Training report day – 14

21 June 2024

Abstraction in Python:

Abstraction in Python is a fundamental concept in object-oriented programming (OOP) that focuses on hiding the complex implementation details of a class or function and exposing only the necessary parts to the user. This allows users to interact with objects at a higher level of abstraction without needing to understand the intricate details of their implementation.

Example

```
class Mobile:
    def __init__(self, brand, price):
        print("Inside constructor")
        self.brand = brand
        self.price = price
    def purchase(self):
        print("Purchasing a mobile")
        print("This mobile has brand", self.brand, "and price", self.price)
```

```
print("Mobile-1")
mob1=Mobile("Apple", 20000)
mob1.purchase()
print("Mobile-2")
mob2=Mobile("Samsung",3000)
mob2.purchase()
```

When we invoke the purchase() on a mobile object, we don't have to know the details of the method to invoke it.

This ability to use something without having to know the details of how it is working is called as abstraction

Encapsulation in Python:

Encapsulation is the concept of Object Oriented Programming. It avoids the accidental change in any variable. The value or content of the variable can be accessed by any method of the object.

Public data access:

Public data can be accessed by the other functions and can be changed by any outer method. This can change any critical value in the code.

Private data access:

We can put a lock on that data by adding a double underscore in front of it, as shown in below code.

Adding a double underscore makes the attribute a private attribute. Private attributes are those which are accessible only inside the class. This method of restricting access to our data is called encapsulation.

```
class Customer:
    def __init__(self, cust_id, name, age, wallet_balance):
        self.cust_id = cust_id
        self.name = name
        self.age = age
        self.__wallet_balance = wallet_balance

    def update_balance(self, amount):
        if amount < 1000 and amount > 0:
            self.__wallet_balance += amount

    def show_balance(self):
        print ("The balance is ",self.__wallet_balance)

c1=Customer(100, "Gopal", 24, 1000)
print(c1.wallet_balance())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-3-11571fc1d7fe> in <cell line: 16>()
     14
     15 c1=Customer(100, "Gopal", 24, 1000)
---> 16 print(c1.wallet_balance())

AttributeError: 'Customer' object has no attribute 'wallet_balance'
```

Note: Private variable can be accessed in other method of same class but cannot accessed by any method outside the class.

```
class Customer:
    def __init__(self, cust_id, name, age, wallet_balance):
        self.cust_id = cust_id
        self.name = name
        self.age = age
        self.__wallet_balance = wallet_balance

    def update_balance(self, amount):
        if amount < 1000 and amount > 0:
            self.__wallet_balance += amount

    def show_balance(self):
        print ("The balance is ",self.__wallet_balance)

c1=Customer(100, "Gopal", 24, 1000)
c1.__wallet_balance = 10000000000
c1.show_balance()
```

```
The balance is  1000
```