## What is Object Oriented Programming?

Object Oriented programming (OOP) is a programming paradigm that relies on the concept of **classes** and **objects**. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects. There are many object-oriented programming languages including JavaScript, [C++](https://www.educative.io/blog/how-to-learn-cpp-the-guide-for-beginners), [Java](https://www.educative.io/blog/object-oriented-programming-concepts-java), and [Python](https://www.educative.io/blog/how-to-use-oop-in-python).

A **class** is an abstract blueprint used to create more specific, concrete objects. Classes often represent broad categories, like Car or Dog that share **attributes**. These classes define what attributes an instance of this type will have, like color, but not the value of those attributes for a specific object.

Classes can also contain functions, called **methods** available only to objects of that type. These functions are defined within the class and perform some action helpful to that specific type of object.

For example, our Car class may have a method repaint that changes the color attribute of our car. This function is only helpful to objects of type Car, so we declare it within the Car class thus making it a method.

Class templates are used as a blueprint to create individual **objects**. These represent specific examples of the abstract class, like myCar or goldenRetriever. Each object can have unique values to the properties defined in the class.

For example, say we created a class, Car, to contain all the properties a car must have, color, brand, and model. We then create an instance of a Car type object, myCar to represent my specific car.

We could then set the value of the properties defined in the class to describe my car, without affecting other objects or the class template.

We can then reuse this class to represent any number of cars.

**Object-oriented programming** (**OOP**) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) based on the concept of "[**objects**](https://en.wikipedia.org/wiki/Object_(computer_science))", which can contain [data](https://en.wikipedia.org/wiki/Data) and code: data in the form of [fields](https://en.wikipedia.org/wiki/Field_(computer_science)) (often known as *attributes* or *properties*), and code, in the form of procedures (often known as [*methods*](https://en.wikipedia.org/wiki/Method_(computer_science))).

Object-oriented programming combines a group of variables (properties) and functions (methods) into a unit called an "object." These objects are organized into classes where individual objects can be grouped together. OOP can help you consider objects in a program's code and the different actions that could happen in relation to the objects.

This programming style widely exists in commonly used programming languages like Java, C++ and PHP. These languages help simplify the structure and organization of software programs. Programmers often use OOP when they need to create complex programs.

**The four basics of object-oriented programming:**

Object-oriented programming has four basic concepts: encapsulation, abstraction, inheritance and polymorphism. Even if these concepts seem incredibly complex, understanding the general framework of how they work will help you understand the basics of a computer program. Here are the four basic theories and what they entail:

* Encapsulation
* Abstraction
* Inheritance
* Polymorphism

1. **Encapsulation:** In computer networking, encapsulation is a method of designing modular communication protocols in which logically separate functions in the network are abstracted from their underlying structures by inclusion or information hiding within higher level objects.

The different objects inside of each program will try to communicate with each other automatically. If a programmer wants to stop objects from interacting with each other, they need to be encapsulated in individual classes. Through the process of encapsulation, classes cannot change or interact with the specific variables and functions of an object.

Just like a pill "encapsulates" or contains the medication inside of its coating, the principle of encapsulation works in a digital way to form a protective barrier around the information that separates it from the rest of the code. Programmers can replicate this object throughout different parts of the program or other programs.

### Example of encapsulation:

Beginning programmers may better understand this concept in relation to how a browser functions. Browsers have local storage objects that allow you to store data locally. These objects have properties, like length, which turns the number of objects into storage, along with methods like (remove item) and (set item).

Another way to consider encapsulation is in terms of an employee's pay. The properties of an employee can include base salary, overtime and rate with a method called factor wage. Code written in an encapsulated, object-oriented way functions with fewer and fewer parameters. The fewer the number of parameters, the easier it is to use and maintain that function.

### Abstraction: Abstraction is like an extension of encapsulation because it hides certain properties and methods from the outside code to make the interface of the objects simpler. Programmers use abstraction for several beneficial reasons. Overall, abstraction helps isolate the impact of changes made to the code so that if something goes wrong, the change will only affect the variables shown and not the outside code.

### Example of abstraction:

Think of a stereo system as an object with a complex logic board on the inside. It has buttons on the outside to allow for interaction with the object. When you press any of the buttons, you're not thinking about what happens on the inside because you can't see it. Even though you can't see the logic board completing functions as a result of pressing a button, it's still performing actions. This is an abstraction, which can be used to understand the concept of programming.

Another way to understand this is to consider the human body. The skin acts as an abstraction to hide the internal body parts responsible for bodily functions like digesting and walking.

### Inheritance:

Using this concept, programmers can extend the functionality of the code's existing classes to eliminate repetitive code. For instance, elements of HTML code that include a text box, select field and checkbox have certain properties in common with specific methods.

Instead of redefining the properties and methods for every type of HTML element, you can define them once in a generic object. Naming that object something like "HTMLElement" will cause other objects to inherit its properties and methods so you can reduce unnecessary code.

The main object is the superclass and all objects that follow it are subclasses. Subclasses can have separate elements while adding what they need from the superclass.

### Example of inheritance:

Consider two classes. One is the superclass (parent) while the subclass (child) will inherit the properties of the parent class and modify its behavior. Programmers applying the technique of inheritance arrange these classes into a hierarchy of "is-a-type-of" relationships.

For instance, in the animal world, an insect would be a superclass. All insects share similar properties, such as having six legs and an exoskeleton. Grasshoppers and ants are both insects and inherited similar properties.

### Polymorphism:

This technique meaning "many forms or shapes" allows programmers to render multiple HTML elements depending on the type of object. This concept allows programmers to redefine the way something works by changing how it is done or by changing the parts in which it is done. Terms of polymorphism are called overriding and overloading.

### Example of polymorphism:

To better understand the two terms of polymorphism called overloading and overriding, it helps to visualize the process of walking. Babies learn to crawl first by using their arms and legs. Once they learn to stand and walk, they are ultimately changing the body part used to accomplish the act of walking. This term of polymorphism is called overloading.

To understand the next term of overriding, think of how you naturally walk in the direction you are facing. When you stop and walk backward, this changes the direction of your path and also the mechanism of function. You are overriding the natural action that you usually complete.

**Building blocks of OOP**

Next, we’ll take a deeper look at each of the fundamental building blocks of an OOP program used above:

* Classes
* Objects
* Methods
* Attributes

### Classes

In a nutshell, classes are essentially **user defined data types**. Classes are where we create a blueprint for the structure of methods and attributes. Individual objects are instantiated, or created from this blueprint.

Classes contain fields for attributes, and methods for behaviors. In our Dog class example, attributes include name & birthday, while methods include bark() and updateAttendance().

Here’s a code snippet demonstrating how to program a Dog class using the [JavaScript](https://www.educative.io/blog/javascript-oop-primer) language.

Remember the class is a template for modeling a dog, and an object is instantiated from the class representing an individual real world thing.

***Enjoying the article?*** Scroll down to [*sign up*](https://www.educative.io/blog/blog-newsletter-annoucement) for our free, bi-monthly newsletter.

### Objects:

Of course OOP includes objects! Objects are **instances of classes** created with specific data, for example in the code snippet below Rufus is an instance of the Dog class.

When the new class Dog is called:

* A new object is created named rufus
* The constructor runs name & birthday arguments, and assigns values

**Programming vocabulary:**

In JavaScript objects are a type of variable. This may cause confusion, because objects can also be declared without a class template in JavaScript, as shown at the beginning.

Objects have states and behaviors. State is defined by data: things like names, birthday, and other information you’d want to store about a dog. Behaviors are methods, the object can undertake.

| N/A | What is it? | Information Contained | Actions | Example |
| --- | --- | --- | --- | --- |
| Classes | Blueprint | Attributes | Behaviors defined through methods | Dog Template |
| Objects | Instance | State, Data | Methods | Rufus, Fluffy |

### Attributes:

Attributes are the information that is stored. Attributes are defined in the Class template. When objects are instantiated individual objects contain data stored in the Attributes field.

The state of an object is defined by the data in the object’s attributes fields. For example, a puppy and a dog might be treated differently at pet camp. The birthday could define the state of an object, and allow the software to handle dogs of different ages differently.

### Methods:

Methods represent behaviors. Methods perform actions; methods might return information about an object, or update an object’s data. The method’s code is defined in the class definition.

When individual objects are instantiated, these objects can call the methods defined in the class. In the code snippet below, the bark method is defined in Dog class, and the bark() method is called on the Rufus object.

class Dog {

    //Declare protected (private) fields

    \_attendance = 0;

    constructor(name, birthday) {

        this.namee = name;

        this.birthday = birthday;

    }

    getAge() {

        //Getter

        return this.calcAge();

    }

    calcAge() {

        //calculate age using today's date and birthday

        return this.calcAge();

    }

    bark() {

        return console.log("Woof!");

    }

    updateAttendance() {

        //add a day to the dog's attendance days at the petsitters

        this.\_attendance++;

    }

}