# OOP

Object-Oriented Programming (OOP) is a programming paradigm in computer science 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++, Java, and Python.

## Benefits of OOP for software engineering

* OOP models complex things as reproducible, simple structures
* Reusable, OOP objects can be used across programs
* Polymorphism allows for class-specific behavior
* Easier to debug, classes often contain all applicable information to them
* Securely protects sensitive information through encapsulation

# Four Principles of OOP

The four pillars of object-oriented programming are:

* **Inheritance**: child classes inherit data and behaviors from the parent class
* **Encapsulation**: containing information in an object, exposing only selected information
* **Abstraction**: only exposing high-level public methods for accessing an object
* **Polymorphism**: many methods can do the same task

## Inheritance

Inheritance allows classes to inherit features of other classes. Put another way, parent classes extend attributes and behaviors to child classes. Inheritance supports reusability.

If basic attributes and behaviors are defined in a parent class, child classes can be created, extending the functionality of the parent class and adding additional attributes and behaviors.

For example, herding dogs have the unique ability to herd animals. In other words, all herding dogs are dogs, but not all dogs are herding dogs. We represent this difference by creating a child class HerdingDog from the parent class Dog, and then adding the unique herd() behavior.

The benefits of inheritance are programs can create a generic parent class and then create more specific child classes as needed. This simplifies programming because instead of recreating the structure of the Dog class multiple times, child classes automatically gain access to functionalities within their parent class.

## Encapsulation

Encapsulation means containing all important information inside an object, and only exposing selected information to the outside world. Attributes and behaviors are defined by code inside the class template.

Then, when an object is instantiated from the class, the data and methods are encapsulated in that object. Encapsulation hides the internal software code implementation inside a class and hides the internal data of inside objects.

Encapsulation requires defining some fields as private and some as public.

* **Private**/ Internal interface: methods and properties accessible from other methods of the same class.
* **Public** / External Interface: methods and properties accessible from outside the class.

The benefits of encapsulation are summarized here:

* Adds security: Only public methods and attributes are accessible from the outside
* Protects against common mistakes: Only public fields & methods are accessible, so developers don’t accidentally change something dangerous
* Protects IP: Code is hidden in a class; only public methods are accessible by the outside developers
* Supportable: Most code undergoes updates and improvements
* Hides complexity: No one can see what’s behind the object’s curtain!

## Abstraction

Abstraction is an extension of encapsulation that uses classes and objects, which contain data and code, to hide the internal details of a program from its users. This is done by creating a layer of abstraction between the user and the more complex source code, which helps protect sensitive information stored within the source code.

* Reduces complexity and improves code readability
* Facilitates code reuse and organization
* Data hiding improves data security by hiding sensitive details from users
* Enhances productivity by abstracting away low-level details

Abstraction also serves an important security role. By only displaying selected pieces of data and only allowing data to be accessed through classes and modified through methods, we protect the data from exposure. To continue with the car example, you wouldn’t want an open gas tank while driving a car.

The benefits of abstraction are summarized below:

* Simple, high-level user interfaces
* Complex code is hidden
* Security
* Easier software maintenance
* Code updates rarely change the abstraction

## Polymorphism

Polymorphism means designing objects to share behaviors. Using inheritance, objects can override shared parent behaviors with specific child behaviors. Polymorphism allows the same method to execute different behaviors in two ways: method overriding and method overloading.

**Method Overriding**

Runtime polymorphism uses method overriding. In method overriding, a child class can implement differently than its parent class. In our dog example, we may want to give TrackingDog a specific type of bark different than the generic dog class.

**Method Overloading**

Compile Time polymorphism uses method overloading. Methods or functions may have the same name but a different number of parameters passed into the method call. Different results may occur depending on the number of parameters passed in.

The benefits of Polymorphism are:

* Objects of different types can be passed through the same interface
* Method overriding
* Method overloading

# 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 from this blueprint.

Classes contain fields for attributes and methods for behaviors.

## Objects

Objects are, unsurprisingly, a huge part of OOP! Objects are instances of a class created with specific data.

## 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 a 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.