1.Object Oriented Programming Concepts

Procedural language: Procedural language is a type of computer programming language that specifies a series of well-structured steps and procedures within its programming context to compose a program. It contains a systematic order of statements, functions and commands to complete a computational task or program.

Procedural language is also known as imperative language.

Characteristics:

- **Predefined functions:** A predefined function is a function available in a procedural programming language from a library of available functions. These functions allow a programmer to complete common tasks without creating the required code themselves. This can help a developer save time during production.
- **Local variables:** A local variable is a programming variable that has a local scope of use. This means the variable only functions in the function in which the developer defines it. Local variables only work in this capacity, so they can cause code to fail, leaving a task unfinished if a professional or user attempts to use the variable in a method outside of its scope.
- **Global variables:** Global variables increase functionality when local variables are insufficient. Developers can use global variables in nearly all functions. When defined globally, a variable makes itself available to all methods and functions in the code, allowing the developer to access key data throughout the program's many procedures.
- **Parameter passing:** Parameters are the data values that transfer from each function within a code sequence. When using parameter passing, developers place input parameters into a module or device and receive output parameters in return. Parameters allow a programmer to assign passed data to variables, equations and commands within a function to complete computing actions on the data.
- **Modularity:** Modularity is a structure in which a developer divides the functionality of its code into a series of smaller blocks. The programmer can then call these blocks, often called methods or functions in procedural programming languages, in their code to access them. This makes important functions repeatable to create a more efficient setup code, compared to one that requires the programmer to reuse the same code at multiple points when including a task that they need more than once.
- **Top-down approach:** Procedural programming relies on a top-down approach to design and creation. Under this approach, a developer first defines the primary goal of the program, then assesses the components required to complete it. They may choose to further subdivide some components of the larger goal into smaller modules. From there, the developer can begin creating their modules to build the code required to run their program.

Advantages:

- **Versatility:** Procedural programming is a versatile paradigm that allows developers to create coding projects that accomplish significantly varied goals. With procedural programming languages designed for many different types of development projects,

1.Object Oriented Programming Concepts

including software and web development, there's likely an effective procedural programming language you can use to accomplish your goals.

- **Simplicity:** Procedural programming is a relatively simple approach to computer programming. This is why many developers start working with procedural programming languages, as they provide a foundation for coding that the developer can apply as they learn other languages, such as an object-oriented language.
- **Accessibility:** Many popular programming languages use procedural programming, so there are many resources available to an aspiring developer hoping to learn them. This includes both paid courses and free online resources and communities you can access when you encounter challenges, which can help expedite your development.

Disadvantages:

- **Complexity:** The simplicity of procedural programming languages can create a challenge when you're attempting to make complex programs. Often, choosing a language with an object-oriented approach may be easier for more in-depth projects.
- **Troubleshooting:** The use of global data in procedural programming languages can make it challenging to identify the source of errors in your code. This can cause a complex debugging process, adding time to your development schedule.
- **Strict data types:** Data in procedural programming languages is immutable. This means you can't change its structure or functionality after creation, which can be limiting, compared to the nearby option in other languages.

Object oriented programming (OOP):

OOP: This is a technique of Programming that provides a collection of reusable object that interact with each other to offer a solution of given problem. The main emphasis is on objects.

Characteristics of OOP:

- **Classes**
- **Objects**
- **Encapsulation**
- **Abstraction**
- **Inheritance**
- **Polymorphism**
- **Message Passing**
- **Data Binding**
- **Modularity**
- **Flexibility**
- **Extensibility**
- **Event-driven Programming**

- **Classes:** It is a collection of objects having similar characteristics and it is a blueprint for the objects and its types.

1.Object Oriented Programming Concepts

- **Objects:** Objects are the instance of a class. Objects have properties (data) and behaviours (methods), which can be defined by the class.
- **Encapsulation:** OOP allows for the encapsulation of data and code into a single entity (class), hiding implementation details from the user and protecting the data from external manipulation. **Or** Wrapping up/Binding of data & function in a single unit called class is known as encapsulation.
- **Abstraction:** It is the technique in which we separate essential member of class in form of public interface. It provides a simple view of the object so that a user can easily access objects.
- **Inheritance:** OOP supports inheritance, which allows for the creation of new classes that inherit the properties and methods of existing classes. This promotes code reuse and reduces code duplication.
- **Polymorphism:** OOP supports inheritance, which allows for the creation of new classes that inherit the properties and methods of existing classes. This promotes code reuse and reduces code duplication.
- **Message Passing:** OOP relies on message passing between objects to perform tasks. Objects communicate with each other by sending messages, which trigger specific methods to be executed.
- **Data Binding:** OOP supports two types of data binding - static and dynamic. Static data binding occurs during compile time, where the method to be called is determined at compile time. Dynamic data binding occurs during runtime, where the method to be called is determined at runtime based on the object that is calling it.
- **Modularity:** OOP promotes modularity by breaking down a complex system into smaller, more manageable modules. Each module can be designed and tested separately, making it easier to understand and maintain the system as a whole.
- **Flexibility:** OOP provides flexibility by allowing developers to create objects and classes that can be easily modified and adapted to different situations. This is achieved through the use of interfaces and abstract classes, which provide a level of indirection between the code and the implementation details.
- **Extensibility:** OOP promotes extensibility by allowing developers to add new features to a system without modifying existing code. This is achieved through inheritance and polymorphism, which allow new classes to be created that inherit properties and methods from existing classes.
- **Event-Driven Programming:** OOP is often used for event-driven programming, where the program responds to events triggered by the user or the system. In event-driven programming, objects communicate with each other by sending events, which are handled by specific methods.

1.Object Oriented Programming Concepts

Category	Object-Oriented Programming (OOP)	Procedure Oriented Programming (POP)
Primary focus	Object-oriented programming focuses on objects and their interactions.	Procedure-oriented programming focuses on procedures or functions.
Approach	OOP is a bottom-up approach.	POP is a top-down approach.
Code structure	OOP code is structured in classes and objects.	POP code is structured in functions or procedures.
Data types	OOP has user-defined data types (classes) and built-in data types.	POP has only built-in data types.
Data security	OOP has encapsulation which provides data security.	POP does not provide data security.
Access to data	In OOP, access to data is controlled by methods.	In POP, access to data is controlled by passing parameters to functions.
Code reusability	OOP provides high code reusability through inheritance and polymorphism.	POP has lower code reusability.
Maintenance	OOP code is easier to maintain due to modularity and encapsulation.	POP code can be difficult to maintain because of the lack of modularity.
Code flexibility	OOP provides more flexibility and scalability due to modularity and encapsulation.	POP can be less flexible due to the lack of modularity.

1.Object Oriented Programming Concepts

Category	Object-Oriented Programming (OOP)	Procedure Oriented Programming (POP)
Complexity	OOP is well-suited for managing complex programs.	POP can become cumbersome for managing complex programs.
Ease of debugging	OOP code is easier to debug because of encapsulation and modularity.	POP code can be difficult to debug because of its non-modular nature.
State management	OOP supports state management through object behavior.	POP has limited support for state management.
Functionality expansion	OOP supports easy expansion of functionality through inheritance and polymorphism.	POP has limited support for expanding functionality.
Code execution speed	OOP code can run slower than POP code due to the additional overhead of objects.	POP code can run faster than OOP code due to the lack of overhead.
Memory usage	OOP code can use more memory than POP code due to the overhead of objects.	POP code uses less memory than OOP code.
Code reuse	OOP code is designed to be reusable through inheritance and polymorphism.	POP code can be less reusable due to its non-modular nature.
Code readability	OOP code is often more readable and easier to understand.	POP code can be less readable due to its non-modular nature.
Scalability	OOP is well-suited for scalable	POP can be less scalable.

1.Object Oriented Programming Concepts

Category	Object-Oriented Programming (OOP)	Procedure Oriented Programming (POP)
	applications.	
Code organization	OOP code is well-organized and structured in classes and objects.	POP code can be unorganized and difficult to structure.
Modularization	OOP promotes modularization and code reuse.	POP has limited support for modularization.
Program structure	OOP is well-suited for large programs with complex data relationships.	POP is better suited for smaller programs with simple data relationships.
Type of programming	OOP is a type of programming that is focused on objects and their interactions.	POP is a type of programming that is focused on functions and procedures.
Code maintenance	OOP code is easier to maintain and modify.	POP code can be difficult to maintain and modify.
Code testing	OOP code is easier to test due to modularity and encapsulation.	POP code can be more difficult to test.
Design philosophy	OOP follows a design philosophy that is focused on modeling real-world objects and their behaviors.	POP follows a design philosophy that is focused on decomposing a problem into a sequence of steps or procedures.
Code encapsulation	OOP code is encapsulated, meaning that data and methods are bundled together into a single unit.	POP code is not encapsulated, meaning that data and methods are often defined separately.

1.Object Oriented Programming Concepts

Category	Object-Oriented Programming (OOP)	Procedure Oriented Programming (POP)
Code abstraction	OOP supports code abstraction, meaning that complex systems can be simplified and abstracted into smaller, more manageable components.	POP has limited support for code abstraction.
Error handling	OOP provides robust error handling through exception handling mechanisms.	POP has limited support for error handling.
Programming languages	OOP is commonly associated with languages like Java, C++, and Python.	POP is commonly associated with languages like C and Pascal.

Procedure Oriented programming (POP) Approach:

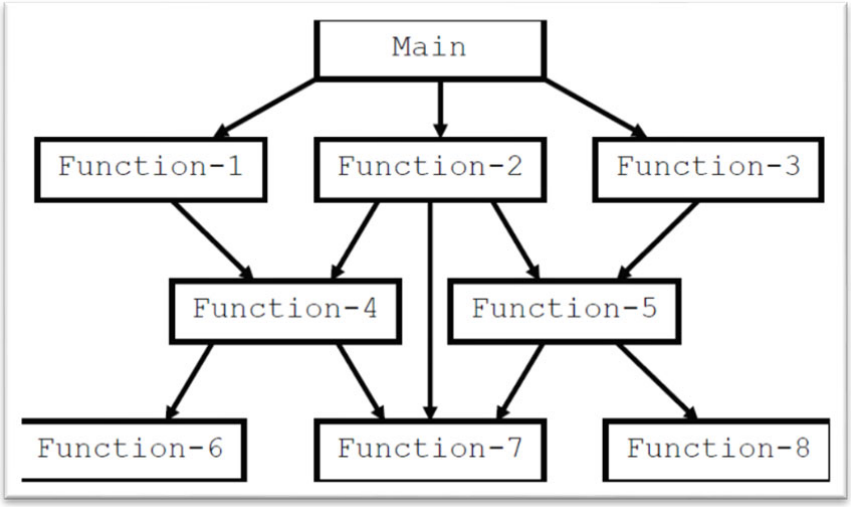
Procedural Oriented Programming (POP) is a programming paradigm that is focused on writing a sequence of instructions to tell the computer what to do step-by-step. In POP, the program is organized around procedures or functions, which are sets of instructions that perform a specific task. These procedures or functions can be called multiple times from different parts of the program.

Language used are: C, Fortran(formula for translation), Pascal etc.

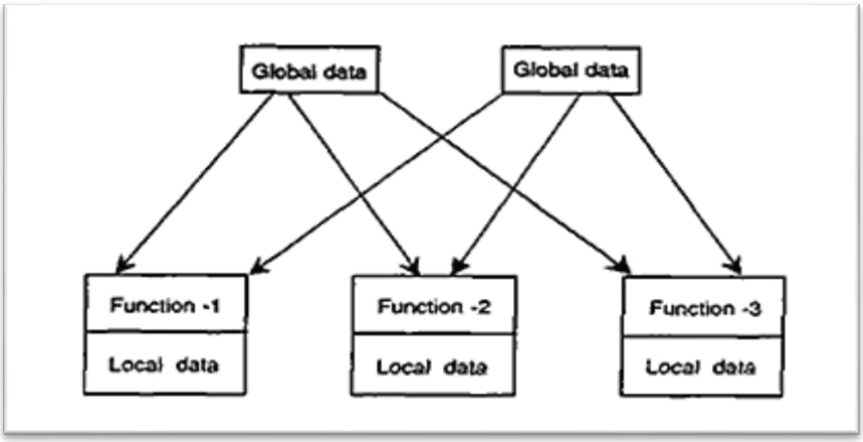
Characteristics:

- It emphasis on doing things (Algorithms).
- Data moves freely in system from function to function.
- Programs are divided into functions.
- Most of the function share global data.
- It follows top-down approach.
- Function transfers data from one form to another.
- Employs top-down approach in program design.

1.Object Oriented Programming Concepts



Structure of procedural oriented approach



Relationship of data and functions

Object Oriented Programming (OOP) Approach:

Object-Oriented Programming (OOP) is a programming paradigm that is based on the concept of objects, which are instances of classes. OOP focuses on organizing code into self-contained objects that have data and behaviour, and communicate with other objects to perform a task.

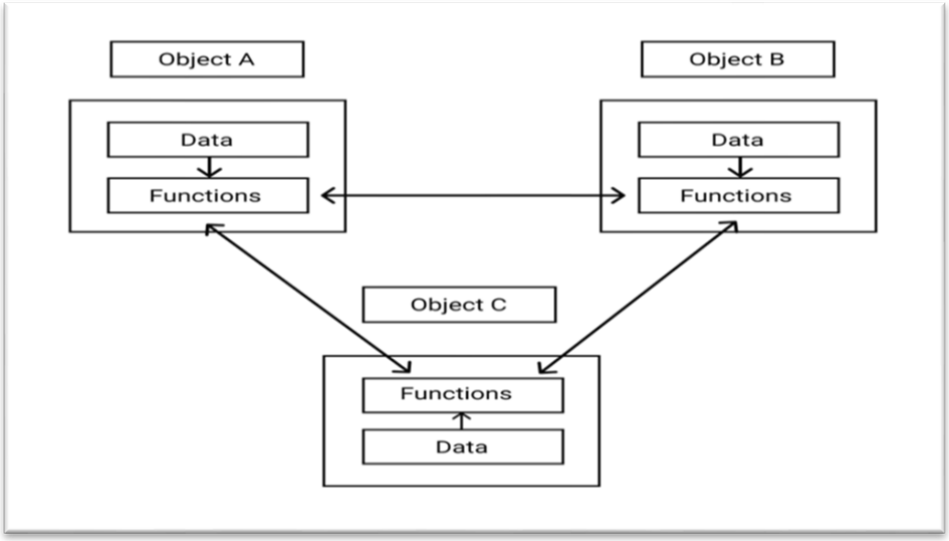
Language used: Java, C++, Python etc.

Characteristics:

- It emphasis on data rather than procedure.
- Programs are divided into smaller objects.
- Data Structures are designed such that they characterised the objects.

1.Object Oriented Programming Concepts

- Functions that operate on data of objects are tied together in data structures.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added.
- Follows bottom-up approach in program design.



Organization of data & functions in OOP

Operators in C++:

1. **Arithmetic operators:** These operators perform basic arithmetic operations on numeric data types. The arithmetic operators in C++ are:
    - + (addition): adds two values
    - - (subtraction): subtracts two values
    - * (multiplication): multiplies two values
    - / (division): divides two values
    - % (modulus): returns the remainder of a division operation

2. **Relational operators:** These operators compare two values and return a Boolean value of true or false. The relational operators in C++ are:
    - == (equal to): checks if two values are equal
    - != (not equal to): checks if two values are not equal
    - < (less than): checks if one value is less than another
    - > (greater than): checks if one value is greater than another
    - <= (less than or equal to): checks if one value is less than or equal to another
    - >= (greater than or equal to): checks if one value is greater than or equal to another

1.Object Oriented Programming Concepts

3. **Logical operators:** These operators perform logical operations on Boolean data types. The logical operators in C++ are:
    - && (logical AND): returns true if both operands are true
    - || (logical OR): returns true if either operand is true
    - ! (logical NOT): returns the opposite of the operand's value

4. **Bitwise operators:** These operators perform bit-level operations on integer data types. The bitwise operators in C++ are:
    - & (bitwise AND): performs bitwise AND operation on two values
    - | (bitwise OR): performs bitwise OR operation on two values
    - ^ (bitwise XOR): performs bitwise XOR operation on two values
    - ~ (bitwise NOT): performs bitwise NOT operation on a value
    - << (left shift): shifts the bits of a value to the left
    - >> (right shift): shifts the bits of a value to the right

5. **Assignment operators:** These operators assign values to variables. The assignment operators in C++ are:
    - = (simple assignment): assigns a value to a variable
    - += (addition assignment): adds a value to a variable and assigns the result to the variable
    - -= (subtraction assignment): subtracts a value from a variable and assigns the result to the variable
    - *= (multiplication assignment): multiplies a variable by a value and assigns the result to the variable
    - /= (division assignment): divides a variable by a value and assigns the result to the variable
    - %= (modulus assignment): calculates the remainder of a division operation and assigns the result to the variable
    - &= (bitwise AND assignment): performs bitwise AND operation on a variable and a value, and assigns the result to the variable
    - |= (bitwise OR assignment): performs bitwise OR operation on a variable and a value, and assigns the result to the variable
    - ^= (bitwise XOR assignment): performs bitwise XOR operation on a variable and a value, and assigns the result to the variable
    - <<= (left shift assignment): shifts the bits of a variable to the left and assigns the result to the variable
    - >>= (right shift assignment): shifts the bits of a variable to the right and assigns the result to the variable.

6. **Increment and decrement operators:** These operators increment or decrement the value of a variable. The increment and decrement operators in C++ are:
    - ++ (increment): increases the value of a variable by 1
    - (decrement): decreases the value of a variable by 1

    These operators can be used in two ways: as prefix (before the variable name) or postfix (after the variable name). The prefix version increments or decrements the

value of the variable before using it in an expression, while the postfix version increments or decrements the value after using it in an expression.

7. **Conditional operator:** This operator is also known as the ternary operator, and is used to assign a value to a variable based on a condition. The conditional operator in C++ is:
    - ? : (conditional operator): evaluates a condition and returns one value if the condition is true, and another value if the condition is false.

8. **Comma operator:** This operator is used to separate multiple expressions in a single statement. The comma operator in C++ is:
    - , (comma operator): evaluates two expressions and returns the value of the second expression.
      The comma operator is typically used in for-loops, where multiple expressions need to be evaluated in a single loop.

9. **Pointer operator:** This operator is used to declare and manipulate pointers. The pointer operator in C++ is:
    - * (pointer operator): declares a pointer variable or dereferences a pointer variable to access the value it points to.

10. **Scope resolution operator:** This operator is used to access members of a class or namespace. The scope resolution operator in C++ is:
    - :: (scope resolution operator): specifies the scope of a variable or function.