OOPS

Object-Oriented Programming (OOPs) is a way to **organize** and **design software** using **"objects"** which are instances of **"classes."** Classes define attributes and behaviors, while objects are specific instances of those classes. OOP helps in creating reusable, modular, and maintainable code.

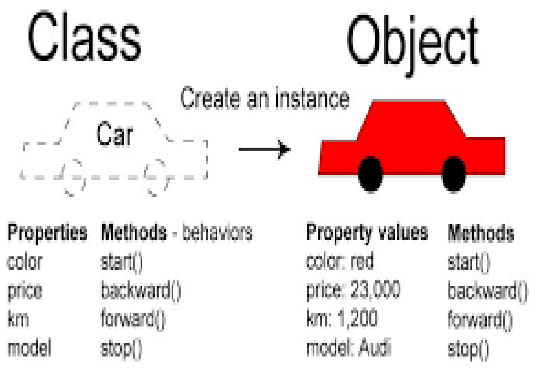
It offers several benefits like:

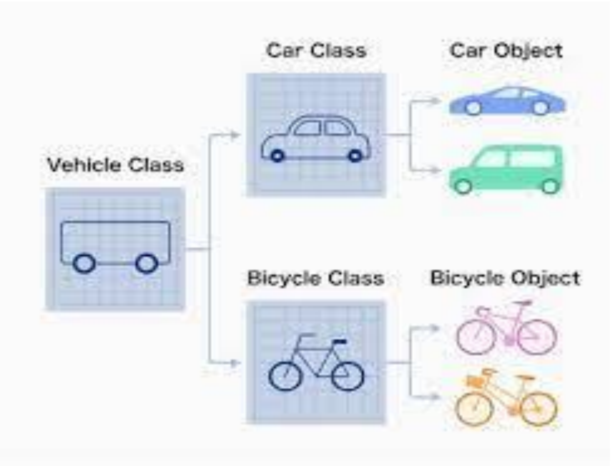
1. **Modularity:** Code is organized into classes, making it easier to manage and understand.
2. **Reusability:** Classes and objects can be reused across different programs.
3. **Scalability:** Easier to manage and scale large projects.
4. **Maintainability:** Code is easier to maintain and update.
5. **Encapsulation:** Data and methods are bundled together, enhancing security and data integrity.
6. **Inheritance:** New classes can inherit properties and methods from existing ones, reducing redundancy.
7. **Polymorphism:** Allows objects to be treated as instances of their parent class, enabling flexible and dynamic code.

CLASSES & OBJECTS

Objects: ***Entities*** in ***real world***. An object is an instance of a class, representing a specific entity that can use the class's attributes and methods.

Classes: ***Blueprint*** of these ***entities.*** It defines attributes (data) and methods (functions) that the objects will have





DEFINING A CLASS:

```
class ClassName {
public:
    // Attributes (or member variables)
    int attribute1;
    double attribute2;

    // Methods (or member functions)
    void method1() {
        // Method implementation
    }

    double method2(int param) {
        // Method implementation
        return param * attribute2;
    }
};
```

CREATING AN OBJECT:

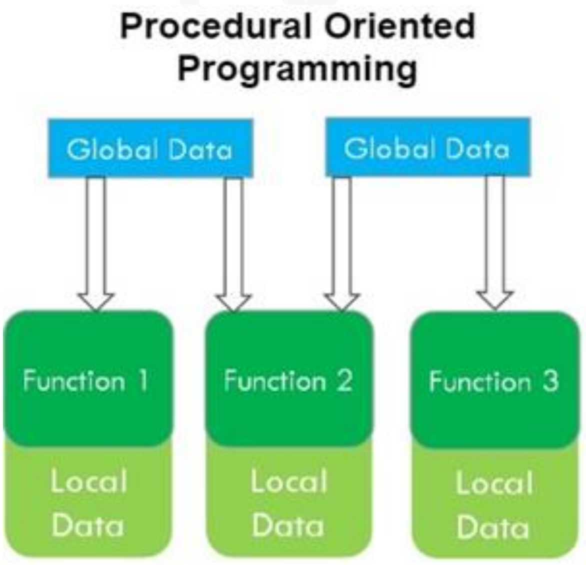
```
int main() {
    // Creating an object of ClassName
    ClassName objectName;

    // Accessing attributes and methods
    objectName.attribute1 = 10;
    objectName.attribute2 = 5.5;

    objectName.method1();
    double result = objectName.method2(3);

    return 0;
}
```

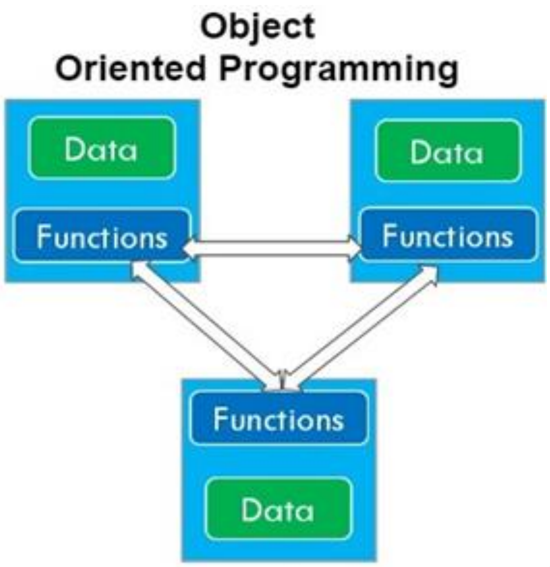
CODE EXAMPLE:

```
#include <iostream>
#include <string>
Using namespace std;

// Class definition for Car
class Car {
private:
    string brand;
    string model;
    int year;
    float mileage;

public:
    // Constructor
    Car(string b, string m, int y, float ml) {
        brand = b;
        model = m;
        year = y;
        mileage = ml;
    }

    void displayDetails() {
        cout << "Brand: " << brand << endl;
        cout << "Model: " << model << endl;
        cout << "Year: " << year << endl;
        cout << "Mileage: " << mileage << " km" << endl;
    }

    // Method to update mileage
    void updateMileage(float newMileage) {
        mileage = newMileage;
    }
};

int main() {
    // Creating a Car object
    Car myCar("Toyota", "Corolla", 2020, 15000);

    // Displaying car details
    myCar.displayDetails();

    // Updating mileage
    myCar.updateMileage(20000);

    // Displaying updated car details
    myCar.displayDetails();

    return 0;
}
```

DIFFERENCE BETWEEN POP & OOP

Feature	**Procedural Oriented Programming (POP)**	**Object Oriented Programming (OOP)**
Approach	Follows a top-down approach	Follows a bottom-up approach
Structure	Program is divided into functions	Program is divided into objects
Data Access	Data is global and shared by functions	Data is encapsulated in objects
Data Security	Less secure as data is exposed	More secure due to encapsulation
Focus	Focuses on functions	Focuses on objects
Code Reusability	Less reusability	High reusability through inheritance
Inheritance	Does not support inheritance	Supports inheritance
Polymorphism	Does not support polymorphism	Supports polymorphism
Examples	C, Pascal	C++, Java, Python


Procedural Oriented Programming
Global Data
Global Data
Function 1
Local Data
Function 2
Local Data
Function 3
Local Data


Object Oriented Programming
Data
Functions
Data
Functions
Functions
Data