oop-with-python

March 10, 2024

0.1 Object Oriented Programming

- Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects," which can contain data (attributes) and code (methods).
- In Python, we can create classes to define objects and their behavior.

1 Create class and objects in Python

1.0.1 Create Class

[ ]:
```python
# Creating a Class
class Car:
    # Constructor method (__init__)
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    # Method to display car information
    def display_info(self):
        print(f"Car: {self.year} {self.make} {self.model}")
```

1.0.2 Creating Objects (Instances)

- Use the class name followed by parentheses to create an object (instance) of the class.
- Pass any required arguments to the constructor (**init** method) if defined.

[ ]:
```python
# Creating objects of class Car
car1 = Car("Toyota", "Camry", 2020)
car2 = Car("Honda", "Accord", 2018)
```

1.0.3 Accessing Attributes and Calling Methods

Use dot notation (.) to access attributes and call methods of an object.

[ ]:
```python
# Accessing attributes
print(car1.make)   # Output: Toyota
print(car2.model)  # Output: Accord
```

```
# Calling methods
car1.display_info()  # Output: Car: 2020 Toyota Camry
car2.display_info()  # Output: Car: 2018 Honda Accord
```

```
Toyota
Accord
Car: 2020 Toyota Camry
Car: 2018 Honda Accord
```

1.0.4 Constructor (init) Method

- The **init** method is a special method used to initialize objects.
- It is called automatically when an object of the class is created.

[ ]:
```
class Car:
    # Constructor method (__init__)
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
```

1.0.5 Instance and Class Attributes

- Instance attributes are specific to each object (instance) and are defined inside the **init** method using self.
- Class attributes are shared among all instances of a class and are defined directly within the class body.

[ ]:
```
class Car:
    # Class attribute
    car_count = 0

    # make,model and year are the parameters that needs to
    # passed while creating its object
    def __init__(self, make, model, year):
        # Instance attributes
        self.make = make
        self.model = model
        self.year = year

car = Car('Toyota','v8','2020')
```

1.1 Encapsulation

[ ]:
```
# encapsulation
class Person :
    age = 50; # public
    _address = 'Kathmandu' #protected
```

```
    __bank_account = '4324lk-324kl' #private
    def __init__(self): # constructor
      pass

    def __getAccountNumber(self): #private
        return self.__bank_account

    def _getAddress(self): #protected
        return self._address

    def getAge(self): # pubic
        return self.age

person = Person()
```

1.2 Inheritance

[ ]:
```
# Inheritance
class Student(Person):
    def __init__(self):
        super().__init__()

    def depositFee(self):
        print (self._getAddress())

    def getAge(self): # polymorphism
        return self.age

student = Student()
student.depositFee()
```

```
Kathmandu
```

1.3 Abstraction

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

class BankAccount(ABC):
    def __init__(self,account_number,balance):
        self.account_number = account_number
        self.balance = balance


    @abstractmethod
    def deposit(self,amount):
        pass
```

```
    @abstractmethod
    def withdraw(self,amount):
        pass



class SavingAccount(BankAccount):
    def __init__(self, account_number, balance):
        super().__init__(account_number, balance)

    def deposit(self, amount):
        return super().deposit(amount)

    def withdraw(self, amount):
        if(amount>self.balance):
            print('Insufficient balance')
            return None
        print('Amount Deposited')
```