<https://www.geeksforgeeks.org/a-step-by-step-approach-to-learn-object-oriented-programming/>

* PATTERNS
* REACT (HOOKS), NODE (CRUD), PROJECTS, JS, PHP, DSA, APPTITUDE EXAM (BASIC KNOWLAGE), NEXT JS (EXTRA).

**// Basics :**

**Object-Oriented Programming (OOP)** is a programming approach that organizes code into objects, which represent **real-world entities** with attributes (data) and methods (functions). It focuses on four key principles: **Encapsulation, Abstraction, Inheritance,** and **Polymorphism** to make code reusable, modular, and easier to manage.

A **class** is a blueprint that defines the structure and behavior of objects. It represents a group of objects that share common attributes and functions.

An **object** is an instance of a class. It is a specific entity that has properties and behaviors defined by its class.

**Encapsulation** is the concept of bundling data and methods that operate on that data into a single unit (class). It also involves restricting direct access to some of an object’s components, typically by using access modifiers (private, protected, public) to control visibility.

* Members declared as **public** are accessible from **anywhere** in the program.
* Members declared as **private** can only be accessed **within the class**.
* Members declared as **protected** can be accessed **within the class and by subclasses**.

**Abstraction** is the process of hiding complex implementation details and exposing only the essential features of an object. It focuses on "what" an object does rather than "how" it does it.

**Inheritance** allows a class (child class) to acquire the properties and behaviors of another class (parent class). It promotes code reuse and establishes a parent-child relationship.

**Polymorphism** means "many forms." It allows objects of different classes to be treated as objects of a common superclass. It is often implemented using method overriding or operator overloading.

* When a class has multiple methods with the **same name** but **different parameters**, it is called **overloading**.
* When a **child class** has a method with the **same name** and **parameters** as in its **parent class**, but gives it a different behavior, it is called **overriding**.

**Example :**

class Animal:

def sound(self):

return "Some sound"

class Dog(Animal):

def sound(self): # Overrides the parent class method

return "Bark"

**1. static Keyword**

* **Definition**: Used to define class-level members (variables or methods) that belong to the class itself, rather than to any specific instance of the class.

**2. virtual Keyword**

* **Definition**: Used in a base class to indicate that a method can be overridden by derived classes.

**3. abstract Keyword**

* **Definition**: Used to define a method or class that cannot be instantiated or fully implemented in the base class. Derived classes must provide an implementation for abstract methods.

**4. final Keyword**

* **Definition**: Used to prevent further modification. For methods, it prevents overriding; for classes, it prevents inheritance; for variables, it makes the value constant after initialization.

**5. explicit Keyword**

* **Definition**: Used in C++ to mark constructors or conversion operators that should not be automatically invoked by the compiler in implicit conversions.

**6. this Keyword**

* **Definition**: Refers to the current instance of the class. Used to access instance members and differentiate between instance variables and parameters with the same name.

**7. new Keyword**

* **Definition**: Used to create new instances of objects or to hide members in derived classes that have the same name as those in the base class.

**8. const Keyword**

* **Definition**: Used to declare variables whose value cannot be modified after initialization. It can also be applied to pointers and methods.

**9. super Keyword**

* **Definition**: Used to refer to the parent class. It allows access to the parent class’s methods and constructors from the derived class.