Assignment 7

August 31, 2023

Q1. What is Abstraction in OOps? Explain with an example.

ANS:

Abstraction in Object-Oriented Programming (OOP) is a concept that involves simplifying complex reality by modeling classes based on the essential properties and behaviors of objects, while ignoring irrelevant details. It allows you to create classes that represent high-level concepts without specifying all the implementation details. Abstraction focuses on what an object does rather than how it does it.

In other words, abstraction helps you create a clear separation between the interface (what an object can do) and the implementation (how it does it). This separation makes it easier to manage complexity, promote code reusability, and adapt to changes in the future.

[1]:
```python
from abc import ABC, abstractmethod

class Vehicle(ABC):
    def __init__(self, make, model):
        self.make = make
        self.model = model

    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def stop(self):
        pass

class Car(Vehicle):
    def start(self):
        return f"{self.make} {self.model} engine started."

    def stop(self):
        return f"{self.make} {self.model} engine stopped."

class Motorcycle(Vehicle):
    def start(self):
        return f"{self.make} {self.model} engine started."
```

```
    def stop(self):
        return f"{self.make} {self.model} engine stopped."
```

[4]:
```
car=Car('BMW',"M40")
motorcycle=Motorcycle('YAMAHA',"R1M")
```

[6]:
```
car.make
```

[6]: 'BMW'

[7]:
```
motorcycle.model
```

[7]: 'R1M'

[10]:
```
car.start()
```

[10]: 'BMW M40 engine started.'

[11]:
```
motorcycle.stop()
```

[11]: 'YAMAHA R1M engine stopped.'

[ ]:
```

```

Q2. Differentiate between Abstraction and Encapsulation. Explain with an example.

ANS:

Abstraction and encapsulation are two important concepts in object-oriented programming (OOP), but they serve different purposes and have distinct roles in designing and structuring classes and objects.

Abstraction:

Abstraction involves simplifying complex reality by modeling classes based on essential properties and behaviors, while ignoring unnecessary details. It focuses on defining the interface or contract that a class should adhere to, without providing the full implementation. Abstraction hides the complex implementation details and allows you to work with high-level concepts. Encapsulation:

Encapsulation involves bundling data (attributes) and the methods (functions) that operate on that data into a single unit, called a class. It is about restricting direct access to an object's internal state and controlling how that state is accessed and modified. Encapsulation promotes data hiding and information hiding, preventing unintended modification of an object's state.

[12]:
```
class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.balance = balance
```

```
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount

    def withdraw(self, amount):
        if amount > 0 and self.balance >= amount:
            self.balance -= amount

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

[21]:
```
account = BankAccount("123456", 1000)
```

[22]:
```
account.deposit(1500)
```

[23]:
```
account.balance
```

[23]: 2500

[24]:
```
account.withdraw(500)
```

[25]:
```
account.balance
```

[25]: 2000

[26]:
```
print(f"Account Number: {account.account_number}")
print(f"Balance: {account.get_balance()}")
```

```
Account Number: 123456
Balance: 2000
```

[ ]:
```

```

Q3. What is abc module in python? Why is it used?

ANS:

The abc module in Python stands for "Abstract Base Classes." It provides a way to define abstract base classes and abstract methods, promoting abstraction and enforcing a certain structure for classes that inherit from these abstract classes. Abstract base classes are classes that cannot be instantiated directly but are meant to serve as blueprints for other classes.

The main purposes of the abc module are:

1.Creating Abstract Base Classes: You can use the abc module to create abstract base classes by subclassing the ABC class provided by the module. These abstract base classes can define abstract methods (methods without an implementation) that subclasses must override.

2.Enforcing Method Implementation: By defining abstract methods in abstract base classes, you can enforce that subclasses provide their own implementation for these methods. This helps ensure

that specific behavior is implemented consistently across different subclasses.

3.Promoting Code Structure: Abstract base classes provide a clear structure for your code. They allow you to define a common interface and share behavior among multiple classes that inherit from the same base class.

[ ]:

Q4. How can we achieve data abstraction?

ANS:

Data abstraction in object-oriented programming refers to the concept of hiding the complex implementation details of data and providing a simplified interface for interacting with that data. It involves defining classes in such a way that users of those classes don't need to be aware of the internal structure and implementation of the data. This can be achieved through a combination of encapsulation and providing well-defined methods for data access and manipulation.

Here's how you can achieve data abstraction in Python:

Encapsulation: Encapsulation involves bundling the data (attributes) and methods that operate on that data within a single class. By setting access modifiers (such as public, protected, and private), you control how the attributes can be accessed from outside the class. This helps in hiding the implementation details of the data.

Getter and Setter Methods: Instead of allowing direct access to attributes, you can provide getter methods to retrieve attribute values and setter methods to modify attribute values. This allows you to control how data is accessed and modified, enabling you to implement additional logic or validation.

Private Attributes: While Python doesn't enforce strict access control like some other languages, you can use naming conventions to indicate that certain attributes are meant to be treated as private. By convention, attributes starting with an underscore (e.g., _private_attribute) are considered private, and users are encouraged not to access them directly.

[27]:
```python
class BankAccount:
    def __init__(self, account_number, balance):
        self._account_number = account_number  # Private attribute
        self._balance = balance                # Private attribute

    def get_account_number(self):
        return self._account_number

    def get_balance(self):
        return self._balance

    def deposit(self, amount):
        if amount > 0:
            self._balance += amount

    def withdraw(self, amount):
```

```
        if 0 < amount <= self._balance:
            self._balance -= amount
```

[29]:
```
account = BankAccount("65465484", 10000)
```

[30]:
```
print("Account Number:", account.get_account_number())
print("Balance:", account.get_balance())
```

```
Account Number: 65465484
Balance: 10000
```

[31]:
```
account.deposit(5500)
account.withdraw(3400)
```

[32]:
```
print("Balance after transactions:", account.get_balance())
```

```
Balance after transactions: 12100
```

[ ]:
```

```

Q5. Can we create an instance of an abstract class? Explain your answer.

ANS:

No, you cannot create an instance of an abstract class in Python. An abstract class is a class that is meant to be a blueprint for other classes and cannot be instantiated directly. It serves as a template that defines a common structure and behavior for its subclasses, but it doesn't provide a complete implementation for certain methods. Abstract classes are created using the abc module's ABC class and the @abstractmethod decorator.

Attempting to create an instance of an abstract class will result in a TypeError because the abstract class lacks a complete implementation for one or more abstract methods.

[ ]:
```

```