**Introduction to Object-Oriented Programming**

Table of Contents

[Class 4](#_Toc49598129)

[Inheritance 5](#_Toc49598130)

[Polymorphism and Overloading 6](#_Toc49598131)

[Overriding 7](#_Toc49598132)

Procedural languages, such as any structured programming language, are ones that follow a set of instructions one by one. For very long lists of instructions, the instructions are divided into functions to make it easier to maintain. Ideally, each function deals with a specific task.

This format has several problems. Firstly, each function works with some data. The data that is stored inside the function, within its ‘scope’, is local and can only be accessed by that function. They are safe from manipulation by other functions. However, for most programs, there is at least some data that must be accessed by several functions. Thus, they must be declared globally. For programs with a large number of functions, the relationship between the different functions and global variables is difficult to keep track of. The unrestricted access by all functions in the program, even ones that do not need access to it can eventually lead to the data being edited by other functions due to some mistake, which can lead to unintended consequences.

Secondly, changes are difficult to make. Say 20 different functions deal with a global variable of int data type. If the variable is changed to a float data type, all the 20 functions must be edited to deal with this change. This is both cumbersome and prone to mistakes.

The procedural programming language does not model real life conditions. In real life, we have ‘objects’ that have characteristics and behaviours. For example, for an individual car, the car is the object, it has a characteristic or property that it has four wheels, and it has some behaviour, that it accelerates when a pedal is pressed. In programming, we can say that the characteristics are data about the object and the behaviours are functions of the object. The functions are thus related to the data. The car cannot accelerate if the number of wheels is 1 instead of 4. Structured Programming does not relate data and functions into one unit like this. The closest thing available is a structure, but it does not contain any functions. Thus, real life situations are difficult to implement in this way.

In Object-Oriented Programming, objects can interact. The data and functions related to a particular object are all part of that object and the only way other objects can interact with the data is through the functions. This provides security to the data since it cannot be edited by an outside function. This process is called encapsulation.

## Class

A class is like a blueprint for objects. All objects are part of some class. For example, a car object is part of the vehicle class. There are other types of objects that fall under the vehicle class. A specific object is said to be an instance of that class.

A class is simply a definition. It does not contain any data, but it contains what types of data is present in every instance of that class and all the functions related to it. For example, int wheel\_num will be present in a class and wheel\_num = 4 will be present in a specific object of that class.

## Inheritance

The way classes work gives us an opportunity to use inheritance. We can essentially create classes that are children of some class. The child class is called a subclass and the parent class is called a superclass. If the parent has no parent of its own it is also called a base class. The subclasses can essentially inherit all of the data types and functions present in the superclass and then add their own. For a class trucks under the base class vehicles, int wheel\_num could be an inherited data type since that is present in all vehicles but int carriage\_capacity would be added in that particular class since it is not a property of all vehicles.

## Polymorphism and Overloading

Polymorphism is the ability of something to behave differently in different environments. Here, it could be the same function working with different parameter types depending on which type is given to it. cout is one such function. It prints out whatever data type is passed to it. This is done with another capability of object-oriented programming called Overloading. For function overloading, it allows us to have more than one function of the same name but with different input parameters, so that they can work differently for different inputs.

Another part of overloading is Operator Overloading. Say we have two objects, position1 and origin that each contain pairs of integers that represent their locations. We have in a sense created a new data type. If we create a new object position2 and set it to position2 = position1 + origin, we would expect it to work like it does in mathematics, i.e. add the -coordinates and add the -coordinates. The + and = operators are not designed to work with pairs of numbers, but we can make them do so with operator overloading.

## Overriding

Overriding occurs when a subclass that inherits some function from the superclass declares the exact same function with different code. In this scenario, an object of the subclass will run the function associated with the subclass and not the one associated with the superclass. For example, if we have a function horn() declared in the vehicles class that plays a quiet noise and we have the same function declared again in the trucks class that plays a loud noise, an object created in the vehicles class will have a quiet horn and an object created in the trucks class will have a loud horn.