'''import pandas as pd

import matplotlib.pyplot as plt

# Load the Bank Marketing dataset

df = pd.read\_csv("BankMarketing.csv")

# Calculate the distribution of job roles

job\_distribution = df['job'].value\_counts()

# Plot the bar chart

plt.figure(figsize=(10, 6))

job\_distribution.plot(kind='bar', color='skyblue')

plt.title('Distribution of Job Roles in Bank Marketing Dataset')

plt.xlabel('Job Roles')

plt.ylabel('Count')

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()  # Ensures the labels fit properly

plt.show()

'''

import pandas as pd

import matplotlib.pyplot as plt

# Load the Bank Marketing dataset

df = pd.read\_csv("bank\_marketing.csv")

# Plot the histogram for age distribution

plt.figure(figsize=(10, 6))

plt.hist(df['age'], bins=20, color='lightcoral', edgecolor='black')

plt.title('Age Distribution in Bank Marketing Dataset')

plt.xlabel('Age')

plt.ylabel('Count')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

# Load the Bank Marketing dataset

df = pd.read\_csv("BankMarketing.csv")

# Create a scatter plot for account balance vs last contact duration

plt.figure(figsize=(10, 6))

plt.scatter(df['balance'], df['duration'], color='green', alpha=0.5)

plt.title('Scatter Plot: Account Balance vs Last Contact Duration')

plt.xlabel('Account Balance')

plt.ylabel('Last Contact Duration (seconds)')

plt.grid(True, linestyle='--', alpha=0.7)

plt.show()

'''import pandas as pd

df = pd.read\_csv("bank\_marketing.csv")

df = df.drop("Unnamed: 0", axis=1, errors="ignore")

df = df.dropna()

df = df.drop\_duplicates()

shape\_after\_operations = df.shape

print(f"Shape of data after dropping 'Unnamed: 0', missing values, and duplicates: {shape\_after\_operations}")

average\_age\_subscribed = df[df['deposit'] == 'yes']['age'].mean()

print(f"Average age of clients who have subscribed to a deposit: {average\_age\_subscribed:.2f}")

def divide(a, b):

    assert b != 0, "Cannot divide by zero"

    return a / b

numerator = 10

denominator = 0

try:

    result = divide(numerator, denominator)

    print(f"Result: {result}")

except AssertionError as e:

    print(f"AssertionError: {e}")

class ZeroProductError(Exception):

    def \_\_init\_\_(self, message="The product of (b \* d) is zero."):

        self.message = message

        super().\_\_init\_\_(self.message)

def calculate\_product(b, d):

    result = b \* d

    if result == 0:

        raise ZeroProductError()

    return result

try:

    b\_value = 0

    d\_value = 5

    product\_result = calculate\_product(b\_value, d\_value)

    print(f"The product of ({b\_value} \* {d\_value}) is: {product\_result}")

except ZeroProductError as e:

    print(f"Error: {e}")

class MarksExceedError(Exception):

    def \_\_init\_\_(self, message="Marks cannot exceed above 100"):

        self.message = message

        super().\_\_init\_\_(self.message)

def calmarks(math, python):

    total\_marks = math + python

    if total\_marks > 100:

        raise MarksExceedError()

    return total\_marks

try:

    math = 90

    python = 90

    res = calmarks(math, python)

    print(f"Total Marks: {res}")

except MarksExceedError as e:

    print(f"ERROR!! {e}")

class BankAccount:

    def \_\_init\_\_(self, account\_holder, initial\_balance=0, interest\_rate=0.01):

        self.account\_holder = account\_holder

        self.balance = initial\_balance

        self.interest\_rate = interest\_rate

    def deposit(self, amount):

        if amount > 0:

            self.balance += amount

            print(f"Deposit of ${amount} successful.")

        else:

            print("Invalid deposit amount. Please enter a positive value.")

    def withdraw(self, amount):

        if amount > 0 and amount <= self.balance:

            self.balance -= amount

            print(f"Withdrawal of ${amount} successful.")

        else:

            print("Invalid withdrawal amount or insufficient funds.")

    def compute\_interest(self):

        interest\_earned = self.balance \* self.interest\_rate

        self.balance += interest\_earned

        print(f"Interest of ${interest\_earned:.2f} earned and added to the balance.")

    def display\_balance(self):

        print(f"Account Holder: {self.account\_holder}")

        print(f"Current Balance: ${self.balance:.2f}")

account\_holder\_name = input("Enter the account holder's name: ")

initial\_balance = float(input("Enter the initial balance: "))

interest\_rate = float(input("Enter the interest rate (e.g., 0.01 for 1%): "))

account1 = BankAccount(account\_holder\_name, initial\_balance, interest\_rate)

while True:

    print("\nOptions:")

    print("1. Deposit")

    print("2. Withdraw")

    print("3. Compute Interest")

    print("4. Display Balance")

    print("5. Exit")

    choice = input("Enter your choice (1-5): ")

    if choice == '1':

        amount = float(input("Enter the deposit amount: "))

        account1.deposit(amount)

    elif choice == '2':

        amount = float(input("Enter the withdrawal amount: "))

        account1.withdraw(amount)

    elif choice == '3':

        account1.compute\_interest()

    elif choice == '4':

        account1.display\_balance()

    elif choice == '5':

        print("Exiting the banking application.")

        break

    else:

        print("Invalid choice. Please enter a number between 1 and 5.")

def fact(n):

    if n==0 or n==1:

        return 1

    else:

        return n\*fact(n-1)

num=int(input("enter the factorial: "))

res=fact(num)

print(f"answer: {res}")

largest = lambda x, y: x if x > y else y

# Example usage:

num1 = float(input("Enter the first number: "))

num2 = float(input("Enter the second number: "))

result = largest(num1, num2)

print(f"The largest of {num1} and {num2} is: {result}")

def calculate\_arithmetic\_mean(\*values):

    if not values:

        return None

    return sum(values) / len(values)

num1 = float(input("Enter the first number: "))

num2 = float(input("Enter the second number: "))

num3 = float(input("Enter the third number: "))

mean\_result = calculate\_arithmetic\_mean(num1, num2, num3)

print(f"The arithmetic mean is: {mean\_result}")

import numpy as np

array2d=np.array([[1,2,3],

[4,5,6],

[7,8,9]])

cols=np.sum(array2d,axis=0)

print(array2d)

print(cols)

#axis=1->rowwise

#axis=0->colwise

import numpy as np

array2d=np.array([

    [10,20,30],

    [40,50,60],

    [70,80,90]

])

array1d=array2d.flatten()

print(array1d)

import numpy as np

array = np.array([1, 2, 3, 2, 4, 2, 5, 2, 6, 2, 7, 8, 9, 2, 1, 2, 3, 2, 4, 2])

unique\_values, counts = np.unique(array, return\_counts=True)

most\_frequent\_index = np.argmax(counts)

most\_frequent\_value = unique\_values[most\_frequent\_index]

print("Original array:")

print(array)

print("\nMost frequent value:", most\_frequent\_value)

\*\*\*

def add\_matrices(matrix\_a, matrix\_b):

    if len(matrix\_a) != len(matrix\_b) or len(matrix\_a[0]) != len(matrix\_b[0]):

        raise ValueError("Matrices must have the same dimensions for addition.")

    result = []

    for i in range(len(matrix\_a)):

        row = [matrix\_a[i][j] + matrix\_b[i][j] for j in range(len(matrix\_a[0]))]

        result.append(row)

    return result

def multiply\_matrices(matrix\_a, matrix\_b):

    if len(matrix\_a[0]) != len(matrix\_b):

        raise ValueError("Number of columns in matrix A must be equal to the number of rows in matrix B for multiplication.")

    result = []

    for i in range(len(matrix\_a)):

        row = []

        for j in range(len(matrix\_b[0])):

            element\_sum = sum(matrix\_a[i][k] \* matrix\_b[k][j] for k in range(len(matrix\_a[0])))

            row.append(element\_sum)

        result.append(row)

    return result

# Example matrices

matrix\_a = [[1, 2], [3, 4]]

matrix\_b = [[5, 6], [7, 8]]

# Add matrices

matrix\_sum = add\_matrices(matrix\_a, matrix\_b)

# Multiply matrices element-wise

matrix\_product = multiply\_matrices(matrix\_a, matrix\_b)

# Display the results

print("Matrix A:")

for row in matrix\_a:

    print(row)

print("\nMatrix B:")

for row in matrix\_b:

    print(row)

print("\nMatrix Sum (A + B):")

for row in matrix\_sum:

    print(row)

print("\nMatrix Product (A \* B):")

for row in matrix\_product:

    print(row)

import random

# Randomly select 10 integers from the range 100 to 200

random\_integers = random.sample(range(100, 201), 10)

# Find the smallest among all

smallest\_integer = min(random\_integers)

# Display the selected integers and the smallest value

print("Randomly Selected Integers:")

print(random\_integers)

print("\nThe Smallest Integer:", smallest\_integer)

# Create a dictionary of 5 countries with currency details

countries\_dict = {

    'USA': {'Currency': 'US Dollar', 'Symbol': '$'},

    'UK': {'Currency': 'British Pound', 'Symbol': '£'},

    'Japan': {'Currency': 'Japanese Yen', 'Symbol': '¥'},

    'India': {'Currency': 'Indian Rupee', 'Symbol': '₹'},

    'Australia': {'Currency': 'Australian Dollar', 'Symbol': 'A$'}

}

# Display the dictionary

print("Countries and Currency Details:")

for country, details in countries\_dict.items():

    print(f"{country}: {details['Currency']} ({details['Symbol']})")

# Take a sentence as input

sentence = input("Enter a sentence: ")

# Replace each blank with a hyphen

modified\_sentence = sentence.replace(' ', '-')

# Print the modified sentence

print("Modified Sentence:")

print(modified\_sentence)

# Solve the quadratic equation ax\*\*2 + bx + c = 0

# import complex math module

import cmath

a = 1

b = 5

c = 6

# calculate the discriminant

d = (b\*\*2) - (4\*a\*c)

# find two solutions

sol1 = (-b-cmath.sqrt(d))/(2\*a)

sol2 = (-b+cmath.sqrt(d))/(2\*a)

print('The solution are {0} and {1}'.format(sol1,sol2))

def generate\_fibonacci(n):

    fibonacci\_sequence = [0, 1]

    while len(fibonacci\_sequence) < n:

        next\_number = fibonacci\_sequence[-1] + fibonacci\_sequence[-2]

        fibonacci\_sequence.append(next\_number)

    return fibonacci\_sequence

# Specify the number of Fibonacci numbers to print

num\_fibonacci = int(input("Enter the number of Fibonacci numbers to print: "))

# Print the specified number of Fibonacci numbers

print("Fibonacci Numbers:")

print(generate\_fibonacci(num\_fibonacci))

\*\*\*\*importrandomnumbers=[random.randint(100,200) fori inrange(10)]

small =min(numbers)

print("Following numbers are:")print(numbers)\*\*\*

\*\*\*currency={"India":"Rupee" , "Europe":"Euro","America":"Dollars","UK":"Pounds"}

print("Dictionary is :")

print(currency)print(currency["India"])\*\*\*\*

import numpy as np

# Create a NumPy array filled with all ones

ones\_array = np.ones((3, 3))  # Specify the shape (3x3 in this example)

# Display the array

print("NumPy Array Filled with All Ones:")

print(ones\_array)

import numpy as np

# Create a NumPy array

matrix = np.array([

    [1, 2, 3],

    [4, 5, 6],

    [7, 8, 9]

])

# Specify a row to check

specified\_row = np.array([4, 5, 6])

# Check whether the array contains the specified row

contains\_specified\_row = np.any(np.all(matrix == specified\_row, axis=1))

# Display the result

print("Matrix:")

print(matrix)

print("\nDoes the Matrix Contain the Specified Row?", contains\_specified\_row)

'''

players = ["Neymar", "Ronaldo", "Messi" , "Mbappe", "Salah", "Kane", "Lewandowski", "De Bruyne", "Modric", "Kante"]

print("Original List:")

print(players)

players.sort()

print("\nSorted List:")

print(players)

player\_to\_remove = "Neymar"

if player\_to\_remove in players:

    players.remove(player\_to\_remove)

    print("\nList after removing", player\_to\_remove + ":")

    print(players)

'''

def calculate\_factorial(n):

    result = 1

    for i in range(1, n + 1):

        result \*= i

    return result

# Example usage:

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

print(f"The factorial of {number} is: {calculate\_factorial(number)}")

def rev\_string(s):

    return s[::-1]

s=input("enter a string: ")

print(f"{rev\_string(s)}")

def find\_largest\_smallest(arr):

    if not arr:

        return None, None

    else:

        return max(arr), min(arr)

array = [int(x) for x in input("Enter integers: ").split(',')]

largest, smallest = find\_largest\_smallest(array)

print(f"Largest: {largest}, Smallest: {smallest}")

def is\_palindrome(input\_str):

    return input\_str == input\_str[::-1]

word = input("Enter a string: ")

if is\_palindrome(word):

    print(f"{word} is a palindrome.")

else:

    print(f"{word} is not a palindrome.")

def merge\_dictionaries(dict1, dict2):

    merged\_dict = dict1.copy()

    merged\_dict.update(dict2)

    return merged\_dict

dict1 = {'a': 1, 'b': 2}

dict2 = {'b': 3, 'c': 4}

result\_dict = merge\_dictionaries(dict1, dict2)

print(f"Merged Dictionary: {result\_dict}")

class Rectangle:

    def \_\_init\_\_(self, length, width):

        self.length = length

        self.width = width

    def calculate\_area(self):

        return self.length \* self.width

    def calculate\_perimeter(self):

        return 2 \* (self.length + self.width)

rectangle = Rectangle(4, 5)

print(f"Area: {rectangle.calculate\_area()}, Perimeter: {rectangle.calculate\_perimeter()}")

import numpy as np

matrix = np.array([[1, 2], [3, 4]])

inverse\_matrix = np.linalg.inv(matrix)

print(f"Original Matrix:\n{matrix}")

print(f"Inverse Matrix:\n{inverse\_matrix}")

--

def gcd(a, b):

    while b:

        a, b = b, a % b

    return a

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

result = gcd(num1, num2)

print(f"GCD of {num1} and {num2} is: {result}")

def sum\_of\_digits(n):

    if n == 0:

        return 0

    else:

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

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

print(f"Sum of digits: {sum\_of\_digits(number)}")--

try:

    result = 10 / 0

except (ZeroDivisionError, ArithmeticError) as e:

    print(f"An error occurred: {e}")

except Exception as e:

    print(f"Caught an exception: {e}")

import numpy as np

array = np.array([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5])

sorted\_array = np.sort(array)

print(f"Original Array: {array}")

print(f"Sorted Array: {sorted\_array}")

import pandas as pd

# Example DataFrames (Replace these with your actual DataFrames)

data1 = {'A': [1, 2, 3], 'B': [4, 5, 6]}

data2 = {'C': [7, 8, 9], 'D': [10, 11, 12]}

df1 = pd.DataFrame(data1)

df2 = pd.DataFrame(data2)

merged\_df = pd.concat([df1, df2], axis=1)

print("Merged DataFrame:")

print(merged\_df)

'''

def sum\_of\_digits(n):

    if n == 0:

        return 0

    else:

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

number = abs(int(input("Enter a number: ")))  # Use abs to handle negative numbers

print(f"Sum of digits: {sum\_of\_digits(number)}")