Programming in Python-I
Unit-I

By:
Mani Butwall,
Asst. Prof. (CSE)

Contents

- Introduction to Programming Fundamentals
- Programming Environment
- Principles of Programming
- What is Debugging ?
- Text editors and Debuggers
- Introduction to Flow-Chart
- Algorithm

What is Computer Programming?

- Program is a collection of instructions that can be executed by a computer to perform a specific task.
- **COMPUTER PROGRAMMING** is a step by step **process** of designing and developing various sets of computer programs to accomplish a specific computing outcome.
- The process comprises several tasks like analysis, coding, algorithm generation, checking accuracy and resource consumption of algorithms, etc.
- The purpose of computer programming is to find a sequence of instructions that solve a specific problem on a computer.
- Computer Programming is very easy if it is appropriately managed. There are many computer programming languages available so finalizing the right programming language is not an easy task.

Basic of programming

- English is the most popular and well-known Human Language. The English language has its own set of grammar rules, which has to be followed to write in the English language correctly.
- Likewise, any other Human Languages (German, Spanish, Russian, etc.) are made of several elements like nouns, adjective, adverbs, propositions, and conjunctions, etc. So, just like English, Spanish or other human languages, programming languages are also made of different elements.
- Just like human languages, programming languages also follow grammar called **syntax**. There are certain basic program code elements which are common for all the programming languages.

- Most important basic elements for programming languages are:

1. Programming Environment
2. Data Types
3. Variables
4. Keywords
5. Logical and Arithmetical Operators
6. If else conditions
7. Loops
8. Numbers, Characters and Arrays
9. Functions
10. Input and Output Operations

Applications of computer programming languages

Python	Web and Internet Development, Scientific and Numeric applications, Desktop GUIs, Business applications. It is widely used in AI and Machine Learning space.
Java	Mostly used for developing Android apps, web apps, and Big data.
R	Data Science projects, Statistical computing, Machine learning
Javascript	JavaScript usage include web/mobile app development, game development, and desktop app development.
C++	C++ is widely used in Game Development, Advance Computations, and Graphics Compilers
C#	Widely used in Enterprise Cross-Applications Development, Web Applications
PHP	Web Development, Content Management Systems, eCommerce Applications
SQL	Used in Any Database

Types of Programming Languages

- There are two **types of programming languages**, which can be categorized into the following ways:

1. **Low level language**
    a) Machine language
    b) Assembly language
2. **High level language**
    a) Procedural-Oriented language
    b) Object Oriented Language
    c) Scripting Language
    d) Natural language

1. Low level language

- This language is the most understandable language used by computer to perform its operations. It can be further categorized into:

a) Machine Language (1GL)

- Machine language consists of strings of binary numbers (i.e. 0s and 1s) and it is the only one language, the processor directly understands. Machine language has an Merits of very fast execution speed and efficient use of primary memory.
- **Merits:**
- It is directly understood by the processor so has faster execution time since the programs written in this language need not to be translated.
- It doesn't need larger memory.
- **Demerits:**
- It is very difficult to program using 1GL since all the instructions are to be represented by 0s and 1s.
- Use of this language makes programming time consuming.
- It is difficult to find error and to debug.
- It can be used by experts only.

- **b) Assembly Language**
- Assembly language is also known as low-level language because to design a program programmer requires detailed knowledge of hardware specification. This language uses mnemonics code (symbolic operation code like 'ADD' for addition) in place of 0s and 1s. The program is converted into machine code by assembler. The resulting program is referred to as an object code.
- **Merits:**
- It is makes programming easier than 1GL since it uses mnemonics code for programming. Eg: ADD for addition, SUB for subtraction, DIV for division, etc.
- It makes programming process faster.
- Error can be identified much easily compared to 1GL.
- It is easier to debug than machine language.
- **Demerits:**
- Programs written in this language is not directly understandable by computer so translators should be used.
- It is hardware dependent language so programmers are forced to think in terms of computer's architecture rather than to the problem being solved.
- Being machine dependent language, programs written in this language are very less or not portable.
- Programmers must know its mnemonics codes to perform any task.

- ADD A,B A=20 B=30
- SUB A,B
- DIV A,B
- MUL A,B

2. High level language

- Instructions of this language closely resembles to human language or English like words. It uses mathematical notations to perform the task.
- The high level language is easier to learn. It requires less time to write and is easier to maintain the errors.
- The high level language is converted into machine language by one of the two different languages translator programs; **interpreter or compiler.**
-
- High level language can be further categorized as:

- **a) Procedural-Oriented language (3GL)**
- Procedural Programming can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure.
- Procedures, also known as routines, subroutines or functions, simply consist of a series of computational steps to be carried out.
- Follows top-down approach.
- During a program's execution, any given procedure might be called at any point, including by other procedures or itself.
- **Languages used in Procedural Programming:**
- FORTRAN, ALGOL, COBOL, BASIC, Pascal and C.

- **Merits:**
- Because of their flexibility, procedural languages are able to solve a variety of problems.
- Programmer does not need to think in term of computer architecture which makes them focused on the problem.
- Programs written in this language are portable.
- **Demerits:**
- It is easier but needs higher processor and larger memory.
- It needs to be translated therefore its execution time is more.

- **Object-oriented Programming Language**
- This programming language views the world as a group of objects that have internal data and external accessing parts of that data.
- The aim this programming language is to think about the fault by separating it into a collection of objects that offer services which can be used to solve a specific problem.
- One of the main principle of object oriented programming language is encapsulation that everything an object will need must be inside of the object.
- Follows bottom up approach
- This language also emphasizes reusability through inheritance and the capacity to spread current implementations without having to change a great deal of code by using polymorphism.

- **Scripting Programming Language**
- These programming languages are often procedural and may comprise object-oriented language elements, but they fall into their own category as they are normally not full-fledged programming languages with support for development of large systems.
- Scripting languages do not require the compilation step and are rather interpreted. For example, normally, a C program needs to be compiled before running whereas normally, a scripting language like JavaScript or PHP need not be compiled.
- For example, they may not have compile-time type checking. Usually, these languages require tiny syntax to get started.
- Example: javascript,perl,python

- **Natural language (5GL)**
- Natural language are still in developing stage where we could write statements that would look like normal sentences.
- **Merits:**
- Easy to program.
- Since, the program uses normal sentences, they are easy to understand.
- The programs designed using 5GL will have artificial intelligence (AI).
- The programs would be much more interactive and interesting.
- **Demerits:**
- It is slower than previous generation language as it should be completely translated into binary code which is a tedious task.
- Highly advanced and expensive electronic devices are required to run programs developed in 5GL. Therefore, it is an expensive approach.

How to choose a programming language?

- Computer programming is a set of written instructions that the computer follows. These instructions can be written in various languages.
- Each programming languages have their unique ways of organizing the commands which are called syntax.
- Multiple programming languages can help you solve the same programming problem. However, you need to select a language that you feel is relevant to perform your task.
- If you decide that a language does not suit your business requirement, you can always move on to a new language. Your skill in the chosen language will also be a deciding factor.
- Expected software system response time, a number of simultaneous users, security, maintains, compatibility with web, mobile, devices are few other factors to consider while choosing a language.

Programming Environments

- The term *programming environment* is sometimes reserved for environments containing language specific editors and source level debugging facilities; here, the term will be used in its broader sense to refer to all of the hardware and software in the environment used by the programmer.
- All programming can therefore be properly described as taking place in a programming environment.
- Programming environments may vary considerably in complexity. An example of a simple environment might consist of a text editor for program preparation, an assembler for translating programs to machine language, and a simple operating system consisting of input-output drivers and a file system.

- Though Environment Setup is not an element of any Programming Language, it is the first step to be followed before setting on to write a program.
- When we say Environment Setup, it simply implies a base on top of which we can do our programming. Thus, we need to have the required software setup, i.e., installation on our PC which will be used to write computer programs, compile, and execute them. For example, if you need to browse Internet, then you need the following setup on your machine −
  - A working Internet connection to connect to the Internet
  - A Web browser such as Internet Explorer, Chrome, Safari, etc.

Principles of Programming

- **1. Keep It Simple**
- It sounds a little harsh, but it's a coding principle to live by. What does this mean?
- It means you should be writing code as simple as possible. Don't get caught up in trying to be overly clever or showing off with a paragraph of advanced code. If you can write a script in one line, write it in one line.
- Here's a simple function:
- function addNumbers(num1,num2)
- {

```
        return num1 + num2;
}
```

- Pretty simple. It's easy to read and you know exactly what is going on.
- Use clear variable names. Take advantage of coding libraries to use existing tools. Make it easy to come back after six months and get right back to work. Keeping it simple will save you the headache.

2. Write DRY Code

- The **Don't Repeat Yourself (DRY)** principle means, plainly, not repeating code. It's a **common coding mistake**. When writing code, avoid duplication of data or logic. If you've ever copied and pasted code within your program, it's not DRY code.
- Take a look at this script:
- function addNumberSequence(number)
- {

```
        number = number + 1;
        number = number + 2;
        number = number + 3;
        number = number + 4;
        number = number + 5;
        return number;
}
```

- Instead of duplicating lines, try to find an algorithm that uses iteration. For loops, and while loops are ways to control code that needs to run multiple times.
- DRY code is easy to maintain. It's easier to debug one loop that handles 50 repetitions than 50 blocks of code that handle one repetition.

3. Open/Closed

- This principle means you should aim to make your code open to extension but closed to modification. This is an important principle when releasing a library or framework that others will use.
- For example, suppose you're maintaining a GUI framework. You could release for coders to directly modify and integrate your released code. But what happens when you release a major update four months later?
- Their code will break. This will make engineers unhappy. They won't want to use your library for much longer, no matter how helpful it may be.
- Instead, release code that prevents direct modification and encourages extension. This separates core behavior from modified behavior. The code is more stable and easier to maintain.

4. Single Responsibility

- The **single responsibility principle** says that every class or module in a program should only provide one specific functionality. As Robert C. Martin puts it, "A class should have only one reason to change."
- **Classes and modules** often start off this way. Be careful not to add too many responsibilities as classes get more complicated. Refactor and break them up into smaller classes and modules.
- The consequence of overloading classes is twofold. First, it complicates debugging when you're trying to isolate a certain module for troubleshooting. Second, it becomes more difficult to create additional functionality for a specific module.

5. Separation Of Concerns

- The **separation of concerns principle** is an abstract version of the single responsibility principle. This idea states that a program should be designed with different containers, and these containers should not have access to each other.
- A well-known example of this is the model-view-controller (MVC) design. MVC separates a program into three distinct areas: the data (model), the logic (controller), and what the page displays (view). Variations of MVC are common in today's most popular web frameworks.
- For example, the code that handles the database doesn't need to know how to render the data in the browser. The rendering code takes input from the user, but the logic code handles the processing. Each piece of code is completely independent.
- The result is code that is easy to debug. If you ever need to rewrite the rendering code, you can do so without worrying about how the data gets saved or the logic gets processed.

6. You Aren't Going To Need It (YAGNI)

- This principle means you should never code for functionality on the chance that you may need in the future. Don't try and solve a problem that doesn't exist.
- In an effort to write DRY code, programmers can violate this principle. Often inexperienced programmers try to write the most abstract and generic code they can. Too much abstraction causes bloated code that is impossible to maintain.
- Only apply the DRY principle only when you need to. If you notice chunks of code written over and over, then abstract them. Don't think too far out at the expense of your current code batch.

7. Document Your Code

- Any senior developer will stress the importance of documenting your code with proper comments. All languages offer them and you should make it a habit to write them. Leave comments to explain objects, enhance variable definitions, and make functions easier to understand.
- Here's a python function with comments guiding you through the code:
-

```
//This function will add 5 to the input if odd, or return the number if even

def evenOrOdd(number)
{
        //Determine if the number is even
        if(number % 2 == 0):
    {
                        return number
    }
        //If the number is odd, this will add 5 and return
        else:
    {
                        return number + 5
    }
    }
```

- leaving comments is a little more work while you're coding, and you understand your code pretty well right?
- Leave comments anyway!
- Try writing a program, leaving it alone for six months, and come back to modify it. You'll be glad you documented your program instead of having to pour over every function to remember how it works. Work on a coding team? Don't frustrate your fellow developers by forcing them to decipher your syntax.

8. Refactor

- It's hard to accept, but your code isn't going to be perfect the first time. Refactoring code means reviewing your code and looking for ways to optimize it. Make it more efficient while keeping the results exactly the same.
- Codebases are constantly evolving. It's completely normal to revisit, rewrite, or even redesign entire chunks of code. It doesn't mean you didn't succeed the first time you wrote your program.
- You're going to get more familiar with a project over time. Use that knowledge to adjust your existing code to be DRY, or following the KIS principle.

9. Clean Code At All Costs

- Don't try to pack a ton of logic into one line. Leave clear instructions in comments and documentation. If your code is easy to read it will be easy to maintain.
- Good programmers and readable code go hand-in-hand. Leave comments when necessary. Adhere to style guides, whether dictated by a language or your company.

What is Debugging?

- Software programs undergo heavy testing, updating, troubleshooting, and maintenance during the development process. Usually, the software contains **errors and bugs**, which are routinely removed. **Debugging** is the process of **fixing a bug** in the software.

- It refers to identifying, analyzing and removing errors. This process begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. But, it is considered to be an extremely **complex** and **tedious** task because errors need to be resolved at all stages of debugging.

Why do we need Debugging?

- The process of debugging begins as soon as the **code** of the **software** is written. Then, it continues in successive stages as code is combined with other units of programming to form a software product. Debugging has many benefits such as:
- It **reports** an **error condition immediately.** This allows earlier detection of an error and makes the process of software development stress-free and unproblematic.
- It also provides **maximum useful information** of data structures and allows **easy interpretation.**
- Debugging assists the developer in **reducing useless** and **distracting information.**
- Through debugging the developer can **avoid complex one-use testing code** to save time and energy in software development.

Steps involved in Debugging

- The different steps involved in the process of debugging are:

- **1. Identify the Error:** A bad identification of an error can lead to wasted developing time. It is usual that production errors reported by users are hard to interpret and sometimes the information we receive is misleading. It is import to identify the actual error.
- **2. Find the Error Location:** After identifying the error correctly, you need to go through the code to find the exact spot where the error is located. In this stage, you need to focus on finding the error instead of understanding it.
- **3. Analyze the Error:** In the third step, you need to use a bottom-up approach from the error location and analyze the code. This helps you in understanding the error. Analyzing a bug has two main goals, such as checking around the error for other errors to be found, and to make sure about the risks of entering any collateral damage in the fix.

- **4. Prove the Analysis:** Once you are done analyzing the original bug, you need to find a few more errors that may appear on the application. This step is about writing automated tests for these areas with the help of a test framework.
- **5. Cover Lateral Damage:** In this stage, you need to create or gather all the unit tests for the code where you are going to make changes. Now, if you run these unit tests, they all should pass.
- **6. Fix & Validate:** The final stage is the fix all the errors and run all the test scripts to check if they all pass.

Debugging Tools

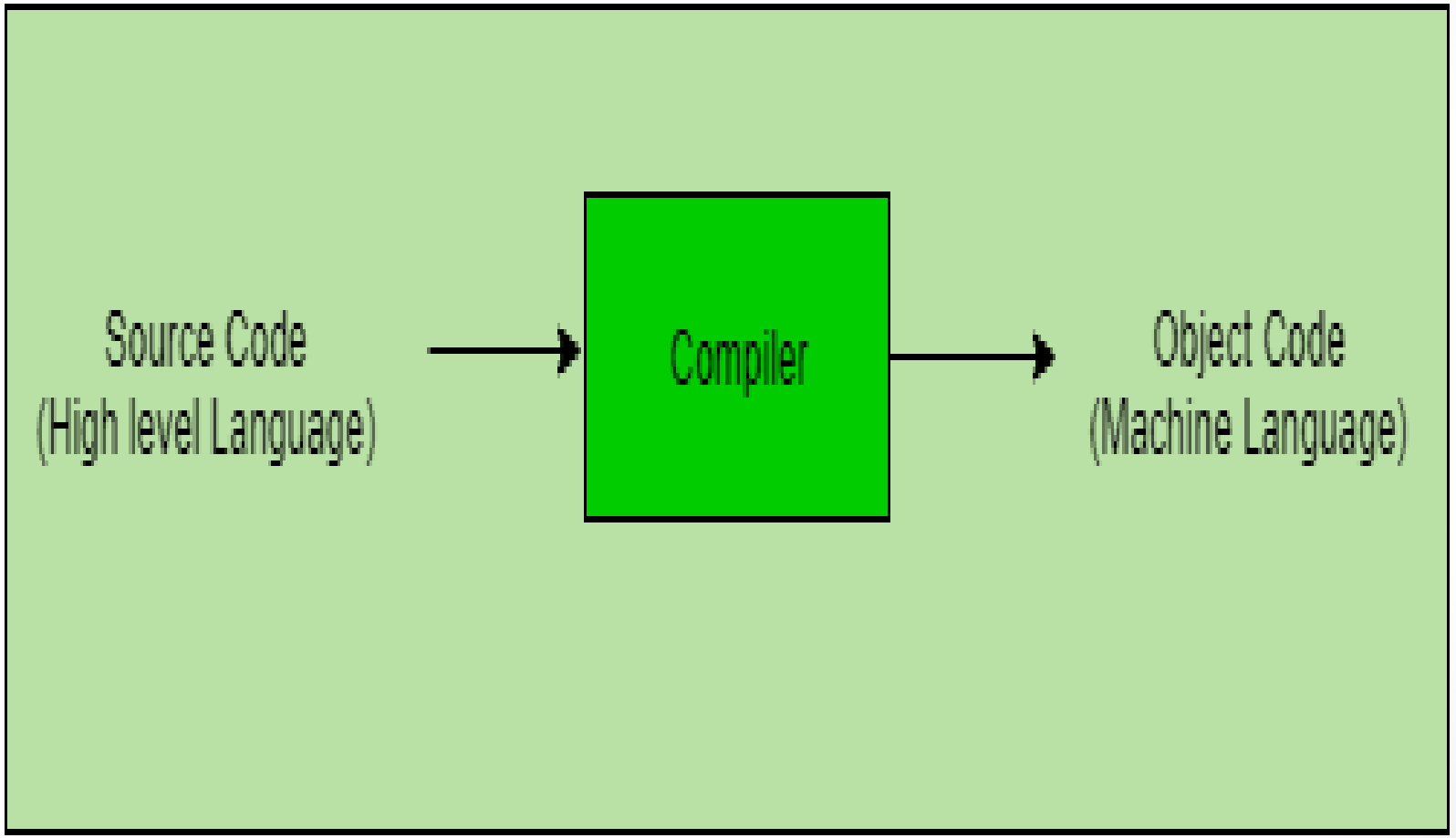
- Debugging tool is a computer program that is used to test and debug other programs. A lot of public domain software like gdb and dbx are available for debugging. They offer console-based command line interfaces. Examples of automated debugging tools include code based tracers, profilers, interpreters, etc.
  Some of the widely used debuggers are:

1. Radare2
2. WinDbg
3. Valgrind

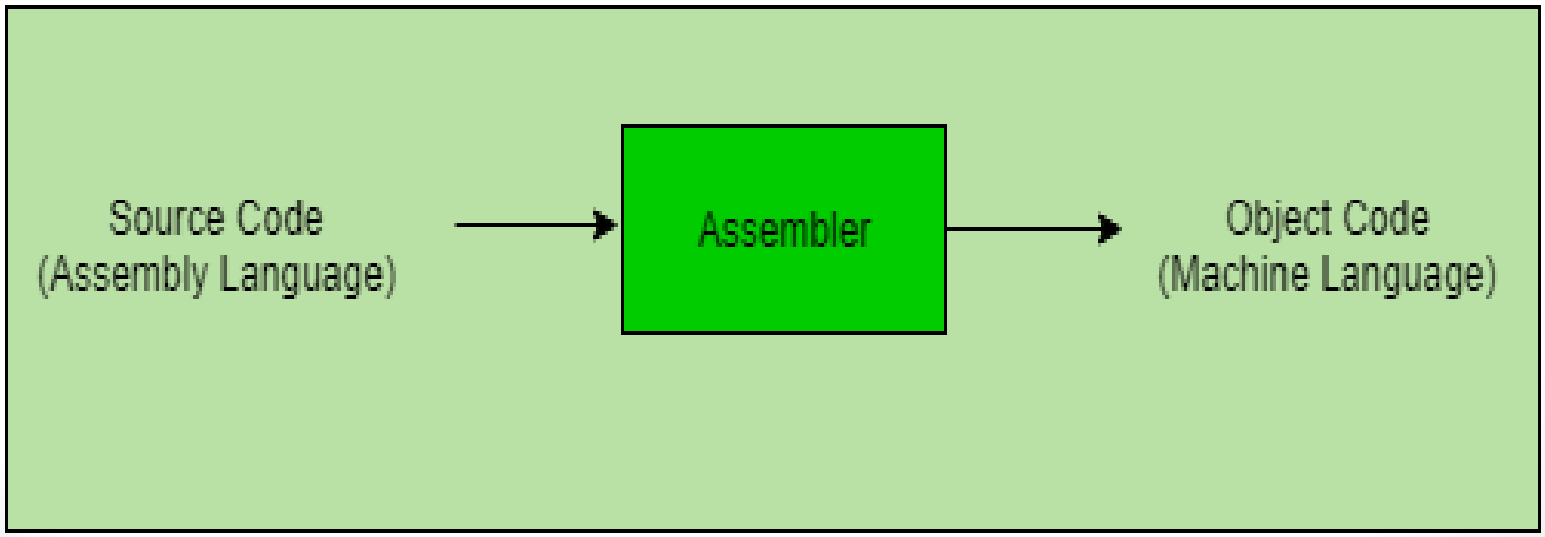
Language Processors

- Assembly language is machine dependent yet mnemonics that are being used to represent instructions in it are not directly understandable by machine and high Level language is machine independent.
- A computer understands instructions in machine code, i.e. in the form of 0s and 1s. It is a tedious task to write a computer program directly in machine code.
- The programs are written mostly in high level languages like Java, C++, Python etc. and are called **source code**. These source code cannot be executed directly by the computer and must be converted into machine language to be executed.
- Hence, a special translator system software is used to translate the program written in high-level language into machine code is called **Language Processor** and the program after translated into machine code (object program / object code).

- The language processors can be any of the following three types:
- **1. Compiler**
- The language processor that reads the complete source program written in high level language as a whole in one go and translates it into an equivalent program in machine language is called as a Compiler.
  Example: C, C++, C#, Java In a compiler, the source code is translated to object code successfully if it is free of errors. The compiler specifies the errors at the end of compilation with line numbers when there are any errors in the source code. The errors must be removed before the compiler can successfully recompile the source code again.>


Source Code
(High level Language)
Compiler
Object Code
(Machine Language)

- **Assembler**
  The Assembler is used to translate the program written in Assembly language into machine code. The source program is a input of assembler that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.



Interpreter

The translation of single statement of source program into machine code is done by language processor and executes it immediately before moving on to the next line is called an interpreter.

- If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message.
- The interpreter moves on to the next line for execution only after removal of the error. An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code.
  Example: Perl, Python and Matlab.

Difference between Compiler and Interpreter

COMPILER	INTERPRETER
A compiler is a program which coverts the entire source code of a programming language into executable machine code for a CPU.	interpreter takes a source program and runs it line by line, translating each line as it comes to it.
Compiler takes large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster.	Interpreter takes less amount of time to analyze the source code but the overall execution time of the program is slower.
Compiler generates the error message only after scanning the whole program, so debugging is comparatively hard as the error can be present any where in the program.	Its Debugging is easier as it continues translating the program until the error is met
Generates intermediate object code.	No intermediate object code is generated.
Examples: C, C++, Java	Examples: Python, Perl

Flow Chart

- WHAT IS A FLOWCHART?
- A flowchart is a diagrammatic representation of an algorithm.
- A flowchart can be helpful for both writing programs and explaining the program to others.
- Also called: process flowchart, process flow diagram

- A flowchart is a picture of the separate steps of a process in sequential order. It is a generic tool that can be adapted for a wide variety of purposes, and can be used to describe various processes, such as a manufacturing process, an administrative or service process, or a project plan.
- Elements that may be included in a flowchart are a sequence of actions, materials or services entering or leaving the process (inputs and outputs), decisions that must be made, people who become involved, time involved at each step, and/or process measurements.

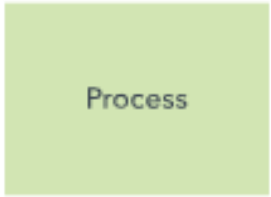
WHEN TO USE A FLOWCHART

1. To develop understanding of how a process is done
2. To study a process for improvement
3. To communicate to others how a process is done
4. When better communication is needed between people involved with the same process
5. To document a process
6. When planning a project

Flowchart symbols

Terminal/Terminator

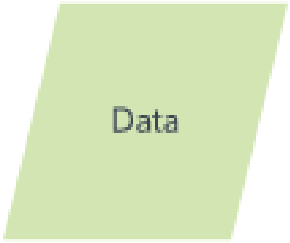
Process



Decision

Document

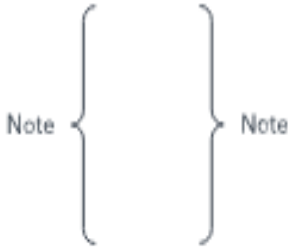
Data, or Input/Output



Stored Data

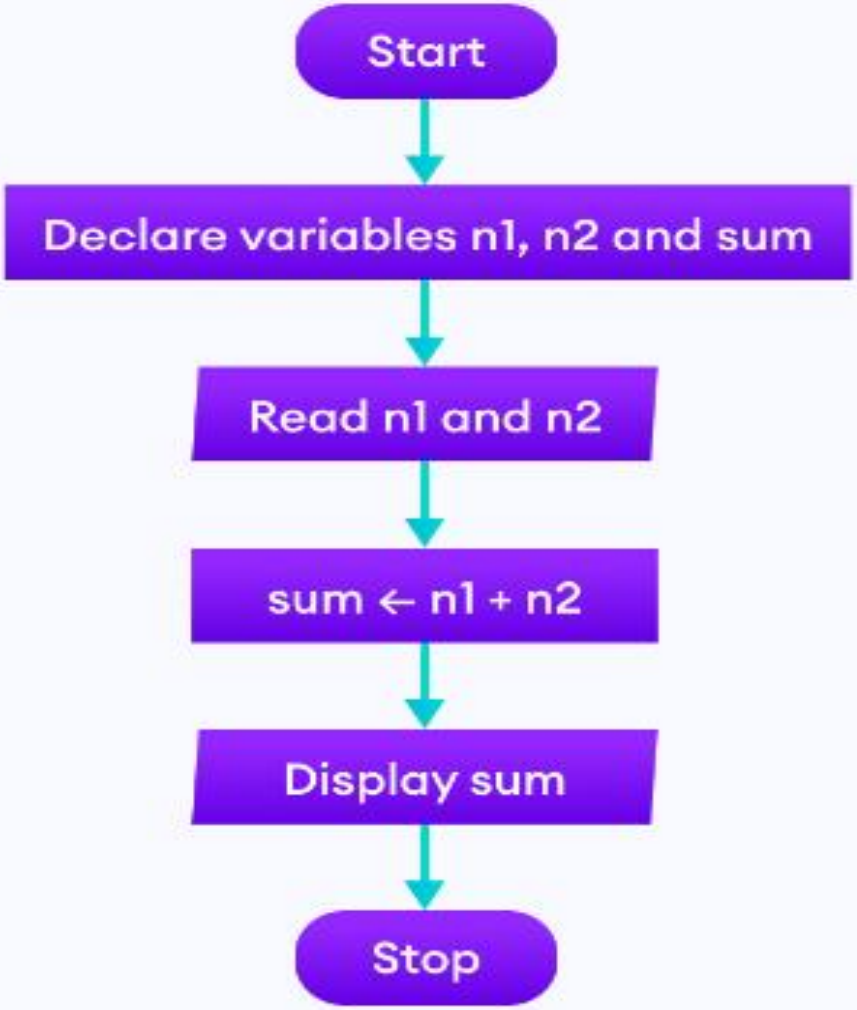
Flow Arrow

Comment or Annotation

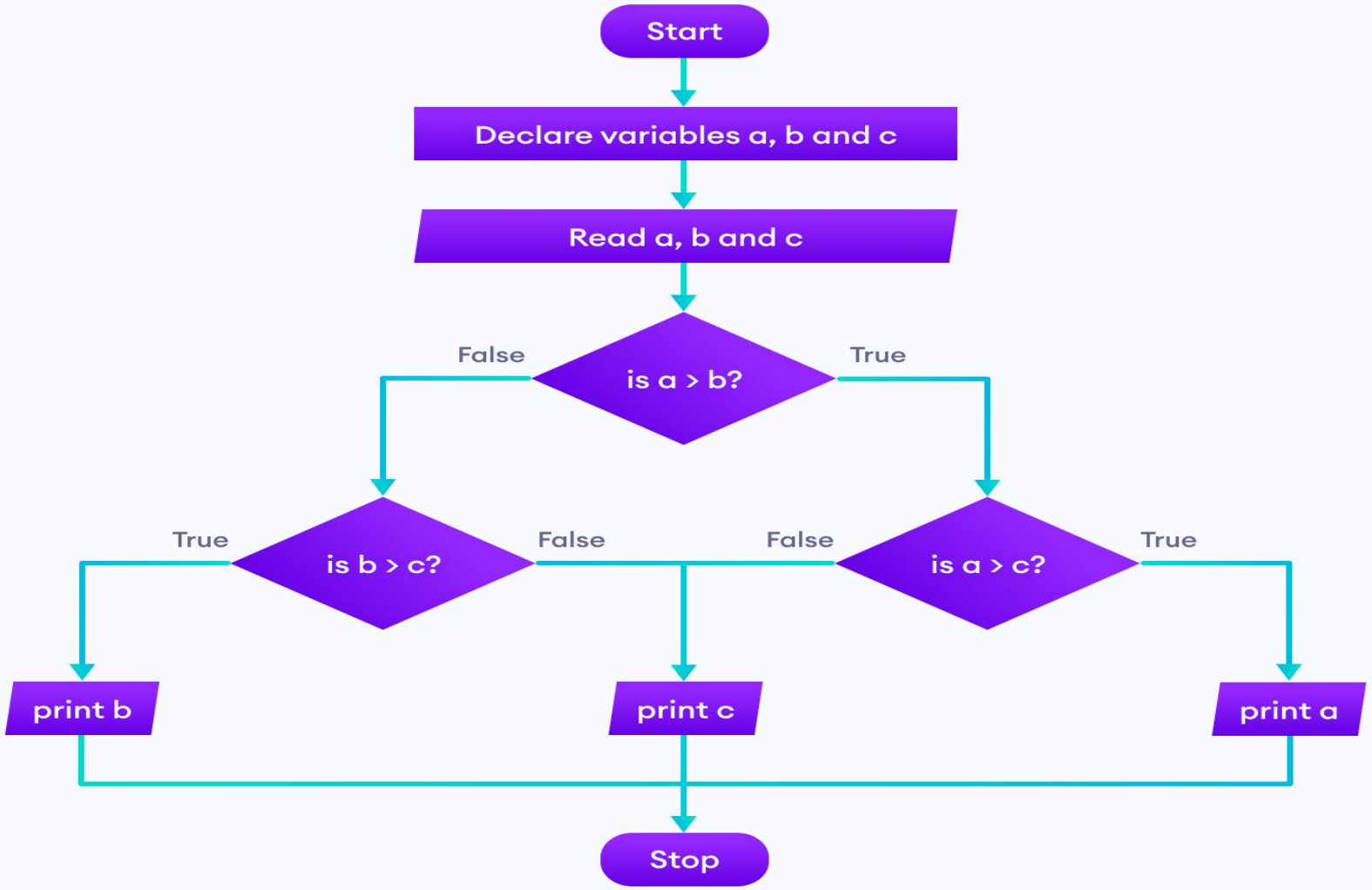


Predefined process

Add two numbers entered by the user



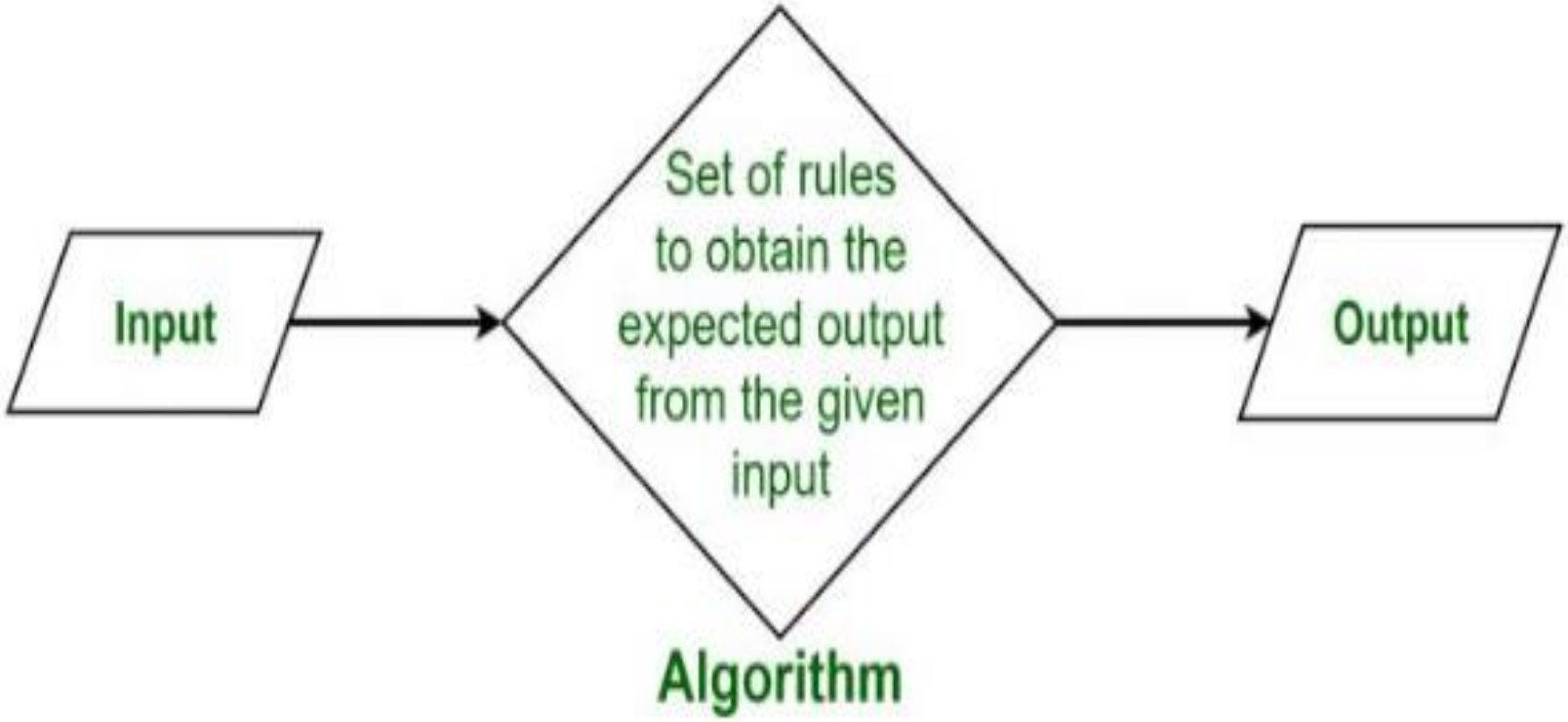
Find the largest among three different numbers entered by the user



Algorithm

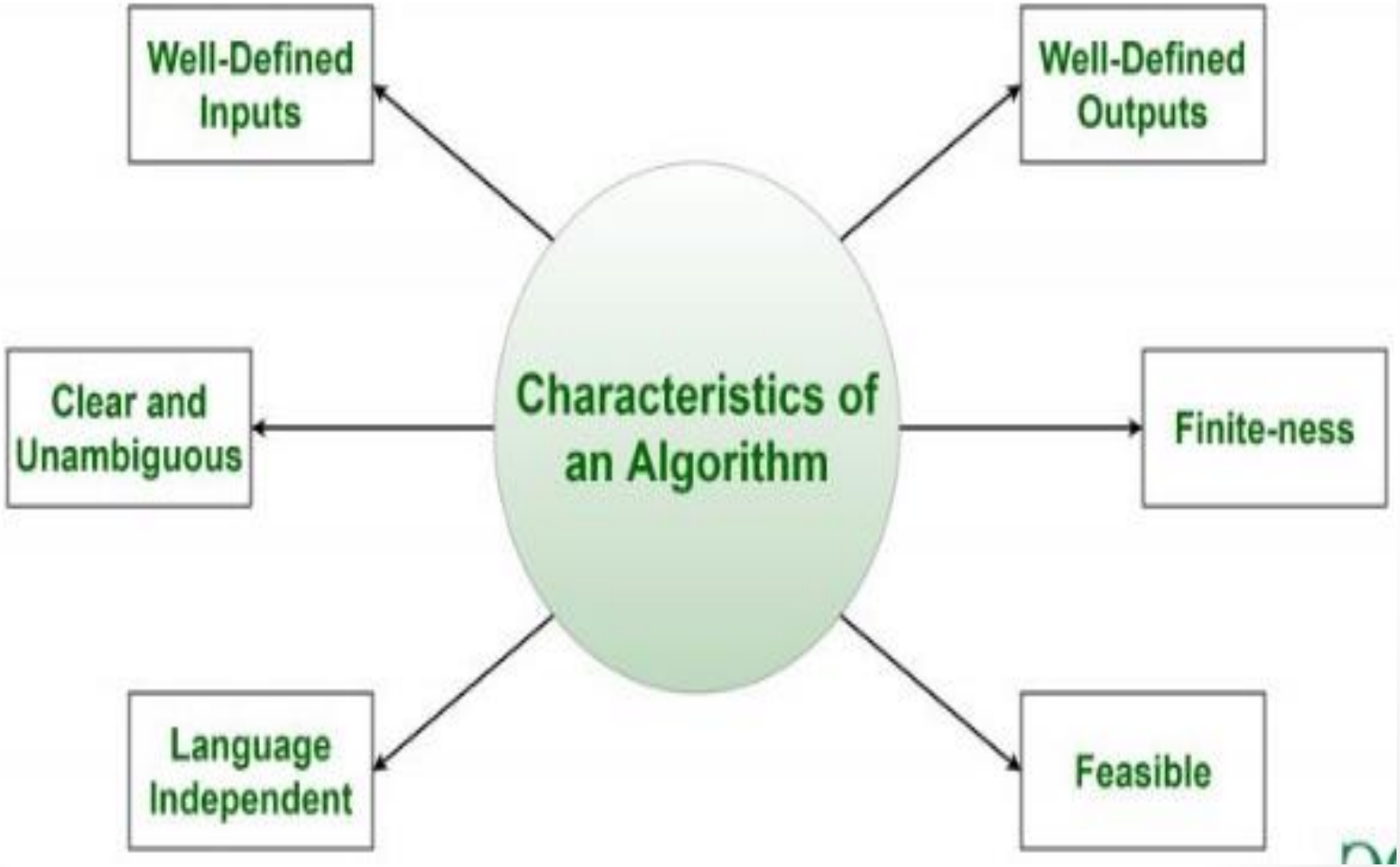
- **What is Algorithm?**
- Algorithms are simply a series of instructions that are followed, step by step, to do something useful or **solve a problem**
- The word **Algorithm** means “a process or set of rules to be followed in calculations or other problem-solving operations”.
- Therefore Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed upon in order to get the expected results.

What is Algorithm?



- It can be understood by taking an example of cooking a new recipe. To cook a new recipe, one reads the instructions and steps and execute them one by one, in the given sequence.
- The result thus obtained is the new dish cooked perfectly. Similarly, algorithms help to do a task in programming to get the expected output.
- The Algorithm designed are language-independent, i.e. they are just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.

Characteristics of an Algorithm



- As one would not follow any written instructions to cook the recipe, but only the standard one. Similarly, not all written instructions for programming is an algorithm. In order for some instructions to be an algorithm, it must have the following characteristics:
- **Clear and Unambiguous**: Algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- **Well-Defined Inputs**: If an algorithm says to take inputs, it should be well-defined inputs.
- **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well.

- **Finite-ness:** The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.
- **Feasible:** The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources. It must not contain some future technology, or anything.
- **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.

How to Design an Algorithm?

- In order to write an algorithm, following things are needed as a pre-requisite:

1. The **problem** that is to be solved by this algorithm.
2. The **constraints** of the problem that must be considered while solving the problem.
3. The **input** to be taken to solve the problem.
4. The **output** to be expected when the problem the is solved.
5. The **solution** to this problem, in the given constraints.
6. Then the algorithm is written with the help of above parameters such that it solves the problem.

- **Example:** Consider the example to add three numbers and print the sum.
- **Step 1: Fulfilling the pre-requisites** As discussed above, in order to write an algorithm, its pre-requisites must be fulfilled.
  - **The problem that is to be solved by this algorithm**: Add 3 numbers and print their sum.
  - **The constraints of the problem that must be considered while solving the problem**: The numbers must contain only digits and no other characters.
  - **The input to be taken to solve the problem:** The three numbers to be added.
  - **The output to be expected when the problem the is solved:** The sum of the three numbers taken as the input.
  - **The solution to this problem, in the given constraints:** The solution consists of adding the 3 numbers. It can be done with the help of '+' operator, or bit-wise, or any other method.

Algorithm to add 3 numbers and print their sum:

1. START
2. Declare 3 integer variables num1, num2 and num3.
3. Take the three numbers, to be added, as inputs in variables num1, num2, and num3 respectively.
4. Declare an integer variable sum to store the resultant sum of the 3 numbers.
5. Add the 3 numbers and store the result in the variable sum.
6. Print the value of variable sum
7. END

Algorithm for Simple Interest

- 1. Start
- 2. Declare 3 floating point variables p,r,t
- 3. Declare SI as floating point variable
- 4. SI=(p*r*t)/100
- 5. Print and store the result in SI
- 6.Stop

Algorithm Complexity

What is Algorithm Complexity and How to find it?

- An algorithm is defined as complex based on the amount of Space and Time it consumes. Hence the Complexity of an algorithm refers to the measure of the Time that it will need to execute and get the expected output, and the Space it will need to store all the data (input, temporary data and output). Hence these two factors define the efficiency of an algorithm.

The two factors of Algorithm Complexity are:

1. **Time Factor:** Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
2. **Space Factor:** Space is measured by counting the maximum memory space required by the algorithm.

- Therefore the complexity of an algorithm can be divided into two types:

1. Space Complexity: Space complexity of an algorithm refers to the amount of memory that this algorithm requires to

2. Time Complexity: Time complexity of an algorithm refers to the amount of time that this algorithm requires to execute and get the result. This can be for normal operations, conditional if-else statements, loop statements, etc.