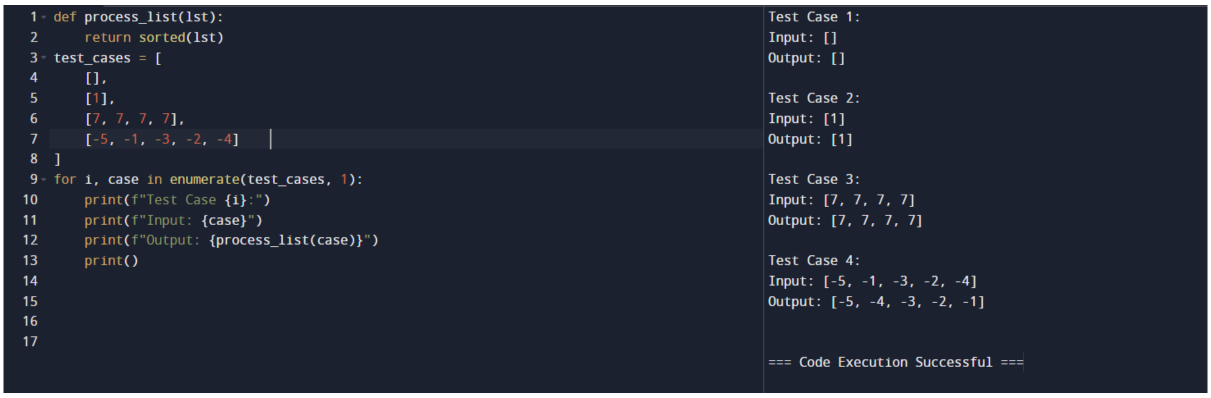
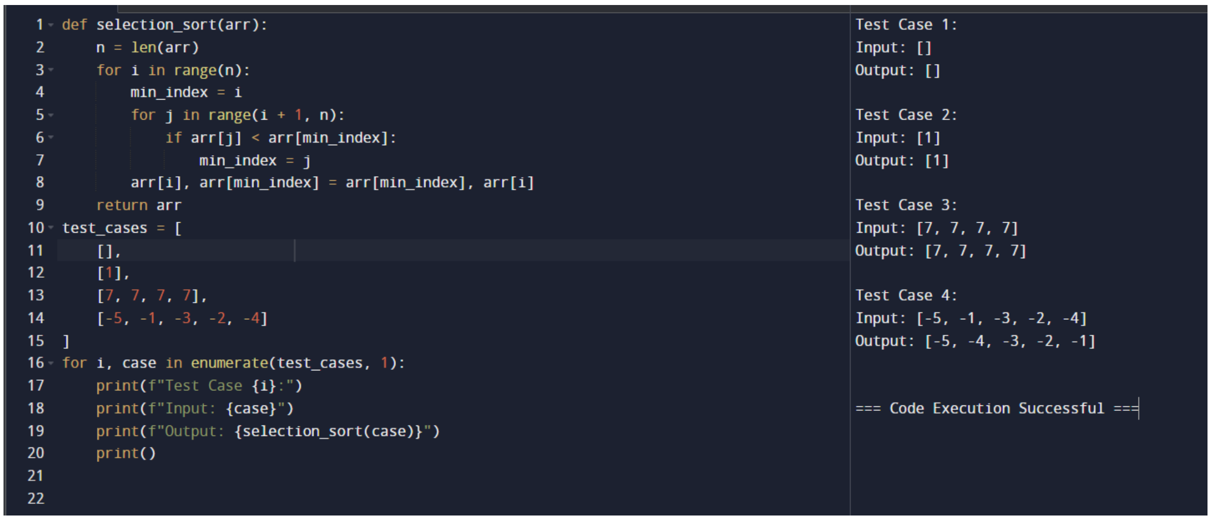
1.



```python
def process_list(lst):
    return sorted(lst)
test_cases = [
    [],
    [1],
    [7, 7, 7, 7],
    [-5, -1, -3, -2, -4]
]
for i, case in enumerate(test_cases, 1):
    print(f"Test Case {i}:")
    print(f"Input: {case}")
    print(f"Output: {process_list(case)}")
    print()
```

```
Test Case 1:
Input: []
Output: []

Test Case 2:
Input: [1]
Output: [1]

Test Case 3:
Input: [7, 7, 7, 7]
Output: [7, 7, 7, 7]

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


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

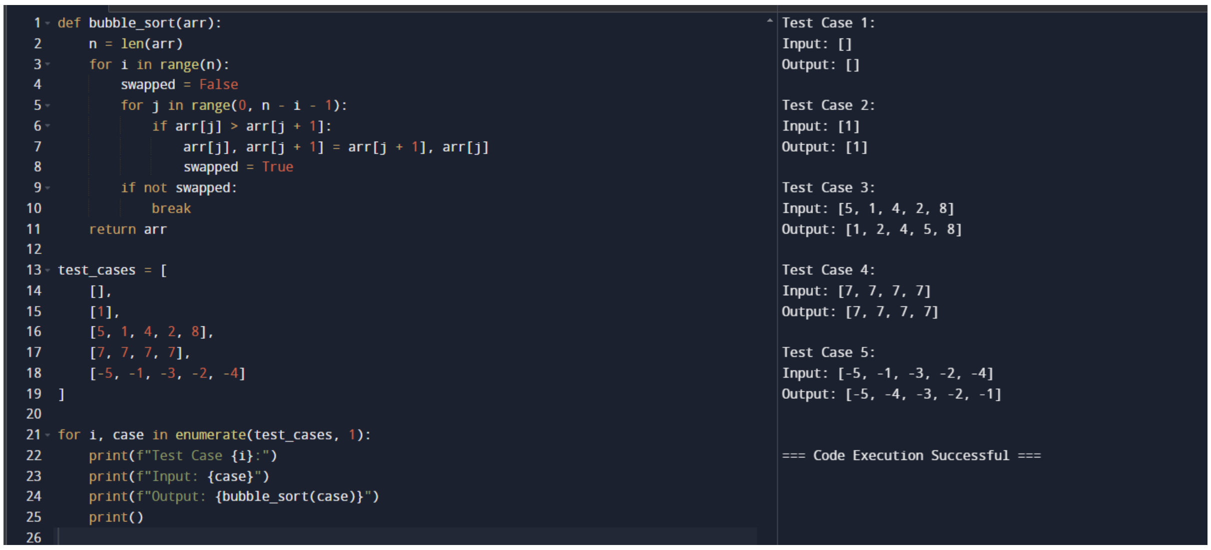
2.



```python
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
test_cases = [
    [],
    [1],
    [7, 7, 7, 7],
    [-5, -1, -3, -2, -4]
]
for i, case in enumerate(test_cases, 1):
    print(f"Test Case {i}:")
    print(f"Input: {case}")
    print(f"Output: {selection_sort(case)}")
    print()
```

```
Test Case 1:
Input: []
Output: []

Test Case 2:
Input: [1]
Output: [1]

Test Case 3:
Input: [7, 7, 7, 7]
Output: [7, 7, 7, 7]

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


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

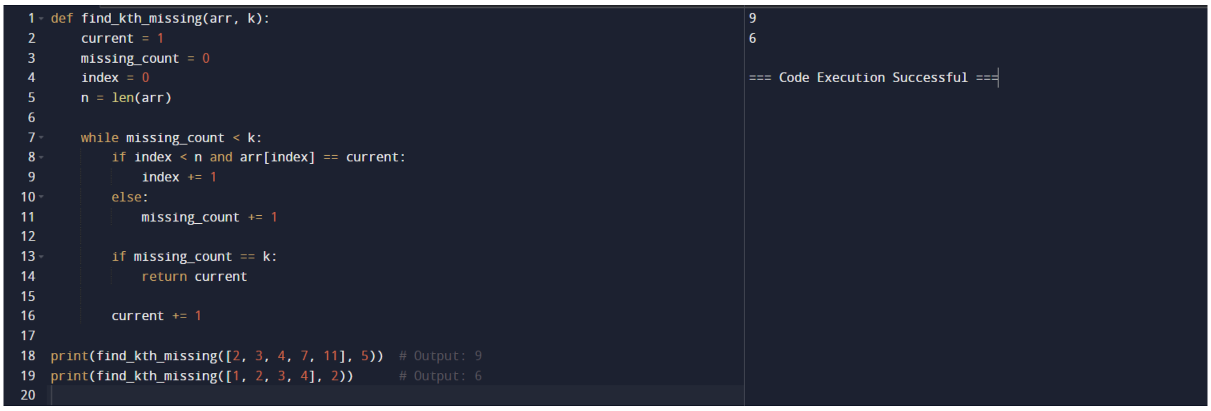
3



```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

test_cases = [
    [],
    [1],
    [5, 1, 4, 2, 8],
    [7, 7, 7, 7],
    [-5, -1, -3, -2, -4]
]

for i, case in enumerate(test_cases, 1):
    print(f"Test Case {i}:")
    print(f"Input: {case}")
    print(f"Output: {bubble_sort(case)}")
    print()
```

```
Test Case 1:
Input: []
Output: []

Test Case 2:
Input: [1]
Output: [1]

Test Case 3:
Input: [5, 1, 4, 2, 8]
Output: [1, 2, 4, 5, 8]

Test Case 4:
Input: [7, 7, 7, 7]
Output: [7, 7, 7, 7]

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


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

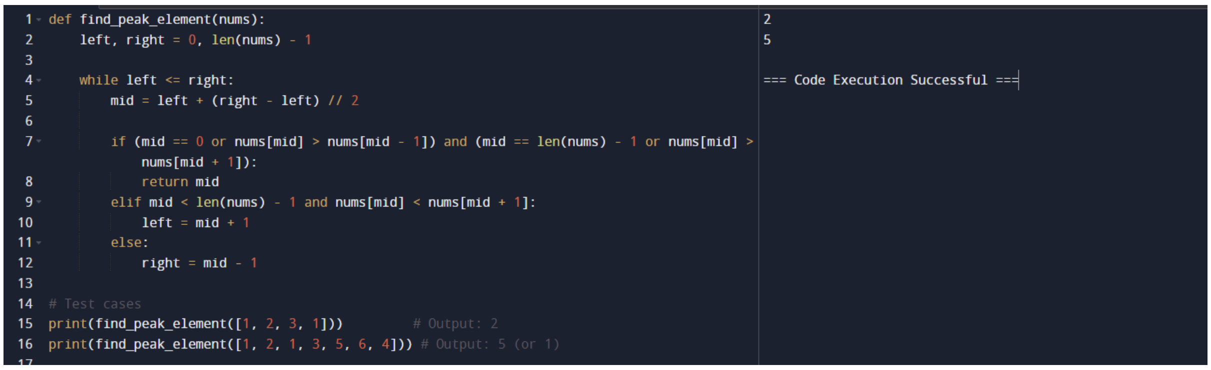
4.



```python
def find_kth_missing(arr, k):
    current = 1
    missing_count = 0
    index = 0
    n = len(arr)

    while missing_count < k:
        if index < n and arr[index] == current:
            index += 1
        else:
            missing_count += 1

        if missing_count == k:
            return current

        current += 1

print(find_kth_missing([2, 3, 4, 7, 11], 5))  # Output: 9
print(find_kth_missing([1, 2, 3, 4], 2))       # Output: 6
```

```
9
6

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

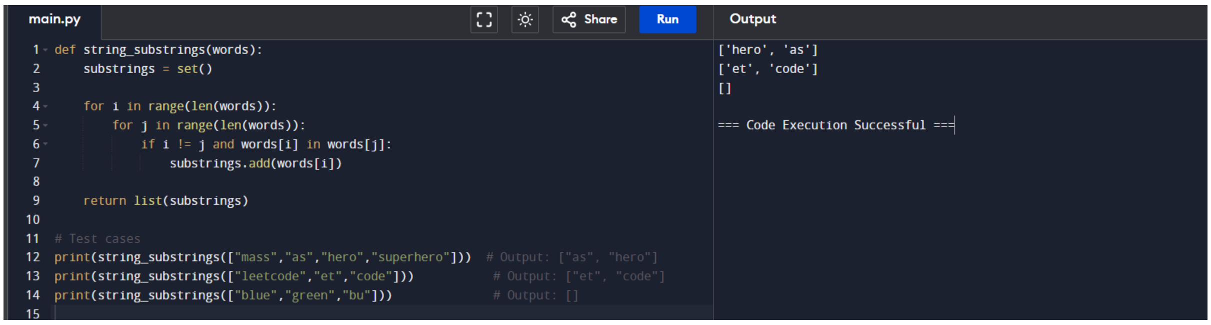
5.



```python
def find_peak_element(nums):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = left + (right - left) // 2

        if (mid == 0 or nums[mid] > nums[mid - 1]) and (mid == len(nums) - 1 or nums[mid] >
            nums[mid + 1]):
            return mid
        elif mid < len(nums) - 1 and nums[mid] < nums[mid + 1]:
            left = mid + 1
        else:
            right = mid - 1

# Test cases
print(find_peak_element([1, 2, 3, 1]))         # Output: 2
print(find_peak_element([1, 2, 1, 3, 5, 6, 4])) # Output: 5 (or 1)
```

```
2
5

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

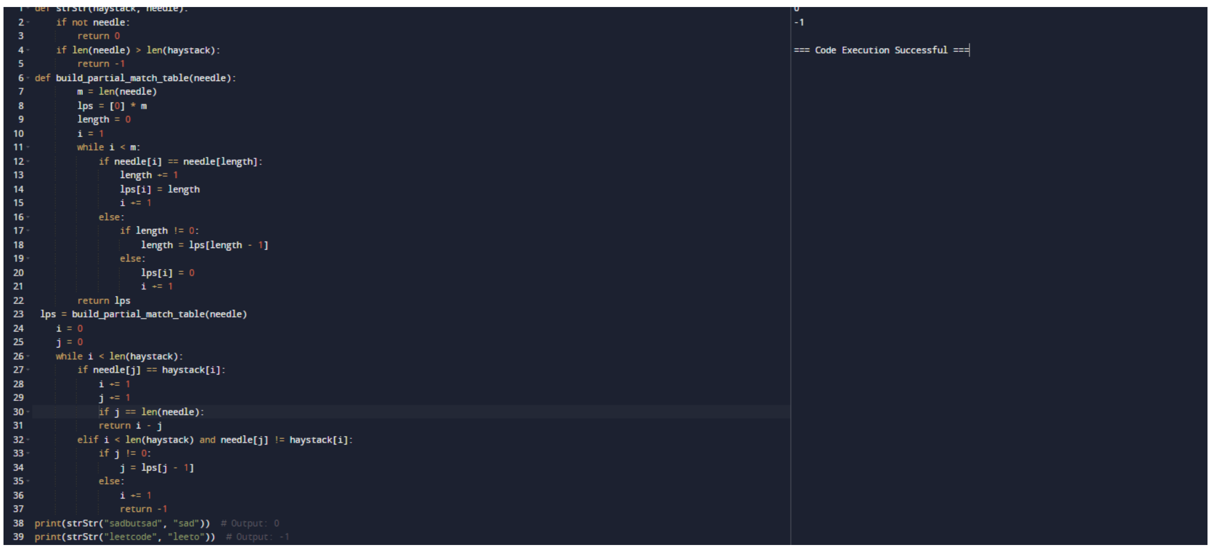
6.



```python
def string_substrings(words):
    substrings = set()

    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                substrings.add(words[i])

    return list(substrings)

# Test cases
print(string_substrings(["mass","as","hero","superhero"]))  # Output: ["as", "hero"]
print(string_substrings(["leetcode","et","code"]))           # Output: ["et", "code"]
print(string_substrings(["blue","green","bu"]))              # Output: []
```

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

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

7.



```
def strStr(haystack, needle):
    if not needle:
        return 0
    if len(needle) > len(haystack):
        return -1
def build_partial_match_table(needle):
        m = len(needle)
        lps = [0] * m
        length = 0
        i = 1
        while i < m:
            if needle[i] == needle[length]:
                length += 1
                lps[i] = length
                i += 1
            else:
                if length != 0:
                    length = lps[length - 1]
                else:
                    lps[i] = 0
                    i += 1
        return lps
 lps = build_partial_match_table(needle)
    i = 0
    j = 0
    while i < len(haystack):
        if needle[j] == haystack[i]:
            i += 1
            j += 1
            if j == len(needle):
            return i - j
        elif i < len(haystack) and needle[j] != haystack[i]:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
                return -1
print(strStr("sadbutsad", "sad"))  # Output: 0
print(strStr("leetcode", "leeto"))  # Output: -1
```

```
0
-1

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