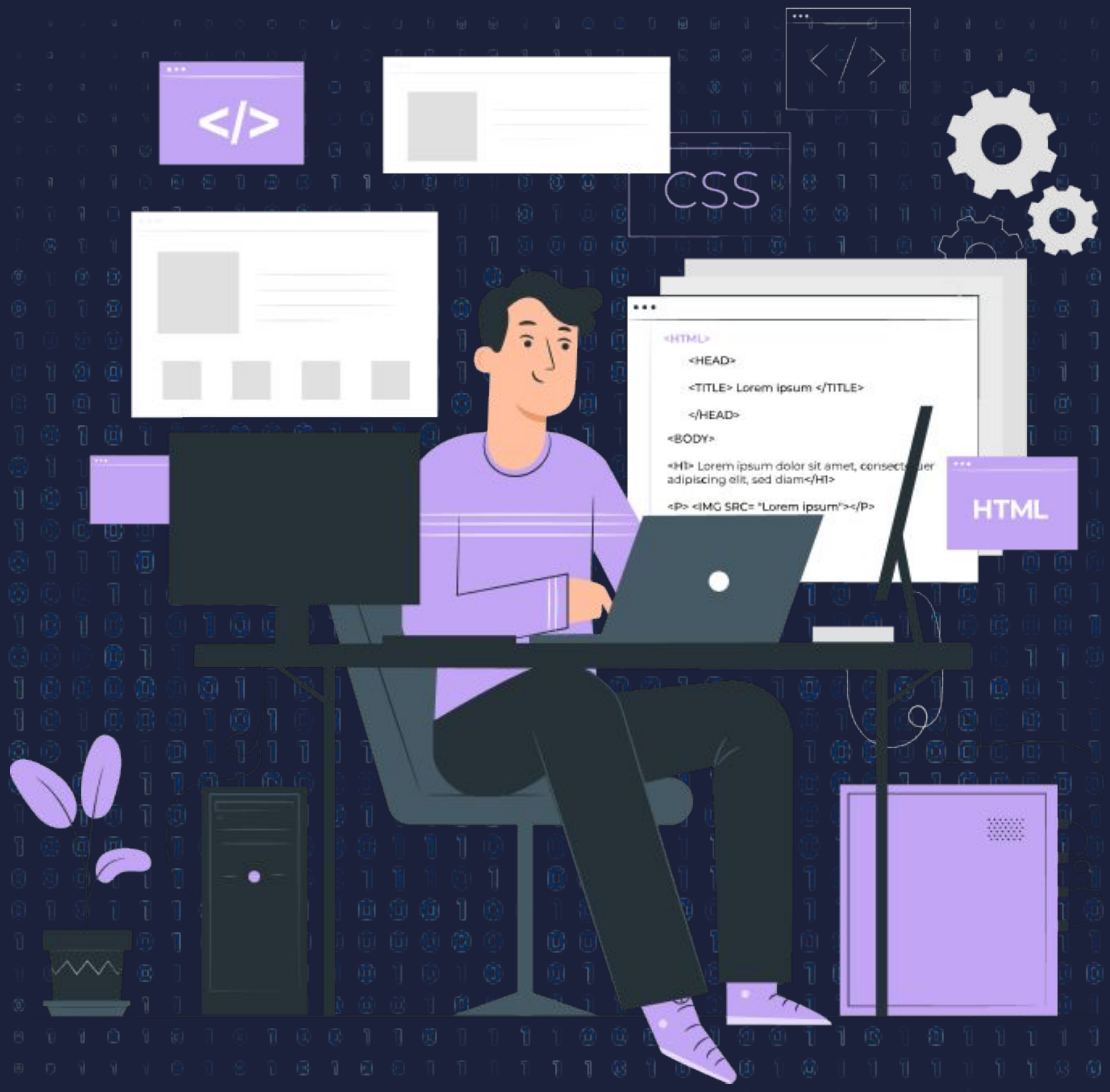
PW SKILLS

Introduction to OOPS



</>
CSS
<HTML>
<HEAD>
<TITLE> Lorem ipsum </TITLE>
</HEAD>
<BODY>
<H1> Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam</H1>
<P> <IMG SRC= "Lorem ipsum"></P>
HTML

Topics

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

What is a programming paradigm?

Programming paradigms encompass various approaches and styles by which a program or programming language can be structured.

Programming paradigms are distinct from languages or tools.

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

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.

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.

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

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)
```

Procedural Programming

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

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

Example : Filtering even values from array using function/procedure defined

```
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]));
```

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 utilise Function programming.

Example : Filtering even values from array using function/procedure defined

```
// 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>
```

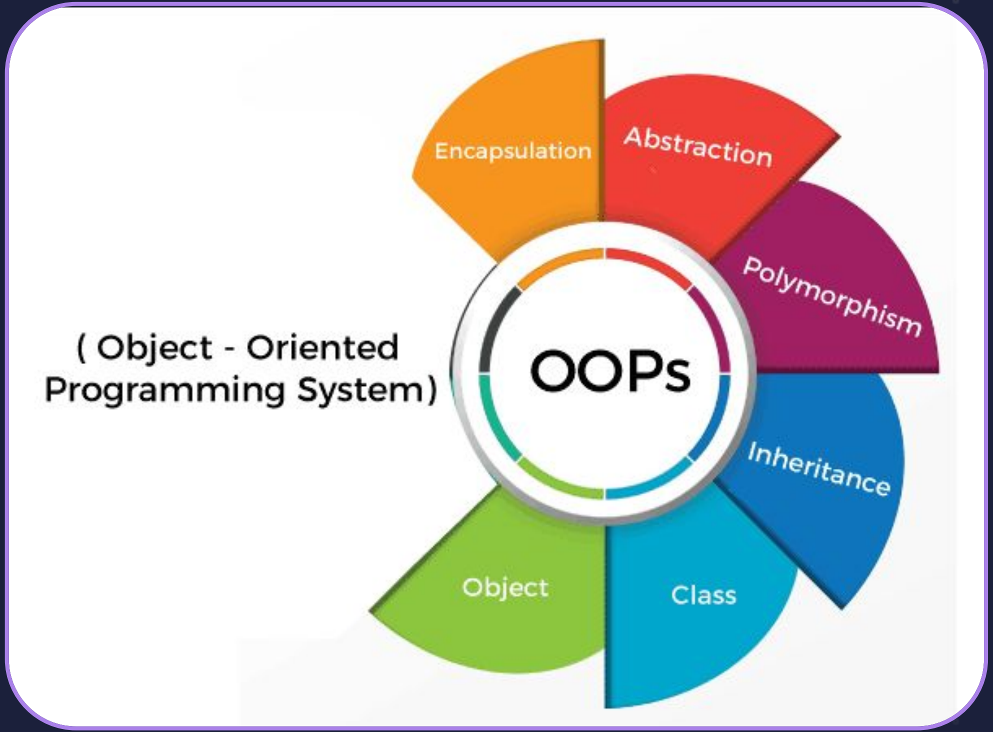
Object-Oriented Programming

OOP partitions concerns into entities represented as objects. Each entity encapsulates a specific collection of information (properties) and behaviours (methods) that can be performed by the entity.

Languages like C++, JavaScript

OOPs have two main entities

- Classes
- Objects



Example to Understand Class and Object

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.

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

This defines a **Employee** class with:

- two data properties: name and designation
- two methods: **leave**(date) to apply leave and **introduceSelf**() to introduce themselves

Employee class is just a blueprint.

Each specific object we create from the **class** is referred to as an **object**.

What are OOPs concepts?

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?

The key principles of OOP are:

1. **Modularity and Reusability:** OOP promotes the modular design of software, where code is organised into self-contained objects.
2. **Code Organization and Maintainability:** By breaking down a system into objects with well-defined responsibilities, code becomes more organised and easier to manage.
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.
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.

SKILLS

THANK YOU