05 feb assignment

February 14, 2023

Q1. Explain Class and Object with respect to Object-Oriented Programming. Give a suitable example.

ANS -

Classes are user-defined data types that act as the blueprint for individual objects, attributes for individual objects, attributes and methods. Objects are instances of a class created with specifically defined data. Objects can correspond to real world objects or an abstract entity.

0.0.1 EXAMPLE

[2]:
```
class pwskills:

    def welcome_msg(self):
        print("welcome to pwskills  data science master course")
```

[6]:
```
shadab = pwskills()
```

[7]:
```
shadab.welcome_msg()
```

```
welcome to data science master course
```

[ ]:
```

```

Q2. Name the four pillars of OOPs.

ANS-

The four pillars of Object-Oriented Programming(OOP). The four pillars are Inheritance ,Polymorphism , Encapsulation and Abstraction.

Inheritance - In python it is possible to create an object that inherits the methods and properties of another object. this is called inheritance. in inheritance, there is a parent class and a child class. a child class inherits the properties and methods of the parent class.

Polymorphism - Polymorphism simply means 'many forms'. in python this means that you can have one function or object that can be used in diffeent ways.

Encapsulation - Encapsulation is the process of making data private by wrapping data and its methods in a 'capsule' or unit, so that it can not be accessed or modified outside of that unit.This is achieved by making variable inside a class private.

Abstarction - The fourth pillar of OOP is abstraction. abstraction is about keeping the process simple by hiding unnecessary details from the user

[ ]:

Q3. Explain why the **init**() function is used. Give a suitable example.

ANS-

The **ini** function is called every time an object is created from a class. The **init** method lets the class initialize the object's attributes and serves no other purpose. It is only used within class.

0.0.2 EXAMPLES

[12]:
```
class pwskills1:

    def __init__(self , phone_number , city , student_id , email_id):

        self.phone_number = phone_number
        self.city = city
        self.student_id = student_id
        self.email_id = email_id

        def return_student_details(self):
            return self.phone_number , self.city , self.student_id , self.
 ↪email_id
```

[13]:
```
shadab = pwskills1(8879437062 , "mumbai" , 765 , "shadab@gmail,com" )
```

[ ]:
```
shadab.phone_number
```

[15]:
```
shadab.city
```

[15]: 'mumbai'

[16]:
```
shadab.student_id
```

[16]: 765

[17]:
```
shadab.email_id
```

[17]: 'shadab@gmail,com'

[ ]:

Q4. Why self is used in OOPs?

ANS-

The self-variable is used to represent the instance of the class which is often used in object-oriented programming. It works as a reference to the object, Python uses the self parameter to refer to instance attributes nad methods os the class.

[ ]:

Q5. What is inheritance? Give an example for each type of inheritance.

ANS -

Inheritance allows us to define a class that inherits all the methods and properties from another class. Parent class is the class being inherited from, also called base class. Child class is the class that inheertis from another class , also called derived class.

0.1 EXAMPLES

[ ]:
```
1 . Single Inheritance
```

[19]:
```
class parent:

    def test_parent(self):
        print("this is my parent class")
```

[20]:
```
class child(parent):
    pass
```

[21]:
```
child_obj = child()
```

[22]:
```
child_obj.test_parent()
```

```
this is my parent class
```

[ ]:
```
2 . Multilevel Inheritance
```

[23]:
```
class class1:

    def test_class1(self):
        print("this is my class1 ")
```

[27]:
```
class class2(class1):

    def test_class2(self):
        print("this is my class2 ")
```

[28]:
```
class class3(class2):

    def test_class3(self):
        print("this is my class3 ")
```

[29]:
```
obj_class3 = class3()
```

[30]:
```
obj_class3.test_class1()
```

```
this is my class1
```

[31]:
```
obj_class3.test_class2()
```

```
this is my class2
```

[32]:
```
obj_class3.test_class3()
```

```
this is my class3
```

[ ]:
```
3. Multiple Inheritance
```

[33]:
```
class class1:
    def test_class1(self):
        print("this is my class 1")
```

[39]:
```
class class2:
    def test_class2(self):
        print("this is my class 2")
```

[41]:
```
class class3 (class1 , class2):
    pass
```

[42]:
```
obj_class3 = class3()
```

[43]:
```
obj_class3.test_class1()
```

```
this is my class 1
```

[44]:
```
obj_class3.test_class2()
```

```
this is my class 2
```

[ ]:
```

```