=========================================================

Number of approaches for development of Programs in Python

=========================================================

----------------------------------------

=>Definition of Program:

----------------------------------------

=>Set of Optimized Instructions is called Program.

=>Programs are always developed by Language Programers for solving Real Time Applications.

=>To solve any Real Time Application, we must write Set of Optimized Instructions and save

those Instructions on Some File Name with an extension .py

(FileName.py------>Considered as Python Program)

-----------------------------------------------------------------------

=>In Python Programming, we can develop any Program with Approaches. They are

1. Interactive Mode Approach

2. Batch Mode Approach

----------------------------------------------------

1. Interactive Mode Approach

----------------------------------------------------

=>In This approach, the programmer can issue one Instruction at a time and gets One Output at a time

=>This Approach is more useful to test one Instruction at a time.

=>This Approach is not useful for Developing Code for Big Problems and more over we are

unable to save the instrcutions.

Examples:

------------------

>>> a=10

>>> b=20

>>> c=a+b

>>> print(a)-----------10

>>> print(b)-----------20

>>> print(c)-----------30

--------------------

Softwares----- Python Command Command (Will come on the Installation Python)

Python IDLE Shell (Will come on the Installation Python )

-----------------------------------------------------------------------

2. Batch Mode Approach

-----------------------------------------------------------------------

=>The Process of definining Group of Instructions under one editor and save those instructions on some file name with an extension .py (FileName.py--Source Code) is called Batch Mode Approach

=>This Approach is more useful for solving Big Problems

Software:Python IDLE Shell (Will come on the Installation Python )

Edit Plus (Install Explicitly)

Pycharm

Jupiter NoteBook

Spider

Visual Studio (VS Code)

Google Clab

Atom

sub lime Text....etc

------------------------------------------------------------------------

#Program for mul of two numbers

#MulEx.py---File Name

a=float(input("Enter Value a:"))

b=float(input("Enter Value b:"))

c=a\*b

print("----------------------")

print("Val of a=",a)

print("Val of b=",b)

print("Mul=",c)

print("----------------------")

--------------------------------------------------------------------------------------------------------------------------

=>To run the Python Program from Windows Command Prompt , we use "python" or "py"

=>Syntax:

E:\KVR-PYTHON-11am\Batch-Mode>python MulEx.py

(OR)

E:\KVR-PYTHON-11am\Batch-Mode>py MulEx.py

==================================x=======================================

#Program for mul of two numbers

a=float(input("Enter Value a:"))

b=float(input("Enter Value b:"))

c=a\*b

print("----------------------")

print("Val of a=",a)

print("Val of b=",b)

print("Mul=",c)

print("----------------------")

#Program for computing sum of two numbers

a=10

b=20

c=a+b

print("value of a=",a)

print("Value of b=",b)

print("sum=",c)

#program for computing sum of two numbers

a=float(input("Enter First value:"))

b=float(input("Enter Second value:"))

c=a+b

print("========Result========")

print("val of a=",a)

print("val of b=",b)

print("Sum=",c)

print("=====================")

=====================================================

Display the Result of Python Program on the console

=====================================================

=>To display the result of Python Program on the console, we use a pre-defined Function called print ().

=>print () is one of the pre-defined Function used for displaying the result of Python Program on the console

=>print () contains the following Syntaxes

-----------------------------------------------------------------------Syntax-1:

-----------------------------------------------------------------------

=>Syntax print(value)

(OR)

print(Variable Name)

(OR)

print(Val1,val2....val-n)

(OR)

print(var1,var2.....var-n)

=>This Syntax used for Displaying only Values or Values of variables.

-----------------------------------------------------------------------

Examples:

-----------------------------------------------------------------------

>>> sno=10

>>> sname="Rossum"

>>> sub="Python"

>>> print(sno)----------------10

>>> print(sname)-----------Rossum

>>> print(sub)--------------Python

>>> print(sno,sname,sub)--------10 Rossum Python

>>> print(100,200,300)-------------100 200 300

-----------------------------------------------------------------------

Syntax-2

-----------------------------------------------------------------------

Syntax: print(Message)

(OR)

print(Message1,Message2,....,Message-n)

=>This Syntax display only Messges.

----------------

Examples:

----------------

>>> print("hello Python world")--------------hello Python world

>>> print('hello Python world')---------------hello Python world

-----------------------------------------------------------------------

Syntax-3

-----------------------------------------------------------------------

Stntax: print(Values cum Messages)

(OR)

print(Messages cum Values)

=>This Syntax displayes the values cum messages or Messages cum Values

--------------------------

Examples:

--------------------------

>>> a=10

>>> print(a)-----------10

>>> print("Value of a=",a)--------Value of a= 10

>>> print(a,' is the value of a')-------10 is the value of a

>>> a=10

>>> b=20

>>> c=a+b

>>> print("sum=",c)------------------sum= 30

>>> print(c," is the sum")----------30 is the sum

>>> print("sum of ",a," and ",b,"=",c)--------sum of 10 and 20 = 30

>>> a=10

>>> b=20

>>> c=30

>>> d=a+b+c

>>> print("Sum of ",a,",",b," and ",c,"=",d)----Sum of 10 , 20 and 30 = 60

-----------------------------------------------------------------------

Syntax-4

----------------------------------------------------------------------

Syntax: print(Values cum Messages with format() )

(OR)

print(Messages cum Values with format() )

Examples:

------------------

>>> a=10

>>> b=20

>>> c=a+b

>>> print("Sum={}".format(c))-------------Sum=30

>>> print("{} is the sum".format(c))------30 is the sum

>>> print("sum of ",a," and ",b,"=",c)-----sum of 10 and 20 = 30

>>> print("sum of {} and {}={}".format(a,b,c))----sum of 10 and 20=30

>>> sno=10

>>> sname="Rossum"

>>> print(" '{}' is a student and roll number is {}".format(sname,sno))--- 'Rossum' is a student and roll number is 10

-----------------------------------------------------------------

Syntax-5

-----------------------------------------------------------------------

Stntax: print(Values cum Messages with format specifiers)

(OR)

print(Messages cum Values with format specifiers )

-----------------------

Examples:

-----------------------

>>> a=10

>>> b=20

>>> c=a+b

>>> print ("Sum=%d" %c)---------Sum=30

>>> print("%d is the sum" %c)-------30 is the sum

>>> print("Sum of %d and %d = %d" %(a,b,c))------Sum of 10 and 20 = 30

>>> sno=10

>>> sname="Elite Elderson"

>>> marks=33.33

>>> print("My Number is %d and name is '%s' and Marks=%f" %(sno,sname,marks))

My Number is 10 and name is 'Elite Elderson' and Marks=33.330000

>>> print("My Number is %d and name is '%s' and Marks=%0.2f" %(sno,sname,marks))

My Number is 10 and name is 'Elite Elderson' and Marks=33.33

>>> print ("My Number is %d and name is '%s' and Marks=%0.1f" %(sno,sname,marks))

My Number is 10 and name is 'Elite Elderson' and Marks=33.3

-----------------

>>> a=1.2

>>> b=20

>>> c=a+b

>>> print ("sum of %f and %f=%f".format(a,b,c))----sum of %f and %f=%f

>>> print("sum of %f and %f=%f" %(a,b,c) )----sum of 1.200000 and 20.000000=21.200000

>>> print("sum of %0.2f and %0.2f=%0.3f" %(a,b,c) )--sum of 1.20 and 20.00=21.200

>>> t=(10,"Mr.Crazy",33.33,"Sathish")

>>> print(t)------------(10, 'Mr.Crazy', 33.33, 'Sathish')

>>> print("content of t=",t)--------content of t= (10, 'Mr.Crazy', 33.33, 'Sathish')

>>> print("content of t={}".format(t))----content of t=(10, 'Mr.Crazy', 33.33, 'Sathish')

>>> print("content of t=%s" %str(t))---content of t=(10, 'Mr.Crazy', 33.33, 'Sathish')

-----------------------------------------------------------------------

Syntax-6:

-----------------------------------------------------------------------

Syntax: print(Value cum Message, end=" ")

=>This syntax displays the data in same Line

Examples:

--------------------

>>> lst=[10,20,30,40,50,60]

>>> for val in lst:

... print(val)

...

10

20

30

40

50

60

>>> for val in lst:

... print(val,end=" ")-------- 10 20 30 40 50 60

>>> for val in lst:

... print(val,end="-->")----- 10-->20-->30-->40-->50-->60-->

-------------------------------------------

>>> lst=[10,20,30,40,50,60]

>>> for val in lst:

... print("{}".format(val), end="\t") 10 20 30 40 50 60 >>>

----------------------

>>> lst=[10,20,30,40,50,60]

>>> for val in lst:

... print("%d" %val, end=" ")-----10 20 30 40 50 60

-----------------------------------------------------------------------

==============================================

Reading the data or input from Key Board

==============================================

=>To read the data from Keyboard, we use Two pre-defined Functions. They are

1. input()

2. input(Message)

-----------------------------------------------------------------------

1) input ()

-----------------------------------------------------------------------

=>This Function is used for Reading any type of data from Key board in the form of str type only.

=>Syntax:- varname=input()

=>Here input() reads the value in the form str and place that value in varname.

=>The value of str can type casted to any other types by using Type Casting functions.

Examples

----------------------

#Program for accepting two integer values and multipy them

#MulExample3.py

print("Enter two Values:")

a=float( input() )

b=float( input() )

#Multiply them

c=a\*b

print("Mul({},{})={}".format(a,b,c))

---------------------------------------------------------------------------------------------------------------------

2) input(Message)

---------------------------------------------------------------------------------------------------------------------

=>This Function is used for Reading any type of data from Key board in the form of str type only and with Function additionally we can provide User-Prompting Message.

=>Syntax: varname=input(Message)

=>here Message Represents User-Prompting Message.

=>Here input (Message) reads the value in the form str and place that value in varname by giving User-Prompting Message.

=>The value of str can type casted to any other types by using Type Casting functions.

Examples:

--------------------

#MulExample6.py

a=float(input("Enter First value:"))

b=float(input("Enter Second value:"))

c=a\*b

print("Mul({},{})={}".format(a,b,c))

-----------------------------------------------------------------------

#Program for accepting two integer values and multipy them

#MulExample1.py

print("Enter First Value:")

s1=input()

print("Enter Second Value:")

s2=input()

#convert s1 and s2 into float

a=float(s1)

b=float(s2)

#Multiply them

c=a\*b

print("Mul({},{})={}".format(a,b,c))

#Program for accepting two integer values and multipy them

#MulExample2.py

print("Enter two Values:")

s1=input()

s2=input()

#convert s1 and s2 into float

a=float(s1)

b=float(s2)

#Multiply them

c=a\*b

print("Mul({},{})={}".format(a,b,c))

#Program for accepting two integer values and multipy them

#MulExample3.py

print ("Enter two Values:")

a=float(input() )

b=float(input() )

#Multiply them

c=a\*b

print("Mul({},{})={}".format(a,b,c))

#Program for accepting two integer values and multipy them

#MulExample4.py

print("Enter two Values:")

c=float( input() )\*float( input() )

print("Mul={}".format(c))

#Program for accepting two integer values and multipy them

#MulExample5.py

print("Enter two Values:")

a=float( input() )

b=float( input() )

print("Mul({},{})={}".format(a,b, a\*b))

#Program for accepting two integer values and multipy them

#MulExample6.py

s1=input("Enter First value:")

s2=input("Enter Second value:")

a=float(s1)

b=float(s2)

c=a\*b

print("Mul({},{})={}".format(a,b,c))

#Program for accepting two integer values and multipy them

#MulExample6.py

a=float(input("Enter First value:"))

b=float(input("Enter Second value:"))

c=a\*b

print("Mul({}, {})={}".format(a,b,c))

=========================================

Operators and Expressions in python

=========================================

=>An Operator is a symbol which is used to perform certain operations.

=>If any operator connected with two or more Objects / Variables then is it called Expression.

=>An Expression is a collection of objects or variables connected with Operators.

=>In python Programming, we have 7 types of Operators. They are

1. Arithmetic Operators

2. Assignment Operator

3. Relational Operators

4. Logical Operators

5. Bitwise Operators (Most Imp )

6. Membership Operators

a) in

b) not in

7. Identity Operators

a) is

b) is not

==========================================================

1. Arithmetic Operators

=======================================

=>The purpose of Arithmetic Operators is that "To Perform Arithmetic Operations such as addition, substraction...etc"

=>If Two or More Objects or Variables connected with Arithmetic Operators then it is called Arithmetic Expressions.

=>In Python programming, we have 7 types of Arithmetic Operators. They are given in the

following Table.

=======================================================================

SLNO SYMBOL MEANING EXAMPLES a=10 b=3

=======================================================================

1. + Addition print(a+b)------13

2. - Subtract print(a-b)--------7

3. \* Multiplication print(a\*b)-------30

4. / Division print(a/b)------3.3333333333335

(Float Quotient)

5. // Floor Division print(a//b)----3

(Integer Quotient)

6. % Modulo Division print(a%b)----1

(Reminder after Division)

7. \*\* Exponentiation print(a\*\*b)---100

=======================================================================

#Program for demonstrating Arithmetic Operators

#Aop.py

a=int(input("Enter Value of a:"))

b=int(input("Enter Value of b:"))

print("-"\*50)

print("Arithemtic Operators")

print("\*"\*50)

print("\tsum({},{})={}".format(a,b,a+b))

print("\n\tsub({},{})={}".format(a,b,a-b))

print("\n\tmul({},{})={}".format(a,b,a\*b))

print("\n\tDiv({},{})={}".format(a,b,a/b))

print("\n\tFloor Div({},{})={}".format(a,b,a//b))

print("\n\tMod({},{})={}".format(a,b,a%b))

print("\n\tExp({},{})={}".format(a,b,a\*\*b))

print("#"\*50)

#Program for cal square root of a given number

#sqrtex.py

n=float(input("Enter a number for can square root:"))

res=n\*\*(1/2)

print("square root({})={}".format(n,res))

==================================================

2. Assigment Operator ==================================================

=>The purpose of assignment operator is that " To assign or transfer Right Hand Side (RHS) Value / Expression Value to the Left Hand Side (LHS) Variable "

=>The Symbol for Assigment Operator is single equal to ( = ).

=>In Python Programming,we can use Assigment Operator in two ways.

1. Single Line Assigment

2. Multi Line Assigment

1. Single Line Assigment:

----------------------------------------

=>Syntax: LHS Varname= RHS Value

LHS Varname= RHS Expression

=>With Single Line Assigment at a time we can assign one RHS Value / Expression to the single LHS Variable Name.

------------------

Examples:

------------------

>>> a=10

>>> b=20

>>> c=a+b

>>> print(a,b,c)------------10 20 30

-----------------------------------------------------------------------------------

2. Multi Line Assigment:

----------------------------------------

=>Syntax: Var1,Var2.....Var-n= Val1,Val2....Val-n

Var1,Var2.....Var-n= Expr1,Expr2...Expr-n

Here The values of Val1, Val2...Val-n are assigned to Var1,Var2...Var-n Respectively.

Here The values of Expr1, Expr2...Expr-n are assigned to Var1,Var2...Var-n Respectively.

Examples:

-------------------

>>> a,b=10,20

>>> print(a,b)------------10 20

>>> c,d,e=a+b,a-b,a\*b

>>> print(c,d,e)-------------30 -10 200

======================================

Relational Operators

=====================================

=>The purpose of Relational Operators is that "To Compare Two or More Values "

=>If two or more Variables or Objects connected with Relational Operator then it is Relational Expression.

=>The Result of Relational Expression is either True or False.

=>The Relational Expression is called Test Condition

=>In Python Program, The Relational Operators are classified into 6 types. They are given in the following table

======================================================================

SLNO SYMBOL MEANING EXAMPLE

======================================================================

1. > Greater than print(10>20)------False

print(10>2)--------True

2. < Less Than print(10<20)------True

print(10<5)--------False

3. == Equality print(10==10)----True

(Double Equal to) print(10==5)------False

4. != Not Equal to print(10!=20)----True

print(10!=10)----False

5. >= Greater Than print(10>=5)-----True

or Equal to print(10>=11)----False

6. <= Less Than print(-30<=-34)----False

or Equal to print(-15>=-16)---True

======================================================================

#Program for performing all type of arithmetic Operations by using Multi line assigment

#MultiLineaop.py

a,b=int(input("Enter Value of a:")) , int(input("Enter Value of b:")) # Multi line assigment-reading

sum,sub,mul,div,fdiv,mod,exp=a+b,a-b,a\*b,a/b,a//b,a%b,a\*\*b # Multi line assigment--cal

print("="\*50)

print("Sum=",sum)

print("Sub=",sub)

print("Mul=",mul)

print("Div=",div)

print("Floor Div=",fdiv)

print("Mod=",mod)

print("Exp({},{})={}".format(a,b,exp))

print("="\*50)

#program for demonstaring Relational Operators

#relOprEx.py

a,b=float(input("Enter Value of a:")),float(input("Enter Value of b:"))

print("="\*40)

print("\tResults of Relational Operators")

print("="\*40)

print("\t{} > {}={}".format(a,b,a>b))

print("\n\t{} < {}={}".format(a,b,a<b))

print("\n\t{} == {}={}".format(a,b,a==b))

print("\n\t{} != {}={}".format(a,b,a!=b))

print("\n\t{} >= {}={}".format(a,b,a>=b))

print("\n\t{} <= {}={}".format(a,b,a<=b))

print("="\*40)

#Program for accepting any two values and swap them

#SwapValues.py

a,b=input("Enter Value of a:"),input("Enter Value of b:")

print("-"\*50)

print("\nOriginal value of a={}".format(a))

print("Original value of b={}".format(b))

print("-"\*50)

#swapping logic

a,b=b,a # Multi Line assigment

print("\nSwapped value of a={}".format(a))

=======================================

Bitwise Operators (Most Imp)

=======================================

=>Bitwise Operators are Performing the Operation on Integer Data in the form Binary Bits.

=>Bitwise Operators are applicable on Integer Data but not floating-point values bcoz floating values does not have certainity.

=>In Bitwise Operators, First Given Integer Data Converted into Binary data and they starts performing operation Bit by Bit and hence they named Bitwise Operators.

=>In Python Programming, we have 6 types of Bitwise Operators. They are

1. Bitwise Left Shift Operator (<<)

2. Bitwise Right Shift Operator (>>)

3. Bitwise AND Operator ( & )

4. Biriwse OR Operator ( | )

5. Bitwise Complement Operator (~ )

6. Bitwise XOR Operator ( ^ )

-----------------------------------------------------------------------1. Bitwise Left Shift Operator ( << ):

-----------------------------------------------------------------------

Syntax:- varname = Given Number << No. of Bits

=>This Operator Shits or Fipping-off No. of Bits of Given Number from Left Side and add Number of Zeros (depends on No. of Bits) at Right Side.

Examples:

---------------------

>>> print(10<<3)---------------80

>>> print(4<<4)-----------------64

>>> print(8<<3)-----------------64

>>> print(2<<3)----------------16

>>> print(5<<2)------------------20

----------------------------------------------------------------------------------------------------------------------------

2. Bitwise Right Shift Operator ( >> ):

----------------------------------------------------------------------------------------------------------------------------

Syntax:- varname = Given Number >> No. of Bits

=>This Operator Shits or Fipping-off No. of Bits of Given Number from Right Side and add Number of Zeros (depends on No. of Bits) at Left Side.

Examples:

---------------------

>>> print(10>>3)-------------1

>>> print(10>>2)-------------2

>>> print(12>>2)-------------3

>>> print(100>>4)------------6

----------------------------------------------------------------------------------------------------------------------------

3. Bitwise AND Operator ( & )

----------------------------------------------------------------------------------------------------------------------------

=>Syntax:- Varname = Var1 & Var2

=>The Functionality of Bitwise AND Operator (& ) is expressed in the following Truth table.

-----------------------------------------------------------------------

Var1 Var2 Var1 & Var2

-----------------------------------------------------------------------

0 1 0

1 0 0

0 0 0

1 1 1

-----------------------------------------------------------------------

Examples:

-----------------

>>>a=10-------------------> 0000 0000 0000 1010

>>>b=4--------------------->0000 0000 0000 0100

-----------------------------------------------------------------------

>>>c=a&b----------------->0000 0000 0000 0000

>>>print(c)-----0

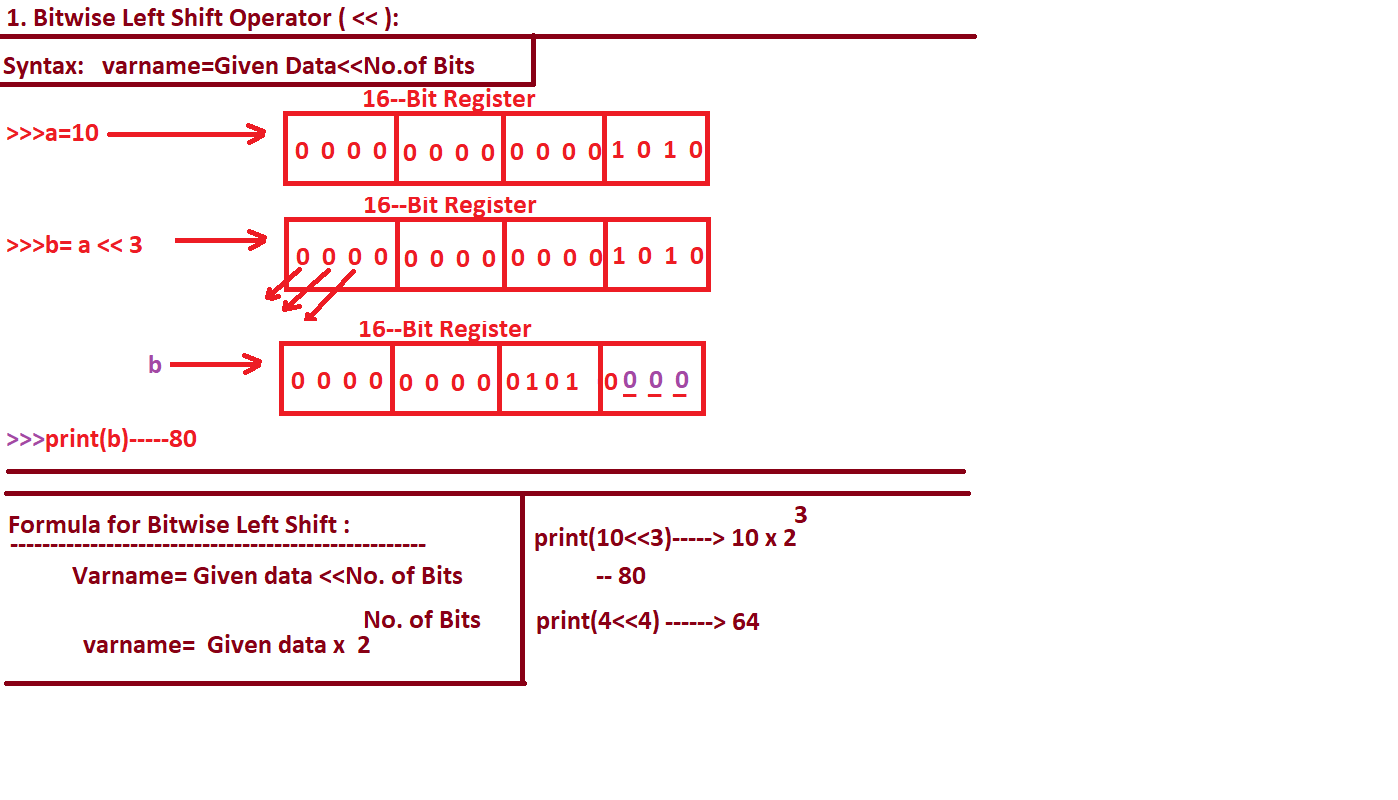
>>> print(7&4)--------------4

>>> print(6&10)-----------2

>>> 10 & 20------------0

>>> 10 and 20-----------20

print("-"\*50)



4. Bitwise OR Operator ( | )

-----------------------------------------------------------------------

=>Syntax: - Varname = Var1 | Var2

=>The Functionality of Bitwise OR Operator (| ) is expressed in the following Truth table.

-----------------------------------------------------------------------

Var1 Var2 Var1 | Var2

-----------------------------------------------------------------------

0 1 1

1 0 1

0 0 0

1 1 1

-----------------------------------------------------------------------

Examples:

----------------------

>>>a=4--------------------0100

>>>b=3--------------------0011

------------------------------------------

>>>c=a|b-----------------0111

>>>print(c)----------7

>>> print(10|15)----------15

>>> print(7|3)----------7

>>> print(2|5)-----------7

-----------------------------------------------------------------------

5. Bitwise Complement Operator ( ~ )

-----------------------------------------------------------------------

=>Bitwise Complement Operator ( ~ ) is used obtaining complement of a Given Number.

=>complement of a Given Number= - (Given Number+1 )

=>Internally, Bitwise Complement Operator invert the bits (Nothing But 1 becomes 0 and 0 becomes 1--- called 1 's complement)

Examples:

------------------

>>> a=17

>>> ~a---------------------------18

>>> a=-98

>>> ~a--------------------------- 97

>>> n=200

>>> ~n------------------------------ -201

>>> n=304

>>> ~n------------------------------ -305

--------------------------------------------------

Working Examples:

--------------------------------------------------

4---------------------------> 0100

~4----------------------------> -(0100+1)

0100

0001

--------

-0101

10-------------------------------->1010

~10-------------------------------> -(1010+1)

1010

0001

-------

-1011

---------

------------------------------------------------------------------------------------------------------

OR

10--------------------> 1010

~10----------------------------- 0101 (Inverting the bits)

------------------------------------------------------------------

11-----------------------------> 1011

1's complement------------ 0100 (Inverting the bits)

2's complement----1's complement of 11 + 1

0100

0001

--------

0101

====================================================================

4--------------------0100

~4------------------1011

---------------------------------------

What is -5 ( 2's complement 5=1 's complement of 5+1)

5--------------------------- 0101

1's complement-------1010

2'2complement= 1's complement +1

1010 +1

1010

0001

--------

1011

-----------------------------------------------------------------------

6. Bitwise XOR Operator ( ^ )

----------------------------------------------------------------------

=>Syntax:- Varname = Var1 ^ Var2

=>The Functionality of Bitwise XOR Operator (^ ) is expressed in the following Truth table.

-----------------------------------------------------------------------

Var1 Var2 Var1 ^ Var2

-----------------------------------------------------------------------

0 1 1

1 0 1

0 0 0

1 1 0

-----------------------------------------------------------------------

Examples:

-----------------

>>> a=3

>>> b=4

>>> c=a^b

>>> print(c)------------7

>>> print (10^15) -----5

>>> print (4^6)--------2

-----------------------------------------------------------------------

Special Case:

-----------------------

>>>s1= {10,20,30,40}

>>>s2= {10,15,25}

>>>s3=s1.union(s2)

>>> print(s3) -------------{20, 40, 25, 10, 30, 15}

>>> s4=s1|s2 # Bitwise OR

>>> print(s4,type(s4))-----------{20, 40, 25, 10, 30, 15} <class 'set'>

-------------------------------------

>>>s1={10,20,30,40}

>>>s2={10,15,25}

>>> s3=s1.intersection(s2)

>>> print(s3) -----------{10}

>>> s4=s1&s2 # Bitwise AND

>>> print(s4,type(s4))-----------{10} <class 'set'>

---------------------------------------------

>>> s1={10,20,30,40}

>>> s2={10,15,25}

>>> s3=s1.difference(s2)

>>> print(s3)-----------{40, 20, 30}

>>> s4=s1-s2

>>> print(s4)-----------{40, 20, 30}

>>> s5=s2-s1

>>> print(s5)-----------{25, 15}

---------------------------------------------------------------

>>> s1={10,20,30,40}

>>> s2={10,15,25}

>>> s3=s1.symmetric\_difference(s2)

>>> print(s3)--------------{40, 15, 20, 25, 30}

>>> s4=s1^s2 # Bitwise XOR

>>> print(s4,type(s4))---------{40, 15, 20, 25, 30} <class 'set'>

-----------------------------------------------------------------------Special Examples

----------------------------------------------------------------------->>>a=3

>>>b=4

>>>print(a,b)---- 3 4

>>>a=a^b

>>>b=a^b

>>>a=a^b

>>>print(a,b)----- 4 3

Membership Operators

===========================================

=>The purpose of Membership Operators is that "To Check the existence of specific value in Iterable object".

=>An Iterable Object is one which contains Two or More Number of values

=>Sequece Types (str,bytes,bytearray,range) , List (list, tuple) Types , set (set , frozenset) Types , and dict type(dict) are comes under Iterable object.

=>In Python Programming, we have two type of Membership Operators. They are

1) in

2) not in

-----------------------------------------------------------------------

1) in

-----------------------------------------------------------------------

Syntax: Value in Iterable Object

------------

=>"in" operator returns True provided "Value" present in Iterable Object

=>"in" operator returns False provided "Value" present not in Iterable Object

----------------------------------------------------------------------------------------------------------------

2) not in

----------------------------------------------------------------------------------------------------------------

Syntax: Value not in Iterable Object

------------

=>"not in" operator returns True provided "Value" not present in Iterable Object

=>"not in" operator returns False provided "Value" present in Iterable Object

----------------------------------------------------------------------------------------------------------------

Examples:

----------------------------------------------------------------------------------------------------------------

>>> s="PYTHON"

>>> print(s)

PYTHON

>>>

>>>

>>> s="PYTHON"

>>> "P" in s

True

>>> "O" in s

True

>>> "O" not in s

False

>>> "k" not in s

True

>>> "k" in s

False

>>> "p" in s

False

>>> "p" not in s

True

>>> not ("p" not in s)

False

>>>

>>>

>>>

>>>

>>> s="PYTHON"

>>> print(s)

PYTHON

>>> "PYT" in s

True

>>> "PYTK" in s

False

>>> "PYTK" not in s

True

>>>

>>>

>>> s="PYTHON"

>>> print(s)

PYTHON

>>> "PON" in s

False

>>> "PYN" in s

False

>>> "PYN" not in s

True

>>>

>>>

>>> s="PYTHON"

>>> print(s)

PYTHON

>>> "NOH" in s

False

>>> "HON" not in s

False

>>> "NOH" in s

False

>>> "OTP" in s

False

>>> "OTP" not in s

True

>>>

>>>

>>> s="PYTHON"

>>> print(s)

PYTHON

>>> "NOH" in s[::-1]

True

>>> "OTP" not in s[::-2]

True

>>> s[::-2]

'NHY'

>>> s="PYTHON"

>>> print(s)

PYTHON

>>> s in s

True

>>> s in s[::-1]

False

>>> s="MADAM"

>>> s in s[::-1]

True

----------------------------

>>> s="MADAM"

>>> s not in s[::-1][::]

False

---------------------------

>>> lst=[10,"Rossum",True,45,2+3j]

>>> print(lst)

[10, 'Rossum', True, 45, (2+3j)]

>>> 10 in lst

True

>>> True in lst

True

>>> False not in lst

True

>>> False in lst

False

--------------------------------

>>> lst=[10,"Rossum",True,45,2+3j]

>>> print(lst)

[10, 'Rossum', True, 45, (2+3j)]

>>> "sum" in lst

False

>>> "sum" in lst[1]

True

>>> lst[1] in lst[-4][::]

True

>>> lst[1][::-2] not in lst[-4][::-2]-----------False

-------------------------------

>>> lst=[10,"Rossum",True,45,2+3j]

>>> lst[-1].real in lst[-1]-------------TypeError: argument of type 'complex' is not iterable

-------------------------------------------------------------------------------------------------------------

===========================================

4. Logical Operators

===========================================

=>The purpose of Logical Operators is that "To Connect two or more Relational Expressions".

=>If two or more Relational Expressions connected with Logical Operators then it is called Logical Expression or Compound Conditions(Multiple condition ).

=>The result of Logical Expression or Compound Conditions is either True or False.

=>In Python Programming, we have three types of Logical Operators. They are given in the following Table.

========================================================================

SLNO SYMBOL MEANING

========================================================================

1 and Physical ANDing

2. or Physical ORing

3. not -------------------------------

=======================================================================

1) and:

---------------

=>The Functionality of "and" operator is described in the following Truth Table.

---------------------------------------------------------------------

Rel Expr1 RelExpr2 RelExpr1 and Rel Expr2

-----------------------------------------------------------------------

False False False

True False False

False True False

True True True

--------------------------------------------------------------------

Examples:

----------------

>>> print (100>20 and 20>4)---------------True

>>> print (100>200 and 20>4)------------False------Short Circuit Evaluation

>>> print (-100>200 and 20>4 and 10>2)--False---Short Circuit Evaluation

-----------------------------------------------------------------------

=>Short Circuit Evaluation (or) Lazy Evaluation in the case of "and"

-----------------------------------------------------------------------

In the case of "and" operator, if First Relational Expression result is False Then PvM will not evaluate rest of the Relational Expression and total Logical Expression result will be considered as False. This process is called Short Circuit Evaluation (or) Lazy Evaluation of "and" operator.

-----------------------------------------------------------------------

2) or :

---------------------------------------------------------------------------------------------------

=>The Functionality of "or" operator is described in the following Truth Table.

-----------------------------------------------------------------------

Rel Expr1 RelExpr2 RelExpr1 or Rel Expr2

-----------------------------------------------------------------------

False False False

True False True

False True True

True True True

-----------------------------------------------------------------------

Examples:

-----------------

>>> print(10>2 or 10>4)----------True-----Short Circuit Evaluation

>>> print(10<2 or 20>3 or 5>50)-------True----Short Circuit Evaluation

>>> print(10>2 or 20==3 or 5!=50)------True---Short Circuit Evaluation

>>> print(10==2 or 20==3 or 5>=50)------False

=>Short Circuit Evaluation (or) Lazy Evaluation in the case of "or"

-----------------------------------------------------------------------

In the case of "or" operator, if First Relational Expression result is True Then PvM will not evaluate rest of the Relational Expressions and total Logical Expression result will be considered as True. This process is called Short Circuit Evaluation (or) Lazy Evaluation of "or" operator.

-----------------------------------------------------------------------

3) not operator:

--------------------------

=>The Functionality of "not" operator is described in the following Truth Table.

--------------------------------------------------------------------

Rel Expr1 not RelExpr1

-----------------------------------------------------------------------

False True

True False

-----------------------------------------------------------------------

Examples:

----------------

>>> a=10

>>> b=20

>>> a==b-----------False

>>> not(a==b) -----------True

>>> print(not (10==2 or 20==3 or 5>=50))-----------True

-----------------------------

>>> a=True

>>> not a-------------False

>>> not False-----------True

-----------------------------------------------------------------------

Special Examples:

-------------------------------

>>> 100>20 and 100>40---------------True

>>> 100>20 or 100>40-----------True

============================

>>> 100 and -100

-100

>>> 100 and 0

0

>>> -100 and -225

-225

>>> 0 and 100

0

>>>

>>> 100 and -1234567

-1234567

>>> 100 and 0

0

>>> 0 and 345

0

>>> 0 and 0

0

>>>

>>> 100 or 200

100

>>> -100 or -223

-100

>>> 0 or -223

-223

>>> 0 or 0

0

>>> not (0 or 0)

True

>>> 100 and -100

-100

>>> 0 and 10

0

>>> 10 and 20

20

>>> 0 and -100

0

>>> 100 and 0

0

>>> 100 and 200 and -100

-100

>>> 100 and 200 and 0

0

>>> 100 and -200 and 0 and 234

0

>>>

>>>

>>> "KVR" and "PYTHON"

'PYTHON'

>>> "KVR" and 0

0

>>> "KVR" and "Python" and True

True

>>>

>>>

>>>

>>> 100 or 200

100

>>> 100 or 0

100

>>> 0 or 200

200

>>> 0 or 300 or 300 or 100

300

>>>

>>>

>>> "KVR" or "PYTHON"

'KVR'

>>>

>>>

>>>

>>> 10 or 20 and 30 or 30 and 450

10

>>> 10 and 20 or 300 and 450 and 0 or 23

20

>>> 100 and "or"

'or'

>>> "and" and "or"

'or'

>>> "and" or "or"

'and'

==========================================

Identity Operators (Python Command Prompt)

==========================================

=>The purpose of Identity Operators is that "To Chech the Memory Addres of Two Objects".

=>In Python Programming, we have two types of Identity Operators. They are

1. is

2. is not

-----------------------------------------------------------------------

1) is

-----------------------------------------------------------------------

Syntax:- object1 is object2

=>"is" opetrator Returns True provided Memory address of Object1 and Object2 are SAME

=>"is" opetrator Returns False provided Memory address of Object1 and Object2 are

DIFFERENT

-----------------------------------------------------------------------

2) is not

-----------------------------------------------------------------------Syntax: - object1 is not object2

=>"is not" opetrator Returns True provided Memory address of Object1 and Object2 are

DIFFERENT

=>"is" opetrator Returns False provided Memory address of Object1 and Object2 are SAME

-----------------------------------------------------------------------

Examples:

-----------------------------------------------------------------------

>>> a=None

>>> b=None

>>> print(a,id(a))-----------------None 140709648996344

>>> print(b,id(b))-----------------None 140709648996344

>>> a is b-------------------True

>>> a is not b-------------False

-------------------------------------------------------------

>>> d1={10:"Apple",20:"Mango",30:"CApple"}

>>> d2={10:"Apple",20:"Mango",30:"CApple"}

>>> print(d1,id(d1))---------{10: 'Apple', 20: 'Mango', 30: 'CApple'} 1938668998592

>>> print(d2,id(d2))------------{10: 'Apple', 20: 'Mango', 30: 'CApple'} 1938668998656

>>> d1 is d2-----False

>>> d1 is not d2---------True

-------------------------------------------------------------

>>> s1={10,20,30,40}

>>> s2={10,20,30,40}

>>> print(s1,id(s1))------------{40, 10, 20, 30} 1938669202432

>>> print(s2,id(s2))------------{40, 10, 20, 30} 1938673175904

>>> s1 is s2-----------False

>>> s1 is not s2-------True

>>> fs1=frozenset(s1)

>>> fs2=frozenset(s1)

>>> print(fs1,id(fs1))-------------------frozenset({40, 10, 20, 30}) 1938673176352

>>> print(fs2,id(fs2))----------------frozenset({40, 10, 20, 30}) 1938673177696

>>> fs1 is fs2--------------------False

>>> fs1 is not fs2--------------True

-------------------------------------------------------------

>>> t1=(10,20,30)

>>> t2=(10,20,30)

>>> print(t1,id(t1))---------------(10, 20, 30) 1938669461184

>>> print(t2,id(t2))-------------(10, 20, 30) 1938673242496

>>> t1 is t2-----------False

>>> t1 is not t2--------------True

>>> l1=[10,"Python","R"]

>>> l2=[10,"Python","R"]

>>> print(l1,id(l1))--------------[10, 'Python', 'R'] 1938673238208

>>> print(l2,id(l2))-------------[10, 'Python', 'R'] 1938669045952

>>> l1 is l2-------------False

>>> l1 is not l2---------True

-------------------------------------------------------------

>>> r1=range(10)

>>> r2=range(10)

>>> print(r1,id(r1))---------------range(0, 10) 1938669658224

>>> print(r2,id(r2))--------------range(0, 10) 1938669663312

>>> r1 is r2----------------False

>>> r1 is not r2----------True

>>> b1=bytes([10,20,30])

>>> b2=bytes([10,20,30])

>>> print(b1,id(b1))---------b'\n\x14\x1e' 1938669663408

>>> print(b2,id(b2))---------b'\n\x14\x1e' 1938669663456

>>> b1 is b2----------False

>>> b1 is not b2-------True

>>> ba1=bytearray((10,20,123))

>>> ba2=bytearray((10,20,123))

>>> print(ba1,id(ba1))----------bytearray(b'\n\x14{') 1938673243440

>>> print(ba2,id(ba2))-----------bytearray(b'\n\x14{') 1938673243632

>>> ba1 is ba2----------------False

>>> ba1 is not ba2-------------True

MOST IMP

>>> s1="PYTHON"

>>> s2="PYTHON"

>>> print(s1,id(s1))-------------------PYTHON 1938673243696

>>> print(s2,id(s2))-------------------PYTHON 1938673243696

>>> s1 is s2------------------------True

>>> s1 is not s2------------------False

>>> s1="INDIA"

>>> s2="INDIA"

>>> s1 is s2-----------------------True

>>> s1 is not s2-----------------False

>>> s1="INDIA"

>>> s2="INDia"

>>> s1 is s2------------False

>>> s1 is not s2---------True

-------------------------------------------------------------------------------------------------------------------------------

>>> a=2+3j

>>> b=2+3j

>>> print(a,id(a))---------(2+3j) 1938668707664

>>> print(b,id(b))----------(2+3j) 1938668707696

>>> a is b-------False

>>> a is not b------True

>>> a=True

>>> b=True

>>> print(a,id(a))----------True 140709648943976

>>> print(b,id(b))--------------True 140709648943976

>>> a is b---------------True

>>> a is not b----------False

>>> a=1.2

>>> b=1.2

>>> print(a,id(a))------1.2 1938668708560

>>> print(b,id(b))--------1.2 1938668708144

>>> a is b--------False

>>> a is not b-----True

------------------------------------------------------------------

>>> a=10

>>> b=10

>>> print(a,id(a))

10 1938667667984

>>> print(b,id(b))

10 1938667667984

>>> a is b

True

>>> a is not b

False

>>> a=256

>>> b=256

>>> print(a,id(a))

256 1938667675856

>>> print(b,id(b))

256 1938667675856

>>> a is b

True

>>> a is not b

False

>>> a=300

>>> b=300

>>> print(a,id(a))

300 1938668707664

>>> print(b,id(b))

300 1938668706064

>>> a is b

False

>>> a is not b

True

>>> a=257

>>> b=257

>>> print(a,id(a))

257 1938668711440

>>> print(b,id(b))

257 1938668707664

>>> a is b

False

>>> a is not b

True

>>> a=0

>>> b=0

>>> print(a,id(a))

0 1938667667664

>>> print(b,id(b))

0 1938667667664

>>> a is b

True

>>> a is not b

False

>>> a=-4

>>> b=-4

>>> print(a,id(a))

-4 1938667667536

>>> print(b,id(b))

-4 1938667667536

>>> a is b

True

>>> a is not b

False

>>> a=-1

>>> b=-1

>>> print(a,id(a))

-1 1938667667632

>>> print(b,id(b))

-1 1938667667632

>>> a is b

True

>>> a is not b

False

>>> a=-5

>>> b=-5

>>> print(a,id(a))

-5 1938667667504

>>> print(b,id(b))

-5 1938667667504

>>> a is b

True

>>> a is not b

False

>>> a=-6

>>> b=-6

>>> print(a,id(a))

-6 1938668707664

>>> print(b,id(b))

-6 1938668711440

>>> a is b

False

>>> a is not b

True

>>>

>>>

>>>

>>> a,b=300,300

>>> print(a,id(a))

300 1938668707696

>>> print(b,id(b))

300 1938668707696

>>> a is b

True

>>> a is not b

False

>>> a,b=-256,-256

>>> print(a,id(a))

-256 1938668706064

>>> print(b,id(b))

-256 1938668706064

>>> a is b

True

>>> a is not b

False

>>> l1,l2=[10,"KVR"],[10,"KVR"]

>>> print(l1,id(l1))

[10, 'KVR'] 1938669059648

>>> print(l2,id(l2))

[10, 'KVR'] 1938673238272

>>> l1 is l2

False

>>> l1 is not l2

True

-----------------------------------------------------------------------

===========================================

Python Ternary Operator

===========================================

=>The name of Python Ternary Operator is " if else Operator "

Syntax:- varname= Expr1 if Test Cond else Expr2

Explanation:

--------------------

=>Here "if" and " else " are called Keywords

=>The Execution Process of if..else operator (Python Ternary Operator) is that" if the Test Cond result is True then PVM executes Expr1 and whose Result assigned to Varname. If the Resul of Test Cond is False PVM executes Expr2 and whose Result assigned to Varname".

#Program for finding big and small and equality of two numbers by using Ternary Operator

#bigsmallex1.py

a=float(input("Enter Value of a:")) # a=1

b=float (input("Enter Value of b:")) # b=20

big=a if a>b else b

small= a if a<b else b

print("big({},{})={}".format(a,b,big))

print("small({},{})={}".format(a,b,small))

#Program for finding big and small and equality of two numbers by using Ternary Operator

#bigsmallex2.py

a,b=float(input("Enter Value of a:")),float(input("Enter Value of b:"))

print("big({},{})={}".format(a,b,a if a>b else b))

print("small({},{})={}".format(a,b,a if a<b else b))

#Program for finding big and small and equality of two numbers by using Ternary Operator

#bigsmallex3.py

a,b=float(input("Enter Value of a:")),float(input("Enter Value of b:"))

big="BOTH VALUE ARE EQUAL" if a==b else a if a>b else b

small=a if (a<b) else b if b<a else "BOTH VALUE ARE EQUAL"

print("big({},{})={}".format(a,b,big))

print("small({},{})={}".format(a,b,small))

#Program for finding big and small and equality of Three numbers by using Ternary Operator

#bigsmallex4.py

a,b,c=float(input("Enter Value of a:")),float(input("Enter Value of b:")) ,float(input("Enter Value of c:"))

big="ALL VALUES ARE EQUAL" if (a==b) and (b==c) else a if (a>b) and (a>c) else b if (b>a) and (b>c) else c

sm=a if (a<b) and (a<c) else b if (b<a) and (b<c) else c if (c<a) and (c<b) else "ALL ARE EQUAL"

print("big({},{},{})={}".format(a,b,c,big))

print("small({},{},{})={}".format(a,b,c,sm))

#Program for finding big and small and equality of Three numbers by using Ternary Operator

#bigsmallex5.py

a,b,c=float(input("Enter Value of a:")),float(input("Enter Value of b:")) ,float(input("Enter Value of c:"))

big="ALL VALUES ARE EQUAL" if (a==b==c) else a if (b<a>c) else b if (a<b>c) else c

sm=a if (b>a<c) else b if (a>b<c) else c if (a>c<b) else "ALL ARE EQUAL"

print("big({},{},{})={}".format(a,b,c,big))

print("small({},{},{})={}".format(a,b,c,sm))

=============================================

Flow Control Statements in Python (OR)

Control Structures in Python

=============================================

Index

--------------

=>Purpose of Flow Control Statements in Python

=>Types of Control Statements in Python

I) Conditional or Selection or Branching Statements

1. Simple if statement

2. if..else statement

3. if..elif..else statement

4. match case statement (Python 3.10 Version Onwards)

=>Programming Examples

II) Looping or Iterating or Repeatative Statements

1. while loop or while..else loop

2. for loop or for..else loop

=>Programming Examples

III) Transfer Flow Control Statements

1. break

2. continue

3. pass

=>Programming Examples

-----------------------------------------------------------------------

=>Inner or Nested Loops

a) while loop in while Loop

b) for loop in for loop

c) while loop in for loop

d) for loop in while loop

=>Programming Examples

=============================================

Flow Control Statements in Python

(OR)

Control Structures in Python

=============================================

=>The Purpose of Flow Control Statements in Python is that "To Perform certain Operation Either ONCE (True--X-Operation and False--Y-Operation) OR Repeatedly for finite number of Times Until Condtion Becomes False".

=>In Python Programming, Flow Control Statements in Python are classfied into 3 types. They are

1. Conditional or Selection or Branching Statements

2. Looping or Iterating or Repeatative Statements

3. Transfer Flow Control Statements

=========================================================================

Type

1. Conditional or Selection or Branching Statements

===============================================

=>The purpose Conditional or Selection or Branching Statements is that " To perform Certain Operation Only Once depends on Condition Evaluation".

=>The purpose Conditional or Selection or Branching Statements is that "To Perform X-Operation Only once when the condition is True or To Perform Y-Operation Only once when the condition is False.".

=>In Python programming, Conditional or Selection or Branching Statements are classified into 4 types. They are

1. Simple if statement

2. if..else statement

3. if..elif..else statement

4. match case statement (Python 3.10 Version Onwards)

-------------------------------------------------------------------------------------------------------------------------------

#Moviee.py

tkt=input("D u have ticket(yes/no):")

if(tkt.lower()=="yes"):

print("Enter into theater")

print("Watch Moviee")

print("Enjoy")

print("\nGoto Home and Read")

#WAP program which will accept any numerical number and decide the biggest among them two numbers

#bigex1.py

a=float(input("Enter value of a:")) # a=100

b=float(input("Enter value of b:")) # b=200

if ( a>b ):

print("max({},{})={}".format(a,b,a))

if(b>a):

print("max({},{})={}".format(a,b,b))

print("\nProgram execution Completed!")

#WAP program which will accept any numerical number and decide the biggest among them two numbers

#bigex2.py

a=float (input("Enter value of a:")) # a=100

b=float (input("Enter value of b:")) # b=100

if ( a>b ):

print("max({},{})={}".format(a,b,a))

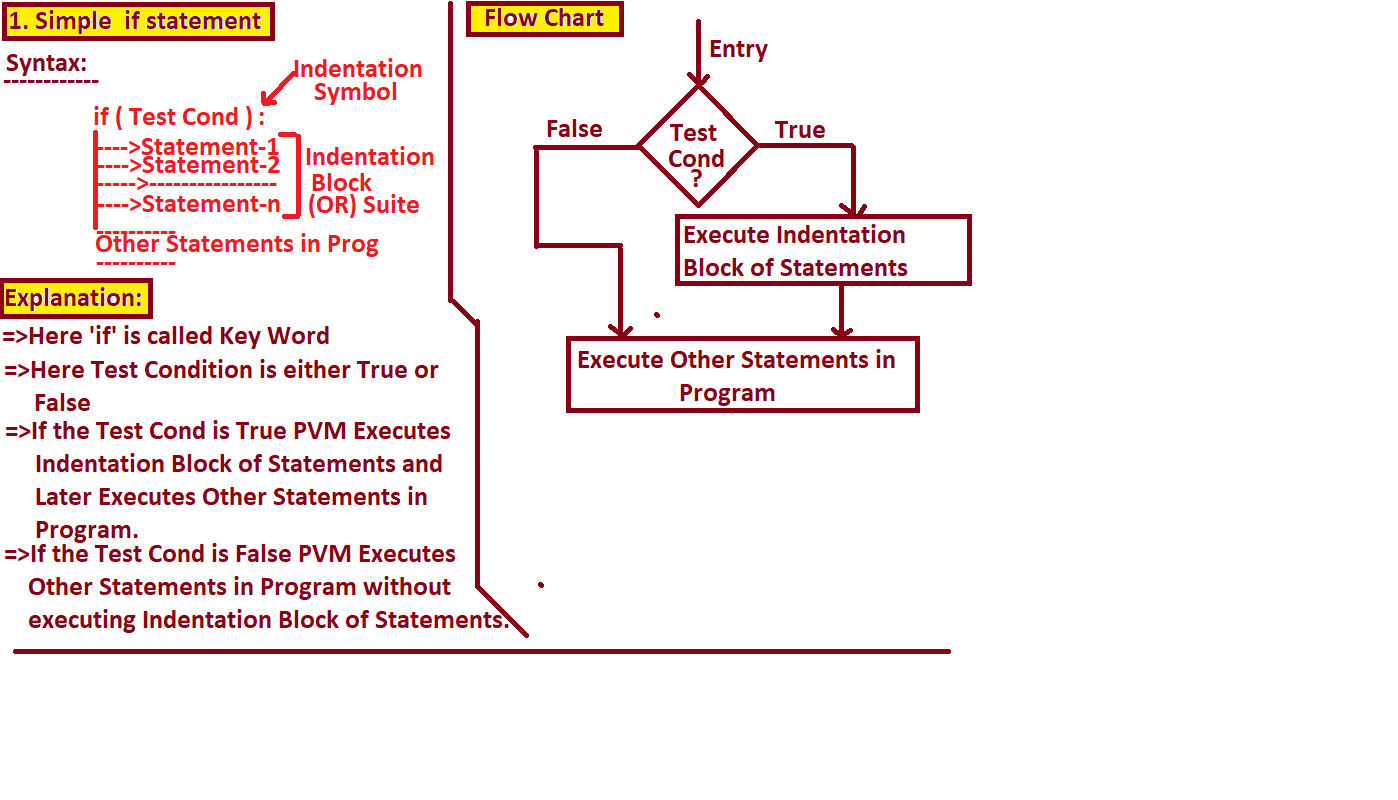
if(b>a):

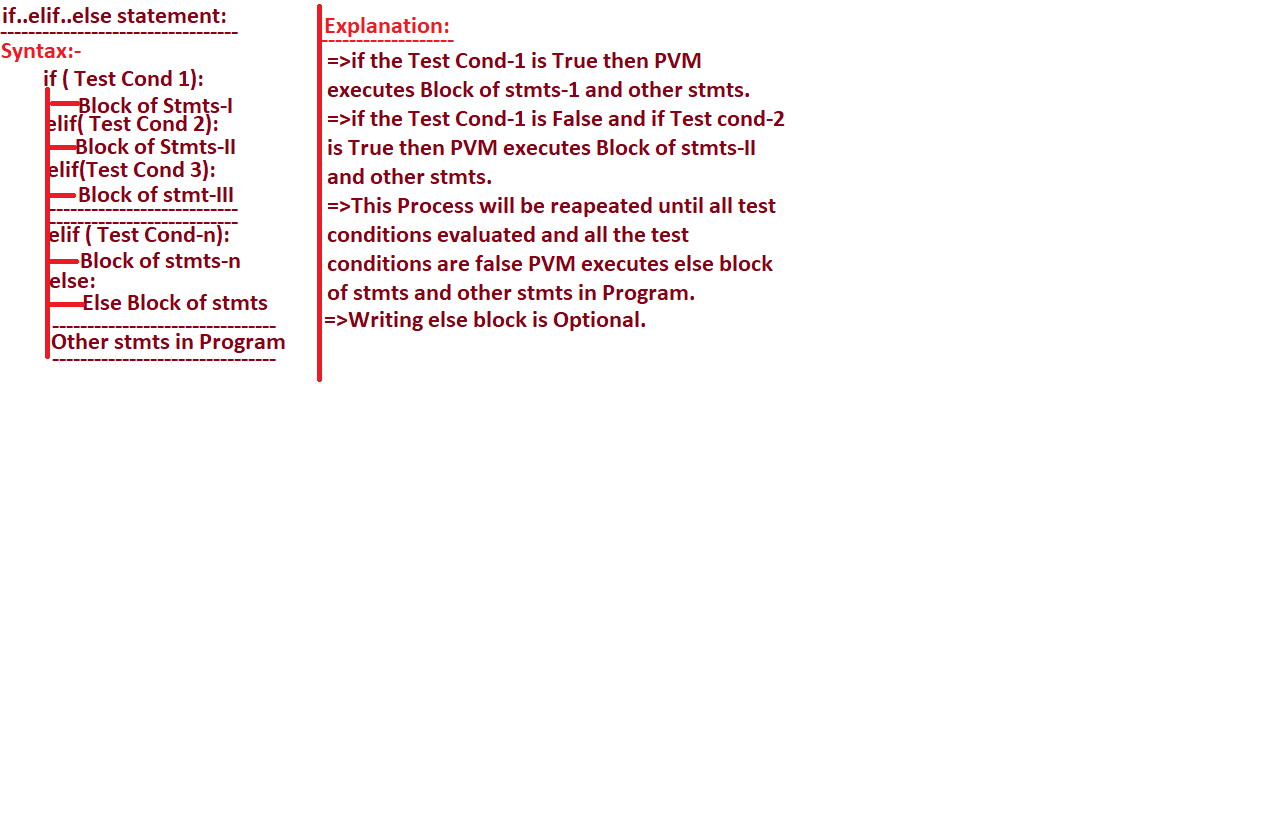
print("max({},{})={}".format(a,b,b))

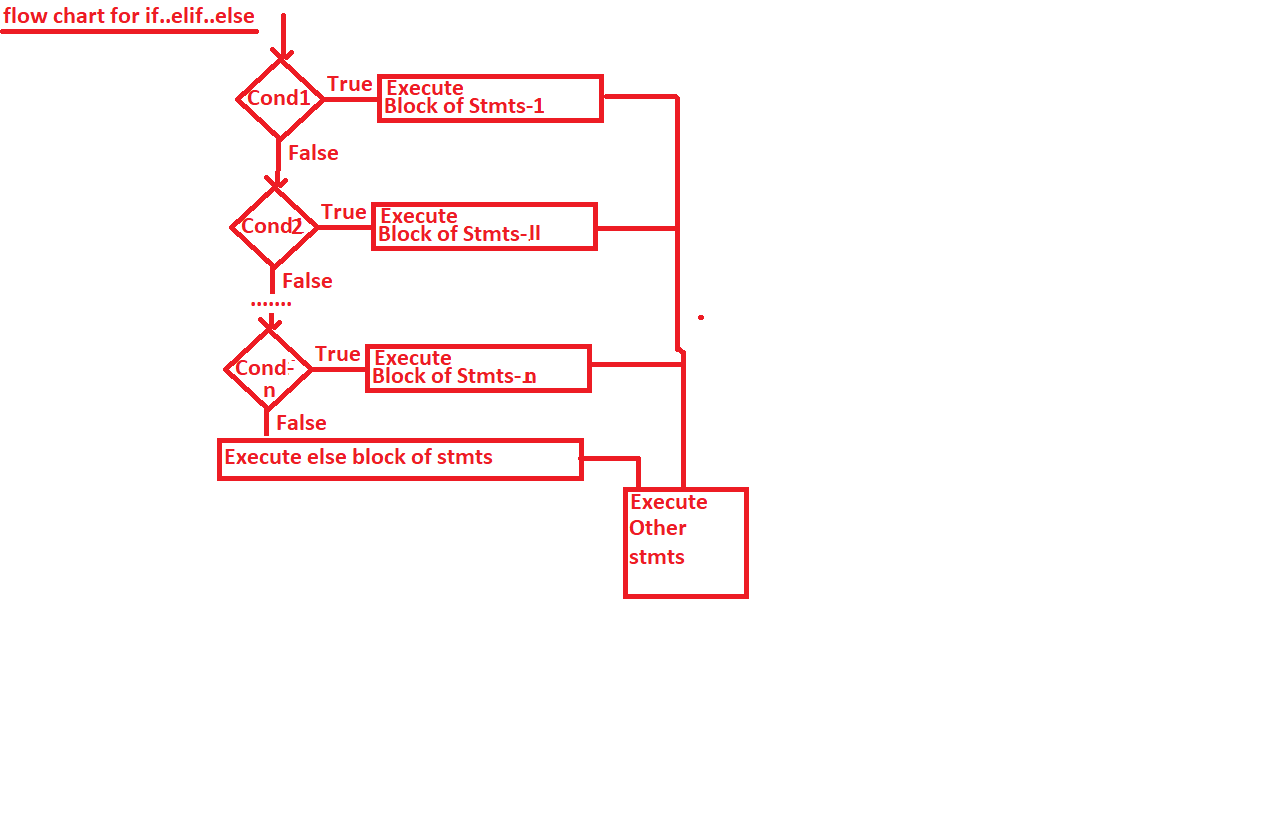
if(a==b):

print("Both values are Equal:")

print("\nProgram execution Completed!")







#Program for accepting 3 values and find biggest among them by using if..else statement

#BigThreeEx1.py

a=float(input("Enter Value of a:")) # a=10

b=float(input("Enter Value of b:"))# b=20

c=float(input("Enter Value of c:")) # c=20

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

print("big({},{},{})={}".format(a,b,c,a))

else:

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

print("big({},{},{})={}".format(a,b,c,b))

else:

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

print("big({},{},{})={}".format(a,b,c,c))

else:

print("ALL VALUES ARE EQUAL")

#Program for accepting 3 values and find biggest among them by using if..else statement

#BigThreeEx3.py

a=float(input("Enter Value of a:")) # a=20

b=float(input("Enter Value of b:"))# b=20

c=float(input("Enter Value of c:")) # c=20

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

print("big({},{},{})={}".format(a,b,c,a))

elif(b>a) and (b>=c):

print("big({},{},{})={}".format(a,b,c,b))

elif(c>a) and (c>b):

print("big({},{},{})={}".format(a,b,c,c))

elif(a==b==c):

print("ALL VALUES ARE EQUAL")

print("\nProgram is completed")

Program accepting a digit from key board and display its name

#digitex1.py

d=int(input("Enter a digit:")) # d= 0 1 2 3 4 5 6 7 8 9 99

if(d==0):

print("{} is ZERO".format(d))

else:

if(d==1):

print("{} is ONE".format(d))

else:

if(d==2):

print("{} is TWO".format(d))

else:

if(d==3):

print("{} is THREE".format(d))

else:

if(d==4):

print("{} is FOUR".format(d))

else:

if(d==6):

print("{} is SIX".format(d))

else:

if(d==5):

print("{} is FIVE".format(d))

else:

if(d==8):

print("{} is EIGHT".format(d))

else:

if(d==7):

print("{} is SEVEN".format(d))

else:

if(d==9):

print("{} is NINE".format(d))

else:

print("{} is a number:".format(d))

digitex1.py

Open with

#Program accepting a digit from key board and display its name

#digitex2.py

d=int(input("Enter a digit:")) # d= 0 1 2 3 4 5 6 7 8 9 99

if(d==0):

print("{} is ZERO".format(d))

elif(d==1):

print("{} is ONE".format(d))

elif(d==2):

print("{} is TWO".format(d))

elif(d==3):

print("{} is THREE".format(d))

elif(d==4):

print("{} is FOUR".format(d))

elif(d==5):

print("{} is FIVE".format(d))

elif(d==7):

print("{} is SEVEN".format(d))

elif(d==6):

print("{} is SIX".format(d))

elif(d==8):

print("{} is EIGHT".format(d))

elif(d==9):

print("{} is NINE".format(d))

else:

print("{} is NUMBER".format(d))

#Program accepting a digit from key board and display its name

#digitex3.py

d=int(input("Enter a digit:")) # d= 0 1 2 3 4 5 6 7 8 9 99

if(d==0):

print("{} is ZERO".format(d))

if(d==1):

print("{} is ONE".format(d))

if(d==2):

print("{} is TWO".format(d))

if(d==3):

print("{} is THREE".format(d))

if(d==4):

print("{} is FOUR".format(d))

if(d==5):

print("{} is FIVE".format(d))

if(d==7):

print("{} is SEVEN".format(d))

if(d==6):

print("{} is SIX".format(d))

if(d==8):

print("{} is EIGHT".format(d))

if(d==9):

print("{} is NINE".format(d))

if(d not in [0,1,2,3,4,5,6,7,8,9] ):

print("{} is NUMBER".format(d))

#Program accepting a digit from key board and display its name

#digitex4.py

d={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:"SIX",7:"SEVEN",8:"EIGHT",9:"NINE"}

dig=int(input("Enter a digit:"))

res=d.get(dig)

if(res!=None):

print("{} is {}".format(dig,res))

else:

print("{} is Number:".format(dig))

#Program accepting a digit from key board and display its name

#digitex4.py

d={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:"SIX",7:"SEVEN",8:"EIGHT",9:"NINE"}

dig=int(input("Enter a digit:"))

res= d.get(dig) if (d.get(dig)!=None) else "Not a Number"

print("{} is {}".format(dig,res))

print("===================OR=========================")

print("{} is {}".format(dig,d.get(dig) if (d.get(dig)!=None) else "Not a Number"))

#Program for acl are and perimeter of circle

#Circle.py

r=float (input("Enter Radious:"))

if(r>=0):

ar=3.14\*r\*\*2

pr=2\*3.14\*r

print ("Area of circle={}".format(ar))

print("Perimter of circle={}".format(pr))

else:

print("{} is Invalid input:".format(r))

#Program for acl are and perimeter of circle

#Circle.py

r=float (input("Enter Radious:"))

if(r>=0):

ar=3.14\*r\*\*2

pr=2\*3.14\*r

print ("Area of circle={}".format(ar))

print ("Perimter of circle={}".format(pr))

else:

print("{} is Invalid input:".format(r))

==========================================

match .. case statement (Python 3.10)

==========================================

=>It is one of new Feature in Python 3.10

=>This feature is also known as Multi way decision making statement.

=>It is always recommended to handling Pre-designed Conditions.

-----------------

=>Syntax:

-----------------

match (Choice Expression):

case label1:

block of statements-I

case label2:

block of statements-II

--------------------------------------

--------------------------------------

case label-n:

block of statements-n

case \_ : # default case block

default block of statements

--------------------------------------------------

--------------------------------------------------

Other statements in Program

--------------------------------------------------

-----------------------

Explanation:

-----------------------

1) here 'match' and 'case' are the keywords

2) here 'Choice expression' can be any data type value except float.

3) here the value of choice expression is comparing with case label1 . If it is True then execute Block of statements-I and also execute other statements in Program. Choice expression is not matching with case label1 then it compares with case label2 and if it matches then execute Block of statements-II and also execute other statements in Program and so on.

4) In general, if the value of choice expression is matching with any case label, then PVM executes corresponding block of statements and also executes other statements in the program.

5) if value of choice expression is not matching with any case labels, then PVM executes block of statements written under default case block and also executes Other statements in the program.

6) Writing default case label is optional.

7) If we write default case label then we must write it at last otherwise we get SyntaxError.

#WAPP which will implement the following mean driven application

import sys

#matchcaseex1.py

print("="\*50)

print("\tArithemetic Operations")

print("="\*50)

print("\t1.Addition:")

print("\t2. Substraction:")

print("\t3.Multiplication:")

print("\t4. Division:")

print("\t5.Modulo Div:")

print("\t6.Exponentiation:")

print("\t7.Exit:")

print("="\*50)

ch=int(input("Enter Ur Choice:"))

match(ch):

case 1:

a=float(input("Enter First Value for Addition:"))

b=float(input("Enter Second Value for Addition:"))

print("\tsum({},{})={}".format(a,b,a+b))

case 2:

a=float(input("Enter First Value for Substraction:"))

b=float(input("Enter Second Value for Substraction:"))

print("\tsub({},{})={}".format(a,b,a-b))

case 3:

a=float(input("Enter First Value for Multiplication:"))

b=float(input("Enter Second Value for Multiplication:"))

print("\tMul({},{})={}".format(a,b,a\*b))

case 4:

a=float(input("Enter First Value for Division:"))

b=float(input("Enter Second Value for Division:"))

print("\tDivision({},{})={}".format(a,b,a/b))

print("\tFloor Division({},{})={}".format(a,b,a//b))

case 5:

a=float(input("Enter First Value for Modulo Div:"))

b=float(input("Enter Second Value for Modulo Div:"))

print("\tModulo Div({},{})={}".format(a,b,a%b))

case 6:

a=float(input("Enter Base:"))

b=float(input("Enter Power:"))

print("\tpow({},{})={}".format(a,b,a\*\*b))

case 7:

print("Thx for using Program")

sys.exit()

case \_: # Default Case Block

print("{} is invalid Choice, try again:".format(ch))

#matchcaseex2.py

wkd=input("Enter a week name:")

match(wkd.lower()):

case "monday":

print("{} is working day".format(wkd))

case "tuesday":

print("{} is working day".format(wkd))

case "wednessday":

print("{} is working day".format(wkd))

case "thursday":

print("{} is working day".format(wkd))

case "friday":

print("{} is working day".format(wkd))

case "saturday":

print("{} is week end--underground action plans".format(wkd))

case "sunday":

print("{} is holiday and implementing UG Plans".format(wkd))

#matchcaseex3.py

wkd=input("Enter a week name:")

match(wkd.lower()):

case "monday"| "tuesday" | "wednessday"|"thursday"|"friday":

print("{} is working day".format(wkd))

case "saturday":

print("{} is week end--underground action plans".format(wkd))

case "sunday":

print("{} is holiday and implementing UG Plans".format(wkd))

case \_:

print("{} is not a week day".format(wkd))

#matchcaseex4.py

wkd=input("Enter a week name:")

match(wkd.lower()[0:3]):

case "mon"| "tue" | "wed"|"thu"|"fri":

print("{} is working day".format(wkd))

case "sat":

print("{} is week end--underground action plans".format(wkd))

case "sun":

print("{} is holiday and implementing UG Plans".format(wkd))

case \_:

print("{} is not a week day".format(wkd))

import sys

#nonmatchcaseex1.py

print("="\*50)

print("\tArithemetic Operations")

print("="\*50)

print("\t1.Addition:")

print("\t2.Substraction:")

print("\t3.Multiplication:")

print("\t4.Division:")

print("\t5.Modulo Div:")

print("\t6.Exponentiation:")

print("\t7.Exit:")

print("="\*50)

ch=int(input("Enter Ur Choice:"))

if(ch==1):

a=float(input("Enter First Value for Addition:"))

b=float(input("Enter Second Value for Addition:"))

print("\tsum({},{})={}".format(a,b,a+b))

elif(ch==2):

a=float(input("Enter First Value for Substraction:"))

b=float(input("Enter Second Value for Substraction:"))

print("\tsub({},{})={}".format(a,b,a-b))

elif(ch==3):

a=float(input("Enter First Value for Multiplication:"))

b=float(input("Enter Second Value for Multiplication:"))

print("\tMul({},{})={}".format(a,b,a\*b))

elif(ch==4):

a=float(input("Enter First Value for Division:"))

b=float(input("Enter Second Value for Division:"))

print("\tDivision({},{})={}".format(a,b,a/b))

print("\tFloor Division({},{})={}".format(a,b,a//b))

elif(ch==5):

a=float(input("Enter First Value for Modulo Div:"))

b=float(input("Enter Second Value for Modulo Div:"))

print("\tModulo Div({},{})={}".format(a,b,a%b))

elif(ch==6):

a=float(input("Enter Base:"))

b=float(input("Enter Power:"))

print("\tpow({},{})={}".format(a,b,a\*\*b))

elif(ch==7):

print("Thx for using Program")

exit()

else: # Default Case Block

print("{} is invalid Choice, try again:".format(ch))

=================================================== Looping or Iterating or Repeatative Statements

===================================================

=>The purpose of Looping statements is that "To perform Certain Operation Repeatedly for finite number of times until Test Cond Becomes False."

=>In Python Programming, we have 2 types of Looping statements. They are

1. while loop (OR) while ... else loop

2. for loop (OR) for.... else loop

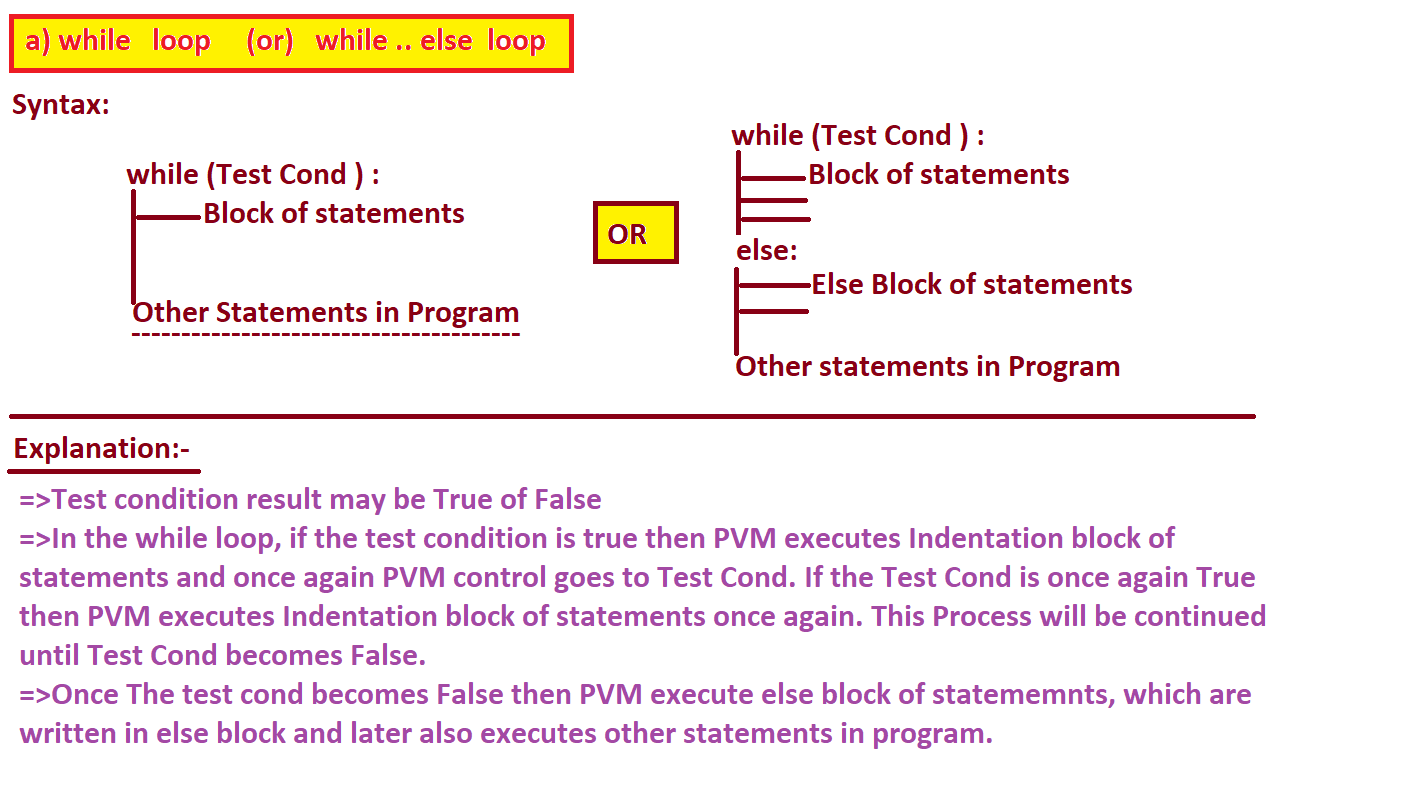
=>At the time of dealing with looping statements, Programmer must ensure there must 3 parts. They are

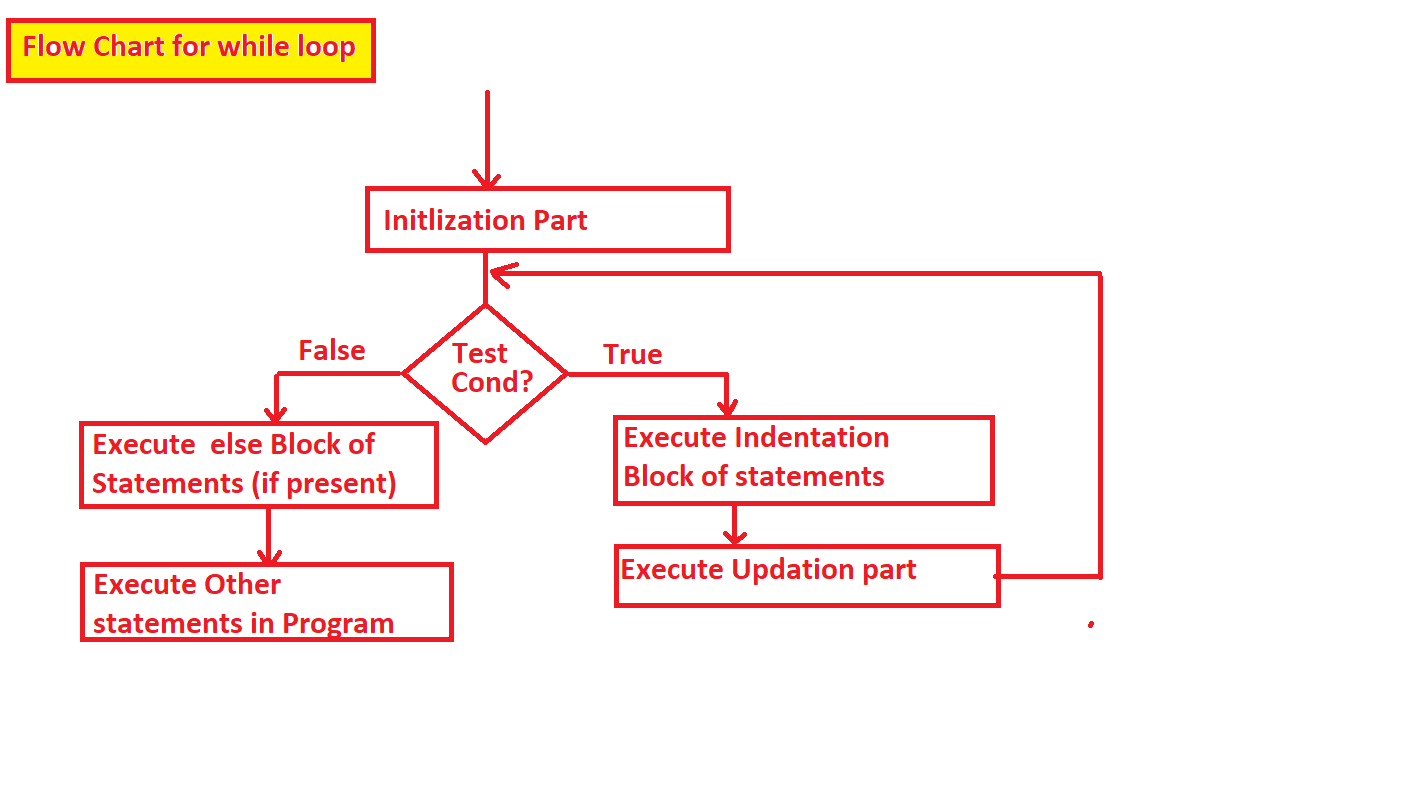
1. Initlization Part (From Where to Start )

2. Conditioinal Part( Upto How Many times to repeat

3. Updation Part (Incrmentation or decrementation )

---------------------------------------------------------------------------------------------------------------------------





==============================================

2. for loop or for ...else loop

==============================================

Syntax1:-

-----------

for varname in Iterable\_object:

----------------------------------------

Indentation block of stmts

----------------------------------------

---------------------------------------------------

Other statements in Program

---------------------------------------------------

---------------

Syntax2:

---------------

for varname in Iterable\_object:

----------------------------------------

Indentation block of stmts

----------------------------------------

else:

----------------------------------------

else block of statements

----------------------------------------

---------------------------------------------------

Other statements in Program

---------------------------------------------------

Explanation:

-----------------------

=>Here 'for' and 'else' are keywords

=>Here Iterable\_object can be Sequence(bytes,bytearray,range,str),

list(list,tuple),set(set,frozenset) and dict.

=>The execution process of for loop is that " Each of Element of Iterable\_object selected, placed in varname and executes Indentation block of statements".This Process will be repeated until all elements of Iterable\_object completed.

=>After execution of Indentation block of statements, PVM executes else block of statements which are written under else block and later PVM executes Other statements in Program.

=>Writing else block is optional.

#WAP program which will generate 1 to n numbers where n must be positive value

#NumGenEx1.py

n=int(input("Enter How Many Values u want to generate:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

print("="\*50)

print("Numbers within :{}".format(n))

print("="\*50)

i=1 # Initlization Part

while(i<=n): # Cond Part

print("\t{}".format(i))

i=i+1 # Updation Part

#WAP program which will generate 1 to n numbers where n must be positive value

#NumGenEx2.py

n=int(input("Enter How Many Values u want to generate:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

print("="\*50)

print("Numbers within :{}".format(n))

print("="\*50)

i=1 # Initlization Part

while(i<=n): # Cond Part

print("\t{}".format(i))

i=i+1 # Updation Part

else:

print("\*"\*50)

#WAP program which will generate even number within n where n must be the positive int value

#EvenNumbers.py

n=int(input("Enter How Many Range of Even Numbers u want to genertate:"))

if(n<=0):

print("{} is invalid input:".format(n))

else:

print("="\*50)

print("List of Even Numbers within:{}".format(n))

print("="\*50)

i=2 # Initlization Part

while(i<=n): # Cond Part

print("\t{}".format(i))

i=i+2 #Updation Part

else:

print("="\*50)

#WAP program which will generate even number within n where n must be the positive int value

#EvenNumbers.py

n=int(input("Enter How Many Range of Even Numbers u want to genertate:"))

if(n<=0):

print ("{} is invalid input:".format(n))

else:

print("="\*50)

print("List of Even Numbers within:{}".format(n))

print("="\*50)

i=2 # Initlization Part

while(i<=n): # Cond Part

print("\t{}".format(i))

i=i+2 #Updation Part

else:

print("="\*50)

#WAP program which will generate n to 1 where n is the positive int value

#

n=int(input("Enter a number for generating mul table:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

print("="\*50)

print("Mul Table for {}".format(n))

print("="\*50)

for i in range(1,11):

print("\t{} x {} = {}".format(n,i,n\*i))

else:

print("#"\*50)

#whileforloop.py

s="PYTHON” # Iterable-object

print("====By using While Loop=====")

i=len(s)-1

while(i>=0):

print("\t{}”. format(s[i]))

i=i-1

print("----------------------------------------------")

print("\n====By using For Loop=====")

for ch in s[::-1]:

print("\t{}".format(ch))

#Wap program which will accept the line of text and find and display number of vowels

#VowelsCount.py

line=input ("Enter Line of Text:") # Python is an oop lang

cnt=0

for ch in line:

if ch in ['a','e','i','o','u','A','E','I','O','U']:

print("\t{}".format(ch))

cnt=cnt+1

else:

print("Number of Vowels={}".format(cnt))

#Wap program which will accept the line of text and find and display number of vowels

#VowelsCount1.py

line=input("Enter Line of Text:") # Python is an oop lang

cnt=0

for ch in line:

if ch.lower() in ['a','e','i','o','u']:

print("\t{}".format(ch))

cnt=cnt+1

else:

print("Number of Vowels={}".format(cnt))

#Wap program which will Test Whether the given number is perfect or not

#perfect.py

n=int(input("Enter a Number:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

print("-"\*40)

print("Factors of {}".format(n))

print("-"\*40)

fs=0

for i in range(1,(n//2)+1):

if(n%i==0):

print("\t{}".format(i))

fs=fs+i

else:

print("-"\*50)

print("Factors Sum=",fs)

if(n==fs):

print("{} is Perfect:".format(n))

else:

print("{} is Not Perfect:".format(n))

#WAP program which will find some of first n natural numbers.some of square naturals number ,some of cubes of natural numbers..

#NatSumsSumEx1.py

n=int(input("Enter How Many Natural Numbers sum u want to Find:"))

if(n<=0):

print("{} is invalid input”. Format0028n))

else:

print("="\*50)

print("\tNatSums\tSquare Nums\tCubes Nums")

s,ss,cs=0,0,0

print("="\*50)

for i in range(1,n+1):

print("\t{}\t\t{}\t{}".format(i,i\*\*2,i\*\*3))

s=s+i

ss=ss+i\*\*2

cs=cs+i\*\*3

else:

print("="\*50)

print("\t{}\t\t{}\t{}".format(s,ss,cs))

print("="\*50)

#Program for cal factorial of a Number--- 1 x 2 x 3 x 4.......n

#FactorialEx1.py

n=int(input("Enter a number:"))

if(n<0):

print("{} is invalid input".format(n))

else:

f=1

for i in range(1,n+1):

f=f\*i

else:

print("Factorial({})={}".format(n,f))

#Program for cal factorial of a Number--- n x n-1 x n-2------ 0!

# 4 x 3 x 2 x 1

#FactorialEx2.py

n=int(input("Enter a number:"))

if(n<0):

print("{} is invalid input".format(n))

else:

f=1

for i in range(n,0,-1):

f=f\*i

else:

print("{}!={}".format(n,f))

#WAP program which will accept any numerical int value and find its factors..

# HINT: n=6----Factors-------- 1 2 3

#FactorsEx1.py

n=int(input("Enter For Which Number u want to Find factors:"))

print("="\*50)

print("Factors of {}".format(n))

for i in range(1,(n//2)+1):

if(n%i==0):

print("\t{}".format(i))

print("="\*50)

break statement

---------------------------------

=>break is a key word

=>The purpose of break statement is that "To terminate the execution of loop logically when certain condition is satisfied and PVM control comes of corresponding loop and executes other statements in the program".

=>when break statement takes place inside for loop or while loop then PVM will not execute corresponding else block(bcoz loop is not becoming False) but it executes other statements in the program

-------------------

=>Syntax1:

-------------------

for varname in Iterable\_object:

------------------------------

if (test cond):

break

------------------------------

------------------------------

------------------

=>Syntax2:

-------------------

while(Test Cond-1):

------------------------------

if (test cond-2):

break

------------------------------

------------------------------

============================X=============================

continue statement

-------------------------------------

=>continue is a keyword

=>continue statement is used for making the PVM to go to the top of the loop without executing the following statements which are written after continue statement for that current Iteration only.

=>continue statement to be used always inside of loops.

=>when we use continue statement inside of loop then else part of corresponding loop also executes provided loop condition becomes false.

-----------------

=>Syntax:-

----------------

for varname in Iterable-object:

------------------------------------------

if ( Test Cond):

continue

statement-1 # written after continue statement

statement-2

statement-n

-----------------------------------------

-----------------------------------------

=>Syntax:-

----------------

while (Test Cond):

------------------------------------------

if ( Test Cond):

continue

statement-1 # written after continue stateemnt

statement-2

statement-n

-----------------------------------------

-----------------------------------------

#breakex1.py

s="PYTHON"

for ch in s:

print(ch)

print("-----------------------------")

for ch in s:

if(ch=="H"):

break

else:

print(ch)

else:

print("else part of for loop")

print("\tOther part of the program")

#breakex2.py

s="PYTHON"

for ch in s:

print(ch)

print("-----------------------------")

i=0

while(i<len(s)):

if(s[i]=="H"):

break

else:

print("\t{}".format(s[i]))

i=i+1

else:

print("else part of while loop")

print("\tOther part of the program")

lst=[10,"Rossum",34.56,True,False,2+3j]

for val in lst:

if(val==True):

break

else:

print("\t{}".format(val))

else:

print("Else part of for loop")

print("Other part of program")

#Program for deciding whether the given number is prime or Not

#Prime Number = Dividing by 1 and itself.---- 2 3 5 7 11 13 17....

#PrimeEx1.py

n=int(input("Enter a Number:")) #n=9

if(n<2):

print("{} is invalid input".format(n))

else:

dec="PRIME"

for i in range(2,n):

if(n%i==0):

dec="NOTPRIME"

break

if(dec=="PRIME"):

print("{} is Prime Number".format(n))

else:

print("{} is Not Prime Number".format(n))

#Program for deciding whetehr the given number is prime or Not

#Prime Number = Dividing by 1 and itself.---- 2 3 5 7 11 13 17....

#PrimeEx2.py

n=int(input("Enter a Number:")) #n=9

if(n<2):

print("{} is invalid input".format(n))

else:

prmno=True

for i in range(2,n):

if(n%i==0):

prmno=False

break

if(prmno==True):

print("{} is Prime Number".format(n))

else:

print("{} is Not Prime Number".format(n))

#continueex1.py

s="PYTHON"

for ch in s:

print("\t{}".format(ch))

print("-----------------------------------------")

for ch in s:

if(ch=="H"):

continue

else:

print("\t{}".format(ch))

else:

print("else part of for loop")

print("Other part of program")

#continueex2.py

s="PYTHON"

for ch in s:

if(ch=="Y") or (ch=="O"):

continue

else:

print("\t{}".format(ch))

else:

print("else part of for loop")

print("Other part of program")

#continueex3.py

lst=[10,20,-34,56,-12,0,45,-56,5]

print("Possitive Elements:")

for val in lst:

if(val<=0):

continue

print("\t{}".format(val))

#continueex4.py

lst=[10,20,-34,56,-12,0,45,-56,5]

print("Negative Elements:")

for val in lst:

if(val>=0):

continue

print("\t{}".format(val))

#continueex5.py

lst=[10,20,-34,56,-12,0,45,-56,5]

ps=0

print("List of Possitive Values:")

for val in lst:

if(val<=0):

continue

print("\t{}".format(val))

ps=ps+val

else:

print("Sum of Possitive Elements={}".format(ps))

print("---------------------------------------------------------------")

ns=0

for val in lst:

if(val>=0):

continue

print("\t{}".format(val))

ns=ns+val

else:

print("Sum of Negative Elements={}".format(ns))

print("---------------------------------------------------------------")

#continueex6.py

lst=[11,-24,12,56,-23,57,0,27,-45,71]

print("List of Even Numbers")

for val in lst:

if(val%2!=0) or (val<0):

continue

print("\t{}".format(val))

else:

print("List of Odd Numbers")

for val in lst:

if(val%2==0) or (val<0):

continue

print("\t{}”. format(val))

==================================================

Inner or Nested Loops

==================================================

=>The Process of Defining One Loop in another Loop is called Inner or Nested Loop

=>The Execution Process of Inner or Nested Loop is that "For Every Value of Outer Loop, inner Loop process repeates Multiple Finite number of times until Test Cond becomes False".

=>We can define Inner or Nested Loops in Four Ways. They are

------------------

Syntax-1: for loop in for loop

------------------

for varname1 in Iterable\_object:

----------------------------

----------------------------

for varname2 in Iterable\_object:

-----------------------

-----------------------

else:

---------------------------

else:

-----------------------------------

----------------------------------------------------------------------------------------------------------

Syntax-2: while loop in while loop

----------------------------------------------------------------------------------------------------------

----------------------------------

while(Test Cond1):

-----------------------------

-----------------------------

while (Test Cond2):

--------------------------

--------------------------

else:

-------------------

-------------------

else:

---------------------

----------------------

----------------------------------------------------------------------------------------------------------

Syntax-3: while loop in for loop

----------------------------------------------------------------------------------------------------------

for varname in Iterable\_object:

----------------------------

----------------------------

while (Test Cond):

--------------------------

--------------------------

else:

-------------------

-------------------

else:

-----------------------------------

----------------------------------------------------------------------------------------------------------

Syntax-4: for loop in while loop

----------------------------------------------------------------------------------------------------------

----------------------------------

while(Test Cond):

-----------------------------

-----------------------------

for varname in Iterable\_object:

-----------------------

-----------------------

else:

---------------------------

else:

---------------------

----------------------

#Program busing Inner for loops ---for in for loop

#InnerLoopEx1.py

for i in range(1,5):

print("Val of i-outer for loop:{}".format(i))

print("-"\*50)

for j in range(1,4):

print("\tVal of j-inner for loop:{}".format(j))

else:

print("I am out of inner-for loop")

print("-"\*50)

else:

print("I am out of outer-for loop")

""""

E:\KVR-PYTHON-11AM\LOOPS>py InnerLoopEx1.py

Val of i-outer for loop:1

--------------------------------------------------

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-for loop

--------------------------------------------------

Val of i-outer for loop:2

--------------------------------------------------

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-for loop

--------------------------------------------------

Val of i-outer for loop:3

--------------------------------------------------

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-for loop

--------------------------------------------------

Val of i-outer for loop:4

--------------------------------------------------

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-for loop

--------------------------------------------------

I am out of outer-for loop

"""

#Program by using Inner for loops ---while in while loop

#InnerLoopEx2.py

i=1

while(i<=4):

print ("Val of i-outer while loop: {}".format(i))

print ("-"\*50)

j=1

while(j<=3):

print("\tVal of j-inner for loop:{}".format(j))

j=j+1

else:

i=i+1

print("I am out of inner-while loop")

print("-"\*50)

else:

print ("I am out of outer-while loop")

""""

E:\KVR-PYTHON-11AM\LOOPS>py InnerLoopEx2.py

Val of i-outer while loop:1

--------------------------------------------------

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-while loop

--------------------------------------------------

Val of i-outer while loop:2

--------------------------------------------------

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-while loop

--------------------------------------------------

Val of i-outer while loop:3

--------------------------------------------------

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-while loop

--------------------------------------------------

Val of i-outer while loop:4

--------------------------------------------------

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-while loop

--------------------------------------------------

I am out of outer-while loop

"""

#Program by using Inner for loops ---while in for loop

#InnerLoopEx3.py

for i in range(4,0,-1):

print("Val of i--Outer For loop:{}".format(i))

print("-"\*50)

j=3

while(j>=1):

print("\tValue of j={}".format(j))

j=j-1

else:

print("I am out of inner while loop")

print("-"\*50)

else:

print("I am out of outer for loop")

""""

E:\KVR-PYTHON-11AM\LOOPS>py InnerLoopEx3.py

Val of i--Outer For loop:4

--------------------------------------------------

Value of j=3

Value of j=2

Value of j=1

I am out of inner while loop

--------------------------------------------------

Val of i--Outer For loop:3

--------------------------------------------------

Value of j=3

Value of j=2

Value of j=1

I am out of inner while loop

--------------------------------------------------

Val of i--Outer For loop:2

--------------------------------------------------

Value of j=3

Value of j=2

Value of j=1

I am out of inner while loop

--------------------------------------------------

Val of i--Outer For loop:1

--------------------------------------------------

Value of j=3

Value of j=2

Value of j=1

I am out of inner while loop

--------------------------------------------------

I am out of outer for loop """

#Program by using Inner for loops ---while in for loop

#InnerLoopEx4.py

i=1

while(i<=4):

print("Val of i--Outer while loop:{}".format(i))

print("-"\*50)

for j in range (3,0,-1):

print("\tVal of j-inner for loop:{}".format(j))

else:

print("I am out of inner-for loop")

print("-"\*50)

i=i+1

else:

print ("I am out of outer-while loop")

"""

E:\KVR-PYTHON-11AM\LOOPS>py InnerLoopEx4.py

Val of i--Outer while loop:1

--------------------------------------------------

Val of j-inner for loop:3

Val of j-inner for loop:2

Val of j-inner for loop:1

I am out of inner-for loop

--------------------------------------------------

Val of i--Outer while loop:2

--------------------------------------------------

Val of j-inner for loop:3

Val of j-inner for loop:2

Val of j-inner for loop:1

I am out of inner-for loop

--------------------------------------------------

Val of i--Outer while loop:3

--------------------------------------------------

Val of j-inner for loop:3

Val of j-inner for loop:2

Val of j-inner for loop:1

I am out of inner-for loop

--------------------------------------------------

Val of i--Outer while loop:4

--------------------------------------------------

Val of j-inner for loop:3

Val of j-inner for loop:2

Val of j-inner for loop:1

I am out of inner-for loop

--------------------------------------------------

I am out of outer-while loop

"""

#MulTablesUsingInnerLoops.py

n=int(input("Enter How Many Mul Tables u Want:"))

if(n<=0):

print ("{} is invalid input”. Format(n))

else:

for i in range(1,n+1): # Outer loop supply the Number

print("-"\*50)

print("Mul Table for :{}".format(i))

print("-"\*50)

for j in range(1,11): # Inner Loop--generates Mul table

print("\t {} x {} = {}".format(i,j,i\*j))

else:

print("-"\*50)

#RandomMulTablesUsingInnerLoops.py

n=int(input("Enter How Many Mul Tables u Want:"))

if(n<=0):

print ("{} is invalid input”. Format(n))

else: # 5 19 3 -3 6

lst=[] # create an empty list for adding random numbers

for i in range(1,n+1):

val=int(input("Enter {} Value:".format(i)))

lst.append(val)

else:

print("-"\*50)

print("Given Values:{}".format(lst)) # [5, 9, -4, 19, 2]

print("-"\*50)

for num in lst:

if(num<=0):pass

else:

print("Mul Table for :{}".format(num))

print("-"\*50)

for j in range(1,11): # Inner Loop--generates Mul table

print("\t {} x {} = {}".format(num,j,num\*j))

else:

print("-"\*50)

#WAPP Which will generate list of prime number within the given range

#RangePrimes.py

n=int(input("Enter How Many Primes u want within the range:"))

if(n<=1):

print("{} is invalid input".format(n))

else:

for num in range(2,n+1):

res="PRIME"

for i in range(2,num):

if(num%i==0):

res="NOTPRIME"

break

if(res=="PRIME"):

print("\t{}".format(num))

#RandomPrimes.py

n=int(input("Enter How Many Numbers u have to decide whether they are prime or not:"))

if(n<=1):

print("{} is invalid input".format(n))

else:

lst=list() # create an empty list for adding random numbers

for i in range(1,n+1):

val=int(input("Enter {} Value:".format(i)))

lst.append(val)

else:

print("-"\*50)

print("Given Values:{}".format(lst)) # [12, 11, -5, 7, 1, 3]

print("-"\*50)

#get primes

for num in lst:

if(num<=1):pass

else:

res="PRIME"

for i in range(2,num):

if(num%i==0):

res="NOTPRIME"

break

if(res=="PRIME"):

print("\t{}".format(num))

====================================

String Handling Part-2

====================================

=>On String Data, we can perform Indexing, Slicing Operations and with these operations, we can also perform different type of operations by using pre-defined functions present in str object.

--------------------------------------------------------------------------------------------------------------------------------

Pre-defined Functions in str object

--------------------------------------------------------------------------------------------------------------------------------

1) capitalize ()

--------------------------------------------------------------------------------------------------------------------------------

=>This Function is used for capitalizing the first letter of First word of a given Sentence only.

=>Syntax: strobj.capitalize()

(OR)

strobj=strobj.capitalize()

-----------------

Examples:

-----------------

>>> s="python"

>>> print(s,type(s))-------------------python <class 'str'>

>>> s.capitalize()--------------------'Python'

>>> s="python is an oop lang"

>>> print(s,type(s))-------------------------python is an oop lang <class 'str'>

>>> s.capitalize()-----------------------------'Python is an oop lang'

-------------------------------------

>>> s="python"

>>> print(s,type(s))--------------------python <class 'str'>

>>> s.capitalize()--------------------'Python'

>>> print(s,type(s))----------------python <class 'str'>

>>> s=s.capitalize()

>>> print(s,type(s))-----------------Python <class 'str'>

--------------------------------------------------------------------------------------------------------------------------------

2) title ():

--------------------------------------------------------------------------------------------------------------------------------

=>This is used for obtaining Title Case of a Given Sentence (OR) Making all words First

Letters are capital.

Syntax: s.title()

(OR)

s=s.title()

------------------

Examples:

------------------

>>> s="python"e

>>> print(s,type(s))-------------------python <class 'str'>

>>> s.capitalize()---------------------'Python'

>>> s.title()-----------------------------'Python'

----------------------------------------------------------

>>> s="python is an oop lang"

>>> print(s,type(s))------------------python is an oop lang <class 'str'>

>>> s.capitalize()--------------------'Python is an oop lang'

>>> s.title()----------------------------'Python Is An Oop Lang'

>>> print(s)----------------------------python is an oop lang

>>> s=s.title()

>>> print(s)--------------------------Python Is An Oop Lang

--------------------------------------------------------------------------------------------------------------------------------

3) index()

--------------------------------------------------------------------------------------------------------------------------------

=>This Function obtains Index of the specified Value

=>If the specified value does not exist then we get ValueError

=>Syntax: strobj.index(Value)

=>Syntax: indexvalue=strobj.index(value)

Examples:

-----------------

>>> s="python"

>>> s.index("p")------------------0

>>> s.index("y")-------------------1

>>> s.index("o")-----------------4

>>> s.index("n")----------------5

>>> s.index("K")----------------ValueError: substring not found

=>enumerate () is one the general function, which is used for finding Index and Value of an Iterable object.

NOTE:

------------

>>> for i,v in enumerate(s):

... print("Index:{} and Value:{}".format(i,v))

---------------

OUTPUT

---------------

Index:0 and Value:p

Index:1 and Value:y

Index:2 and Value:t

Index:3 and Value:h

Index:4 and Value:o

Index:5 and Value:n

-----------------------------

>>> lst=[10,"Rossum",23.45,True]

>>> for i,v in enumerate(lst):

... print("Index: {} and Value:{}".format(i,v))

---------------

OUTPUT

---------------

Index:0 and Value:10

Index:1 and Value: Rossum

Index:2 and Value:23.45

Index:3 and Value: True

--------------------------------------------------------------------------------------------------------------------------------

4) upper()

--------------------------------------------------------------------------------------------------------------------------------

=>It is used for converting any type of Str Data into Upper Case.

=>Syntax:- strobj.upper()

OR

strobj=strobj.upper()

-----------------

Examples:

=---------------

>>> s="python"

>>> print(s)------------------------------python

>>> s.upper()-----------------------'PYTHON'

>>> s="python is an oop lang"

>>> print(s)---------------------------------python is an oop lang

>>> s.upper()--------------------------------'PYTHON IS AN OOP LANG'

>>> s="Python IS an OOP lang"

>>> print(s)-------------------------------Python IS an OOP lang

>>> s.upper()--------------------------'PYTHON IS AN OOP LANG'

>>> s="AbCdEf"

>>> print(s)------------------------AbCdEf

>>> s.upper()----------------------'ABCDEF'

>>> s="PYTHON"

>>> print(s)--------------------PYTHON

>>> s.upper()-----------------'PYTHON'

>>> s="123"

>>> print(s)------------------123

>>> s.upper()----------------'123'

--------------------------------------------------------------------------------------------------------------------------------

5) lower ()

--------------------------------------------------------------------------------------------------------------------------------

=>It is used for converting any type of Str Data into lower Case.

=>Syntax: - strobj.lower()

OR

strobj=strobj.lower()

Examples:

-----------------

>>> s="Data Science"

>>> print(s)--------------Data Science

>>> s.lower()------------'data science'

>>> s="python"

>>> print(s)-------------python

>>> s.lower()-----------'python'

>>> s="PYTHON"

>>> print(s)-------------PYTHON

>>> s.lower()------------'python'

>>> s="PYThon"

>>> print(s)----------PYThon

>>> s.lower()---------'python'

--------------------------------------------------------------------------------------------------------------------------------

6) isupper()

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided the given str object data is purely Upper Case otherwise it returns False.

Syntax: strobj.isupper()

Examples:

-----------------

>>> s="PYTHON"

>>> s.isupper()-----------True

>>> s="python"

>>> s.isupper()----------False

>>> s="Python"

>>> s.isupper()----------False

>>> s="PYThon"

>>> s.isupper()----------False

>>> s="123"

>>> s.isupper()------------False

>>> s="%$#^&@"

>>> s.isupper()-----------False

--------------------------------------------------------------------------------------------------------------------------------

7)islower()

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided the given str object data is purely lower Case otherwise it returns False.

Syntax: strobj.islower()

-----------------

Examples:

-----------------

>>> s="pythopn"

>>> s.islower()------------True

>>> s="pythOn"

>>> s.islower()------------False

>>> s="PYTHON"

>>> s.islower()-----------False

>>> s="123"

>>> s.islower()----------False

--------------------------------------------------------------------------------------------------------------------------------

8) isalpha()

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided str object contains Purely Alphabets otherwise returns False.

Syntax: strobj.isalpha()

-------------------

Examples:

-------------------

>>> s="Ambition"

>>> s.isalpha()--------------------True

>>> s="Ambition123"

>>> s.isalpha()-------------------False

>>> s="1234"

>>> s.isalpha()------------------False

>>> s=" "

>>> s.isalpha()------------------False

>>> s="#$%^@"

>>> s.isalpha()-----------------False

>>> s="AaBbZz"

>>> s.isalpha()----------------True

--------------------------------------------------------------------------------------------------------------------------------

9) isdigit()

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided given str object contains purely digits otherwise returns False

Examples:

--------------------

>>> s="python"

>>> s.isdigit()------------------False

>>> s="python123"

>>> s.isdigit()----------------False

>>> s="123"

>>> s.isdigit()-----------------True

>>> s="123 456"

>>> s.isdigit()---------------False

>>> s="1\_2\_3"

>>> s.isdigit()---------------False

>>> s="123KV"

>>> s.isdigit()-------------False

--------------------------------------------------------------------------------------------------------------------------------

10) isalnum()

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided str object contains either Alpabets OR Numerics or Alpha-Numerics only otherwise It returns False.

=>Syntax: strobj. isalphanum()

---------------------------

=>Examples:

---------------------------

>>> s="python310"

>>> s.isalnum()-----------------True

>>> s="python"

>>> s.isalnum()-----------------True

>>> s="310"

>>> s.isalnum()-----------------True

>>> s="$python310"

>>> s.isalnum()-----------------False

>>> s="python 310"

>>> s.isalnum()----------------False

>>> s="$python3.10"

>>> s.isalnum()----------------False

>>> s="python3.10"

>>> s.isalnum()-------------False

-------------------------------------------------------------------------------------------------------------------------------

11) isspace()

-------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided str obj contains purely space otherwise it returns False.

=>Syntax: strobj.isspace()

------------------------

Examples:

----------------------

>>> s=" "

>>> s.isspace()-----------True

>>> s=""

>>> s.isspace()--------------False

>>> s="python Prog"

>>> s.isspace()-------------False

>>> s="Prasana Laxmi"

>>> s.isspace()--------------False

>>> s.isalpha()-----------False

>>> s.isalpha() or s.isspace()-----------False

-------------------------------------------------------------------------------------------------------------------------------

12) split()

-------------------------------------------------------------------------------------------------------------------------------

=>This Function is used for splitting the given str object data into different words base specified delimter ( - \_ # % ^ ^ , ; ....etc)

=>The dafeult deleimter is space

=>The Function returns Splitting data in the form of list object

=>Syntax: strobj.split("Delimter")

(OR)

strobj.split()

(OR)

listobj= strobj.split("Delimter")

(OR)

listobj=strobj.split()

----------------

Examples:

----------------

>>> s="Python is an oop lang"

>>> print(s)----------------Python is an oop lang

>>> s.split()----------------['Python', 'is', 'an', 'oop', 'lang']

>>> len(s.split())-----------5

>>> x=s.split()

>>> print(x,type(x))---------['Python', 'is', 'an', 'oop', 'lang'] <class 'list'>

>>> len(x)---------------5

>>> s="12-09-2022"

>>> print(s)-------------12-09-2022

>>> s.split("-")----------['12', '09', '2022']

>>> s="12-09-2022"

>>> dob=s.split("-")

>>> print(dob,type(dob))------------['12', '09', '2022'] <class 'list'>

>>> print("Day",dob[0])----------Day 12

>>> print("Month ",dob[1])---------Month 09

>>> print("Year ",dob[2])----------Year 2022

---------------------------------------------------------

>>> s="Apple#Banana#kiwi/Guava"

>>> words=s.split("#")

>>> print(words)-----------['Apple', 'Banana', 'kiwi/Guava']

>>> words=s.split("/")

>>> print(words)------------------['Apple#Banana#kiwi', 'Guava']

-------------------------------------------------------------------------------

13) join():

-------------------------------------------------------------------------------

=>This Function is used for combining or joining list of values from any Iterable object

=>Syntax: strobj.join(Iterableobject)

Examples:

------------------------------

>>> lst=["HYD","BANG","AP","DELHI"]

>>> print(lst,type(lst))------------------['HYD', 'BANG', 'AP', 'DELHI'] <class 'list'>

>>> s=""

>>> s.join(lst)---------------'HYDBANGAPDELHI'

>>> s=" "

>>> s.join(lst)------------------'HYD BANG AP DELHI'

-------------------------------------------------------------------

>>> t=("Rossum","is", "Father" "of" ,"Python")

>>> print(t,type(t))

('Rossum', 'is', 'Fatherof', 'Python') <class 'tuple'>

>>> k=" "

>>> k.join(t)

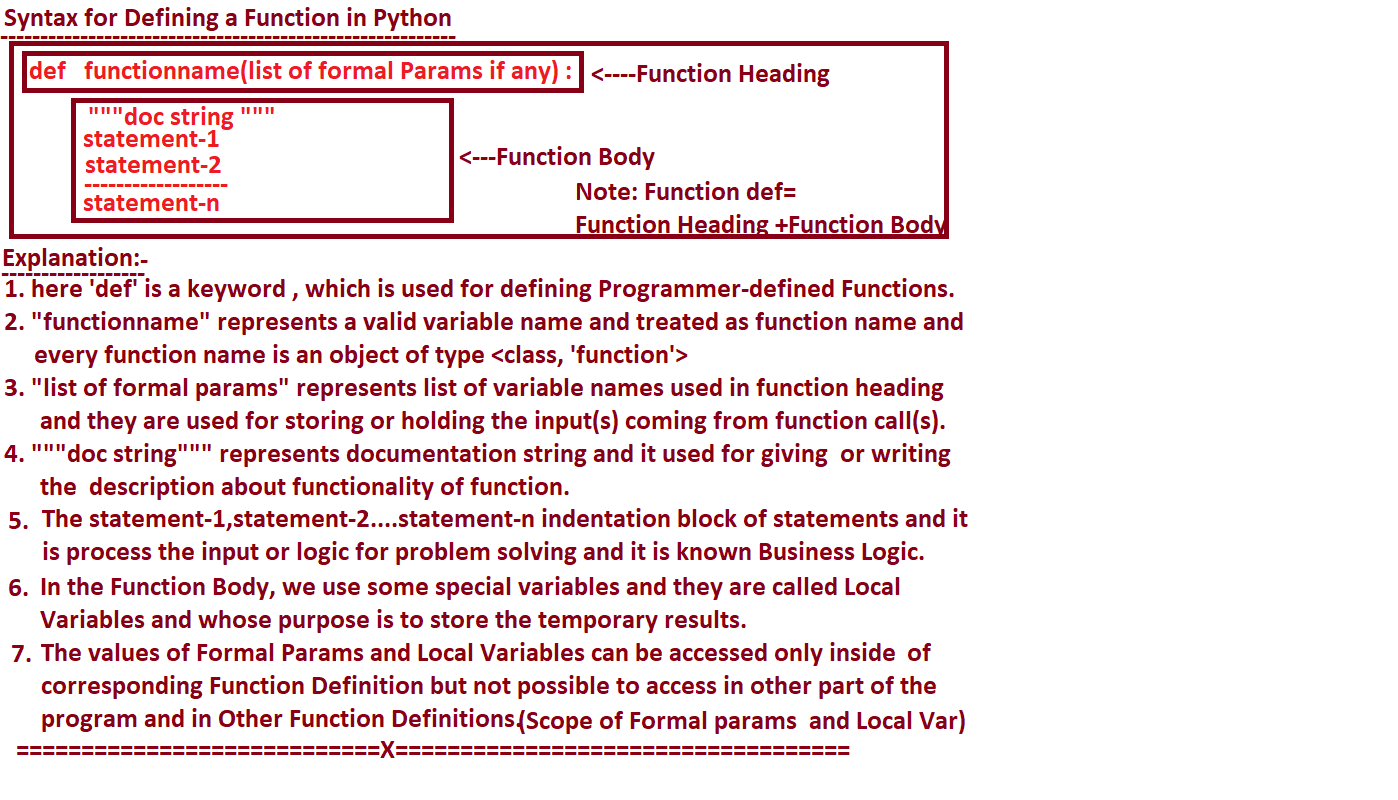
'Rossum is Fatherof Python'

>>> t=("Rossum","is", "Father", "of" ,"Python")

>>> k=" "

>>> k.join(t)

'Rossum is Father of Python'



=================================================

Functions in Python

===================================================

Index

----------

=>Purpose of Functions

=>Advantages of Functions

=>Definition of Functions

=>Parts of Functions

=>Phases in Functions

=>Syntax for Defining Functions

=>Number of Approaches for Defining Functions

=>Programming Examples

-----------------------------------------------------------------------

=>Arguments and Parameters

=>Types of Arguments and Parameters

a) Positional Arguments and Parameters

b) Default Arguments and Parameters

c) Keyword Arguments and Parameters

d) Variable Length Arguments and Parameters

e) Keyword Variable Length Arguments and Parameters

=>Programming Examples

----------------------------------------------------------------------

=>Global Variables and Local Variables

=>global keyword and global ()

=>Programming Examples

=>Anonymous Functions OR Lambda Functions

=>Programming Examples

-----------------------------------------------------------------------

=>Special Function in Python

a) filter () with Normal and Anonymous Functions

b) map () with Normal and Anonymous Functions

c) reduce () with Normal and Anonymous Functions

=>Programming Examples

=================================================

Types of Languages in the context of Functions

=================================================

=>In IT, we have two types of Programming Languages. They are

1. Un-Structured Programming Languages

2. Structured Programming Languages

--------------------------------------------------------------------------------------------------------------------------------

1. Un-Structured Programming Languages

--------------------------------------------------------------------------------------------------------------------------------

=>Un-Structured Programming Languages does not contain the concept of Functions and

hence whose applications having the following Limitations.

1. Application Development time is More

2. Application Memory Space is More

3. Application Execution Time is More

4. Application Performance is degraded

5. Redundancy of the code is More

Examples: GW-BASIC

--------------------------------------------------------------------------------------------------------------------------------

2. Structured Programming Languages

--------------------------------------------------------------------------------------------------------------------------------

=>Structured Programming Languages contains the concept of Functions and

hence whose applications having the following Advantages.

1. Application Development time is Less

2. Application Memory Space is Less

3. Application Execition Time is Less

4. Application Performance is Enhanced (Improved)

5. Redundency of the code is Minimized

Examples: C, C++, Java, PYTHON.....etc

-----------------------------------------------------------------------

===================================================

Functions in Python

===================================================

=>The Purpose of Functions is that " To Perform Certain Operation and provides Code Re-Usability ".

--------------------------------------

=>Definition of Function:

--------------------------------------

=>A part of main program is called Function

(OR)

=>Sub Program of Main Program is called Function.

--------------------------------------------------------------------------

Parts of Functions

-------------------------------------------------------------------------

=>When we are dealing with Functions, we must ensure that, there must exist Two parts. They are

1. Function Definition

2. Function Call(s)

=>Here Particular Function Definition Exist Only Once

=>For Every Function Call There Must Exist Function Definition otherwise we get NameError.

=> Function Definition can’t Execute by itself but ther are executed through a function call. In Otherwords Function Definition will be executed when we call the Function by using Function Call.

---------------------------------------------------------------------------------------

Phases In Functions

---------------------------------------------------------------------------------------

=>At the time Defining the functions, we must ensure that there must exist 3 Phases.

1. Every Function must take INPUT

2. Every Function must PROCESS the input

3. Every Function must give RESULT / OUTPUT

---------------------------------------------------------------------------------------

=====================================================

Number of Approaches to Define a Function

=====================================================

=>If any Problem Statement is given then it be solved by using Function in 4 Approaches. They are

----------------------------------------------------------------------------------------------------------------------

#Approach1:

----------------------------------------------------------------------------------------------------------------------

=>INPUT Taking From Function Call

=>PROCESSING done in Function Body

=>OUTPUT / RESULT giving to Function Call

------------------

Examples:

------------------

#Program for defining a function for addition of two numbers

#ApproachEx1.py

def addop(a,b): # Here 'a' and 'b' are called Formal Parameters

c=a+b # Here 'c' is called Local variable

return c # here return statement is used for giving the result back

#main program

x=float(input("Enter First Value:"))

y=float(input("Enter Second Value:"))

res=addop(x,y) # Function Call

print("sum({},{})={}".format(x,y,res))

--------------------------------------------------------------------------------------------------------------------------------

#Approach2:

--------------------------------------------------------------------------------------------------------------------------------

=>INPUT Taking in Function Body

=>PROCESSING done in Function Body

=>OUTPUT / RESULT displaying in Function Body

Examples:

-----------------

#Program for defining a function for addition of two numbers

#ApproachEx2.py

def addop():

a=float (input("Enter First Value:"))

b=float (input("Enter Second Value:"))

c=a+b

print("sum({},{})={}".format(a,b,c))

#main program

addop() # Function Call

--------------------------------------------------------------------------------------------------------------------------------

#Approach3:

--------------------------------------------------------------------------------------------------------------------------------

=>INPUT Taking in Function Body

=>PROCESSING done in Function Body

=>OUTPUT / RESULT giving to Function Call

#Program for defining a function for addition of two numbers

#ApproachEx3.py

def addop():

a=float(input("Enter First Value:"))

b=float(input("Enter Second Value:"))

c=a+b

return a,b,c # In Python, return stmt can return one or more values

#main program

a,b,c=addop() # Function Call with multi line assigment

print("sum({},{})={}".format(a,b,c))

print("----------------------------------------------------------")

kvr=addop() # Here kvr is an object of type tuple.

print("sum({},{})={}".format(kvr[0],kvr[1],kvr[2]))

--------------------------------------------------------------------------------------------------------------------------------

#Approach4:

--------------------------------------------------------------------------------------------------------------------------------

=>INPUT Taking from Function Call

=>PROCESSING done in Function Body

=>OUTPUT / RESULT displaying in Function Body

#Program for defining a function for addition of two numbers

#ApproachEx4.py

def addop(a,b):

c=a+b

print("sum({},{})={}".format(a,b,c))

#main program

a=float(input("Enter First Value:"))

b=float(input("Enter Second Value:"))

addop(a,b) # Function Call

-------------------------------------------------------------------------------------------------------------------------------

#Program for defining a function for addition of two numbers

#Approach1.py

def addop(a,b): # Here 'a' and 'b' are called Formal Parameters

print("Line-5: i am from addop()")

c=a+b # Here 'c' is called Local variable

return c # here return statement is used for giving the result back

#main program

print("Line-10: I am from Main Program:")

res=addop(10,20) # Function Call

print("Line-12--sum=",res)

print("\nLine-13: I am from Main Program:")

res=addop(100,200) # Function Call

print("Line-15--sum=",res)

print("\nLine-16: I am from Main Program:")

res=addop(-10,-20) # Function Call

print("Line-18--sum=",res)

#Program for defining a function for addition of two numbers

#ApproachEx1.py

def addop(a,b): # Here 'a' and 'b' are called Formal Parameters

c=a+b # Here 'c' is called Local variable

return c # here return statement is used for giving the result back

#main program

x=float(input("Enter First Value:"))

y=float(input("Enter Second Value:"))

res=addop(x,y) # Function Call

print("sum({},{})={}".format(x,y,res))

#Program for defining a function for addition of two numbers

#ApproachEx2.py

def addop():

a=float(input("Enter First Value:"))

b=float(input("Enter Second Value:"))

c=a+b

print("sum({},{})={}".format(a,b,c))

#main program

addop() # Function Call

#Program for defining a function for addition of two numbers

#ApproachEx3.py

def addop():

a=float(input("Enter First Value:"))

b=float(input("Enter Second Value:"))

c=a+b

return a,b,c # In Python, return stmt can return one or more values

#main program

a,b,c=addop() # Function Call with multi line assigment

print("sum({}, {})={}".format(a,b,c))

print("----------------------------------------------------------")

kvr=addop() # Here kvr is an object of type tuple.

print("sum ({}, {})={}".format(kvr[0],kvr[1],kvr[2]))

#Program for defining a function for addition of two numbers

#ApproachEx4.py

def addop(a,b):

c=a+b

print("sum({}, {})={}".format(a,b,c))

#main program

a=float(input("Enter First Value:"))

b=float(input("Enter Second Value:"))

addop(a,b) # Function Call

#Program generating Mul Table for a give number by using Functions.

#MulTable.py

def table(n):

if(n<=0):

print ("{} is invalid input".format(n))

else:

print("-"\*50)

print("Mu Table for {}".format(n))

print("-"\*50)

for i in range(1,11):

print("\t{} x {} = {}".format(n,i,n\*i))

else:

print("-"\*50)

#main program

n=int(input("Enter a value:"))

table(n) # Function call

#rectareaperi.py

def area():

l=float (input ("Enter Length for cal area of Rect:"))

b=float (input ("Enter Breadth for cal area of Rect:"))

area=l\*b

print ("Area of Rect=“, area)

def peri():

l=float(input("Enter Length for cal peri of Rect:"))

b=float(input("Enter Breadth for cal peri of Rect:"))

peri=2\*(l+b)

print("Perimter of Rect=",peri)

#main program

peri()

area()

#write a python program which will accept list of values and find sum and average.

# HINT l1= [10,20,30,40]

# OUTPUT : sum=100 and AVG=25

#SumAvg.py

def readvalues():

lst=list() # create an empy list

n=int(input("Enter How Many values u want fint sum and Avg:"))

if(n<=0):

return lst # returning empty list

else:

for i in range(1,n+1):

val=float(input("Enter {} Value:".format(i)))

lst.append(val)

return lst # returning non-empty list

def sumavg(lst): # By using Logic

print("-"\*50)

print("List of Values:")

print("-"\*50)

for val in lst:

print("\t{}".format(val))

print("-"\*50)

#Find sum and avg

s=0

for val in lst:

s=s+val

else:

print("-"\*50)

print("Sum=“, s)

print("Avg=",s/len(lst))

print("-"\*50)

#Main program

lstobj=readvalues () # Function Call

if(len(lstobj)==0):

print ("List is Empty-- and can't find sum and Avg")

else:

sumavg(lstobj) # Function Call

#write a python program which will accept list of values and find sum and average.

# HINT l1=[10,20,30,40]

# OUTPUT : sum=100 and AVG=25

#SumAvg1.py

def readvalues():

lst=list () # create an empy list

n=int(input("Enter How Many values u want fint sum and Avg:"))

if(n<=0):

return lst # returning empty list

else:

for i in range (1, n+1):

val=float (input ("Enter {} Value:".format(i)))

lst.append(val)

return lst # returning non-empty list

def sumavg(lst): # By using pre-defined sum ()

print("Sum ({})={}".format(lst, sum(lst)) )

print ("Avg= {}”. format(sum(lst)/len(lst)) )

#main program

lstobj=readvalues() # Function Call

if(len(lstobj)==0):

print ("List is Empty-- and can't find sum and Avg")

else:

sumavg(lstobj) # Function Call

write a python program which will accept list of numerical values and sort them in ascending and descending order.

#SortData.py

def readvalues():

lst=list () # create an empty list

n=int (input ("Enter How Many values u want Sort:"))

if(n<=0):

return lst # returning empty list

else:

for i in range(1,n+1):

val=float(input("Enter {} Value:".format(i)))

lst.append(val)

return lst # returning non-empty list

def sortdata(lst):

print("Orginal Data={}".format(lst))

lst.sort()

print("ASC Order :{}".format(lst))

lst.sort(reverse=True)

print("DESC Order :{}".format(lst))

#main program

listobj= readvalues()

if(len(listobj)==0):

print("List is Empty-- and can't sort")

else:

sortdata(listobj)

# Program for displaying the values of Iterable Objects by using Functions

#IterableObjects.py

def disp(kvr):

print ("-"\*50)

print ("Type of kvr=“, type(kvr))

for val in kvr:

print("\t {}”. format(val))

print ("-"\*50)

def dispdictvalues(kvr):

print("-"\*50)

print("Type of kvr=",type(kvr))

for key,val in kvr.items():

print("\t{}-->{}".format(key,val))

print("-"\*50)

#main program

lst=[10,"Rossum",34.56,True,2-3.4j]

disp(lst) # Function Call

tpl=(10,20,30,40,23.45)

disp(tpl) # Function Call

s={10,"Rajesh",45.67}

disp(s) # Function Call

d1={10:"Rossum",20:"Travis",30:"Kinney"}

dispdictvalues(d1)# Function Call

#write a Python program which will accept list of values and find highest element and lowest element

#MaxMinEx1.py

def readvalues():

lst=list () # create an empty list

n=int (input("Enter How Many values u want find max and min:"))

if(n<=0):

return lst # returning empty list

else:

for i in range(1,n+1):

val=float (input("Enter {} Value:".format(i)))

lst.append(val)

return lst # returning non-empty list

def kvrmax(lst):

#Logic for max

maxv=lst[0]

for i in range(1,len(lst)):

if (lst[i] > maxv):

maxv=lst[i]

return maxv

def kvrmin(lst):

print("-"\*50)

#Logic for min

minv=lst[0]

for i in range(1,len(lst)):

if (lst[i] < minv):

minv=lst[i]

return minv

#main program

listobj=readvalues()

if(len(listobj)==0):

print("List is Empty-- and can't find max and min")

else:

if(len(set(listobj))==1):

print("\nMax({})={}".format(listobj,"ALL VALUES ARE EQUAL"))

print("\nMin({})={}".format(listobj,"ALL VALUES ARE EQUAL"))

else:

mv=kvrmax(listobj) # Function Call

print("\nMax({})={}".format(listobj,mv))

minv=kvrmin(listobj) # Function Call

print("\nMIn({})={}".format(listobj,minv))

==============================================

Arguments and Parameters

==============================================

=>Parameters represents list of Variables used in Function Heading and they are used for Storing the inputs coming Functions Calls and these Parameters are called Formal Params Meters.

=>Arguments are the Variables used inside of Function Call(s) and they are also called Actual Arguments.

Syntax for Function Definition

--------------------------------------------------------------------

def function name (param1, param2...param-n)

--------------------------------------------------------------------

-----------------------------------------------

Syntax for Function Call

----------------------------------------------

Function name(args1, args2.......args-n)

=>Hence relationship between arguments and Parameters is that Every Value of arguments are passing Parameters.

===========================================

Types of Arguments and Parameters

===========================================

=>Based on Passing the values of Arguments from Function Calls to Parameters of Function Definition, Arguments and Parameters are classified into 5 types. They are

1. Positional Arguments OR Parameters

2. Default Arguments OR Parameters

3. Keyword Arguments OR Parameters

4. Variable Length Arguments OR Parameters

5. Keyword Variable Length Arguments OR Parameters

=========================================

1) Positional Arguments (or) Parameters

=========================================

=>The Concept of Positional Parameters (or) arguments says that "The Number of Arguments (Actual arguments) must be equal to the number of formal parameters".

=>This Parameter mechanism also recommends Order and Meaning of Parameters for Higher accuracy.

=>To pass the Specific Data from Function Call to Function Definition then we must take Positional Argument Mechanism.

=>The default Argument Passing Mechanism is Positional Arguments (or) Parameters.

-----------------------------------------------

Syntax for Function Definition:

-----------------------------------------------

def function name (parm1, param2......param-n):

-------------------------------------------------

-------------------------------------------------

-----------------------------------------------

Syntax for Function Call:

-----------------------------------------------

Function name(arg1,arg2....arg-n)

=>Here the values of arg1, arg2...arg-n are passing to param-1,param-2..param-n respectively.

=>PVM gives First Priority for Positional Arguments (or) Parameters

#Program for demonstarting Possitional Arguments OR Parameters

#PossArgsEx1.py

def dispstudinfo(sno,sname,marks):

print("\t{}\t{}\t{}".format(sno,sname,marks))

#main program

print("-"\*50)

print("\tSno\tName\tMarks")

print("-"\*50)

dispstudinfo(10,"RS",34.56)

dispstudinfo(20,"TR",54.56)

dispstudinfo(30,"RJ",14.56)

print("-"\*50)

#Program for demonstarting Possitional Arguments OR Parameters

#PossArgsEx2.py

def dispstudinfo(sno,sname,marks,crs):

print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))

#main program

print("-"\*50)

print("\tSno\tName\tMarks\tCourse")

print("-"\*50)

dispstudinfo(10,"RS",34.56,"PYTHON")

dispstudinfo(20,"TR",54.56,"PYTHON")

dispstudinfo(30,"RJ",14.56,"PYTHON")

dispstudinfo(40,"NN",24.56,"PYTHON")

dispstudinfo(50,"MM",34.56,"PYTHON")

print("-"\*50)

======================================

2) Default Parameters (or) arguments

======================================

x=>When there is a Common Value for family of Function Calls then Such type of Common Value(s) must be taken as default parameter with common value (But not recommended to pass by using Posstional Parameters)

Syntax: for Function Definition with Default Parameters

----------------------------------------------------------------------------------------

def functionname(param1,param2,....param-n-1=Val1, Param-n=Val2):

------------------------------------------------------------------

Here param-n-1 and param-n are called "default Parameters"

and param1, param-2... are called "Possitional parameters"

Rule-: When we use default parameters in the function definition, They must be used as last Parameter(s) otherwise we get Error(SyntaxError: non-default argument (Possitional ) follows default argument).

#Program for demonstrating Default Arguments OR Parameters

#DefArgsEx1.py

def dispstudinfo(sno,sname,marks,crs="PYTHON"):

print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))

#main program

print("-"\*50)

print("\tSno\tName\tMarks\tCourse")

print("-"\*50)

dispstudinfo(10,"RS",34.56)

dispstudinfo(20,"TR",54.56)

dispstudinfo(30,"RJ",14.56)

dispstudinfo(40,"NN",24.56)

dispstudinfo(50,"MM",34.56)

dispstudinfo(60,"KV",14.56,"JAVA")

dispstudinfo(70,"AV",54.56)

print("-"\*50)

#Program for demonstrating Default Arguments OR Parameters

#DefArgsEx2.py

def dispstudinfo(sno,sname,marks,crs="PYTHON",cnt="INDIA"):

print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,cnt))

#main program

print("-"\*50)

print("\tSno\tName\tMarks\tCourse\tCountry")

print("-"\*50)

dispstudinfo(10,"RS",34.56)

dispstudinfo(20,"TR",54.56)

dispstudinfo(30,"RJ",14.56)

dispstudinfo(40,"NN",24.56)

dispstudinfo(50,"MM",34.56)

dispstudinfo(60,"KV",14.56,"JAVA")

dispstudinfo(70,"AV",54.56)

dispstudinfo(80,"DT",4.56,"PHP","USA")

dispstudinfo(90,"JB",64.56,cnt="USA")

dispstudinfo(85,"AF",34.56,"PYTHON,JAVA","UK")

print("-"\*50)

#Program for cal area of circle using Functions

#DefArgsEx4.py

def areacircle(r,PI=3.14):

area=PI\*r\*\*2

print("Area of Circle=",area)

#Main program

areacircle(1.2)

areacircle(2.2)

areacircle(2)

areacircle(4.2)

areacircle(float(input("Enter Radious:")))

============================================

3) Keyword Parameters (or) arguments

============================================

=>In some of the circumstances, we know the function name and formal parameter names and we don't know the order of formal Parameter names and to pass the data / values accurately we must use the concept of Keyword Parameters (or) arguments.

=>The implementation of Keyword Parameters (or) arguments says that all the formal parameter names used as arguments in Function call(s) as keys.

Syntax for function definition: -

-------------------------------------------------

def functionname(param1,param2...param-n):

---------------------------------------------

---------------------------------------------

Syntax for function call:-

-------------------------------------------------

functionname(param-n=val-n,param1=val1,param-n-1=val-n-1,.........)

Here param-n=val-n,param1=val1,param-n-1=val-n-1,...... are called Keywords arguments

==========================X============================================

#Program for demonstarting Keyword args

#KwdArgsEx1.py

def disp(a,b,c):

print("\t{}\t{}\t{}".format(a,b,c))

#main program

print("-"\*50)

print("\tA\tB\tC")

print("-"\*50)

disp(10,20,30) # Function Call--Possitional args

disp(c=30,b=20,a=10) # Function Call--keyword args

disp(b=20,c=30,a=10) # Function Call--keyword args

disp(10,c=30,b=20)# Function Call--Possitional keyword args

disp(10,20,c=30)# Function Call--Possitional keyword args

#disp(b=20,10,c=30) # SyntaxError: positional argument follows keyword argument

print("-"\*50)

#Program for demonstarting Keyword args

#KwdArgsEx2.py

def disp(a,b,c,E=2.71):

print("\t{}\t{}\t{}\t{}".format(a,b,c,E))

#main program

print("-"\*50)

print("\tA\tB\tC\tE")

print("-"\*50)

disp(10,20,30) # Function Call--Possitional args

disp(c=30,b=20,a10) # Function Call--keyword args

disp(b=20,c=30,a=10) # Function Call--keyword args

disp(10,c=30,b=20)# Function Call--Possitional keyword args

disp(10,20,c=30)# Function Call--Possitional keyword args

#disp(b=20,10,c=30) # SyntaxError: positional argument follows keyword argument

disp(c=30,E=3.41,a=10,b=20)# Function Call--Possitional ,keyword args, default

print("-"\*50)

================================================

4) Variables Length Parameters (or) arguments

================================================

=>When we have familiy of multiple function calls with Variable number of values / arguments then with normal python programming, we must define mutiple function defintions. This process leads to more development time. To overcome this process, we must use the concept of Variable length Parameters.

=>To Impelement, Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called astrisk (\* param) and the formal parameter with astrisk symbol is called Variable length Parameters and whose purpose is to hold / store any number of values coming from similar function calls and whose type is <class, 'tuple'>.

-----------------------------------------------------------------------

Syntax for function definition with Variables Length Parameters:

-----------------------------------------------------------------------

def functionname(list of formal params, \*param1,param2=value)

--------------------------------------------------

=>Here \*param is called Variable Length parameter and it can hold any number of argument values (or) variable number of argument values and \*param type is <class,'tuple'>

=>Rule: - The \*param must always written at last part of Function Heading and it must be only one (but not multiple)

=>Rule: - When we use Variable length and default parameters in function Heading, we use default parameter as last and before we use variable length parameter and in function calls, we should not use default parameter as Key word argument bcoz Variable number of values are treated as Posstional Argument Value(s) .

#Program for demonstrating Var arguments concept

#VarAgrsEx1.py-----This Program will not execute as it is

def disp(a):

print(a)

def disp(a,b):

print("{}\t{}".format(a,b))

def disp(a,b,c):

print("{}\t{}\t{}".format(a,b,c))

def disp(a,b,c,d):

print("{}\t{}\t{}\t{}".format(a,b,c,d))

#main program

disp(10) # Function call

disp(10,20) # Function Call

disp(10,20,30) # Function Call

disp(10,20,30,40) # Function Call

#Program for demonstrating Var arguments concept

#VarAgrsEx2.py-----This Program will execute as it is

def disp(a):

print(a)

disp(10) # Function call

def disp(a,b):

print("{}\t{}".format(a,b))

disp(10,20) # Function Call

def disp(a,b,c):

print("{}\t{}\t{}".format(a,b,c))

disp(10,20,30) # Function Call

def disp(a,b,c,d):

print("{}\t{}\t{}\t{}".format(a,b,c,d))

disp(10,20,30,40) # Function Call

#Program for demonstrating Var arguments concept

#PureVarAgrsEx1.py

def disp( \*kvr) : # \*kvr is called Variable Length Param--Holds variable number of values

print(kvr,type(kvr),len(kvr))

#main program

disp(10) # Function call

disp(10,20) # Function Call

disp(10,20,30) # Function Call

disp(10,20,30,40) # Function Call

disp()

disp(10,"RS","Python","DS",34.56,True)

#Program for demonstrating Var arguments concept

#PureVarAgrsEx2.py

def disp( \*kvr) : # \*kvr is called Variable Length Param--Holds variable number of values

print("-"\*40)

print("WAY-1:Type of kvr={} and Number of values={}".format(type(kvr),len(kvr)))

print("-"\*40)

for val in kvr:

print("\t{}".format(val))

print("-"\*40)

print("WAY-2:Type of kvr={} and Number of values={}".format(type(kvr),len(kvr)))

print("-"\*40)

for i in range(0,len(kvr)):

print("\t{}".format(kvr[i]))

print("-"\*40)

#main program

disp(10) # Function call

disp(10,20) # Function Call

disp(10,20,30) # Function Call

disp(10,20,30,40) # Function Call

disp()

disp(10,"RS","Python","DS",34.56,True)

#Write a python program which will calculate sum and average of variable number of values by various person

#PureVarAgrsEx2.py

def findsumavg(sname,\*k):

print("-"\*50)

print("Name of Student:{}".format(sname))

print("{}, Having {} Values:".format(sname, len(k)))

print("-"\*50)

s=0

for v in k:

print("\t{}".format(v),end=" ")

s=s+v

else:

print()

print("-"\*50)

if(len(k)!=0):

print("Sum={}".format(s))

print("Avg={}".format(s/len(k)))

else:

print("Sum={}".format(s))

print("-"\*50)

#main program

findsumavg("Naveen",10,20,30,40)

findsumavg("Muzamil",100,200,300)

findsumavg("Sarat",1.2,3.4,4.5,5.6,7.8)

findsumavg("Rossum",10,20.5)

findsumavg("KVR")

#Write a python program which will calculate sum and average of variable number of values by various person

#PureVarAgrsEx4.py

def findsumavg(sname,\*k,cnt="HYD"):

print("-"\*50)

print("Name of Student:{} and living in {}".format(sname,cnt))

print("{}, Having {} Values:".format(sname, len(k)))

print("-"\*50)

s=0

for v in k:

print("\t{}".format(v),end=" ")

s=s+v

else:

print()

print("-"\*50)

if(len(k)!=0):

print("Sum={}".format(s))

print("Avg={}".format(s/len(k)))

else:

print("Sum={}".format(s))

print("-"\*50)

#main program

findsumavg("Naveen",10,20,30,40)

findsumavg("Muzamil",100,200,300)

findsumavg("Sarat",1.2,3.4,4.5,5.6,7.8)

findsumavg("Rossum",10,20.5,cnt="Nether Lands")

findsumavg("KVR",cnt="AP")

#findsumavg("TRAVIS",cnt="Finland", 23)--SyntaxError: positional argument follows keyword argument

findsumavg("TRAVIS", 23,cnt="Finland")

#findsumavg(sname="Kinney",2,3,5,6,cnt="USA")--SyntaxError: positional argument follows keyword argument

#findsumavg(2,3,5,6,cnt="USA",sname="Kinney")--TypeError: findsumavg() got multiple values for argument 'sname'

findsumavg("Kinney",2,3,5,6,cnt="USA")

================================================

5) Key Word Variables Length Parameters (or) arguments

================================================

=>When we have familiy of multiple function calls with Key Word Variable number of values / arguments then with normal python programming, we must define mutiple function defintions. This process leads to more development time. To overcome this process, we must use the concept of Keyword Variable length Parameters.

=>To Implement, Keyword Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called double astrisk (\*\* param) and the formal parameter with double astrisk symbol is called Keyword Variable length Parameters and whose purpose is to hold / store any number of (Key,Value) coming from similar function calls and whose type is <class, 'dict'>.

----------------------------------------------------------------------

Syntax for function definition with Keyword Variables Length arameters:

-----------------------------------------------------------------------

def functionname (list of formal params, \*\*param) :

-----------------------------------------------------------------------

=>Here \*\*param is called Keyword Variable Length parameter and it can hold any number of Key word argument values (or) Keyword variable number of argument values and \*\*param type is <class,'dict'>

=>Rule: - The \*\*param must always written at last part of Function Heading and it must be only one (but not multiple)

---------------------------------------------------------------

Final Syntax for defining a Function

---------------------------------------------------------------

def funcname(PosFormal parms, \*Varlenparams, default params, \*\*kwdvarlenparams):

#Program for demonstarting keyword variable length args

#KwdVarArgsEx1.py----This Program will not execute as it is

def dispinfo(a,b,c,d): # Function Def-1

print("\t{}\t{}\t{}\t{}".format(a,b,c,d))

def dispinfo(k,v,r): # Function Def-2

print("\t{}\t{}\t{}".format(k,v,r))

def dispinfo(P,Q): # Function Def-3

print("\t{}\t{}".format(P,Q))

#main program

dispinfo(a=10,b=20,c=30,d=40) # Function Call-1

dispinfo(k=100,v=200,r=300) # Function Call-2

dispinfo(P=1.2,Q=2.3) # Function Call-3

#Program for demonstarting keyword variable length args

#KwdVarArgsEx2.py----This Program will execute as it is

def dispinfo(a,b,c,d): # Function Def-1

print("\t{}\t{}\t{}\t{}".format(a,b,c,d))

dispinfo(a=10,b=20,c=30,d=40) # Function Call-1

def dispinfo(k,v,r): # Function Def-2

print("\t{}\t{}\t{}".format(k,v,r))

dispinfo(k=100,v=200,r=300) # Function Call-2

def dispinfo(P,Q): # Function Def-3

print("\t{}\t{}".format(P,Q))

dispinfo(P=1.2,Q=2.3) # Function Call-3

#Program for demonstarting keyword variable length args

#PureKwdVarArgsEx1.py

def dispinfo( \*\*hyd): # Here \*\*hyd is called Keyword Variable length param---dict

print(hyd,type(hyd), len(hyd))

#main program

dispinfo(a=10,b=20,c=30,d=40) # Function Call-1

dispinfo(k=100,v=200,r=300) # Function Call-2

dispinfo(P=1.2,Q=2.3) # Function Call-3

#Program for demonstarting keyword variable length args

#PureKwdVarArgsEx2.py

def dispinfo( \*\*hyd): # Here \*\*hyd is called Keyword Variable length param---dict

print("-"\*50)

print("Type of hyd=",type(hyd))

print("-"\*50)

for k,v in hyd.items():

print("\t{}--->{}".format(k,v))

print("-"\*50)

#main program

dispinfo(a=10,b=20,c=30,d=40) # Function Call-1

dispinfo(k=100,v=200,r=300) # Function Call-2

dispinfo(P=1.2,Q=2.3) # Function Call-3

dispinfo(P1="Ranjan")

dispinfo()

#write a python program which will compute the Total marks of different subjects who are securing in different classes and studying in various classes.

#PureKwdVarArgsEx3.py

def findtotalmarks(sno,sname,cls,cnt="INDIA",\*\*marks):

print("-"\*50)

print("Roll Number :{}".format(sno))

print("Student Name :{}".format(sname))

print("Class Name :{}".format(cls))

print("Country:{}".format(cnt))

print("-"\*50)

totmarks=0

for sn,sm in marks.items():

print("\t{}\t{}".format(sn,sm))

totmarks=totmarks+sm

else:

print("-"\*50)

print("\tTotal Marks={}".format(totmarks))

print("-"\*50)

#main program

findtotalmarks(10,"Ranjan","X",Eng=60,Hindi=70,Telugu=56,Maths=80,Science=78,Social=67)

findtotalmarks(20,"Minhaj","XII",Mathematics=70,Physics =58,Chemistry=50)

findtotalmarks(30,"Shilpa","B.Tech(CSE)",CM=70,Cpp=60,Python=50,DBMS=50)

findtotalmarks(40,"Rossum","Research")

#write a python program which will compute the Total marks of different subjects who are securing in different classes and studying in various classes.

#PureKwdVarArgsEx4.py

def findtotalmarks(sno,sname,cls,cnt="INDIA",\*\*marks):

print("-"\*50)

print("Roll Number :{}".format(sno))

print("Student Name :{}".format(sname))

print("Class Name :{}".format(cls))

print("Country:{}".format(cnt))

print("-"\*50)

totmarks=0

for sn,sm in marks.items():

print("\t{}\t{}".format(sn,sm))

totmarks=totmarks+sm

else:

print("-"\*50)

print("\tTotal Marks={}".format(totmarks))

print("-"\*50)

#main program

findtotalmarks(10,"Ranjan","X",Eng=60,Hindi=70,Telugu=56,Maths=80,Science=78,Social=67)

findtotalmarks(20,"Minhaj","XII",Mathematics=70,Physics =58,Chemistry=50)

findtotalmarks(30,"Shilpa","B.Tech(CSE)",CM=70,Cpp=60,Python=50,DBMS=50

findtotalmarks(40,"Rossum","Research")

findtotalmarks(cnt="USA",cls="VIII",sno=50,sname="Shruthi",Sub1=20, Sub2=30)

findtotalmarks(cls="V",sno=60,sname="Akshy")

findtotalmarks(cls="VIII",sno=50,sname="Raj",Sub3=70,cnt="UK")

#write a python program which will compute the Total marks of different subjects who are securing in different classes and studying in various classes.

#PureKwdVarArgsEx5.py

def findtotalmarks(sno,sname,cls, \*v, cnt="INDIA",\*\*marks):

print("\*"\*50)

print("Variable Length Values:{}".format(len(v)))

print("\*"\*50)

for val in v:

print("\t{}".format(val))

print("\*"\*50)

print("-"\*50)

print("Roll Number :{}".format(sno))

print("Student Name :{}".format(sname))

print("Class Name :{}".format(cls))

print("Country:{}".format(cnt))

print("-"\*50)

totmarks=0

for sn,sm in marks.items():

print("\t{}\t{}".format(sn,sm))

totmarks=totmarks+sm

else:

print("-"\*50)

print("\tTotal Marks={}".format(totmarks))

print("-"\*50)

#main program

findtotalmarks(10,"Ranjan","X",10,20,30,Eng=60,Hindi=70,Telugu=56,Maths=80,Science=78,Social=67)

findtotalmarks(20,"Minhaj","XII",100,200,Mathematics=70,Physics =58,Chemistry=50)

findtotalmarks(30,"Shilpa","B.Tech(CSE)",1.2,1.3,1.4,1.5,CM=70,Cpp=60,Python=50,DBMS=50)

findtotalmarks(40,"Rossum","Research",15,25)

findtotalmarks(cnt="USA",cls="VIII",sno=50,sname="Shruthi",Sub1=20, Sub2=30)

findtotalmarks(cls="V",sno=60,sname="Akshy")

findtotalmarks(cls="VIII",sno=50,sname="Raj",Sub3=70,cnt="UK")

=====================================================

Global Variables and Local Variables

=====================================================

------------------------

Local Variables

------------------------

=>Local Variables are those which are used in side of Function Body and they are used for storing Temporary result of Processing Logic.

=>We can access the value of Local Variables Inside of Function Body only but no other part of the program.

-------------------------

Global Variables

-------------------------

=>Global variables are those are they are used for accesssing as common values in Multiple Different function Calls

=>Global Variables Must be defined before all Function Calls. so that we can Global Variables values in corresponding Function Definitions.

=>if we define Global Variables after all Function Calls then we can't access Global Variables values in corresponding Function Definitions (NameError we get).

#Program for demonstarting Local anf Global variables

#GlobalLocalVarEx1.py

lang="PYTHON” # Here lang is called Global Variable

def learnDS():

sub1="Data Science"

print ("\tTo Develop '{}' Apps, we use '{}' Programming".format(sub1,lang))

def learnML():

sub2="Machine Learning"

print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub2,lang))

def learnDL():

sub3="Deep Learning"

print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub3,lang))

#main program

learnDS()

learnML()

learnDL()

#Program for demonstarting Local anf Global variables

#GlobalLocalVarEx2.py

def learnDS():

sub1="Data Science"

print("\tTo Develop '{}' Apps, we use '{}' Programming".format(sub1,lang))

lang="PYTHON" # Here lang is called Global Variable

def learnML():

sub2="Machine Learning"

print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub2,lang))

def learnDL():

sub3="Deep Learning"

print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub3,lang))

#main program

learnDS()

learnML()

learnDL()

#Program for demonstarting Local and Global variables

#GlobalLocalVarEx3.py

def learnDS():

sub1="Data Science"

print("\tTo Develop '{}' Apps, we use '{}' Programming".format(sub1,lang))

def learnML():

sub2="Machine Learning"

print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub2,lang))

def learnDL():

sub3="Deep Learning"

print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub3,lang))

#main program

lang="PYTHON" # Here lang is called Global Variable

learnDS()

learnML()

learnDL()

#Program for demonstarting Local anf Global variables

#GlobalLocalVarEx4.py

def learnDS():

sub1="Data Science"

print("\tTo Develop '{}' Apps, we use '{}' Programming".format(sub1,lang))

def learnML():

sub2="Machine Learning"

print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub2,lang))

def learnDL():

sub3="Deep Learning"

print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub3,lang))

#main program

#learnDS()

#learnML()

#learnDL()

lang="PYTHON" # Here lang is called Global Variable

======================================

global key word

======================================

=>When we want MODIFY the GLOBAL VARIABLE values in side of function defintion then global variable names must be preceded with 'global' keyword otherwise we get "UnboundLocalError: local variable names referenced before assignment"

Syntax:

-----------

var1=val1

var2=val2

var-n=val-n # var1,var2...var-n are called global variable names.

------------------

def fun1():

------------------------

global var1,var2...var-n

# Modify var1,var2....var-n

--------------------------

def fun2():

------------------------

global var1,var2...var-n

# Modify var1,var2....var-n

--------------------------

Examples:

-----------------------

#globalvarex1.py

a=10

def access1():

print("Val of a=",a) # Here we are accessing the global variable 'a' and No Need to use global kwd.

#main program

access1()

---------------------------------------

#globalvarex2.py

a=10

def access1():

global a # refering global Varaible before its updation / Modification

a=a+1 # Here we are modifying the global variable value then we need to use global keyword.

print("Val of a inside of access1()=",a) # 11

#main program

print("Val of a in main before access1():",a) # 10

access1()

print("Val of a in main after access1():",a) # 11

-----------------------------------------------------------------------

Examples:

------------------

#globalvarex3.py

def update1():

global a,b # refering global Variables.

a=a+1 #updating global Variable a

b=b+1 #updating global Variable b

def update2():

global a,b # refering global Variables.

a=a\*10 #updating global Variable a

b=b\*10 #updating global Variable b

#Main program

a,b=1,2 # here a and b are called Global Variables

print("Val of a={} and Value of b={} in main program before update functions :".format(a,b))

# Val of a=1 and Value of b=2 in main program before update functions:

update1()

print("Val of a={} and Value of b={} in main program after update1():".format(a,b))

#Val of a=2 and Value of b=3 in main program after update1():

update2()

print("Val of a={} and Value of b={} in main program after update2():".format(a,b))

#Val of a=20 and Value of b=30 in main program after update1():

================================X=====================================

#Program demonstrating global keyword functionality.

#GlobalKwdEx1.py

a=0 # Global Variable

def update1():

global a

a=a+1

#main program

print("Val of a in main program before update()=",a) # 0

update1()

print("Val of a in main program after update()=",a) # 1

#Program demonstrating global keyword functionality.

#GlobalKwdEx2.py

a=10 # Global Variable

def update1():

global a

a=a+1

def update2():

global a

a=a\*2

#main program

print("Val of a in main program before update1()=",a) # 10

update1()

print("Val of a in main program after update1()=",a) # 11

update2()

print("Val of a in main program after update2()=",a) # 2

#Program demonstrating global keyword functionality.

#GlobalKwdEx3.py

def modification():

global a,b # here we are refering Global Varaibles and doing modifications

a=a+1

b=b+2

#main program

a,b=10,20 # Here 'a' and 'b' are called Global Variables

print("Val of a in main program Before modification()=",a) # 10

print("Val of b in main program Before modification()=",b) # 20

modification()

print("Val of a in main program after modification()=",a) # 11

print("Val of b in main program after modification()=",b) # 22

global and local variables and globals()

============================================

=>When we come acrosss same global Variable names and Local Variable Names in same function definition then PVM gives preference for local variables but not for global variables.

=>In this context, to extract / retrieve the values of global variables names along with local variables, we must use globals() and it returns an object of <class,'dict'> and this dict object stores all global variable Names as Keys and global variable values as values of value.

=>Syntax: -

var1=val1

var2=val2

--------------

var-n=val-n # var1, var2...var-n are called global Variables

def functionname():

------------------------

var1=val11

var2=val22

-----------------

var-n=val-nn # var1, var2...var-n are called local Variables

# Extarct the global variables values

dictobj=globals()

------------------------

globalval1=dictobj['var1'] # or dictobj.get("var1") or globals()['var1']

globalval2=dictobj['var2'] # or dictobj.get("var2") or globals()['var2']

==================================================================

Examples:

============

#globalsfunex3.py

a=10

b=20

c=30

d=40

def operations ():

obj=globals()

for gvn,gvv in obj.items():

print("\t{}---->{}".format(gvn,gvv))

print("="\*50)

print("\nProgrammer-defined Global Variables")

print("="\*50)

print("Val of a=", obj['a'])

print("Val of b=", obj['b'])

print("Val of c=", obj['c'])

print("Val of d=", obj['d'])

print("="\*50)

print("\nProgrammer-defined Global Variables")

print("="\*50)

print("Val of a=", obj.get('a'))

print("Val of b=", obj.get('b'))

print("Val of c=", obj.get('c'))

print("Val of d=", obj.get('d'))

print("="\*50)

print("\nProgrammer-defined Global Variables")

print("="\*50)

print("Val of a=", globals().get('a'))

print("Val of b=", globals().get('b'))

print("Val of c=", globals().get('c'))

print("Val of d=", globals().get('d'))

print("="\*50)

print("\nProgrammer-defined Global Variables")

print("="\*50)

print("Val of a=", globals()['a'])

print("Val of b=", globals()['b'])

print("Val of c=", globals()['c'])

print("Val of d=", globals()['d'])

print("="\*50)

=================================================

#main program

operations ()

==================================================

Examples:

-----------------------

#Program for demonstrating globals()

#globalsfunex2.py

a=10

b=20

c=30

d=40 # Here a,b,c,d are called Global Variables

def operation():

a=100

b=200

c=300

d=400 # Here a,b,c,d are called Local Variables

res=a+b+c+d+globals()['a']+globals().get('b')+globals()['c']+globals()['d']

print(res)

#main program

operation()

====================================X==================================

#program demonstarting globals()

#NOTE: globals() is used for obtaining Global Variables Names and Values ( Visible and Invisible) in the form dict type. gloabls() returns dict object and it contains Global Variables Names and Values in the form of (Key,Value)

#globalsFunEx1.py

a=10

b=20

def getvalues():

dictobj=globals()

print ("type of dict obj=",type(dictobj))

print("Number of Global Var names and Values=", len(dictobj))

print("---------------------------------------------------")

for gvn,gvv in dictobj.items():

print("\t{}------->{}".format(gvn,gvv))

print("---------------------------------------------------------------")

print("\nProgrammer-Defined Global Variable Names and Values-Way-1")

print("---------------------------------------------------------------")

print("Val of a--Global Variable={}".format(dictobj['a'])) # 10

print("Val of a--Global Variable={}".format(dictobj['b'])) # 20

print("-----------------------------------------------------------------")

print("\nProgrammer-Defined Global Variable Names and Values-Way-2")

print("---------------------------------------------------------------")

print("Val of a--Global Variable={}".format(dictobj.get('a')))

print("Val of a--Global Variable={}".format(dictobj.get('b')))

print("-----------------------------------------------------------------")

print("\nProgrammer-Defined Global Variable Names and Values-Way-3")

print("---------------------------------------------------------------")

print("Val of a--Global Variable={}".format( globals()['a'] ) )

print("Val of b--Global Variable={}".format( globals()['b'] ) )

print("-----------------------------------------------------------------")

print("\nProgrammer-Defined Global Variable Names and Values-Way-4")

print("---------------------------------------------------------------")

print("Val of a--Global Variable={}".format( globals().get('a')))

print("Val of b--Global Variable={}".format( globals().get('b')))

#main program

getvalues()

#program demonstarting globals()

#globalsFunEx2.py

a=10

b=20

c=30

d=40 # Here a,b,c,d are called Global Variable Names

def operation():

x1=10

x2=20

x3=30

x4=40 # Here x1,x2,x3,x4 are called Local Variable Names

res=a+b+c+d+x1+x2+x3+x4

print("sum of local and global var vals=",res)

#main program

operation()

#program demonstarting globals()---globals()

#globalsFunEx3.py

a=10

b=20

c=30

d=40 # Here a,b,c,d are called Global Variable Names

def operation():

a=100

b=200

c=300

d=400 # Here a,b,c,d are called Local Variable Names

res=a+b+c+d+globals()['a']+globals()['b']+globals()['c']+globals()['d']

print("sum of local var vals=",res)

print("=============OR=============")

res=a+b+c+d+globals().get('a')+globals().get('b')+globals().get('c')+globals().get('d')

print("sum of local var vals=",res)

#main program

operation()

==================================================

Anonymous Functions OR Lambda Functions

==================================================

=>Anonymous Functions are those which does not contain Name Explictly.

=>The purpose of Anonymous Functions is that " To Perform Instant Operations".

=>Instant Operations are those Which are used at that Point Time Only but No Longer interested to use next Part of the project".

=>To define Anonymous Functions, we use lambda keyword and hence Anonymous Functions are called Lamda Functions.

=>Anonymous Functions contains Single executable Statement Only But never contains Multiple Executable Statements.

=>Anonymous Functions automatically or Implcitly returns the value (No Need to use return statement)

----------------------------------------------------------------------

Syntax: varname=lambda params-list : statement

----------------------------------------------------------------------------------

Explanation

---------------------------------------------------------------------------------

=>Here Varname is an object of <class, 'function'> and varname indirectly treated as Anonymous Function Name.

=>Here lambda is Keyword used for Defining Anonymous Functions

=>Params-list represents list of Variable Names used for Holding / Storing the inputs coming from Functions.

=>Gere Statement represents Single Executable Statement and whose Value returns Automatically or Implicitly.

---------------------------------------------------------------------------------------------

Question: Define a Function for adding two values

Normal Function---Definition Anonymous Function

-------------------------

def sumop(a,b): sumop=lambda a,b:a+b

c=a+b

return c

---------------------------

Main program main program

----------------------

res=sumop(10,20) # Function Call res=sumop(3,4)

print(res)----30 print(res)

#Program defining Function for addition of two values

#AnonymousFunEx1.py

def sumop(a,b): # Normal Function

c=a+b

return c

addop=lambda x,y: x+y # Anonymous Function definition

#main program

a,b=int(input("Enter First Value:")),int(input("Enter Second Value:"))

res=sumop(a,b)

print("Type of sumop--Normal Function=",type(sumop))

print("Sum by using Normal Fun={}".format(res))

print("-"\*50)

r=addop(a,b)

print("Type of addop--Anonymous Function=",type(addop))

print("Sum by using Anonymous Function={}".format(r))

#Write a python program which will accept two numerical values and find the biggest and smallest among them by using anonymous functions

#BigSmall.py

big=lambda a,b: a if a>b else b # anonymous function

small=lambda a,b: a if a<b else b # anonymous function

#mian program

x=int(input("Enter First Value:"))

y=int(input("Enter Second Value:"))

bv=big(x,y)

sv=small(x,y)

print("Big({}, {})={}".format(x,y,bv))

print("Small({}, {})={}".format(x,y,sv))

#Write a python program which will accept two numerical values and find the biggest and smallest amoung them by using anonymous functions

#EqualBigSmall.py

big=lambda a,b: a if a>b else b if b>a else "BOTH VALUES ARE EQUAL" # anonymous function

small=lambda a,b: a if a<b else b if b<a else "BOTH VALUES ARE EQUAL" # anonymous function

#mian program

x=int (input("Enter First Value:"))

y=int (input("Enter Second Value:"))

bv=big(x,y)

sv=small(x,y)

print("Big({}, {})={}".format(x,y,bv))

print("Small({}, {})={}".format(x,y,sv))

#WAPP which will accept list of numerical values and find max and min

#MaxMinValues.py

def readvalues():

lst=[]

n=int(input("Enter How Many Values:"))

if(n<=0):

return lst

else:

print("Enter {} Values:".format(n))

for i in range(1,n+1):

lst.append(int(input()))

return lst

maxvalue=lambda lst:max(lst) # Anonymous Functions

minvalue=lambda lst:min(lst) # Anonymous Functions

#Main Program

lst=readvalues()

print("Content of lst=",lst)

if(len(lst)==0):

print("Empty List and can't find max and min")

else:

maxv=maxvalue(lst)

minv=minvalue(lst)

print("max({})={}".format(lst,maxv))

print("minv({})={}".format(lst,minv))

MaxMinValues.py

Open wi

==========================================

List comprehension

==========================================

=>The purpose of List comprehension is that to read the values dynamically from key board separated by a delimeter (space, comma, colon..etc) .

=>List comprehension is the most effective way for reading the data for list instead tradtional reading the data and also we can perform Various Operations.

=>Syntax:- listobj=[ expression for varname in Iterable\_object ]

=>here expression represents either type casting or mathematical expression

----------------------

Examples:

----------------------

print("Enter List of values separated by space:") # [10,2,222,50,10,4,55,-3,0,22]

lst= [float(val) for val in input().split() ]

print("content of lst",lst)

Examples:

------------------

lst=[4,3,7,-2,6,3]

newlst=[ val\*2 for val in lst ]

print("new list=",newlst) # [ 8, 6, 14,-4,12,6 ]

#ListComprehenEx1.py

lst=[10,20,30,40,50,60,70,80,90]

print('Original Content=',lst)

lst1= [ val+1 for val in lst ] # List Comprehension

print('Modified Content=',lst1)

print("------------------------------------------")

set1={val\*2 for val in lst } # set Comprehension

print(type(set1))

print('Modified Content=',set1)

tpl1=(val\*\*0.2 for val in lst ) # Not Tuple Comprehension

tpl2=tuple(tpl1) # Convert generator object into tuple obj

print(type(tpl2))

print('Modified Content=',tpl2)

#ListComprehenEx2.py

lst=[10,20,30,40,50,60,70,80,90,101]

print('Original Content=',lst)

lst1=[ val for val in lst if (val%3==0) ]

print("Multiples of 3=",lst1)

lst2=[ val for val in lst if (val%4==0) ]

print("Multiple of 4=",lst2)

#ListComprehenEx3.py

lst=[-10,20,-30,40,50,-60,70,-80,90,101,0,12]

print('Original Content=',lst)

lst1=[ val for val in lst if (val>0) ]

print("+Ve Values=",lst1)

lst2=[ val for val in lst if (val<0) ]

print("-ve values=",lst2)

#ListComprehenEx4.py

print("Enter List of values separated by space:") # 10 20 30 40 50 50 60 70 80 90

lst=[int (val) for val in input().split() ]

print("Content of list=",lst)

print("------------------------------------------------")

print("Enter List of Names separated by Comma:")

lst1=[val for val in input().split(",") ]

print("List of Names=",lst1)

#write a pyhton program which will accept list of numerical value and obtain sqaure of each number

#ListComprehenEx5.py

print("Enter List of value separated by hash symbol:")

lst=[int(val) for val in input().split("#")] # 10#24#45#24#78#12

print("Content of list=",lst)

sqrlist=[val\*\*2 for val in lst]

print("Square List=",sqrlist)

#ListComprehenEx6.py

print("Enter List of Names separated by Comma:")

lst1=[val for val in input().split(",") ]

print("List of Names=",lst1)

print("---------------------------------------------------")

nameslist=[name for name in lst1 if (len(name)>=3) and (len(name)<=6)]

print("Names list whose length is between 3 and 6=",nameslist)

================================================

Special Functions in Python

================================================

=>In Python Programming, we have 3 Types of Special Functions. They are

1) filter()

2) map()

3) reduce()

-----------------------------------------------------------------------

=========================================

1) filter()

=========================================

=>The purpose of filter () is that " To filter out some elements from given list of elements based on some condition".

-------------

=>Syntax: varname=filter(FunctionName , IterableObject )

-------------

=>Here varname is an object of <class, 'filter'>. Programatically we can convert an object of filter into Sequence, List, set and dict type by using Type Casting Functions.

=>filter () is one of the pre-defined special functions.

=>Function Name can be either Normal Function or Anonymous Function.

=>Iterable Object can any Sequence, List, set and dict type

=>The Execution Process of filter() is that " Each Element of Iterable object passing to Specified Function Name, Function Name Takes that value, appiled to the condition, If the condition is True then that Element filtered otherwise the element will be neglected". This Process will be repeated until all elements of Iterable object completed.

------------------------------------------------------------------------------------------------------------

#Program for obtaining list of Positive Values by using filter()

#FilterEx1.py

def positive(n):

if(n>0):

return True

else:

return False

def negative(n):

if n<0:

return True

else:

return False

#main program

lst=[10,-20,30,-40,50,-56,34,-24,0,-56]

x=filter(positive,lst)

print("Type of x=",type(x)) # <class, 'filter'>

print("content of x=",x) # content of x= <filter object at 0x000001E57B8263E0>

#Convert Filter object into list / set / tuple

listobj=list(x)

print("Given Data=",lst)

print("Possitive Elements=",listobj)

y=filter(negative,lst)

print("Type of y=",type(y)) # <class, 'filter'>

#Convert Filter object into list / set / tuple

tplobj=tuple(y)

print("Negative Elements=",tplobj)

#Program for obtaining list of Possittive Values by using filter()

#FilterEx2.py

poss=lambda k: k>0 # Anonymous Functions

negs=lambda k: k<0 # Anonymous Functions

#main program

print("Enter List of values separated by space:")

lst=[int(val) for val in input().split()]

pslist=list(filter(poss,lst))

nglist=tuple(filter(negs,lst))

print("Given data=",lst)

print("Possitive Elements=",pslist)

print("Negative Elements=",nglist)

#Program for obtaining list of Possittive Values by using filter()

#FilterEx3.py

print("Enter List of values separated by space:")

lst=[int(val) for val in input().split()]

pslist=list(filter(lambda k: k>0 , lst ))

nglist=tuple(filter(lambda k: k<0,lst))

print("Given data=",lst)

print("Possitive Elements=",pslist)

print("Negative Elements=",nglist)

#WAPP which will accept a line of text and count the number of vowels by filtering them

#FilterEx4.py

def vowels(ch):

if ch.lower() in ['a','e','i','o','u']:

return True

else:

return False

#main program

line=input ("Enter Line of Text:")

print("\nGiven Line:{}".format(line)) # python is an oop lang

vlst=list(filter(vowels,line))

print("Vowles Found={}".format(vlst))

print("No. of Vowels={}".format(len(vlst)))

#FilterEx5.py

vowels=lambda ch: ch.upper() in ['A','E','I','O','U']

#main program

line=input("Enter Line of Text:")

print("\nGiven Line:{}".format(line)) # python is an oop lang

vlst=list(filter(vowels,line))

print("Vowles Found={}".format(vlst))

print("No. of Vowels={}".format(len(vlst)))

#FilterEx6.py

line=input("Enter Line of Text:")

print("\nGiven Line:{}".format(line)) # python is an oop lang

vlst=list(filter(lambda ch: ch.upper() in ['A','E','I','O','U'] , line))

print("Vowles Found={}".format(vlst))

print("No. of Vowels={}".format(len(vlst)))

print("===============OR===================")

vl=[ str(ch) for ch in line if ch.upper() in ['A','E','I','O','U'] ]

print("Vowles Found={}".format(vlst))

print("No. of Vowels={}".format(len(vlst)))

#FilterEx7.py

def consonants(ch):

if (ch.isalpha() and ch.lower() not in ['a','e','i','o','u'] ):

return True

#main program

line=input("Enter Line of Text:")

print("\nGiven Line:{}".format(line)) # python is an oop lang

cons=list(filter(consonants, line))

print("List of Cons=",cons)

print("Number of Cons=",len(cons))

vlst=list(filter(lambda ch: ch.upper() in ['A','E','I','O','U'] , line))

print("Vowles Found={}".format(vlst))

print("No. of Vowels={}".format(len(vlst)))

====================================

2) map()

====================================

=>map() is used for obtaining new Iterable object from existing iterable object by applying old iterable elements to the function.

=>In otherwords, map() is used for obtaining new list of elements from existing existing list of elements by applying old list elements to the function.

=>Syntax:- varname=map(FunctionName,Iterable\_object)

=>here 'varname' is an object of type <class,map'> and we can convert into any iteratable object by using type casting functions.

=>"FunctionName" represents either Normal function or anonymous functions.

=>"Iterable\_object" represents Sequence, List, set and dict types.

=>The execution process of map() is that " map() sends every element of iterable object to the specified function, process it and returns the modified value (result) and new list of elements will be obtained". This process will be continued until all elements of Iterable\_object completed.

-----------------------------------------------------------------------

#Program for map()

#MapEx1.py

def hike(sal):

return (sal+sal\*(20/100) )

#main program

oldsallist=[10,15,12,20,18,8]

m=map(hike,oldsallist)

#print("type of m=",type(m))# <class, 'map'>

#print("content of m=",m) # <map object at 0x00000181D0666380>

#Type Cast map object into list

newsallist=list(m)

print("Old Salaries=",oldsallist)

print("New Salaries=",newsallist)

#Program for map()

#MapEx2.py

print("Enter List of Old Salaries separated by Comma:")

oldsal=[int(sal) for sal in input().split(",")]

newsal=list(map(lambda sal:sal\*1.2,oldsal))

print("Old Salaries=",oldsal)

print("New Salaries:")

for val in newsal:

print("{}".format(round(val,2)),end=" ")

print()

#Program for map()

#MapEx1.py

def hike(sal):

return (sal+sal\*(20/100) )

#main program

oldsallist=[10,15,12,20,18,8]

m=map(hike,oldsallist)

#print("type of m=",type(m))# <class, 'map'>

#print("content of m=",m) # <map object at 0x00000181D0666380>

#Type Cast map object into list

newsallist=list(m)

print("Old Salaries=",oldsallist)

print("New Salaries=",newsallist)

#Program for map()

#MapEx2.py

print("Enter List of Old Salaries separated by Comma:")

oldsal=[int(sal) for sal in input().split(",")]

newsal=list(map(lambda sal:sal\*1.2,oldsal))

print("Old Salaries=",oldsal)

print("New Salaries:")

for val in newsal:

print("{}".format(round(val,2)),end=" ")

print()

#Program for addition of two list elements

#MapEx3.py

def addition(x,y):

return (x+y)

#main program

print("Enter List of Values Separated By Space for List1:")

lst1=[int(val) for val in input().split()]

print("Enter List of Values Separated By Space for List2:")

lst2=[int(val) for val in input().split()]

lst3=list(map(addition,lst1,lst2))

print("Content of lst1=",lst1)

print("Content of lst2=",lst2)

print("Sum List=",lst3)

#Program for addition of two list elements

#MapEx4.py

print("Enter List of Values Separated By Space for List1:")

lst1=[int(val) for val in input().split()]

print("Enter List of Values Separated By Space for List2:")

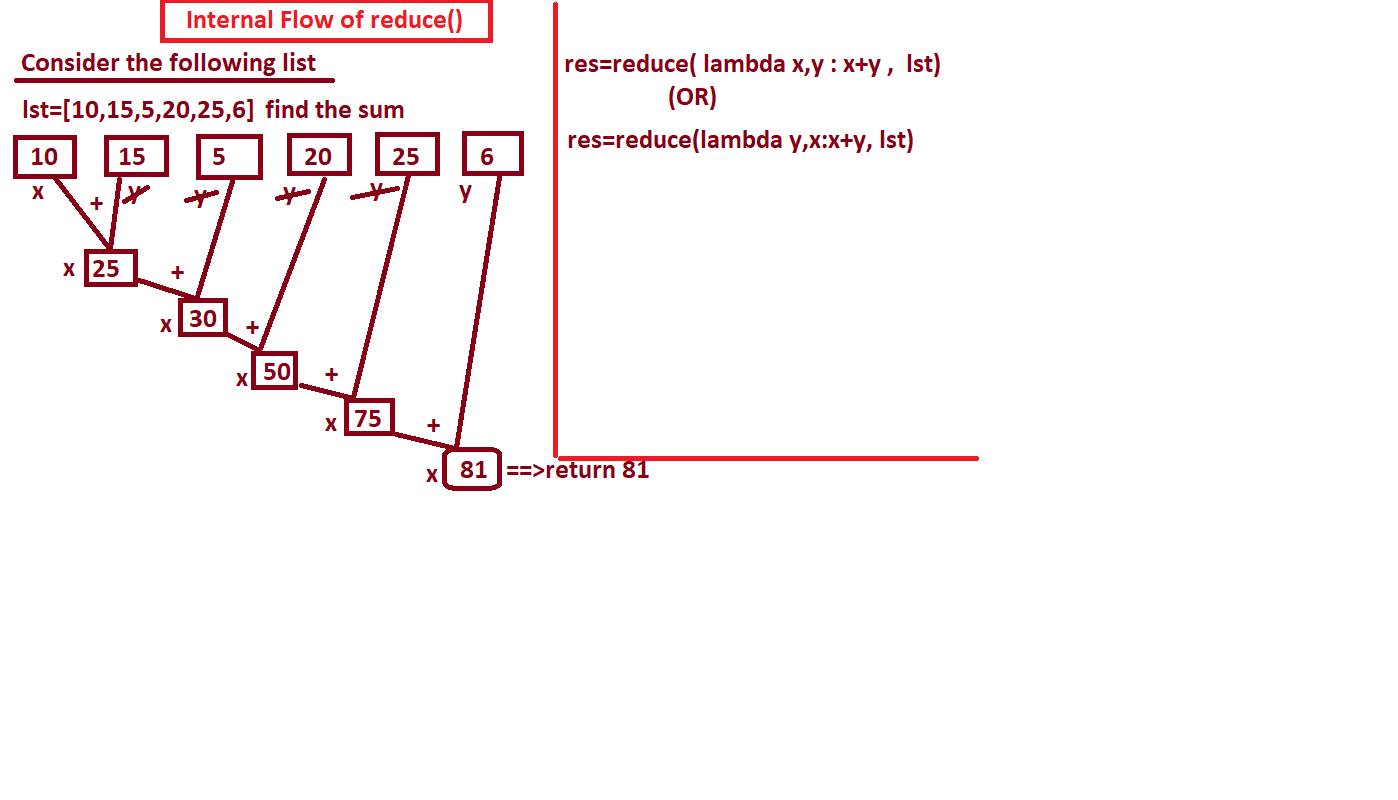
lst2=[int(val) for val in input().split()]

lst3=list(map(lambda k,v:k+v , lst1 , lst2))

print("Content of lst1=",lst1)

print("Content of lst2=",lst2)

print("Sum List=",lst3)

#Program for finding sum of list of Values

#reduceex1.py

import functools

lst=[5,10,6,3,14,8,2]

s=functools.reduce(lambda x,y:x+y,lst)

print("Sum({})={}".format(lst,s))

#Program for finding sum of list of Values

#reduceex2.py

import functools

def sumop(x,y):

return (x+y)

#Main Program,

print("List of Values Separated By Space:")

lst=[int(val) for val in input().split()]

s=functools.reduce(sumop,lst)

print("Sum({})={}".format(lst,s))

#write a python program which will accept list of value and find the max and min value

#reduceex3.py

import functools

print("List of Values Separated By Space:")

lst=[int(val) for val in input().split()] # lst=23 45 78 -2 23 1 2

bv=functools.reduce(lambda k,v: k if k>v else v, lst)

sv=functools.reduce(lambda k,v: k if k<v else v, lst)

print("\nMax({})={}".format(lst,bv))

print("\nMin({})={}".format(lst,sv))

#FilterReduceEx.py

import functools

print("List of Values Separated By Space:")

lst=[int(val) for val in input().split()] # lst=[10,-20,34,56,-4,-5,6,12,-6]

psnums=list(filter(lambda x: x>0 , lst))

pslistsum=functools.reduce(lambda x,y:x+y,psnums)

nnnums=list(filter(lambda x: x<0 , lst))

nnlistsum=functools.reduce(lambda x,y:x+y,nnnums)

print("-"\*50)

print("List of values :{}".format(lst))

print("\nList of Possitive Values :{}".format(psnums))

print("Sum of Possitive Values :{}".format(pslistsum))

print("\nList of Negative Values :{}".format(nnnums))

print("Sum of Negative Values :{}".format(nnlistsum))

print("-"\*50)

#FilterMapReduceEx.py

import functools

print("List of Salaries of Employees Separated By Space:")

#lst=[int(val) for val in input().split() if int(val)>=0 and int(val)<=1000 ] OR

lst=[int(val) for val in input().split() if 1000>=int(val)>=0 ]

print("-"\*50)

print("\nList of Salaries:",lst)

print("-"\*50)

#Filter for Those salaries which are in range of 0 to 500

sal\_0\_500=list(filter(lambda sal:500>=sal>=0,lst))

print("\nList of Slaries Ranges from 0-500={}".format(sal\_0\_500))

hike\_sal\_0\_500=list(map(lambda sal:sal\*1.1,sal\_0\_500))

print("List of Hiked Salaries Ranges from 0-500={}".format(hike\_sal\_0\_500))

sum\_hike\_sal\_0\_500=functools.reduce(lambda x,y:x+y,hike\_sal\_0\_500)

print("Total Salary paid by Company within the range 0-500={}".format(sum\_hike\_sal\_0\_500))

print("-"\*50)

#Filter for Those salaries which are in range of 501 to 1000

sal\_501\_1000=list(filter(lambda sal:1000>=sal>=501,lst))

print("\nList of Slaries Ranges from 501--1000={}".format(sal\_501\_1000))

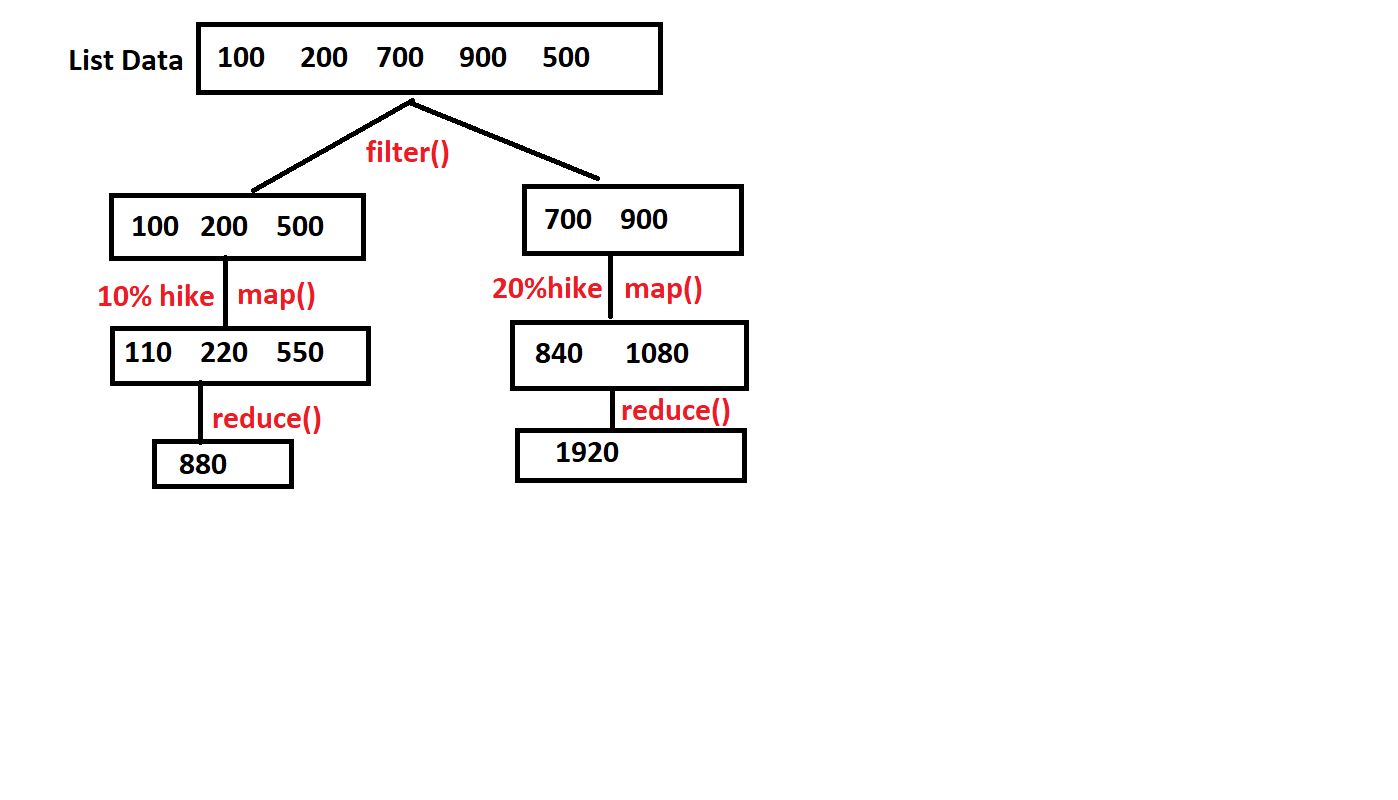
hike\_sal\_501\_1000=list(map(lambda sal:sal\*1.2,sal\_501\_1000))

print("List of Hiked Salaries Ranges from 501--1000={}".format(hike\_sal\_501\_1000))

sum\_hike\_sal\_501\_1000=functools.reduce(lambda x,y:x+y,hike\_sal\_501\_1000)

print("Total Salary paid by Company within the range 501-1000={}".format(sum\_hike\_sal\_501\_1000))

print("-"\*50)



===============================================

Modules in Python

===============================================

Index

---------

=>Purpose of Modules

=>Definition of Module

=>Types of Modules

=>Number of Approaches to Re-use the Modules

a) By Using import statement

b) By Using from ... import statement

=>Programming Examples

=>Re-Loading the Modules

=>Programming Examples

=============================================

Modules in Python

=============================================

=>We know that Functions concept makes us to understand How to perform operations and we can re-use within the same program but not able to re-use the functions across the programs.

=>To reuse the functions and global variables across the programs, we must use the concept of MODULES.

---------------------------------------

=>Definition of Module:

---------------------------------------

=>A Module is a collection of variables (global variables) , Functions and Classes.

-----------------------------------

=>Types of Modules:

-----------------------------------

=>In Python Programming, we have two types of Modules. They are

1) Pre-defined (or) Built-in Modules

2) Programmer or user or custom-defined modules.

-----------------------------------------------------

1) Pre-defined (or) Built-in Modules:

----------------------------------------------------

=>These modules are developed by Python Language Developers and they are avialable in Python Software (APIs) and they are used python programmers for dealing with Universal Requirements.

Examples: math cmath functools sys calendar os

re threading pickle random.......etc

=>Out of many pre-defined modules, in python programming one implicit pre-defined module imported to every python program called "builtins" .

--------------------------------------------------------------------------------------------------------

2) Programmer or user or custom-defined modules:

---------------------------------------------------------------------------------------

=>These modules are developed by Python Programmers and they are avialable in Python Project and they are used by other python programmers who are in project development to deal with common requirements.

=>Examples:- aop mathsinfo icici ......etc

---------------------------------------------------------------------------------------------------------------------------

=================================================

Development of Programmer-Defined Module

=================================================

=>To develop Programmer-Defined Modules, we must use the following steps

Step-1 : Define Variables (Global variables)

Step-2: Define Functions

Step-3: Define Classes

=>After developing step-1, step-2 and step-3 , we must save on some file name with an extension .py (FileName.py) and it is treated as module name.

=>When a file name treated as a module name , internally Python execution environment creates a folder automatically on the name of \_\_pycache\_\_ and it contains module name on the name of "filename.cpython-310.pyc ".

-------------------

Examples:

------------------

\_\_pycache\_\_ <-----Folder Name

-------------------------------------------

aop.cpathon-310.pyc <-------------------Module Name

mathsinfo.cpython-310.pyc<--------------Module Name

icici.cpython-310.pyc<----------------------Module Name

==============================================

Number of approaches to re-use Modules

===============================================

=>We know that A Module is a collection of variables, Functions and Classes.

=>To re-use the features(Variable Names, Function Names and Class Names ) of module, we have 2 approaches.They are

1) By using import statement

2) By using from.... import statement.

---------------------------------------------------------------------------------------

1) By using import statement:

---------------------------------------------------------------------------------------

=>'import' is a keyword

=>The purpose of import statement is that "To refer or access the variable names, function names and class names in current program"

=>we can use import statement in 4 ways.

-------------------

=>Syntax-1: import module name

-------------------

=>This syntax imports single module

----------------

Example: import icici

import aop

import mathsinfo

--------------------------------------------------------------------

=>Syntax-2: import module name1, module name2....Module name-n

-------------------

=>This syntax imports multiple modules

----------------

Example: import icici , aop, mathsinfo

----------------------------------------------------------------------------------------------------------------------------

=>Syntax-3: import module name as alias name

-------------------

=>This syntax imports single module and aliased with another name

----------------

Example: import icici as i

import aop as a

import mathsinfo as m

----------------------------------------------------------------------------------------------------------------------------

=>Syntax-4: import module name1 as alias name, module name2 as alias name......module name-n as alias name

-------------------

=>This syntax imports multiple modules and aliased with another names

----------------

Example: import icici as i, aop as a , mathsinfo as m

=>Hence after importing all the variable names, Function names and class names by using "import statement" , we must access variable names, Function names and class names w.r.t Module Names or alias names.

Module Name.Variable Name

Module Name.Function Name

Module Name.Class Name

(OR)

Alias Name.Variable Name

Alias Name.Function Name

Alias Name.Class Name

============================================================================

2) By using from.... import statement.

=======================================

=>Here "form" "import" are the key words

=>The purpose of from.... import statement is that " To refer or access the variable names, function names and class names in current program directly without writing module name as alias name of Module name."

=> we can use from.... import statement in 3 ways.

-------------------

Syntax-1: from module name import Variable Names,Function Names, Class Names

------------------

=>This syntax imports the Variable Names,Function Names, Class Names of a module.

Example: from calendar import month

from aop import addop,subop

from mathinfo import pi,e

from icici import bname,loc, calsimpleint

-----------------------------------------------------------------------------------------------------------

Syntax-2: from module name import Variable Names as alias name,Function Names as

alias name , Class Names as alias names.

-----------------------------------------------------------------------------------

=>This syntax imports the Variable Names,Function Names, Class Names of a module with alias Names

Example: from calendar import month as m

from aop import addop as a,subop as s, mulop as m

from mathinfo import pi as p ,e as k

from icici import bname as b,addr as n , simpleint as si

---------------------------------------------------------------------------------------------------------------------

Syntax-3: from module name import \*

---------------

=>This syntax imports ALL Variable Names,Function Names, Class Names of a module.

=>This syntax is not recommmended to use bcoz it imports required Features of Module and also import un-interrested features also imported and leads more main memory space.

Example: from calendar import \*

from aop import \*

from mathsinfo import \*

=>Hence after importing all the variable names, Function names and class names by using "from ....import statement" , we must access variable names, Function names and class names Directly without using Module Names or alias names.

Variable Name

Function Name

Class Name

=>Hence with "import statement” we can give alias name for module names only but not for Variables Names, Function Names and Class Names. Where as with "from ... import statement " we can give lias names for Variables Names, Function Names and Class Names but not for Module Name.

=======================================X===============================

#Program for demonstrating the need of modules

#MathsInfo.py--File Name & acts module name

PI=3.14

E=2.71 # here PI and E are called Global Variables

#Program for demonstarting the need of Modules

#aop.py---file name & acts as Module name

def addop(a,b):

print("sum({},{})={}".format(a,b,a+b))

def subop(a,b):

print("sub({},{})={}".format(a,b,a-b))

def mulop(a,b):

print("mul({},{})={}".format(a,b,a\*b))

#icici.py-----File Name & acts module name

bname="ICICI"

addr="HYD" # Here bname and addr are called Global variables

def calsimpleint():

p=float(input("\nEnter Principle Amount:"))

t=float(input("Enter Time:"))

r=float(input("Enter Rate of Interest:"))

#cal si

si=(p\*t\*r)/100

print("Simple Interest={}".format(si))

#Programmer1.py

print("Val of PI=",PI)

print("Val of E=",E)

#Programmer2.py

addop(10,20) # Function Call

mulop(4,5) # Function Call

#Programmer3.py

import MathsInfo

import aop

print("Val of PI=",MathsInfo.PI)

print("Val of E=",MathsInfo.E)

aop.addop(10,20)

#Programmer4.py

import icici

print("Bank Name:",icici.bname)

print("Bank Address:",icici.addr)

icici.calsimpleint()

#Progammer5.py

from calendar import \*

print(month(2022,12))

print(calendar(2023))

#ImportStmtsSyntax-1.py---

import aop

import icici

aop.addop(100,200)

icici.calsimpleint()

================================

reduce()

================================

=>reduce() is used for obtaining a single element / result from given iterable object by applying to a function.

=>Syntax:-

varname=reduce(function-name,iterable-object)

=>here varname is an object of int, float,bool,complex,str only

=>The reduce() belongs to a pre-defined module called" functools".

---------------------------------------

Internal Flow of reduce()

---------------------------------------

step-1:- Initially, reduce() selects First Two values of Iterable object and place them in First

var and Second var .

step-2:- The function-name(vlambda or normal function) utilizes the values of First var and Second var and applied to the specified logic and obtains the result.

Step-3:- reduce () places the result of function-name in First variable and reduce()

selects the succeeding element of Iterable object and places in second variable.

Step-4: Repeat Step-2 and Step-3 until all elements completed in

Iterable object and returns the result of First Variable.

------------------------------------------------------------------------------------------------------------------------------

==========================================

realoding a modules in Python

==========================================

=>To reaload a module in python , we use a pre-defined function called reload(), which is present in imp module and it was deprecated in favour of importlib module.

=>Syntax:- imp.reload(module name)

(OR)

importlib.reload(module name) ----->recommended

----------------------------------

=>Purpose / Situation:

-----------------------------------

=>reaload() reloads a previously imported module.

=>if we have edited the module source file by using an external editor and we want to use the changed values/ updated values / new version of previously loaded module then we use reload().

===================================X======================================

#shares.py---file and treated as module name

def sharesinfo():

d={"Tech":19,"Pharma":11,"Auto":1,"Finance":00}

return d

#main program

#sharesdemo.py

import shares

import time

import importlib

def disp(d):

print("-"\*50)

print("\tShare Name\tValue")

print("-"\*50)

for sn,sv in d.items():

print("\t{}\t\t:{}".format(sn,sv))

else:

print("-"\*50)

#main program

d=shares.sharesinfo()

disp(d)

time.sleep(15)

importlib.reload(shares) # relodaing previously imported module

d=shares.sharesinfo() # obtaining changed / new values of previously imported module

disp(d)

#Shares.py--File Name and Module Name

def sharesinfo():

d1={"IT":1000,"Fin":1001,"Auto":1111,"Pharma":7}

return d1

#SharesDemo.py

import Shares,time

import importlib

def dispdata(d):

print("-"\*50)

print("\tShare Name\tShare Value:")

print("-"\*50)

for sn,sv in d.items():

print("\t{}\t\t{}".format(sn,sv))

print("-"\*50)

#main program

dictobj= Shares.sharesinfo()

dispdata(dictobj)

print("I am from going to sleep for 15 secs--first time")

time.sleep(15)

print("I am coming out of sleep ")

importlib.reload(Shares) #------imp.reload(Shares)

dictobj= Shares.sharesinfo()

dispdata(dictobj)

print("I am from going to sleep for 15 secs --second time")

time.sleep(15)

print("I am coming out of sleep ")

importlib.reload(Shares)

dictobj= Shares.sharesinfo()

dispdata(dictobj)