**PYTHON**

Python is a high-level, versatile programming language known for its readability and simplicity. It is popular across industries from web development to scientific computing, AI, and more. Its ease of learning and extensive libraries make it an excellent choice for both beginners and experienced developers.

**Functions**

Functions are reusable blocks of code that perform a specific task. They help in organizing code, improving readability, and facilitating reusability. Functions are defined using the **def** keyword followed by the function name and parentheses ( ). Input values (arguments) can be passed into functions via parameters placed within the parentheses. **Parameters** are the variables listed inside the parentheses of a function definition. They represent the input data that a function can accept. An **Argument** is the value that is sent to the function when it is called.

**Types of Arguments**

**Default Arguments -** Parameters with default values specified in the function definition.If the argument is not provided, the default value is used.

**Keyword Arguments -** Arguments passed with the parameter names explicitly mentioned.Allows passing arguments in any order, improving readability.

**Variable-length Arguments -** \*args and \*\*kwargs allow functions to accept a variable number of arguments.\*args collects additional positional arguments into a tuple. \*\*kwargs collects additional keyword arguments into a dictionary.

**Recursive Functions** are functions that allow functions to call themselves, enabling the implementation of recursive algorithms.

**Lambda Functions (Anonymous Functions)** are small, anonymous functions defined using the lambda keyword. They are typically used for short operations and as arguments in higher-order functions.

**Object Oriented Programming**

Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of objects, allowing developers to structure their code by creating objects that encapsulate data (attributes) and behaviors (methods). In Python, OOP is implemented through classes and objects.

**CLASS**

A class is a blueprint or a template for creating objects. It defines attributes and methods that describe the object's behavior. Classes are created using the class keyword.

**OBJECT**

An object is an instance of a class. It is created using the class as a blueprint. Objects have their own unique data and share the methods defined in the class.

**Encapsulation** Encapsulation refers to the bundling of data (attributes) and the methods (functions) that operate on that data within a class. It hides the internal state of an object from the outside and only exposes the necessary functionalities, promoting data integrity and security.

**Abstraction** Abstraction focuses on displaying essential features of an object and hiding its complexities. It allows developers to create simplified models that capture the essential characteristics without including unnecessary details.

**Inheritance** Inheritance is the mechanism by which one class (subclass or child class) can inherit properties and behavior from another class (superclass or parent class). This promotes code reusability and allows the creation of a hierarchy of classes with shared attributes and methods. **Types of Inheritance Single Inheritance -**  A class inherits from only one superclass **Multiple Inheritance -** A class inherits from multiple superclasses. . **Multilevel Inheritance** - This involves a chain of inheritance where a derived class (subclass) becomes a superclass for another class. For instance, Class C inherits from Class B, which in turn inherits from Class A. **Hierarchical Inheritance -** Multiple classes are derived from a single superclass. Several subclasses inherit from one common superclass. **Hybrid (or Mixed) Inheritance -**  This is a combination of multiple types of inheritance. It involves a mix of two or more types of inheritance. For instance, combining multiple inheritance with multilevel inheritance.

**Polymorphism** Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to be used for entities of different types, providing a way for objects to take multiple forms and behave differently based on the context.