1.1 Procedure Oriented Programming

Procedure Oriented Programming (POP) is a programming paradigm that focuses on writing procedures or functions to perform operations on data. In POP, the program is divided into a set of functions, where each function performs a specific task. These functions are called in a sequential manner to accomplish the desired outcome.

Here are some key characteristics of Procedure Oriented Programming:

1. **Emphasis on Functions**:
   - In POP, the main building blocks are functions. Each function is responsible for performing a specific task or operation. These functions can take inputs, process them, and return outputs.
2. **Global Data**:
   - In POP, data is often declared as global variables that can be accessed by any part of the program. This can lead to potential issues with data integrity and security.
3. **Top-Down Approach**:
   - The program is designed in a top-down manner, starting with a main procedure or function that calls other functions as needed. This can make it easier to understand the flow of execution.
4. **Limited Code Reusability**:
   - While functions can be reused within the same program, they are not designed for easy reuse in other programs. This can lead to code duplication and reduced maintainability.
5. **Procedural Decomposition**:
   - The problem is divided into smaller, manageable tasks (procedures or functions) that can be tackled separately. Each function focuses on a specific aspect of the problem.
6. **Data and Functions are Separate**:
   - In POP, data and functions are distinct entities. Data is stored in variables, and functions operate on this data. Functions do not have an inherent association with specific data.
7. **Less Emphasis on Data Abstraction**:
   - POP does not strongly emphasize data abstraction or hiding implementation details. Data is often accessible globally, and functions have direct access to it.
8. **Limited Support for Object-Oriented Concepts**:
   - POP does not have built-in support for concepts like classes, objects, inheritance, and polymorphism, which are fundamental to Object-Oriented Programming (OOP).

Examples of languages that primarily follow the Procedure Oriented Programming paradigm include C, Pascal, and early versions of Fortran.

While Procedure Oriented Programming is a valid and effective paradigm, it has been largely superseded by Object-Oriented Programming (OOP) in modern software development due to OOP's emphasis on encapsulation, inheritance, polymorphism, and abstraction, which can lead to more modular, reusable, and maintainable code.

1.2 Object-Oriented Programming

The word **object-oriented** is the combination of two words i.e. **object** and **oriented**. The dictionary meaning of the object is an article or entity that exists in the real world. The meaning of oriented is interested in a particular kind of thing or entity. In layman's terms, it is a programming pattern that rounds around an object or entity are called **object-oriented programming.**

The technical definition of object-oriented programming is as follows:

The **object-oriented programming** is basically a computer programming design philosophy or methodology that organizes/ models software design around data, or objects rather than functions and logic.

. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Any entity that has state and behavior is known as **an object**. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Collection of objects is called **class.** It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

It is the most popular programming model among developers. It is well suited for programs that are large, complex, and actively updated or maintained. It simplifies software development and maintenance by providing major concepts such as **abstraction, inheritance, polymorphism**, and **encapsulation**. These core concepts support OOP.

A real-world example of OOP is the automobile. It more completely illustrates the power of object-oriented design.

Points to Remember

- Everything is an object
- Developer manipulates objects that uses message passing.
- Every object is an instance of a class.
- The class contains the attribute and behavior associated with an object.

1.3 Procedure Oriented Programming Vs Object-Oriented Programming

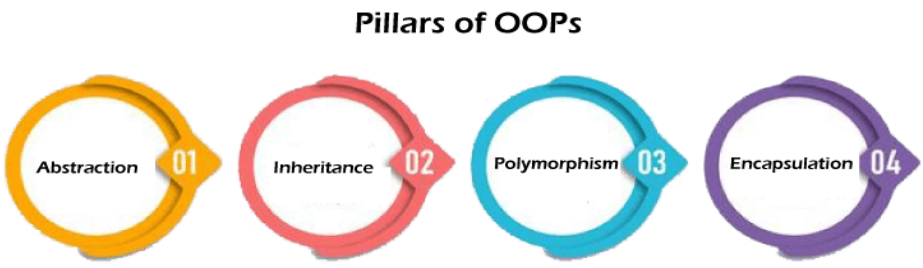
On the basis of	Procedural Programming	Object-oriented programming
Definition	It is a programming language that is derived from structure programming and based upon the concept of calling procedures. It follows a step-by-step approach in order to break down a task into a set of variables and routines via a sequence of instructions.	Object-oriented programming is a computer programming design philosophy or methodology that organizes/ models software design around data or objects rather than functions and logic.
Security	It is less secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
Approach	It follows a top-down approach.	It follows a bottom-up approach.
Data movement	In procedural programming, data moves freely within the system from one function to another.	In OOP, objects can move and communicate with each other via member functions.
Orientation	It is structure/procedure-oriented.	It is object-oriented.

Access modifiers	There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
Inheritance	Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
Code reusability	There is no code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.
Overloading	Overloading is not possible in procedural programming.	In OOP, there is a concept of function overloading and operator overloading.
Importance	It gives importance to functions over data.	It gives importance to data over functions.
Virtual class	In procedural programming, there are no virtual classes.	In OOP, there is an appearance of virtual classes in inheritance.
Complex problems	It is not appropriate for complex problems.	It is appropriate for complex problems.
Data hiding	There is not any proper way for data hiding.	There is a possibility of data hiding.
Program division	In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
Examples	Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++.

1.4 OOP principles (pillars of OOP)

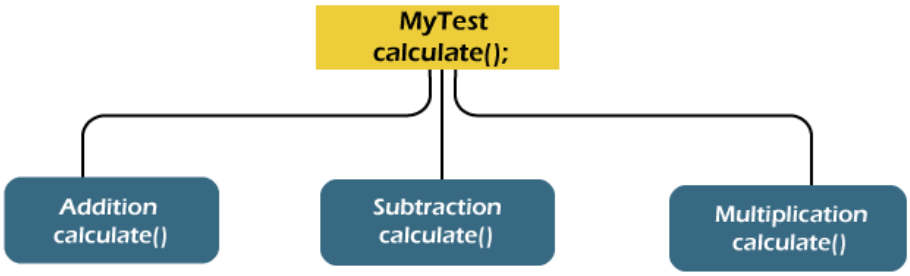
There are **four** main principle of OOP are:

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism



Abstraction

The concept allows us to hide the implementation from the user but shows only essential information to the user. Using the concept developer can easily make changes and added over time.

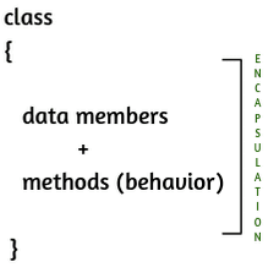


There are the following advantages of abstraction:

- It reduces complexity.
- It avoids delicacy.
- Eases the burden of maintenance
- Increase security and confidentially.

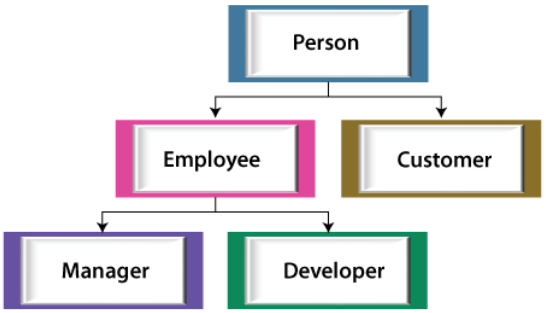
Encapsulation

Encapsulation is a mechanism that allows us to bind data and functions of a class into an entity. It protects data and functions from outside interference and misuse. Therefore, it also provides security. A class is the best example of encapsulation.



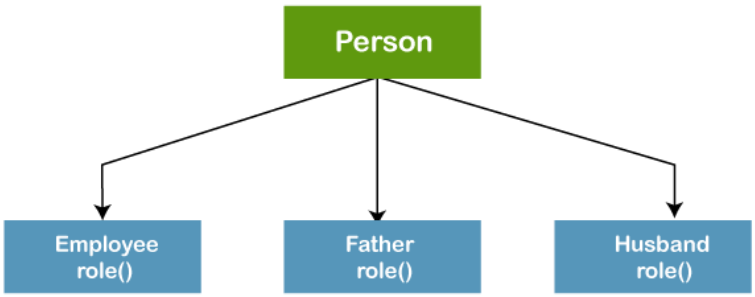
Inheritance

The concept allows us to inherit or acquire the properties of an existing class (parent class) into a newly created class (child class). It is known as **inheritance**. It provides code reusability.



Polymorphism

The word **polymorphism** is derived from the two words i.e. **ploy** and **morphs**. Poly means many and morphs means forms. It allows us to create methods with the same name but different method signatures. It allows the developer to create clean, sensible, readable, and resilient code.



The above figure best describes the concepts of polymorphism. A person plays an employee role in the office, father and husband role in the home.

1.4 Advantages and Disadvantages of OOP

Advantages:

1. **Modularity**:
   - OOP promotes the organization of code into small, manageable pieces (objects) that can be developed, tested, and maintained independently. This enhances code reusability and maintainability.
2. **Code Reusability**:
   - Through concepts like inheritance and composition, OOP allows for the reuse of code across different parts of a program or in different programs altogether. This can lead to significant time and effort savings.
3. **Flexibility and Extensibility**:
   - OOP allows for easy modification and extension of existing code. New classes and methods can be added without affecting the functioning of the existing codebase.
4. **Encapsulation**:
   - Encapsulation hides the internal state of an object and restricts direct access to its data. This reduces the likelihood of unintended interference, making the code more robust and secure.
5. **Polymorphism**:
   - Polymorphism enables objects to take on multiple forms or types. This allows for more flexible and adaptable code, as the same method can behave differently depending on the object it is called on.

6. **Easier Debugging and Maintenance**:
    - OOP's modular structure and encapsulation make it easier to identify and fix bugs. Changes made to one part of the code are less likely to affect other parts, reducing the risk of introducing new bugs.

Disadvantages:

1. **Steep Learning Curve**:
    - OOP can be more challenging for beginners to grasp compared to procedural programming. Understanding concepts like inheritance, polymorphism, and abstraction may require more time and practice.
2. **Overhead**:
    - OOP can sometimes involve more complex code structures, which may result in slightly slower execution times and increased memory usage compared to more streamlined procedural code.
3. **Not Suitable for All Types of Projects**:
    - While OOP is well-suited for large, complex projects, it may be overkill for simpler applications. In some cases, a procedural or functional approach might be more efficient.
4. **Difficulty in Designing Proper Class Hierarchies**:
    - Designing effective class hierarchies and relationships can be challenging. Poorly designed class structures can lead to code that is hard to understand, maintain, and extend.
5. **Potential for Inefficient Resource Usage**:
    - If not designed carefully, OOP programs can lead to inefficient use of system resources due to the overhead of managing objects and their interactions.
6. **Difficulty in Debugging Inheritance Issues**:
    - Inheritance can sometimes lead to complex relationships between classes, making it harder to track down and fix issues related to method overriding or conflicts in inherited behavior.