# Lesson:

# Introduction to OOPS

PW SKILLS

# What we'll learn:

- What is a programming paradigm?
- Types of programming paradigms.
- What is OOPS?
- What are OOPS concepts?
- Why is OOPS important?

In this lesson we will study OOPS concepts in theory, later we will see how JavaScript implements those concepts.

# What are programming paradigms?

- Programming paradigms encompass various approaches and styles by which a program or programming language can be structured.
- Each paradigm comprises specific structures, features, and perspectives on addressing typical programming challenges.
- Programming paradigms are distinct from languages or tools.

**Note:** Some programming languages follow more than one programming paradigm called multiple paradigm programming languages.

# Types of programming paradigms

## Imperative Programming:

Imperative programming involves providing the computer with a sequence of precise instructions to be executed in a specific order. The term imperative refers to the fact that programmers explicitly specify the precise actions the computer must perform.

**Example:** Filtering even values from array

```
nums = [1,4,3,6,7,8,9,2]
result = []

for i in nums:
    if (i % 2 == 0):
       result.append(i)
print(result)
```

See that we're telling the program to iterate through each element in the num array, and check if value % 2 gives 0 remainder, push it into the result array. So we are providing detailed and specific instructions, that is what imperative means.

C, C++, Java, Kotlin, PHP, Python, and Ruby are examples of imperative languages.

## Procedural Programming

- Procedural programming builds upon imperative programming by incorporating functions (referred to as "procedures" or "subroutines") as an additional feature.

We can extend our previous example with procedural programming as below,

```
def filterEven(nums):
   nums = [1,4,3,6,7,8,9,2]
   result = []

   for i in nums:
    if (i % 2 == 0):
       result.append(i)
   return result;

print(filterEven([1,4,3,6,7,8,9,2]));
```

We have made one procedure or function called filterEven for filtering even values from an array and returning it.

C, C++, Lisp, PHP, and Python are examples of procedural languages.

## Functional Programming

- Functional programming extends the concept of functions further. In functional programming, functions are regarded as first-class citizens, enabling them to be assigned to variables, passed as arguments, and returned from other functions.
- Languages like C++, Java, and JavaScript utilize Function programming.

**Example:** Higher order function in JS is a good example of function programming.

```
// returns mutiplier function
function getMutipler(x){
   return (y) => return x*y;
}

const double = getMutiplier(2);
const triple = getMutiplier(3);

double(10) // 20
double(40) // 80

triple(10) // 30
triple(40) // 120
```

## Declarative Programming

- Declarative programming reduces complexity furth. It focuses on specifying what needs to be done, not how to do it.
- Languages like HTML, SQL, CSS, etc. are declarative in nature.

**Example:** Display HTML heading

```
<h1>This is declarative language</h1>
```

In the above, we are commanding the browser to display a heading to the browser, but we have not specified how to display it.

## Object-Oriented Programming

Let us understand the Object Oriented concepts in general without the context of JavaScript. We will look into Object Oriented JavaScript in the upcoming chapters

- OOP stands for Object-Oriented Programming. Object Oriented is one of the programming paradigms, widely accepted among developers. We will study what OOPs are and how OOPs concepts are helpful in programming.
- OOP partitions concerns into entities represented as objects. Each entity encapsulates a specific collection of information (properties) and behaviors (methods) that can be performed by the entity.

## What is OOP?

- It is a programming paradigm that revolves around the concept of objects, which are instances of classes. OOP focuses on organizing code by representing real-world objects and their interactions.
- Let's understand, two main entities of OOPs,
    - Classes
    - Objects
- When we model a problem in terms of **objects** in OOP, we create abstract definitions representing the types of objects we want to have in our system.
- For example, if we were modelling an **organisation**, we might want to have objects representing employees. Every **employee** has some properties in common: they all have a **name** and a **designation**.
- Additionally, every **employee** can do certain things: they can all apply for leave and they can introduce themselves to their colleagues when they join the organisation.
- In pseudocode, an **Employee** class could be written like this:

```
class Employee
    properties
        name
        designation
    methods
        leave(date)
        introduceSelf()
```

- This defines an Employee class with:
    - two data properties: name and designation
    - two methods: leave(date) to apply leave and introduceSelf() to introduce themselves

- By itself, a class doesn't have any functionality; it serves as a blueprint for creating specific objects of that class type like in above example Employee class is just a blueprint.
- Each specific object we create from the class is referred to as an instance. The creation of an instance is carried out using a special function known as a constructor. We provide values to the constructor to initialise any internal state within the newly created instance.
- Typically, the constructor is included within the class definition and typically shares the same name as the class itself. Let's see the pseudocode of the class with constructor

```
class Employee
    properties
        name
        designation
    constructor
        Employee(name, designation)
    methods
        leave(date)
        introduceSelf()
```

## What are OOPs concepts?

In OOP, data and behaviour are encapsulated within objects. Objects are created from classes, which serve as blueprints or templates defining the properties (attributes or data) and behaviours (methods or functions) of objects.

The key principles of OOP are:

1. **Encapsulation:** It involves bundling data and related behaviours into objects and providing methods to interact with the object.
2. **Inheritance:** It allows classes to inherit properties and behaviours from other classes, forming a hierarchy of classes. Inherited features can be extended or overridden in derived classes.
3. **Polymorphism:** It enables objects of different classes to be treated as instances of a common superclass. Polymorphism allows methods to be defined in a general way and overridden in specific classes to provide different implementations.
4. **Abstraction:** It focuses on representing essential features of objects while hiding unnecessary details. It allows developers to create abstract classes or interfaces that define a common structure for related classes.

## Why is OOP important?

Object-oriented programming (OOP) is important for several reasons:

1. **Modularity and Reusability:** OOP promotes the modular design of software, where code is organised into self-contained objects. This modularity enhances reusability, as objects can be easily reused in different parts of an application or in different projects. This leads to more efficient and maintainable code development.
2. **Code Organization and Maintainability:** OOP provides a structured approach to code organisation. By breaking down a system into objects with well-defined responsibilities, code becomes more organised and easier to manage. OOP principles such as encapsulation and abstraction contribute to cleaner code and better maintenance over time.
3. **Collaboration and Teamwork:** OOP promotes modular and organised code, making it easier for developers to collaborate and work on different parts of a project simultaneously. The clear structure and well-defined interfaces of objects allow teams to work independently on different modules while ensuring compatibility and integration.
4. **Modeling Real-World Concepts:** OOP aligns with the real world by allowing developers to model objects, their relationships, and their behaviours. This makes it easier to understand, conceptualise, and solve complex problems by representing them in a familiar and intuitive way.