GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING

UNIT – II

DATA, EXPRESSIONS, STATEMENTS

PART-B

INTRODUCTION TO PYTHON;

- Python is a general purpose programming language created by Guido Van Rossum during 1985 – 1990 at Netherlands.

- Python got its name from a BBC comedy series named “Monty Python’s Flying Circus”.

- It is an Interpreted, Interactive, object Oriented, High level programming language.

FEATURES OF PYTHON PROGRAMMING;

→ Python is interpreted

Python is processed at run time by the Interpreter. No compilation is needed.

→ Python is Interactive

Using python prompt python programs can be executed directly.

→ Python is Object Oriented

It Supports object oriented style of programming that encapsulates lodes within Objects.

→ Python is Extensible

Python can be embedded with C, C++, Java Script etc.

→ Python is a Beginner’s Language

→ Python is Portable

Python programs can be executed on various platforms without altering them.

→ Python is Free and Open Source.

Python software can be downloaded and used freely.

→ Python has rich set of Built – in standard Libraries.

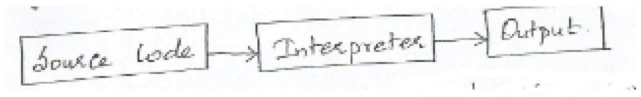
→ Python is simple, easy to learn and maintain.

PYTHON INTERPRETER;

- Python is considered an Interpreted language because python programs are executed by an

Interpreter.

- An Interpreter reads a high level program and executes it.



- Interpreter takes single instruction as input.

- It executes one line at a time.

- No Intermediate object code is generates.

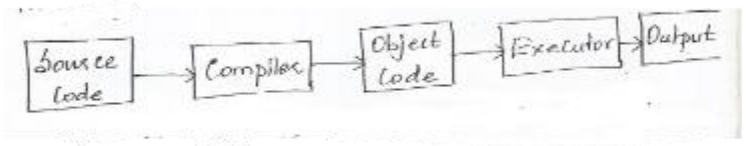
- Errors are displayed for every instruction.

- It requires less amount of Memory.

- Example: Python, JAVA.

COMPILER;

- It is a program used to convert a high level language into a low level language.



- It takes entire program as input.

- Intermediate object code is generated.

- Errors are displayed after entire program is checked.

- It requires more memory since object code is generated.

- Example: C, C++

MODES OF PYTHON INTERPRETER;

Python Interpreter can be used in two different ways. They are,

(i) Interactive Mode

(ii) Script Mode.

(i) Interactive Mode:

- It allows us to write codes in python command prompt.

- The chevron (>>>) is the prompt the interpreter use to indicate that it is ready to accept instruction.

- So that the Interpreter with >>> prompt indicates it is in Interactive mode.

- In this mode python code can be typed directly and results will be displayed immediately.

- This mode is use full to test small pieces of code.

- It can also be used as a calculator.

- Drawback: We cannot sare the statements for further use. We have to retype all the Statements to re – run them.

- Example:

```
>>> 5 + 10

15

>>> 5 + 50 * 10

505

>>> print (“Python programming”)

Python programming

>>> x = 10

>>> y = 15
>>> z = x + y

>>> print ("Sum = ", z)

Sum = 25
```

(ii) Script Mode:

- Script is a text file containing the python statements.
- In this mode python program is typed in a file and then using interpreter it will be executed to get the output.
- Python Scripts must be sared with py extension before execution.
- Once python Scripts are sared it can be used again and again for execution without re typing.
- Python Scripts can be edited.
- Example:

```
Print ( " My First Program")
x = 10
y = 20
z = x + y
Print ( " Sum of two Numbers = ", z )
```

Output:

```
My First Program

Sum of two Numbers = 30
```

VALUES AND TYPES;

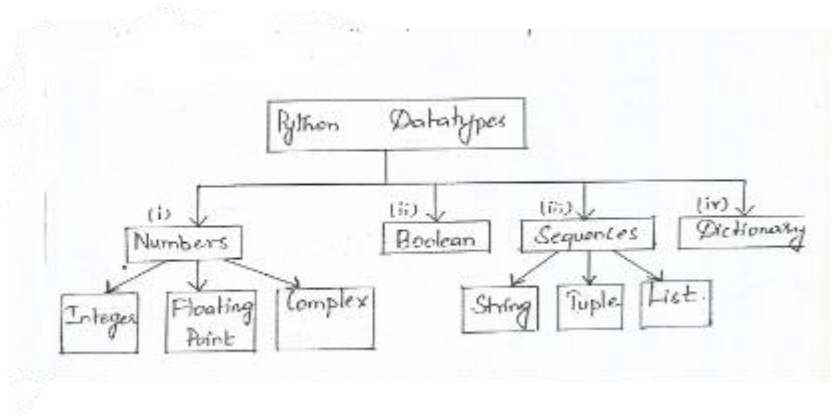
VALUES;

- Value can be any data represented by Number, letter or string.
- Example: 2, 42.5, 'a', "Hello".
- These values belongs to different data types.

DATATYPES IN PYTHON;

- Data types are used to classify one particular type of data.

- This is important because the specific data type will determine what values can be assign in it and what operations can be perform on it.



(i) Numbers;

- Number data type stress Numerical values.
- It is immutable (i.e.) values cannot be changed.
- Numbers are again classified into the following types:

  1. Integer – They are positive or negative whole numbers with no decimal Point. Ex: 56, 125

  2. Float - They are written with a decimal point dividing the integer and the fractional part.
     Ex: 5.63, 0.75

  3. Complex - They are of the form a + bj
     Where 'a' is the real part and
     and 'b' is the imaginary part.
     Ex; 3 + 2j

(ii) Boolean:

- Boolean is a data type having two values denoted by true or False.
- It is the value returned by conditional statements.
- The internal value of true is 1 and false is 0.
- Example:

```
>>> True + 1
2
>>> False + 1
```

<u>(iii) Sequence:</u>

- A sequence is an ordered collection of data.
- It is a combination of mutable and immutable data types.
- Three types of sequence data types available in python. They are,

  1. Strings - Collection of characters,
     Numbers and symbols.
     - represented by single or double Quotes.
     - <u>Example:</u> "abc ", 'python 3 '
  2. Tuple - Group of elements enclosed in parentheses. ()
     - They are immutable (i.e.) values cannot be changed.
     - <u>Example:</u> x = (1, 2, 3, " abc ")
  3. List - Group of elements enclosed in square brackets [ ]
     - They are Mutable (i.e.) values in a list can be changed
     - <u>Example:</u> l1 = [1, 2, " a " , 1.5]

<u>(iv) Dictionary:</u>

- Dictionary is a data type that contains elements based on key: value pour.
- All elements in dictionary are enclosed within curly brackets { }.
- In Dictionary keys are immutable and values are mutable.
- <u>Example :</u> $d1 = \{1: \text{"flowers"}, 2: \text{"fruit "}\}$

→ To find the type of data type () function can be used. It is a Built – in Library function.

→ <u>Example:</u> x = 10

```
y = "abc "
z = [1, 2, "a ", 3.5]          o/p:
Print (type (x))               < Class 'int' >
Print (type (y))               < Class 'Str'>
Print (type (z))               < Class 'list'>
```

DATA TYPE CONVERSION;

- In order to convert data from one type to another type Data type conversion is needed.
- It can be done by using the type name as a function.

Function	Description
int (x , base)	Converts x to an integer, base specifies the Base if x is a string.
float (x)	Converts x to a floating point number.
Complex (real, image)	Creates a complex number.
Str (x)	Converts x to a string.
Chr (x)	Converts an integer to a character.
List (x)	Converts x to a list.
tuple (x)	Converts x to a tuple.
dict (x)	Creates a dictionary, where x must be a sequence of (key : value) tuples / List
hex (x)	Converts an integer to hexadecimal.
Oct (x)	Converts an integer to an octal.

Example: 1

```
>>> int (12, 25)
12
>>> Float (35)
35.0
>>> Complex (5, 2)
5 + 2j
>>> Str (2.5)
“2.5 “
```

Example: 2

```
X = {1: “Sun “, 2: “Moon “ }
print (list (x))
print (tuple (x))
Y = [[11, “Flower”], [22, “Fruit”]]
Print (dict (y))
```

o/p:

```
[1, 2]
(1, 2)
```

{11: “Flower”, 22: “Fruit”}

KEYWORDS;

- Keywords are the reserved words in python.
- They have special meaning and cannot be used for any other purpose.
- Keywords cannot be used as Identifiers.
- In python 3 it has 33 keywords. They are as follows:

and	elf	import	raise	False	These
as	else	in	return	None	3
assert	except	is	try	True	keywords
break	finally	lambda	While		starts
class	for	nonlocal	With		with
Continue	from	not	yield		capital
def.	global	or			letters.
del	if	pass			

IDENTIFIERS;

- Identifier is the name given to identify variable, Function, Module, class etc. in python.
- Rules for Naming Identifier:
  - An Identifier must start with an alphabet or underscore
  - It should not start with numbers.
  - It is Case – Sensitive (i.e.) Lower case and upper case letters are different.
  - It should not be a keyword.
  - It can be of any length.

Example:

Valid Identifier	Inralid Identifier
Sum	5 Name ≠ starts with Number
reg – name	Stud Name ≠ space is n
Number 12	total
- abc	
XYZ	

VARIABLES:

- A variable is a name that refers to a value.
- When a variable is created it will allocate some memory space to store values in memory.
- In python declaration of variable is not needed. We can simply assign a value to a variable.
- Values can be assigned using Assignment statement. It creates a new variable and assigns them value.
- Syntax:

  Variable Name = value

  (Or)

  Variable Name = Expression
- Example :

  A1 = 10

  B1 = (5 * 20) + 10
- Single value can be assigned to several variables.

  Ex: a = b = c = 15
- Multiple values can be assigned to multiple variables.

  Ex: name, regno, Addr = "Geetha ", 8, "Ngl "

STATEMENT AND EXPRESSION;

Statement:

- Instructions that a python Interpreter can execute are called statements.
- A statement is a unit of code like creating a variable a displaying a value.
- Example :

  n = 17 ≠ Assignment Statement

  Print ("Good Morning ") ≠ print statement.

Expression:

- An Expression is a combination of values, variables and operators.
- A value all by itself is considered an expression and also a variable,

Example:

42

a + b

10 + 5 – 6

INPUT & OUTPUT STATEMENTS IN PYTHON;

Input Statement;

- It is used to get input from user using keyboard.
- Syntax :

  Variable Name = input ()
  (Or)
  Variable Name = input (“String “)
- Example :

  a = input () (or) a = input (“Enter a value “)
- In python by default all the input values are considered as string. So if we want to get values of different type, then input () statement must be written within type conversion function.

- Example :

  (i) X = input () ≠ accept string

  (ii) X = int (input ()) ≠ accept Integer

  (iii) X = float (input ()) ≠ accept Float

Method: 1	Method: 2
Ex: print (“Enter Name “) Name = input ()	name = input (“Enter Name’
Print (“Enter Reg No “) regno = int (input ())	reg = int (input (“Enter Reg No”))

Output Statement:

- If 1’s used to display the output value on screen.
- Function used is print ().

- Syntax :

  Print (value / Expression / variable)

- Example : 1 — O/P:

  Print ("Hello ") — Hello

  Example: 2

  ```
  x = 10

  y = 20

  z = x + y                O/P:

  Print ("Sum = ", z )     Sum = 30
  ```

INDENTATION:

- To define a block of code python use Indentation.
- It can be a single tab space (or) four space (By pressing space bar).
- When a Block ends the next line of code must be unindented.
- Example :

  ```
  a = 10
  if (a > 20):
      Print ("A is Bigger than 20 " )
  else:
      Print ("A is smaller than 20 " )
  ```

COMMENT STATEMENT IN PYTHON;

- Comment statements are used to add additional information about a program or a statement to explain it in Natural language.
- It starts with a hash sign (≠).
- With this user can able to understand what the code does? and why the code is used?
- It has no effect on the execution of the program. (i.e.) when python Interpreter see ≠ it ignores all the text after ≠ till the end of that line.
- ≠ is known as single line comment.

  Example:

  ```
  ans = l * b       ≠ Calculate area of Rectangle.
  ```

Multiline Comment (or) Doc String:

- It can be used within Function or class to explain what a Function or class will do.
- Triple Quotes (‘’’ - ‘’’) is the symbol used to represent it.
- Example : ‘’’ Program to Add two values
  Created on: 06 – 06 -2019
  Author: Geetha T. V ‘’’

TUPLE ASSIGNMENT:

- Python has a very powerful tuple assignment feature that allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.
- Left side is a tuple of variables and Right side is a tuple of values.
- The Number of variables on the Left and Number of values on Right must be same
- Each value is assigned to its respective variables.
- Tuple Assignment can be done by a process known as “Tuple Packing / Un packing “. (i.e.) values in a tuple on right are unpacked into the variables on right.
- Example : 1

```
T1 = (“Anu “, 25, 25000 )          ≠ Tuple packing
(name, age, salary)  = t1          ≠ Tuple Un packing
Print (name)
Print (age)
Print (Salary)
O/P:   Anu
       25
       25000
```

Example: 2 [To split Email Ii

```
mailied = “god @ abc. Org “
(uid, domain) = mailed – split
print (uid)
print (domain)     O/P:
                     god
                     abc. Org
```

Split () – Splits a string with a given character.

OPERATOR IN PYTHON;

- Operators are symbols used to perform operations on operands.
- Operands are variable or values to which operation must be performed.

- Example: a + b   Where a, b → Operands
                                        + → Operator.
- The following types of operators are supported by python.
    (i) Arithmetic Operators.
    (ii) Comparison Operators.
    (iii) Assignment Operators.
    (iv) Logical Operators.
    (v) Bitwise Operators.
    (vi) Membership Operators.
    (vii) Identity Operators.

(i) Arithmetic Operators:

- They are used to perform basic arithmetic operations. Such as Addition, subtraction, Multiplication and Division.

Operator	Description	Example a = 10, b = 3
+	Used to Add values	a + b = 13
-	Used to Subtract values	a - b = -7
*	Used to Multiply Values	a * b = 30
/	Used to Divide values	a / b = 3.33
//	Used to perform Quotient Division	a // b = 3
%	Used to perform Remainder Division	*a* % *b* = 1
**	Perform Exponential Operation	a ** b = 1000

(ii) Comparison / Relational Operators:

- These operators are used to compare the values to find the relation among them.
- It return Boolean value. (i.e) True / False.

Operator	Description	Example a = 10, b = 3

= =	It checks two values are Equal	a = = b → False
! =	It checks two values are Not Equal	a! = b → True.
>	It checks one value greater than another	a > b → True
<	It checks one value less than another value.	a < b → False
> =	It checks one value greater than or equal to another value.	a > = b → True
< =	It checks one value less than or equal to another value.	a < = b → False

(iii) Logical Operators:

- These operators are used to check two or more conditions at the same time.
- It returns Boolean value.

Operator	Description	Example a = True, b = False
And	If all conditions are true it returns True otherwise False	(a and b) ↓ False
or	If any one of the condition is True it return True.	(a or b) ↓ True
not	If the input is True output will be False and vice versa.	not (b) → True not (a) → False

Truth Table:

Condition 1	Condition 2	and	or
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Not

Input	Output
True	False
False	True

(iv) Bitwise Operators:

- These operators are used to perform operation on binary digits of a value.
- It perform bit – bit operation.

Operator	Description	Example a = 010, b = 011
& (Bitwise and)	Works on the principle of logical and.	a & b = 010
1 (Bitwise or)	Works on the principle of logical or.	a / b = 011
~ (Bitwise not)	Works on the principle of logical not.	~ a = 101
(XoR)	It gives 1, when the input values are different. It gives 0, when the input values are same.	
<< (Left Shift)	Used to shift all the bits to Left side…….	a << 1 = 100
>>		a >> 1 = 001

(Right Shift)	Used to shift all the bits to Right side.	

(v) Assignment Operator:

- It is used to assign a value or an expression to another variable.

Operator	Description	Example

=	Assign values from right side operand to Left side variable.	C = 12
		Z = x + y

- It can be combined with all Arithmetic operators to form Arithmetic Assignment Operator.
- They are, +=, -=, *=, /=, % =, //=, **=
- Ex: y = 5 ; y + = 2 ; → y = y + 2 O/P: y = 7

(vi) Membership Operators:

- It is used to check a value is present in a sequence or not.
- The sequence may be string, list or tuple.
- It returns Boolean value.

Operator	Description	Example x = 10, y = 10
In	Returns True, if the value present in the sequence otherwise False.	" h " in x ↓ True
not in	Returns True, if the value not present in the sequence otherwise False.	" S " not in x ↓ True.

(vii) Identity Operators:

- They are used to check the memory locations of two objects.
- It will check if two values are located on the some part of the memory.
- It returns Boolean value.

Operator	Description	Example x = 10, y = 10
Is	Returns True, if the variables on both sides of the operator has same value otherwise False.	x is y ↓ True
is not	Returns True, if the variables on both side of the operator has different value otherwise return False.	x is not y ↓ False

The following table list all operators from highest precedence to lowest with its associativity.

Operator	Operator Name	Associativity
()	Parentheses	Left → Right
**	Exponentiation	Right → Left
*, /, //, %	Multiply, Divide, Modulo	Left → Right
+, -	Addition, Subtraction	Left → Right
<<, >>	Left Shift, Right Shift	Left → Right
&	Bitwise and	Left → Right
	Bitwise xoR & or	Left → Right
< , > , < = , > =	Comparison Operator	Left → Right
= = , ! =	Equality Operator	Left → Right
= , + = , - = , * = ,/ =, % = , // =	Assignment Operator	Right → Left
is , is not	Identity Operators	Left → Right
in , not in	Membership Operators	Left → Right
not , and , or	Logical Operators	Left → Right

Examples:

1. Evaluate sum = x – y / 3 + z * 3 – 1

Where x = 3, y = 9, z = 77

Soln:

Sum = x – y / 3 + z * 3 – 1

3 – 9 / 3 + 77 * 3 – 1

3 – 3 + 77 * 3 – 1

3 – 3 + 231 -1

0 + 231 – 1

231 – 1

230

Ans: Sum = 230

2. Evaluate: (3 – 9) / 3 + 77 * (3 – 1)

Soln:

(3 – 9) / 3 + 77 * (3 – 1)

- 6 / 3 + 77 * (3 – 1)

- 6 / 3 + 77 * 2

- 2 + 77 * 2

- 2 + 154

152

Ans = 152

3. If a = 2, b = 12 and c = 1, Evaluate the following expression: d = a < b > c – 1

Soln:

d = a < b > c – 1

= 2 < 12 > 1 – 1

2 < 12 > 0

1 > 0

1

Ans: d = 1

4. Evaluate: 2 * * 3 + 4 // 7 – 6 * 9

Soln:

2 * * 3 + 4 // 7 – 6 * 9

8 + 4 // 7 – 6 * 9

8 + 0 – 6 * 9

8 + 0 – 54

8 – 54

- 46

Ans = - 46

FUNCTION IN PYTHON;

Definition:

- Functions are set of related statements that will perform a specific task.
- If helps us to break the program into small units or modules.
- So it is called as Modular programming.

Need for Function: / Use of Function

- Debugging, Testing and maintenance becomes eary when the program is divided into subprograms.
- Functions can make a program smaller by eliminating repetitive code.
- Functions provide code re – usability.

- The length of the program is reduced.
- Testing small parts of the program can be done separately.

Types of Functions:

(i) Built – in Function
(ii) User – defined Function.

(i) Built – in – Function:

- They are already created by the develop and stored in system Libraries.
- We can able to use them but we cannot able to modify them.
- Examples:
    - max () – returns the largest element
    - len () - returns the length of a string
    - sqrt () - returns the square root of a Number.
    - Print () – used to display a value.
    - input () – get an input from user.

(ii) User – defined Function:

- They are defined by the programmer according to their need.
- Function components :

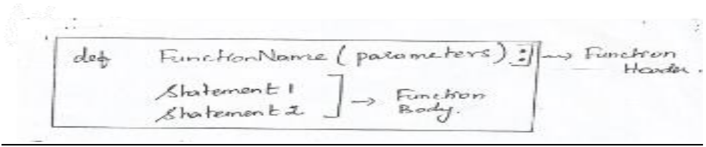
  (i) Function Definition:

- It defines the need of the function with the implemented logic. It contain two parts.

  (a) Function Header: It must begin with the keyword "def. "followed by Function Name and parenthesis () and ends with colon (:). Inside parenthesis parameters can be use.

  (b) Function Body: It's with indentation and ends with an optional return statement.

- Syntax :



```
def    FunctionName ( parameters ) :
          Statement 1
          Statement 2
```

(ii) Function Call:

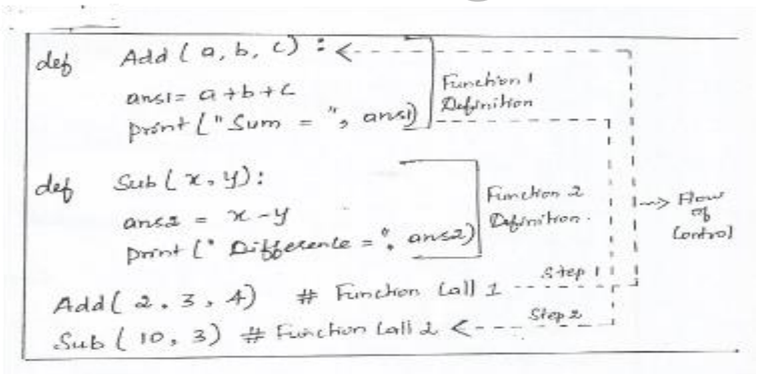
- When a function is defined it will execute only when it is called.
- A function can be called for number of times.
- Syntax ;

```
Function Name (arguments)
```

Flow of Execution:

- It means the order in which statements are executed.
- Function starts its execution when it is called.
- When a function is called, python Interpreter will check the Name and No. of arguments present in Function call is same as in Function Definitions.
- If they are same the Function Definition Start to execute.
- Execution always begin at the first statement of the program.
- Statements are executed one at a time in order from top to bottom.
- When the end of Function definition is reached then the next line of the Function call will be executed.

Example:



```
def  Add(a, b, c):
     ans1= a+b+c
     print("Sum = ", ans1)

def  Sub(x, y):
     ans2 = x-y
     print(" Difference =", ans2)

Add(2, 3, 4)  # Function Call 1
Sub(10, 3)  # Function Call 2
```

Output:

```
Sum = 9

Difference = 7
```

Explanation:

In the above program two Functions named as Add and sub are defined. When Add(2,3,4) is called its corresponding Definition will get executed and after its completion the second function call sub(10,3) will gets called and its definition will get executed, by assigning value of 10 and 3 to x and y.

PARAMETERS AND ARGUMENTS;

Parameters:

- Parameters are the values provided in function header.
- These are the values required by function to work.
- It is also called as Formal parameters.
- Example: def. Add (a , b) :

  ↓

  a, b → Parameters.

Arguments:

- Arguments are the values provided in function call.
- List of arguments should be in same number and type of arguments as present in function header.
- It is also called as Actual Arguments.
- Example : (i) Add (a, b)

  ↓

  Actual arguments as variables.

  (ii) Add (10, 20)

  ↓

  Actual arguments as values.

TYPES OF ARGUMENTS;

- Values passed in Function call are called arguments. They can be of the following types.
  (i) Required arguments
  (ii) Keyword arguments
  (iii) Default arguments
  (iv) Variable length arguments.

(i) Required arguments:

- It must passed to a function in correct positional order.

- The number of arguments in the function call must match exactly with the function definition.
- Example :

```
def.    Example (name, age):              O/P:
        Print ("Name: ", name )           Name: Anju
        Print ("Age: ", age )             Age: 18
Example ("Anju ", 18 )
```

(ii) Keyword arguments:

- In function call if the arguments are not in positional order using keywords they can be matched with the corresponding values.

  Example

```
def.    Example (name, age):              O/P:
        Print ("Name: ", name )           Name: Anju
        Print ("Age: ", age )             Age: 18
Example (age = 18, name = "Anju " )
```

(iii) Default arguments:

- In function call if proper values are not given then default values will be taken.
- Default values must be added in Function header.
- Example :

```
def     Example (name, age = 25):     O/P:
        Print ("Name: ", name )           Name = Anu
        Print ("Age: ", age )             Age = 25
Example ("Anu " )
```

(iv) Variable Length argument:

- If the number of arguments passed to a function is not known in advance it can be used.
- But we must use * asterisk before the parameter name.
- Example :

```
def     Example (* name):                 O/P:
        for i in name:                    Anu
                Print (i)                 Python
```

Example (“Anu “)	C
Example (“Python “, “C “, “C++”)	C++

PARAMETER PASSING METHODS;

- This method represent how values are passed to a function.
- It can be done by two ways.

(i) Pass By value

- In this method actual Value is passed.
- Changes done in formal argument will not reflect in actual argument.
- In python if the parameter belongs to integer, string it works by this Method.

Example:

```
def   swap (a, b):
      t = a
      a = b
      b = t
Swap (10, 20)
Print (a)        O/P:   10
Print (b)               20
```

(ii) Pass By Reference

- In this method changes done in formal argument will reflect in actual argument.
- In python if the parameter belongs to list, dictionary it works by reference method.

Example:

```
def   change List (x):
        x. append (5) ≠ Add c value to list
l1 = [100, 200, 300]
change List (l1)
 print (l1)

  O/P:  [100, 200, 300, 5]
```

Python will follow a method for parameter passing method called as call by object (or) call by object Reference (or) call by sharing. Based on the data type the working will be of pass by value or Reference.

MODULES;

- A module is a file containing python definitions, functions, statements and classes.
- Group of functions will form a module.

- Standard Library in python is considered as modules.
- To use a module in our program it must be imported using "import "keyword.
- Once a module is imported into a program we can access all the functions and variables of that module.
- Each module contain number of functions.
- To access one of the function from a module dot (·) notation should be used.
- Syntax :

  import Module Name ≠ To import a Module.

  Module Name. Function Name (arguments)

  ↳ To access a function from a Module. It is called as Dot Notation.
- Modules can be of two types.
  (i) Built – in Module
  (ii) User – defined Module.

(i) Built – in Module ;

- Modules that are already created by the developer.
- We can access all the function from them but we cannot able to modify it.
- Example: Standard Library in python.
- Some of the modules from python Library.

(a) math :

- It contains lot of mathematical functions and contants.
- Ex:

→ Sqrt (x)	- To find square root of x
→ Factorial (x)	- To find factorial value x
→ Sin (x)	- To find sin value of x.
→ Cos (x)	- To find cos value of x.
→ Log (x)	- To find log value of x.
→ Pi	- To print value of pi

(b) random:

- It is used to generate random Numbers.

- Ex:

→ randint (a, b)	- Generate random number between a and t
→ randrange (x)	- Generate random number less than x.
→ random ( )	- It generates any random number.

(ii) User – defined Module:

- Module created by user according to their need.
- Steps to create a Module :
  1. Create a python file by including one or more functions and save it with py extension.
  2. Include import statement.
  3. Access the function inside module using dot Notation.
- Example :

new. Py

```
def     Add  ( a , b ) :
        Print ( " Sum = " , a + b )
def     Sub ( a , b ) :
        Print ( " Diff = " , a – b )
def     Mul ( a , b ) :
        print ( " Product = " , a * b )
```

Main. Py

```
import   new
new. Add ( 2 , 3 )
new. Sub ( 15, 5 )
new. Mul ( 10 , 5 )
```

O/P:

```
Sum = 5
Diff   = 10
Product = 50
```

Explanation:

In the above program python file named as "new. Py "is created with three functions named as Add, Sub, Mul.

Again in another python File the module "new "is imported and its function are accessed. To get the output execute "main. Py "first.

Different ways to import a Module:

1. Using import:

- It is the simple and common way to import module.
- Example : import math

```
Print ("value of pi = ", math. Pi)
```

O/P: value of pi = 3.14159

2. Using import with renaming:

- Using this method module name can be renamed.
- Example: import math as m

```
Print ("Value of pi = ", m. pi)
```

O/P: Value of pi = 3.14159

3. using from import:

- It is used to get specific code from a module.
- Example : from math import pi

```
Print ("value of pi = ", pi )
```

O/P: value of pi = 3.14159

4. Using import *:

- It is used to import all function definitions from a module.
- Example : from math import *

```
Print ("value of pi = ", pi )
```

O/P: value of pi = 3.14159

ANONYMOUS FUNCTION; (or) LAMBDA FUNCTION

- Single line functions are called as Anonymous Function.
- It is otherwise called as Lambda Function.

- It contains a single expression as a body and not a block of statements.
- It does not have a return statement.
- It can have multiple argument but only one expression.
- Syntax:

  Variable = lambda (arguments): expression
- Example :
  - Sum = lambda (a, b, c) : a + b + c

    Print (“Answer = “, sum (10, 20, 30))

    O/P: Answer = 60
  - area = lambda (l , b) : l * b

    Print (“Area of Rectangle = “, area (5, 15))

    O/P:

    Area of Rectangle = 75

LIPETIME AND SCOPE OF A VARIABLE;

Lifetime:

- It defines the liveliness of a variable present in memony.
- Liveliness of a variable inside a Function live as long as the function executes and it destroyed when it returned from a function.

Scope:

- It defines the portion of a program where the variable is recognized.
- Variable inside a function has Local scope and outside has Global scope.
- Two types of variables.

  1. Local Variable
     - They are defined inside a function.
     - They are visible and available only within the function.

  2. Global Variable
     - They are defined outside a function
     - It can be accessed and visible anywhere in the program

Example:

```
a = 20    ≠ Global Variable

def   Display ():

      a = 30    ≠ Local variable.

      Print ("Value inside Function ", a )

Display ()

Print ("Value outside Function ", a )
```

O/P:

```
        Value inside Function 30

        Value outside Function 20
```

→ Local variable can be accessed like

Global variable using "global "keyword.

Example:

```
a = 20   ≠   Global variable

def  Display ():

     global   a

     a = 30

     Print ("Inside Function: ", a )

Display ()

Print ("Outside Function: ", a )
```

O/P:

```
        Inside Function: 30

        Outside Function: 30        [Since variable a is declared as global]
```

ILLUSTRATIVE PROGRAMS;

1. Exchange the value of two variables

<u>Method: 1</u> [Using temporary variable]

```
Print (“Enter value 1 “ )

a = int (input ())

Print (“Enter value 2 “ )

b = int (input ())

t = a

a = b

b = t

Print (“Swapped values “, a, “/t”, b )
```

<u>Output:</u>

```
        Enter    value 1

        50

        Enter    value 2

        80

        Swapped values 80 50
```

<u>Method: 2</u> [Without using temporary variable]

```
Print (“Enter    value 1 “ )

a = int (input ())

Print (“Enter    value 2 “ )

b = int (input ())

a, b = b, a

Print (“Swapped values: “, a, “/t”, b )
```

<u>Output:</u>

```
        Enter    value 1
```

```
5

Enter    value 2

10

Swapped value:   10    5
```

2. Circulate the values of ‘n’ variables

```
def    Circulate (a, n):

       x = a [n:] + a [: n]

      Print [“Circulated List: “, x ]

a = [10, 20, 30, 40]

Circulate (a, 2)     ≠ clockwise Rotation

Circulate (a, - 1) ≠ Anticlockwise Rotation
```

Output:

```
        Circulated List: [30, 40, 10, 20]

        Circulated List: [40, 10, 20, 30]
```

Explanation:

1 2 3 → Index position

Consider, a = [10, 20, 30, 40]

-4 -3 -2 -1 → Negative Index position

During function call, circulate (a, 2),

x = a [2 :] + a [: 2] ≠ a [2 :] → [30 , 40] will b

Sliced from List a.

≠ a [: 2] → [10, 20] will b

Sliced from List a.

Using ‘+ ‘operator a [2:] + a [: 2] both List a are concatenated to produce the result x = [30, 40, 10, 20]

During function call, circulate (a, -1):

x = a [-1 :] + a [: -1] ≠ a [-1 :] → [40] from the

Negative index position -1 the value is sliced from List a.

≠ a [: -1] → [10, 20, 30] from position 0 to -2 the

element are sliced from List a.

(i.e.) End value -1 , [-1 -1 = - 2]

Finally concatenating a [-1 :] + a [: -1] the result will be [40, 10 , 20, 30]

3. Distance between two points:

Formula: d = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

```
Import    math

Print (“Enter value of x1: “ )

x1 = int (input ())

Print (“Enter value of x2 “ )

x2 = int (input ())

Print (“Enter value of y1 “ )

y1 = int (input ())

Print (“Enter value of y2 “ )

y2 = int (input ())

d = math.  Sqrt (( x2 – x1 ) * * 2 + ( y2 – y1 ) * * 2)

Print (“Distance between two points = “, d )
```

Output:
```
Enter value of x1

9

Enter value of x2
```

```
3

Enter value of y1

7

Enter value of y2

2

Distance between two points = 7.81024
```