Module - 1

1.1 Why Should you learn to Write programs:

→ Programs are generally Written to Solve the Real time, arithemetical / logical problems.

→ Now a days, Computation devices like personal Computer, laptop and Cell phones are embedded With operating System, memory and processing Unit.

→ Using Such devices one Can Write a program in the language (which a Computer Can understand) of One's Choice to Solve Various types of problems.

→ Humans are tend get bored by doing Computational task multiple times.

→ Hence, the Computer Can act as a personal assistant for people to do their job.

→ To make a Computer to Solve the Required Problem One has to feed the program to it.

Hence, One Should know how to Write a Program.

→ There are many programming languages that Suit Several Situations.

→ The programmer must be able to choose the Suitable programming language for Solving the Required problem based on the factor like Computational ability of the device, data Structures that are Supported in the language, Complexity Involved in

implementing the algorithm in that language.

1.1.1 Creativity & Motivation :

→ When a person starts programming, he/she himself will be both the programmer and the end-user, because he will be learning to solve the problems. But later he may become a proficient programmer.

→ A programmer should have logical thinking ability to solve a given problem.

→ He/She should have a Creative in analyzing the given problems, finding the possible solutions, optimizing the resources available and delivering the best possible results to the end-user.

→ Motivation behind programming may be a job-requirement and such other prospects.

→ But the programmer should follow certain ethics in delivering the best possible outputs to his/her clients.

→ The responsibilities of a Programmer includes developing a feasible, user friendly software with very less or no hassles.

→ The user is expected to have only the abstract knowledge about the working of software, but not implementation details.

Hence the programmer should strive hard towards developing most effective software.

1.1.2 Computer Hardware Architecture :-

To understand the art programming it is better to know the basic architecture of computer hardware.

The Computer System involves some of the important parts as shown in figure

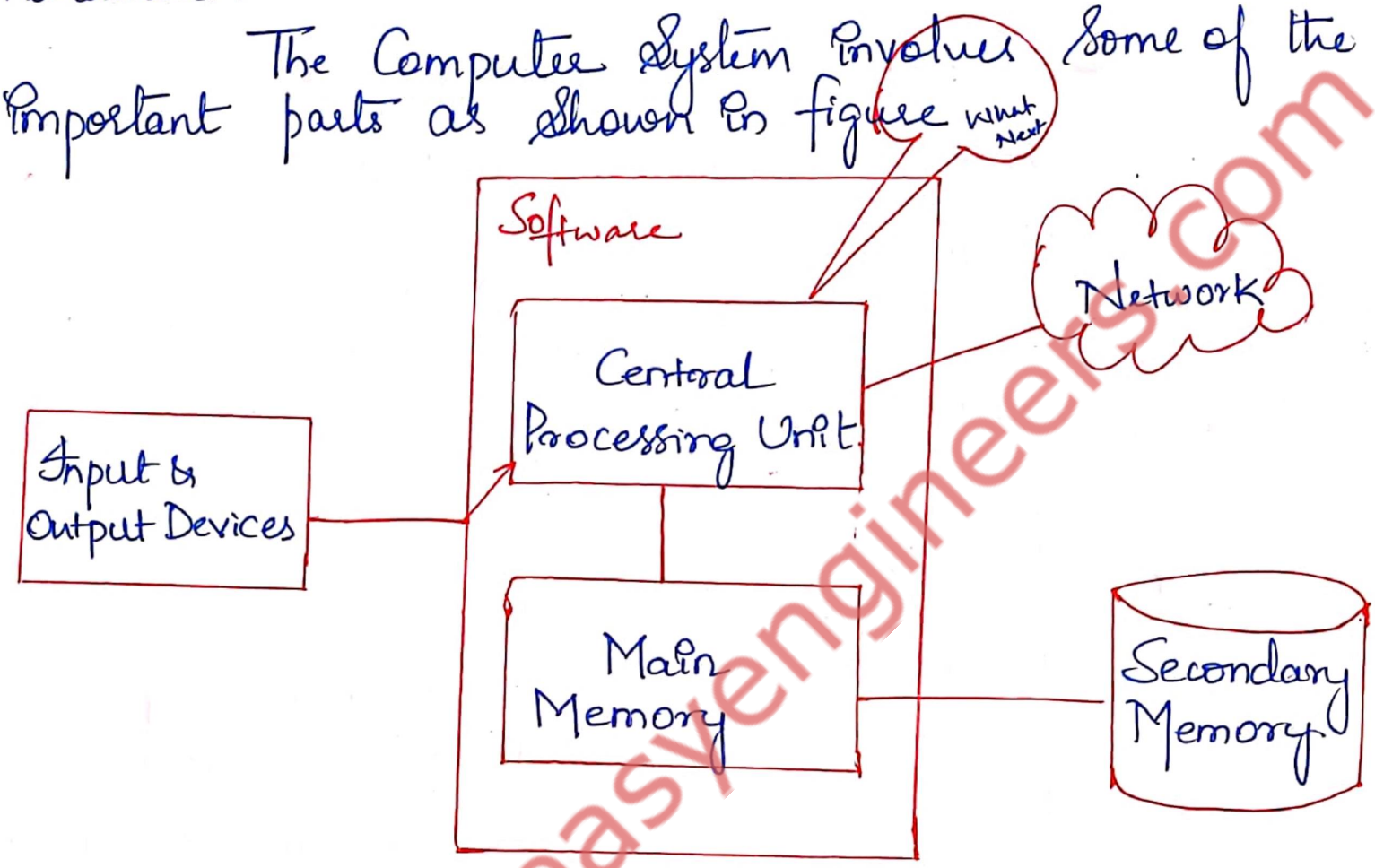


Fig :- Computer Hardware Architecture.

1) Central processing Unit :-

It performs basic arithematic, logical, Control and I/o operations Specified by the program instructions.

Cpu will perform the given tasks with a tremendous Speed.

Hence the good programmer keep the CPU busy by providing enough task to it.

2) Main Memory :-

It is the Storage area to which the CPU has a direct access. Usually programs Stored in Secondary Storage are brought in to Main memory before the execution.

The processor (CPU) will pick a job from the main memory & performs the task.

Usually, information Stored in main memory will be Vanished when the Computer is turned-off

3) Secondary Memory :-

The Secondary memory is the permanent Storage of Computer.

Usually the Size of Secondary memory will be Considerably larger than that of Main memory.

Hard disk, USB drive etc Can be Consid-ered as Secondary memory Storage.

4. I/o Devices :-

These are the medium of Communication between the User and the Computer.

Keyboard, mouse, monitor, printer etc are the examples of I/o devices.

5. Network Connection :-

Now a days, most of the Computer are Connected to Network and hence they Can Communi-Cate With other Computers in a network.

Retrieving the information from other Computers Via network will be Slower Compared to accessing the Secondary Memory.

1.1.3 Understanding Programming :-

A programmer must have skills to look at the data / information available about the problem, analyze it and then to build a program to solve the Problem. The skills to be processed by a good programmer includes -

→ **Through knowledge of Programming language :** One needs to know the vocabulary & grammar (technically known as Syntax) of the Programming language. This will help in Constructing Proper instructions in the program.

→ **Skill of implementing an idea :** A programmer should be like a 'story teller'. That is he must be capable of conveying something effectively.

He/She must be able to solve the problem by designing suitable algorithm & implementing it. And the program must provide appropriate output as expected.

Thus, the art of programming requires the knowledge about the problem's requirement & the strength/weakness of the programming language choosen for the implementation. It is always advisable to choose appropriate programming language that can cater the complexity of the problem to be solved.

1.1.4. Words & Sentences :-

Every Programming language has its own constructors to form Syntax of the language. Basic Constructs of a programming language includes set of Characters & keywords that it supports. The keywords have Special Meaning in any language & they are intended for doing Specific task.

Python have a finite set of 33 keywords.

and	as	assert	break	class	Continue
def	del	elif	else	except	False
finally	for	from	global	if	import
in	is	lambda	None	nonlocal	not
or	pass	raise	return	True	try
while	With	Yield			

Table: Keywords in Python.

A programmer may use Variables to store the values in a program. Unlike many other programming languages, a Variable in Python need not to be declared before its use.

1.1.5 Conversing With Python :-

→ Once the Python is Installed, One can go ahead With Working Python.

→ Use the IDE of your choice for doing programs in python.

→ After Installing Python. Just type "Python" in Command prompt, you will get the message as Shown in the following figure.

→ The prompt >>> (called Chevron) indicates the System is ready to take python instructions. If we would like to use the default IDE of python, that is the IDLE, then you can just run IDLE & we will get the editor as shown in below fig.

After Understanding the basics of few editors of Python, let us start our Communication With python, by saying Hello World. The python uses print( ) function for displaying the Contents.

Consider the following Codes

```
>>> Print("Hello World")   # type this & press Enter ↵
Hello World                # Output displayed.
>>>                        # prompt Returns again.
```

→ Hence after typing this the first line of code & pressing the enter key. We Could able to get the Output of that line Immediately.

Then the prompt (>>>) Returned on the Screen. This Indicates, python is Ready to take next Instruction as Input for processing.

Once We are done With the program We Can Close or terminate python by giving Quit() Command as shown —

```
>>> Quit()          # python terminates.
```

1.1.6 Terminology: Interpreter & Compiler

All digital Computers Can Understand only the Machine language Written in terms of Zero's & Ones. But for the programmer. It is difficult to code in Machine language.

Hence We generally Use high level programming languages like java, C++, PHP, perl, javascript etc.

Python is also one of the high level programming languages. The programs Written in high level language are then translated to machine level instructions so as to be executed by CPU. How this translation behaves depending on that type of translators Viz. Compilers & Interpreters.

A Compiler translates the Source Code of high-level programming language into Machine level language.

For this purpose the Source Code must be a Complete program stored ~~with~~ in a file. (Extensions .cpp, .c, .java, .py). The Compiler generates executable files (usually with extensions .exe, .dll, etc) that are in Machine language. Later these executable files are executed to give the Output of the program.

On the other hand, Interpreter performs the instructions directly, Without requiring them to be Pre Compiled.

Interpreter parses the Source Code & interprets it immediately.

Hence every line of code Can generate the output immediately & the Source Code as a Complete set, need not be Stored in a file.

That is why in the previous section, the usage of single line print("Hello World") could be able to generate the output immediately.

Consider an example of adding two numbers:

```
>>> x=10
>>> y=20
>>> z = x+y
>>> print(z)
30.
```

Here x, y and z are variables storing respective values. As each line of code above is processed immediately after the line,

The variables are storing the given values. Observe that though each line is treated independently the knowledge (or information) gained in the previous line will be retained by python & hence, the further lines can make use of previously used variables.

1.1.7 Writing a Program :-

As python is interpreted language, one can keep typing every line of code one after the another as shown in previous section.

But in real time scenario, typing a big program is not a good idea. It is not easy to logically debug such lines.

Hence python programs can be stored in a file with extension .py and then can be run.

Programs written with in file are obviously reusable & can be run whenever we want.

1.1.8 What is a Program ?

A program is a sequence of instructions intended to do some task.

For ex:- If we need to count the no of occurances of each word in a text document. We can write a program to do so.

Writting a program will make the task easier compared to manually counting the words in a document.

Moreover, most of the times, the program is a generic solution.

Hence the same program may be used to count the fruquency of words in another file.

The person who doesnot know anything about the programming also can run this program to count this words.

Programming languages like python will act as an intermediary between the computer & the programmer. The endusee can request the programmer to write a program to solve one's problem.

1.1.9 The Building Blocks of Programs :-

There are certain low-level conceptual structures to construct a program in any programming language.

They are called as building blocks of a program & listed below :-

1. Input :- Every Program may take Some Inputs from outside. The Input may be through keyboard, mouse, disk-file etc. or even through Some Sensors like microphone, GPS etc

2. Output :- Purpose of a program itself is to find the Solution to a problem. Hence every Program must generate at least One Output.

Output may be displayed on a monitor or Can be Stored in a file. Output of a program may even be a music/Voice Message.

3. Sequential Execution :-

In general the Instructions in the Program are Sequentially executed from the top.

4. Conditional Execution :-

In Some Situations, a Set of Instructions have to be executed based on the truth-Value of a Variable or expression.

Then Conditional Constructs (like if) have to be Used. If the Condition is true, one Set of Instructions will be executed & if the Condition is false the true block is Skipped.

5. Repeated Execution :-

Some of the problems require Set of Instruction to be repeated multiple times. Such Statement Can be Written with the help of looping Structures like for, While etc

6. Reuse :- When we write the programs for general Purpose- Utility tasks, it is better to write them with a seperate name, so that they can be used multiple times when/where ever required. This is possible with the help of functions.

1.1.10 What Could possibly Go Wrong ? :

It is Obvious that one can do mistakes while Writing a program. The possible mistake are categorized as below :-

1) Syntax Error.
2) Logical Error.
3) Semantic Error.

1) Syntax Error :-

The statements which are not following the grammer (or Syntax) of the programming language are tend to result in a Syntax errors.

Python is a Case-Sensitive language. Hence there is a Chance that a beginner may do some syntatically Mistakes while Writing a pelogram.

The lines Involving such mistakes are encountered by the python when you run the Program & the errors are thrown by Specifying possible reasons for the error. The programmer has to Correct them & then proceed further.

2) Logical Errors :-

Logical Errors occurs due to poor Understanding of the problem. Syntatically, the program will be Correct. But it may not give the expected Output.

For ex :- You are intended to find a%b but by mistake you have typed a/b. Then it is logical error.

3. Semantic Errors :-

A Semantic Error may happen due to Wrong Use of Variables, Wrong operations or in Wrong order.

For ex :- Trying to modify un-initialized Variable etc

NOTE :- There is one more type of error : Runtime Error Usually Called as expect exceptions. It may occur due to Wrong Input, problem in database Connectivity. etc

Exceptions like:
1. Arithemetic Error.
2. Floating Point Error.
3. EOFError.
4. Memory Error.

Ex :- >>Print 'Hello'
File "<stdin>", line 1
Print 'Hello'
^
SyntaxError: Invalid Syntax

1.2 Variables, Expressions and Statements

1.2.1 Values & Types:

A Value is one of the basic things in a Program. It may be like 2, 10.5, "Hello" etc. Each Value in python has a type. Type of 2 is Integer type of 10.5 is a floating point Number. "Hello" is String etc

The type of Value can be checked Using "type" function as shown below

Examples:-

```
>>> type("hello")
<class 'str'>    #output
>>> type(3)
<class 'int'>    #output
>>> type(10.5)
<class 'float'>  #output
>>> type("15")
<class 'str'>    #output.
```

In the above 4 examples One can make out Various types, Str, Int & float.

By Observing the 4th example - It Clearly Indicates that Whatever enclosed Within a double Quote is a String.

The print Statement also Working for Integers, We use the python Command to start Interpre

Ex:- Python

```
>>> print (4)
4   # output.
```

When you type a large Integer, we might be tempted to use Commas between groups of three digits as in 1,000,000. This is not legal Integer in python, but it is legal:

```
>>> print (1, 000, 000)
1 0 0   # output.
```

2.2 Variables :-

A Variable is a named literal which helps to store a Value in the program. Variables may take Value that Can be modified whenever required in the program.

In python a Variable need not to be declared with a specific type before its Usage.

Whenever we want a Variable, directly we Can declare & use it.

The type of the Variable will be declared by the Value assigned to it.

A Value can be assigned to a Variable using assignment operator (=).

Ex:-
```
>>> Msg = "Hello World"
>>> n = 17.
>>> Pi = 3.142.

>>> print(Msg)
Hello World  #output

>>> print(n)
17           #output

>>> Print(pi)
3.142        #output
```

The type of the Variable is the type of the Value it refers to.

```
>>> type(Msg)
    <class 'str'>  #output
>>> type(n)
    <class 'int'>  #output
>>> type(pi)
    <class 'float'>  #output.
```

1.2.3 Variables Names & Keywords :-

It is a good programming practice to name the Variable such that its name indicates its purpose in the program.

There are certain Rules to be followed while naming a Variable:

1) Variable name must not be a Keyword.

2) They can Contain alphabets & numbers, but should not start with a number.

3) It may Contain a Special Character underscore which is Usually Used to Combine Variables with two words like my_name etc

No any other Special Characters like @, $ etc are allowed.

4) ~~Any~~ python is a Case-Sensitive, Variable name Sum is different from sum, Sum. etc

Ex :- >>> 3a = 5 # Starting with a number

Syntax Error: Invalid Syntax.

>>> a$ = 10 # Containing $

Syntax Error: Invalid Syntax.

>>> Pf = 15 # If is a Keyword.

Syntax Error: Invalid Syntax.

1.4 Statements :-

A Statement is a small unit of code that can be executed by the python interpreter.

It indicates some action to be carried out.

In fact, a program is a sequence of such statements.

Following are the examples of statements.

```
>>> print ("hello")        # printing statement.
hello
>>> x=5                    # assignment.
>>> print (x)              # printing assignment.
```

A script usually contains a sequence of statements. If there is more than one statement, the result appear one at a time as the statement execute.

```
Print(1)
X=2
Print(X)
```

Produces the output

1

2.

1.2.5 Operands and operators

Special Symbols used to indicate specific tasks are called operators.

An operator may work on single operand (unary operator) or two operands (Binary operators).

There are several types of operators like arithmetic operators, relational operators, logical operators etc in python

Arithmetic operators are used to perform basic operations are listed below.

OPERATOR	MEANING	EXAMPLE.
+	Addition	Sum = x+y
-	Subtraction	Sub = c-d
*	Multiplication	Mul = a*b.
/	Division	Q = a/b X = 10/3
//	Floor Division - Returns only integral part of the division.	F = a//b X = 5//3 ∴ X = 1.
%	Modulus - Remainder after division.	R = c%d.
**	Exponent	E = x**y.

Relation or Comparision Operators are used to check the relationship (likes less than, greater than etc) between two operands.

These operators return a Boolean Value - either True or false.

→ Assignment Operators :-

Apart from simple assignment operator = which is used for assigning values to Variables Python provides Compound assignment operators.

Ex: x = x+y

OR

x += y.

Now += is Compound assignment operator. Similarly one can use most of the arithmetic & bitwise operators (only binary operators, but not Unary). like *, /, %, //, &, ^ etc as Compound assignment operators.

```
>>> x=3
>>> y=5
>>> x+=y      # x = x+y
>>> print(x)
    8
>>> y//2      # y = y//2
>>> print(y)
    2          # only integer part will be printed
```

NOTE :-

1) Python has a Special feature : One can assign Values of different types to multiple Variables in a single Statement.

Ex:- >>> x, y, st=3, 4.2, "Hello"

>>> print ("x=", x, "y=", y, "st=", st)

x=3 y=4.2 st=Hello

2) Python Supports bitwise operators like &(AND), |(OR), ~(NOT), ^(XOR), >>(Right Shift) & <<(left Shift). These operators will operate on every bit of the operands Working procedure of these operators is same as that in other languages like C & C++.

1.2.6 Expressions :-

A Combination of values, Variables and operators is known as expressions.

Following are the few examples.

```
X=5
Y=X+10
Z=X-Y*5
```

The python Interpreter evaluates simple expression and give results even without "print()"

Ex:- >>> 5
5 # displayed as it is
>>> 1+4
5 # displayed the sum.

2.7 Order of Operations :-

When an expression Contains more than one operator, the evaluation of Operators depends on the Precedence of operators.

The python operators follow the precedence Rule (which can be remembered as PEMDAS) as given below:

1) Parenthesis :- Have the highest Precedence in any expression. The operations with in parenthesis will be evaluated first.

For ex: The expression (a+b)*c the addition has to be done first & then the sum is multiplied with C.

2) Exponentiation :- Has the 2nd precedence, But it is right associative. That is if there are two exponentiation operations Continuously, it will be evaluated from right to left.

For ex:-
```
>>> print (2**3)      # it is 2^3
8
>>> print (2**3**2)   # It is 2^(3^2)
512
```

3) Multiplication & Division :- are the next priority. Out of these 2 operations which ever Comes first in the expressions is evaluated.
```
>>> print (5*2/4)     # Multiplication & then division
2.5
>>> Print (5/4*2)     # division and then Multiplication
2.5
```

4. Addition & Subtraction :-

Are the least priority, out of these two operations whichever appears first in the expression is evaluated.

Ex :- >>> print (5+2-1)

6

1.2.8 String Operations :-

The + operators works with strings, but it is not addition in the mathematical sense.

Instead it performs Concatenation, which means joining the strings by linking them end-to end.

Ex :-
```
>>> first = 10
>>> Second = 15
>>> print (first + second)
25

>>> first = '101'
>>> Second = '200'
>>> print (first + second)
101200.
```

NOTE :- We can also use double Quotes to enclose String Value.

2.9 Asking the User for Input :-

Python uses the built-in function "Input()" to read the data from the keyboard.

When this function is invoked, the user input is expected.

The Input is read till user presses enter key.

Ex :-
```
>>> Str1 = Input()
Hello, How are you?  # User Input
>>> print ("String is", Str1)
String is Hello, How are you?  # printing Str1
```

When Input() function is used, the Curser will be blinking to receive the data.

For a better Understanding, it is better to have a Prompt message for the User informing what needs to be entered as an Input.

Ex :-
```
>>> Str1 = input ("Enter the String:")
Enter the String: Hello.
>>> print ("you have entered:", Str1)
You have entered: Hello.
```

One can use new line character \n in the function Input() to make the Cursor to appear in the next line of prompt message :-

Ex :-
```
>>> Str1 = Input ("Enter a String:\n")
Enter the String:
Hello.  # Cursor is pushed here
```

The Key-board Input Received Using input function is always treated as a String type.

If We need an Integer, We need to Convert it Using the function int(). Observe the following example:

Ex1:-
```
>>> x = input("Enter x: ")
Enter x: 10
>>> type(x)      # x takes the Value "10" but not 10
<class 'str'>    # so, type of x would be str.
```

Ex2:-
```
>>> x = int(input("Enter x: "))   # use int().
Enter x: 10
>>> type(x)      # Now, type of x is int.
<class 'int'>
```

A function float() is used to Convert a Valid Value enclosed within Quotes into float number as shown below.

```
>>> f = input("Enter a float Value: ")
Enter a float Value: 3.5
>>> type(f)
<class 'str'>    # f is actually String "3.5"
>>> f = float(f)  # Converting "3.5" into float Value 3.5.
>>> type(f)
<class 'float'>
```

A function ~~float~~ chr() is used to Convert an Integer Input into equivalent ASCII Character

```
>>> a = int(input("Enter an Integer:"))
Enter an Integer: 65
>>> ch = chr(a)
>>> print("Character Equivalent of", a, "is", ch)
Character Equivalent of 65 is A.
```

1.2.10 Comments :-

It is a good programming practice to add Comments to the program wherever required.

This will help someone to understand the logic of the program.

Comment may be single line or spread into multiple lines.

A single Comment in Python starts with the symbol #.

Multiline Comments are enclosed within a pair of 3 single Quotes '''

Ex1 :- # This is single line Comment.

Ex2 :-
```
''' This is a
good Multi line
Comment'''.
```

1.2.11 Choosing Mnemonic Variable Names :-

Choosing an appropriate name for Variables in the program is always at Stake.

Consider the following exs -

Ex1 :-

```
a = 10000
b = 0.3 * a
c = a + b
Print (c)        # Output is 13000
```

Ex2 :-

```
basic = 10000
da = 0.3 * basic
gross_sal = basic + da
Print ("Gross Sal =", gross_sal)   #output is 13000
```

One Can Observe that both of these two examples are Performing Same task. But Compared to Ex1, the Variables in Ex2 are indicating what is being Calculated. That is Variable names in Ex2 are indicating the purpose for which they are being used in the program. Such Variables names are known as mnemonic Variable names.

The Word mnemonic means memory aid. The mnemonic Variables are Created to help the programmer to remember the purpose for which they have been Created.

2.12 Debugging :-

Some of the common errors in beggineers Programmer may make are Syntax errors.

Though python flashes the error with messages Sometimes it may become hard to understand the cause of errors.

Ex1 :- >>> avg sal = 50000
Syntax Error: invalid syntax.

Here, there is a space between the terms avg & sal, which is not allowed.

Ex2 :- >>> m = 09
Syntax Error: Invalid token.

Python doesnot allow preceding zeros for numeric values.

Ex3 :-
>>> basic = 200
>>> da = 0.3 * Basic
Name Error: name 'Basic' is not defined

As python is Case Sensitive, "basic" is different from "Basic"

As shown in above examples the Syntax errors will be alerted by python. But, programmer is responsible for logical errors or Semantic errors.

Because, if the program does not yield into expected Output, it is due to mistake done by the programmer, about which python is unaware of it.

1.3 Conditional Execution :-

In general, the statements in a program will be executed sequentially. But sometimes we need a set of statements to be executed based on some conditions.

1.3.1 Boolean Expression :-

A Boolean Expression is an Expression which results in "True" or "False".

The "True" or "False" are special values that belong to class bool.

Ex1 :-
```
>>> type (True)
<class 'bool'>
>>> type ('False')
<class 'bool'>
```

Boolean expression may be as below :-

Ex2 :-
```
>>> 10 == 12
False
>>> x = 10
>>> y = 10
>>> x == y
True
```

Operators	Meaning	Example
>	Greater than	a>b
<	Less than	a<b
>=	Greater than or equal to	a>=b
<=	Less than or equal to	a<=b
==	Comparision	a==b

Operator	Meaning	Example.
!=	Not equal to	a!=b
is	Is same as	a is b
is not	Is not same as	a is not b

Ex:-

```
>>> a=10
>>> b=20
>>> x=a>b
>>> print(x)
    False
>>> print(a==b)
    False.
>>> print("a<b is", a<b)
    a<b is True.
>>> print("a!=b is", a!=b)
    a!=b is True.
>>> 10 is 20
    False.
>>> 10 is 10
    True.
```

Note:- == & is looks same and != & is not looks same. But the == & != operators does the equality Test. They will Compare the Values stored in Variables.

is & is not does the identity test. that means they will Compare whether two objects are same.

Usually two objects are same when their memory locations are same.

1.3.2 Logical Operators :-

There are 3 logical operators in python, As shown below:

Operators	Meaning	Example.
and	Returns true, if both operands are true	a and b
Or	Returns true, if any one of two operands is true	a or b
not	Returns true, if ~~any~~ the operand is false (It is a Unary operator)	not a.

NOTE :-

1. Logical operators treat the operands as boolean (True or False).
2. Python treats any non-zero number as True & Zero as false.
3. While Using and operator, if the first operand is false, then the second operator is not evaluated by Python. Because False and'ed with anything is False.
4. In Case of OR operator. If the first operand is True, the second operator is not evaluated. because True or'ed with anything is True.

Ex1 :- With Boolean operands.

```
>>> x = True.
>>> y = False.
>>> print ('x and y is'; x and y)
    x and y is False.
>>> print ('x or y is'; x or y)
    x or y is True.
>>> Print ("Complement of x is", not x)
    Complement of x is False
```

Ex2 :- (With numeric operands):

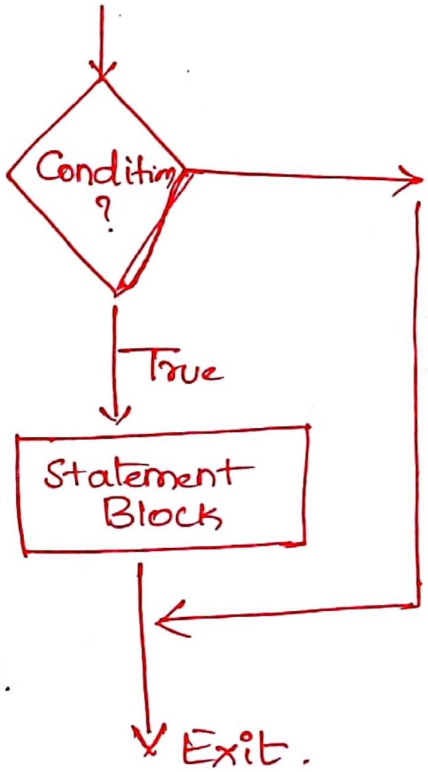
```
>>> a=-3
>>> b=10
>>> print(a and b) # and operation.
    10          # a is true hence b is evaluated & Printed.
>>> print(a or b)  # or operation
    -3          # a is true, hence b is not evaluated

>>> print(0 and 5) #  0 is false, so printed.
    0
```

1.3.3 Condition Execution :-

The basic level of Conditional execution can be achieved in python by using if Statement. The Syntax & flow charts are as shown bellow.

```
if Condition :
    Statement block.
```



Observe the Colon Symbol after Condition. When the Condition is true, the Statement block will be executed.

Otherwise it is Skipped. A set(block) of Statements to be executed under if is decided by the Indentation given.

Ex:-

```
>>> x=10.
>>> if x>40 :
        Print ("Fail")   # observe Indentation after if.

Fail                    # output.
```

Usually, the If Conditions have a statement block In anycase, the programmer feels to do nothing when the Condition is true. the Statement block can be Skiped by just typing pass Statement as shown below:

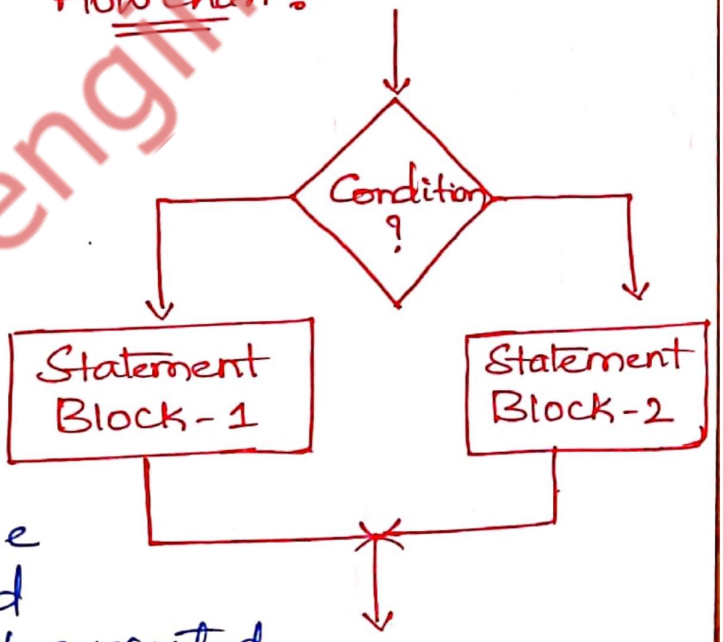
Ex1:- >>>
```
if x < 0:
    Pass  # do nothing when x is negative.
```

1.3.4 Alternative Execution :-

A second form of If Statements is alternative execution. Here when the Condition is true, one set of Statements will be executed and when the Condition is false, another set of Statements will be executed.

Syntax:
```
If Condition:
    Statement block-1
else:
    Statement block-2
```

Flow chart :-



As the Condition will be either true or false, only one among Statement block-1 and Statement block-2 will be get executed. These two alternatives are known as branches.

Ex:-
```
x = int(input("Enter x:"))
if (x%2 ==0)
    Print ('x is even")
else:
    Print ("x is odd")
```

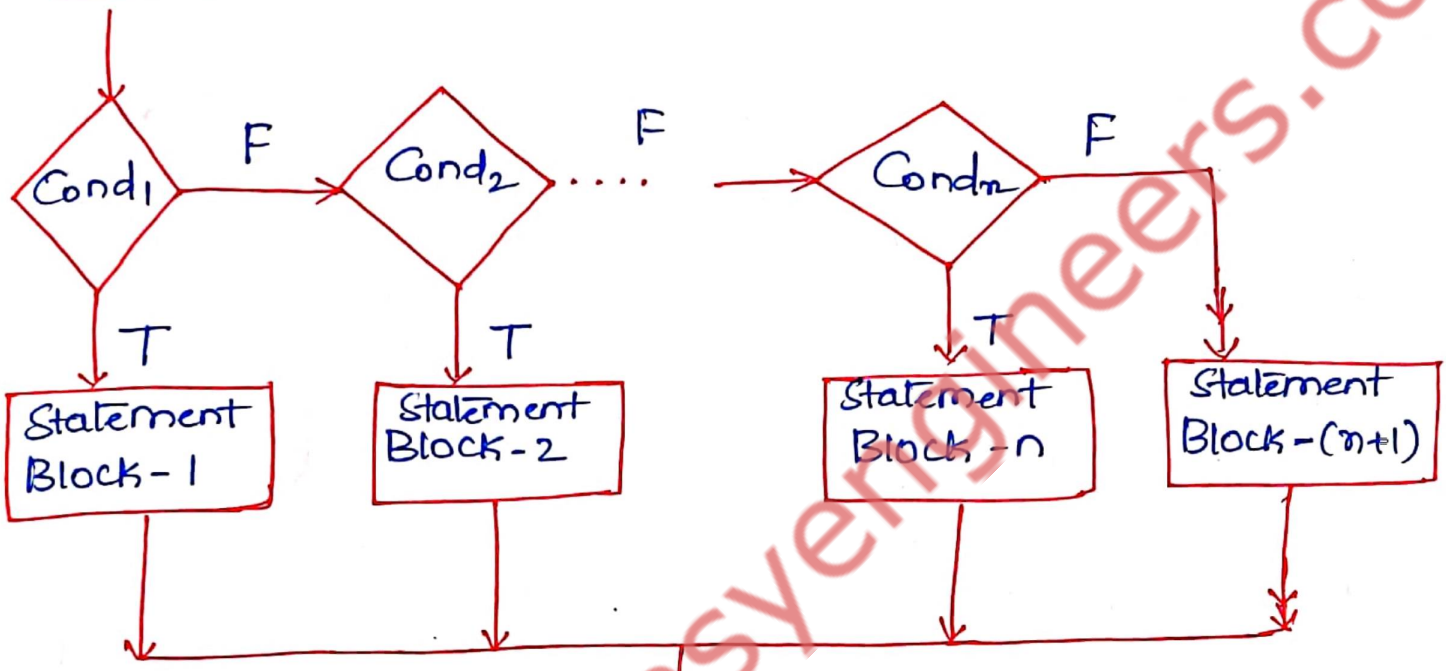
Output:-
```
Enter x : 13
x is odd.
```

1.3.5 Chained Conditionals :-

Some of the programs require more than one possibility to be checked for executing a set of statements.

That means, we may have more than one branch that is solved with the help of chained conditionals.

Flow chart :-



Syntax :-

```
If Condition1 :
        Statement Block-1
elif Condition2 :
        Statement Block-2
        :
        :
elif : Condition-n :
          Statement Block-n
else :
        Statement Block-(n+1).
```

The Conditions are Checked One by One sequentially. If any Condition is Satisfied, the respective Statement block will be executed and further Conditions are not checked.

Note that the last else block is not necessary always.

Ex:-
```
marks = float(input("Enter marks:"))
if marks >=80:
    Print("FCD")
elif marks >=60 and marks<80:
    Print("FC")
elif marks >=50 and marks<60:
    Print("Second Class")
elif marks >=35 and marks <50:
    Print("Third Class")
else:
    Print("Fail").
```

Output:-
```
Enter marks: 80
FCD.
```

1.3.6 Nested Conditions:-

The Conditional Statements Can be nested. That is One Set of Conditional Statements Can be Nested Inside the other. It Can be done in multiple way depending on programmers requirements.

Ex1:-
```
marks = float(input("Enter marks:"))
if marks >= 60:
    if marks < 70:
        print("First Class")
    else:
        print("Distinction")
```

Sample Output:-

```
Enter marks: 68
First Class.
```

Here the outer condition marks >= 60 is checked first.

If it is true then there are two branches for the inner conditional. If the outer condition is false the above code does nothing.

Ex2:-
```
gender = input("Enter gender:")
age = int(input("Enter age:"))
if gender == "M"
    if age >= 21:
        print("Boy, Eligible for Marriage")
    else:
        print("Boy, Not Eligible for Marriage")
elif gender == "F"
    if age >= 18
        print("Girl, Eligible for Marriage")
    else:
        print("Girl, Not Eligible for Marrig")
```

Sample Output :-

```
Enter gender : F
Enter age : 17
Girl, Not Eligible for Marriage.
```

NOTE :-

Nested Conditionals make the Code difficult to Read, even though there are proper Indentations. Hence it is advised to use logical operators like and to Simplify the nested Conditionals.

For ex the Outer & Inner Conditions in Ex, ~~above~~ can be joined as:

```
if marks >= 60 and marks < 70
    # do Something.
```

1.3.7 Catching Exceptions Using try & except :-

While We Executing programs there might be the chance of runtime error, ~~while doing some~~ The possible can be Wrong input.

Ex:-

```
a = int (input("Enter a:"))
b = int (input("Enter b:"))
c = a/b
Print(c)
```

When We Run the above Code, one of the possible situations would be :

```
Enter a : 12
Enter b : 0
```

Traceback (most recent call last):
.....

ZeroDivisionError : division by zero.

For the enduser, such type of System-generated error message is difficult to handle. So the code which is prone to Runtime error must be executed conditionally with try block.

The try block contains the statements involving suspicious code and the except block contains the possible remedy.

If something goes wrong with the statements inside try block, the except block will be excuited.

Otherwise the except block will be skipped

Ex:-
```
a = int(input("Enter a:"))
b = int(input("Enter b:"))
try:
    c = a/b
    Print(c)
except:
    Print("Div by zero is not possible")
```

Output:- Enter a: 12
Enter b: 0
Division by zero is not possible.

Standard Exceptions in Python

Sl No	Name	Purpose
1.	Exception	Base class for all exceptions
2.	ArithmeticError	Base class for all errors that occur for numeric calculations
3.	OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
4.	FloatingPointError	Raised when a floating point calculation fails
5.	ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
6.	EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
7.	ImportError	Raised when an import statement fails.
8.	KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing ctrl+c
9.	NameError	Raised when an identifier is not found in the local or global namespace.
10.	IOError	Raised when an input/output operation fails.
11.	SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the python interpreter does not exit.
12.	SystemExit	Raised when python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
13.	TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
14.	ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
15.	RuntimeError	Raised when a generated error does not fall into any category.

1:3:8 Short-Circuit Evaluation of logical Expression.

When a logical expression (expression involving operands and, or, not) is being evaluated, it will be processed from left to right.

For ex:-

```
X = 10
Y = 20
if X < 10 and X+Y > 25 :
    # do something.
```

Here the expression X<10 and X+Y>25 involves the logical operator "and".

Now X<10 is evaluated first, which results to be false.

As there is an "and" operator, irrespective of the result of X+Y>25, the whole expression will be false.

In such situations, python ignores the remaining part of the expression.

This is known as "Short-Circuiting" the evaluation. When the first part of logical expression results is True. the second part has to be evaluated to know the overall result.

The "Short-Circuiting" not only saves the Computational time, but it also leads to a technique known as "guardian pattern".

Consider the following Sequence of Statements :-

```
>>> x=5
>>> y=0
>>> x>=10 and (x/y)>2
False.

>>> x>=2 and (x/y)>2
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    x>=2 and (x/y)>2
ZeroDivisionError: division by zero.
```

Here, When we executed the Statement x>=10 and (x/y)>2, the first half of logical expression itself was False and hence by applying Short-Circuit ~~time~~ Rule, the remaining part was not executed at all.

Where as in the Statement x>=2 and (x/y)>2, the first half is True & the Second half is resulted in Runtime Error.

Thus in the expression, x>=10 and (x/y)>2, Short Circuit Rule acted as a guardian by preventing an error.

One Can Construct the logical expression to strategically place a guard evaluation just before the evaluation that might Cause an error as follows:

```
>>> x=5
>>> y=0          ①
>>> x>=2 and y!=0 and (x/y)>2
                  ②          ③
False.
```

∴ Here x>=2 result in True, but y!=0 evaluated to be false. So. exp3 never reached.

1.3.9 Debugging

One can observe from previous few examples that when a runtime error occurs, it displays a term Traceback followed by few indicators about errors.

A traceback is a stack trace from the point of error-occurance down to the call sequence till the point of call..

This is helpful when we start using functions and when there is a sequence of multiple function calls from one to another.

The traceback will help the programmer to identify the exact position where the error occurred. Most useful part of error message in traceback are:-

1) What kind of error it is.
2) Where it occurred.

Compared to runtime errors, Syntax errors are easy to find most of the times. But white space errors in Syntax are quite tricky because space & tabs are invisible.

Ex:-
```
>>> x=10
>>> y=15
    Syntax Error: Unexpected indent.
```

The error here is because of additional space given before y. As python has a different meaning (seperate block of code) for indentation, one cannot give extra spaces as shown above.

In general error messages indicate where the problem has occured, But the actual error may be before that point, or even in previous line of code.

→ Features of Python :-

1) Easy to code :-

Python is a high level programming language Python is very easy to learn languages as compared to other language. It is developer friendly language

2) Expressive language :-

Python language is more expressive means that it is more understandable and readable.

3. Object-oriented language :

One of the key features of python is oop Python supports oop & concepts of classes, objects, inheritence etc ...

4. Portable :-

If we have Python code for Windows & if we want to run this code on other platform such as Linux, Unix & mac.

5. Dynamically Typed language :-

That means the type (int, double, etc) for a variable is declared at run time not in advance, because of this feature we dont need to specify the type of variable.

2.1 Iteration :-

Iteration is a processing Repeating Some task. In Real time programming, We Require a set of Statements to be Repeated Certain number of times and/or till a Condition is met.

Every programming language provides Certain Constructs to acheive the Repetition of tasks.

2.1.1 The While Statement :-

The While loop has the Syntax below:

```
While Condition:
    Statement1
    Statement2
    ....
    Statementn
Statements after While.
```

Here While is a keyword. The Condition is evaluated first. Till its Value Remains true, the Statement, to Statement$_n$ will be executed.

When the Condition becomes false, the loop is terminated and Statements after the loop will be executed.

Ex :-

```
n=1
while n<=5:
    print(n)
    n=n+1
print("over").
```

The output of above code segment would be :-

1
2
3
4
5
Over.

In the above example, a Variable n is initialized to 1. Then the Condition n<=5 is being checked. As the Condition is true, the block of Code Containing print Statement (print(n)) & Increment Statement (n=n+1) are executed.

After these two lines, Condition is checked again. The procedure Continues till Condition become false, that is when n becomes 6.

Now the while loop is terminated & next Statement after the loop will be executed.

Thus in this example, the loop is iterated for 5 times.

2.1.2 Infinite loops, breaks, and Continue :-

A loop may execute infinite number of times when the condition is never going to become false.

Ex :-

```
n=1
While True:
    print(n)
    n=n+1
```

Here the condition specified for the loop is the constant True, which will never get terminated. Sometimes, the condition is given such a way that it will never become false & hence by restricting the program control to go out of the loop. This situation may happen either due to wrong condition or due to not updating the counter variable.

In some situations, we deliberately want to come out of the loop even before the normal termination of the loop. For this purpose break statement is used. The following example depicts the usage of break. Here the values are taken from keyboard until a negative number is entered. Once the input is found to be negative the loop terminates.

```
While True:
    x=int(input("Enter a number:"))
    if x>=0:
        print("You have entered", x)
    else:
        print("You have entered a -ve no")
        break # terminates the loop.
```

Sample Output :-

```
Enter a number : 60
You have entered: 60

Enter a number : 0
You have entered : 0

Enter a number : -4
You have entered a -ve number!!
```

In the above example, we have used the Constant True as Condition for While loop, which will never become false. So there was a probability of Infinite loop.

This has been avoided by Using break Statement with a Condition. The Condition is kept Inside the loop such a way that, If the User Input is a negative number, the loop terminates.

This Indicates that, the loop may terminate with just One Iteration (If User gives -ve no for the 1st Iteration) or It may take thousands of Iteration (If User give only +ve no).

Hence the no of Iterations here is unpredictable. But, We are making Sure that It will not be an Infinite-loop. Instead the User has Control on the loop.

Sometimes, programmer would like to move to next Iteration by Skipping few Statements in the loop, based on Some Condition. For this purpose Continue Statement is used.

For ex we would like to find the sum of 5 even numbers taken as input from the keyboard. The logic is.

1) Read a number from the keyboard

2) If that number is odd, without doing anything else, just move to the next iteration for reading another number.

3) If the number is even, add it to sum & increment the accumulator variable.

4) When accumulator crosses 5, stop the program.

Program for above task:-

```
Sum=0
Count=0
While True:
    X= Int(Input("Enter a Number"))
    If X%2!=0
        Continue.
    else:
        Sum+=x
        Count+=1

        If Count == 5 :
            Break

print("Sum=", Sum).
```

Sample Output :-

Enter a number : 13
——— " ——— : 12
——— " ——— : 4
——— " ——— : 5
——— " ——— : 8

Sum = 32.

2.1.3 Definite Loops Using for :-

The While loop iterates till the condition is met & hence, the number of iterations are Usually Unknown prior to the loop. Hence it is Sometimes called as Indefinite loop.

When We know total number of times the set of Statements to be executed, for loop will be Used. This is called definite loop.

The for loop iterates over a Set of numbers a Set of Words/ lines in a file etc

Syntax :-

```
for Var in list/Sequence:
    Statement1
    Statement2
    ....
    Statement n

Statement after for
```

Here "for" and "in" are Keywords

"list/sequence" is a set of elements on which loop is iterated. That is, the loop will be executed till there is an element in list/sequence.

"Statements" Constitutes body of the loop

Ex:- In the below given ex, a list names containing three strings has been created. Then the counter variable x in the for-loop iterates over this list.

The variable x takes the elements in names one by one & the body of the loop is executed.

```
names = ["Ram", "Shyam", "Bheem"]
for x in names:
    print(x)
```

Output:-

```
Ram
Shyam
Bheem.
```

NOTE:- In python, list is an important data-type. It can take a sequence of elements of different types. It can take values as a comma separated sequence enclosed within square brackets.

The for loop can be used to print (or extract) all the characters in a string as shown below :-

```
for i in "Hello"
    Print (i, end='\t')
```

Output :-

H e L L O

When we have a fixed set of numbers to iterate in a for loop we can use a function range(). The function range() takes the following format.

The "start" and "end" indicates starting & ending values in the sequence, where end is excluded in the sequence. The default value of start is 0.

The argument "steps" indicates the increment/decrement in the values of sequence with the default value as 1.

Hence, the argument 'steps' is options, let us consider few example on usage of range() function.

Ex :- Printing the values from 0 to 4

```
for i in range(5):
    Print(i, end = '\t')
```

Output :-

0 1 2 3 4.

Here 0 is the default starting value. The statement range(5) is same as range(0, 5) and range(0, 5, 1).

Ex 2:- Printing the values from 5 to 1

```
for i in range(5, 0, -1):
    print(i, end='\t')
```

Output:-

5 4 3 2 1

The function range(5, 0, -1) indicates that the sequence of values are 5 to 0 (excluded) in steps of -1 (downwards).

Ex 3:- Print only even numbers less than 10

```
for i in range(0, 10, 2):
    print(i, end='\t')
```

Output:-

0 2 4 6 8

2.1.4 Loop Patterns :-

The while-loop and for-loop are usually used to go through a list of items or the contents of a file & to check max or min data value.

These loops are generally Constructed by the following procedure.

1) Initializing one or more Variables before the loop Starts.

2) performing Some Computation on each item in the loop body, possibly changing the Variables in the body of the loop.

3) looking at the resulting Variables when the loop Completes.

The Construction of these loops patterns are demonstrated in the following examples:

Counting & Summing loops :-

One Can use the for loop for Counting number of items in the list as shown Eg-

```
Count = 0
for i in [4, -6, 32, 42, 12]:
    Count = Count + 1
Print ('Count =', Count)
```

Here the Variable Count is Initialized before the loop. Though the Counter Variable i is not being used inside the body of the loop, It Controls the number of iterations. The Variable Count is Incremented in every iteration, and at the end of the loop the total number of elements in the list is stored in it.

One more loop similar to the above is finding the sum of elements in the list:

```
total = 0
for x in [4, -6, 32, 42, 12]:
    total = total + x
Print ("Total:", total)
```

Here the variable total is called as accumulator because in every iteration, it accumulates the sum of elements. In each iteration this variable contains running total of values so far.

NOTE :- Buil in func : len(), sum().

Maximum & Minimum loops :-

To find the maximum element in the list, the following code can be used.

```
big = None
Print ('Before loop:', big)
for x in [12, 0, 21, -3]
    if big is None or x > big:
        big = x
    Print ('Iteration Variable:', x, 'Big:', big)
Print ('Biggest Value:', big).
```

Output :-

```
Before loop: None
Iteration Variable: 12   Big: 12
Iteration Variable: 0    Big: 12
Iteration Variable: 21   Big: 21
Iteration Variable: -3   Big: 21
Biggest Value: 21.
```

Here We Initialize the Variable big to None. It is a Special Constant Indicating Empty. Hence We Cannot Use Relational operator == While Comparing it With big.

Instead, the is operator must be Used. In every iteration, the Counter Variable x is Compared, With Previous Value of big. If x > big, then x is assigned to big.

Similarly, One Can have a loop for finding Smallest of elements in the list as given below :-

```
Small = None
Print ('Before Loop:', Small)
for x in [12, 0, 21, -3]:
    If Small is None or x < Small:
        Small = x
    Print ("Iteration Variable:' x, 'Small:', Small)
Print ("Smallest:', Small).
```

Output :-

```
Before Loop : None
Iteration Variable : 12     Small : 12
Iteration Variable : 0      Small : 0
Iteration Variable : 21     Small : 0
Iteration Variable : -3     Small : -3
Smallest : -3
```

NOTE :- Built-in functions max() & min().

Questions on Iteration

Q1: Write a python program for Computing the Sum of n naturals numbers & finding out the average of it.

Q2:- Write a python program to display Square of n numbers Using While loop.

Q3:- Write a python program for displaying even or odd numbers between 1 to n.

Q4:- Write a python program to display Fibonacci numbers for the Sequence of length n.

Q5:- Write a python program to find the Sum of 1 to 10 numbers.

Q6:- Write a python program to display the multiplication table.

Q7:- Write a program to print $1+1/2+1/3+\ldots+1/N$ Series.

Q8:- Write a program to check Whether the given number is prime or not.

Q9:- Write a python Program to display the fol pattern.

```
1 2 3 4 5
  2 3 4 5
    3 4 5
      4 5
        5
```

Q10 :- Write a python program to print the pattern as given below.

```
A
A B
A B C
A B C D
A B C D E.
```

Q11 :- Write a python program to print the pattern of Alphabets:

```
A A A A A
  B B B B
    C C C
      D D
        E
```

```
A A A A A
B B B B
C C C
D D
E
```

X