Ass 5 (oops)

August 16, 2023

[ ]:
```
[1.]  a class in object-oriented programming is like a blueprint or template␣
 ↪that defines the structure and behavior
     of objects. It specifies the attributes (data) and methods (functions)␣
 ↪that objects of that class will have.
    It acts as a blueprint for creating objects.

   On the other hand, an object is an instance of a class. It is a specific␣
 ↪entity created based on the class
   blueprint. An object has its own unique values for the attributes defined␣
 ↪in the class and can perform the
   methods defined in the class.
```

[1]:
```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def start_engine(self):
        print(f"The {self.brand} {self.model} engine is starting.")

my_car = Car("Toyota", "Camry")

print(my_car.brand)
print(my_car.model)

my_car.start_engine()
```

```
Toyota
Camry
The Toyota Camry engine is starting.
```

[ ]:
```
[2.]  The four pillars of Object-Oriented Programming (OOP) are:

(a.) Abstraction: This refers to the process of hiding complex implementation␣
 ↪details while showing only the necessary information to the user.
    It helps in reducing complexity and increasing efficiency
```

```
(b.) Encapsulation: This refers to the process of binding data and functions␣
 ↪that operate on that data, into a single unit. It helps in keeping the
    data safe from outside interference and misuse

(c.) Inheritance: This refers to the process of creating new classes from␣
 ↪existing ones. The new class inherits the properties and methods of the
    existing class and can add its own unique features. It helps in reducing␣
 ↪code duplication and increasing reusability

(d.) Polymorphism: This refers to the ability of objects to take on multiple␣
 ↪forms. It allows objects of different classes to be treated as if
    they are of the same class, and enables the use of a single interface to␣
 ↪represent a group of related objects
```

[ ]:
```
[3.] The __init__() function is used to initialize the attributes of a class␣
 ↪object when it is created. It is called automatically every time
    a new object is created from the class. The __init__() function can be␣
 ↪used to set the initial values of the object's attributes, or to
    perform any other necessary initialization.

   For example, the following code defines a class called Person with an␣
 ↪__init__() function that sets the name and age attributes
    of the object:
```

[1]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

```
John
36
```

[ ]:
```
[4.] The self keyword is used in object-oriented programming (OOP) to refer to␣
 ↪the current instance of the class. It is used to access the
    attributes and methods of the class, and to bind the arguments passed to a␣
 ↪method to the current instance.

    In Python, methods are defined with a first parameter called self. This␣
 ↪parameter is used to refer to the current instance of
    the class when  the method is called.
```

[ ]:
```
[5.] Inheritance is a mechanism in object-oriented programming where a class
 ↪(known as the child or derived class) can inherit properties
    and behaviors from another class (known as the parent or base class). This
 ↪allows the child class to reuse code and extend the
   functionality of the parent class.

   There are different types of inheritance:
       1) Single Inheritance
       2) Multiple Inheritance
       3) Multilevel Inheritance
       4) Hierarchical Inheritance
```

[2]:
```
"""1) Single Inheritance"""

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

    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

dog = Dog("Buddy")
dog.speak()
```

```
Dog barks
```

[3]:
```
"""2) Multilevel Inheritance"""

class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

class Bulldog(Dog):
    def speak(self):
        print("Bulldog growls")

bulldog = Bulldog()
bulldog.speak()
```

```
Bulldog growls
```

[4]:
```python
"""3.) Multiple Inheritance"""

class Animal:
    def speak(self):
        print("Animal speaks")

class Pet:
    def play(self):
        print("Playing with pet")

class Dog(Animal, Pet):
    pass

dog = Dog()
dog.speak()
dog.play()
```

```
Animal speaks
Playing with pet
```

[ ]:
```python

```