Darshan UNIVERSITY

योग: कर्मसु कौशलम्

Python Programming - 2101CS405

Lab - 12

In [ ]:
```
Name :- Vora Yagnik
Enrollment No :- 23010101661
```

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 Student:
    def __init__(self,name,age,grade):
        self.name=name
        self.age=age
        self.grade=grade

s1 = Student('Yagnik',19,'A++')
print("Student Name :",s1.name)
print("Student Age :",s1.age)
print("Student Grade :",s1.grade)
```

```
Student Name : Yagnik
Student Age : 19
Student Grade : 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 [8]:
```
class Bank_Acoount:
    def GetAccountDetails(self):
```

```
        self.A_No=int(input('Enter Account Number: '))
        self.User_Name=input('Enter User Name: ')
        self.Email=input('Enter Email: ')
        self.A_Type=input('Enter Account Type: ')
        self.A_Balance=int(input('Enter Account Balance'))

    def DisplayAccountDetails(self):
        print('-----------------------------------------')
        print('Account No is : ', self.A_No)
        print('User Name is : ',self.User_Name)
        print('Email ID is: ',self.Email)
        print('Account Type is: ',self.A_Type)
        print('Account Balance is: ',self.A_Balance)
        print('-----------------------------------------')

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

```
-----------------------------------------
Account No is :  23010101661
User Name is :  Vora Yagnik
Email ID is:  xyz123@gmail.com
Account Type is:  shaving
Account Balance is:  120000
-----------------------------------------
```

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

In [10]:
```
class Circle:
    def __init__(self,r):
        self.r = r
    def area(self):
        print('Area of circle is: ',3.14*self.r*self.r)

    def perimeter(self):
        print('Perimeter of cicle is: ',2*3.14*self.r)

c1 = Circle(7)
c1.area()
c1.perimeter()
```

```
Area of circle is:  153.86
Perimeter of cicle is:  43.96
```

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

In [15]:
```
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 :  Yagnik
Age is :  19
Salary is :  60000
-------------------------
-------------------------
Name is :  Yagnik Vora
Age is :  19
Salary is :  60000
-------------------------
```

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

In [18]:
```
class BankAccount:
    def __init__(self):
        self.balance = 0

    def deposit(self,a):
        self.balance=self.balance + a
        print("Amount",self.balance,"is deposit")

    def withdraw(self,a):
        self.balance=self.balance - a
        print("Amount",self.balance,"is withdraw")

    def checkBalance(self):
        print("Your Account Balance is - ",self.balance)

b = BankAccount()
b.deposit(1000)
b.deposit(2000)
b.withdraw(2000)
b.withdraw(500)
b.checkBalance()
```

```
Amount 1000 is deposit
Amount 3000 is deposit
Amount 1000 is withdraw
Amount 500 is withdraw
Your Account Balance is -  500
```

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 [24]:

```
Item Details:
Product Name: Laptop ,Stock Count: 100 ,Price: 500.0
None
Product Name: Mobile ,Stock Count: 110 ,Price: 450.0
None
Product Name: Desktop ,Stock Count: 120 ,Price: 500.0
None
Product Name: Tablet ,Stock Count: 90 ,Price: 550.0
None

Update the price of item code - 'I001':
Item updated...
Product Name: Laptop ,Stock Count: 100 ,Price: 505.0
None

Update the stock of item code - 'I003':
Item updated...
Product Name: Desktop ,Stock Count: 115 ,Price: 500.0
None
```

09) Create a Class with instance attributes

In [25]:

```
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_cou

    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_c
        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
```

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 [26]:
```
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('Madhav')
print(Student_Kit.Principal_Name ,'has issued a certificate to',s1.name,',who was p
```

```
Mr. ABC has issued a certificate to Madhav ,who was present 15 days
```

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

In [29]:
```
class Time:
    def __init__(self,h,m):
        self.hour=h
        self.minute=m

    def add(self,a,b):
        self.hour=a.hour + b.hour
        self.minute=a.minute + b.minute

    def display(self):
        print(self.hour,self.minute)

obj1 = Time(11,35)
obj2 = Time(19,20)
obj3 = Time(0,0)
obj3.add(obj1,obj2)
obj1.display()
obj2.display()
obj3.display()
```

```
11 35
19 20
30 55
```

09) WAP to demonstrate inheritance in python.

In [30]:
```
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 [38]:
```
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:", self.name)
        print("Max Speed:", self.max_speed)
        print("Mileage:", self.mileage)

my_bus = Bus("My Bus", 70, 15)
my_bus.display()
```

```
Name: My Bus
Max Speed: 70
Mileage: 15
```

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 [41]:
```
class Animal:
    def __init__(self, name):
        self.name = name

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("Cat")
my_dog = Dog("Dog")
my_bird = Bird("Bird")

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())
```

```
Cat says Meow
Dog says Woof
Bird 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 [42]:

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

    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)

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 [43]:
```
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 [44]:
```python
class Shap:
    def draw(self):
        raise NotImplementedError("draw method not implemented")


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")

shapes = [Rectangle(), Circle(), Triangle()]

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 [46]:
```python
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 display(self):
        print("Name - ",self.name)
        print("Age - ",self.age)
        print("Salary - ",self.salary)

e = Employee("abc",27,50000)
e.display()
```

```
Name -  abc
Age -  27
Salary -  50000
```