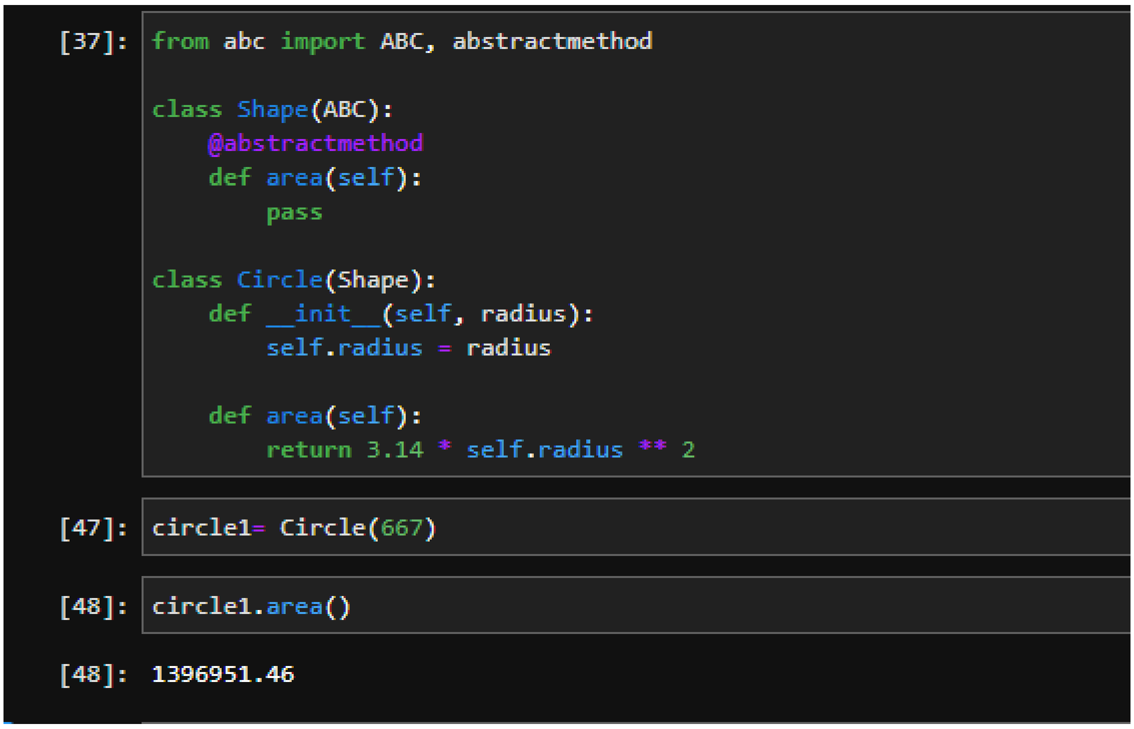
ASSIGNMENT-7

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

A:

Abstraction:

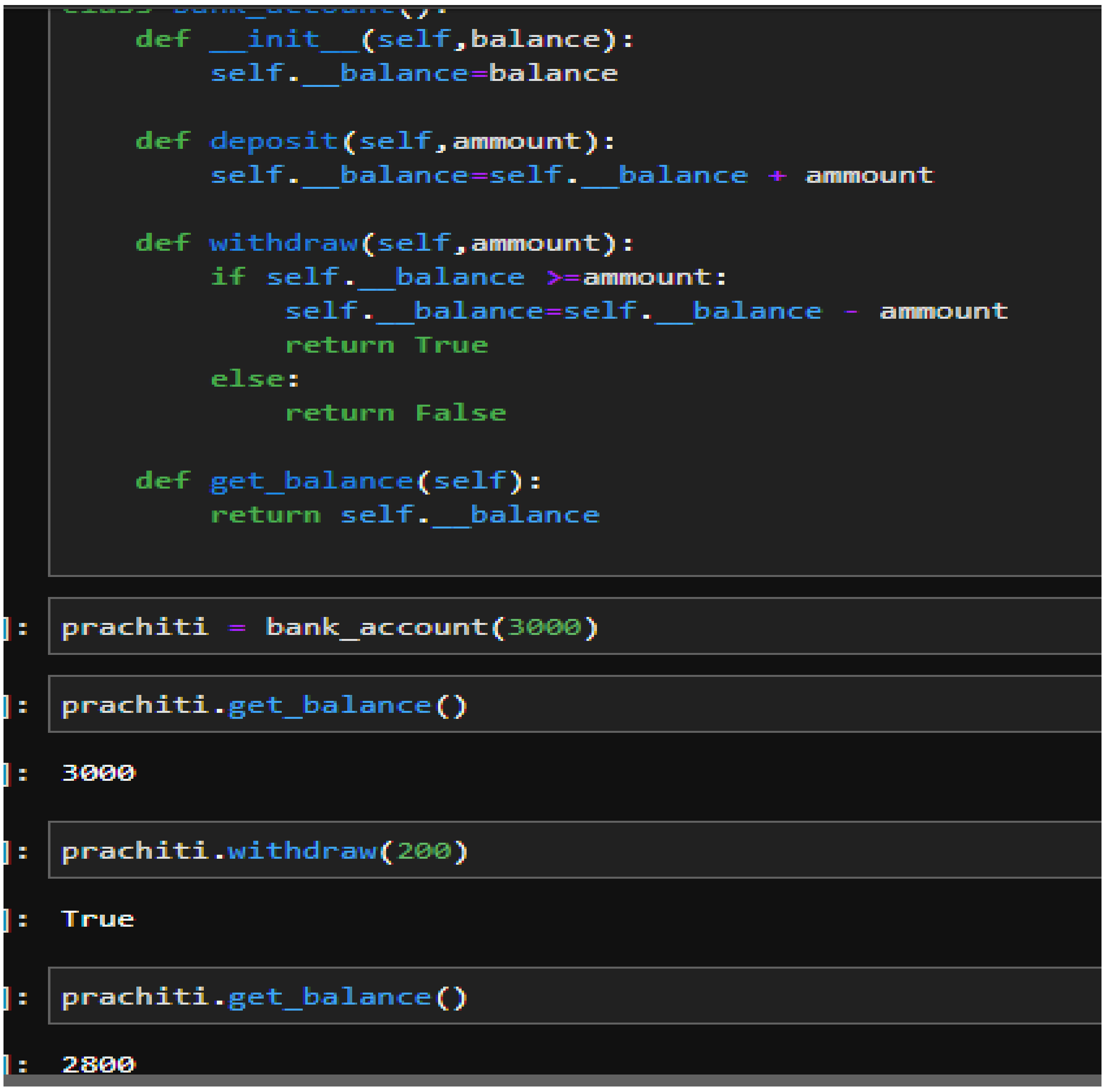
Abstraction allows you to simplify complex systems by modeling classes based on their essential characteristics and hiding unnecessary details.



```
[37]: from abc import ABC, abstractmethod

      class Shape(ABC):
          @abstractmethod
          def area(self):
              pass

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

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

[47]: circle1= Circle(667)

[48]: circle1.area()

[48]: 1396951.46
```

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

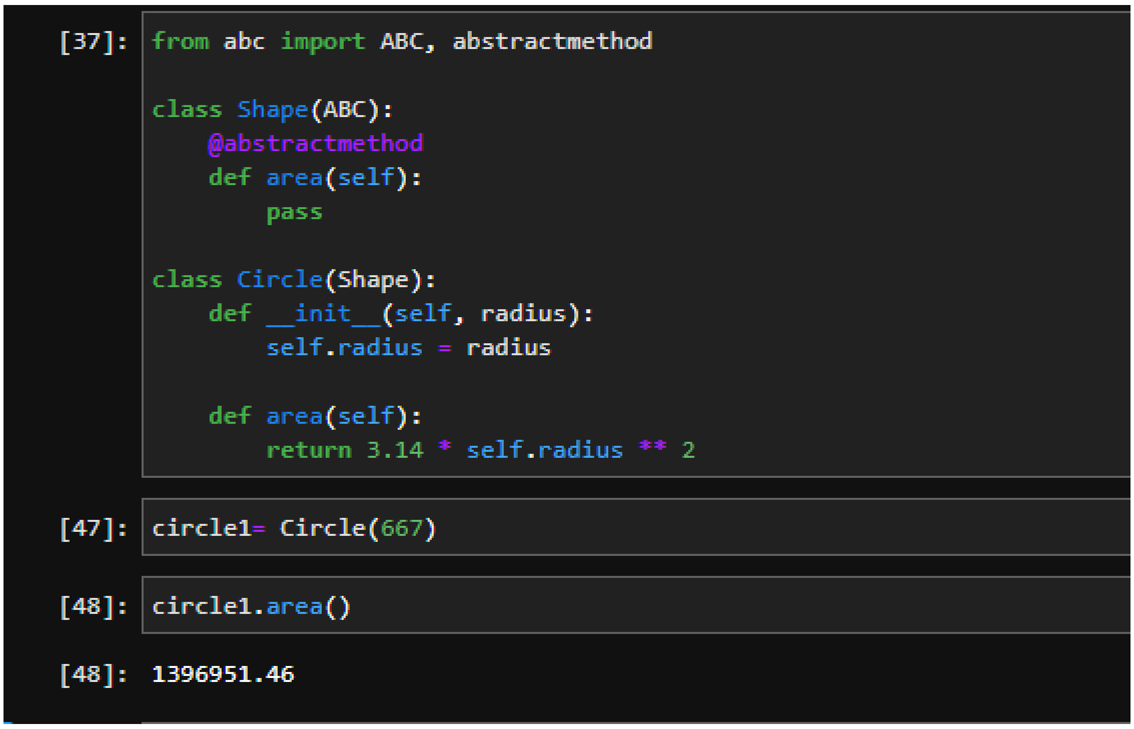
Encapsulation:
Encapsulation is the concept of bundling the data (attributes or variables) and the methods (functions) that operate on that data into a single unit called a class. It hides the internal details of how a class works from the outside and provides controlled access to the data through methods.



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

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

    def withdraw(self,ammount):
        if self.__balance >=ammount:
            self.__balance=self.__balance - ammount
            return True
        else:
            return False

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

```
]: prachiti = bank_account(3000)
]: prachiti.get_balance()
]: 3000
]: prachiti.withdraw(200)
]: True
]: prachiti.get_balance()
]: 2800
```

Abstraction:
Definition: Abstraction is the process of simplifying complex reality by modeling classes based on their essential characteristics while hiding unnecessary details. It focuses on what an object does (its interface) rather than how it does it (its implementation).



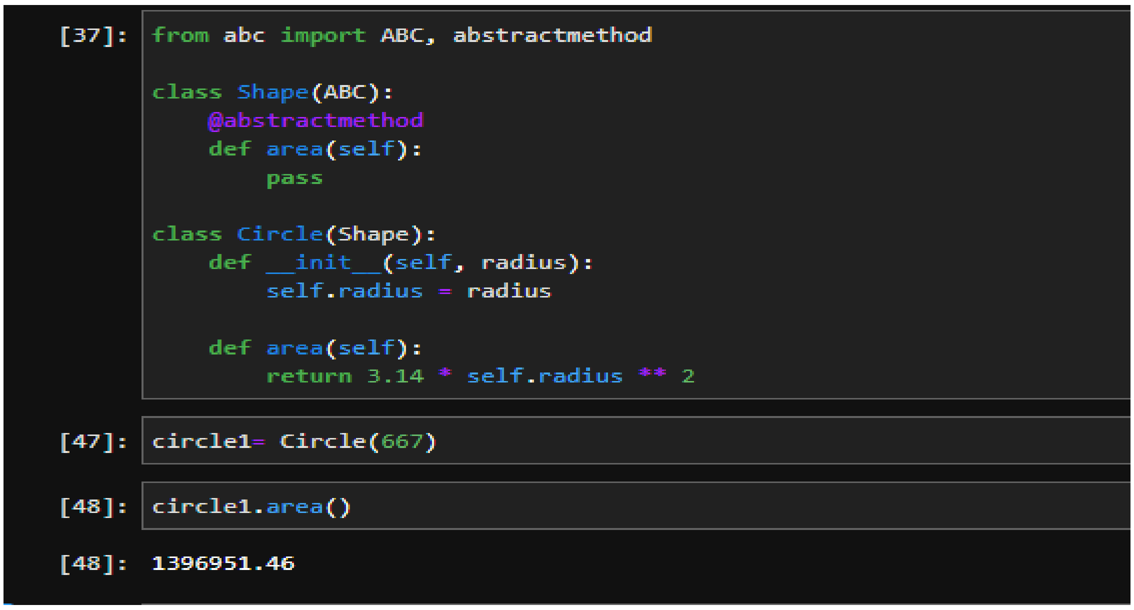
```
[37]: from abc import ABC, abstractmethod

      class Shape(ABC):
          @abstractmethod
          def area(self):
              pass

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

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

[47]: circle1= Circle(667)

[48]: circle1.area()

[48]: 1396951.46
```

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

A:

The abc module in Python stands for "Abstract Base Classes." It is a module that provides the infrastructure for defining and working with abstract base classes, which are classes that cannot be instantiated themselves but are intended to serve as a blueprint for other classes.

-



```
[37]: from abc import ABC, abstractmethod

      class Shape(ABC):
          @abstractmethod
          def area(self):
              pass

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

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

[47]: circle1= Circle(667)

[48]: circle1.area()

[48]: 1396951.46
```

Q4. How can we achieve data abstraction?

A:

Data abstraction is a fundamental concept in computer science and programming, especially in object-oriented programming, where it's often used to hide the complex implementation details of data structures while exposing a simplified and high-level interface for interacting with the data. You can achieve data abstraction through a few key techniques:

Classes and Objects: Use classes to encapsulate data and methods. Objects are instances of these classes that allow you to work with the data and methods.

Access Control: Use access control mechanisms, such as private and protected variables and methods, to restrict direct access to the internal data and operations of a class.

Getter and Setter Methods: Provide getter and setter methods to access and modify the class's internal data.

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

A:

No, you cannot create an instance of an abstract class in Python. Abstract classes are meant to be used as a blueprint for other classes, and they are incomplete on their own. Abstract classes are designed to define common methods and attributes that should be implemented by their concrete (i.e., non-abstract) subclasses.
In Python, you can create an abstract class using the abc module (Abstract Base Classes). The abc module provides the ABC (Abstract Base Class) metaclass and the @abstractmethod decorator to define abstract methods within a class.

my_instance = MyAbstractClass()