**CLASS PROGRAMS**

DATE:28.11.2024

single inheritance

class Employee: #parent class

def getEmployeeInfo(self):

self.id=input("Enter the ID")

self.name=input("Enter the name:")

def displayEmployeeInfo(self):

print("ID=",self.id,"\n Name=",self.name)

class Perks(Employee): #child class

def getDetails(self):

self.getEmployeeInfo()

self.sal=int(input("Enter the Salary"))

def displayInfo(self):

self.displayEmployeeInfo()

print("salary",self.sal)

p=Perks()

p.getDetails()

p.displayInfo()

#Test 1

class Inventory:

def \_\_init\_\_(self,prodid,prodname,prodcount):

self.prodid=prodid

self.prodname=prodname

self.prodcount=prodcount

def display(self):

print(self.prodid, self.prodname, self.prodcount)

i=Employee(123,"Lipstick",3)

i.display()

#test 2

class Inventory:

def \_\_init\_\_(self,prodid,prodname,prodcount):

self.prodid=prodid

self.prodname=prodname

self.prodcount=prodcount

class Employee(Inventory):

def display(self):

print(self.prodid, self.prodname, self.prodcount)

i=Employee(123,"Lipstick",3)

i.display()

DATE:29.11.2024

#Mulitple Inheritance (2 parent class,1 child class)

class Employee:

def \_\_init\_\_(self,name,Id,position):

self.name=name

self.Id=Id

self.position=position

def displayEmployeeInfo(self):

print(f"Name:{self.name}\nId:{self.Id}\nPosition:{self.position}")

class Address:

def \_\_init\_\_(self,street,state,country):

self.street=street

self.state=state

self.country=country

def displayaddressInfo(self):

print(f"Street Name:{self.street}\nState Name={self.state}\nCountry Name:{self.country}")

class EmployeeDetails(Employee,Address):

def \_\_init\_\_(self,name,Id,position,street,state,country):

super().\_\_init\_\_(name,Id,position) #it access the first parent class don't have self

Address.\_\_init\_\_(self,street,state,country) #we use self comlusory while calling with use of parent class name

def displayEmp(self):

self.displayEmployeeInfo()

self.displayaddressInfo()

ed=EmployeeDetails("Rickshi",100,"Manager","ABC Street","TamilNadu","India")

ed.displayEmp()

DATE:02.12.2024

'''MULTILEVEL INHERITEANCE-->a class inherits from a child class or derived class'''

class Person:

def \_\_init\_\_(self,name,age):

self.name=name

self.age=age

def displaypersoninfo(self):

print(f"Person Name:{self.name}\nPerson age:{self.age}")

class Employee(Person):

def \_\_init\_\_(self,name,age,Id):

Person.\_\_init\_\_(self,name,age)

self.Id=Id

def displayemployeeInfo(self):

print(f"Id:{self.Id}")

class Manager(Employee):

def \_\_init\_\_(self,name,age,Id,salary):

super().\_\_init\_\_(name,age,Id)

self.salary=salary

def dispalymanagerInfo(self):

print(F"salary:{self.salary}")

man=Manager("Rickshi",18,2537,500000)

man.displaypersoninfo()

man.displayemployeeInfo()

man.dispalymanagerInfo()

'''HIERARCHICAL INHERITANCE--> More than one child class is derived from a single parent /we can say one parent class and muliplt child class'''

class Teacher: #parent class-->teacher

def \_\_init\_\_(self,Incharge\_name):

self.Incharge\_name=Incharge\_name

def displayteacher(self):

print(f"Incharge Name:{self.Incharge\_name}")

class Student1(Teacher):

def \_\_init\_\_(self,Incharge\_name,name):

super().\_\_init\_\_(Incharge\_name) #it access teacher

self.name=name

def displaystudent1(self):

self.displayteacher()

print(f"Student Name:{self.name}")

class Student2(Teacher):

def \_\_init\_\_(self,Incharge\_name,stu\_name):

Teacher.\_\_init\_\_(self,Incharge\_name) #it access teacher

self.stu\_name=stu\_name

def displayStudent2(self):

print(f"student name 2:{self.stu\_name}")

stu=Student1("Geetha","Rickshi") #object for 1st child class

stu2=Student2("Geetha","Rickshitha") #obeject for 2nd child cass

stu.displaystudent1() #Display for child 1

stu2.displayStudent2() #display for child 2

DATE:09.12.2024

'''HYBRID INHERITANCE-->COMBINATION OF DIFFERENT INHERITANCE'''

class person:

def \_\_init\_\_(self,name,age):

self.name=name

self.age=age

def person\_info(self):

print(f"Name of the person: {self.name}\nAge of the person: {self.age}")

class student(person):

def \_\_init\_\_(self,stu\_id):

self.stu\_id=stu\_id

def student\_info(self):

print(f"Student Id: {self.stu\_id}")

class Employee(person):

def \_\_init\_\_(self,emp\_id):

self.emp\_id=emp\_id

def Employee\_info(self):

print(f"Employee Id: {self.emp\_id}")

class display\_details(student,Employee,person):

def \_\_init\_\_(self,name,age,stu\_id,emp\_id):

person.\_\_init\_\_(self,name,age)

student.\_\_init\_\_(self,stu\_id)

Employee.\_\_init\_\_(self,emp\_id)

def display\_all(self):

self.person\_info()

self.student\_info()

self.Employee\_info()

m=display\_details("Anika",18,"E24AI003",207645)

m.display\_all()

'''Test'''

class Management:

def getmarks(self):

self.name=input("Enter the student Name:")

self.mark1=int(input("Enter the Mark1:"))

self.mark2=int(input("Enter the Mark2:"))

self.mark3=int(input("Enter the Mark3:"))

def show(self):

print(self.name,self.mark1,self.mark2,self.mark3)

def percen(self):

total=self.mark1+self.mark2+self.mark3

self.percentage=(total/300)\*100

def display(self):

print(self.percentage)

class Details(Management):

def detailsinfo(self):

self.getmarks()

self.percen()

def markdetails(self):

self.show()

self.display()

stu=Details()

stu.detailsinfo()

stu.markdetails()

DATE:10.12.2024

'''HYBRID INHERITANCE'''

class Employee:

def \_init\_(self,name,age):

self.name=name

self.age=age

def displayEmployeeInfo(self):

print("Name of the employee:",self.name)

print("Age of the employee:",self.age)

class Manager(Employee):

def \_init\_(self,name,age,eid):

Employee.\_init\_(self,name,age)

self.eid=eid

def displayManagerInfo(self):

print("ID:",self.eid)

class Developer(Employee):

def \_init\_(Self,name,age,dept):

Employee.\_init\_(self,name,age)

self.dept=dept

def displayDeveloperInfo(self):

print("Department:",self.dept)

class TeamLeader(Manager,Developer):

def \_init\_(self,name,age,eid,dept,teamsize):

Employee.\_init\_(self,name,age)

self.eid=eid

self.dept=dept

self.teamsize=teamsize

def displayTeamInfo(self):

print(f"Team size={self.teamsize}")

Name=input()#we can use different cases here

Age=int(input())

Eid=input()

Dept=input()

ts=input()

tl=TeamLeader(Name,Age,Eid,Dept,ts)

tl.displayEmployeeInfo()

tl.displayManagerInfo()

tl.displayDeveloperInfo()

tl.displayTeamInfo()

'''input inside the constructor''''

class Apartment:

def \_init\_(self):

self.floor=int(input("Enter the floor where your house in the apartment"))

self.block=input("Enter the block where your house is located")

def displayapartment(self):

print("Floor:",self.floor)

print("Blok:",self.block)

class Rent:

def \_init\_(self):

self.rent=int(input("Enter the rent of the apratment"))

def displayrent(self):

print("Rent:",self.rent)

class Information(Apartment,Rent):

def \_init\_(self):

Apartment.\_init\_(self)

Rent.\_init\_(self)

self.room=input("Enter the rooms in the house")

def displayinfo(self):

print(f"The rooms available in the house is {self.room}")

self.displayapartment()

self.displayrent()

a=Information()

a.displayinfo()

'''input outside the consructor'''

class Apartment:

def \_init\_(self,floor,block):

self.floor=floor

self.block=block

def displayapartment(self):

print("Floor:",self.floor)

print("Blok:",self.block)

class Rent:

def \_init\_(self,rent):

self.rent=rent

def displayrent(self):

print("Rent:",self.rent)

class Information(Apartment,Rent):

def \_init\_(self,floor,block,rent,room):

Apartment.\_init\_(self,floor,block)

Rent.\_init\_(self,rent)

self.room=room

def displayinfo(self):

print(f"The rooms available in the house is {self.room}")

self.displayapartment()

self.displayrent()

floor=int(input("Enter the floor where your house in the apartment"))

block=input("Enter the block where your house is located")

rent=int(input("Enter the rent of the apratment"))

room=input("Enter the rooms in the house")

a=Information(floor,block,rent,room)

a.displayinfo()

DATE:11.12.2024

'''Encapsulation'''

class Student:

ID=123 #class variable #public

def \_\_init\_\_(self,name):

self.\_\_name=name #private

def display(self):

print("Name=",self.\_\_name)

s=Student("Rickshi")

s.display()

print(s.id)

class Student:

Admission\_no=int(input("Enter the Admission Number:"))

def \_\_init\_\_(self):

self.\_\_name=input("Enter a Name:")

self.\_\_ID=int(input("Enter A ID:"))

def display(self):

print(self.\_\_name,self.\_\_ID)

s=Student()

s.display()

print(s.Admission\_no)

#name Handling

class Employee:

def \_\_init\_\_(self,name,salary):

self.name=name #public class

self.\_\_salary=salary #private class

emp=Employee('Rickshi',1000000)

print('Name:',emp.name)

print('Salary:',emp.\_Employee\_\_salary) #we aslo access using object name.singleunderscore class name followed by double underscore thst salary

#user Input

class School:

def \_\_init\_\_(self):

self.name=input("Enter the School Name:")

self.\_\_address=input("Enter the Address:")

emp=School()

print('School Name:',emp.name)

print('Address:',emp.\_School\_\_address)

#inherit

class Employee:

def \_\_init\_\_(self):

self.name=input("Enter the Name:")

self.\_\_salary=int(input("Enter the Salary:"))

class Staff(Employee):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.Staff\_Name=input("Enter the Staff Name:")

emp=Staff()

print('Name:',emp.name)

print('Salary:',emp.\_Employee\_\_salary) #if it is private means must use the class name

print('Staff Name:',emp.Staff\_Name)

Date:16.12.2024

class Student:

def \_\_init\_\_(self,name,age):

self.name=name

self.age=age

def get\_name(self):

return self.name

def get\_age(self):

return self.age

def set\_name(self,name):

self.name=name

def set\_age(Self,age):

self.age=age

name=input("Enter thr Name:")

age=int(input("Enter the Age:"))

s=Student(name,age)

print(f"Name:{s.get\_name()}\nAge:{s.get\_age()}")

class Student:

def \_\_init\_\_(self,name,age):

self.name=name

self.age=age

def get\_name(self):

return self.name

def get\_age(self):

return self.age

def set\_name(self,name):

self.name=name

def set\_age(Self,age):

self.age=age

name=input("Enter thr Name:")

age=int(input("Enter the Age:"))

s=Student(name,age)

n=input()

s.set\_name(n)

print(f"Name:{s.get\_name()}\nAge:{s.get\_age()}")

Date:17.12.2024

class Teacher:

def \_\_init\_\_(self,name,age):

self.name=name

self.age=age

def show(self):

print("Details:",self.name,self.age)

def max\_speed(self):

print("Teaching Speed is 100X")

def change\_mood(self):

print("Teacher is in happy Mood")

class Student(Teacher):

def max\_speed(self):

print("Student Learning speed is 10X")

def change\_mood(self):

print("Student in Happy Mood")

stu=Student("Rickshi",18)

stu.show()

stu.max\_speed()

stu.change\_mood()

tec=Teacher("Nazeera",23)

tec.show()

tec.max\_speed()

tec.change\_mood()

class Library:

def issue\_book(self,book\_name,user):

return f"Book issued:{book\_name} to {user}"

def returned\_book(self,book\_name,user):

return f"Book Returned:{book\_name} by {user}"

class DigitalLibrary:

def issue\_book(self,book\_name,user):

return f"Book issued:{book\_name} to {user}"

def returned\_book(self,book\_name,user):

return f"Book Returned:{book\_name} by {user}"

lib=Library()

dig=DigitalLibrary()

book=input()

username=input()

returnbook=input()

retusername=input()

print(lib.issue\_book(book,username))

print(lib.returned\_book(returnbook,retusername))

print(dig.issue\_book(book,username))

print(dig.returned\_book(returnbook,retusername))

Date:18.12.2024

#Magic Method

class Book:

def \_\_init\_\_(self,pages):

self.pages=pages

#Overloading + operator with magic method

def \_\_add\_\_(self,other):

return self.pages+other.pages

b1=Book(400)

b2=Book(300)

print("Total Number of Pages:",b1+b2)

'''Method Overloading'''

class Addition:

def add(self,a,b,c=0):

result=0

if a!=None and b!=None and c==None:

result=a+b

return result

elif a!=None and b!=None and c!=None:

result=a+b+c

return result

ad=Addition()

ans=ad.add(2,3)

print(ans)

ans1=ad.add(1,2,3)

print(ans1)

Date:19.12.2024

import re

word="simple reggular expression example"

result=re.findall("gula",word) #findall in the sentence

print(result)

space=re.search('\s',word) #secrh-->it's going to search\s--> check the whitespace

print("\n The first space is at:",space.start()) #it count the index value of space

DATE:13.01.2025

'''NUMBERS FIRST ZERO LAST'''

n=int(input()) #4

arr=[]

for i in range(n):

L=int(input(f"Enter the element {i+1}:")) # 4 0 2 0

arr.append(L)

print(arr) #[4,0,2,0]

for i in arr:

if i==0: #i value is 0

arr.remove(0) #it removes the current position

arr.append(0) #its append to the end

print("After Array:",arr) #[4,2,0,0]

'''ZERO FIRST ZERO LAST'''

for i in arr[:]: #[:] slice notation that create a copy of a list

if i!=0:

arr.remove(i) #its remove non zero

arr.append(i) #its append to end

print("After Array:",arr)

Best Time to Buy and Sell Stock II

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day.

Find and return the maximum profit you can achieve.

Example 1:

Input: prices = [7,1,5,3,6,4]

Output: 7

Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Total profit is 4 + 3 = 7.

Example 2:

Input: prices = [1,2,3,4,5]

Output: 4

Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

Total profit is 4.

Example 3:

Input: prices = [7,6,4,3,1]

Output: 0

Explanation: There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

Constraints:

1 <= prices.length <= 3 \* 104

0 <= prices[i] <= 104

n=int(input("Enter the number of elements in the array:"))

arr=[]

p=0

for i in range(n):

e=int(input("Enter the elements"))

arr.append(e)

length=len(arr)

for i in range(1,length):

if arr[i]>arr[i-1]:

p+=arr[i]-arr[i-1]

print(p)

DATE:17.01.2025

'''write a recursive function to reverse a string'''

def reverse(s):

if len(s)<=1:

return s

else:

return reverse(s[1:])+s[0]

user\_input=input("Enter the String to reverse:")

reverse=reverse(user\_input)

print("Reversed string:",reverse)

'''palindrome'''

s=input("enter the string to palindrome:")

p\_s=s[::-1]

if p\_s==s:

print("Plaindrome")

else:

print("NOt palindrome")

'''write a recurvise function to find the sum of digits of a number'''

def sum\_of\_digit(n):

if n==0:

return 0

else:

return n%10 +sum\_of\_digit(n//10)

num=int(input("Enter the number:"))

print("sum of digit",sum\_of\_digit(num))

'''2D array sum of all elements'''

n=int(input("Enter the number of elements in the array"))

arr=[]

tot=0

for i in range(n):

e=list(map(int,input().split()))

arr.append(e)

for i in range(n):

for j in range(n):

print(arr[i][j],end=" ")

print()

for i in range(n):

for j in range(n):

tot+=arr[i][j]

print(tot)

'''2D array transpose of a matrix'''

n=int(input("Enter the number of elements in the array:"))

arr=[]

for i in range(n):

e=list(map(int,input().split()))

arr.append(e)

print("Original matrix")

for i in range(n):

for j in range(n):

print(arr[i][j],end=" ")

print()

print("Transpose")

for i in range(n):

for j in range(n):

print(arr[j][i],end=" ")

print()

'''recursive function to print x^n'''

x=int(input("Enter the base"))

n=int(input("Enter the power"))

def power(x,n):

if n==0:

return 1

else:

return x\*power(x, n-1)

res=power(x,n)

print(res)

DATE:20.01.2025

STAR PATTERN

import turtle

star=turtle.Turtle()

star.right(75)

star.forward(100)

for i in range(4):

star.right(144)

star.forward(100)

turtle.done()

SQUARE PATTERN

import turtle

s=turtle.Turtle()

s.forward(100)

s.left(90)

s.forward(100)

s.left(90)

s.forward(100)

s.left(90)

s.forward(100)

s.left(90)

s.hideturtle() #it hides the arrow

using loop

import turtle

s=turtle.Turtle()

for i in range(4):

s.forward(100) #moves the turtle forward by the specified amount

s.left(90)

turtle.done()

'''using penup and pendown'''

import turtle

s=turtle.Turtle()

s.forward(100)

s.penup() #picks up the turtle's pen

s.right(90)

s.forward(100)

s.right(90)

s.pendown() #puts down the turtle's pen

s.forward(100)

COLOUR FILLING

import turtle

s=turtle.Turtle()

s.fillcolor("black")

s.begin\_fill() #start colour filling inside

for i in range(4):

s.forward(100)

s.right(90)

s.end\_fill() #completing the colour

'''Drawing heart include name'''

import turtle

s=turtle.Turtle()

s.fillcolor("red")

s.begin\_fill()

s.right(45)

s.forward(100)

s.left(90)

s.forward(100)

s.circle(50,180)

s.right(90)

s.circle(50,180)

s.end\_fill()

s.write("DUKE",align="left",font=("Arial", 37, "bold"))

'''Draw the rangoli'''

import turtle

screen = turtle.Screen()

screen.bgcolor("black")

spiral = turtle.Turtle()

spiral.speed(20)

spiral.width(3)

colors = ["red","green","orange", "yellow", "blue", "purple","teal"]

for i in range(360):