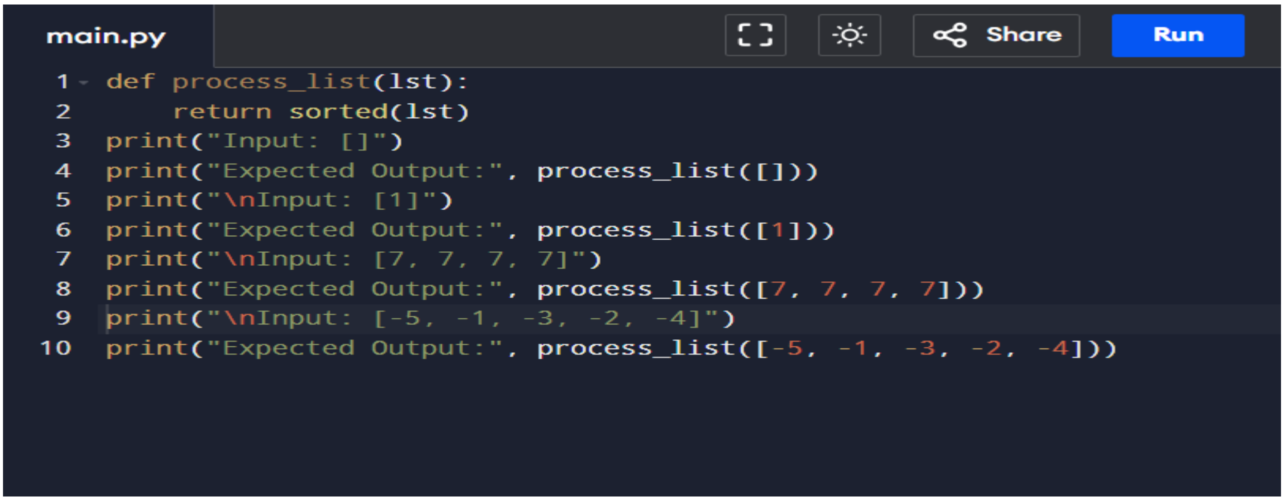
DAA LAB EXERCISE

TOPIC 2 : BRUTE FORCE

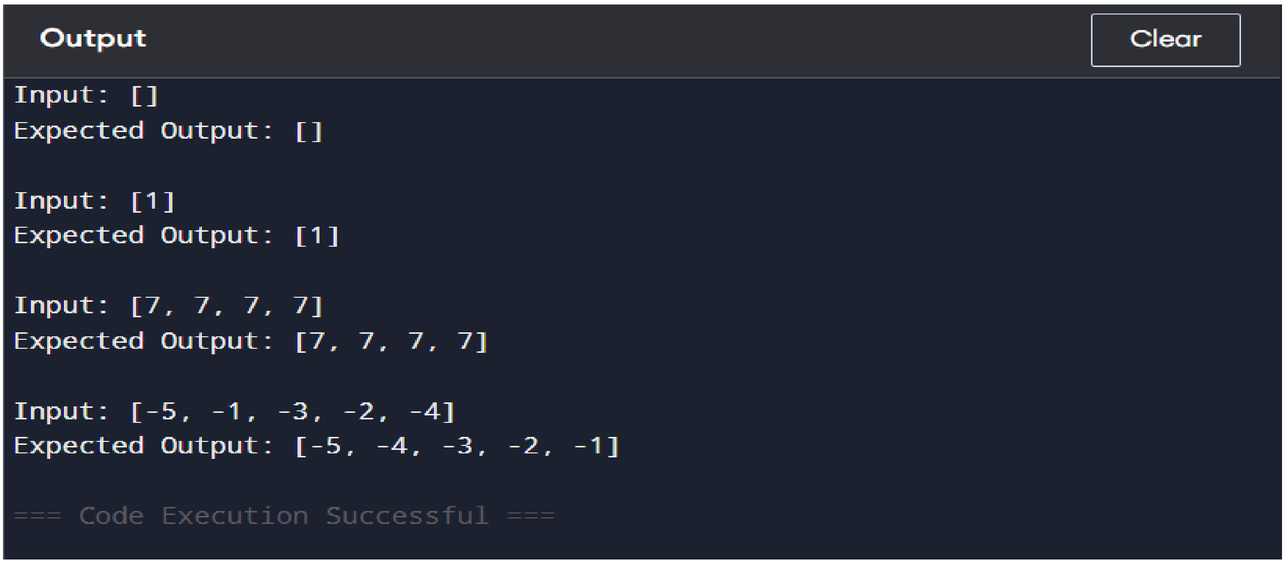
EXP 1: Program to handle different types of lists and sort them

CODE



```python
def process_list(lst):
    return sorted(lst)
print("Input: []")
print("Expected Output:", process_list([]))
print("\nInput: [1]")
print("Expected Output:", process_list([1]))
print("\nInput: [7, 7, 7, 7]")
print("Expected Output:", process_list([7, 7, 7, 7]))
print("\nInput: [-5, -1, -3, -2, -4]")
print("Expected Output:", process_list([-5, -1, -3, -2, -4]))
```

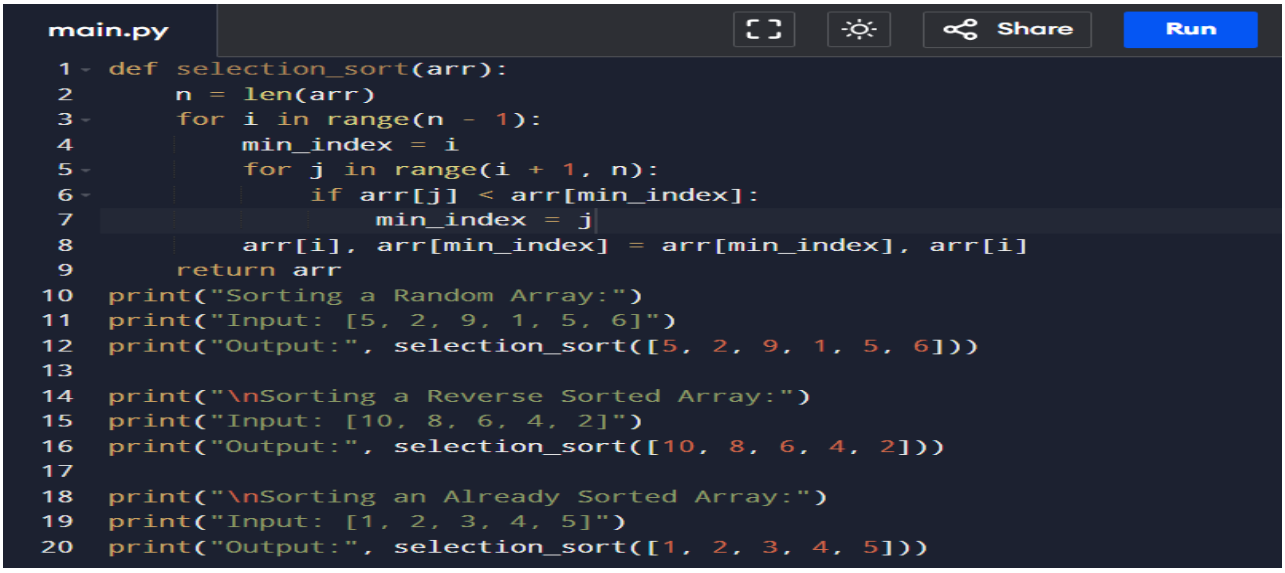
OUTPUT



```
Input: []
Expected Output: []

Input: [1]
Expected Output: [1]

Input: [7, 7, 7, 7]
Expected Output: [7, 7, 7, 7]

Input: [-5, -1, -3, -2, -4]
Expected Output: [-5, -4, -3, -2, -1]

=== Code Execution Successful ===
```

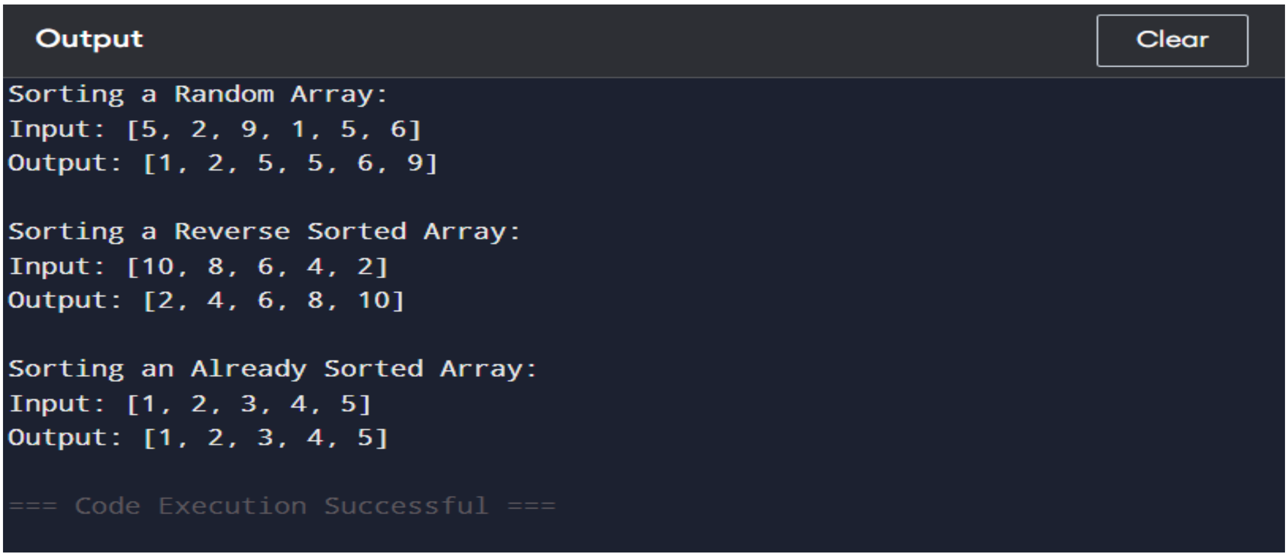
EXP 2: Implementation of selection sort algorithm

CODE



```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        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
print("Sorting a Random Array:")
print("Input: [5, 2, 9, 1, 5, 6]")
print("Output:", selection_sort([5, 2, 9, 1, 5, 6]))

print("\nSorting a Reverse Sorted Array:")
print("Input: [10, 8, 6, 4, 2]")
print("Output:", selection_sort([10, 8, 6, 4, 2]))

print("\nSorting an Already Sorted Array:")
print("Input: [1, 2, 3, 4, 5]")
print("Output:", selection_sort([1, 2, 3, 4, 5]))
```

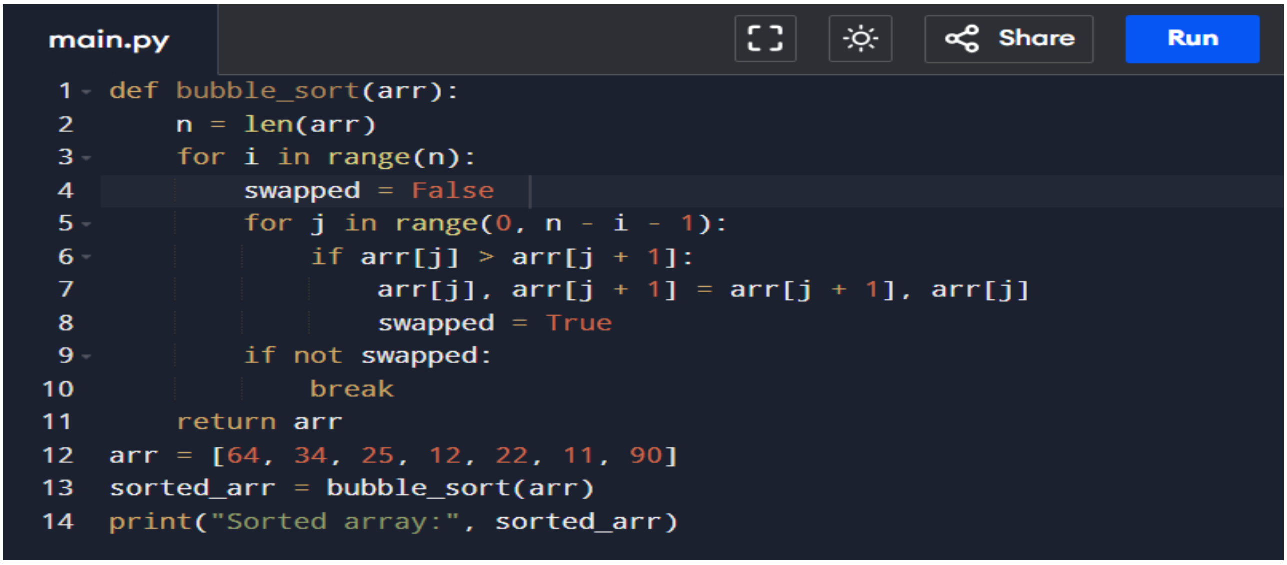
OUTPUT



```
Sorting a Random Array:
Input: [5, 2, 9, 1, 5, 6]
Output: [1, 2, 5, 5, 6, 9]

Sorting a Reverse Sorted Array:
Input: [10, 8, 6, 4, 2]
Output: [2, 4, 6, 8, 10]

Sorting an Already Sorted Array:
Input: [1, 2, 3, 4, 5]
Output: [1, 2, 3, 4, 5]

=== Code Execution Successful ===
```

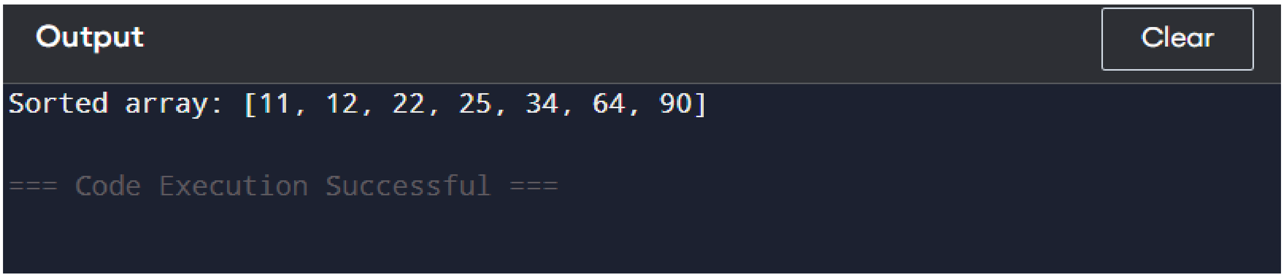
EXP 3 : Optimized Bubble sort with early exit

CODE



```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
arr = [64, 34, 25, 12, 22, 11, 90]
sorted_arr = bubble_sort(arr)
print("Sorted array:", sorted_arr)
```

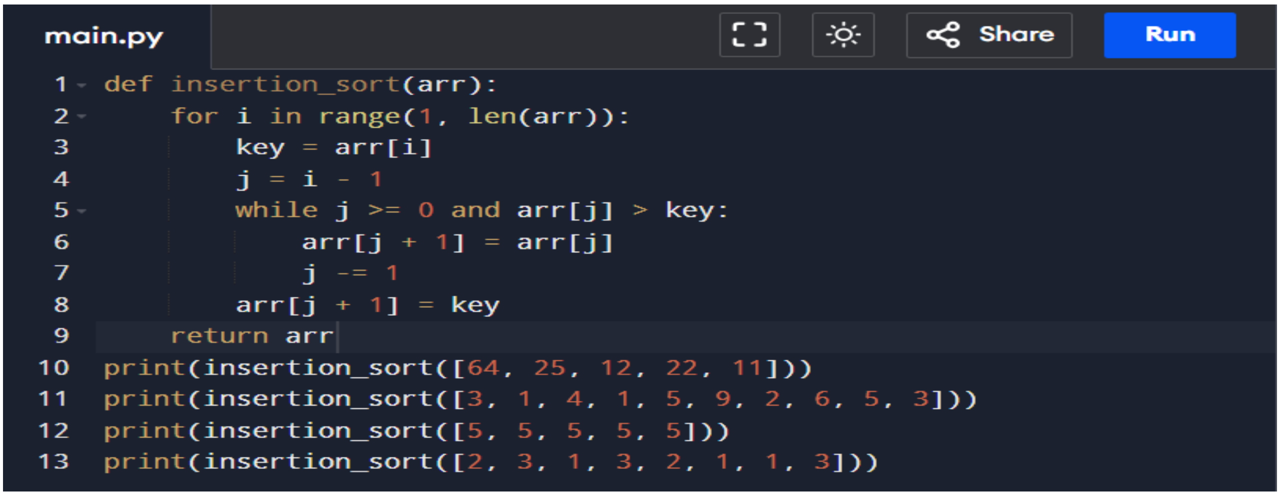
OUTPUT



```
Sorted array: [11, 12, 22, 25, 34, 64, 90]


=== Code Execution Successful ===
```

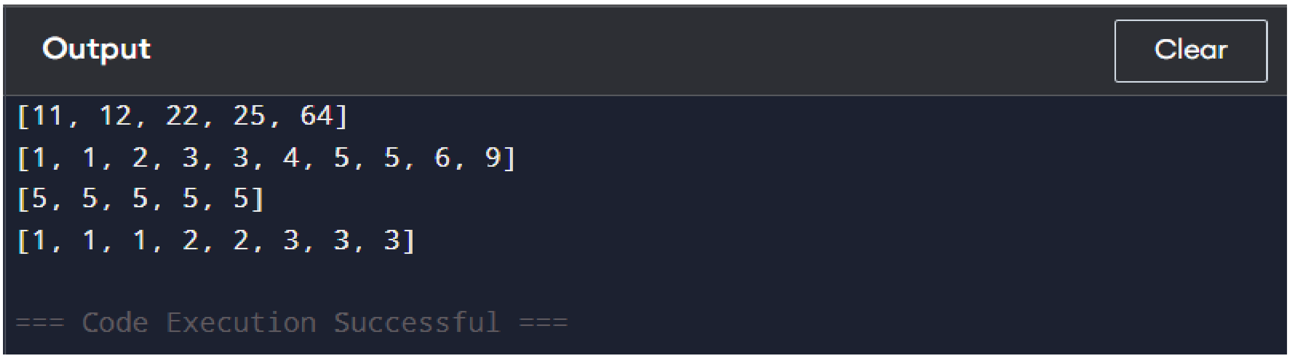
EXP 4 : Insertion Sort Handling Duplicates

CODE



```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
print(insertion_sort([64, 25, 12, 22, 11]))
print(insertion_sort([3, 1, 4, 1, 5, 9, 2, 6, 5, 3]))
print(insertion_sort([5, 5, 5, 5, 5]))
print(insertion_sort([2, 3, 1, 3, 2, 1, 1, 3]))
```

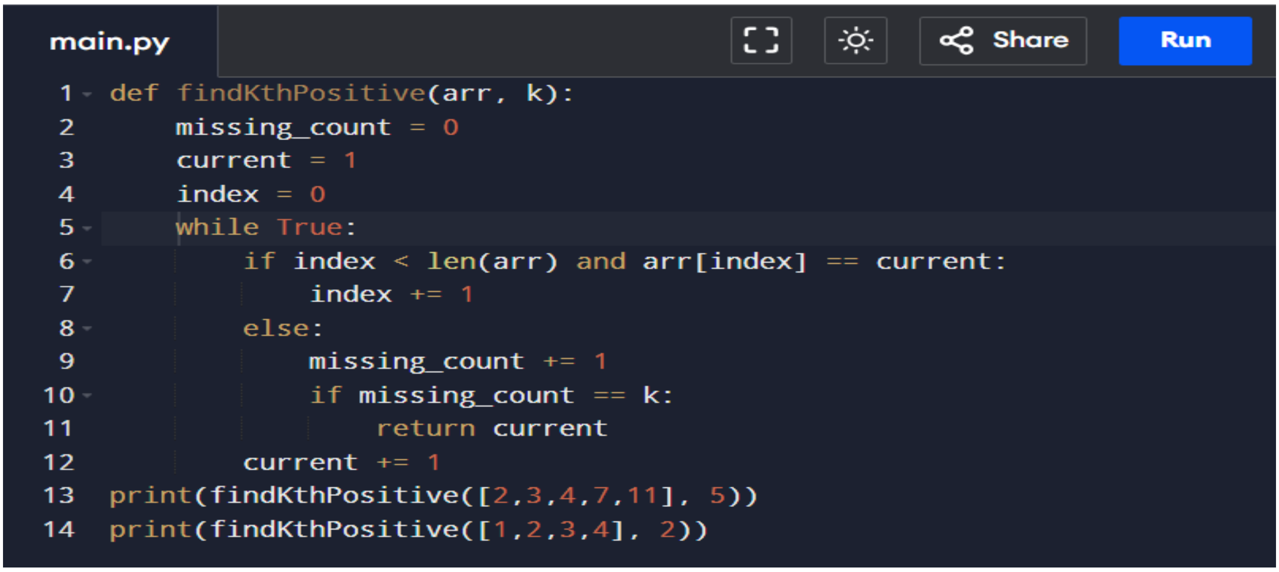
OUTPUT



```
[11, 12, 22, 25, 64]
[1, 1, 2, 3, 3, 4, 5, 5, 6, 9]
[5, 5, 5, 5, 5]
[1, 1, 1, 2, 2, 3, 3, 3]

=== Code Execution Successful ===
```

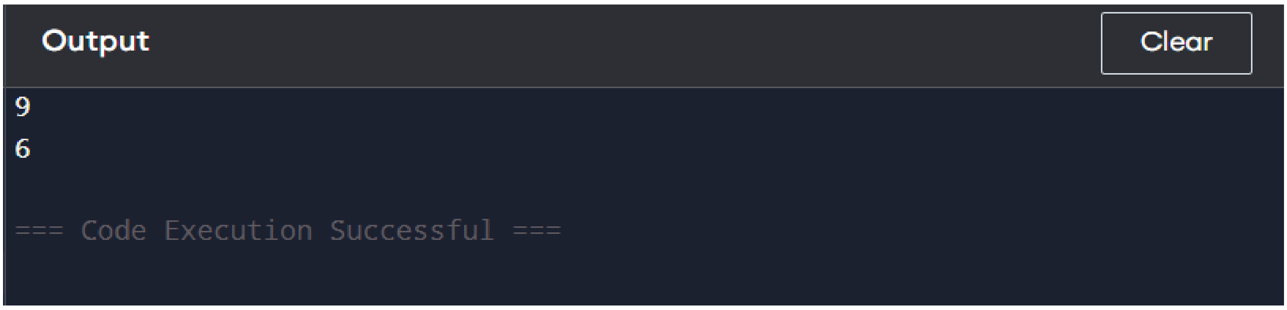
EXP 5 : Find the Kth Missing positive number

CODE



```python
def findKthPositive(arr, k):
    missing_count = 0
    current = 1
    index = 0
    while True:
        if index < len(arr) and arr[index] == current:
            index += 1
        else:
            missing_count += 1
            if missing_count == k:
                return current
        current += 1
print(findKthPositive([2,3,4,7,11], 5))
print(findKthPositive([1,2,3,4], 2))
```

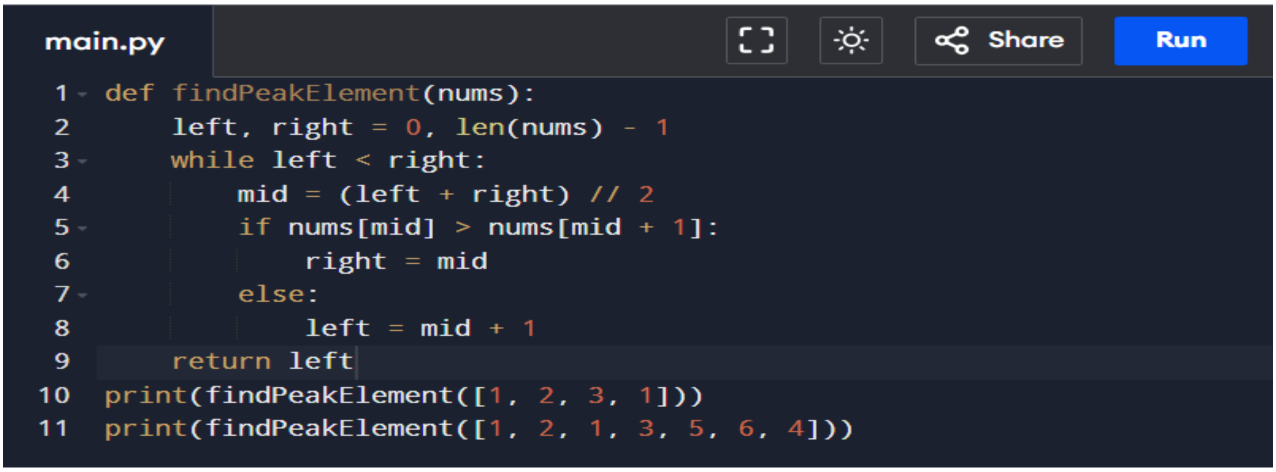
OUTPUT



```
9
6

=== Code Execution Successful ===
```

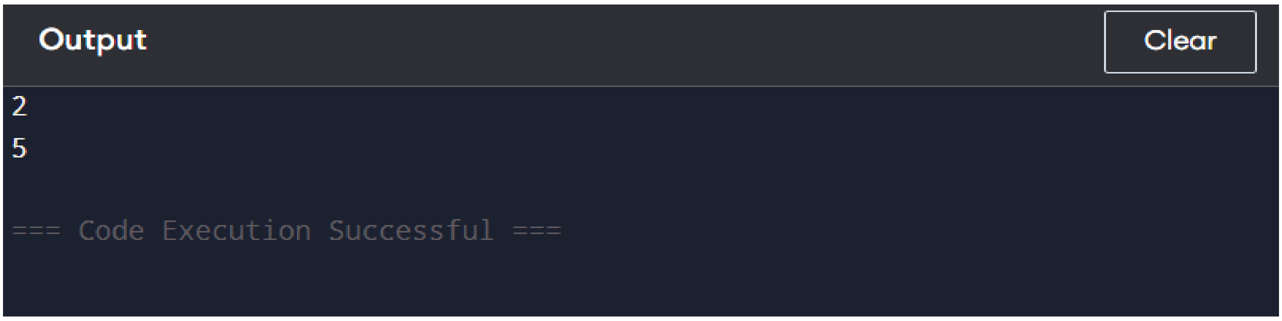
EXP 6 :Find Peak element in an array using Binary Search

CODE



```python
def findPeakElement(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = (left + right) // 2
        if nums[mid] > nums[mid + 1]:
            right = mid
        else:
            left = mid + 1
    return left
print(findPeakElement([1, 2, 3, 1]))
print(findPeakElement([1, 2, 1, 3, 5, 6, 4]))
```

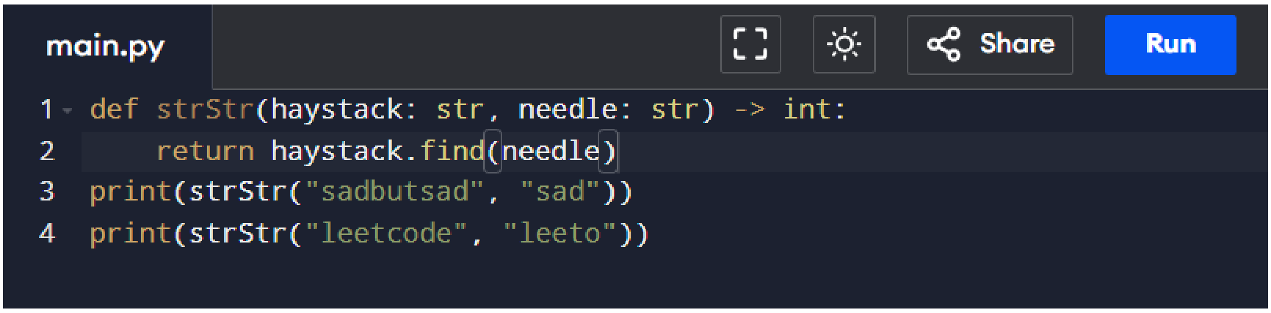
OUTPUT



```
2
5

=== Code Execution Successful ===
```

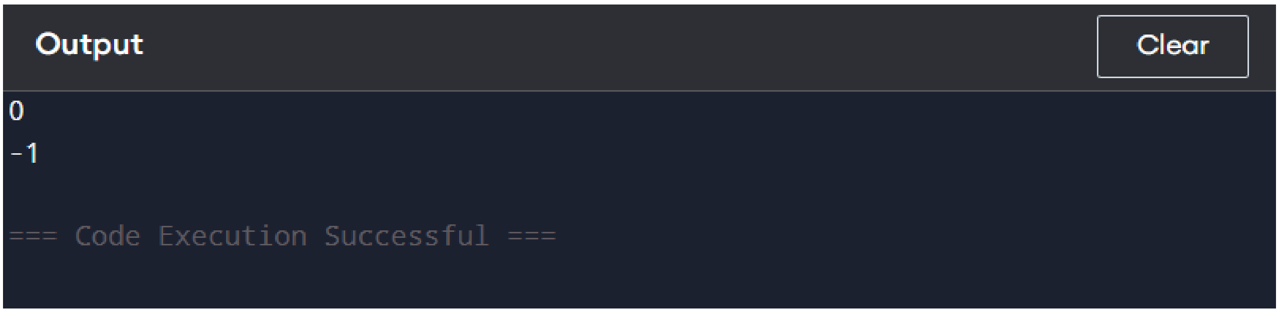
EXP 7 :Program to Find the First occurrence of substring

CODE



```python
def strStr(haystack: str, needle: str) -> int:
    return haystack.find(needle)
print(strStr("sadbutsad", "sad"))
print(strStr("leetcode", "leeto"))
```

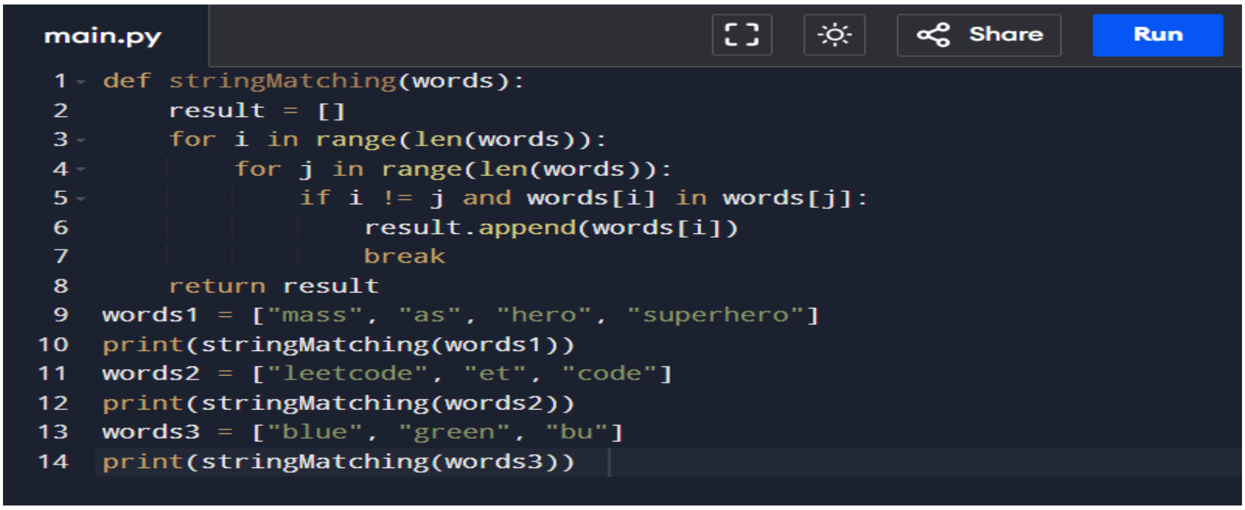
OUTPUT



```
0
-1

=== Code Execution Successful ===
```

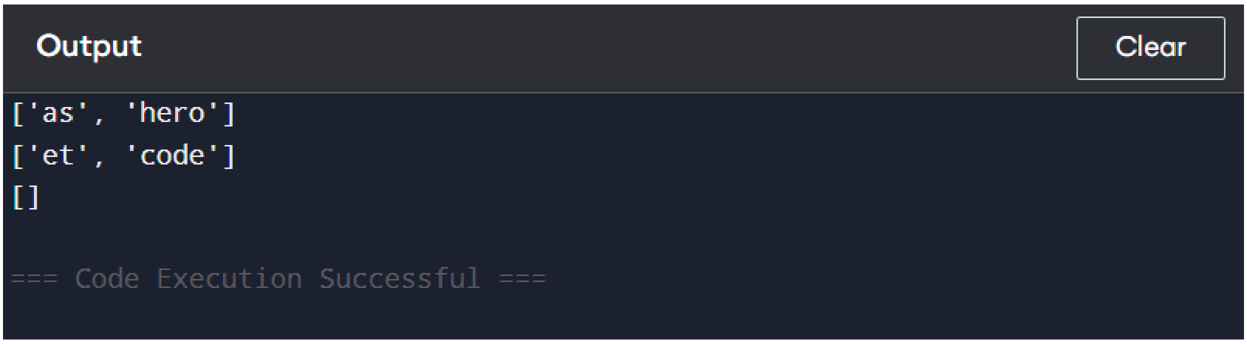
EXP 8 : Find Substring in word list

CODE



```python
def stringMatching(words):
    result = []
    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                result.append(words[i])
                break
    return result
words1 = ["mass", "as", "hero", "superhero"]
print(stringMatching(words1))
words2 = ["leetcode", "et", "code"]
print(stringMatching(words2))
words3 = ["blue", "green", "bu"]
print(stringMatching(words3))
```

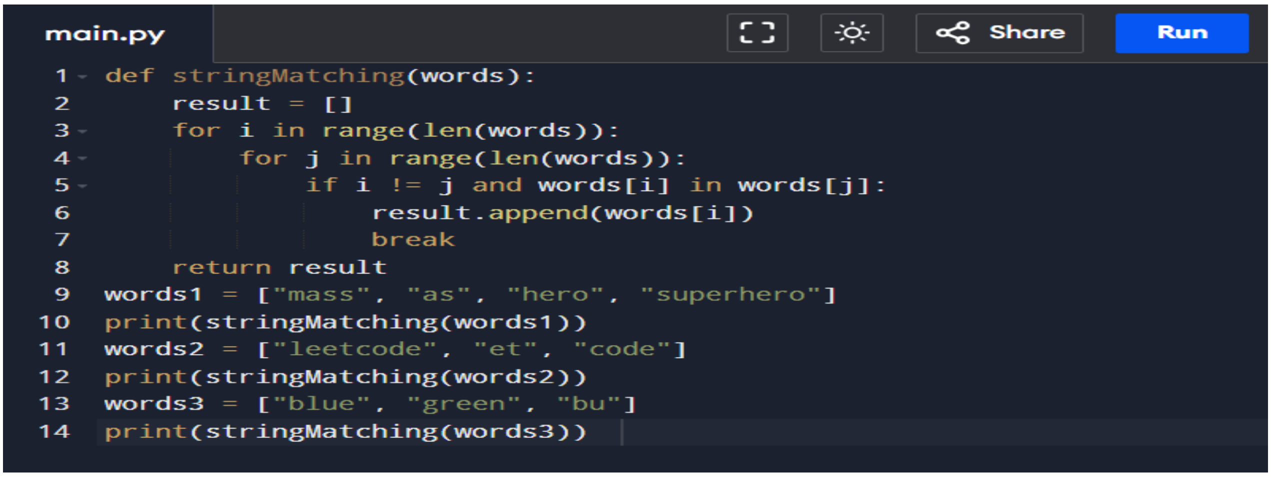
OUTPUT



```
['as', 'hero']
['et', 'code']
[]

=== Code Execution Successful ===
```

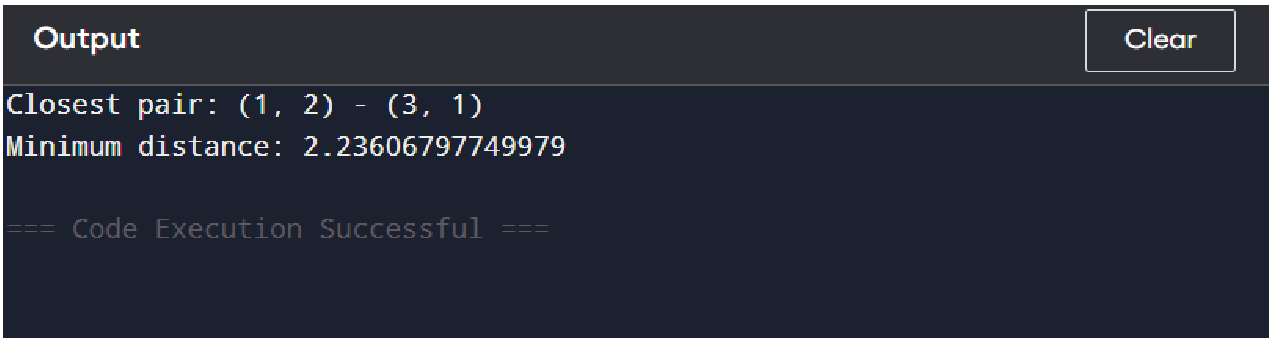
EXP 9 : Program to find the closest pair of points using Brust force method

CODE



```python
def stringMatching(words):
    result = []
    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                result.append(words[i])
                break
    return result
words1 = ["mass", "as", "hero", "superhero"]
print(stringMatching(words1))
words2 = ["leetcode", "et", "code"]
print(stringMatching(words2))
words3 = ["blue", "green", "bu"]
print(stringMatching(words3))
```

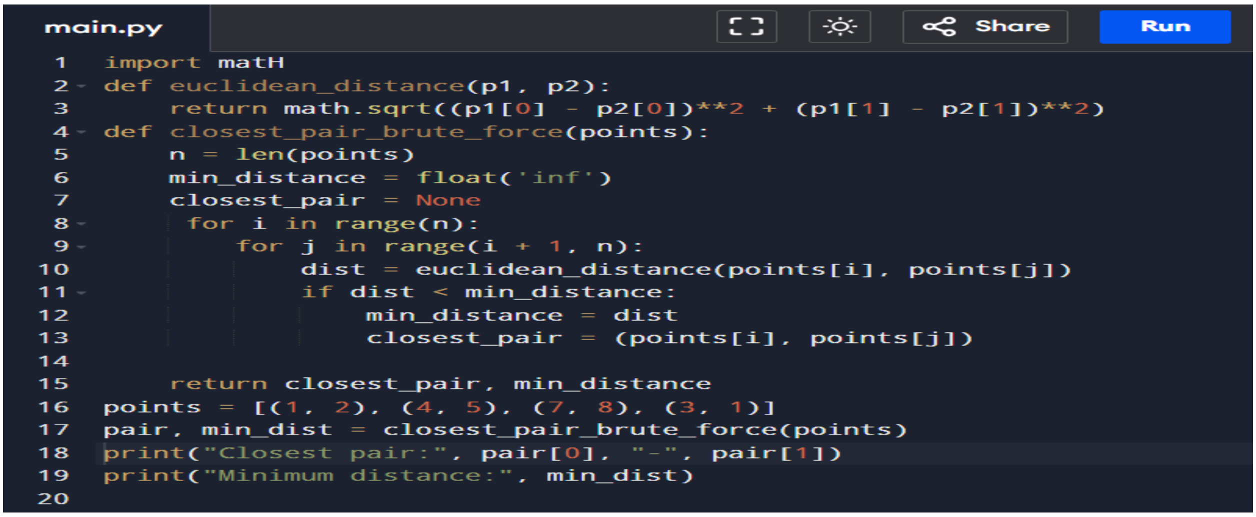
OUTPUT



```
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979

=== Code Execution Successful ===
```

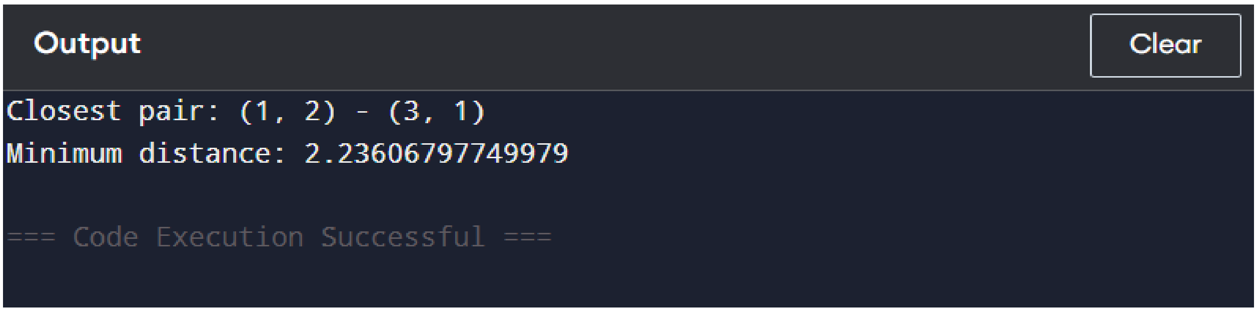
EXP 10 : Part-1 Closest pair of points using Brute force

CODE



```python
import matH
def euclidean_distance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
def closest_pair_brute_force(points):
    n = len(points)
    min_distance = float('inf')
    closest_pair = None
    for i in range(n):
        for j in range(i + 1, n):
            dist = euclidean_distance(points[i], points[j])
            if dist < min_distance:
                min_distance = dist
                closest_pair = (points[i], points[j])

    return closest_pair, min_distance
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
pair, min_dist = closest_pair_brute_force(points)
print("Closest pair:", pair[0], "-", pair[1])
print("Minimum distance:", min_dist)

```

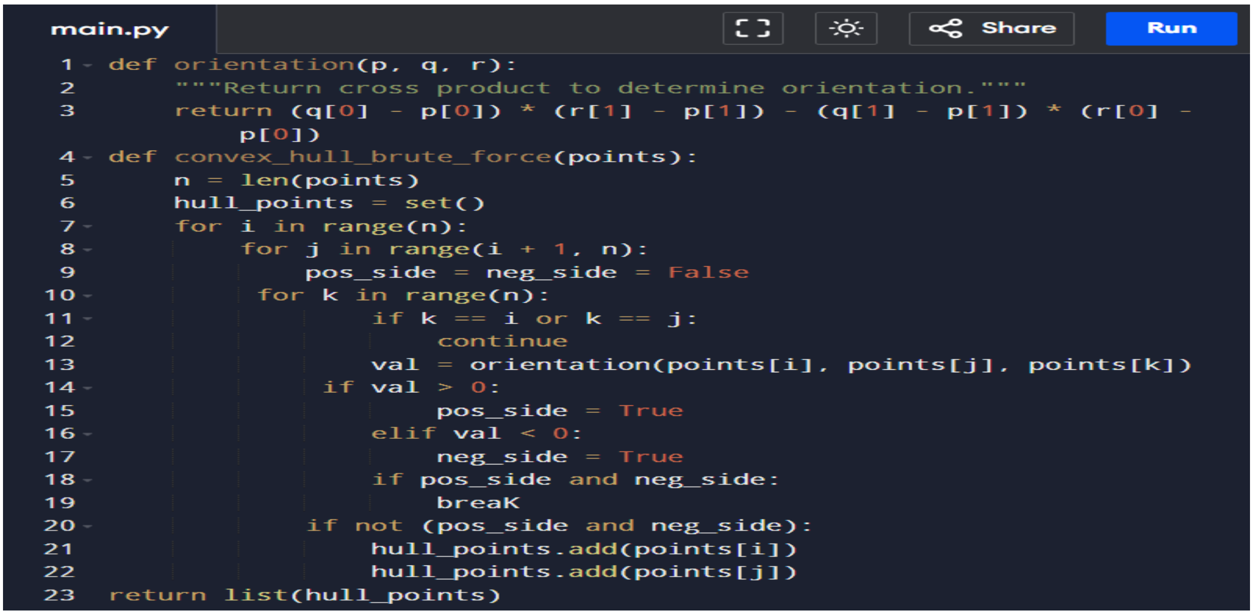
OUTPUT



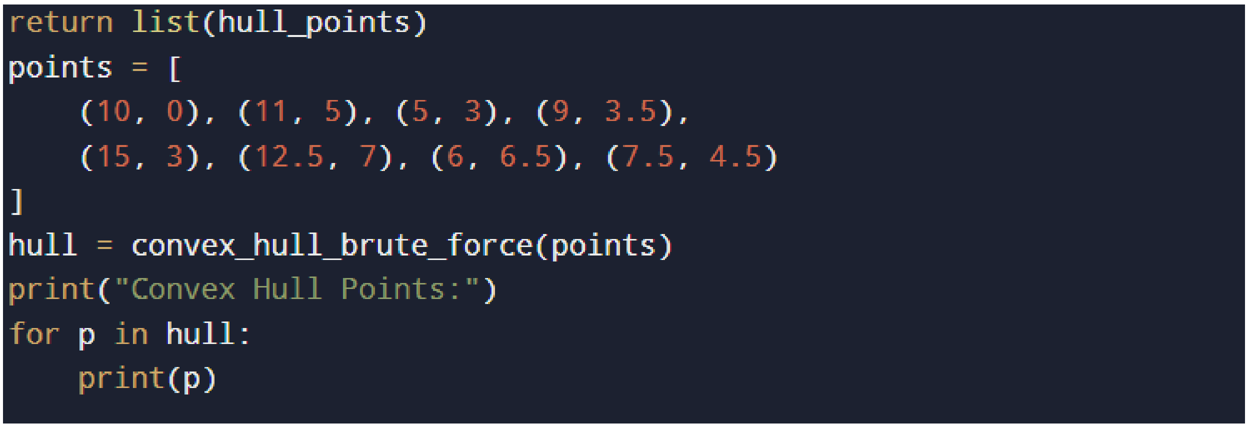
```
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979


=== Code Execution Successful ===
```

PART 2: Convex Hull(Brute Force Method)

CODE

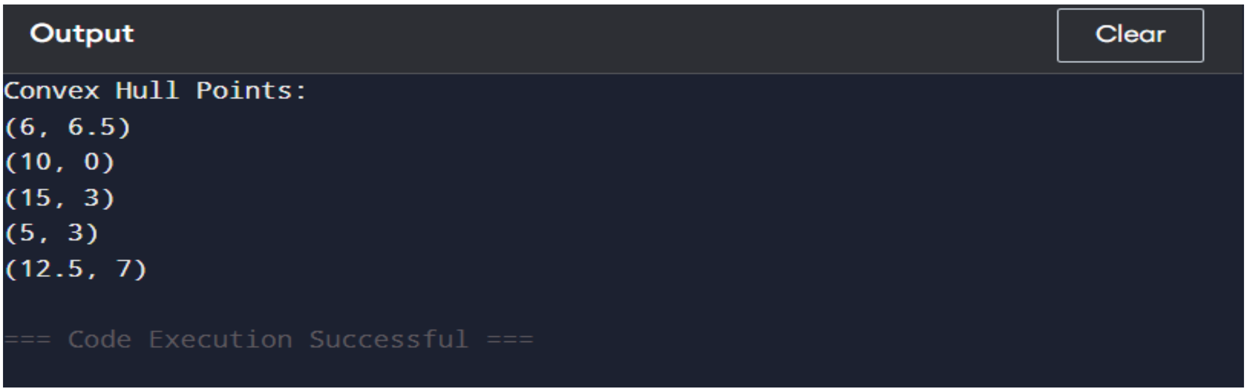


```python
def orientation(p, q, r):
    """Return cross product to determine orientation."""
    return (q[0] - p[0]) * (r[1] - p[1]) - (q[1] - p[1]) * (r[0] - p[0])
def convex_hull_brute_force(points):
    n = len(points)
    hull_points = set()
    for i in range(n):
        for j in range(i + 1, n):
            pos_side = neg_side = False
         for k in range(n):
                if k == i or k == j:
                    continue
                val = orientation(points[i], points[j], points[k])
             if val > 0:
                    pos_side = True
                elif val < 0:
                    neg_side = True
                if pos_side and neg_side:
                    break
            if not (pos_side and neg_side):
                hull_points.add(points[i])
                hull_points.add(points[j])
return list(hull_points)
```



```python
return list(hull_points)
points = [
    (10, 0), (11, 5), (5, 3), (9, 3.5),
    (15, 3), (12.5, 7), (6, 6.5), (7.5, 4.5)
]
hull = convex_hull_brute_force(points)
print("Convex Hull Points:")
for p in hull:
    print(p)
```

OUTPUT



```
Convex Hull Points:
(6, 6.5)
(10, 0)
(15, 3)
(5, 3)
(12.5, 7)

=== Code Execution Successful ===
```