Solution for Task1 answers

*Qn1: Explain the concept of Object-Oriented Programming (OOP). What are the four main pillars of OOP, and how do they contribute to better software design?

Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of objects and classes.

The 4 main pillars of OOP;
1.Encapsulation.
Bundling data and methods into a single unit (class) or Encapsulation is the concept of bundling data and methods into a single unit (class). This helps,
- Hide implementation details
- Protect data from external interference
- Improve code organization

Example.

```
class BankAccount:
    def __init__(self, balance=0):
        self.__balance = balance

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance
```

2.Abstraction.
Abstraction is the concept of showing only necessary information while hiding implementation details.

Example.

```
class CoffeeMachine:
    def __init__(self):
        self.__water_temp = 0

    def make_coffee(self):
        # Complex process of making coffee
        pass
```

```
    def get_coffee(self):
        return "Coffee is ready!”
```

3.Inheritance.
Creating new classes from existing one.
This allows a new class to inherit the properties and methods of an existing class. It promotes code reusability and establishes a natural hierarchy between classes.

Example.

```
class Animal:
    def __init__(self, name):
        self.name = name

    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!”
```

4. Polymorphism.
Ability of objects to take multiple forms.
This allows objects of different classes to be treated as objects of a common superclass. It is particularly useful for implementing dynamic method dispatch, where the method to be invoked is determined at runtime.

Example.

```
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2
```

Note;
The 4 pillars contribute to better software design by:
- Promoting modularity and reusability

- Enhancing code organization and readability
- Facilitating easier maintenance and updates
- Supporting scalability and flexibility

Qn2: What is the purpose of a constructor in Python? Explain how the __init__ method is used to initialize an object’s attributes with an example.

Constructors in Python.

In Python, a constructor is a special method called `__init__` that initializes an object's attributes when it's created.

Example:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

person = Person("John", 30)
print(person.name)
 # Output: John
print(person.age)
# Output: 30
```

Qn3:
Difference between Class Variables and Instance Variables.

Class Variables.
- Shared among all instances of a class
- Defined inside the class definition
- Accessible through the class name or instance
  while
Instance Variables.
- Unique to each object
- Defined inside methods or `__init__`
- Accessible only through the instance

Class variables are variables that are shared among all instances of a class. They are defined within the class but outside any instance methods. Instance variables, on the other hand, are unique to each instance of a class and are typically defined within the __init__ method.

Example
Class variable for wheels 4

```
class Car:
    def __init__(self, color):
        self.color = color  # instance variable

car1 = Car("Red")
car2 = Car("Blue")

print(Car.wheels)
# Output: 4
print(car1.wheels)
# Output: 4
print(car1.color)
# Output: Red
print(car2.color)
 # Output: Blue
```

*Qn4:Define and provide an example of a class method and a static method. How do they differ from instance methods?
Class Methods and Static Methods

Class Methods.
- Bound to the class, not instances
- Use the `classmethod` decorator
- Receive the class as the first argument (cls)

Static Methods.
- Not bound to the class or instances
- Use the `staticmethod` decorator
- Don't receive any implicit arguments

Example.
class Math:

```
    classmethod
    def add(cls, x, y):
        return x + y

    staticmethod
    def multiply(x, y):
        return x * y

print(Math.add(2, 3))
 # Output: 5
print(Math.multiply(2, 3))
 # Output: 6
```

Qn5: Describe a real-world scenario where using class variables would be more appropriate than using instance variables. Explain why class variables are suitable in this scenario.

Real-World Scenario for Class Variables;
A real-world scenario where class variables are more appropriate is in a situation where you need to keep track of a value that is shared among all instances of a class. For example, consider a class that represents a library. You might want to keep track of the total number of books in the library, regardless of how many instances of the library class you create.

Example.

```
class User:
    def __init__(self, name):
        User.total_users += 1
        self.name = name

user1 = User("John")
user2 = User("Jane")

print(User.total_users)
```

Class variables are suitable because,
- The total user count is shared among all instances
- It's updated automatically when new users are created
- It's accessible through the class name or instances