DAY – 07

1.Selection Sort

def selectionSort(array, size):

for step in range(size):

min\_idx = step

for i in range(step + 1, size):

if array[i] < array[min\_idx]:

min\_idx = i

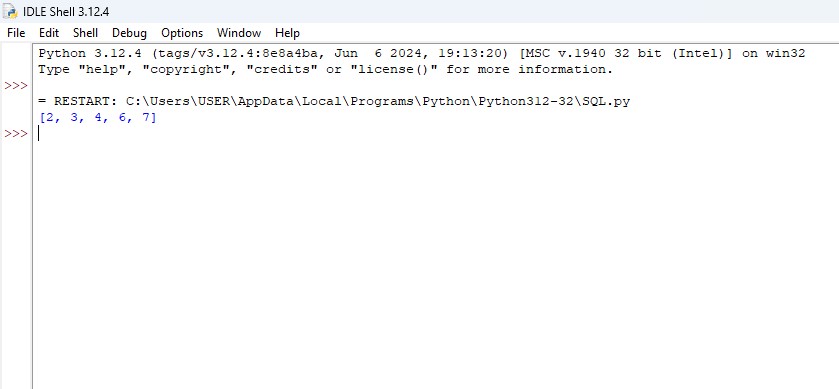
(array[step], array[min\_idx]) = (array[min\_idx], array[step])

data = [-2, 45, 0, 11, -9]

size = len(data)

selectionSort(data, size)

print('Sorted Array in Ascending Order:')

print(data)

2. Bubble Sort

def swap(a,i,j):

a[i],a[j]=a[j],a[i]

return

def bubble(a):

for i in range(len(a)):

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

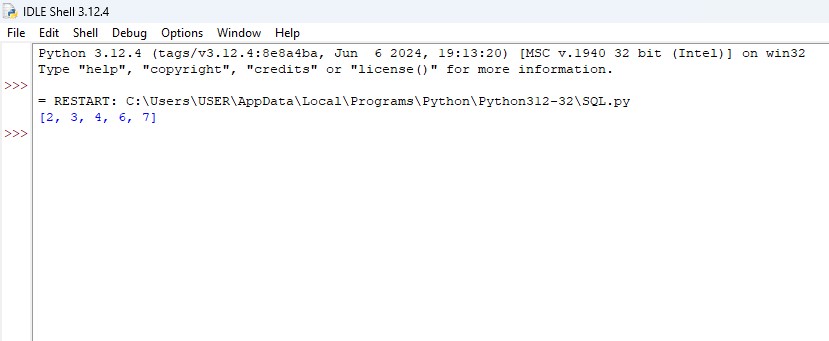
if a[i]>a[j]:

swap(a,i,j)

return a

a=[6,3,2,7,4]

print(bubble(a))



3.Insertion Sort

def insertionSort(arr):

n = len(arr)

if n <= 1:

return

for i in range(1, n):

key = arr[i]

j = i-1

while j >= 0 and key < arr[j]:

arr[j+1] = arr[j]

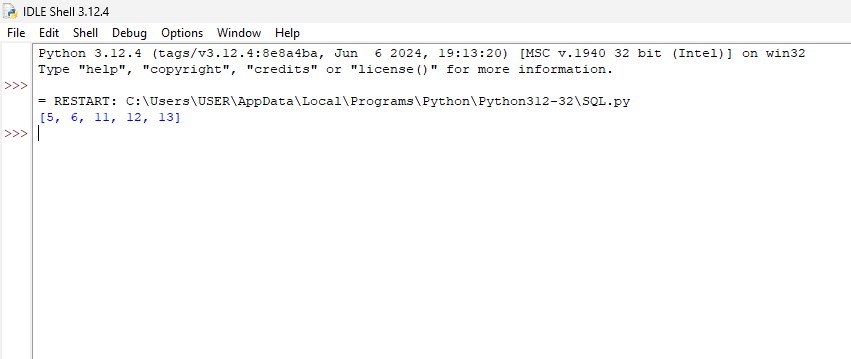
j -= 1

arr[j+1] = key

arr = [12, 11, 13, 5, 6]

insertionSort(arr)

print(arr)



4. Sequential Search

def sequential\_search(a,target):

for i in range(len(a)):

if a[i]==target:

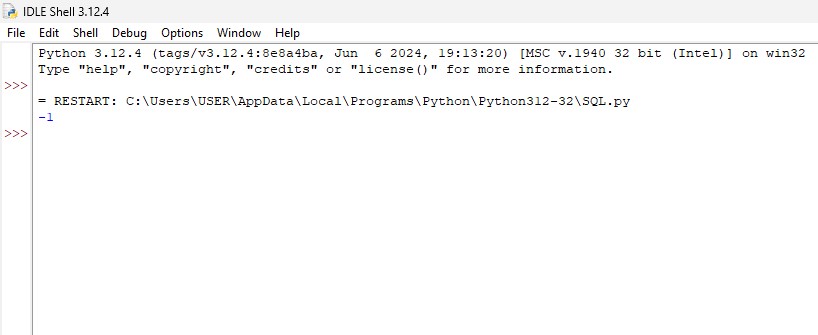
return i

return -1

a=[1,2,3,4,5,6,7]

target=9

print(sequential\_search(a,target))



5. Brute Force String Matching

def matching(s,t):

for i in range(len(s)-len(t)):

j,m=0,i

while(j!=len(t)):

if s[m]==t[j]:

j+=1

m+=1

else:

break

if j == len(t):

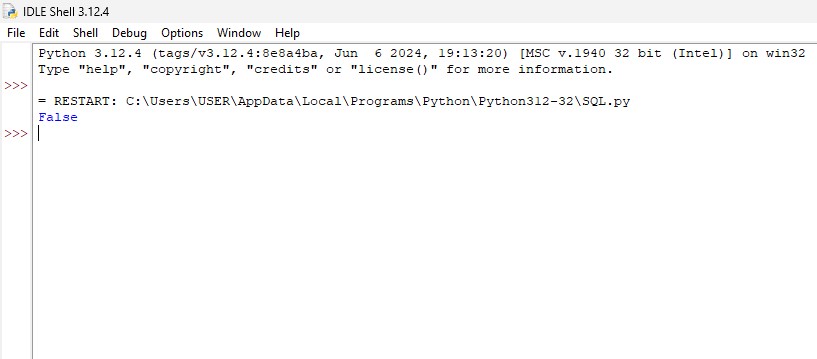
return True

return False

s="Philip"

t="ild"

print(matching(s,t))



6. Closest Pair

def closest\_pair(points):

def euclidean\_distance(p1, p2):

return ((p1[0] - p2[0]) \*\* 2 + (p2[1] - p1[1]) \*\* 2) \*\* 0.5

def closest\_pair\_recursive(points\_sorted\_by\_x, points\_sorted\_by\_y):

n = len(points\_sorted\_by\_x)

if n <= 3:

min\_dist = float('inf')

for i in range(n):

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

min\_dist = min(min\_dist, euclidean\_distance(points\_sorted\_by\_x[i], points\_sorted\_by\_x[j]))

return min\_dist

mid = n // 2

midpoint = points\_sorted\_by\_x[mid]

left\_half\_x = points\_sorted\_by\_x[:mid]

right\_half\_x = points\_sorted\_by\_x[mid:]

left\_half\_y = list(filter(lambda p: p[0] <= midpoint[0], points\_sorted\_by\_y))

right\_half\_y = list(filter(lambda p: p[0] > midpoint[0], points\_sorted\_by\_y))

d1 = closest\_pair\_recursive(left\_half\_x, left\_half\_y)

d2 = closest\_pair\_recursive(right\_half\_x, right\_half\_y)

d = min(d1, d2)

strip = [p for p in points\_sorted\_by\_y if abs(p[0] - midpoint[0]) < d]

strip\_len = len(strip)

for i in range(strip\_len):

for j in range(i + 1, min(i + 7, strip\_len)):

d = min(d, euclidean\_distance(strip[i], strip[j]))

return d

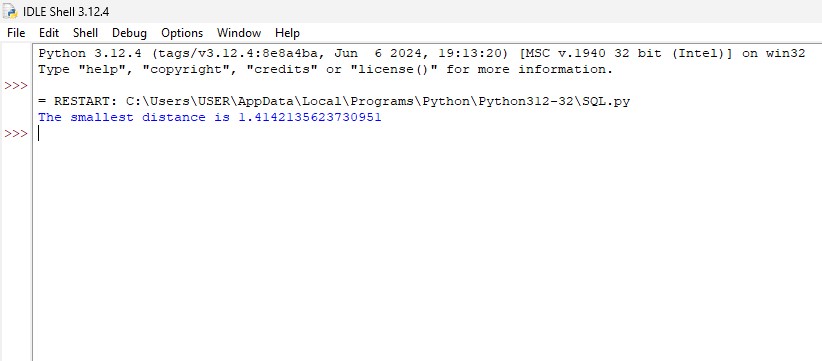
points\_sorted\_by\_x = sorted(points, key=lambda p: p[0])

points\_sorted\_by\_y = sorted(points, key=lambda p: p[1])

return closest\_pair\_recursive(points\_sorted\_by\_x, points\_sorted\_by\_y)

points = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]

print("The smallest distance is", closest\_pair(points))



7. Convex Hull

def convex\_hull(points):

def ccw(p1, p2, p3):

return (p2[1] - p1[1]) \* (p3[0] - p2[0]) > (p2[0] - p1[0]) \* (p3[1] - p2[1])

points = sorted(points)

lower = []

for p in points:

while len(lower) >= 2 and not ccw(lower[-2], lower[-1], p):

lower.pop()

lower.append(p)

upper = []

for p in reversed(points):

while len(upper) >= 2 and not ccw(upper[-2], upper[-1], p):

upper.pop()

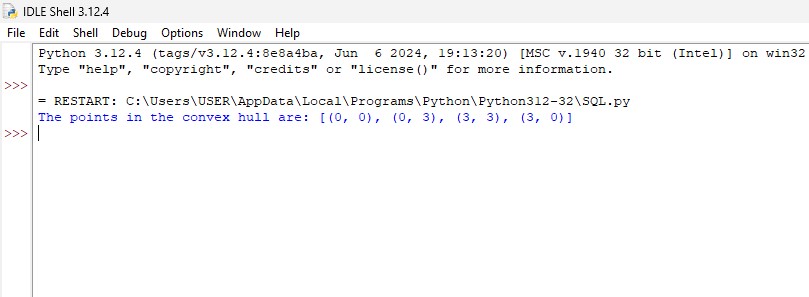
upper.append(p)

return lower[:-1] + upper[:-1]

points = [(0, 3), (2, 2), (1, 1), (2, 1), (3, 0), (0, 0), (3, 3)]

hull = convex\_hull(points)

print("The points in the convex hull are:", hull)



8. Exhaustive Search

def exhaustive\_search(data, target):

for i in range(len(data)):

if data[i] == target:

return i

return -1

data\_list = [4, 7, 2, 9, 1, 5]

target\_value = 9

result = exhaustive\_search(data\_list, target\_value)

print(f"Target value {target\_value} found at index: {result}")

