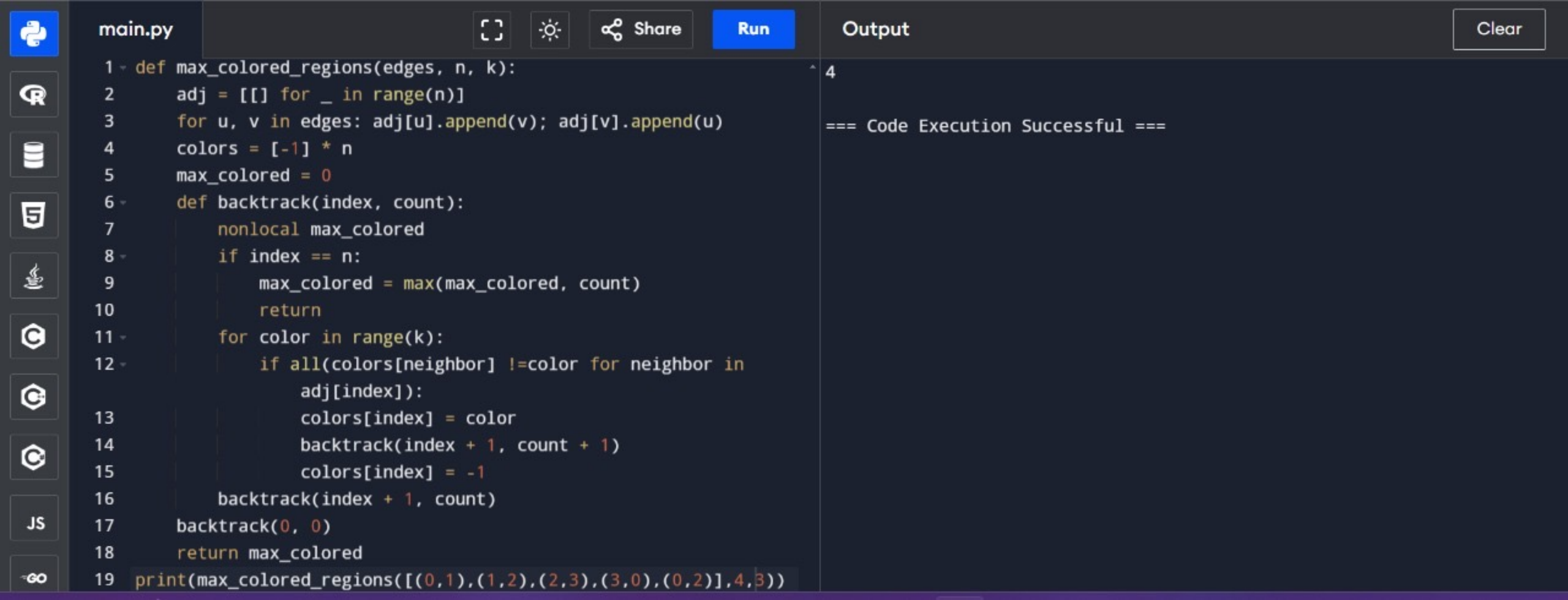
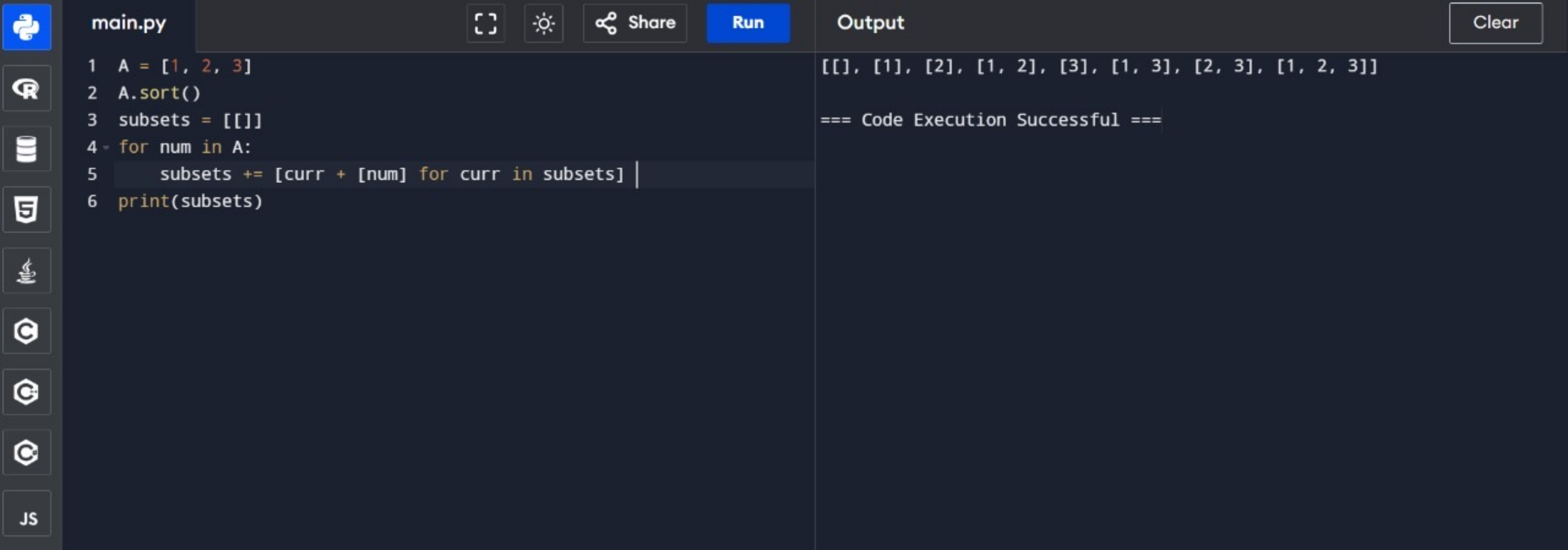


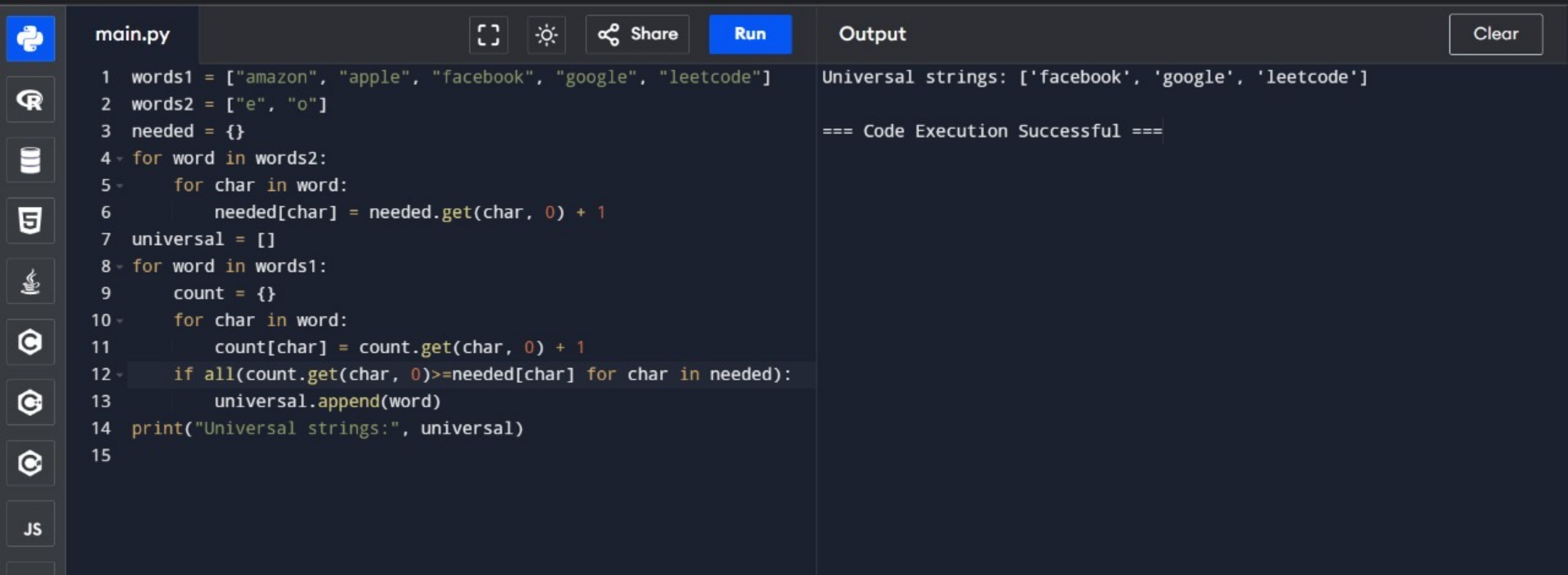
```python
def max_colored_regions(edges, n, k):
    adj = [[] for _ in range(n)]
    for u, v in edges: adj[u].append(v); adj[v].append(u)
    colors = [-1] * n
    max_colored = 0
    def backtrack(index, count):
        nonlocal max_colored
        if index == n:
            max_colored = max(max_colored, count)
            return
        for color in range(k):
            if all(colors[neighbor] !=color for neighbor in
                adj[index]):
                colors[index] = color
                backtrack(index + 1, count + 1)
                colors[index] = -1
        backtrack(index + 1, count)
    backtrack(0, 0)
    return max_colored
print(max_colored_regions([(0,1),(1,2),(2,3),(3,0),(0,2)],4,3))
```

```
4


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



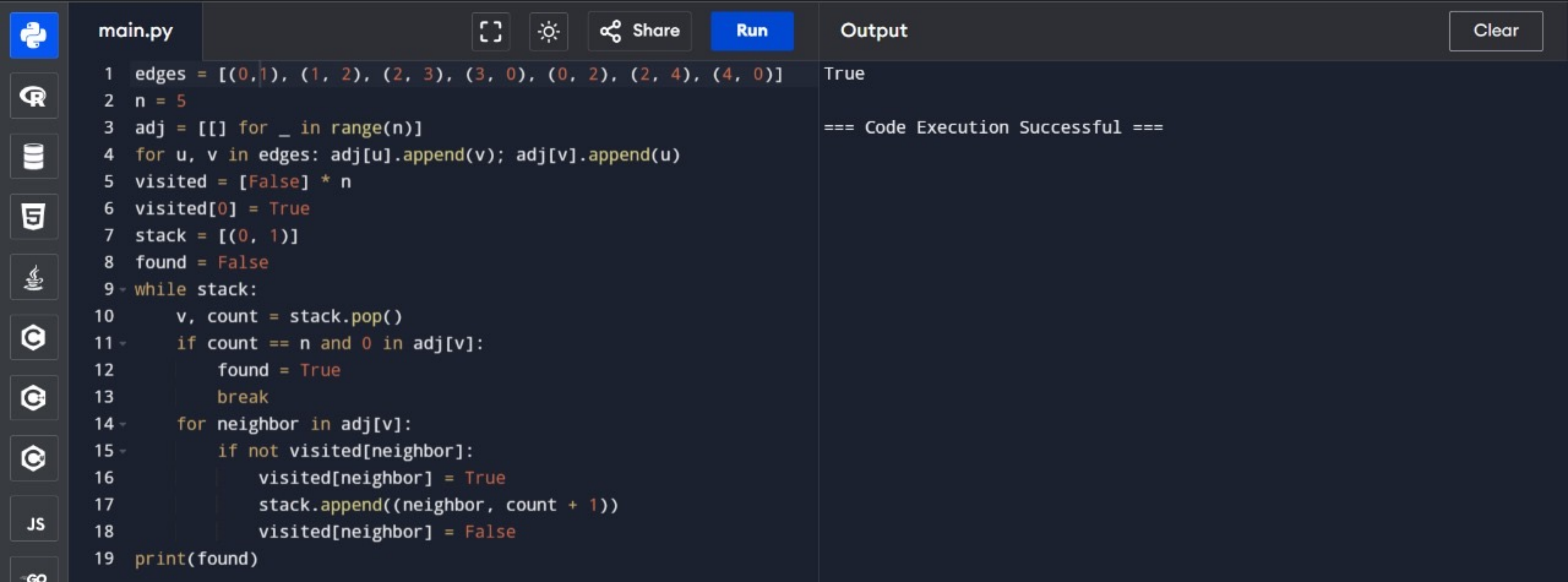
```python
A = [1, 2, 3]
A.sort()
subsets = [[]]
for num in A:
    subsets += [curr + [num] for curr in subsets]
print(subsets)
```

```
[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]

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



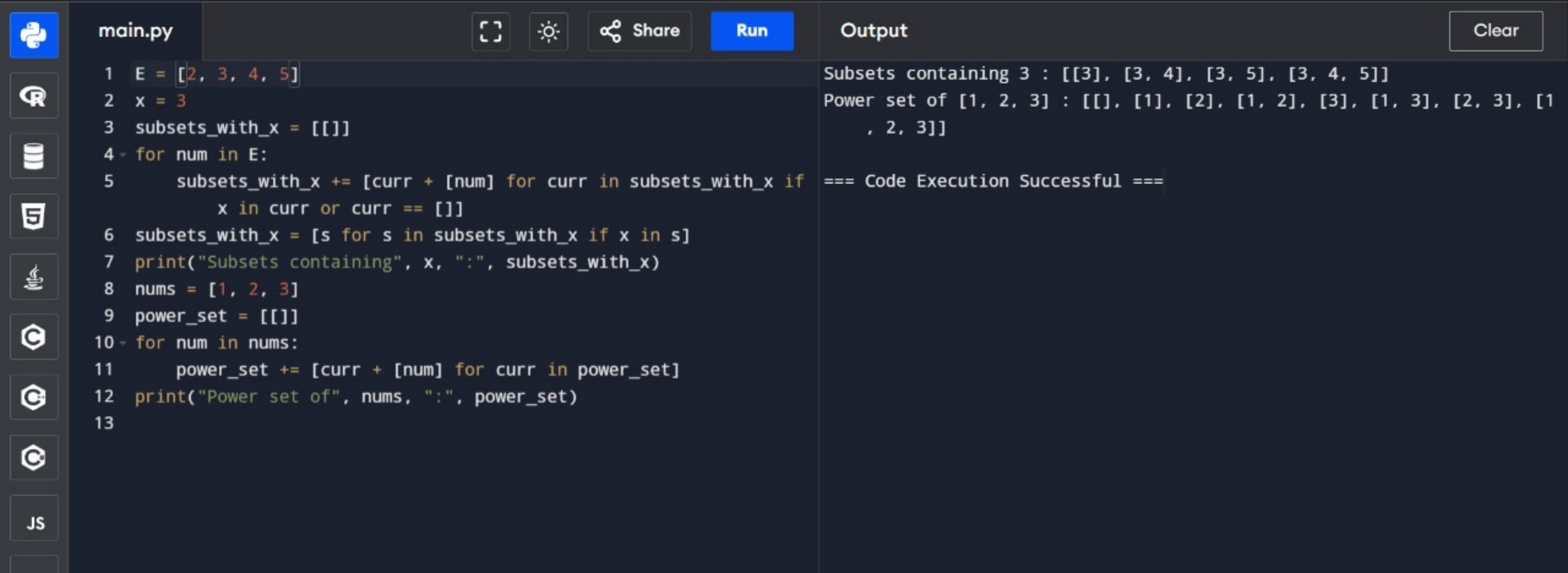
```python
words1 = ["amazon", "apple", "facebook", "google", "leetcode"]
words2 = ["e", "o"]
needed = {}
for word in words2:
    for char in word:
        needed[char] = needed.get(char, 0) + 1
universal = []
for word in words1:
    count = {}
    for char in word:
        count[char] = count.get(char, 0) + 1
    if all(count.get(char, 0)>=needed[char] for char in needed):
        universal.append(word)
print("Universal strings:", universal)
```

```
Universal strings: ['facebook', 'google', 'leetcode']

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



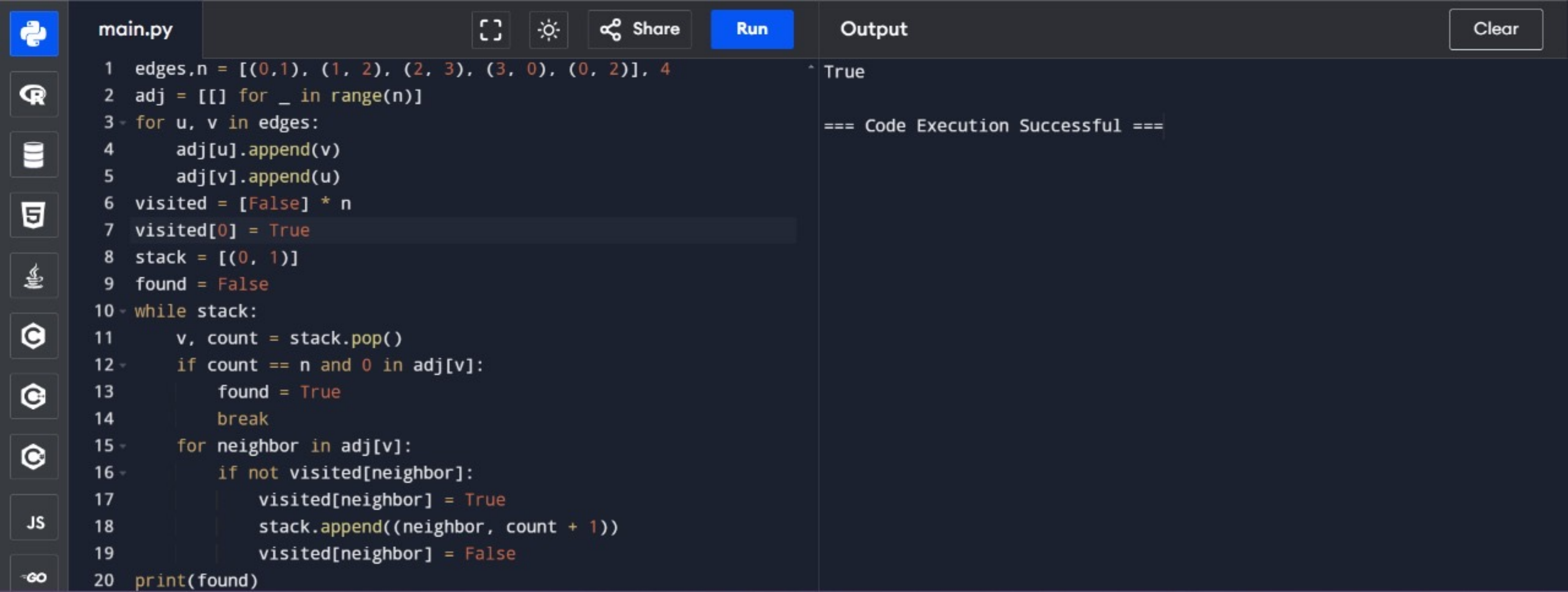
```python
edges = [(0,1), (1, 2), (2, 3), (3, 0), (0, 2), (2, 4), (4, 0)]
n = 5
adj = [[] for _ in range(n)]
for u, v in edges: adj[u].append(v); adj[v].append(u)
visited = [False] * n
visited[0] = True
stack = [(0, 1)]
found = False
while stack:
    v, count = stack.pop()
    if count == n and 0 in adj[v]:
        found = True
        break
    for neighbor in adj[v]:
        if not visited[neighbor]:
            visited[neighbor] = True
            stack.append((neighbor, count + 1))
            visited[neighbor] = False
print(found)
```

```
True


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



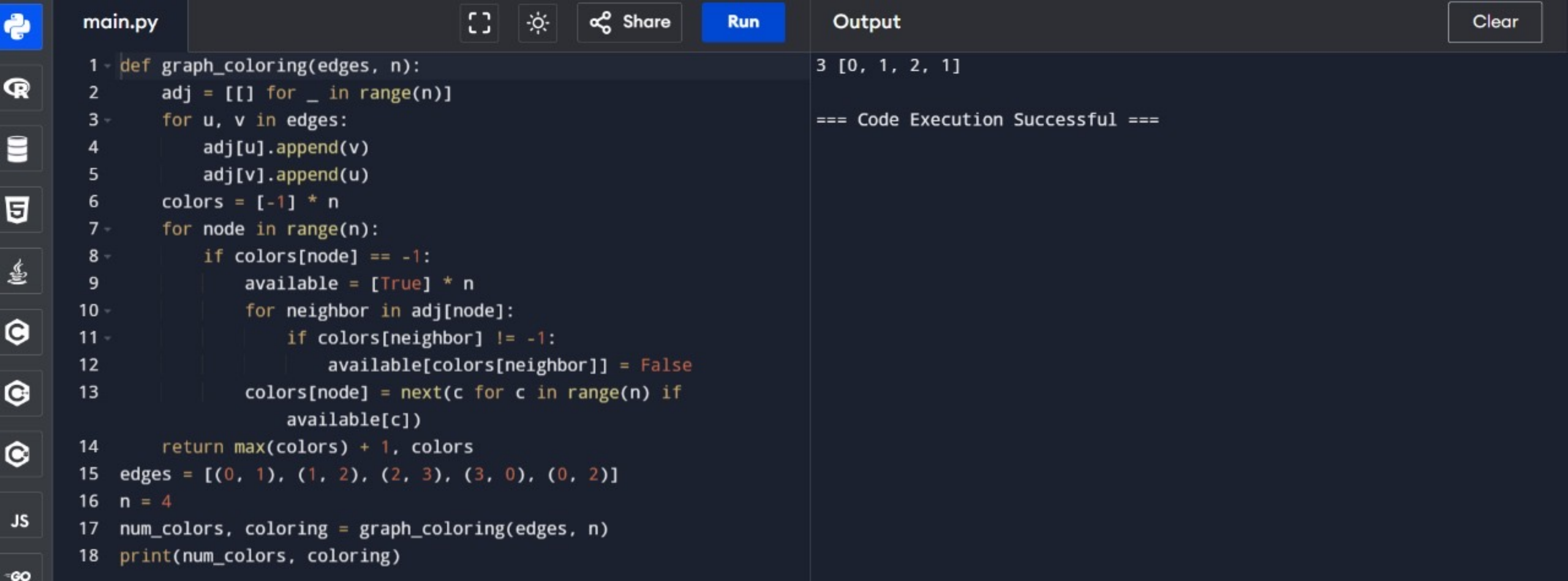
```python
E = [2, 3, 4, 5]
x = 3
subsets_with_x = [[]]
for num in E:
    subsets_with_x += [curr + [num] for curr in subsets_with_x if
        x in curr or curr == []]
subsets_with_x = [s for s in subsets_with_x if x in s]
print("Subsets containing", x, ":", subsets_with_x)
nums = [1, 2, 3]
power_set = [[]]
for num in nums:
    power_set += [curr + [num] for curr in power_set]
print("Power set of", nums, ":", power_set)
```

```
Subsets containing 3 : [[3], [3, 4], [3, 5], [3, 4, 5]]
Power set of [1, 2, 3] : [[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1
    , 2, 3]]

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



```python
edges,n = [(0,1), (1, 2), (2, 3), (3, 0), (0, 2)], 4
adj = [[] for _ in range(n)]
for u, v in edges:
    adj[u].append(v)
    adj[v].append(u)
visited = [False] * n
visited[0] = True
stack = [(0, 1)]
found = False
while stack:
    v, count = stack.pop()
    if count == n and 0 in adj[v]:
        found = True
        break
    for neighbor in adj[v]:
        if not visited[neighbor]:
            visited[neighbor] = True
            stack.append((neighbor, count + 1))
            visited[neighbor] = False
print(found)
```

```
True

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



```python
def graph_coloring(edges, n):
    adj = [[] for _ in range(n)]
    for u, v in edges:
        adj[u].append(v)
        adj[v].append(u)
    colors = [-1] * n
    for node in range(n):
        if colors[node] == -1:
            available = [True] * n
            for neighbor in adj[node]:
                if colors[neighbor] != -1:
                    available[colors[neighbor]] = False
            colors[node] = next(c for c in range(n) if
                available[c])
    return max(colors) + 1, colors
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n = 4
num_colors, coloring = graph_coloring(edges, n)
print(num_colors, coloring)
```

```
3 [0, 1, 2, 1]

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