**PYTHON PROGRAM USING CONTROL STRUCTURES AND ARRAYS**

**1. Implement a python program to find the first largest and second largest numbers in an Array. Note: should not use any built-in sorting functions or libraries.**

def find\_first\_and\_second\_largest(arr):

if len(arr) < 2:

print("Array should contain at least two elements")

return

first\_largest = second\_largest = float('-inf')

for num in arr:

if num > first\_largest:

second\_largest = first\_largest

first\_largest = num

elif num > second\_largest and num < first\_largest:

second\_largest = num

if second\_largest == float('-inf'):

print("There is no second largest element.")

else:

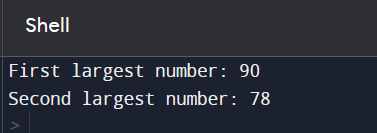
print("First largest number:", first\_largest)

print("Second largest number:", second\_largest)

# Test the function

arr = [12, 45, 1, 78, 90, 45, 23]

find\_first\_and\_second\_largest(arr)



**2. Write a Python program to calculate the sum of even numbers and the sum of odd numbers in an array.**

def sum\_even\_and\_odd(arr):

even\_sum = 0

odd\_sum = 0

for num in arr:

if num % 2 == 0:

even\_sum += num

else:

odd\_sum += num

return even\_sum, odd\_sum

# Input from the user

try:

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

arr = []

for i in range(n):

arr.append(int(input(f"Enter element {i + 1}: ")))

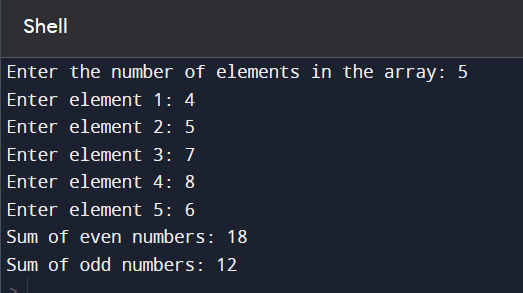
even\_sum, odd\_sum = sum\_even\_and\_odd(arr)

print(f"Sum of even numbers: {even\_sum}")

print(f"Sum of odd numbers: {odd\_sum}")

except ValueError:

print("Invalid input. Please enter valid integers.")



**3. Write a python program to count the Occurrences of a Specific Element in an Array.**

def count\_occurrences(arr, target):

count = 0

for element in arr:

if element == target:

count += 1

return count

# Input from the user

try:

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

arr = []

for i in range(n):

arr.append(int(input(f"Enter element {i + 1}: ")))

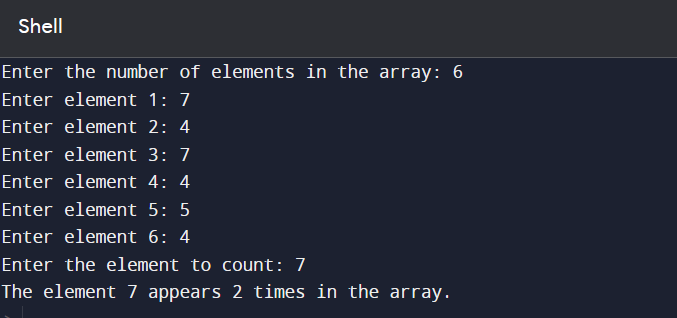
target = int(input("Enter the element to count: "))

occurrences = count\_occurrences(arr, target)

print(f"The element {target} appears {occurrences} times in the array.")

except ValueError:

print("Invalid input. Please enter valid integers.")



**4. Write a Python program that takes a sentence as input and identifies and prints all the palindromic words in the sentence. Use an array to store the palindromic words.**

def is\_palindrome(word):

return word == word[::-1]

def find\_palindromic\_words(sentence):

words = sentence.split()

palindromic\_words = []

for word in words:

if is\_palindrome(word):

palindromic\_words.append(word)

return palindromic\_words

# Input from the user

sentence = input("Enter a sentence: ")

palindromic\_words = find\_palindromic\_words(sentence)

if palindromic\_words:

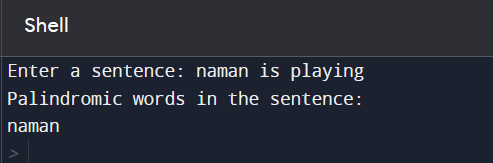
print("Palindromic words in the sentence:")

for word in palindromic\_words:

print(word)

else:

print("No palindromic words found in the sentence.")



**5. Write a Python program that takes a list of numbers and removes all duplicates from the list, preserving the original order of elements.**

def remove\_duplicates(input\_list):

unique\_list = []

for item in input\_list:

if item not in unique\_list:

unique\_list.append(item)

return unique\_list

# Input from the user

try:

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

num\_list = []

for i in range(n):

num = int(input(f"Enter element {i + 1}: "))

num\_list.append(num)

unique\_list = remove\_duplicates(num\_list)

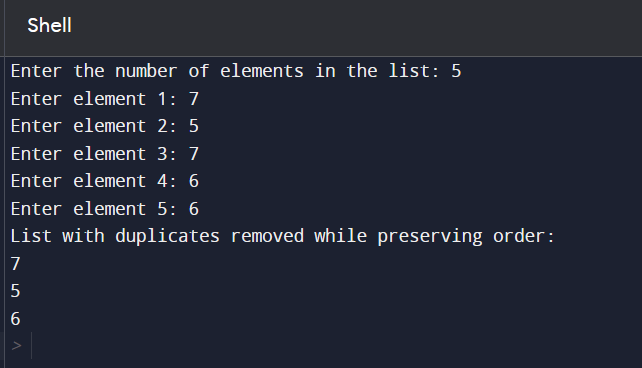
print("List with duplicates removed while preserving order:")

for item in unique\_list:

print(item)

except ValueError:

print("Invalid input. Please enter valid integers.")



**6. Write a Python program that performs matrix multiplication. Ask the user to input two matrices as lists of lists (2D arrays) and then multiply them if possible. Make sure to check if the matrices are compatible for multiplication and handle errors gracefully.**

def matrix\_multiplication(mat1, mat2):

if len(mat1[0]) != len(mat2):

raise ValueError("Matrix dimensions are not compatible for multiplication.")

result = [[0 for \_ in range(len(mat2[0]))] for \_ in range(len(mat1))]

for i in range(len(mat1)):

for j in range(len(mat2[0])):

for k in range(len(mat2)):

result[i][j] += mat1[i][k] \* mat2[k][j]

return result

# Input from the user for the first matrix

try:

n1 = int(input("Enter the number of rows for the first matrix: "))

m1 = int(input("Enter the number of columns for the first matrix: "))

mat1 = []

for i in range(n1):

row = []

for j in range(m1):

element = int(input(f"Enter element at position ({i+1}, {j+1}): "))

row.append(element)

mat1.append(row)

# Input from the user for the second matrix

n2 = int(input("Enter the number of rows for the second matrix: "))

m2 = int(input("Enter the number of columns for the second matrix: "))

mat2 = []

for i in range(n2):

row = []

for j in range(m2):

element = int(input(f"Enter element at position ({i+1}, {j+1}): "))

row.append(element)

mat2.append(row)

if m1 != n2:

print("Matrix dimensions are not compatible for multiplication.")

else:

result\_matrix = matrix\_multiplication(mat1, mat2)

print("Matrix multiplication result:")

for row in result\_matrix:

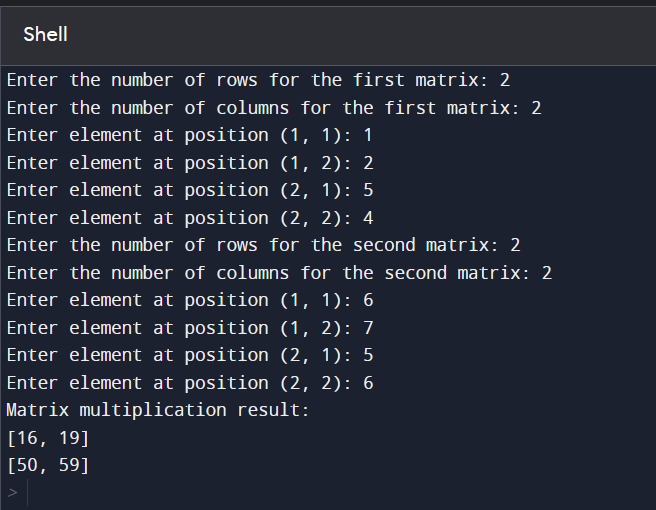
print(row)

except ValueError:

print("Invalid input. Please enter valid integers.")

except Exception as e:

print("An error occurred:", e)



**7. Write a python program to print diamond number pattern using Nested Loops.**

def print\_diamond\_pattern(n):

# Upper half of the diamond

for i in range(1, n + 1, 2):

spaces = (n - i) // 2

numbers = range(1, i + 1, 2)

row = " " \* spaces + " ".join(map(str, numbers)) + " " \* spaces

print(row)

# Lower half of the diamond

for i in range(n - 2, 0, -2):

spaces = (n - i) // 2

numbers = range(1, i + 1, 2)

row = " " \* spaces + " ".join(map(str, numbers)) + " " \* spaces

print(row)

# Input the number of rows (should be an odd number for a proper diamond)

try:

n = int(input("Enter the number of rows for the diamond pattern: "))

if n % 2 == 0:

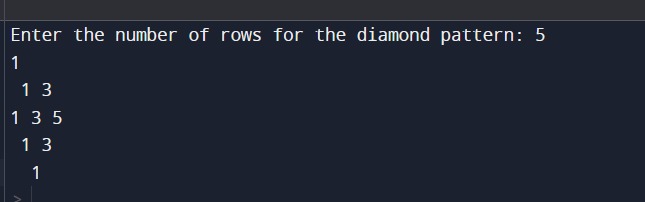
print("Please enter an odd number for a proper diamond pattern.")

else:

print\_diamond\_pattern(n)

except ValueError:

print("Invalid input. Please enter a valid integer.")



**8. Write a Python program that simulates a simple guessing game. Generate a random number and have the user guess it. Provide hints like &quot;too high&quot; or &quot;too low&quot; until they guess correctly.**

import random

def guessing\_game():

# Generate a random number between 1 and 100 (you can adjust the range)

secret\_number = random.randint(1, 10)

attempts = 0

print("Welcome to the Guessing Game! Can you guess the secret number?")

while True:

try:

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

attempts += 1

if guess < secret\_number:

print("Too low! Try again.")

elif guess > secret\_number:

print("Too high! Try again.")

else:

print(f"Congratulations! You guessed the secret number {secret\_number} in {attempts} attempts.")

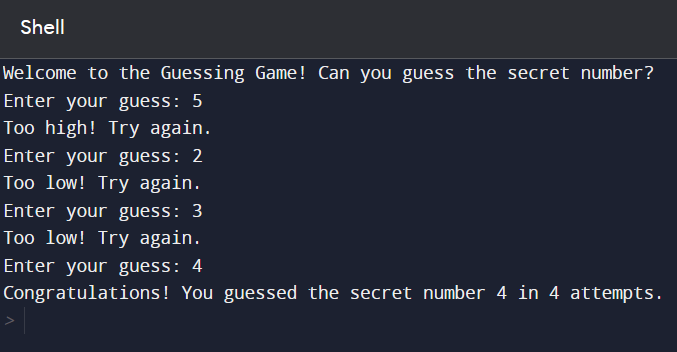
break

except ValueError:

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

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

guessing\_game()



**9. Write a Python program that checks the strength of a password entered by a user. The program should assess the password based on criteria like length, use of uppercase and lowercase letters, digits, and special characters. Use control structures and arrays to provide a detailed evaluation.**

def check\_password\_strength(password):

length\_criteria = len(password) >= 8

uppercase\_criteria = any(char.isupper() for char in password)

lowercase\_criteria = any(char.islower() for char in password)

digit\_criteria = any(char.isdigit() for char in password)

special\_char\_criteria = any(char in "!@#$%^&\*()\_+" for char in password)

criteria\_met = [length\_criteria, uppercase\_criteria, lowercase\_criteria, digit\_criteria, special\_char\_criteria]

return criteria\_met

# Input from the user

password = input("Enter a password: ")

criteria\_met = check\_password\_strength(password)

print("Password Strength Evaluation:")

print(f"1. Minimum length of 8 characters: {'Pass' if criteria\_met[0] else 'Fail'}")

print(f"2. At least one uppercase letter: {'Pass' if criteria\_met[1] else 'Fail'}")

print(f"3. At least one lowercase letter: {'Pass' if criteria\_met[2] else 'Fail'}")

print(f"4. At least one digit: {'Pass' if criteria\_met[3] else 'Fail'}")

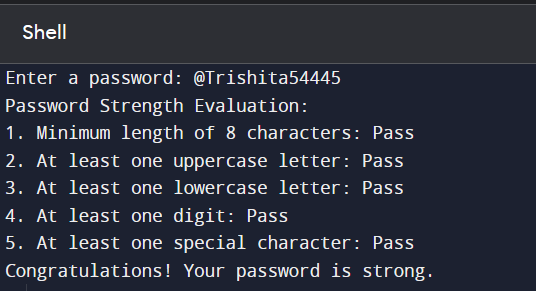
print(f"5. At least one special character: {'Pass' if criteria\_met[4] else 'Fail'}")

if all(criteria\_met):

print("Congratulations! Your password is strong.")

else:

print("Your password does not meet all criteria for a strong password.")



**10. Write a Python program that generates the Fibonacci sequence up to a specified number of terms using a loop and stores it in an array.**

def generate\_fibonacci\_sequence(n):

if n <= 0:

return []

fibonacci\_sequence = [0, 1]

while len(fibonacci\_sequence) < n:

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

fibonacci\_sequence.append(next\_term)

return fibonacci\_sequence

# Input from the user

try:

n = int(input("Enter the number of Fibonacci terms to generate: "))

if n <= 0:

print("Please enter a positive number.")

else:

fibonacci\_sequence = generate\_fibonacci\_sequence(n)

if n >= 2:

print("Fibonacci sequence:")

print(fibonacci\_sequence)

else:

print(f"Fibonacci sequence with {n} term(s): {fibonacci\_sequence[0]}")

except ValueError:

print("Invalid input. Please enter a valid integer.")

