Python Programming - 2101CS405

Lab - 12

Name :Vyas Bhagyesh Y.

Enrollment No : 23010101662

Roll N0 : 23010101662

OOP

01) Write a Program to create a class by name Students, and initialize attributes like name, age, and grade while creating an object.

In [4]:

```
class Students:
    def __init__(self,name,age,grade):
        self.name=name
        self.age=age
        self.grade=grade
s1=Students('Bhagyesh',19,'A++')
print(s1.name,s1.age,s1.grade,sep="\n")
```

```
Bhagyesh
19
A++
```

02) Create a class named Bank_Account with Account_No, User_Name, Email,Account_Type and Account_Balance data members. Also create a method GetAccountDetails() and DisplayAccountDetails(). Create main method to demonstrate the Bank_Account class.

In [5]:

```
class Bank_Acoount:
    def GetAccountDetails(self):
        self.Account_No=int(input('Enter Account Number: '))
        self.User_Name=input('Enter User Name: ')
        self.Email=input('Enter Email: ')
        self.Account_Type=input('Enter Account Type: ')
        self.Account_Balance=int(input('Enter Account Balance'))
    def DisplayAccountDetails(self):
        print('Account No is : ', self.Account_No)
        print('User Name is : ',self.User_Name)
        print('Email ID is: ',self.Email)
        print('Account Type is: ',self.Account_Type)
        print('Account Balance is: ',self.Account_Balance)

b1 = Bank_Acoount()
b1.GetAccountDetails()
b1.DisplayAccountDetails()
```

```
---------------------------------------------
Account No is :  23010101662
User Name is :  Bhagyesh vyas
Email ID is:  23010101662@darshan.ac.in
Account Type is:  Saving
Account Balance is:  435627
---------------------------------------------
```

03) WAP to create Circle class with area and perimeter function to find area and perimeter of circle.

In [8]:

```
class Circle:
    def area(self,r):
        self.r=r
        print("Area = ",3.14*self.r*self.r)
    def perimeter(self,r):
        self.r=r
        print("Perimeter= ",3.14*self.r*self.r)
c=Circle()
c.area(5)
c.perimeter(5)
```

```
Area =  78.5
Perimeter=  78.5
```

04) Create a class for employees that includes attributes such as name, age, salary, and methods to update and display employee information.

In [10]:

```
class Employee:
    def add(self):
        self.name=input('Enter Name: ')
        self.age=int(input('Enter Age: '))
        self.salary=int(input('Enter Salary: '))

    def display(self):
        print('--------------------------')
        print('Name is : ',self.name)
        print('Age is : ',self.age)
        print('Salary is : ',self.salary)
        print('--------------------------')

    def updatename(self):
        self.name=input('Enter New Name: ')

e1 = Employee()
e1.add()
e1.display()
e1.updatename()
e1.display()
```

```
-------------------------
Name is :  Bhagyesh
Age is :  19
Salary is :  12345
-------------------------

-------------------------
Name is :  Bhagyesh Vyas
Age is :  19
Salary is :  12345
-------------------------
```

05) Create a bank account class with methods to deposit, withdraw, and check balance.

In [21]:

```python
class BankAccount:
    def __init__(self,bal):
        self.bal=bal
    def deposit(self,amt):
        self.amt=amt
        self.bal+=self.amt
        print("Balance after deposit = ",self.bal)
    def withdraw(self,amt):
        self.amt=amt
        self.bal-=self.amt
        print("Balance after Withdraw = ",self.bal)

ba=BankAccount(5000)
ba.deposit(100)
ba.withdraw(100)
```

```
Balance after deposit =  5100
Balance after Withdraw =  5000
```

06) Create a class for managing inventory that includes attributes such as item name, price, quantity, and methods to add, remove, and update items.

In [1]:

```python
class Inventory:
    def __init__(self):
        self.inventory = {}
    def add_item(self, item_id, item_name, stock_count, price):
        self.inventory[item_id] = {"item_name": item_name, "stock_count": stock_count, "

    def update_item(self, item_id, stock_count, price):
        if item_id in self.inventory:
            self.inventory[item_id]["stock_count"] = stock_count
            self.inventory[item_id]["price"] = price
        else:
            print("Item not found in inventory.")

    def check_item_details(self, item_id):
        if item_id in self.inventory:
            item = self.inventory[item_id]
            return f"Product Name: {item['item_name']}, Stock Count: {item['stock_count'
        else:
            return "Item not found in inventory."

inventory = Inventory()

inventory.add_item("I001", "Laptop", 100, 500.00)
inventory.add_item("I002", "Mobile", 110, 450.00)
inventory.add_item("I003", "Desktop", 120, 500.00)
inventory.add_item("I004", "Tablet", 90, 550.00)
```

```
print("Item Details:")
print(inventory.check_item_details("I001"))
print(inventory.check_item_details("I002"))
print(inventory.check_item_details("I003"))
print(inventory.check_item_details("I004"))
print("\nUpdate the price of item code - 'I001':")
inventory.update_item("I001", 100, 505.00)
print(inventory.check_item_details("I001"))
print("\nUpdate the stock of item code - 'I003':")
inventory.update_item("I003", 115, 500.00)
print(inventory.check_item_details("I003"))
```

```
Item Details:
Product Name: Laptop, Stock Count: 100, Price: 500.0
Product Name: Mobile, Stock Count: 110, Price: 450.0
Product Name: Desktop, Stock Count: 120, Price: 500.0
Product Name: Tablet, Stock Count: 90, Price: 550.0

Update the price of item code - 'I001':
Product Name: Laptop, Stock Count: 100, Price: 505.0

Update the stock of item code - 'I003':
Product Name: Desktop, Stock Count: 115, Price: 500.0
```

09) Create a Class with instance attributes

In [23]:

```
class Data:
    def __init__(self):
        self.name = input("Enter Name : ")
        self.salary = int(input("Enter Salary : "))

    def display(self):
        print(self.name,self.salary)

d = Data()
d.display()
```

```
Bhagyesh 23345
```

07)

Create one class student_kit

Within the student_kit class create one class attribute principal name (Mr ABC)

Create one attendance method and take input as number of days.

While creating student take input their name .

Create one certificate for each student by taking input of number of days present in class.

In [25]:

```
class Student_Kit:
    Principal_Name="Mr. ABC"

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

    def attendance(self,days):
        self.days=days
        return self.days

s1 = Student_Kit('Bhagyesh')
print(Student_Kit.Principal_Name ,'has issued a certificate to',s1.name,',who was presen
```

```
Mr. ABC has issued a certificate to Bhagyesh ,who was present 11 days
```

08) Define Time class with hour and minute as data member. Also define addition method to add two time objects.

In [38]:

```
class Time:
    def __init__(self,h,m):
        self.h=h
        self.m=m
    def add(self,t1,t2):
        self.h=t1.h+t2.h
        self.m=t1.m+t2.m
    def printTime(self):
        if self.m>60:
            self.h+=1
            self.m-=60
        print("Time = ",self.h," : ",self.m)

t1=Time(5,58)
t2=Time(4,3)
t2.add(t1,t2)
t2.printTime()
```

```
Time =  10  :  1
```

09) WAP to demonstrate inheritance in python.

In [39]:

```
class A:
    def displayA(self):
        print('This is class A')

class B(A):
    def displayB(self):
        print('This is class B')

class C(B):
    def displayC(self):
        print('This is class C')

obj1 = C()
obj1.displayA()
obj1.displayB()
obj1.displayC()
```

```
This is class A
This is class B
This is class C
```

10) Create a child class Bus that will inherit all of the variables and methods of the Vehicle class

class Vehicle:

```
def __init__(self, name, max_speed, mileage):
    self.name = name
    self.max_speed = max_speed
    self.mileage = mileage
```

Create a Bus object that will inherit all of the variables and methods of the parent Vehicle class and display it.

In [40]:

```
class Vehicle:
    def __init__(self, name, max_speed, mileage):
        self.name = name
        self.max_speed = max_speed
        self.mileage = mileage

class Bus(Vehicle):
    def __init__(self,name, max_speed, mileage):
        super().__init__(name, max_speed, mileage)

    def display(self):
        print("Name:", my_bus.name)
        print("Max Speed:", my_bus.max_speed)
        print("Mileage:", my_bus.mileage)

my_bus = Bus("My Bus", 60, 10000)
my_bus.display()
```

```
Name: My Bus
Max Speed: 60
Mileage: 10000
```

11) Create a class hierarchy for different types of animals, with a parent Animal class and child classes for specific animals like Cat, Dog, and Bird.

In [2]:

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

class Cat(Animal):
    def make_sound(self):
        return "Meow"

class Dog(Animal):
    def make_sound(self):
        return "Woof"

class Bird(Animal):
    def make_sound(self):
        return "Chirp"

my_cat = Cat("Fluffy", 3)
my_dog = Dog("Buddy", 5)
my_bird = Bird("Tweety", 2)

print(my_cat.name, "says", my_cat.make_sound())
print(my_dog.name, "says", my_dog.make_sound())
print(my_bird.name, "says", my_bird.make_sound())
```

```
Fluffy says Meow
Buddy says Woof
Tweety says Chirp
```

12) Create a class hierarchy for different types of vehicles, with a parent Vehicle class and child classes for specific vehicles like Car, Truck, and Motorcycle.

In [41]:

```
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
```

```python
    def start(self):
        print("Starting the vehicle.")

    def stop(self):
        print("Stopping the vehicle.")

    def accelerate(self):
        print("Accelerating the vehicle.")

    def brake(self):
        print("Applying the brakes.")


class Car(Vehicle):
    def __init__(self, make, model, year, num_doors):
        super().__init__(make, model, year)
        self.num_doors = num_doors

    def park(self):
        print("Parking the car.")

    def honk(self):
        print("Honking the car horn.")

    def display(self):
        print("Make - ",self.make)
        print("Model - ",self.model)
        print("Year - ",self.year)
        print("Year - ",self.num_doors)

class Truck(Vehicle):
    def __init__(self, make, model, year, payload_capacity):
        super().__init__(make, model, year)
        self.payload_capacity = payload_capacity

    def load_cargo(self):
        print("Loading cargo into the truck.")

    def unload_cargo(self):
        print("Unloading cargo from the truck.")

    def display(self):
        print("Make - ",self.make)
        print("Model - ",self.model)
        print("Year - ",self.year)
        print("Year - ",self.payload_capacity)


class Motorcycle(Vehicle):
    def __init__(self, make, model, year, num_wheels):
        super().__init__(make, model, year)
        self.num_wheels = num_wheels

    def wheelie(self):
        print("Popping a wheelie on the motorcycle.")

    def lean(self):
        print("Leaning into the turn while riding the motorcycle.")

    def display(self):
        print("Make - ",self.make)
        print("Model - ",self.model)
        print("Year - ",self.year)
        print("Year - ",self.num_wheels)
```

```python
t = Truck("Volvo","S20",2020,"23 tonnes")
t.start()
t.accelerate()
t.brake()
t.stop()
t.display()
```

```
Starting the vehicle.
Accelerating the vehicle.
Applying the brakes.
Stopping the vehicle.
Make -  Volvo
Model -  S20
Year -  2020
Year -  23 tonnes
```

13) Create a class hierarchy for different types of bank accounts, with a parent Account class and child classes for specific account types like Checking, Savings, and Credit.

In [3]:

```python
class Account:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited ${amount}. New balance is ${self.balance}.")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds.")
        else:
            self.balance -= amount
            print(f"Withdrew ${amount}. New balance is ${self.balance}.")


class Checking(Account):
    def __init__(self, account_number, balance, overdraft_limit):
        super().__init__(account_number, balance)
        self.overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if amount > self.balance + self.overdraft_limit:
            print("Insufficient funds.")
        else:
            self.balance -= amount
            print(f"Withdrew ${amount}. New balance is ${self.balance}.")


class Savings(Account):
    def __init__(self, account_number, balance, interest_rate):
        super().__init__(account_number, balance)
        self.interest_rate = interest_rate

    def accrue_interest(self):
        interest = self.balance * self.interest_rate
        self.balance += interest
        print(f"Accrued ${interest} in interest. New balance is ${self.balance}.")


class Credit(Account):
    def __init__(self, account_number, balance, credit_limit):
        super().__init__(account_number, balance)
        self.credit_limit = credit_limit

    def make_purchase(self, amount):
        if amount > self.balance + self.credit_limit:
            print("Purchase declined.")
        else:
            self.balance -= amount
            print(f"Made purchase for ${amount}. New balance is ${self.balance}.")

    def make_payment(self, amount):
        self.balance += amount
        print(f"Made payment of ${amount}. New balance is ${self.balance}.")

abc = Checking("122",500,200)
abc.withdraw(900)
```

```
Insufficient funds.
```

14) Create a Shape class with a draw method that is not implemented. Create three child classes Rectangle, Circle, and Triangle that implement the draw method with their respective drawing behaviors. Create a list of Shape objects that includes one instance of each child class, and then iterate through the list and call the draw method on each object.

In [42]:

```
class Shap:
    def draw(self):
        pass

class Rectangle(Shap):
    def draw(self):
        print("Drawing a rectangle")


class Circle(Shap):
    def draw(self):
        print("Drawing a circle")


class Triangle(Shap):
    def draw(self):
        print("Drawing a triangle")


# Create a list of Shape objects
shapes = [Rectangle(), Circle(), Triangle()]

# Iterate through the list and call the draw method on each object
for s in shapes:
    s.draw()
```

```
Drawing a rectangle
Drawing a circle
Drawing a triangle
```

15) Create a Person class with a constructor that takes two arguments name and age. Create a child class Employee that inherits from Person and adds a new attribute salary. Override the **init** method in Employee to call the parent class's **init** method using the super keyword, and then initialize the salary attribute.

In [47]:

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
class Employee(Person):
    def __init__(self,name,age,salary):
        super().__init__(name,age)
        self.salary=salary
    def printData(self):
        print("Name :: ",self.name)
        print("Age :: ",self.age)
        print("Salary :: ",self.salary)

emp=Employee('Bhagyesh',19,43562)
emp.printData()
```

```
Name ::  Bhagyesh
Age ::  19
Salary ::  43562
```