**Introduction to Computer Languages**

A computer programming language is a set of rules and syntax that is used to create programs that can be executed by a computer. These languages allow developers to specify the steps that a computer should take to perform a task, and they form the basis for creating programs that can be used to solve a wide range of problems. There are many different programming languages, each with its own unique set of features and capabilities. Some common programming languages include C++, Java, Python, and JavaScript.

**Programming Language**

The sequences of set of instructions formatted in different **syntaxes** and **styles** used to find the solution of a problem is known as a **programming language** or a **computer language**.

**Program**

The set of instructions of any **programming language** given to the computer to solve a problem is known as a **program**.

**Programming**

The procedure used to write the **commands** and **statements** of a **programming language** to find the solution to a problem is known as **programming**.

**Programmer**

A person who writes **programs** is referred to as a **programmer**.

Now, there are **two** major **classifications** of **programming languages**. They are as follows:

1. **System Programming Languages:** **System programming languages** are a type of **programming language** that are used to write **low-level programs** that interact directly with a **computer's hardware** or **operating System**. These **languages** are typically more **difficult** to **learn** and **use** than **higher-level languages**, but they offer a level of **control** and **efficiency** that is **necessary** for certain types of **programs**. Examples of **system programming languages** include **C**, **C++**, **and assembly languages**. These languages are typically used to write **operating systems**, **device drivers**, **and other types of system software**.
2. **Application Programming Languages: Application programming languages** are a type of **programming language** that is used to **develop software applications** that can be **run on a computer or other device**. These **languages** are generally **easier** to **learn** and **use** than **system programming languages**, and they are **designed** to be **more user-friendly and higher-level**. Some examples of **application programming languages** include **Java**, **Python**, **and C#**.These **languages** are used to create a wide range of **software applications, including web applications, mobile apps, desktop applications, and more**.

**Introduction to Object-Oriented Programming**

Object-oriented programming (OOP) is a programming paradigm that is based on the concept of **"objects"**, which can contain **data** and **code** that **manipulates** that **data**. In object-oriented programming, **objects** are used to represent **real-world** **entities**, and the **code** that **operates** on those **objects** is **organized** into **classes**. OOP languages are designed to make it easier to create, modify, and maintain complex programs by providing a structure for organizing code. Some examples of object-oriented programming languages include Java, C++, and Python.

**What is an Object?**

In object-oriented programming, an object is a **self-contained unit of code and data** that **represents** a **real-world entity**. **Objects** are defined by their **class**, which **specifies** the **characteristics** and **behaviour** of the **object**. An **object** is **created** from a **class** and is also referred to as an **instance** of that **class**. Each **object** has its own **unique set** of **data** and **code**, which is known as its **attributes** and **methods**, **respectively**. **Objects** can **interact** with **each other** through **methods**, and they can be **modified** and **updated** as needed. OOP languages are designed to make it easier to manage and modify complex programs by providing a structure for organizing code and data into objects.

Every object has the following characteristics:

1. **Identity:** The name by which an **object** is recognized is known its **identity**. Example: A laptop or a computer.
2. **State:** It refers to whether an **object** is in **processed** or **unprocessed** **state**. Example: A laptop or a computer may be in any of the following states: “on state”, “off state”, or “out of order state”.
3. **Behaviour:** The **behaviour** of an **object** refers to what that **object can do**.

**Principles of Object-Oriented Programming**

Every Object-Oriented Programming (OOP) Language include objects with the following principles:

1. **Abstraction (Data Abstraction): Data abstraction** is a **programming** **concept** that refers to the **idea** of **hiding** the **implementation** **details** of a **piece** **of** **code** **from** **the** **user**. In other words, it is the process of only presenting the essential information to the user, while hiding the details of how the code is implemented. This allows the user to interact with the code without needing to know how it works internally. Data abstraction is often used to simplify complex code and make it easier to understand and use. It is a fundamental concept in object-oriented programming, and it is achieved using abstract data types and interfaces. Abstract data types are used to define the characteristics of a piece of data, while interfaces define the behaviours that an object can perform. Together, these concepts allow developers to create flexible and modular code that can be easily reused and modified.
2. **Encapsulation: Encapsulation** is a **programming** **concept** that refers to the **bundling of data** and **the methods that operate on that data** **within** a **single** **unit**, or **object**. In object-oriented programming, encapsulation is achieved by defining the data and methods within a class, and then creating objects from that class. Encapsulation helps to protect the data within an object from being accessed or modified by code outside of the object, and it also allows developers to change the implementation of the object's methods without affecting the rest of the code. This makes the code more flexible and easier to maintain. Encapsulation is often used in conjunction with data abstraction and inheritance, which are other fundamental concepts in object-oriented programming.
3. **Inheritance: Inheritance** is a **programming** **concept** that refers to the **ability** **of** **a** **class** **to** **inherit** **the** **properties** and **methods** of **another** **class**. In object-oriented programming, a subclass can be created from an existing class, known as the superclass. The subclass will automatically have all the attributes and methods of the superclass, and it can also have additional attributes and methods of its own. This allows developers to create a hierarchy of classes, where more specialized classes can be derived from more general ones. Inheritance is a useful way to reuse code and create a more organized and efficient program. It is often used in conjunction with data abstraction and encapsulation, which are other fundamental concepts in object-oriented programming.
4. **Polymorphism: Polymorphism** is a **programming** **concept** that refers to the **ability** of a **class** or **object** to **take** on **multiple** **forms**. In object-oriented programming, polymorphism is achieved using inheritance and method overriding. Method overriding occurs when a subclass has a method with the same name and parameters as a method in the superclass, but with a different implementation. This allows the subclass to have its own unique behaviour, while still being able to reuse the code from the superclass. Polymorphism is a useful way to create code that is flexible and adaptable, and it is a fundamental concept in object-oriented programming. It allows developers to create code that can be easily extended and modified, and it helps to make programs more efficient and easier to maintain.