**PW SKILLS**

**OOPS ASSIGNMENT**

1. \*\*Class and Object in Object-Oriented Programming (OOP)\*\*:

- \*\*Class\*\*: A class is a blueprint for creating objects (instances). It defines the properties (attributes) and behaviors (methods) that all objects of that class will have.

- \*\*Object\*\*: An object is an instance of a class. It is a concrete entity based on the class, with its own unique state (attributes) and behavior (methods).

- Example:

```python

class Car:

def \_\_init\_\_(self, make, model):

self.make = make

self.model = model

def drive(self):

return f"{self.make} {self.model} is driving."

# Creating objects (instances) of the Car class

car1 = Car("Toyota", "Camry")

car2 = Car("Honda", "Civic")

# Accessing object attributes and methods

print(car1.make) # Output: Toyota

print(car2.drive()) # Output: Honda Civic is driving.

```

2. \*\*Four Pillars of OOPs\*\*:

- \*\*Encapsulation\*\*: Encapsulation is the bundling of data (attributes) and methods that operate on that data into a single unit (class). It helps in hiding the internal state of an object and only exposing the necessary functionalities.

- \*\*Abstraction\*\*: Abstraction refers to the concept of hiding the complex implementation details and showing only the essential features of the object. It helps in reducing complexity and increasing efficiency.

- \*\*Inheritance\*\*: Inheritance is the mechanism where a new class (subclass) is created by inheriting properties and behaviors from an existing class (superclass). It promotes code reusability and establishes a hierarchical relationship between classes.

- \*\*Polymorphism\*\*: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables the same method to behave differently based on the object it is called upon, thus enhancing flexibility and extensibility.

3. \*\*Purpose of the \_\_init\_\_() Function\*\*:

- The `\_\_init\_\_()` function is used as a constructor method in Python classes. It is automatically called when a new object of the class is created.

- Its purpose is to initialize the object's state by setting initial values for its attributes.

- Example:

```python

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

person1 = Person("John", 30)

```

4. \*\*Purpose of `self` in OOP\*\*:

- In Python, `self` is a reference to the current instance of the class. It is used to access variables and methods of the current object within the class.

- It helps differentiate between instance variables and local variables within methods.

- Example:

```python

class Circle:

def \_\_init\_\_(self, radius):

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

circle1 = Circle(5)

print(circle1.area()) # Output: 78.5

```

5. \*\*Inheritance\*\*:

- Inheritance is a mechanism in OOP that allows a new class (subclass) to inherit properties and behaviors from an existing class (superclass).

- \*\*Types of Inheritance\*\*:

- \*\*Single Inheritance\*\*: When a class inherits from only one superclass.

```python

class Animal:

def speak(self):

pass

class Dog(Animal):

def bark(self):

print("Woof!")

# Dog class inherits from Animal class

```

- \*\*Multiple Inheritance\*\*: When a class inherits from more than one superclass.

```python

class A:

pass

class B:

pass

class C(A, B):

pass

# C class inherits from both A and B classes

```

- \*\*Multilevel Inheritance\*\*: When a class inherits from a superclass, and that superclass itself inherits from another superclass, forming a chain of inheritance.

```python

class A:

pass

class B(A):

pass

class C(B):

pass

# C class inherits from both B and A classes

```

- \*\*Hierarchical Inheritance\*\*: When multiple classes inherit from the same superclass.

```python

class Animal:

pass

class Dog(Animal):

pass

class Cat(Animal):

pass

# Both Dog and Cat classes inherit from Animal class

```