
plt.plot(x,y);
plt.title("Depts");
plt.show();
```

==> Super Title::-
= We can add a title to the entire figure with the suptitle() function

```
##Program (MatPlotLibEx7.py)
#Add a title for the entire figure

import matplotlib.pyplot as plt;
import numpy as np;

#graph1
x = np.array([0, 1, 2, 3]);
y = np.array([0, 5, 2, 8]);

plt.subplot(1, 2, 1);
plt.plot(x,y);
plt.title("Students");

#graph2:
x = np.array([0, 1, 2, 3]);
y = np.array([0, 1, 2, 3]);

plt.subplot(1, 2, 2);
plt.plot(x,y);
plt.title("Depts");

plt.suptitle("College-REPORT")
plt.show()
```

(*****)
==> Matplotlib Scatter::-
=> Creating Scatter Plots/Graphs
= Using a Pyplot module and scatter() function, we can draw a scatter plot
= scatter() function plots one dot for each co-ordinate in graph
= It uses 2-arrays of same-size
(x-axis,y-axis)

```
##Program (MatPlotLibEx8.py)
#Simple scatter plot/graph

import matplotlib.pyplot as plt;
import numpy as np;

x = np.array([2,4,6,8,10,12,14,16,18,20,22,24])
y = np.array([95,80,85,75,70,45,100,25,10,50,35,20])

plt.scatter(x,y);
plt.show();
```

=** it takes corresponding values from each array as co-ordinate

==> Comparing Plots::-
= For comparing multiple-plots, we use multiple (x,y) co-ordinates from multiple-arrays

```
##Program (MatPlotLibEx8.py)
##Draw two plots in the same graph
import matplotlib.pyplot as plt;
import numpy as np;

x = np.array([2,4,6,8,10,12,14,16,18,20,22,24]);
y = np.array([95,80,85,75,70,45,100,25,10,50,35,20]);
```

```
plt.scatter(x, y);

x = np.array([2,4,6,8,10,12,14,16,18,20,22,24]);
y = np.array([90,85,80,70,75,40,95,30,15,55,30,25]);

plt.scatter(x, y);
plt.show();


NOTE:-
=> Colors::-
= We can set colors for scatter plots using "color" (or) "c" argument
Ex:- ==Same-Program==
plt.scatter(x, y, color="#88c999");
plt.scatter(x, y, color="green");

=> Colors for each dot::-
= We can have specific color for each dot by using an array of colors as value for the c argument
(We cannot use the color argument for this, only the c argument)
Ex:- ==Same-Program==
colors = np.array(["red","green","blue","yellow","pink","black","orange","purple","beige","brown","gray","cyan"])
plt.scatter(x, y, c=colors);


=> Sizes::-
= We can change the size of dots using s="" argument
= Same like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis co-ordinatess
Ex:- ==SAME-Prog==
sizes = np.array([10,20,30,40,50,60,70,80,90,100,110,120]);
plt.scatter(x, y, s=sizes);


=> Alpha::-
= We can provide transparency of dots using alpha="" argument
Ex:- ==SAME-Prog==
plt.scatter(x, y, s=sizes, alpha=0.5);  #0.0 to 1.0

=** Finally, we can combinew s="", c="", alpha=0.5 all at a time



(*****)
==>>> Matplotlib Bars::-
=> Creating Bars
= For this, we use bar() function to drawing bar-graphs

##Program (MatPlotLibEx9.py)
#Draw 4 bars using MatPlotLib

import matplotlib.pyplot as plt;
import numpy as np;

x = np.array(["A","B","C","D"]);
y = np.array([5,9,2,6]);

plt.bar(x,y);
plt.show();


**NOTE:-
= For bar() function, we can pass arguments
= Arguments are given as arrays
Ex::-
plt.bar(x, y)


=> Horizontal Bars::-
= For horizontal bars, use the barh() function

##Program (MatPlotLibEx9.py)
#Draw 4 horizontal bars
import matplotlib.pyplot as plt;
import numpy as np;

x = np.array(["A","B","C","D"]);
y = np.array([5,9,2,6]);

plt.barh(x, y);
plt.show();


NOTE:-
=> Bar Colors:-
= bar() and barh() we can give keyword-argument color="", to set the color of the bars
Ex:- ==SAME-Prog==
 plt.bar(x, y, color = "green");  #color-name
 plt.bar(x, y, color = "#4CAF50"); #hexadecimal-value

=> Bar Width:-
= For bar() we can give keyword-argument width="", to set the width of the bars
Ex:- ==SAME-Prog==
 plt.bar(x, y, width=0.1);
 plt.bar(x, y, width=0.5);
(default width is 0.8)

=> Bar Height:-
= For barh() we can give keyword-argument height="", to set the height of the bars
Ex:- ==SAME-Prog==
 plt.barh(x,y,height=0.1);
 plt.barh(x,y,height=0.5);
(default height is 0.8)



(*****)
==>>> Matplotlib with Histograms::-
(Histogram)
= A histogram is a graph showing frequency distributions
= It is a graph with no. of observations with-in given interval-time
Ex::-
=> Create Histogram?
= For this, we use hist() function to create histograms
= hist() function takes an array of numbers as input-para(args)


##Program (MatPlotLibEx10.py)
import matplotlib.pyplot as plt;
import numpy as np;

x = np.random.normal(100,5,200);
```

```
plt.hist(x);
plt.show();


NOTE:-
= Array with 200-values, where the values will concentrate around 100, and the standard deviation is 5


(*****)
==>> Matplotlib with Pie-Charts::-
=> Creating Pie Charts,
= Using Pyplot module, we can use pie() function to draw pie charts

#A simple pie chart
import matplotlib.pyplot as plt;
import numpy as np;

y = np.array([30,20,35,15]);

plt.pie(y);
plt.show();


NOTE:-
**= Here, pie-chart draws one piece (called a wedge) for each value in the array
Ex:- [30,20,35,15]
(by default, the plotting of the first wedge starts from the x-axis and moves anti-clockwise)

**= The size of each wedge is determined by comparing the value with all the other values, by using this formula: x/sum(x)


=> Labels:-
= We can add labels to the pie chart with the label="" parameter
= label="" parameter is an array with one label for each wedge
Ex:- ==SAME-Prog==
mylabels = ["A","B","C","D"]
plt.pie(y, labels = mylabels)


=> Start-Angle::-
= Default start angle is at the x-axis(0-degrees)
= However, we can change the start angle by specifying a startangle="" parameter
Ex:- (==SAME-Prog==)
 plt.pie(y, labels = mylabels,startangle=45);


=> Explode::-
= For any-one wedge to stand-out(separated from pie-chart as single-piece), we use explode="" parameter
= The explode="" parameter, is an array with one value for each wedge
(Each value represents how far from the center each wedge is displayed)
Ex:-
myexplode = [0.2, 0, 0.3, 0];
plt.pie(y, labels= mylabels, explode=myexplode);


=> Shadow::-
= We can shadow to the pie-chart, by giving shadows="" parameter to True
Ex:-
plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)


=> Colors::-
= We can set the color of each wedge with the colors="" parameter
= colors="" parameter, is an array with one value for each wedge
Ex:-
mycolors = ["blue","yellow","#4CAF50","b"];
plt.pie(y, labels=mylabels, colors=mycolors);

NOTE:-
= Hexadecimal-color-values (or) color-names, (or) color-name-shortcuts
Ex:-
'r' - Red
'g' - Green
'b' - Blue
'c' - Cyan
'm' - Magenta
'y' - Yellow
'k' - Black
'w' - White



=> Legend::-
= We can add a list of explanation for each wedge, use the legend() function
Ex:-
mylabels = ["A","B","C","D"]
plt.pie(y, labels = mylabels)
plt.legend()


=> Legend With Header::-
= For adding header to the legend, add the title="" parameter to the legend function
Ex:-
plt.legend(title="Four-Parts");




***Next-class(DJANGO)***
----------------------------
14-SEP-2022(WED) @9am
```