### **🌟 What is OOP?**

**Object-Oriented Programming (OOP)** is like organizing your code by grouping data and actions together into "objects" — just like real-life things!

### 🧒 Real-Life Analogy:

| **Real-Life Object** | **Data (What it has)** | **Actions (What it does)** |
| --- | --- | --- |
| Car | Color, Model, Year | Start, Stop, Drive |
| Dog | Name, Breed, Age | Bark, Eat, Sleep |
| Student | Name, Grade | Study, Take Test |

### **🔹 1. Class & Object**

* **Class** : A blueprint or template.
* **Object** : A real thing made from the blueprint.

class Dog {

constructor(name, breed) {

this.name = name;

this.breed = breed;

}

bark() {

console.log(`${this.name} says: Woof!`);

}

}

// Create objects

let dog1 = new Dog("Buddy", "Golden Retriever");

let dog2 = new Dog("Max", "Poodle");

dog1.bark(); // Buddy says: Woof!

dog2.bark(); // Max says: Woof!

🎯 Activity: Create a **Car** class with color, model, and method like **drive(), hoot()**.

### 🔹 2. Encapsulation

Keep related data and functions inside one class. Protect data from bad values.

class Person {

constructor(name, age) {

this.name = name;

this.\_age = age; // pretend private

}

get age() {

return this.\_age;

}

set age(newAge) {

if (newAge > 0) this.\_age = newAge;

}

}

🎯 Activity: Try changing age to a negative number and see what happens.

### 🔹 3. Abstraction

Hide complex details. Users only need to know how to use something, not how it works.

class CoffeeMachine {

#startEngine() {

console.log("Heating water...");

}

brew() {

console.log("Starting coffee...");

this.#startEngine();

console.log("Coffee is ready!");

}

}

🎯 Analogy: When you press “brew” on a machine, you don’t need to know how the engine works.

### 🔹 4. Inheritance

One class can inherit properties and methods from another.

class Animal {

eat() {

console.log("Animal is eating.");

}

}

class Cat extends Animal {

meow() {

console.log("Meow!");

}

}

let whiskers = new Cat();

jerry.eat(); // Inherited

jerry.meow(); // Own method

🎯 Analogy: Like inheriting traits from your parents — kids get some features automatically.

### 🔹 5. Polymorphism

Same method name, different behaviors.

class Shape {

area() {

return 0;

}

}

class Square extends Shape {

area() {

return this.side \* this.side;

}

}

class Circle extends Shape {

area() {

return Math.PI \* this.radius \*\* 2;

}

}

🎯 Analogy: Different types of calculators do math differently but have same buttons.

## 🛠️ 3. Suggested Activities for Practice

| **Activity** | **Description** |
| --- | --- |
| 🐾 Build a Pet Simulator | Create classes for**Dog**,**Cat**, etc., with properties and methods |
| 🚗 Car Race Game | Use OOP to track speed, position, and collisions |
| 📚 Library System | Create**Book**,**User**, and**Library**classes |
| 💬 Chat App Profile | Create**User**class with name, status, messages, etc. |

## 📖 4. Visual Learning Tools

* A **class diagram**

+-------------------+

| Person |

+-------------------+

| - name |

| - \_age |

+-------------------+

| + getName() |

| + setName() |

| + greet() |

+-------------------+

Or use tools like:

* [draw.io](https://www.draw.io/) – free diagram tool

## 📅 5. Suggested Lesson Plan (Over 1 Week)

## 📝 Summary Sheet

| **Term** | **Meaning** | **Example** |
| --- | --- | --- |
| Class | Blueprint for creating objects | **class Dog** |
| Object | Instance of a class | **let myDog = new Dog()** |
| Constructor | Sets up initial data | **constructor(name, age)** |
| Encapsulation | Group data and behavior | Keeping variables and functions together |
| Abstraction | Hide complexity | Only show necessary parts |
| Inheritance | Reuse code | **class Cat extends Animal** |
| Polymorphism | Same method, different results | **.area()**behaves differently per shape |