SEMESTER 2 HOMETASK

1.Create a BankAccount class that handles common operations like depositing money, withdrawing money, and checking the balance.

class Bank:

def \_\_init\_\_(self,holder\_name,acc\_num,initial\_bal=0):

self.holder\_name=holder\_name

self.acc\_num=acc\_num

self.balance=initial\_bal

def deposit(self,amount):

if amount>0:

print(f"you have DEPOSITED {amount} successfully")

self.balance+=amount

else:

print("your DEPOSIT AMOUNT is invalid")

def withdraw(self,amount):

if amount<self.balance:

print(f"you have WITHDRAWN {amount} successfully")

self.balance-=amount

else:

print("you DON'T have sufficient fund")

def check\_balance(self):

print(f" your CURRENT balance is {self.balance}")

user=Bank("Rickshi",8072688557,10000)

user.deposit(1000)

user.check\_balance()

user.withdraw(5000)

user.check\_balance()

2. Scenario: Cosmetic Product Information

class Cosmetics:

def \_\_init\_\_(self,product\_name="COMPACT POWDER",price=190,brand="DAZZLER",category="MAKEUP"):

self.product\_name=product\_name

self.price=price

self.brand=brand

self.category=category

def show(self):

print(f"The product name is {self.product\_name}")

print(f"Price is {self.price}")

print(f"Brand is {self.brand}")

print(f"Category is {self.category}")

choice=Cosmetics()

choice.show()

Date:27.11.2024

1 Given students details, marks in 5 subjects and we have to find student's grade.In this program, you have to take student name, roll number and marks in 3 subjects and calculating student's grade based on the percentage and printing the all details.

student\_name=input("Enter the Student Name:")

student\_rollno=int(input("Enter The Roll No:"))

a=int(input("Enter The Mark In Tamil:"))

b=int(input("Enter The Mark In English:"))

c=int(input("Enter The Mark In Maths:"))

d=int(input("Enter The Mark In Science:"))

e=int(input("Enter The Mark In Social:"))

total=a+b+c+d+e

percentage=total/5

print(f"Total Percentage Scored:{percentage}%")

if percentage>=85:

print(f"Your Grade is S")

elif percentage>=75:

print("Grade A")

elif percentage>=65:

print("Grade B")

elif percentage>=55:

print("Grade C")

elif percentage>=50:

print("Grade D")

else:

print("Grade E")

or

class Student:

def \_\_init\_\_(self,name,rollno,mark1,mark2,mark3):

self.name=name

self.rollno=rollno

self.mark1=mark1

self.mark2=mark2

self.mark3=mark3

def calculate\_per(self):

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

percentage=(total/300)\*100

return percentage

def display\_percentage(self):

percentage=self.calculate\_per()

print(f"your percentage is {percentage}")

def grade(self):

percentage=self.calculate\_per()

if percentage>=85:

print("your grade is S")

elif percentage>=75 and percentage <=85:

print("your grade is A")

elif percentage>=65 and percentage <=75:

print("your grade is B")

elif percentage>=55 and percentage <=65:

print("your grade is C")

elif percentage>=50 and percentage <=55:

print("your grade is D")

s=Student("Harini","E24AI012",90,91,86)

s.calculate\_per()

s.display\_percentage()

s.grade()

2. Implement destructor and constructors using \_\_del\_\_() and \_\_init\_\_() to display student information.

class Student:

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

self.name=name

self.age=age

self.course=course

self.grade=grade

def show(self):

print(f"student Details-->Student Name {self.name},Student Age {self.age},Student Course {self.course},Student Grade {self.grade}")

def \_\_del\_\_(self):

print("ALL Details Are Deleted")

stu1=Student("Rickshi",18,"AI","A")

stu1.show()

del stu1

Date:28.11.2024

1. a Python program that demonstrates single inheritance. Create a parent class called Person with an attribute name and a method show\_name to display the name. Create a child class called Student that inherits from the Person class and adds a new attribute student\_id with a method show\_student\_id to display the student ID. Create an object of the Student class, and use it to display both the name and student ID.

class Person:

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

self.name=name

def display(self):

print(f"The Person Nmae:{self.name}")

class Student(Person):

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

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

self.stu\_id=stu\_id

def show(self):

print(f"The Student ID is:{self.stu\_id}")

stu=Student("Rickshi",37)

stu.display()

stu.show()

2.Write a Python program to demonstrate single inheritance. Create a parent class Employee with attributes name and salary, and a method display\_details to show the employee's details. Create a

child class Manager that inherits from Employee and adds an attribute department, along with a method display\_department to show the department name. Create an object of the Manager class to

display all details

class Employee: #parent class

def getEmployeeInfo(self):

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

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

def displayEmployeeInfo(self):

print("Name=",self.name,"\n Salary=",self.salary)

class Manager(Employee): #child class

def getDetails(self):

self.getEmployeeInfo()

self.dep=input("Enter the department")

def displayInfo(self):

self.displayEmployeeInfo()

print("salary",self.dep)

emp=Manager()

emp.getDetails()

emp.displayInfo()

Date:29.11.2024

1. #1. Scenario: Library Management System

#Creating a simple library management system where:

#• Library handles book details.• Member handles member details.• LibraryManagement combines the features of both Library and Member and allows borrowing books

class Library:

def \_\_init\_\_(self,library\_name,library\_address):

self.library\_name=library\_name

self.library\_address=library\_address

def displayLibraryinfo(self):

print(f"Name of Library:{self.library\_name}\nLibrary Address:{self.library\_address}")

class Members:

def \_\_init\_\_(self,member\_name,member\_contact):

self.member\_name=member\_name

self.member\_contact=member\_contact

def displaymembersinfo(self):

print(f"Name of Members:{self.member\_name}\nMember Contact:{self.member\_contact}")

class Librarymanagement(Library,Members):

def \_\_init\_\_(self,library\_name,library\_address,member\_name,member\_contact):

super().\_\_init\_\_(library\_name,library\_address)

Members.\_\_init\_\_(self,member\_name,member\_contact)

def display(self):

self.displayLibraryinfo()

self.displaymembersinfo()

lib=Librarymanagement("A to Z","No.59,Kamber Street","Rickshi",9094563354)

lib.display()

2.Scenario: Food Delivery System

Create a system where:

• Restaurant handles the menu and food preparation. • Delivery manages the delivery details and rider information. • Order combines both Restaurant and Delivery to process food orders and manage delivery logistics

class Restaurant:

def \_\_init\_\_(self,Restaurant\_name,Items):

self.Restaurant\_name=Restaurant\_name

self.Items=Items

def displayrestaurantInfo(self):

print(f"Name of the Restaurant:{self.Restaurant\_name}\nItems Available:{self.Items}")

class Delivery:

def \_\_init\_\_(self,Deliveryboy\_name,boy\_contact):

self.Deliveryboy\_name=Deliveryboy\_name

self.boy\_contact=boy\_contact

def displaydelieryinfo(self):

print(f"Name of the delivery Boy:{self.Deliveryboy\_name}\nBoy's COntact:{self.boy\_contact}")

class Order(Restaurant,Delivery):

def \_\_init\_\_(self,Restaurant\_name,Items,Deliveryboy\_name,boy\_contact):

super().\_\_init\_\_(Restaurant\_name,Items)

Delivery.\_\_init\_\_(self,Deliveryboy\_name,boy\_contact)

def display(self):

self.displayrestaurantInfo()

self.displaydelieryinfo()

Del=Order("ABC Kitchen","Idly,Dosa,Poori,Chappthi","Rakshan",8072688557)

Del.display()

DATE:09.12.2024

1.Multiple inheritance example - Imagine a Smartphone that combines features of a Camera and a

Phone. Use multiple inheritance to implement the following.

class Camera:

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

self.resolution=input("Enter the resolution")

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

def display\_camera(self):

print("Resolution:",self.resolution)

print("Lens Name:",self.Lens)

class Phone:

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

self.phone\_no=int(input("Enter the phone number:"))

def display\_phone(self):

print("Phone Number:",self.phone\_no)

class SmartPhone(Camera,Phone):

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

Camera.\_\_init\_\_(self)

Phone.\_\_init\_\_(self)

self.brand=input("Enter the brand")

self.model=input("Enter the model")

def displaydevice(self):

print("Model:",self.model)

print("Brand:",self.brand)

self.display\_camera()

self.display\_phone()

s=SmartPhone()

s.displaydevice()

2.Single inheritance example - Student as the parent class, providing attributes and methods related

to education (e.g., studentInfo()).

class Student:

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

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

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

def studentInfo(self):

print("Student Name:",self.name)

print("Department:",self.department)

class StudentAthelete(Student):

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

Student.\_\_init\_\_(self)

self.sports\_name=input("Enter the sports that student going to participate:")

def atheletedetails(self):

self.studentInfo()

print("Sports Name:",self.sports\_name)

s=StudentAthelete()

s.atheletedetails()

DATE:10.12.2024

1.Vehicle – example hybrid inheritance

1. Vehicle (Base Class): Represents a general vehicle with basic attributes like

make, model, and year.

2. Car (Derived from Vehicle): Represents cars, which have additional

features like the number of doors and trunk capacity.

3. Truck (Derived from Vehicle): Represents trucks, which have attributes

like cargo capacity and number of axles.

4. PickupTruck (Derived from both Car and Truck): A specific type of vehicle

that combines features of both cars (passenger-friendly) and trucks (cargofriendly). Method – display all the feature

class Vehicle:

def \_\_init\_\_(self,make,model,year):

self.make=make

self.model=model

self.year=year

def displayVehicle(self):

print(f"Make:{self.make}\nModel:{self.model}\nYear:{self.year}")

class Car(Vehicle):

def \_\_init\_\_(self,make,model,year,n\_doors,trunk\_cap):

Vehicle.\_\_init\_\_(self,make,model,year)

self.n\_doors=n\_doors

self.trunk\_cap=trunk\_cap

def displayCar(self):

print(f"Number of doors in car:{self.n\_doors}\nTrunk Capacity:{self.trunk\_cap}")

class Truck(Vehicle):

def \_\_init\_\_(self,make,model,year,cargo\_cap,n\_axles):

super().\_\_init\_\_(make,model,year)

self.cargo\_cap=cargo\_cap

self.n\_axles=n\_axles

def displayTruck(self):

print(f"Cargo Capacity:{self.cargo\_cap}\nNumber of Axles:{self.n\_axles}")

class PickupTruck(Car,Truck):

def \_\_init\_\_(self,make,model,year,n\_doors,truck\_cap,cargo\_cap,n\_axles):

Car.\_\_init\_\_(self,make,model,year,n\_doors,truck\_cap)

Truck.\_\_init\_\_(self,make,model,year,cargo\_cap,n\_axles)

def displayfeatures(self):

self.displayVehicle()

self.displayCar()

self.displayTruck()

vec=PickupTruck("Ford","F",2000,6,2000,9000,2)

vec.displayfeatures()

2.Inventory Management System [Hierarchical inheritance]

1. Product (Base Class): Defines common attributes like product ID, name,

and price. Method to display all the info.

2. Electronics (Derived Class): Inherits from Product and adds attributes

like warranty period and brand. Method to display all the info.

3. Clothing (Derived Class): Inherits from Product and adds attributes like

size and material. Method to display all the info

class Product:

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

self.Id=Id

self.name=name

self.price=price

def displayProduct(self):

print("Product ID:",self.Id)

print("Product name:",self.name)

print("Product price:",self.price)

class Electronics(Product):

def \_\_init\_\_(self,Id,name,price,warranty,brand):

Product.\_\_init\_\_(self,Id,name,price)

self.warranty=warranty

self.brand=brand

def displayElectronics(self):

print("Warranty:",self.warranty)

print("Brand:",self.brand)

class Clothing(Product):

def \_\_init\_\_(self,Id,name,price,size,material):

Product.\_\_init\_\_(self,Id,name,price)

self.size=size

self.material=material

def displayClothing(self):

print("Material:",self.material)

print("Size:",self.size)

phone=Electronics(123456,"Moto","25,000.","1-year","motorola")

shirt=Clothing(98765,"Chudithar","600rs.","L","Cotton")

phone.displayProduct()

phone.displayElectronics()

shirt.displayProduct()

shirt.displayClothing()

Date:11.12.2024

1. Create a class that validates the password which contains various methods to validate the

password. Get input from the user and validate the password.

Write a program to validate the password based on these rules and provide feedback(Valid

or invalid).Password rules:

At least one uppercase letter. At least one lowercase letter. At least one digit. At least one special character.

Minimum length of 8 characters.

class Password:

def validate(text):

u\_count=0

l\_count=0

d\_count=0

s\_count=0

length=len(text)

for i in text:

if i.isupper():

u\_count+=1

elif i.islower():

l\_count+=1

elif i.isdigit():

d\_count+=1

else:

s\_count+=1

if u\_count>=1 and l\_count>=1 and d\_count>=1 and s\_count>=1 and length>=8:

print("your password is vaild")

else:

print("Your password is invaild")

user=input("Enter the Password:")

Password.validate(user)

2 Create a class program that takes a paragraph of text as input and splits it into individual

sentences. Additionally, process each sentence to provide useful information, such as word

count or other linguistic analysis."

Requirements:

Design a class called TextProcessor.

Include a method to split the text into sentences (split\_into\_sentences).

Include a method to further process each sentence (process\_sentences), such as counting

words.

Sample :

Input: "Hello! How are you? I am fine. Let's learn NLP."

Output:

Split Sentences:

o "Hello!"

o "How are you?"

o "I am fine."

o "Let's learn NLP."

Processed Sentence Data:

o Sentence: "Hello!", Word Count: 1

o Sentence: "How are you?", Word Count: 3

o Sentence: "I am fine.", Word Count: 3

o Sentence: "Let's learn NLP.", Word Count: 3

class Textprocessor:

def \_init\_(self):

self.text=input("Enter your text").strip()

def split\_into\_sentences(self):

sentences=[]

split=""

for i in range(len(self.text)):

split+=self.text[i]

if self.text[i] in ".!,?":

sentences.append(split)

split=""

if split.strip():

sentences.append(split.strip())

print(sentences)

for sentence in sentences:

word\_count = len(sentence.split())

print(f"{sentence} - count: {word\_count}")

t=Textprocessor()

t.split\_into\_sentences()

Date:14.12.2024

2.You are given two strings word1 and word2. Merge the strings by adding letters in alternating

order, starting with word1. If a string is longer than the other, append the additional letters

onto the end of the merged string.

Return the merged string.

Example 1:

Input: word1 = "abc", word2 = "pqr"

Output: "apbqcr"

Explanation: The merged string will be merged as so:

word1: a b c

word2: p q r

merged: a p b q c r

Example 2:

Input: word1 = "ab", word2 = "pqrs"

Output: "apbqrs"

Explanation: Notice that as word2 is longer, "rs" is appended to the end.

word1: a b

word2: p q r s

merged: a p b q r s

def merge\_strings(word1, word2):

merged = ""

min\_length = min(len(word1), len(word2))

# Alternate characters from both strings

for i in range(min\_length):

merged += word1[i] + word2[i]

# Append remaining characters from the longer string

merged += word1[min\_length:] + word2[min\_length:]

return merged

# Test the function

print(merge\_strings("abc", "pqr")) # Output: "apbqcr"

print(merge\_strings("ab", "pqrs")) # Output: "apbqrs"

2.You have a long flowerbed in which some of the plots are planted, and some are not. However,

flowers cannot be planted in adjacent plots.

Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means

not empty, and an integer n, return true if n new flowers can be planted in the flowerbed

without violating the no-adjacent-flowers rule and false otherwise.

Example 1:

Input: flowerbed = [1,0,0,0,1], n = 1

Output: true

Example 2:

Input: flowerbed = [1,0,0,0,1], n = 2

Output: false

Constraints:

1 <= flowerbed.length <= 2 \* 104

flowerbed[i] is 0 or 1.

There are no two adjacent flowers in flowerbed.

0 <= n <= flowerbed.length

def canPlaceFlowers(flowerbed, n): #[1,0,0,0,1],n=1

count = 0

flowerbed = [0] + flowerbed + [0] #{0,1,0,0,0,1,0}

for i in range(1, len(flowerbed) - 1): #1,6-1=5

if flowerbed[i] == 0 and flowerbed[i - 1] == 0 and flowerbed[i + 1] == 0:

flowerbed[i] = 1

count += 1

if count >= n:

return True

else:

return False

print(canPlaceFlowers([1,0,0,0,1], 1)) # Output: True

print(canPlaceFlowers([1,0,0,0,1], 2)) # Output: False

print(canPlaceFlowers([1,0,0,0,1], 3)) # Output: False

print(canPlaceFlowers([1,0,0,0,1], 4)) # Output: False

print(canPlaceFlowers([1,0,0,0,1], 5)) # Output: False

Date:16.12.2024

1. User Login System: Write a User class with:

o Private attributes \_username and \_password.

o A setter method set\_password to ensure the password:

▪ Is at least 8 characters long.

▪ Contains at least one number and one special character.

o A method check\_password(input\_password) to verify the password.

class User:

def \_\_init\_\_(self,username,password):

self.\_\_username=username

self.\_\_password=password

def get\_username(self):

return self.\_\_username

def get\_password(self):

return self.\_\_password

def set\_username(self):

self.\_\_username=username

def set\_password(Self):

self.\_\_password=password

def checkpassword(self):

digit=False

char=False

if len(self.\_\_password)<8:

return False

if any (i.isdigit() for i in self.\_\_password):

digit=True

if any (not i.isalnum() for i in self.\_\_password):

char=True

if digit and char:

print("password is valid")

return True

else:

if not digit or not char:

return False

username=input("Enter the username")

password=input("Enter the password:")

u=User(username,password)

if u.checkpassword():

print(f"Username:{u.get\_username()}\nPassword:{u.get\_password()}")

else:

print("Invalid credentials")

2. Encapsulation with Validation: Create a Product class with:

o Private attributes \_name, \_price, and \_stock.

o set\_price method to ensure price is greater than 0.

o set\_stock method to ensure stock is an integer and non-negative.

o A getter method get\_stock to retrieve the current stock.

class Product:

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

self.\_\_name=name

self.\_\_price=price

self.\_\_stock=stock

def get\_name(self):

return self.\_\_name

def get\_price(self):

return self.\_\_price

def get\_stock(self):

return self.\_\_stock

def set\_name(self):

self.\_\_name=name

self.\_\_price=price

self.\_\_stock=stock

def check\_price\_and\_stock(self):

price=False

stock=False

if self.\_\_price>0:

price=True

if type(self.\_\_stock)==int and self.\_\_stock>0:

stock=True

if price and stock:

print("Valid details")

return True

else:

return False

name=input("Enter the name")

price=int(input("Enter the price"))

stock=int(input("Enter the stock available"))

p=Product(name,price,stock)

if p.check\_price\_and\_stock():

print(f"Name:{p.get\_name()}\nPrice:{p.get\_price()}\nStock:{p.get\_stock()}")

else:

print("No valid details")

3)Basic Getter and Setter: Create a Student class with the following private attributes:

o \_name

o \_age

o \_marks

Implement getter and setter methods to:

o Set and retrieve the name.

o Ensure age is between 5 and 100; otherwise, raise a ValueError.

Ensure marks are between 0 and 100; otherwise, raise a ValueError.

class Student:

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

self.\_\_name=name

self.\_\_age=age

self.\_\_marks=marks

def get\_name(self):

return self.\_\_name

def get\_age(self):

return self.\_\_age

def get\_marks(self):

return self.\_\_marks

def check\_details(self):

if not self.\_\_age>5 and self.\_\_age<100:

raise ValueError("Age is invalid")

if not self.\_\_marks>0 and self.\_\_marks<=100:

raise ValueError("Marks are invalid")

try:

name=input("Enter the name:")

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

marks=int(input("Enter the marks"))

s=Student(name,age,marks)

s.check\_details()

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

print(f"Age:{s.get\_age()}")

print(f"Marks:{s.get\_marks()}")

except ValueError as e:

print(e)

Date : 18-12-24 Task – Method overloading

Implement a class Calculator with a method calculate that performs different

operations based on the input parameters:

1. If one argument is provided, return its square.

2. If two arguments are provided, return their sum.

3. If three arguments are provided, return their product.

If any of the arguments are not integers or floats, or if the input doesn't match

the constraints, raise a ValueError

class Calculator:

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

result=0

for i in (a,b,c):

if type(i) not in (int,float):

raise ValueError("It must Be in Integer and float")

if a!=0 and b==0 and c==0:

result=a\*a

return result

elif a!=0 and b!=0 and c==0:

result=a+b

return result

else:

result=a\*b\*c

return result

ad=Calculator()

try:

ans=ad.calculate(3)

print(ans)

ans1=ad.calculate(5,2)

print(ans1)

ans2=ad.calculate(5,3,4)

print(ans2)

ans3=ad.calculate(5,3,"A")

except ValueError as e:

print(e)

Date: 19-12-24 Task : Code

Seven different symbols represent Roman numerals with the following values:

Symbol Value

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

Roman numerals are formed by appending the conversions of decimal place

values from highest to lowest. Converting a decimal place value into a Roman

numeral has the following rules:

If the value does not start with 4 or 9, select the symbol of the maximal value

that can be subtracted from the input, append that symbol to the result, subtract

its value, and convert the remainder to a Roman numeral.

If the value starts with 4 or 9 use the subtractive form representing one symbol

subtracted from the following symbol, for example, 4 is 1 (I) less than 5 (V): IV

and 9 is 1 (I) less than 10 (X): IX. Only the following subtractive forms are used:

4 (IV), 9 (IX), 40 (XL), 90 (XC), 400 (CD) and 900 (CM).

Only powers of 10 (I, X, C, M) can be appended consecutively at most 3 times to

represent multiples of 10. You cannot append 5 (V), 50 (L), or 500 (D) multiple

times. If you need to append a symbol 4 times use the subtractive form.

Given an integer, convert it to a Roman numeral

roman =[

(1000, 'M'), (900, 'CM'), (500, 'D'), (400, 'CD'),

(100, 'C'), (90, 'XC'), (50, 'L'), (40, 'XL'),

(10, 'X'), (9, 'IX'), (5, 'V'), (4, 'IV'), (1, 'I')]

n=int(input("Enter the number to convert into a roman number"))

ans=""

for val,letter in roman:

while n>=val:

ans+=letter

n-=val

print("The roman for the given integer is:",ans)

Date: 20-12-24 Task: Inheritance

OOPS Library Management System. It will allow users to manage books, register members, and borrow books. 1. Create the following classes Member Class: o Attributes: member\_id, name, and email. (Check the email format) o Method generate\_member\_id() to generate IDs in the format LIB1234. o Method verify\_member\_id(member\_id) to check the ID format. Library Class: (should inherit member class) o Manage books (book\_id, title, author). o Register members. o Borrow and return books.

import random

import re

class Member:

def \_\_init\_\_(self,member\_id,name,email):

self.member\_id=member\_id

self.name=name

self.email=email

def verify\_email(self):

ex=r'^[a-z0-9.\_%+-]+@gmail\.com$'

if re.match(ex,email):

print("Valid email id")

else:

print("Email id is not valid")

def verify\_member\_id(self):

mem=r'^LIB\d{4}$'

if re.match(mem,member\_id):

print("Valid member id")

else:

self.generate\_member\_id()

def generate\_member\_id(self):

self.member\_id="LIB"+str(random.randint(1000,9999))

class Library(Member):

def \_init\_(self,member\_id,name,email,book\_id,title,author):

Member.\_\_init\_\_(self,member\_id,name,email)

self.book\_id=book\_id

self.title=title

self.author=author

def display\_overall(self):

print("Member id:",self.member\_id)

print("Name:",self.name)

print("Email:",self.email)

print("Book borrowed or returned id:",self.book\_id)

print("Book titile:",self.title)

print("Author:",self.author)

member\_id=input("Enter the id of the member:")

name=input("Enter the name:")

email=input("Enter the email id")

book\_id=input("Enter the book id:")

title=input("Enter the book title:")

author=input("Enter the name of the author:")

l=Library(member\_id,name,email,book\_id,title,author)

l.verify\_email()

l.verify\_member\_id()

l.display\_overall()

Date : 23-12-24

'''1. You are building a program to track user login attempts. Write a function that:

• Records a new login attempt.

• Limits login attempts to 5.

• Locks the account after 5 failed attempts.'''

Id=True

LOGIN\_CHECK="RICK123"

for i in range(5):

login=input("Enter the Login ID:")

if login==LOGIN\_CHECK:

Id=True

print("Login Id is valid")

exit()

else:

Id=False

print("Your login id was wrong")

print(f"You Have {5-(i+1)} More attempts TRY AGAIN")

if not Id:

print("Login In Id failed your account is locked")

print("Try again After Some time")

'''3. Write a program that takes a number as input and returns the sum of its digits'''

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

sum\_of\_digits=0

for digit in str(num):

sum\_of\_digits+=int(digit)

print(sum\_of\_digits)

'''4.4. Identify common elements between two lists.'''

l1=[1,2,3]

l2=[2,3,4]

l3=[]

for i in l1:

if i in l2:

l3.append(i)

print(l3)

'''5. Write a program that counts the number of words in a given string.'''

sen=input()

word\_count=len(sen.split())

print(word\_count)

'''6. Sort a list of integers without using Python's sorted() function.'''

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

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

for i in range(n):

for j in range(0,n-i-1):

if elements[j]>elements[j+1]:

elements[j],elements[j+1]=elements[j+1],elements[j]

print(elements)

'''7)Create a BankAccount class to simulate a bank account with features:

Deposit money.

Withdraw money (ensure sufficient balance).

Check balance.'''

print("1.Deposit 2.Withdraw 3.View balance 4.Exit")

balance=10000

while True:

choice=int(input("Enter the choice you wanted"))

match choice:

case 1:

print("Deposit")

deposit=int(input("Enter the deposit amount"))

balance+=deposit

print(f"You have successfully deposited {deposit}")

case 2:

print("Withdraw")

withdraw=int(input("Enter the withdraw amount"))

if balance>=withdraw:

print(f"You have successfully withdrawn {withdraw}")

balance-=withdraw

else:

print("You have insyfficient balance")

case 3:

print("Check balance")

print(balance)

case 4:

exit()

case \_:

print("Invalid Choice")

'''8)Check if a given string is a valid email address'''

import re

email=input("Enter the email id:")

ex=r'^[a-z0-9.\_%+-]+@gmail\.com$'

if re.match(ex,email):

print("Valid")

else:

print("invalid")

'''9.Extract all phone numbers from a given text.'''

text=input("Enter the text")

phone=[]

for i in text:

if i.isdigit()==True:

phone.append(i)

for i in phone:

print(i,end="")

'''10. Extract all hashtags from a given text'''

import re

text=input("Enter the text:")

ex=r'#\w+'

result=re.findall(ex,text)

print(result)

Date:24.12.2024

1.You are building a program to track user login attempts. Write a function that:

• Records a new login attempt.

• Limits login attempts to 5.

• Locks the account after 5 failed attempts.

Id=True

LOGIN\_CHECK="RICK123"

for i in range(5):

login=input("Enter the Login ID:")

if login==LOGIN\_CHECK:

Id=True

print("Login Id is valid")

exit()

else:

Id=False

print("Your login id was wrong")

print(f"You Have {5-{i+1} More attempts TRY AGAIN")

if not Id:

print("Login In Id failed your account is locked")

print("Try again After Some time")

3.Write a program that takes a number as input and returns the sum of its digits

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

sum\_of\_digits=0

for digit in str(num):

sum\_of\_digits+=int(digit)

print(sum\_of\_digits)

4.Identify common elements between two lists.

l1=[1,2,3]

l2=[2,3,4]

l3=[]

for i in l1:

if i in l2:

l3.append(i)

print(l3)

5.Write a program that counts the number of words in a given string.

sen=input()

word\_count=len(sen.split())

print(word\_count)

'6.Sort a list of integers without using Python's sorted() function.

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

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

for i in range(n):

for j in range(0,n-i-1):

if elements[j]>elements[j+1]:

elements[j],elements[j+1]=elements[j+1],elements[j]

print(elements)

'7)Create a BankAccount class to simulate a bank account with features:

Deposit money.

Withdraw money (ensure sufficient balance).

Check balance.

print("1.Deposit 2.Withdraw 3.View balance 4.Exit")

balance=10000

while True:

choice=int(input("Enter the choice you wanted"))

match choice:

case 1:

print("Deposit")

deposit=int(input("Enter the deposit amount"))

balance+=deposit

print(f"You have successfully deposited {deposit}")

case 2:

print("Withdraw")

withdraw=int(input("Enter the withdraw amount"))

if balance>=withdraw:

print(f"You have successfully withdrawn {withdraw}")

balance-=withdraw

else:

print("You have insyfficient balance")

case 3:

print("Check balance")

print(balance)

case 4:

exit()

case \_:

print("Invalid Choice")

8)Check if a given string is a valid email address

import re

email=input("Enter the email id:")

ex=r'^[a-z0-9.\_%+-]+@gmail\.com$'

if re.match(ex,email):

print("Valid")

else:

print("invalid")

9.Extract all phone numbers from a given text.

text=input("Enter the text")

phone=[]

for i in text:

if i.isdigit()==True:

phone.append(i)

for i in phone:

print(i,end="")

10. Extract all hashtags from a given text

import re

text=input("Enter the text:")

ex=r'#\w+'

result=re.findall(ex,text)

print(result)

Date:23.12.2024

1.You are building a program to track user login attempts. Write a function that:

• Records a new login attempt.

• Limits login attempts to 5.

• Locks the account after 5 failed attempts.

Id=True

LOGIN\_CHECK="RICK123"

for i in range(5):

login=input("Enter the Login ID:")

if login==LOGIN\_CHECK:

Id=True

print("Login Id is valid")

exit()

else:

Id=False

print("Your login id was wrong")

print(f"You Have {5-{i+1} More attempts TRY AGAIN")

if not Id:

print("Login In Id failed your account is locked")

print("Try again After Some time")

3.Write a program that takes a number as input and returns the sum of its digits

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

sum\_of\_digits=0

for digit in str(num):

sum\_of\_digits+=int(digit)

print(sum\_of\_digits)

4.Identify common elements between two lists.

l1=[1,2,3]

l2=[2,3,4]

l3=[]

for i in l1:

if i in l2:

l3.append(i)

print(l3)

5.Write a program that counts the number of words in a given string.

sen=input()

word\_count=len(sen.split())

print(word\_count)

'6.Sort a list of integers without using Python's sorted() function.

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

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

for i in range(n):

for j in range(0,n-i-1):

if elements[j]>elements[j+1]:

elements[j],elements[j+1]=elements[j+1],elements[j]

print(elements)

'7)Create a BankAccount class to simulate a bank account with features:

Deposit money.

Withdraw money (ensure sufficient balance).

Check balance.

print("1.Deposit 2.Withdraw 3.View balance 4.Exit")

balance=10000

while True:

choice=int(input("Enter the choice you wanted"))

match choice:

case 1:

print("Deposit")

deposit=int(input("Enter the deposit amount"))

balance+=deposit

print(f"You have successfully deposited {deposit}")

case 2:

print("Withdraw")

withdraw=int(input("Enter the withdraw amount"))

if balance>=withdraw:

print(f"You have successfully withdrawn {withdraw}")

balance-=withdraw

else:

print("You have insyfficient balance")

case 3:

print("Check balance")

print(balance)

case 4:

exit()

case \_:

print("Invalid Choice")

8)Check if a given string is a valid email address

import re

email=input("Enter the email id:")

ex=r'^[a-z0-9.\_%+-]+@gmail\.com$'

if re.match(ex,email):

print("Valid")

else:

print("invalid")

9.Extract all phone numbers from a given text.

text=input("Enter the text")

phone=[]

for i in text:

if i.isdigit()==True:

phone.append(i)

for i in phone:

print(i,end="")

10. Extract all hashtags from a given text

import re

text=input("Enter the text:")

ex=r'#\w+'

result=re.findall(ex,text)

print(result)

Date: 24-12-24 Task : function with error handling

Use class and object : Provide the method inside the class and call the method using the object.

You are building an e-commerce platform, and one of the functionalities is to calculate the final price

of a product after applying a discount and tax. The price calculation is based on the following:

1. A base price of the product.

2. A discount percentage (e.g., 10% off).

3. A tax percentage (e.g., 5% tax).

Write a Python function, calculate\_final\_price, that takes the following inputs:

• base\_price (float): The original price of the product.

• discount\_percentage (float): The percentage discount to be applied (default is 0%).

• tax\_percentage (float): The percentage tax to be applied (default is 0%).

The function should:

• Return the final price of the product, rounded to two decimal places.

• Include error handling to ensure that negative values for base\_price, discount\_percentage,

or tax\_percentage raise a ValueError.

class Commerce:

def \_\_init\_\_(self,base\_price,discount\_percentage=0,tax\_percentage=0):

self.base\_price=base\_price

self.discount\_percentage=discount\_percentage

self.tax\_percentage=tax\_percentage

def final(self):

if base\_price <0 or discount\_percentage <0 or tax\_percentage<0:

raise ValueError ("Invalid amount")

else:

discount\_amount=((discount\_percentage)/100)\*base\_price

tax\_amount=(tax\_percentage/100)\*base\_price

total\_price=(base\_price+tax\_amount)-discount\_amount

return total\_price

base\_price=float(input("Enter the base price "))

tax\_percentage=float(input("Enter the tax percentage"))

discount\_percentage=float(input("enter the discount percentage"))

try:

c=Commerce(base\_price,discount\_percentage,tax\_percentage)

result=c.final()

print(result)

except ValueError as e:

print(e)

Date : 26-12-24 Task :Exceptional handling

You are building a Python program that interacts with a simple calculator class. The program takes

two numbers and an operation (e.g., addition, subtraction) as input from the user. It should handle

the following exceptions:

1. ZeroDivisionError: If the user attempts to divide by zero, inform the user and allow them to

retry.

2. ValueError: If the user provides non-numeric input for the numbers, inform them and allow

them to retry.

3. KeyError: If the user provides an invalid operation (e.g., not +, -, \*, or /), inform them and

allow them to retry.

4. TypeError: If any unexpected data type issue occurs, catch it and raise the error

class Calculator:

def \_\_init\_\_(self,operator1,operator2):

self.operator1=operator1

self.operator2=operator2

def choices(self):

if not isinstance(operator1,int) or not isinstance(operator2,int):

raise ValueError("DATA TYPE IS WRONG TRY AGAIN")

else:

if choice>3:

raise KeyError("CHOICES ARE INVALID")

else:

match choice:

case 1:

result=self.operator1+self.operator2

return result

case 2:

result=self.operator1-self.operator2

return result

case 3:

if self.operator2==0:

raise ZeroDivisionError ("OPERATOR SHOULD NOT BE ZERO, IT SHOULD STARTS FROM 1")

else:

result=self.operator1/self.operator2

return result

case \_:

exit()

try:

while True:

print("1)addition \n2)subtraction \n3)division")

choice=int(input("Enter your choice : "))

operator1=int(input("Enter the operator number 1:"))

operator2=int(input("Enter the operator number 2:"))

cal=Calculator(operator1,operator2)

result=cal.choices()

print(result)

except ValueError as e:

print(e)

except KeyError as e:

print(e)

except ZeroDivisionError as e:

print(e)

DATE:30.12.2024

1. Write a program to count how many alphabetic characters, numbers, and special

characters are in a string.

Input: Enter a string: Prep123!@#

Output:

Alphabetic Characters: 4

Numeric Characters: 3

Special Characters: 3

string=input("Enter the string to count:")

alphabet=0

number=0

special=0

for i in string:

if i.isalpha():

alphabet+=1

elif i.isdigit():

number+=1

else:

if not i.isalnum():

special+=1

print("Alphabetic Characters Count:",alphabet)

print("Numeric Characters count",number)

print("Special Characters Count",special)

2. Write a program to validate whether the string contains at least one alphabet and one

special character.

Input: Enter a string: Prep!@123

Output: The string contains both alphabets and special character

import re

string=input("Enter the string you want to check:")

ex=r'^(?=.\*[a-zA-z].{1,})(?=.\*[!@#$%^&\*()]).{1,}$'

if re.match(ex,string):

print("Valid ")

else:

print("Not valid")

Date: 02-01-25 Task: Abstract class

Suppose you are creating a system for calculating payments for different types of employees, such as

salaried employees and hourly employees. Each employee type has a different way of calculating

their payment, but all employees should have a common interface.

Classes :

Abstract class : Employee

Abstract method : calculate pay

Concrete class :SalariedEmployee (Atrributes: Annual salary)

-CalculatePay() – should return the monthly pay of the employee

HourlyEmployee(Attributes – hoursworked and hourlyrate)

-CalculatePay() will return the pay based on the hours worked and rate.

Sample Input and Output:

For example, if you create a SalariedEmployee with an annual salary of $60,000 and an

HourlyEmployee with a rate of $20/hour who worked 120 hours, the program should calculate and

display their pay as follows:

Salaried Employee (John Doe) Pay: $5000.00

Hourly Employee (Jane Smith) Pay: $2400.00

from abc import ABC,abstractmethod

class Employee(ABC):

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

self.name=name

@absractmethod

def calculate\_pay(self):

pass

class SalariedEmployee:

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

self.name=name

self.annual\_sal=annual\_sal

def calculate\_pay(self):

return self.annual\_sal/12

class HourlyEmployee:

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

self.name=name

self.hoursworked=hoursworked

self.hourlyrate=hourlyrate

def calculate\_pay(self):

return self.hoursworked\*self.hourlyrate

n=input("Enter the name: ")

a=float(input("Enter the annual amount: "))

s=SalariedEmployee(n,a)

print(f"Salaried Employee ({n}) Pay: ${s.calculate\_pay():.2f}")

n=input("Enter the name: ")

h\_worked=float(input("Enter how many hours worked: "))

h\_rate=float(input("Enter the hourly rate: "))

hour=HourlyEmployee(n,h\_worked,h\_rate)

print(f"Hourly Employee ({n}) Pay: ${hour.calculate\_pay():.2f}")

DATE:17.01.2025

1.Sum of all items in a dictionary

my\_dict = {'a':1,'b': 2,'c': 3}

total\_sum = sum(my\_dict.values())

print(total\_sum)

2.Construct a pattern using a nested loop

**Pattern:**

1

2 2

3 3 3

4 4 4 4

n=int(input())

for i in range(1, n+1):

for j in range(i):

print(i,end="")

print()

DATE:20.01.2025

'''Topic : Encapsulation

Create a BankAccount class with private attributes:

• \_\_account\_number

• \_\_balance

Implement public methods to:

1. Set the account number and initialize the balance.

2. Deposit and withdraw money.

3. Implement a method add\_interest(rate) to calculate and add interest to the balance.'''

class BankAccount:

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

self.\_\_account\_number=int(input("Enter the Your Account Number:"))

self.\_\_initial\_balance=int(input("Enter the inital balance:"))

def get\_account\_number(self):

return self.\_\_account\_number

def get\_initial\_balance(self):

return self.\_\_initial\_balance

def deposit(self,amount):

if amount>0:

self.\_\_initial\_balance+=amount

print(f"Deposited ${amount:.2f}. Current balance: ${self.\_\_initial\_balance:.2f}")

else:

print("Invaild Amount")

def withdraw(self,amount):

if 0<amount<=self.\_\_initial\_balance:

self.\_\_initial\_balance-=amount

print(f"withdraw ${amount:.2f}. Current balance: ${self.\_\_initial\_balance:.2f}")

else:

print("Invaild Amount")

def add\_interest(self,rate):

interest=self.\_\_initial\_balance\*rate

self.\_\_initial\_balance+=interest

print(f"Added ${interest:.2f} interest. Current balance: ${self.\_\_initial\_balance:.2f}")

account = BankAccount()

print(f"Account Number: {account.get\_account\_number()}")

print(f"Initial Balance: ${account.get\_initial\_balance():.2f}")

des=int(input("Enter the despoist amount:"))

account.deposit(des)

withdraw\_amount=int(input("Enter the withdraw amount:"))

account.withdraw(withdraw\_amount)

rate\_per=int(input("Enter the interest rate:"))

account.add\_interest(rate\_per)

DATE:22.01.2025

Scenario: Employee Management System

A company has different types of employees: Full-Time Employees and Part-Time Employees. Each

employee has common details like name, employee ID, and department. However, the salary

calculation differs for each type:

• Full-Time Employee: Salary is calculated as a fixed monthly salary.

• Part-Time Employee: Salary is calculated based on hourly wages and hours worked.

You are tasked with designing an object-oriented solution using an abstract class to represent the

common behavior and structure of all employees while allowing specific implementations for each

type.

Hint:-

1. Create an abstract class Employee with the following:

o Fields: name, employee\_id, department.

o An abstract method calculate\_salary().

o A concrete method display\_details() to display employee details.

2. Create two subclasses:

o FullTimeEmployee: Implements calculate\_salary() for a fixed monthly salary.

o PartTimeEmployee: Implements calculate\_salary() based on hourly wage and hours

worked.

3. Write a program to create objects of both types, calculate their salaries, and display their

details.

from abc import ABC, abstractmethod

class Employee(ABC): #create a abstractmethod

def \_\_init\_\_(self, name, employee\_id, department):

self.name = name

self.employee\_id = employee\_id

self.department = department

@abstractmethod

def calculate\_salary(self):

pass

def display\_details(self):

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

print(f"Employee ID: {self.employee\_id}")

print(f"Department: {self.department}")

class FullTimeEmployee(Employee):

def \_\_init\_\_(self, name, employee\_id, department, monthly\_salary):

super().\_\_init\_\_(name, employee\_id, department)

self.monthly\_salary = monthly\_salary

def calculate\_salary(self):

return self.monthly\_salary

class PartTimeEmployee(Employee):

def \_\_init\_\_(self, name, employee\_id, department, hourly\_wage, hours\_worked):

super().\_\_init\_\_(name, employee\_id, department)

self.hourly\_wage = hourly\_wage

self.hours\_worked = hours\_worked

def calculate\_salary(self):

return self.hourly\_wage \* self.hours\_worked

# Create objects of both types

full\_time\_employee = FullTimeEmployee("Rickshi", "AI001", "Marketing", 9000)

part\_time\_employee = PartTimeEmployee("Kannama", "CS001", "Sales", 60, 100)

# Calculate salaries and display details

print("Full-time Employee:")

full\_time\_employee.display\_details()

print(f"Salary: ${full\_time\_employee.calculate\_salary():.2f}")

print()

print("Part-time Employee:")

part\_time\_employee.display\_details()

print(f"Salary: ${part\_time\_employee.calculate\_salary():.2f}")

Date: 23-01-25 Topic: Encapsulation

Create a Python class HotelRoom to manage details of a hotel room. The class should have the

following:

Attributes:

A private attribute \_\_room\_number (an integer).

A private attribute \_\_is\_occupied (a boolean to indicate if the room is occupied).

A private attribute \_\_rate\_per\_night (a float to store the rate for the room).

Methods:

A getter and setter for room\_number with validation to ensure it is always a positive integer.

A getter and setter for rate\_per\_night with validation to ensure the rate is greater than zero.

A method check\_in() to mark the room as occupied. Raise an exception if the room is already

occupied.

A method check\_out() to mark the room as unoccupied. Raise an exception if the room is already

vacant.

A method display\_details() to print the room details in the format:

Input:

# Create a room instance

room = HotelRoom()

# Set room details

room.set\_room\_number(101)

room.set\_rate\_per\_night(200.50)

# Display details

room.display\_details()

Output:

Room Number: 101

Rate per Night: $200.50

Occupied: No

class HotelRoom:

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

self.\_\_room\_number = None

self.\_\_is\_occupied = False

self.\_\_rate\_per\_night = None

def get\_room\_number(self):

return self.\_\_room\_number

def set\_room\_number(self, room\_number):

if not isinstance(room\_number, int) or room\_number <= 0:

raise ValueError("Room number must be a number and positive integer")

self.\_\_room\_number = room\_number

def get\_rate\_per\_night(self):

return self.\_\_rate\_per\_night

def set\_rate\_per\_night(self, rate\_per\_night):

if not isinstance(rate\_per\_night, (int, float)) or rate\_per\_night <= 0:

raise ValueError("Rate per night must be anumber and greater than zero")

self.\_\_rate\_per\_night = rate\_per\_night

def get\_occupation\_status(self):

return self.\_\_is\_occupied

def check\_in(self):

if self.\_\_is\_occupied:

raise Exception("Room is already occupied")

self.\_\_is\_occupied = True

def check\_out(self):

if not self.\_\_is\_occupied:

raise Exception("Room is already vacant")

self.\_\_is\_occupied = False

def display\_details(self):

occupied\_status = "Yes" if self.\_\_is\_occupied else "No"

print(f"Room Number: {self.\_\_room\_number}")

print(f"Rate per Night: ${self.\_\_rate\_per\_night:.2f}")

print(f"Occupied: {occupied\_status}")

# Test the class

room = HotelRoom()

room.set\_room\_number(101)

room.set\_rate\_per\_night(200.50)

room.display\_details()

DATE:24.01.2024

Keerthana wants to print the UserName and Department and Mobile

number and also print the same in the required format.

Conditions of Input:

Print your name Concatenate with First Name and Last Name

Print Concatenate with College name and Stream

Print the ASCII value of Your Name

Print the ASCII value of your Mobile Number

Print the ASCII value Addition Result of Your Name and Mobile

number with Alternate method

class Details:

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

self.first\_name=input("Enter the first name:")

self.second\_name=input("Enter the last name:")

self.phone=int(input("enter the phone number:"))

self.college\_name=input("Enter the college name:")

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

self.name=self.first\_name+self.second\_name

def display(self):

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

print("Name of College is :",self.college\_name,"Department of",self.stream)

def ASCII\_value(self):

print("ASCII values of Name:")

for i in self.name:

print(i,"-",ord(i),end=" ")

print()

print("ASCII values of Phone:")

for m in str(self.phone):

print(m,"-",ord(m),end=" ")

print()

def ASCII\_addition(self):

print("Result of Addition:")

self.ascii\_addition=[]

for i, m in zip(self.name, str(self.phone)):

addition = ord(i) + ord(m)

self.ascii\_addition.append(f"{i}+{m} = {addition}")

print("-".join(self.ascii\_addition))

det=Details()

det.display()

det.ASCII\_value()

det.ASCII\_addition()

Nithya wants to print the Operational Result with two user input

values.

1. Do the all Arithmetic operation and Print the result.

2. Swapping Three values using bitwise operator without third

variable .

def arithmetic\_operations():

n1 = int(input("Enter the first number: "))

n2 = int(input("Enter the second number: "))

print("Arithmetic Operations Are:")

print("Addition:", n1 + n2)

print("Subtraction:", n1 - n2)

print("Multiplication:", n1 \* n2)

print("Division:", n1 / n2)

print("Modulus:", n1 % n2)

print("Exponentiation:", n1 \*\* n2)

def swap\_three\_values():

n1 = int(input("Enter the first number: "))

n2 = int(input("Enter the second number: "))

n3 = int(input("Enter the third number: "))

print("Before Swapping:")

print("Number 1:", n1)

print("Number 2:", n2)

print("Number 3:", n3)

# Swap num1, num2, and num3 using bitwise XOR operator

n1 = n1 ^ n2

n2 = n1^ n2

n1 = n1 ^ n2

n2 = n2 ^ n3

n3 = n2 ^ n3

n2 = n2 ^ n3

print("After Swapping:")

print("Number 1:", n1)

print("Number 2:", n2)

print("Number 3:", n3)

arithmetic\_operations()

swap\_three\_values()

Kathir has to design a method that will accept the UserName Input:

Print Username with Conditions

Requirements: Name with Special Characters

a. Print the string with special characters.

Input:

Enter First Name : face123@#$%^

Output:

Hi ! Your Entered Input is “face@#$%^”

b. Split Characters and Special Characters from the given string.

Enter Name : faceprep12345!@#$%^&

Your entered Characters are –faceprep

Your entered Special Characters are -!@#$%^&

n=input("Enter the name:")

#Print the string with special characters.

for char in n:

if char.isdigit()==False:

print(char,end="")

print()

#Split Characters and Special Characters from the given string.

for char in n:

if char.isalpha():

print(char,end="")

print()

for char in n:

if not char.isalnum():

print(char,end="")

DATE:27.01.2025

1.Keerthana has to design a method that will accept the UserName and

Password from the user and also print the same in the required format.

Input:

Enter your Name:

Enter your Department:

Enter your Password:

Re-Type your Password:

Output:

Your Encoded Name is: XXXXX – Fresher

Your Department is: XXXX

Your Password is: XXXXX

Re-Type your Password: XXXXX

Condition:

1. User Name Validation: Should not allow Numbers & any special characters

to be entered.

2. Password: Must contain at least one number and one Special character.

Should not be greater than 8 in length

3. User must be allowed to continue till he enters in the correct format'''

def details():

while True:

name=input("Enter your name:")

if not name.isalpha() or name in "!@#$%^&\*\_":

print("Username should contain only letters and should not have any special characters")

continue

dep=input("Enter your Department:")

password=input("Enter your password:")

if not any(i.isdigit() for i in password) or not any(i in "!@#$%^&\*\_" for i in

password) or len(password)<=8:

print("Password must be 8 characters or less, with at least one number and one special character. Try again")

continue

retype=input("Re-Type your Password: ")

if password!=retype:

print("Passwords does not match.Try again")

continue

encoded\_name="X"\*len(name)

encoded\_dep="X"\*len(dep)

encoded\_pass="X"\*len(password)

encoded\_retype="X"\*len(retype)

print(f"Your Encoded Name is: {encoded\_name}")

print(f"Your Department is: {encoded\_dep}")

print(f"Your Password is: {encoded\_pass}")

print(f"Re-Type your Password: {encoded\_retype}")

break

details()

2.Write a Python program to remove the first occurrence of a specified

element from an array.

Sample Output:

Original array: array('i', [1, 3, 5, 3, 7, 1, 9, 3])

Remove the first occurrence of 3 from the said array:

New array: array('i', [1, 5, 3, 7, 1, 9, 3])

n=int(input("Enter the number of items in the list"))

t=int(input("Enter the target"))

arr=[]

for i in range(n):

e=int(input())

arr.append(e)

print(arr)

for i in arr:

if i==t:

arr.remove(i)

break

print("New array:",arr)

3.Create an array with limited Input size and Print Inverse and Non Inverse

Order.

The Array is - [Face, Prep, Campus, Tech, Solutions]

Sample Output Your Inverse order Array is - [Solutions ,Tech, Campus,

Prep, Face]

Your Non - Inverse Order Array is - [Face, Prep, Campus, Tech, Solutions]

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

arr=[]

for i in range(n):

e=input()

arr.append(e)

arr.reverse()

print("Your Inverse order Array is -",arr)

arr.reverse()

print("Your Non - Inverse Order Array is -",arr)

**Date : 28-01-25 Task: OOPS**

● For all the programs given below use Classes and Object. ● Use minimum two classes and one method ● Data has to be got from the user ● Do not have any static values in your program

**Abstract Class and Method**

1. Define a method Welcome which prints

“Welcome To WTS! WewishyoutheBest!!”

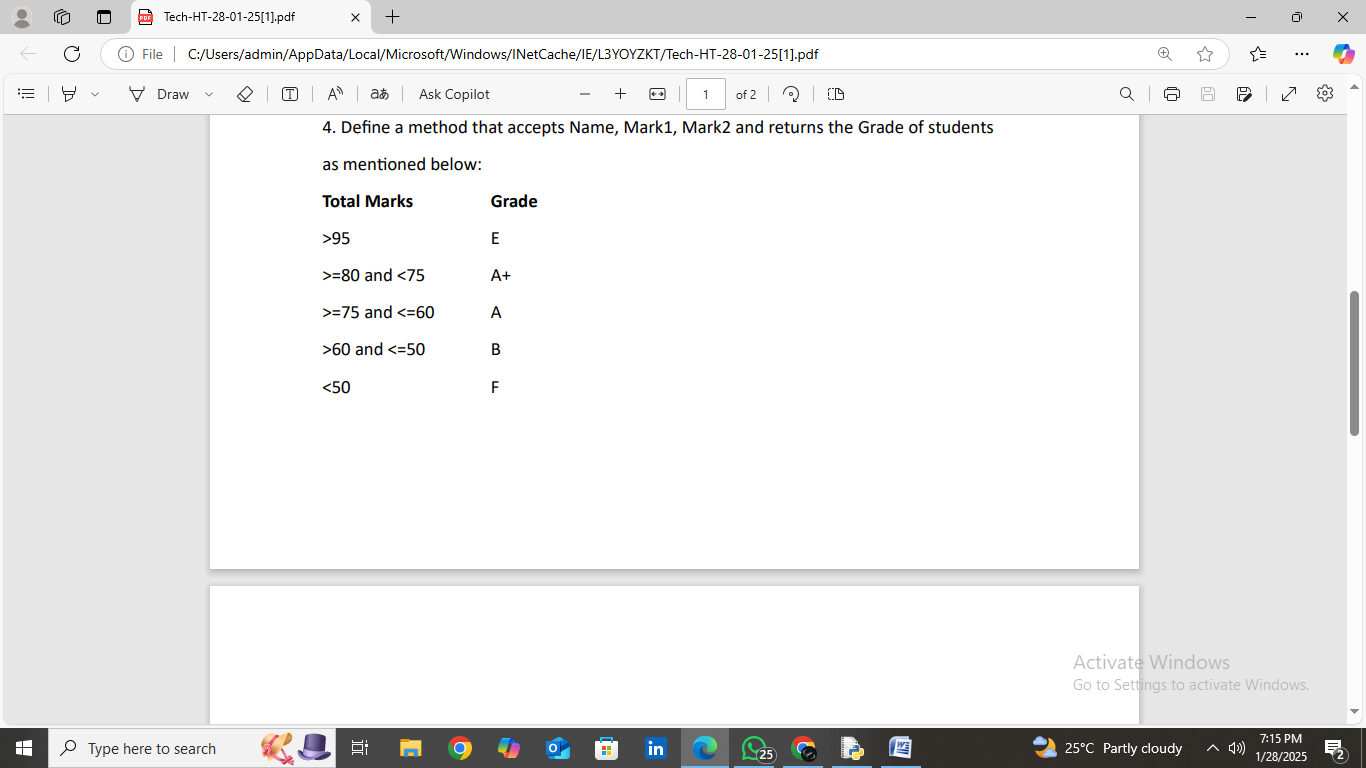
2. Define a method TestData which accepts Name and prints the belowgivenmessage Please input a name : John

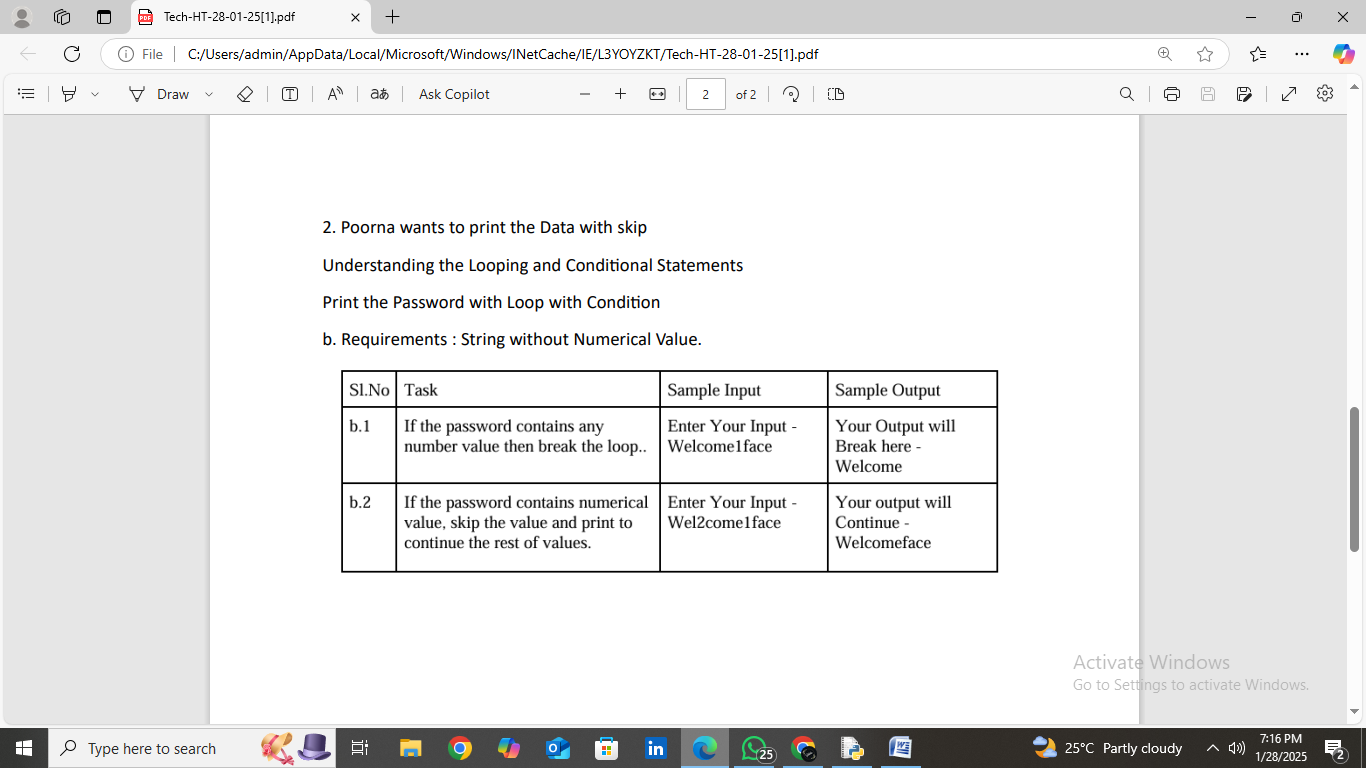
Expected Output : Welcome John! Have a nice day!

3. Define a method that accepts Name and counts the number of vowels in the names. Please input a name : John

Expected Output : Count of Vowels are : 2

a : 1 u : 1 4.





from abc import ABC, abstractmethod

class AbstractClassExample(ABC):

@abstractmethod

def welcome(self):

pass

@abstractmethod

def test\_data(self, name):

pass

@abstractmethod

def count\_vowels(self, name):

pass

@abstractmethod

def calculate\_grade(self, name, mark1, mark2):

pass

class Student(AbstractClassExample):

def welcome(self):

print("Welcome To WTS! WewishyoutheBest!")

def test\_data(self,name):

self.name=name

print(f"Welcome {self.name} ! Have a nice day!")

def count\_vowels(self,name):

vowels = 'aeiou'

count = 0

vowel\_counts = {}

for char in self.name.lower():

if char in vowels:

count += 1

vowel\_counts[char] = vowel\_counts.get(char, 0) + 1

print(f"Count of Vowels are : {count}")

for vowel, count in vowel\_counts.items():

print(f"{vowel} : {count}")

def calculate\_grade(self,name,mark1,mark2):

self.anme=name

self.mark1=mark1

self.mark2=mark2

self.tot=mark1+mark2

print("Total Marks:",{self.tot})

if self.tot>95:

print("Grade E")

elif self.tot>=80:

print("Grade A+")

elif self.tot>=75:

print("Grade A")

elif self.tot>60 and self.tot<=50:

print("Grade B")

else:

print("Grade F")

class PasswordPrinter:

def print\_password(self, password):

for char in password:

if char.isdigit():

break

else:

print(char, end='')

def print\_password\_skip\_numerics(self, password):

for char in password:

if not char.isdigit():

print(char, end='')

student = Student()

student.welcome()

name = input("Please input a name : ")

student.test\_data(name)

student.count\_vowels(name)

mark1 = float(input("Please input mark1 : "))

mark2 = float(input("Please input mark2 : "))

student.calculate\_grade(name, mark1, mark2)

password\_printer = PasswordPrinter()

password = input("Enter Your Password: ")

print("Your Output will Break here")

password\_printer.print\_password(password)

print("\n","Your output will Continue")

password\_printer.print\_password\_skip\_numerics(password)

DATE:29.01.2025

1. In a bank, different customers have savings account. Some customers may have taken a loan

from the bank. So bank always maintains information about bank depositors and borrowers.

Design a Base class Customer (name, phone-number).

Derive a class Depositor(accno, balance) from Customer.

Again, derive a class Borrower (loan-no, loan-amt) from Depositor.

Write necessary member functions to read and display the details of ‘n’ customers.

class Customer:

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

self.name=name

self.phone\_no=phone\_no

class Despostier(Customer):

def \_\_init\_\_(self,name,phone\_no,accno, balance):

super().\_\_init\_\_(name,phone\_no)

self.accno=accno

self.balance=balance

class Borrower (Despostier) :

def \_\_init\_\_(self,name,phone\_no,accno, balance,loan\_no, loan\_amt):

Despostier.\_\_init\_\_(self,name,phone\_no,accno, balance)

self.loan\_no=loan\_no

self.loan\_amt=loan\_amt

def Display(self):

print("Customer Name :",self.name)

print("Customer Phone Number: ",self.phone\_no)

print("Account Number: ",self.accno)

print("Your Balance: ",self.balance)

print("Your Loan Number: ",self.loan\_no)

print("Your Loan Amount: ",self.loan\_amt)

customers = []

n = int(input("Enter number of customer details you want to add: "))

for i in range(n):

print("\nEnter details for customer", i + 1)

na=input("Enter the name:")

ph=int(input("Enter the Phone number:"))

ac=int(input("Enter the Account Number:"))

ba=int(input("Enter the balance:"))

loan=int(input("Enter the LOan Number:"))

amt=int(input("Enter the Loan amount:"))

cus=Borrower(na,ph,ac,ba,loan,amt)

customers.append(cus)

print("\*" \* 50)

print("\nDetails Of Customers:")

for customer in customers:

customer.Display()

You are given an integer array nums. A "good subarray" is a subarray (continuous part of

nums) that contains an equal number of 0s and 1s. Find the length of the longest good

subarray

nums = list(map(int, input("Enter the binary array elements (space-separated): ").split()))

prefix\_sum = {0: -1}

sum\_so\_far = 0

max\_length = 0

for i in range(len(nums)):

if nums[i] == 0:

sum\_so\_far -= 1

else:

sum\_so\_far += 1

if sum\_so\_far in prefix\_sum:

max\_length = max(max\_length, i - prefix\_sum[sum\_so\_far])

else:

prefix\_sum[sum\_so\_far] = i # Store first occurrence of this sum

print("Length of the longest good subarray:", max\_length)

DATE:30.01.2025

'''Benedict needs to write a simple code to ask the user his name and also the number of times

he wants his name to be printed. Use class and method to validate. The format to display has

been given below

INPUT:

Enter your Name:

Enter your Password:

Enter your Email Address:

How many times to want to Print?

OUTPUT:

<<Name>> Wants to Print <<N>> Times and send Mail to <<email-id>>.

Your password <<password>> will be encrypted and will be stored.

Condition:

Include Password to your program

Name Validation: Should allow only one Number & one special character

Password: Must not contain any Number and Special characters except (#, \_ and @). Should not be

greater than 8 in length

Email Id: Should be valid format.

User must be allowed to continue till he enters in correct format'''

import re

class User:

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

self.name = name

self.password = password

self.email = email

def validate\_name(self):

if re.search(r'\d', self.name) and re.search(r'[^a-zA-Z0-9\s]', self.name):

return True

else:

return False

def validate\_password(self):

if len(self.password) <= 8 and re.search(r'[^a-zA-Z#\_@]', self.password) is None:

return True

else:

return False

def validate\_email(self):

email\_regex = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'

if re.match(email\_regex, self.email):

return True

else:

return False

def main():

while True:

name = input("Enter your Name: ")

password = input("Enter your Password: ")

email = input("Enter your Email Address: ")

user = User(name, password, email)

if user.validate\_name() and user.validate\_password() and user.validate\_email():

break

else:

print("Invalid input. Please try again.")

while True:

try:

n = int(input("How many times to want to Print? "))

break

except ValueError:

print("Invalid input. Please enter a number.")

print(f"{name} Wants to Print {n} Times and send Mail to {email}. Your password {password} will be encrypted and will be stored.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Date : 31-01-25

1.Zampa is required to handle his book stall with certain information. He is trying to formulate his

information and also automize the discount process. To do this he has to get the following information

from the customer. [Use class and methods]

INPUT:

Enter your Book Name:

Enter your Customer ID:

OUTPUT:

Price of the Book is <<XYX>> MRP

You are eligible for Discounted of <<X>>%

Your discounted Price for the Book is <<XXX>> MRP

Do you want to continue?

Note: This Output is the basic output. But will vary based on the need.

CONDITION:

1. Do the basic validation of the input before accepting it from the user

2. Based on the discount and the price calculate the discounted price to display to the user.

3. Based on the Customer ID Discount is created.

a. 1-100 – 50% discount

b. 101-300 – 30% discount

c. 301-400 – 20% discount

d. 401-500 – 10% discount

e. >500 – No Discount

4. Customer should not enter the details in the output screen

class BookStall:

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

self.books = {

"Python": 500,

"Maths": 300,

"Tamil": 200,

"English": 400,

"AI": 600

}

print(self.books)

def validate\_input(self, book\_name, customer\_id):

if book\_name not in self.books:

return "Invalid book name"

if not customer\_id.isdigit():

return "Invalid customer ID"

return None

def calculate\_discount(self, customer\_id):

customer\_id = int(customer\_id)

if customer\_id >= 1 and customer\_id <= 100:

return 50

elif customer\_id >= 101 and customer\_id <= 300:

return 30

elif customer\_id >= 301 and customer\_id <= 400:

return 20

elif customer\_id >= 401 and customer\_id <= 500:

return 10

else:

return 0

def calculate\_discounted\_price(self, book\_price, discount):

return book\_price - (book\_price \* discount / 100)

book\_stall = BookStall()

while True:

book\_name = input("Enter your Book Name: ")

customer\_id = input("Enter your Customer ID: ")

error = book\_stall.validate\_input(book\_name, customer\_id)

if error:

print(error)

continue

book\_price = book\_stall.books[book\_name]

discount = book\_stall.calculate\_discount(customer\_id)

discounted\_price = book\_stall.calculate\_discounted\_price(book\_price, discount)

print(f"Price of the Book is {book\_price} MRP")

print(f"You are eligible for Discount of {discount}%")

print(f"Your discounted Price for the Book is {discounted\_price} MRP")

choice = input("Do you want to continue? (yes/no): ")

if choice.lower() != "yes":

break

DATE:03.02.2025

You are a developer at a school, and you have been given a dictionary containing students' names

and their marks in a test. [Use lambda function and methods for each task]

Your task is to:

1. Sort students by marks in ascending order.

2. Sort students by marks in descending order.

3. Find the top 3 students with the highest marks.

4. Sort students by name alphabeticall

class Detail:

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

self.students = {

"Rickshi": 85,

"Mirthula": 92,

"Monash": 78,

"Dev": 88,

"Shreeman": 95,

"Vicky": 80,

"Dharshan": 90,

}

def sorted\_students\_asc(self):

return dict(sorted(self.students.items(), key=lambda item: item[1]))

def sorted\_students\_desc(self):

return dict(sorted(self.students.items(), key=lambda item: item[1], reverse=True))

def top\_students(self):

return dict(sorted(self.students.items(), key=lambda item: item[1], reverse=True)[:3])

def sorted\_students\_alpha(self):

return dict(sorted(self.students.items()))

def Display(self):

print("Students sorted by marks in ascending order:")

for student, marks in self.sorted\_students\_asc().items():

print(f"{student}: {marks}")

print("\nStudents sorted by marks in descending order:")

for student, marks in self.sorted\_students\_desc().items():

print(f"{student}: {marks}")

print("\nTop 3 students with the highest marks:")

for student, marks in self.top\_students().items():

print(f"{student}: {marks}")

print("\nStudents sorted by name alphabetically:")

for student, marks in self.sorted\_students\_alpha().items():

print(f"{student}: {marks}")

c = Detail()

c.Display()

Sorting a List of Tuples Using Lambda

Scenario

You are working as a data analyst for a sports organization. You have a list of tuples where each tuple

represents a player's name and their total number of goals scored in a football season.

Your task is to:

1. Sort the players by goals scored in ascending order.

2. Sort the players by goals scored in descending order.

3. Find the top 3 goal scorers.

4. Sort players by their names alphabetically.

5. Find employees earning more than $5000

class Sports:

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

self.players = [

("Pele", 67),

("Lionel Messi", 45),

("Crisitna Ronaldo", 80),

("Kylian", 65),

("Neymar", 89),

("Erling Haaland", 56),

("Hazard", 26),

("Suarez", 10),

("Griezmann", 21)

]

def sorted\_players\_asc(self):

return sorted(self.players, key=lambda x: x[1])

def sorted\_players\_desc(self):

return sorted(self.players, key=lambda x: x[1], reverse=True)

def top\_scorers(self):

return sorted(self.players, key=lambda x: x[1], reverse=True)[:3]

def sorted\_players\_alpha(self):

return sorted(self.players, key=lambda x: x[0])

def display(self):

print("Players sorted by goals scored in ascending order:")

for player in self.sorted\_players\_asc():

print(player)

print("\nPlayers sorted by goals scored in descending order:")

for player in self.sorted\_players\_desc():

print(player)

print("\nTop 3 goal scorers:")

for player in self.top\_scorers():

print(player)

print("\nPlayers sorted by their names alphabetically:")

for player in self.sorted\_players\_alpha():

print(player)

s = Sports()

s.display()

DATE:04.02.2025

'''Given a string s consisting of words and spaces, return the length of the last word in the

string.A word is a maximal substring consisting of non-space characters only.

Example 1:

Input: s = "Hello World"

Output: 5

Explanation: The last word is "World" with length 5.'''

def length\_of\_last\_word(s):

words = s.split()

if not words:

return 0

return len(words[-1])

# Test the function

s=input("Enter the String:")

print(length\_of\_last\_word(s))

'''Given two binary strings a and b, return their sum as a binary string.

Example 1:

Input: a = "11", b = "1"

Output: "100"

Example 2:

Input: a = "1010", b = "1011"

Output: "10101'''

a=input("Enter the number:")

b=input("Enter the number:")

print(bin(int(a,2)+int(b,2))[2:])

#or

a=input("Enter the numebr:")

b=input("Enetr te number:")

sum\_ab=int(a,2)+int(b,2)

print(sum\_ab)

Date : 5-2-25

1. Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Example 1:

Input: strs = ["flower","flow","flight"]

Output: "fl"

Example 2:

Input: strs = ["dog","racecar","car"]

Output: ""

Explanation: There is no common prefix among the input strings.

def common\_prefix(strs):

if not strs:

return ""

shortest\_str = min(strs, key=len) #fly

for i, char in enumerate(shortest\_str): #0,f 1,l 2,y

for other\_str in strs: #fly-->f flower-->f flew-->f | fly-->l flower-->l flew-->l | fly-->y flower-->w flew-->e

if other\_str[i] != char: # y!=w

return shortest\_str[:i] #fl

return shortest\_str

strs=input("Enter the Elements in list:").split() #fly flower flew

print(strs)

print(common\_prefix(strs)) # Output: "fl"

Given two strings s and t, return true if s is a subsequence of t, or false otherwise.

A subsequence of a string is a new string that is formed from the original string by deleting some (can

be none) of the characters without disturbing the relative positions of the remaining characters.

(i.e., "ace" is a subsequence of "abcde" while "aec" is not).

Example 1:

Input: s = "abc", t = "ahbgdc"

Output: true

Example 2:

Input: s = "axc", t = "ahbgdc"

Output: false

def is\_subsequence(s, t): #abc aghbdec

t\_index = 0

s\_index = 0

while t\_index < len(t) and s\_index < len(s): #0<7 and 0<3| 1<7 and 1<3| 2<7 and 1<3 |3<7 and 1<3| 4<7 and 2<3| 5<7 and 2<3| 6<7 and 2<3 | 7<7 and #<3-->failed

if t[t\_index] == s[s\_index]: #a==a g!=b h!=b b==b d==c e==c c==c

s\_index += 1 #1| 2 | 3

t\_index += 1 #1 | 2 | 3 | 4 | 5 | 6

return s\_index == len(s)

s=input("Enter the substring:") #abc

t=input("Enter the main string:") #aghbdec

print(is\_subsequence(s,t)) #abc

Date : 6-02-25

1. Given a string s, return the longest palindromic substring in s.

Example 1:

Input: s = "babad"

Output: "bab"

Explanation: "aba" is also a valid answer.

Example 2:

Input: s = "cbbd"

Output: "bb"

def longest\_palindrome(s):

if not s:

return ""

longest=""

for i in range(len(s)):

for j in range(i, len(s)):

substring=s[i:j+1]

if substring==substring[::-1] and len(substring)>len(longest):

longest=substring

return longest

s=input("Enter a string: ")

print("Longest palindromic substring:",longest\_palindrome(s))

Date : 7-2-25

2. Count how many trailing zeros are in n! (n factorial).

Input: 10

Output: 2 (since 10! = 3,628,800)

import math

n=int(input("Enter a number: "))

factorial=str(math.factorial(n))

print(f"Factoral of {n} is: ",factorial)

count\_zero=0

for i in factorial:

if i=='0':

count\_zero+=1

print(f"Count of zero's in {n}! is: ",count\_zero)

1. To convert an integer to English words, we need to break it down into groups of thousands

and handle each group separately.

Sample Input: 45

Sample Output : "Forty Five"

Sample Input: 12345

Sample Output : Twelve Thousand Three Hundred Forty Five

def num\_to\_words(n):

ones=["", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine"]

teens=["Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen","Sixteen", "Seventeen", "Eighteen", "Nineteen"]

tens=["", "", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"]

if n==0:

return "Zero"

elif n<10:

return ones[n]

elif n<20:

return teens[n-10]

elif n<100:

return tens[n//10]+(" " +ones[n%10] if n%10!=0 else "")

elif n<1000:

return ones[n//100]+" Hundred"+(" " +num\_to\_words(n%100) if n%100!=0 else "")

elif n < 1000000:

return num\_to\_words(n//1000)+"Thousand"+(" "+num\_to\_words(n%1000) if n%1000!=0 else "")

else:

return "Number too large"

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

print(num\_to\_words(num))

Date : 10-2-25

1. find the Kth largest element in an unsorted list

Input:

Space separated input

K value

Sample Input:

6 2 4 5 7

3

Sampel Output:

5

arr=list(map(int, input("Enter the elements:").split()))

k=int(input("Which largest element you want:"))

arr.sort(reverse=True)

print(f"The {k} Largest Elements is {arr[k-1]}")

2. Check if a Number is a Disarium Number

A Disarium number is a number where the sum of its digits raised to their respective positions is

equal to the number itself.

Example: 135 is a Disarium number because 1^1 + 3^2 + 5^3 = 135

num=int(input()) #135

digits=list(map(int, str(num))) #[1,3,5]

disarium\_sum=sum(d \*\*(i+1) for i,d in enumerate(digits)) #1^1 + 3^2 + 5^3 = 135

if disarium\_sum == num: #135===135

print("Disarium Number")

else:

print("Not a Disarium Number")

Date : 13-2-25

1. Print the following pattern and use method to call the pattern logic.

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

def print\_pattern(n):

for i in range(1, n+1):

for j in range(1, i+1):

print(j, end=" ")

print()

n=int(input("Enter the numbers:"))

print\_pattern(n) #function call

2. A bank's IT department is working on a security feature that involves prime number

verification. They need a program that can check whether a given number is prime using a

method (function)

def is\_prime(n):

if n < 2:

return "Enter a positive number greater than 1"

elif n == 2:

return "Prime"

else:

for i in range(2,n):

if n % i == 0:

return "Not Prime"

return "Prime"

n = int(input("Enter a number: "))

print(is\_prime(n))

Date:14.02.2025

1.Imagine you are building an Employee Management System where you need to store employee

details and calculate their yearly salary.

• The \_\_init\_\_ method initializes attributes: name, emp\_id, and salary.

• display\_info() prints employee details.

• calculate\_yearly\_salary() computes the yearly salary

class Employee:

def \_\_init\_\_(self,name,emp\_id,month\_salary):

self.name=name

self.emp\_id=emp\_id

self.month\_salary=month\_salary

def display(self):

print("Employee Name:",self.name)

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

print("Month Salary:",self.month\_salary)

def calculate\_yearly(self):

self\_yearly\_salary=self.month\_salary\*12

print("Yearly Salary:",self\_yearly\_salary)

e=Employee("Rickshi",108,1000)

e.display()

e.calculate\_yearly()

2. You are developing an Employee Management System for a company where there are two types of

employees:

• Full-Time Employees who receive a fixed monthly salary.

• Part-Time Employees who are paid based on the hours they work.

Since all employees share common attributes (name, employee ID, and department), you decide to

use inheritance to create specialized classes for different types of employees.

Parent Class (Employee)

• Common attributes: name, emp\_id, department.

• display\_info() method prints basic employee details.

Child Class (FullTimeEmployee)

• Inherits from Employee and adds a salary attribute.

• calculate\_annual\_salary() calculates the yearly salary.

• display\_full\_time\_info() shows all details for a full-time employee.

Child Class (PartTimeEmployee)

• Inherits from Employee and adds hourly\_rate and hours\_worked.

• calculate\_salary() computes the total earnings.

• display\_part\_time\_info() shows all details for a part-time employee

class Employee:

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

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

self.emp\_id=int(input("Enter the employee id:"))

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

def display(self):

print("Name of Employee:",self.name)

print("Employess ID:",self.emp\_id)

print("Department:",self.department)

class FullTimeEmployee(Employee):

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

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

self.month\_salary=int(input("Enter the Monthly salary:"))

def calculate\_salary(self):

self.yearly\_salary=self.month\_salary\*12

def display\_full(self):

self.display()

self.calculate\_salary()

print("Annuval salary:",self.yearly\_salary)

print("Monthly salary:",self.month\_salary)

class PartTimeEmployee(Employee):

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

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

self.hourly\_rated=int(input("Enter the per hour rate:"))

self.hour\_worked=int(input("Enter the Hours worked:"))

def calculate(self):

self.salary=self.hourly\_rated\*self.hour\_worked

def display\_part(self):

self.display()

self.calculate()

print("Per Hour amount:",self.hourly\_rated)

print("Hours Worked:",self.hour\_worked)

print("Salary for part time:",self.salary)

print("FULL TIME EMPLOYEE")

f=FullTimeEmployee()

f.display\_full()

print("PART TIME EMPLOYEE")

p=PartTimeEmployee()

p.display\_part()

Date : 17-2-25

Write a function that takes two dictionaries and merges them into one

def merge\_dicts(dict1, dict2):

dict1.update(dict2)

return dict1

d1={"Name":"Rickshi","Age":18,"Blood-Group":"O-ve"}

d2={"Gender":"Female","Nationaly":"Indian","Qualification":"BSC CSC (AI)"}

d3 = merge\_dicts(d1, d2)

print(d3)

Write a function that takes two lists and returns a dictionary with common elements

as keys and their counts as values.

def common\_elements(list1, list2):

l1 = set(list1)

l2 = set(list2)

common = l1 & l2

return {element: list1.count(element) + list2.count(element) for element in common}

list1 = list(map(int,input("Enter the elements in the list:").split()))

list2 = list(map(int,input("Enter the elements in the list:").split()))

print(common\_elements(list1, list2))

Write a function that takes a list of (key, value) tuples and converts it into a

dictionary

def tuples\_to\_dict(tuple\_list):

result = {}

num\_tuples = int(input("Enter the number of tuples: "))

for i in range(num\_tuples):

key = input(f"Enter key for pair {i+1}: ")

value = input(f"Enter value for pair {i+1}: ")

tuple\_list.append((key, value))

for tuple in tuple\_list:

result[tuple[0]] = tuple[1]

return result

tuple\_list = []

print(tuples\_to\_dict(tuple\_list))

DATE:24/02/2025

'''Checks for twin primes (pairs of prime numbers that differ by 2) within a given range.

Sample input”

1

50

Sample Output:

Twin Primes: [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43)'''

def is\_prime(n):# Check if a number is prime.

if n < 2:

return False

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

return False

return True

def find\_twin\_primes(start, end):#FInd twin primes within a given range.

twin\_primes = []

for i in range(start, end - 1):

if is\_prime(i) and is\_prime(i + 2):

twin\_primes.append((i, i + 2))

return twin\_primes

# Sample input

start = int(input("Enter the start Range:"))

end = int(input("Enter the end Range:"))

# Find and print twin primes

twin\_primes = find\_twin\_primes(start, end)

print("Twin Primes:", twin\_primes)

'''2. Write a program that:

Takes an input range (start and end).

Finds all prime numbers within the range.

Checks if they are palindromes.

Outputs all palindromic prime numbers in the given range.

Sample Input:

Enter start: 10

Enter end: 200'''

def is\_prime(n):

if n < 2:

return False

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

return False

return True

def is\_palindrome(n):

return str(n) == str(n)[::-1]

def prime\_palindromes(start, end):

result = []

for num in range(start, end + 1):

if is\_prime(num) and is\_palindrome(num):

result.append(num)

return result

start = int(input("Enter the start Range:"))

end = int(input("Enter the end Range:"))

palindrome\_primes = prime\_palindromes(start, end)

print("Prime Palindromes in the given range:", palindrome\_primes)

Date: 25-2-25

1. Print the following pattern using python

Sample Output:

1 2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9

def generate\_spiral\_pattern(n):

matrix = [[0] \* n for \_ in range(n)]

num = 1

top, left = 0, 0

bottom, right = n - 1, n - 1

while num <= n \* n:

for i in range(left, right + 1):

matrix[top][i] = num

num += 1

top += 1

for i in range(top, bottom + 1):

matrix[i][right] = num

num += 1

right -= 1

# Fill bottom row

for i in range(right, left - 1, -1):

matrix[bottom][i] = num

num += 1

bottom -= 1

for i in range(bottom, top - 1, -1):

matrix[i][left] = num

num += 1

left += 1

for row in matrix:

print(" ".join(map(str, row)))

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

generate\_spiral\_pattern(n)

2. The Collatz Conjecture

For any positive integer n:

1. If n is even, divide it by 2 → n = n / 2.

2. If n is odd, multiply it by 3 and add 1 → n = 3n + 1.

3. Repeat until n becomes 1.

Input: 12

Output: 12 steps to reach 1

\*Example 1: n = 10\*

1. n is even (10), so divide by 2: n = 5

2. n is odd (5), so multiply by 3 and add 1: n = 16

3. n is even (16), so divide by 2: n = 8

4. n is even (8), so divide by 2: n = 4

5. n is even (4), so divide by 2: n = 2

6. n is even (2), so divide by 2: n = 1

total steps 6

def collatz\_steps(n):

steps = 0

while n != 1:

if n % 2 == 0:

n = n // 2

else:

n = 3 \* n + 1

steps += 1

return steps

# Example input

n = int(input())

print(f"{collatz\_steps(n)} steps to reach 1.")

DATE:26/02/2025

1. Given a string containing multiple words, write a function to reverse every alternate

word in the sentence while keeping the others unchanged.

Constraints:

• Words are separated by a single space.

• Punctuation should be preserved in its place.

• The first word remains unchanged, the second word is reversed, the third word

remains unchanged, and so on.

Sample Input and Output:

Hello world this is Python

Hello dlrow this si Python

def reverse\_words(sentence):

words = sentence.split()

for i in range(len(words)):

if i % 2 != 0:

words[i] = words[i][::-1]

return ' '.join(words)

sentence =input()

print(reverse\_words(sentence))

Date:27.02.2025

You are developing a vehicle management system for a transportation company. The system

should handle different types of vehicles, such as Cars and Bikes. Each vehicle has a method

called fuelType() that specifies the type of fuel it uses.

However, different vehicles use different fuels, so you need to

implement method overriding to ensure each vehicle class

provides its own fuel type.

Task:

• Create a base class Vehicle with a method fuelType() that

returns "General Fuel".

• Create two subclasses Car and Bike, both inheriting from

Vehicle.

• Override the fuelType() method in each subclass:

• The Car class should return "Petrol or Diesel".

• The Bike class should return "Petrol".

• Create objects of Car and Bike and call their fuelType()

methods to demonstrate method overriding.

class Vehicle:

def fuelType(self):

return "General Fuel"

class Car(Vehicle):

def fuelType(self):

return "Petrol or Diesel"

class Bike(Vehicle):

def fuelType(self):

return "Petrol"

car= Car()

bike = Bike()

print("Car Fuel Type:", car.fuelType())

print("Bike Fuel Type:", bike.fuelType())

Date:28.02.2025

Write a Python program that takes an integer input N and prints N rows of a pattern where:

1. Each row starts with an increasing character from 'a' onward.

2. The row consists of two alternating characters repeated twice (e.g., "abab", "bcbc").

3. The number of rows printed should be equal to the given input N.

Sample Input:

Enter a number: 3

Sample Output:

abab

bcbc

cdcd

def print\_pattern(N):

for i in range(N):

first\_char = chr(ord('a') + i) # First character in the row

second\_char = chr(ord('a') + i + 1) # Second character in the row

row\_pattern = (first\_char + second\_char) \* 2 # Repeat "xy" twice

print(row\_pattern)

# Take input from the user

N = int(input("Enter a number: "))

print\_pattern(N)

Write a Python program that takes an integer input N and prints N rows of a pattern where:

1. Each row starts with an increasing character from 'a' onward.

2. The row consists of the first character repeated twice, followed by the next character

repeated twice (e.g., "aabb", "bbcc").

3. The number of rows printed should be equal to the given input N.

Sample Input:

Enter a number: 3

Sample Output:

aabb

bbcc

ccdd

def print\_pattern(N):

for i in range(N):

first\_char = chr(ord('a') + i) # First character in the row

second\_char = chr(ord('a') + i + 1) # Second character in the row

row\_pattern = first\_char \* 2 + second\_char \* 2 # "aabb", "bbcc", etc.

print(row\_pattern)

# Take input from the user

N = int(input("Enter a number: "))

print\_pattern(N)