**Q.1 Write a program for the following Conditional Statements in Python with suitable examples.**

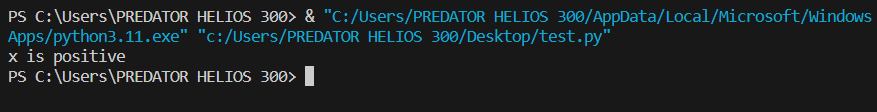
**i) if statement ii) if else statement iii) if - elif - else statement.**

i) if statement

x = 10

if x > 0:

print("x is positive")

Result:  


ii) if else statement

y = -5

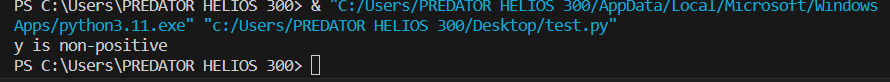
if y > 0:

print("y is positive")

else:

print("y is non-positive")

Result:



iii) if - elif - else statement.

grade = 75

if grade >= 90:

print("Grade is A")

elif grade >= 80:

print("Grade is B")

elif grade >= 70:

print("Grade is C")

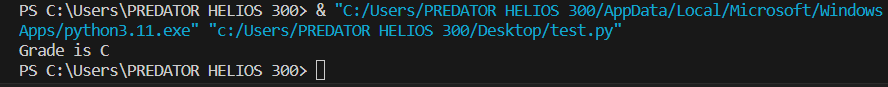
elif grade >= 60:

print("Grade is D")

else:

print("Grade is F")

Result:



Q.2 Implement the following Searching and Sorting techniques in Python by using functions. i)

Linear Search ii) Binary Search iii) Selection Sort iv) Merge Sort v) Quick Sort.

i) Linear Search

def linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

# Example usage:

arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

target = 9

index = linear\_search(arr, target)

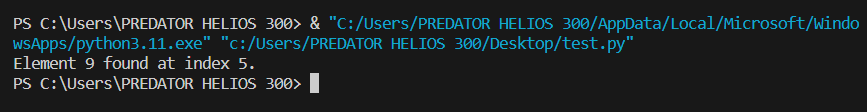
if index != -1:

print(f"Element {target} found at index {index}.")

else:

print(f"Element {target} not found.")

Result:



ii) Binary Search

def binary\_search(arr, target):

low = 0

high = len(arr) - 1

while low <= high:

mid = (low + high) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

low = mid + 1

else:

high = mid - 1

return -1

# Example usage:

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

target = 5

index = binary\_search(arr, target)

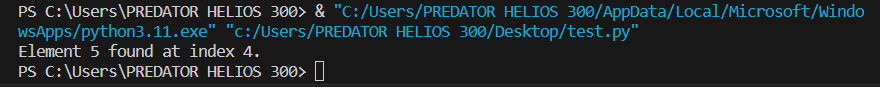
if index != -1:

print(f"Element {target} found at index {index}.")

else:

print(f"Element {target} not found.")

Result:



iii) Selection Sort

def selection\_sort(arr):

n = len(arr)

for i in range(n):

min\_index = i

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

if arr[j] < arr[min\_index]:

min\_index = j

arr[i], arr[min\_index] = arr[min\_index], arr[i]

return arr

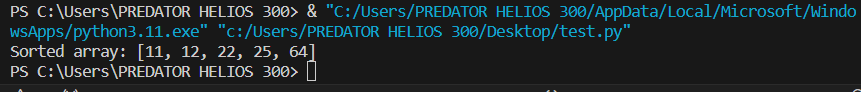
# Example usage:

arr = [64, 25, 12, 22, 11]

sorted\_arr = selection\_sort(arr)

print("Sorted array:", sorted\_arr)

Result:



iv) Merge Sort

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

left\_half = merge\_sort(left\_half)

right\_half = merge\_sort(right\_half)

return merge(left\_half, right\_half)

def merge(left, right):

merged = []

left\_idx, right\_idx = 0, 0

while left\_idx < len(left) and right\_idx < len(right):

if left[left\_idx] < right[right\_idx]:

merged.append(left[left\_idx])

left\_idx += 1

else:

merged.append(right[right\_idx])

right\_idx += 1

merged.extend(left[left\_idx:])

merged.extend(right[right\_idx:])

return merged

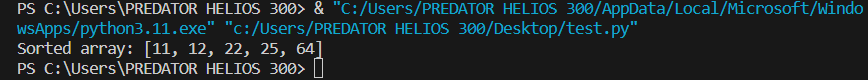
# Example usage:

arr = [64, 25, 12, 22, 11]

sorted\_arr = merge\_sort(arr)

print("Sorted array:", sorted\_arr)

Result:



v) Quick Sort.

def quick\_sort(arr):

if len(arr) <= 1:

return arr

else:

pivot = arr[0]

less\_than\_pivot = [x for x in arr[1:] if x <= pivot]

greater\_than\_pivot = [x for x in arr[1:] if x > pivot]

return quick\_sort(less\_than\_pivot) + [pivot] + quick\_sort(greater\_than\_pivot)

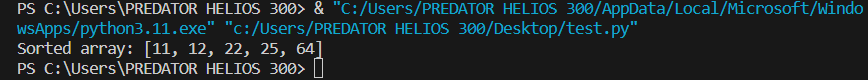
# Example usage:

arr = [64, 25, 12, 22, 11]

sorted\_arr = quick\_sort(arr)

print("Sorted array:", sorted\_arr)

Result:



Q.3 Create a class Calculator capable of finding square and cube of a number.

class Calculator:

def square(self, num):

return num \*\* 2

def cube(self, num):

return num \*\* 3

# Example usage:

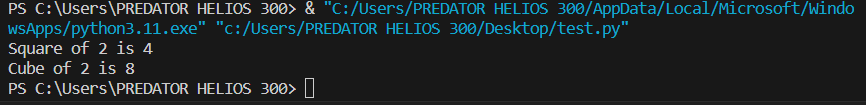
calc = Calculator()

num = 2

print("Square of", num, "is", calc.square(num))

print("Cube of", num, "is", calc.cube(num))

Result:



Q.4 Create a class programmer for storing information of few programmers working at

Microsoft.

class Programmer:

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

self.name = name

self.employee\_id = employee\_id

self.role = role

self.language = language

def display\_info(self):

print("Name:", self.name)

print("Employee ID:", self.employee\_id)

print("Role:", self.role)

print("Programming Language(s):", self.language)

print()

# Example usage:

programmer1 = Programmer("John Doe", 123456, "Software Engineer", "Python, Java")

programmer2 = Programmer("Jane Smith", 789012, "Data Scientist", "R, Python, SQL")

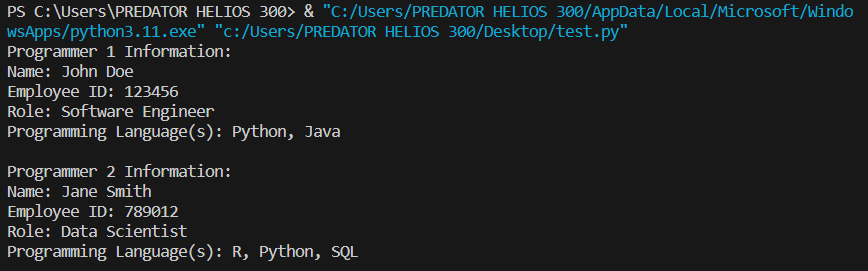
print("Programmer 1 Information:")

programmer1.display\_info()

print("Programmer 2 Information:")

programmer2.display\_info()

Result:



Q.5 Implement a Python function to perform matrix multiplication.

def matrix\_multiplication(matrix1, matrix2):

rows1 = len(matrix1)

cols1 = len(matrix1[0])

rows2 = len(matrix2)

cols2 = len(matrix2[0])

# Check if matrices can be multiplied

if cols1 != rows2:

print("Error: Matrices cannot be multiplied. Number of columns in matrix1 must equal number of rows in matrix2.")

return None

# Initialize the result matrix with zeros

result = [[0 for \_ in range(cols2)] for \_ in range(rows1)]

# Perform multiplication

for i in range(rows1):

for j in range(cols2):

for k in range(cols1):

result[i][j] += matrix1[i][k] \* matrix2[k][j]

return result

# Example usage:

matrix1 = [[1, 2, 3],

[4, 5, 6]]

matrix2 = [[7, 8],

[9, 10],

[11, 12]]

result = matrix\_multiplication(matrix1, matrix2)

if result:

print("Resultant Matrix:")

for row in result:

print(row)

Result:

