Programming and Problem Solving
Unit- 1 Notes

Unit I: Problem Solving, Programming and Python Programming

1.1 General Problem Solving Concepts

1.1.1 Problem Solving in Everyday Life:

Q.1. Define Problem. What are steps in problem solving?

Definition of problem

Problem is defined as a situation or issue or condition which needs to be solve to achieve a specific goal.

Steps in problem solving

1. **Identify the problem** - You need to be clear about what exactly the problem is. If the problem is not identified then it cannot be solved. example- when you want to learn a language then first identify which language do you want to learn
2. **Understand the problem** - Here problem should be defined properly so that it helps in understanding the problem more. It helps to understand knowledge base of the problem. It helps in finding a target for what you want to achieve.
3. **Identify alternative ways to solve the problem** - This step is very crucial because until and unless we explore all options, finding a correct solution is next to possible. This involves the process to create a list of solutions.
4. **Select the best way to solve the problem** - Identify and evaluate pros and cons of each possible solution and then select the best solution. Selecting proper way from alternatives require a lot of study and need to study consequences as well.
5. **List instructions that enable you to solve the problem using selected solution -** Here we need to list out the steps which should be followed so that anyone can solve the problem. This should be in detail and precise at the same time.
6. **Evaluate the solution -** It mean to test the solution and check if the result is satisfactory or not.

Example: I am always late for college

(i) Identification: getting late for college.

(ii) Understanding: (List out probable reasons of or getting late)
(Sleeping late, waking up late, Transportation, etc.)

(iii) List out alternative ways – (Solutions)
(a). Sleeping early at home.
(b). Waking up early.
(c). Waking alarm.
(d). Resolving travelling issue.

(iv) Select the best way – Sleeping early in night.

(v) List out the procedure
(a). Having dinner as early as possible.
(b). Avoid use of mobile phones in night.
(c). Completing homework as early as possible. (d). Then going early to bed.

(vi) Evaluate the solution: Reached College on time.

1.1.2 <u>Types of problems:</u>

Q. 2. Explain types of problems in detail?

Ans:

Problem can be categorized into two types depending on how the solution is found out or the approach which is followed to get that solution.

(i) Algorithmic solution

(a) These problems are solved by following some procedure or steps.

(b) For example: baking a cake. It can be done by following a series of instruction And we can expect results.

(c) Here following some predefined process needs to be done.

(d) Here solution is quite straight forward.

(ii) Heuristic solution

(a) These types of problems solved using knowledge based i.e. by collecting information about the problem.

(b) For example: Expanding one's business.

(c) In this type of approach there is no straight forward defined path which we can follow to solve a problem.

(d) We need to build that path based on trial and error.

(e) In this approach experience and knowledge is very important.

1.1.3 Problem Solving with computers:

Q. 3. Write a note on Problem solving with computers.

Ans: Computer is a multi-purpose electronic machine which is used for storing, organizing and processing the data.

Computer is not able to find the solution by itself; rather the computer is a machine which follows the instruction given by programmer to solve any problem.

So all the efforts required to understand the problem and define procedure to solve that problem are taken by programmer (a human being).

Problem Definition:

Before a program is written for solving a problem, it is important to define the problem clearly.

Define problem: Problem is defined as a situation or issue or condition which needs to solve to achieve the goal.

For most of the software projects, the system analyst approach system users to collect user requirements and define the problem that the system aims to solve.

System analyst typically looks at following issues:

- What input is required for achieving expected output?
- Expected output of the problem.
- Current method of solving the problem.
- Can the problem or part of the problem be more effectively solved by a software solution?
- Computers are built to deal with algorithmic solutions, which are often difficult or very time consuming for humans.

Solution:

- Solution means the instructions listed during problem solving – the instructions that must be followed to produce the best results.
- The result may be: More efficient, faster, more understandable or reusable.

Results:

- The result is outcome or the completed computer assisted answer.
- May take any form: Printout, updated files, output to monitor speakers, etc.

Program:

- Program is the set of instructions that make up the solution after they have been coded into a particular computer language.

1.1.4 Difficulties with problem solving:

Q.4. Explain the difficulties with Problem solving?

Ans:

- Some people may not have got training regarding how to solve problems.
- Some may not able to make a decision for fear it will not be the correct one.
- There may be inaccuracy in defining the problem correctly or may not generate the sufficient list of alternatives.
- While selecting best alternative, they may skip better alternative due to urgency.
- The problem solving process is difficult. It takes practice as well as time to perfect, but in the long run the process proves to be of great benefit.
- Illogical sequencing of instructions: In the process of problem solving on the computer, one of the most complicated jobs for the problem solver is writing the instructions. Example: Consider a task to find which number is the maximum from a set of three numbers. Near about everyone is immediately able to tell which is the largest but several of them are unable to explain the steps they followed to get the result. Computer is nothing but a machine that will perform only task that the user can explain.

1.1.5 Problem Solving Aspects:

Q.6. What is modularization? Explain different approaches to design an algorithm.

Ans:

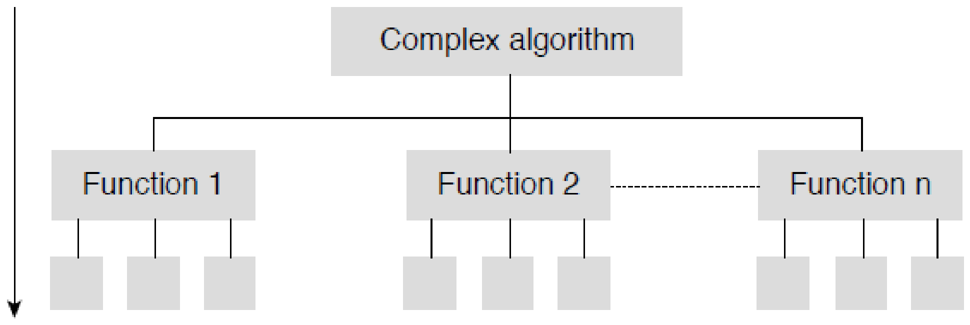
Modularization:

For a complex problem, its algorithm is often divided into smaller units called modules. This process of dividing an algorithm into modules is called as modularization.

Top down design approach:

- A top-down design approach starts by dividing the complex algorithm into one or more functions.
- These functions can further be decomposed into one or more sub-functions, and this process of decomposition is iterated until the desired level of module complexity is achieved.
- Top-down design method is a form of stepwise refinement where we begin with the topmost module and incrementally add functions that it calls.
- Therefore, in a top-down approach, we start from an abstract design and then at each step, this design is refined into more concrete levels until a level is reached that requires no further refinement.

Diagram: Top Down Approach



Bottom up design approach:

- It is just reverse of top down approach.
- In bottom up design, we start with designing the most basic or concrete modules and then proceed towards designing higher level modules.
- The higher level modules are implemented by using the operations performed by lower level modules.
- Thus, in this approach sub-modules are grouped together to form higher level module.
- All the higher level modules are clubbed together to form even higher level modules.
- This process is repeated until the design of the complete algorithm is obtained.

Difference between Top down and Bottom up approach:

Top-Down Approach	**Bottom-Up Approach**
Divides a problem into smaller units and then solve it.	Starts from solving small modules and adding them up together.
This approach contains redundant information.	Redundancy can easily be eliminated.
A well-established communication is not required.	Communication among steps is mandatory.
The individual modules are thoroughly analysed.	Works on the concept of data-hiding and encapsulation.
Structured programming languages such as C use top-down approach.	OOP languages like C++ and Java, etc. uses bottom-up mechanism.
Relation among modules is not always required.	The modules must be related for better communication and work flow.
Primarily used in code implementation, test case generation, debugging and module documentation.	Finds use primarily in testing.

1.1.6 <u>Problem Solving Strategies:</u>

Q.5. Explain the problem solving Strategies?

Ans.

Computer is a very powerful and versatile machine and can do any task given to it provided that the programmer has already fed correct instructions to direct what, how and when the steps have to be done to solve a particular problem. For this he should work to develop a step by step solution to problem. These steps can be given as:

- Clearly define the problem in very simple and precise language.
- Analyse the problem to find out different ways to solve problem and decide the best possible solution.
- Define the selected solution in a detailed step by step manner.

- Write the steps in a particular programming language so that it can be executed by the computer.

The entire software development process is divided into a number of phases where each phase performs a well-defined task. Moreover, the output of one phase provides the input for subsequent phase. The phases in the software development life cycle (SDLC) process are summarized as follows:

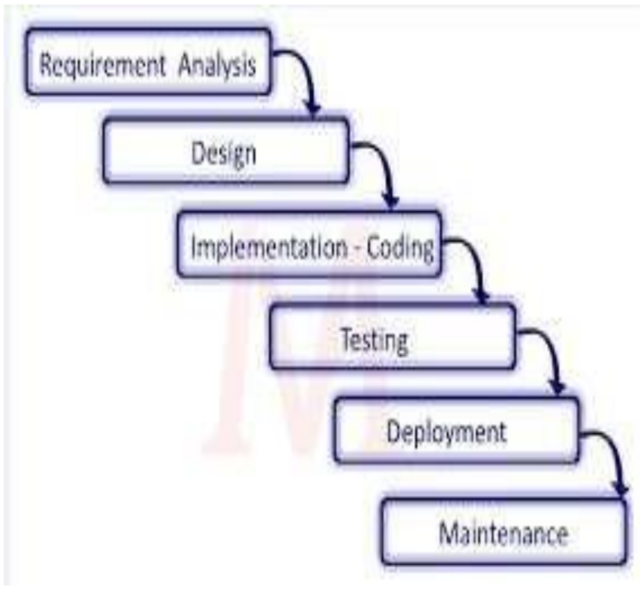


Fig: Software Development Life cycle (SDLC)

- Requirement Analysis: In this phase, the user's expectations are gathered to know why the software has to be built. The gathered requirements are analyzed to decide scope of software. The last activity in this phase includes documenting every identified requirement of the users in order to avoid any doubts regarding functionality of software. The functionality, capability, performance and availability of hardware and software components are all analyzed in this phase.
- Design: The requirements documented in previous phase act as an input to the design phase. In the design phase, core structure of the software is broken down into modules. The solution of the program is specified for each module in the form of algorithms or flowcharts
- Implementation: In this phase, the designed algorithms are converted into program code using any of the high level languages. The particular choice of language will depend on the type of program, such as whether it is system or

application program. This phase is also called as construction phase as the code of the software is generated in this phase. While constructing the code, the development team checks whether the software is compatible with available hardware that are mentioned in requirement specification document.

- Testing: In this phase all modules are tested together to ensure that the overall system works well as a whole product. In this phase the software is tested using a large number of varied inputs, also known as test data, to ensure the software is working as expected by user's requirements.

- Software development, training and support: After the code is tested and the software or the program has been approved by the users, it is installed or deployed in the production environment. In this phase it becomes very crucial to have training classes for users of software.

- Maintenance: Maintenance and enhancement are ongoing activities that are done to cope with newly discovered problems or new requirements. Such activities may take a long time to complete as the requirement may call for the addition of new code that does not fit original design or an extra piece of code, required to fix an unforeseen problem.

1.2 Program Design Tools

1. Algorithm
2. Flowchart
3. Pseudo code

ALGORITHM

It is defined as a sequence of instructions that describe a method for solving a problem. In other words it is a step by step procedure for solving a problem.
An algorithm is precise, step-by-step set of instructions for solving a computational problem.
It is a set of ordered instructions which are written in simple English language.
In this type of problem solving the required input and expected outputs are identified and according to that the processing will be done.
Algorithms help the programmer to write actual logic of a problem on paper and validate it with the help of paper and pencil, and correct it if any fault is noticed.

Characteristics of algorithms:

- **Precision:** The instructions should be written in a precise manner.
- **Uniqueness:** The outputs of each step should be unambiguous, i.e., they should be unique and only depend on the input and the output of the preceding steps.
- **Finiteness:** Not even a single instruction must be repeated infinitely.
- **Effectiveness**: The algorithm should be designed in such a way that it should be the most effective among many different ways to solve a problem.
- **Input:** The algorithm must receive an input. Algorithm may accept zero or more inputs.
- **Output:** After the algorithm gets terminated, the desired result must be obtained.
- **Generality:** The algorithm can be applied to various set of inputs.

Qualities of a good algorithm

The following are the primary factors that are often used to judge the quality of the algorithms.
Time – To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.
Memory – To execute a program, computer system takes some amount of memory space. The lesser is the memory required, the better is the algorithm.
Accuracy – Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, and such algorithms may be suitable.

Example:

Example

Write an algorithm to print „Good Morning"

Step 1: Start
Step 2: Print "Good Morning"
Step 3: Stop

Example: Algorithm to calculate area of circle.

```
Algorithm: Area of circle

 Input: R=Radius
Output:
A=Area
Steps:
Step 1: BEGIN
Step 2: Accept the R
Step3: Find the square of R and store it in A=R*R
Step4: Multiply R with 3.14 and store the result in A
Step5: Display the value of A
Step6: END
```

1. BUILDING BLOCKS OF ALGORITHMS (statements, state, control flow, functions)

Algorithms can be constructed from basic building blocks namely, sequence, selection and iteration.

2.1. Statements:

Statement is a single action in a computer. In a computer statements might include some of the following actions

- input data-information given to the program
- process data-perform operation on a given input
- output data-processed result

2.2. State:

Transition from one process to another process under specified condition with in a time is called state.

2.3. Control flow:

The process of executing the individual statements in a given order is called control flow.

The control can be executed in three ways

1. sequence
2. selection
3. iteration

1.Sequence:

All the instructions are executed one after another is called sequence execution.

Example:

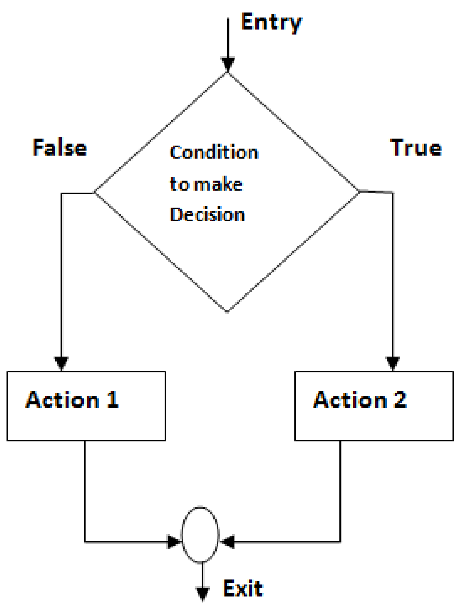
Add two numbers:

Step 1: Start
Step 2: get a , b
Step 3: calculate c=a+b
Step 4: Display c
Step 5: Stop

2.Selection:

A selection statement causes the program control to be transferred to a specific part of the program based upon the condition.

If the conditional test is true, one part of the program will be executed, otherwise it will execute the other part of the program.



Example

Write an algorithm to check whether he is eligible to vote?

```
Step 1: Start
Step 2: Get age
Step 3: if age >= 18 print "Eligible to vote"
Step 4: else print "Not eligible to vote"
Step 6: Stop
```

3.Iteration:

In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called looping or iteration.

Example

Write an algorithm to print all natural numbers up to n

```
Step 1: Start
Step 2: get n value.
Step 3: initialize i=1
Step 4: if (i<=n) go to step 5 else go to step 7
Step 5: Print i value and increment i value by 1
Step 6: go to step 4
Step 7: Stop
```

2.4. Functions:

- ❖ Function is a sub program which consists of block of code (set of instructions) that performs a particular task.
- ❖ For complex problems, the problem is been divided into smaller and simpler tasks during algorithm design.

Benefits of Using Functions

- ❖ Reduction in line of code
- ❖ code reuse
- ❖ Better readability
- ❖ Information hiding
- ❖ Easy to debug and test
- ❖ Improved maintainability

Example:

Algorithm for addition of two numbers using function **Main function()**

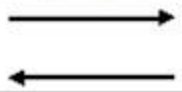
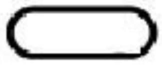
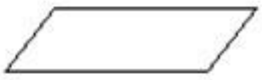
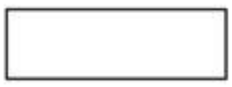
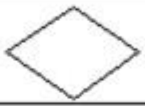
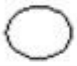
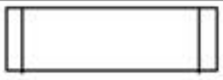
```
Step 1: Start
Step 2: Call the function add ()
Step 3: Stop
```

sub function add()

```
Step 1: Function start
Step 2: Get a, b Values
Step 3: add  c= a+b
Step 4: Print c
Step 5: Return
```

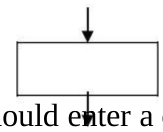
FLOW CHART

Flow chart is defined as graphical representation of the logic for problem solving.
The purpose of flowchart is making the logic of the program clear in a visual representation.

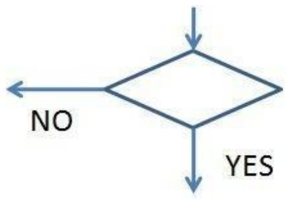
Symbol	Symbol Name	Description
	Flow Lines	Used to connect symbols
	Terminal	Used to start, pause or halt in the program logic
	Input/output	Represents the information entering or leaving the system
	Processing	Represents arithmetic and logical instructions
	Decision	Represents a decision to be made
	Connector	Used to Join different flow lines
	Sub function	used to call function

Rules for drawing a flowchart

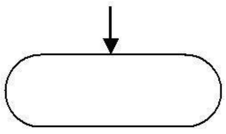
1. The flowchart should be clear, neat and easy to follow.
2. The flowchart must have a logical start and finish.
3. Only one flow line should come out from a process symbol.



4. Only one flow line should enter a decision symbol. However, two or three flow lines may leave the decision symbol.



5. Only one flow line is used with a terminal symbol.



6. Within standard symbols, write briefly and precisely.
7. Intersection of flow lines should be avoided.

Advantages of flowchart:

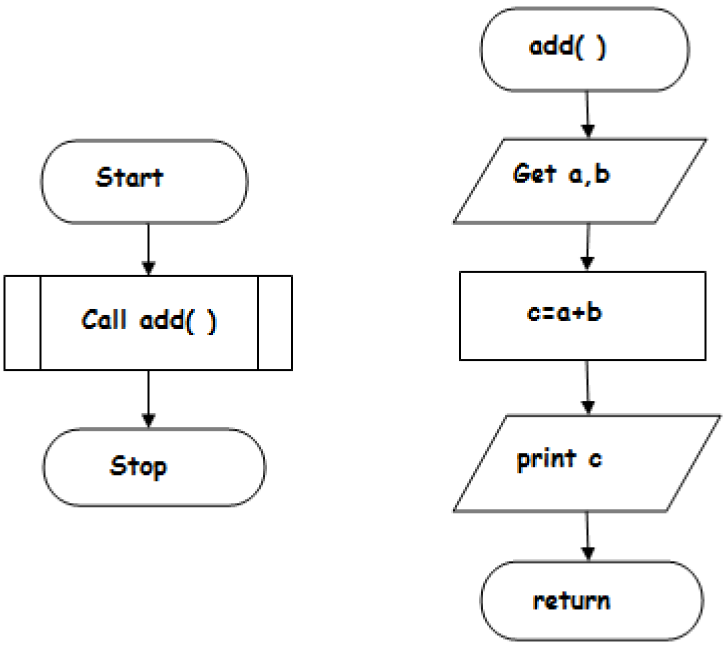
1. **Communication: -** Flowcharts are better way of communicating the logic of a system to all concerned.
2. **Effective analysis: -** With the help of flowchart, problem can be analyzed in more effective way.
3. **Proper documentation: -** Program flowcharts serve as a good program documentation, which is needed for various purposes.
4. **Efficient Coding: -** The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
5. **Proper Debugging: -** The flowchart helps in debugging process.
6. **Efficient Program Maintenance: -** The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of flow chart:

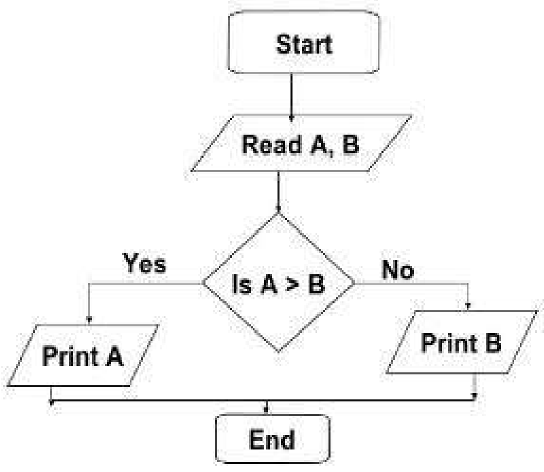
1. **Complex logic: -** Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
2. **Alterations and Modifications: -** If alterations are required the flowchart may require re-drawing completely.
3. **Reproduction: -** As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
4. **Cost: F**or large application the time and cost of flowchart drawing becomes costly.

Example:

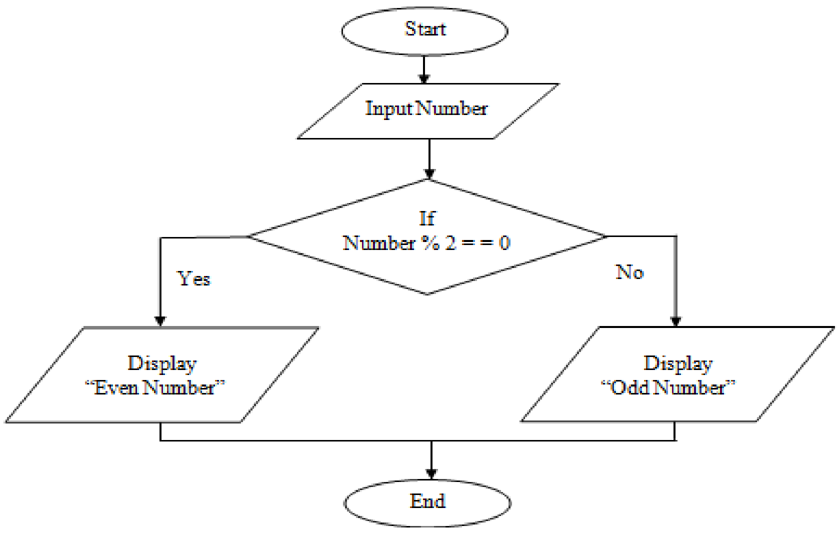
<u>Flowchart for addition of two numbers</u>



Draw a flowchart to find greater number amongst two num0bers.



Draw a flowchart to check whether a given number is even or odd.



PSEUDO CODE

- Pseudo code consists of short, readable and formally styled English languages used for explain an algorithm.
- It does not include details like variable declaration, subroutines.
- It is easier to understand for the programmer or non-programmer to understand the general working of the program, because it is not based on any programming language.
- It gives us the sketch of the program before actual coding.
- It is not a machine readable
- Pseudo code can't be compiled and executed.
- There is no standard syntax for pseudo code.

Guidelines for writing pseudo code:

- Write one statement per line
- Capitalize initial keyword
- Indent to hierarchy
- End multiline structure
- Keep statements language independent

Common keywords used in pseudo code

The following gives common keywords used in pseudo codes.

1. **//:** This keyword used to represent a comment.
2. **BEGIN,END:** Begin is the first statement and end is the last statement.
3. **INPUT, GET, READ**: The keyword is used to inputting data.
4. **COMPUTE, CALCULATE**: used for calculation of the result of the given expression.
5. **ADD, SUBTRACT, INITIALIZE** used for addition, subtraction and initialization**.**
6. **OUTPUT, PRINT, DISPLAY**: It is used to display the output of the program.
7. **IF, ELSE, ENDIF**: used to make decision.
8. **WHILE, ENDWHILE**: used for iterative statements.
9. **FOR, ENDFOR**: Another iterative incremented/decremented tested automatically.

Syntax for if else:	**Example: Greatest of two numbers**

```
IF (condition)THEN statement
    ...
 ELSE
   statement
   ...
 ENDIF
```
	```
	BEGIN READ
	a,b
	IF (a>b) THEN
	DISPLAY a is greater ELSE
	DISPLAY b is greater END IF
	END
	```

Syntax for For:	**Example: Print n natural numbers**

```
FOR( *start-value* to *end-value)* DO
   *statement*
 ...
ENDFOR
```
	```
	BEGIN
	 GET n
	  INITIALIZE i=1
	  FOR (i<=n) DO
	       PRINT i
	     i=i+1
	ENDFOR
	END
	```

Syntax for While:	**Example: Print n natural numbers**

WHILE (condition) DO statement
...
ENDWHILE
	BEGIN
	GET n
	INITIALIZE i=1
	WHILE(i<=n) DO
	    PRINT i
	    i=i+1
	ENDWHILE
	END

Advantages:

- ❖ Pseudo is independent of any language; it can be used by most programmers.
- ❖ It is easy to translate pseudo code into a programming language.
- ❖ It can be easily modified as compared to flowchart.
- ❖ Converting a pseudo code to programming language is very easy as compared with converting a flowchart to programming language.

Disadvantages:

- ❖ It does not provide visual representation of the program's logic.
- ❖ There are no accepted standards for writing pseudo codes.
- ❖ It cannot be compiled nor executed.
- ❖ For a beginner, It is more difficult to follow the logic or write pseudo code as compared to flowchart.

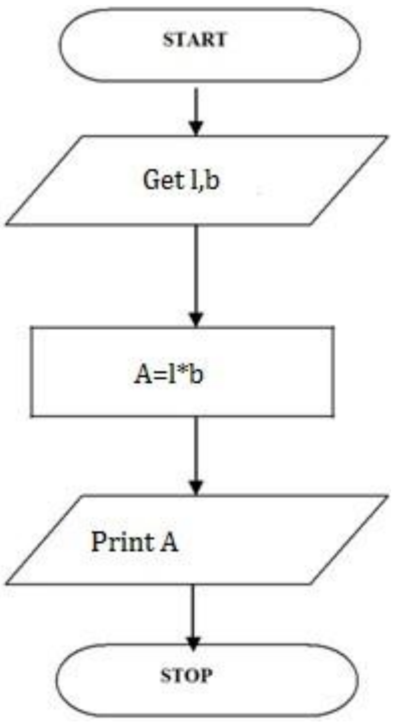
Example:

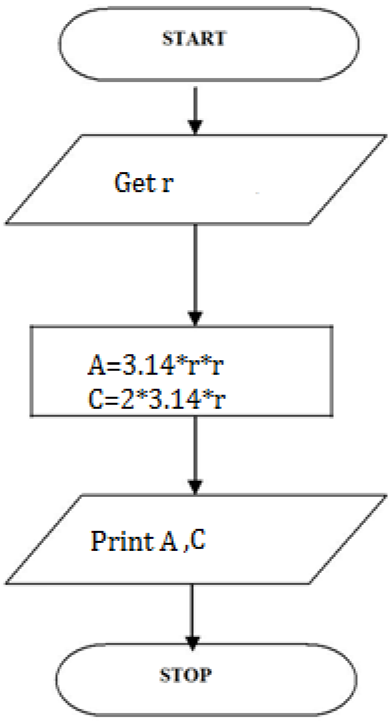
Addition of two numbers:

```
BEGIN
GET a, b ADD
c=a+b PRINT c
END
```

Algorithm	**Flowchart**	**Pseudo code**
An algorithm is a sequence of instructions used to solve a problem	It is a graphical representation of algorithm	It is a language Representation of algorithm.
User needs knowledge to write algorithm.	not need knowledge of program to draw or understand flowchart	Not need knowledge of program language to understand or write a Pseudo code.

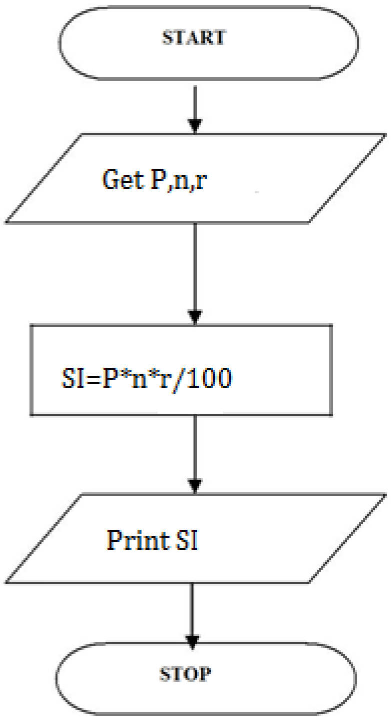
More examples:

Write an algorithm to find area of a rectangle		
Step 1: Start **Step 2:** get l,b values **Step 3:** Calculate A=l*b **Step 4:** Display A **Step 5:** Stop		BEGIN READ l,b CALCULATE A=l*b DISPLAY A END
Write an algorithm for Calculating area and circumference of circle		

Step 1: Start
Step 2: get r value
Step 3: Calculate A=3.14*r*r
Step 4: Calculate C=2.3.14*r
Step 5: Display A,C
Step 6: Stop
	 START
	Get r
	A=3.14*r*r
	C=2*3.14*r
	Print A ,C
	STOP
		BEGIN
		READ r
		CALCULATE A and C
		A=3.14*r*r
		C=2*3.14*r
		DISPLAY A
		END

Write an algorithm for Calculating simple interest

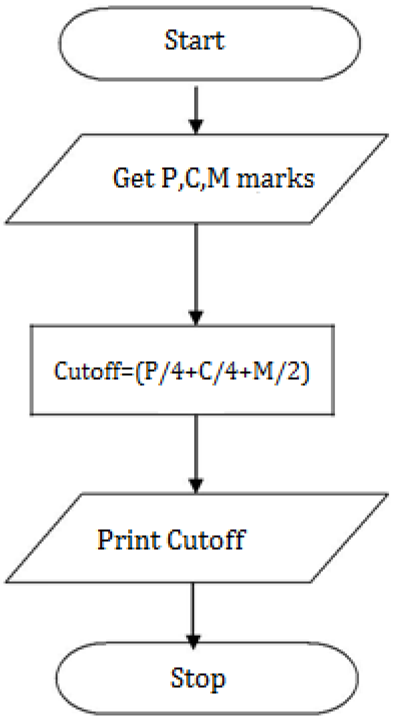
Step 1: Start
Step 2: get P, n, r value
Step3: Calculate SI=(p*n*r)/100
Step 4: Display S
Step 5: Stop



```
BEGIN READ
P, n, r
CALCULATE S
SI=(p*n*r)/100
DISPLAY SI END
```

Write an algorithm for Calculating engineering cutoff

Step 1: Start
Step2: get P,C,M value
Step3:calculate
Cutoff= (P/4+C/4+M/2)
Step 4: Display Cutoff
Step 5: Stop

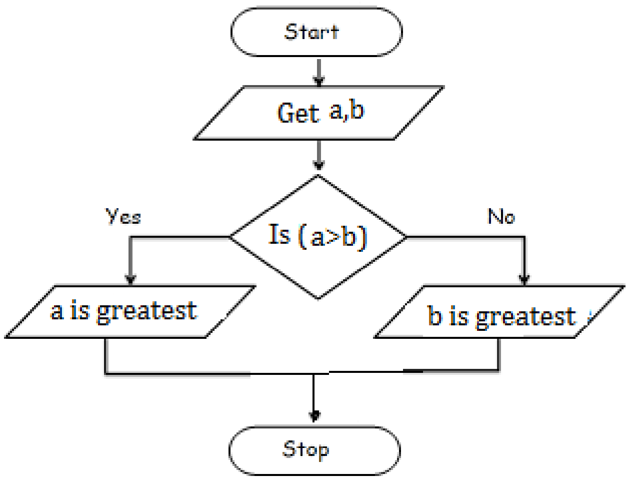


```
BEGIN
READ P,C,M
CALCULATE
Cutoff= (P/4+C/4+M/2)
DISPLAY Cutoff
END
```

To check greatest of two numbers

Step 1: Start
Step 2: get a,b value
Step 3: check if(a>b) print a is greater Step 4: else
b is greater
Step 5: Stop

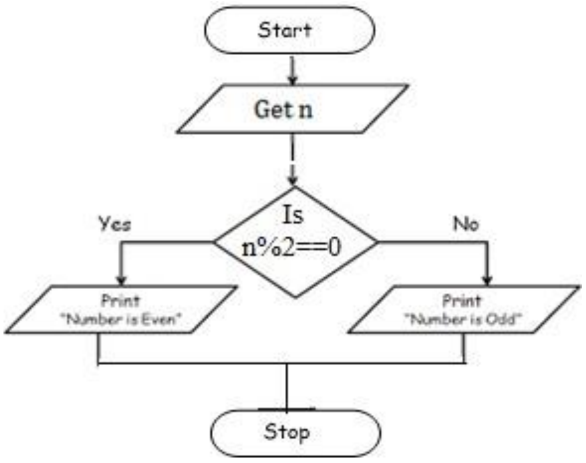
```
BEGIN
READ a,b
IF (a>b) THEN
DISPLAY a is greater
ELSE
DISPLAY b is greater END
IF
END
```



To check leap year or not

Step 1: Start Step
2: get y
Step 3: if(y%4==0) print leap year Step 4: else
print not leap year
Step 5: Stop

```
BEGIN
READ y
IF (y%4==0) THEN
DISPLAY leap year ELSE
DISPLAY not leap year
END IF
END
```



To check positive or negative number

Step 1: Start

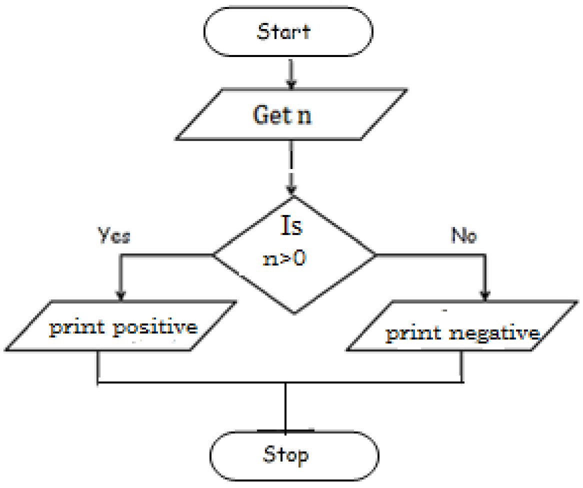
Step 2: get num

Step 3: check if(num>0) print a is positive

Step 4: else num is negative

Step 5: Stop

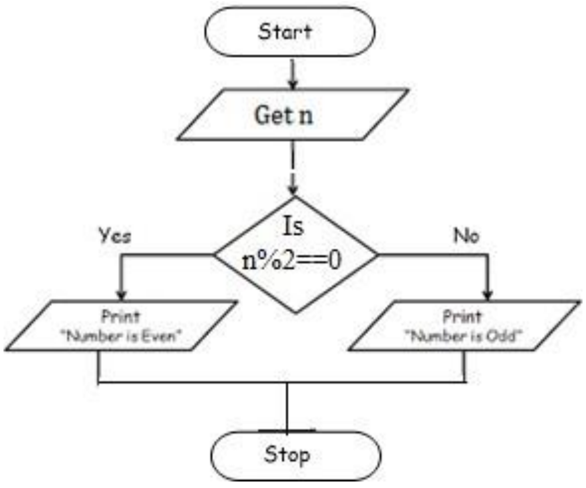
```
BEGIN
READ num
IF (num>0) THEN
DISPLAY num is positive ELSE
DISPLAY num is negative END IF
END
```



To check odd or even number

```
Step 1: Start
Step 2: get  num
Step 3: check if(num%2==0)  print num is even
Step 4: else num is odd
Step 5:  Stop
```

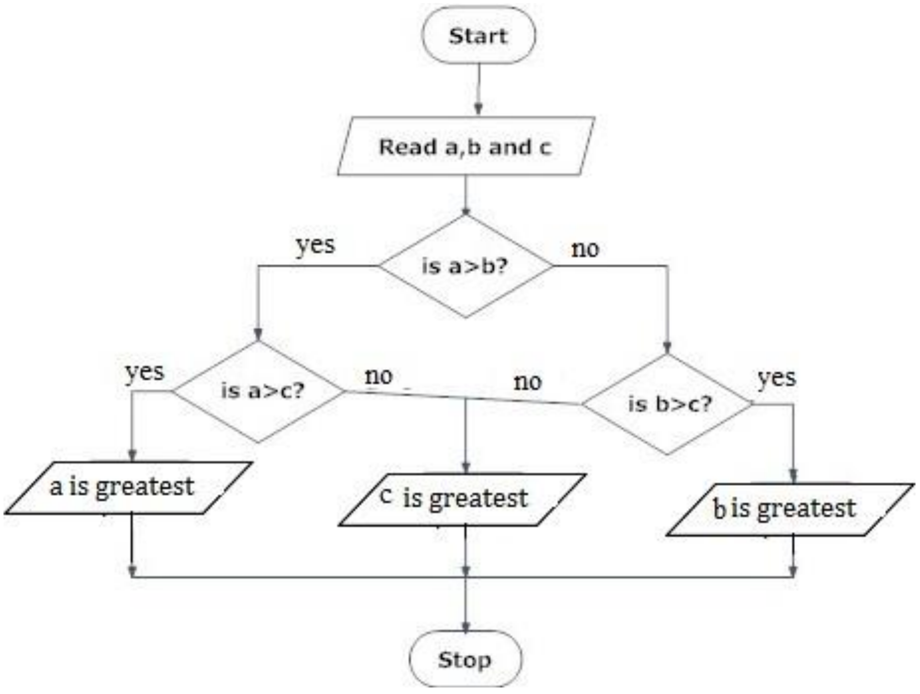
```
BEGIN
READ num
IF (num%2==0) THEN
DISPLAY num is even
ELSE
DISPLAY num is odd END
IF
END
```



To check greatest of three numbers

```
Step1: Start
Step2: Get A, B, C
Step3: if(A>B) goto Step4 else goto step5
Step4: If(A>C) print A else print C
Step5: If(B>C) print B else print C
Step6: Stop
```

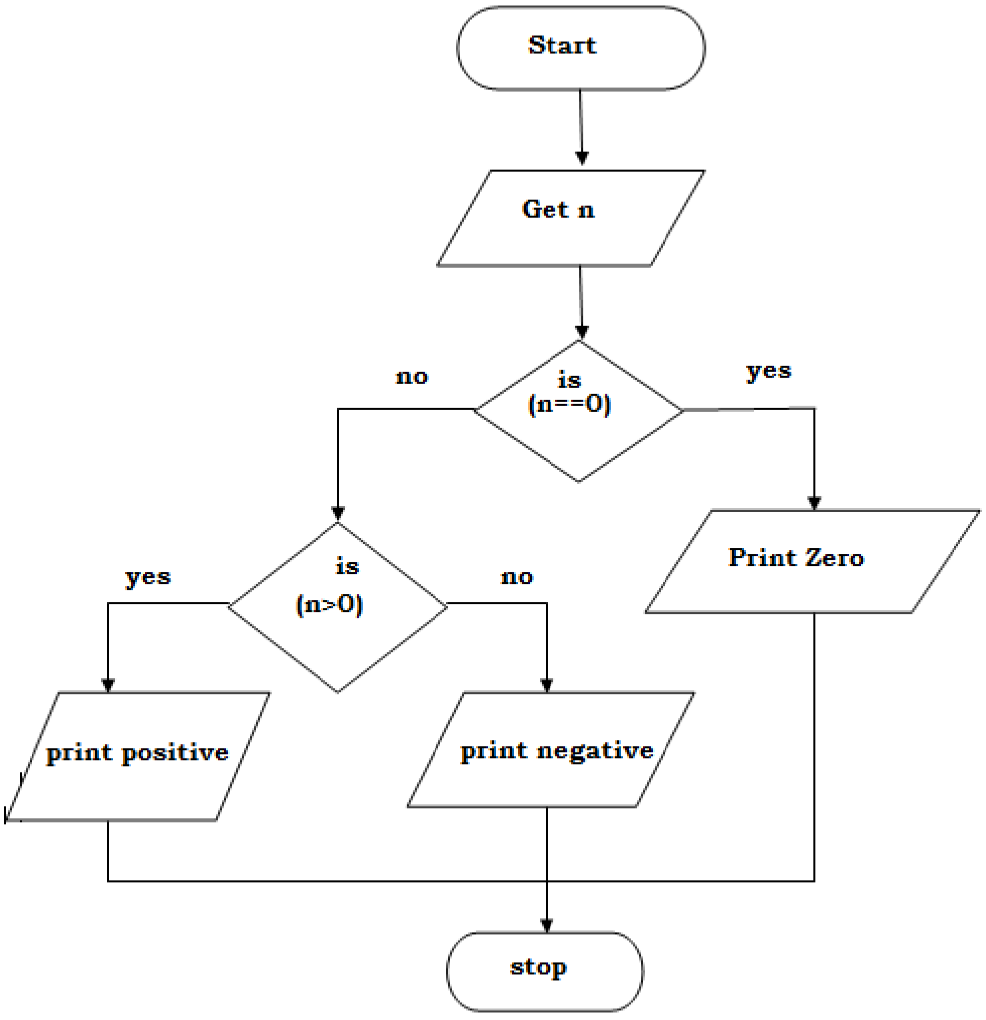
```
 BEGIN
 READ a, b, c
IF (a>b) THEN
     IF(a>c) THEN
          DISPLAY a is greater ELSE
           DISPLAY c is greater END
      IF
 ELSE
     IF(b>c) THEN
           DISPLAY b is greater ELSE
           DISPLAY c is greater END
     IF
 END IF
 END
```



Write an algorithm to check whether given number is +ve, -ve or zero.

Step 1: Start
Step 2: Get n value.
Step 3: if (n ==0) print “Given number is Zero” Else goto step4
Step 4: if (n > 0) then Print “Given number is +ve”
Step 5: else Print “Given number is -ve”
Step 6: Stop

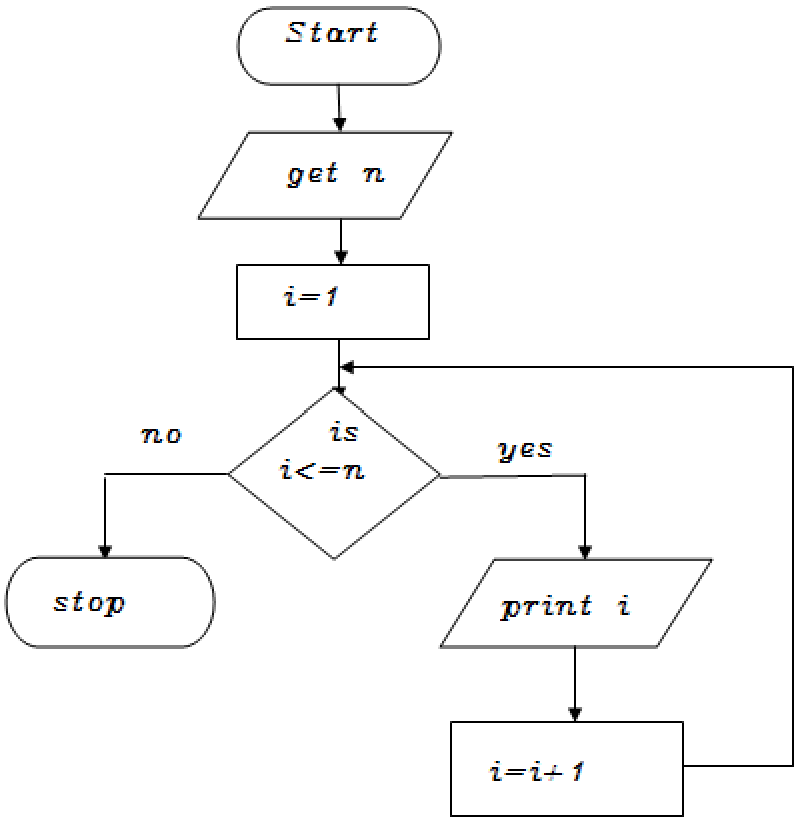
```
BEGIN
GET n
IF(n==0) THEN
    DISPLAY " n is zero"
ELSE
   IF(n>0) THEN
         DISPLAY "n is positive" ELSE
         DISPLAY "n is positive" END
    IF
END IF
END
```



Write an algorithm to print all natural numbers up to n

Step 1: Start
Step 2: get n value.
Step 3: initialize i=1
Step 4: if (i<=n) go to step 5 else go to step 8
Step 5: Print i value
step 6 : increment i value by 1
Step 7: go to step 4
Step 8: Stop

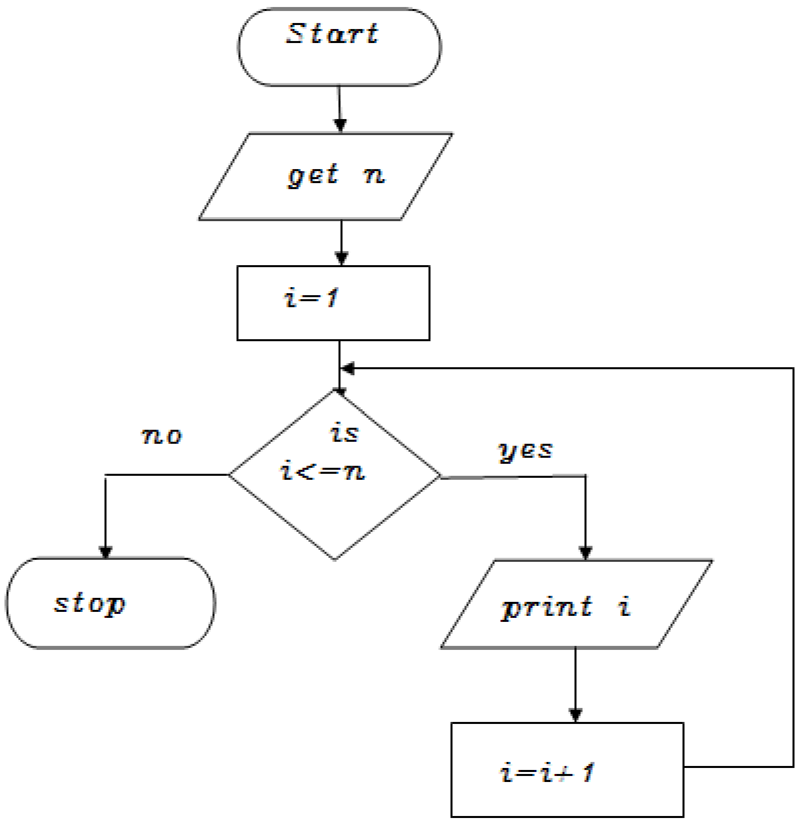
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i
     i=i+1
ENDWHILE
END
```



Write an algorithm to print n odd numbers

Step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check if(i<=n) goto step 5 else goto step 8
step 5: print i value
step 6: increment i value by 2
step 7: goto step 4
step 8: stop

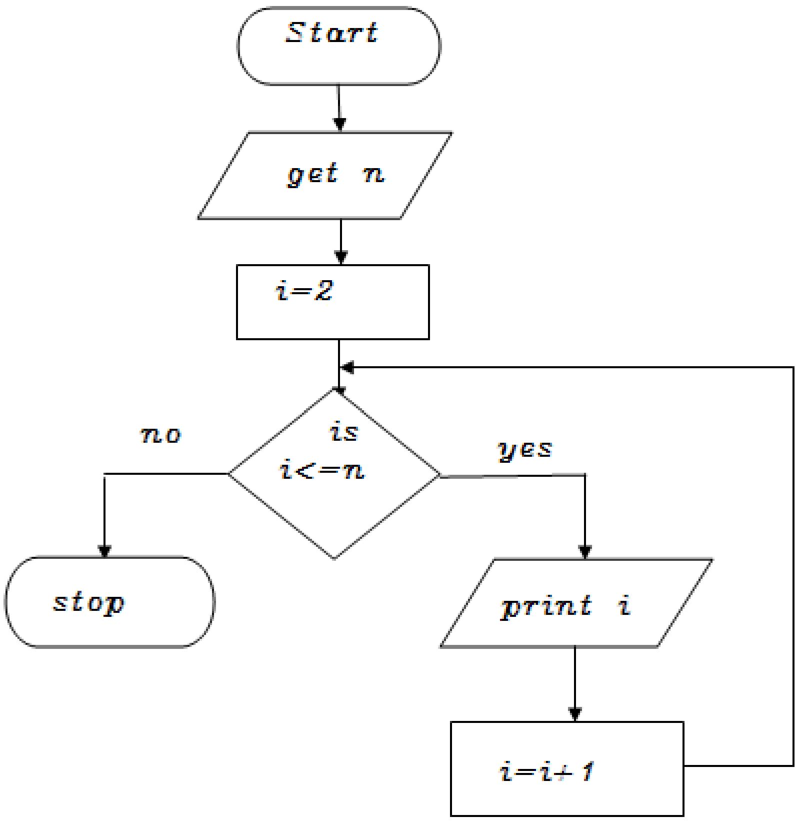
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i
     i=i+2
ENDWHILE
END
```



Write an algorithm to print n even numbers

```
Step 1: start
step 2: get n value
step 3: set initial value i=2
step 4: check if(i<=n) goto step 5 else goto step8
step 5: print i value
step 6: increment i value by 2
step 7: goto step 4
step 8: stop
```

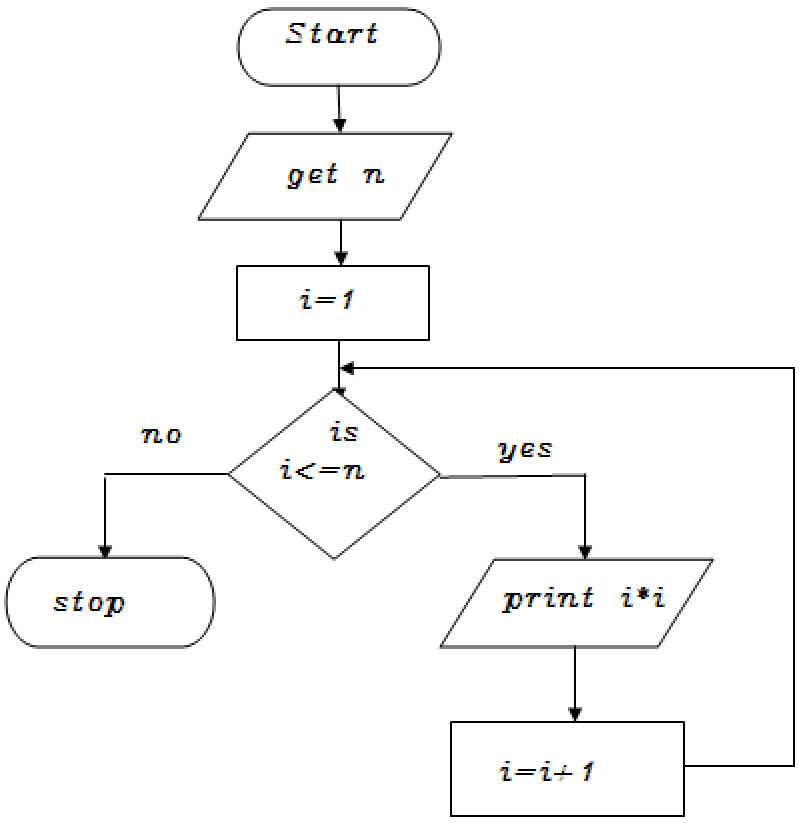
```
BEGIN
GET n
INITIALIZE i=2
WHILE(i<=n) DO
    PRINT i
     i=i+2
ENDWHILE
END
```



Write an algorithm to print squares of a number

Step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: print i*i value
step 6: increment i value by 1
step 7: goto step 4
step 8: stop

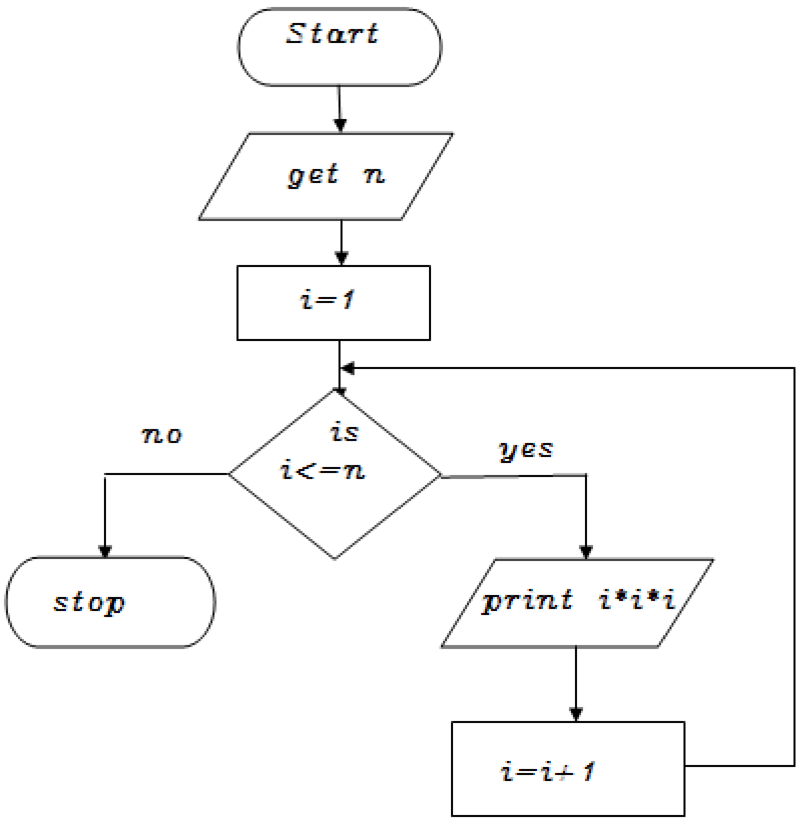
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i*i
    i=i+2
ENDWHILE
END
```



Write an algorithm to print to print cubes of a number

step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: print i*i *i value
step 6: increment i value by 1
step 7: goto step 4
step 8: stop

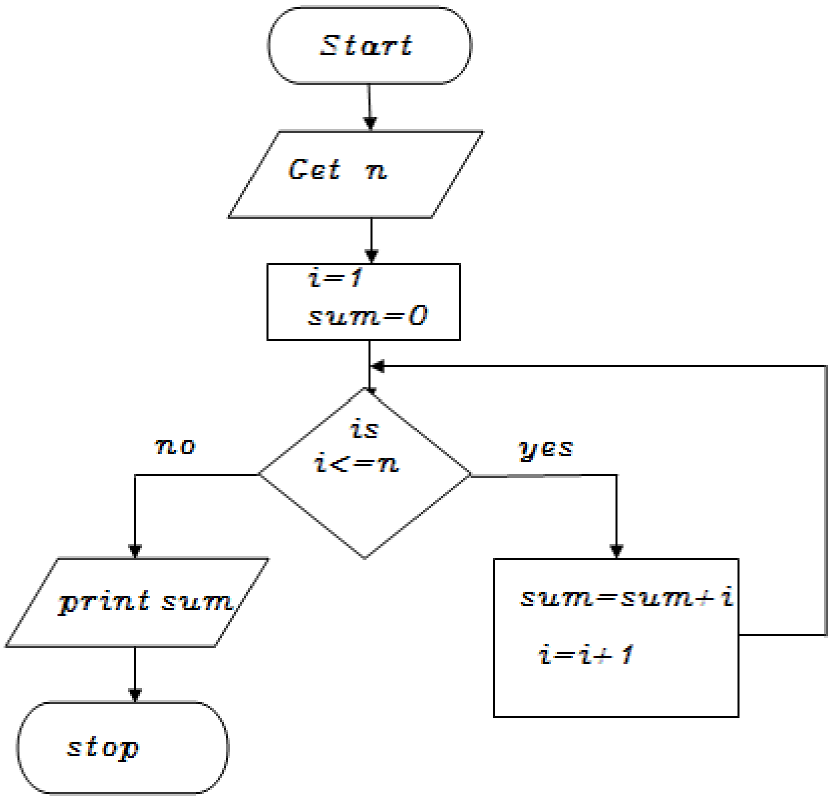
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i*i*i
    i=i+2
ENDWHILE
END
```



Write an algorithm to find sum of a given number

Step 1: start
step 2: get n value
step 3: set initial value i=1, sum=0
Step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: calculate sum=sum+i
step 6: increment i value by 1
step 7: goto step 4
step 8: print sum value
step 9: stop

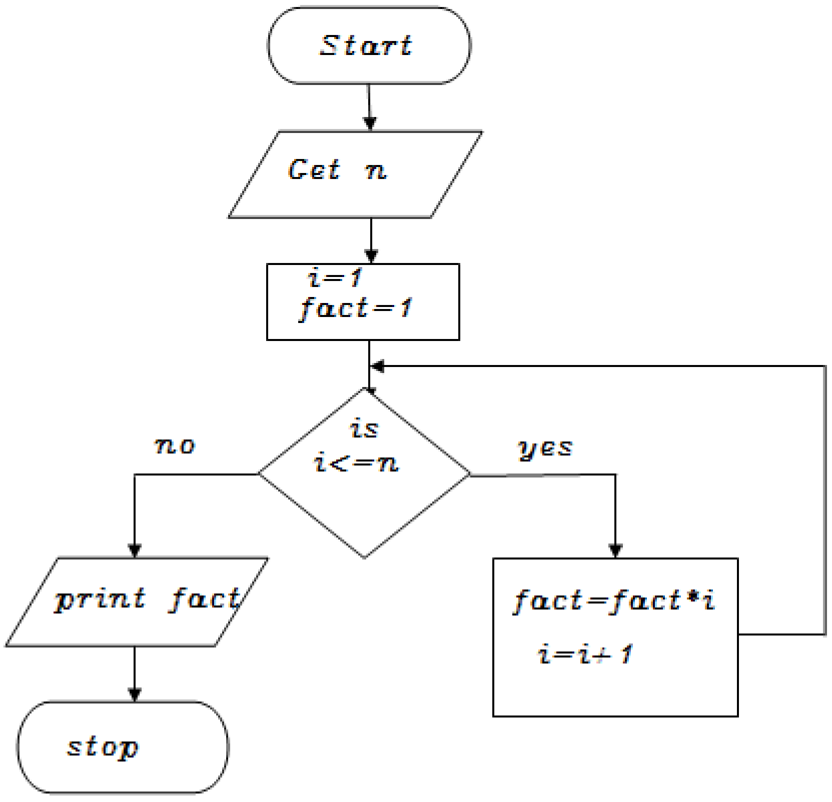
```
BEGIN
GET n
INITIALIZE i=1,sum=0
WHILE(i<=n) DO
    sum=sum+i
    i=i+1
ENDWHILE
PRINT sum
END
```



Write an algorithm to find factorial of a given number

step 1: start
step 2: get n value
step 3: set initial value i=1, fact=1
step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: calculate fact=fact*i
step 6: increment i value by 1
step 7: goto step 4
step 8: print fact value
step 9: stop

```
BEGIN
GET n
INITIALIZE i=1,fact=1
WHILE(i<=n) DO
    fact=fact*i
    i=i+1
ENDWHILE
PRINT fact
END
```



1.3 Features of Python

Q.15. Explain various features of python.
Ans.

1. **Simple:** Python is simple and small language. Reading a program written in python feels like reading English.
2. **Easy to Learn**: Python programming is clearly defined and easily readable.
3. **Versatile:** Python supports wide variety range of applications ranging from text processing to Web applications
4. **Free and Open Source**: Python software is freely available on internet
5. **High Level Language**: Programmers don't have to worry about low level details like managing memory used by program etc.
6. **Interactive**: Programs in python works in interactive mode
7. **Portable (Platform Independent):** Python is portable language. Program can work similarly on different platform like window, Linux, Macintosh, Solaris etc.
8. **Object Oriented:** Python supports Object oriented style of programming like C++, Java languages. It support features like C class, Objects, Inheritance etc.
9. **Interpreted:** Python is processed at run time by the interpreter. So no need to compile program before executing it.
10. **Scalability:** Python support modular programming, so programmer can add module at any stage of the program
11. **Secure:** Python programming environment is secure from tempering

1.4 History and Future of Python

- Python is a general purpose interpreted interactive object oriented and high level programming language.
- It was first introduced in 1991 by Guido van Rossum, a Dutch computer programmer.

  The language places strong emphasis on code reliability and simplicity so that the programmers can develop applications rapidly
- Python is multi-paradigm programming language, which allows user to code in several different programming styles.
- Python supports cross platform development and is available through open source. Python is widely used for scripting in Game menu applications effectively.

History of Python:

- Python is created by **Guido Van Rossum in the 1980s.**
- Rossum published the first version of Python code (0.9.0) in February **1991 at the CWI (Centrum Wiskunde & Informatics) in the Netherlands , Amsterdam.**
- Python is derived from **ABC programming language, which is a general-purpose programming language that had been developed at the CWI.**
- Rossum chose the name "**Python", since he was a big fan of Monty Python's Flying Circus.**
- Python is now maintained by a core development team at the institute, although Rossum still holds a vital role in directing its progress.

Future of Python:

- Python's user base is vast and growing – it's not going away any time soon.
- Utilized by the likes of Nokia, Google, and even NASA for it's easy syntax, it looks to have a bright future ahead of it supported by a huge community of OS developers.
- Its support of multiple programming paradigms, including object-oriented Python programming, functional Python programming, and parallel programming models makes it a highly adaptive choice – and its uptake keeps growing.

1.5 Writing and Executing Python Programs

- Open IDLE.
- Go to **File** > **New**.
- Write Python code in the file.
- Then save the file with .py extension. For example, hello.py, example.py etc.

  You can give any name to the file. However, the file name should end with **.py**
- Run the program.

Python Keywords

Keywords are the reserved words in Python.

We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive.

There are 33 keywords in Python 3.7 and 35 keywords in Python 3.11. This number can vary slightly in the course of time. All the keywords except True, False and none are in lowercase and they must be written as it is. The list of all the keywords is given below.

None	break	except	in	raise
False	await	else	import	pass
and	continue	for	lambda	try
True	class	finally	is	return
as	def	from	nonlocal	while
async	elif	if	not	with
assert	del	global	or	yield

Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

Rules for writing identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _. Names like myClass, var_1 and print_this_to_screen, all are valid example.
- An identifier cannot start with a digit. 1abc is invalid, but abc1 is perfectly fine.
- Keywords cannot be used as identifiers

Example

Abc

_abc

abc@abc ------wrong

abc xyz ------ wrong

Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.

```
if (a > b ):

        print(“ a is greater”)            #indention

else:

        print(“ b is greater”)
```

The enforcement of indentation in Python makes the code look neat and clean. This resultsinto Python programs that look similar and consistent.

Indentation can be ignored in line continuation. But it's a good idea to always indent. Itmakes the code more readable.

```
if (a > b ): print(“ a is greater”)      #valid but use above style..
```

Python Comments

Comments are very important while writing a program. It describes what's going on inside a program. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment

In Python, we can use following comments style.

- Single Line Comment – python use the hash (#) symbol to start single line acomment.

```
#This is simple python program

#Programmer Name : Pankaj
```

- Multi Line Commnet – Python uses triple quote (‘ ‘ ‘) to start and to end multi linecomment.

```
‘ ‘ ‘ Hi This is john
Writing python program for
Addition of two numbers
‘ ‘ ‘
```

Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,

number = 10

Here, we have created a named number. We have assigned value 10 to the variable.

You can think variable as a bag to store books in it and those books can be replaced at anytime.

number = 10

number = 1.1

Initially, the value of number was 10. Later it's changed to 1.1.

Literals

Literal is a raw data given in a variable or constant. In Python, there are various types ofliterals they are as follows:

- **Numeric Literals:** Numeric Literals are immutable (unchangeable). Numeric literalscan belong to 3 different numerical types Integer, Float, and Complex.

```
a = 0b1010 #Binary Literalsb

= 100 #Decimal Literal

c = 0o310 #Octal Literal

d = 0x12c #Hexadecimal Literal

#Float Literal

float_1 = 10.5

float_2 = 1.5e2

#Complex Literal

x = 3.14j
```

- **String literals :** A string literal is a sequence of characters surrounded by quotes. Wecan use both single, double or triple quotes for a string. And, a character literal is asingle character surrounded by single or double quotes.

  Strings = "This is Python"
  char = "C"

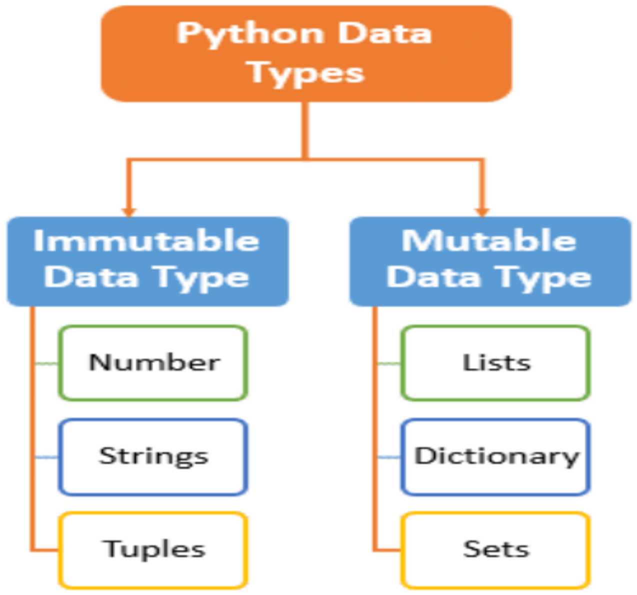
multiline_str = """This is a multiline string with more than one line code."""

- **Boolean literals :** A Boolean literal can have any of the two values: True or False.x = (1 == True)

  y = (1 == False)

- **Special literals :** Python contains one special literal i.e. None. We use it to specifyto that field that is not created.
  drink = "Available"
  food = None

Python Data Types



- **Python Numeric:** Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python. We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

  Example:

  A=10   #python integer

  A= 10.50 #python floating point number

- **Python Strings:** String is sequence of Unicode characters. We can use single quotes or doublequotes to represent strings.

  Multi-line strings can be denoted using triple quotes, ''' or """.Example:

  A=”VPKBIET”

- **Python List :** List is an ordered sequence of items. It is one of the most used data type in Python and is very flexible. All the items in a list do not need to be of the same type.

  Example

  a = [5,10,15,20,25,30,35,40]

  #Displaying number at Second Position

  print("a[2] = ", a[2])

- **Python Tuple:** Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated by commas.
  Example:
  t = (5,'program', 1+3j)
  # Displaying element from tuple
  print("t[1] = ", t[1])

- **Python Dictionary:** Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key: value. Key and value can be of any type.

  Example
  Ex: a = {‘name’:’Bob’,’Age’: 21}

- **Python Set :** Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

  Example

  a = {5,2,3,1,4}

  # printing set variable

  print("a = ", a)

(OR)

DATA TYPES IN PYTHON

Every value in Python has a data type. Since everything is an object in Python programming, data types are actually classes, and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listedbelow

- Python Numbers
- Python List
- Python Tuple
- Python Strings
- Python Set
- Python Dictionary

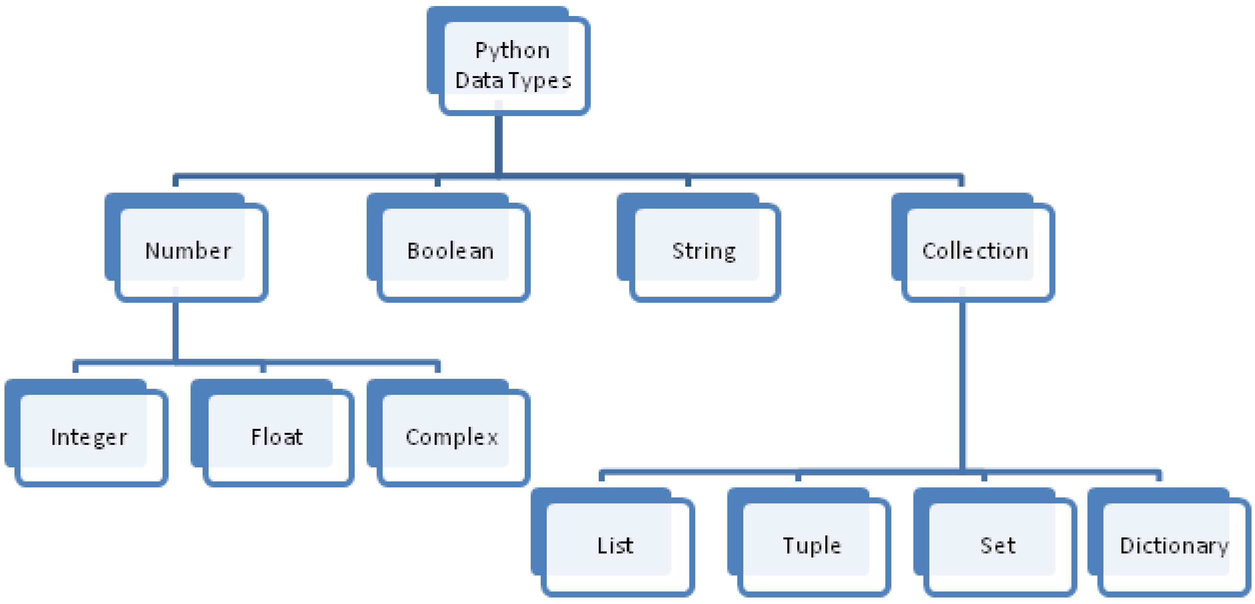


Chart : Python Data Types

1. Python Data Type – Numeric

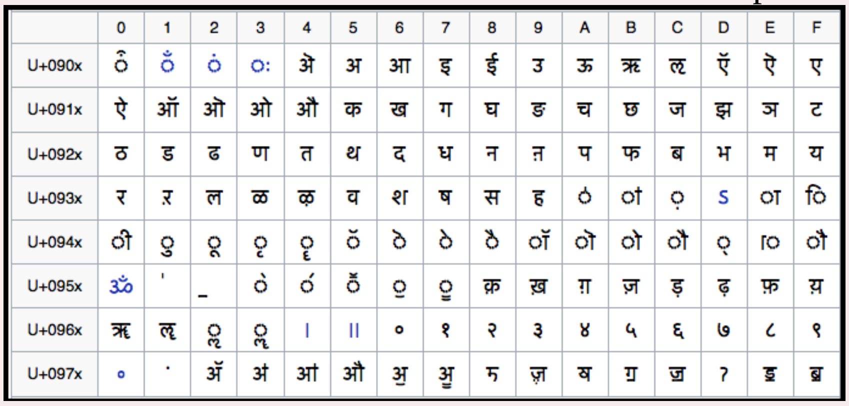
Python numeric data type is used to hold numeric values like;

Data Type	Use
Int	holds signed integers of non-limited length.
Long	holds long integers(exists in Python 2.x,
Float	holds floating precision numbers and it’s
complex	holds complex numbers.

2. Python Data Type – String

String is a **sequence of characters**. Python supports Unicode characters.Generally strings are represented by either single or double quotes.

```
>>> s1 = "This is a string"
>>> s2= '''a multiline String'''
```

Single LineString	“hello world”
Multi Line String	“”””Gwalior Madhya Pradesh””””
Raw String	r"raw \n string" [Used when we want to have a string that contains backslash and don’t want it to be treated as an escape character.]
Character	"C" [Single letter]
Unicode string	u"\u0938\u0902\u0917\u0940\u0924\u093E" will print 'सगं ◌ीत◌ा' 

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+090x	◌ऀ	◌ँ	◌ं	◌ः	ऄ	अ	आ	इ	ई	उ	ऊ	ऋ	ऌ	ऍ	ऎ	ए
U+091x	ऐ	ऑ	ऒ	ओ	औ	क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
U+092x	ठ	ड	ढ	ण	त	थ	द	ध	न	ऩ	प	फ	ब	भ	म	य
U+093x	र	ऱ	ल	ळ	ऴ	व	श	ष	स	ह	◌ऺ	◌ऻ	◌़	ऽ	◌ा	◌ि
U+094x	◌ी	◌ु	◌ू	◌ृ	◌ॄ	◌ॅ	◌ॆ	◌े	◌ै	◌ॉ	◌ॊ	◌ो	◌ौ	◌्	◌ॎ	◌ॏ
U+095x	ॐ	◌॑	◌॒	◌॓	◌॔	◌ॕ	◌ॖ	◌ॗ	क़	ख़	ग़	ज़	ड़	ढ़	फ़	य़
U+096x	ॠ	ॡ	◌ॢ	◌ॣ	।	॥	०	१	२	३	४	५	६	७	८	९
U+097x	॰	ॱ	ॲ	ॳ	ॴ	ॵ	ॶ	ॷ	ॸ	ॹ	ॺ	ॻ	ॼ	ॽ	ॾ	ॿ

3. Python Data Type – List

List is a versatile data type exclusive in Python. In a sense it is same as array inC/C++. But interesting thing about list in Python is it can simultaneously hold different type of data.

Formally list is a ordered sequence of some data written using squarebrackets ([]) and commas(,)

```
my_list = []
# empty list

my_list = [1, 2, 3]
# list of integers

my_list = [1, "Hello", 3.4]
# list with mixed data types

Also, a list can even have another list as an item. This is called
nested list.
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

4. Python Tuple

Tuple is an ordered sequences of items same as list. The only difference is thattuples are immutable. Tuples once created cannot be modified.

Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items areseparated by commas

>>> t = (50,'Learning is fun', 1+3j, 45.67) # Can store mixed data types

Advantages of Tuple over List

➢ **Since tuple are immutable, iterating through tuple is faster than with list. Sothere is a slight performance boost.**
➢ **We generally use tuple for heterogeneous (different) datatypes and list forhomogeneous (similar) datatypes.**
➢ **Tuples that contain immutable elements can be used as key for a dictionary.With list, this is not possible.**
➢ **If you have data that doesn't change, implementing it as tuple will guaranteethat it remains write-protected.**

5. Python Set

Set is an unordered collection of unique items. Set is defined by values
separated by comma inside braces { }.Items in a set are not ordered.

a = {5,2,3,1,4}

printing set variable

print("a = ", a)

data type of variable a

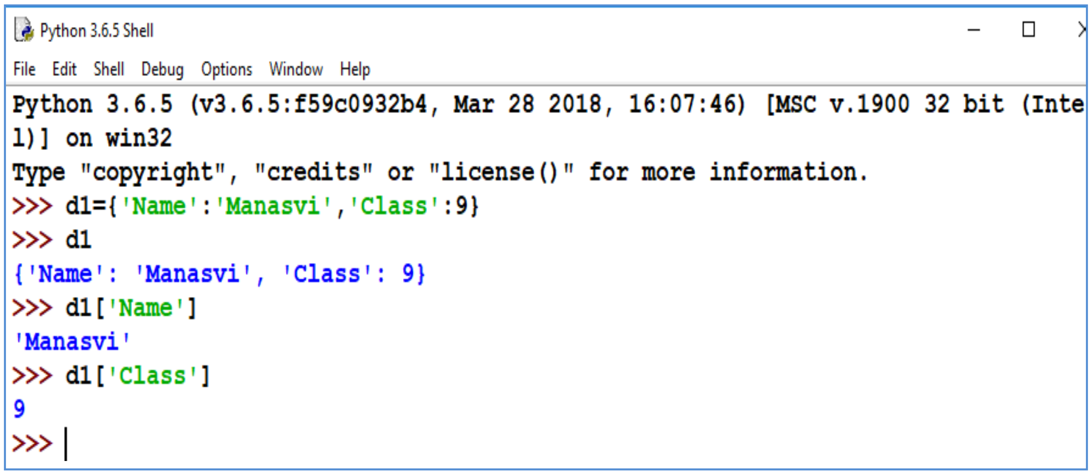
print(type(a))   # **<class 'set'>**

6. Python Dictionary

Dictionary is an unordered collection of key-value pairs.

It is generally used when we have a huge amount of data. Dictionaries areoptimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within Curly braces {} with each item beinga pair in
the form **key: value**.
Key and value can be of any type.



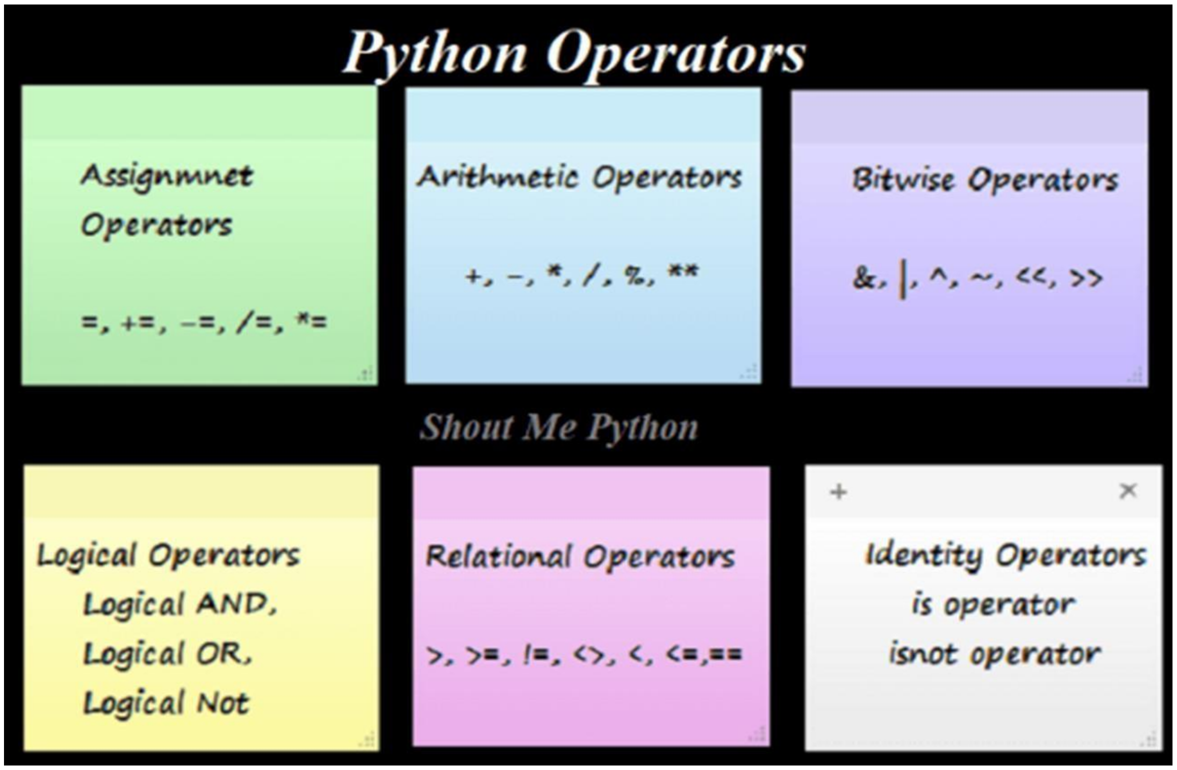
```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> d1={'Name':'Manasvi','Class':9}
>>> d1
{'Name': 'Manasvi', 'Class': 9}
>>> d1['Name']
'Manasvi'
>>> d1['Class']
9
>>> |
```

Python Operators

Operators are special symbols in Python that carry out arithmetic or logical computation.The value that the operator operates on is called the operand.

E.g. c = a + b

Here '+' is used to add two numbers a and b … '=' is used to store result in c.



Arithmetic Operators

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to thepower y)

Comparison operators: Outputs only two value True or False

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Logical operators

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Bitwise operators : Bitwise operators act on operands as if they were string of binarydigits. It operates bit by bit, hence the name.

For example, 2 is 0010 in binary and 7 is 0111.In

the table below:

Let x = 10 (0000 1010 in binary)   and   y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Assignment operators:

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5

Identity operators: is and is not are the identity operators in Python. They are used to checkif two values (or variables) are located on the same part of the memory. Two variables thatare equal does not imply that they are identical.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

```
x1 = 5

y1 = 5

# Output: False

print(x1 is not y1)

# Output: True

print(x2 is y2)
```

Membership operators: in and not in are the membership operators in Python. They areused to test whether a value or variable is found in a sequence.

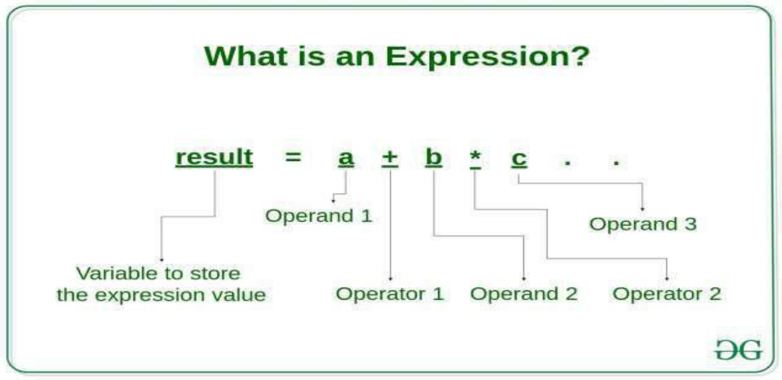
Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

```
x = 'Hello world'

y = {1:'a',2:'b'}

# Output: True

print('H' in x)

# Output: True

print('hello' not in x)
```

Expressions in Python

✓ An expression is any valid combination of symbols like variables, constants and operators that represents a value.

- ✓ In python, an expression must have at least one operand and can have one or more operators.
- ✓ An expression is a combination of operators and operands that is interpreted to
- ✓ produce some other value.
- ✓ An expression is a combination of operators, constants and variables. An
- ✓ expression may consist of one or more operands, and zero or more operators to produce a value.



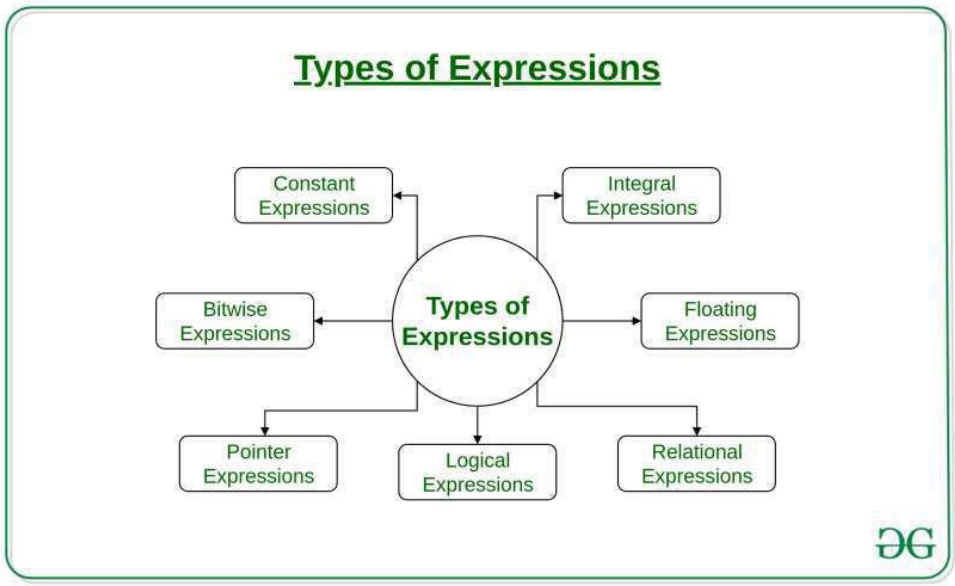
- ✓ An Example of valid expression:
  - A+B*C-5
  - X=A/B
  - Y=A^B

- ✓ An example of invalid expression:
  - A+
  - A*

- ✓ Python supports different types of expressions can be classified as:

1. Based on the position of operators in an expression:
   a. Infix expression:
      - ✓ Example: a=**b-c**
   b. Prefix expression:
      - ✓ Example: a=**+bc**
   c. Postfix expression:
      - ✓ Example: a=**bc+**

2. Based on the data type of the result obtained on evaluating an expression:



a. Constant expression:
    - ✓ Example: 8+2-3

b. Integral expression:
    - ✓ Example: a=10

c. Relational expression:
    - ✓ Example: c=a>b

d. Logical expression:
    - ✓ Example: a>b and a>c