Procedural Oriented Programming: It is a programming model that is based on the concept of calling procedures. The procedures consist of a sequence of statements or the computational steps to be executed. These procedures are the blocks in which the set of instructions to be executed by the computer are organized. These procedures are also known as routines, sub-routines or functions.

In this programming approach, the problem is solved by performing the tasks step by step. It contains a series of logic to be carried out which are executed in a block called procedures/functions. It is a top-down language. The earlier or the conventional programming languages such as COBOL, FORTRAN used this procedural oriented programming.

A typical program structure for procedural programming is shown in figure below.

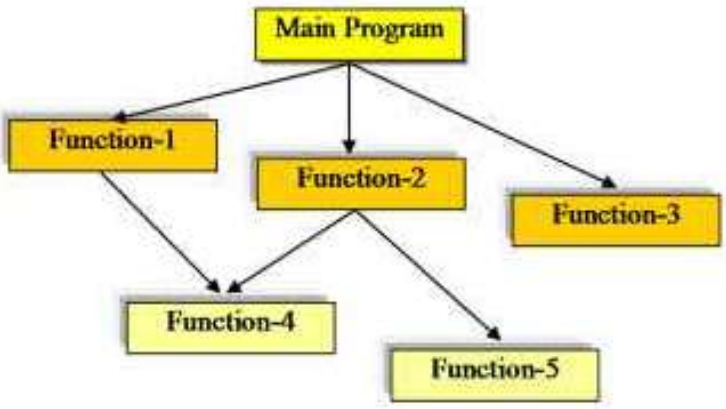

Figure: Procedural programming language

Limitations/shortcomings of Procedural oriented programming:

- The procedural oriented programming languages are difficult to relate with the real world objects.
- These methods of programming are difficult to write and maintain.
- Important data items are placed as global, which may be accessed by all the functions. So, there is no security of data.
- Adding new data and functions is not easy.
- Code reusability feature is absent, as there is no inheritance and polymorphism.
- Memory management is difficult.

Object Oriented Programming: a new Paradigm

It is a programming model that is based on the concept of objects. In this programming paradigm, a problem is decomposed into a number of entities which are known as objects. The data and functions are then built around these objects. The organization of data and functions in the object-oriented programming are shown in the figure below:

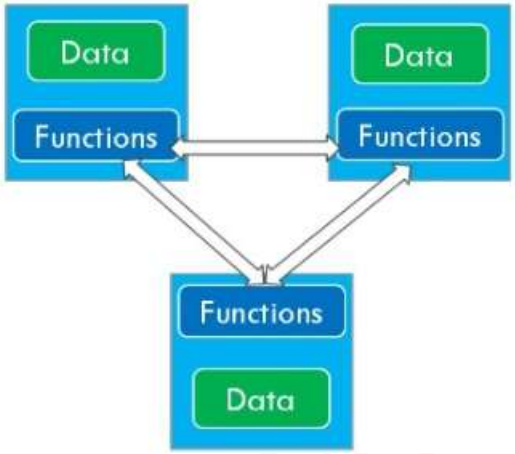

Figure: Object Oriented Programming

In OOP, the emphasis is given for data rather than procedures. Everything is modeled in terms of objects. The data of an object can be accessed only by the function associated with that object. One of major features of OOP is to combine both data and functions that operate on data into a single unit. This unit is called an object and the feature is known as encapsulation. The data that is encapsulated is hidden from the outside world. Hence, it provides security to the system.

The OOP approach supports many other features such as abstraction, inheritance, message passing, polymorphism etc. The object-oriented programming approach can handle almost all kind of real-life problems. The programming languages such as Python, C++, Java etc. are based on these OOP concept.

Features/Characteristics of Object Oriented Programming:

Some of the key features of Object Oriented Programming language are:

- **Object:** In OOP, the problem is decomposed into a number of small units which are known as objects. These objects are the real-world entities. They are the instances of a class. Example of object could be any entity such as particular book, a student, an employee, etc.

- **Class:** It is a collection of objects. It is regarded as a template or blueprint of the objects. It is a user-defined data type that holds various variables and functions under one construct. Example of a class could be a template that represents a collection of books in a library, or a number of students at a college, or number of employees in an organization.
- **Encapsulation and data hiding:** The wrapping of data and functions into a single unit is known as encapsulation. The data are kept private and are not accessible to the outside world. These data are accessed by only the member functions that are present in the same unit (i.e. in the same class). This process is also known as data hiding as the data are hidden from outside.
- **Abstraction:** It refers to the process of displaying only the essential features without including the background details. It helps to increase the security of the system.
- **Inheritance:** It is a feature in which one object can inherit the properties of another. With the help of inheritance, the redundant codes are eliminated and the use of existing classes can be extended.
- **Polymorphism:** It is a feature in which the same function or operator can be used to perform differently in different situations.
- **Dynamic binding:** It is a linkage between the code and the function call in which the code associated with a given function call is not known until the time of the call at run-time.
- **Message passing:** It is a method of sending request to an object to perform a specific action. Objects can communicate with each other by passing messages.

Advantages of OOP:

- The modeling of real-world problems become easy by using OOP as it represents all the real-world entities as objects and the data and functions are built around these objects.
- By the use of inheritance, the redundant codes are eliminated and the existing features can be extended.
- It supports encapsulation, i.e., the data and functions are wrapped inside a single unit. The data is protected from the outside world and only the associated functions can access them. So, it provides more security to the system.

- With the help of polymorphism, the same function or operator can be used for various purposes. This helps to manage the complexity of the software with ease.
- Large problems can be reduced to smaller and manageable ones. It is easy to partition the work in a project based on objects.
- Object oriented system can be easily upgraded from small to a large system.
- Software complexity can be easily managed.
- It supports abstraction so that the internal implementation can be modified without affecting the user. It helps to increase the security of the system.

Differences between OOP and Structured/Procedural Programming:

POP	OOP
1. It is the old programming concept that uses the top-down approach.	1. It is the most recent programming concept that uses the bottom-up approach.
2. Programs are divided into a number of smaller program segments called functions.	2. Programs are divided into a number of entities known as objects.
3. Emphasis is on procedures rather than on data.	3. Emphasis is on data rather than on procedures.
4. It is less secure as there is no data hiding feature.	4. It is more secure as it has data hiding feature.
5. Data and functions don't tie with each other.	5. Data and functions tie with each other.
6. It allows the data to move freely around the program from one function to another.	6. Data is hidden and so cannot be accessed by external functions due to encapsulation.
7. It can solve moderately complex programs.	7. It can solve any complex programs.
8. Problem is not viewed as a real word entity.	8. Problem is viewed as a real world entity.
9. It does not support inheritance and polymorphism.	9. It supports inheritance and polymorphism.
10. Example languages: C, Pascal, FORTRAN	10. Example languages: C++, Java, and C#

A way of viewing the world: Let us consider a real-world scenario that involves several agents of the community to accomplish a small task. Suppose, an individual named Chris wishes to send flowers to a friend named Robin, who lives in another city. Because of far distance, Chris cannot simply pick the flowers and take them to Robin in person. But this task could be easily solved with the following process:

Chris simply walks to a nearby flower shop, run by an owner named Fred. Chris tells Fred the kinds of flowers to be send to Robin and the address to which they should be delivered. Chris then can be assured that the flowers will be delivered. Fred makes contacts and requests various agents of the community and ultimately, the flowers are delivered to Robin.
In above example,

- Fred is an *agent (object)* to deliver flower.
- *Message* of wishes with specification of flowers is passed to Fred who has *responsibility* to satisfy request.
- Fred uses some *methods or set of operations* to perform the task.
- A communication is established to a community of agents to complete the task.
- How Fred satisfies the request or completes the task is not under Chris's concern. *(abstraction/ data hiding)*

Agents and community: In the above example, Chris solved his problem with the community of agents. Including Fred (shop-owner), the agents might be wholesaler, flower arranger, gardener, delivery person etc.

In a similar way to above, an object-oriented programming is modeled as a community of interacting agents called objects. Each object has a role to play. Each object provides a service that is used by other members of the community.

Message and methods: In the above example, the different agents communicate with each other by passing some message. There are a lot of requests involved in communication starting from Chris to Fred and then to other agents until the flower is delivered.
So, in OOP as well, any action is initiated by passing message to other agents (objects) that are responsible for the action. In response to the message, the receiver (object) performs some methods to satisfy the requests.

Responsibilities: In the above example, the request from Chris to Fred only indicates the desired outcome i.e. to send the flower to Robin. Fred has the responsibility but he is free to choose any technique that achieves the desired outcome.

Similarly in OOP, there is greater independence between agents (objects) in solving the complex problem. Responsibilities are the *behaviors (methods)* to respond the messages and describing behaviors in terms of responsibilities is an important concept of OOP.

Computation as simulation: In structural model of programming, the computer is viewed as a data manager. It follows some algorithms, drifts through the memory, pulling values out of various slots (memory locations), process them according to the instructions in the algorithms, and pushing the results back to other slots of the memory. By examining the values in the slots, one can determine the state of the machine or the results produced by a computation. This model may give a picture of what takes place inside a computer. However, solving the real world problem is difficult in this conventional model.

On the other hand, in object oriented programming model, we do not focus on the memory address of the variable. Instead, we speak of objects, messages and responsibilities to carry out certain actions. The objects (agents) interact with various other related agents. The view of programming is similar to the style of computer simulation called 'discrete event-driven simulation'. In this simulation, the user creates computer models of the various elements of the simulation, describes how they will interact with one another and sets them running. This is identical to OOP in which the user describes what the objects in a program are, how they will interact with one another and finally set them moving. In this way, we have a view that computation is a simulation in object oriented programming.

Coping with complexity: At early stages of computing, most of the programs were written in assembly language by a single individual. As programs become more complex, programmers found that they had a difficult time remembering all the information they needed to know in order to develop or debug their software.

The introduction of higher-level language, such as FORTAN, Cobol and Algol solved some difficulties while simultaneously raising people expectations of what a computer could do.

As programmers attempted to solve more complex problems, it became difficult for even the best programmers to solve the tasks. Thus, teams of programmers working together to solve major programming efforts became commonplace.

Non-linear behavior of complexity: As programming projects became larger, an interesting phenomenon was observed. A task that would take one programmer two months to perform could not be accomplished by two programmers working for one month.

A memorable phrase, by Fred Brook is worth a mention here. “The bearing of a child takes nine months, no matter how many women are assigned to the task”.

The reason for this non-linear behavior was the complexity. The interconnections between software components were complicated. And large amounts of information had to be communicated among various members of the programming team.

The complexity is not about the size, because size itself could be partitioned into smaller parts. Due to the unique features of software systems developed using conventional techniques; one portion of code depends on another portion of code. The high degree of interconnectedness (coupling) among various portions of code makes the system complex.

Consider a portion of code (component) performing a particular task. If this task is useful to other parts of the program, there must be communication of information between the component under consideration and other sections. A complete understanding of what is going on requires knowledge of both the portion of code under consideration and the other sections of code that communicates with it. An individual section of code cannot be understood in isolation.

Abstraction: Abstraction is a mechanism of displaying only the essential features without including the background details.

Abstraction mechanism is used to control the complexity of the system. By use of abstraction, the internal implementation can be modified without affecting the user. And hence, it helps to increase the security of the system.

Making use of Abstraction:

- *Procedure and function:* Procedure and function are the two first major abstraction mechanisms to be widely used in programming language. They allowed the repeated tasks to be collected in one place and reused rather than being duplicated several times. The functions written by one programmer can be used by many other programmers without knowing the exact details of implementation.
- *Module:* A module is basically a mechanism that allows the programmers to encapsulate data and functions, provide the necessary information and control the visibility features from outside.
  According to parna's principles:
  - The intended user must be provided with all the information needed to make effective use of the services provided by the modules, and should provide no other information.
  - The modules must be provided with all the information necessary to carry out the given responsibilities assigned to it, and should be provided with no other information.
- *Block scoping:* It is the nesting of function inside another function to share and restrict the scope of certain data and implementation. It partially solves the abstraction mechanism.
- *Abstract data type (ADT):* Abstract data type is a user defined data type that can be used in a manner similar to the primary data types. They can be used to support both the information hiding as well as creating many instances of new data types.
- *Objects-Messages, Inheritance and Polymorphism:*
  - *Message passing:* A request is sent to an object to perform a specific action.
  - *Inheritance:* With the help of inheritance, the redundant codes are eliminated and the use of exiting classes can be extended.
  - *Polymorphism:* The same functions can be used in different forms. It manages the complexity of the software.

OLD/MODEL/IMPORTANT QUESTIONS:

1. What are the advantages of OOP over structural programming/ procedural oriented programming? Explain.

2. Explain the notion of 'everything is an object' in OOP.
Or, Describe OOP as a new paradigm in computer programming field.

3. What are the key features of OOP? Explain with example.

4. What are the differences between OOP and structured programming? Explain.

5. With the help of OOP, explain how OOP can cope in solving the complex program. Explain computation as simulation.

6. Software development process is not linear, justify.
Or, Why behavior of complexity is regarded as non-linear?
Or, Explain non-linear behavior of complexity.

7. Explain abstraction mechanism technique in C++ with examples.
Or, What is the use of abstraction mechanism in C++? Explain with example.

8. Describe how OOP models the real-world problem with reference of agents, methods, behavior and responsibilities.