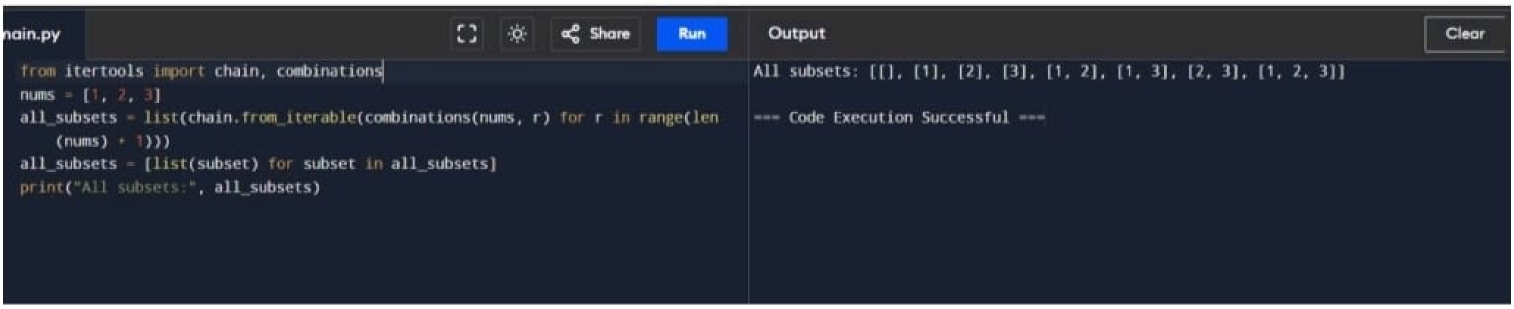
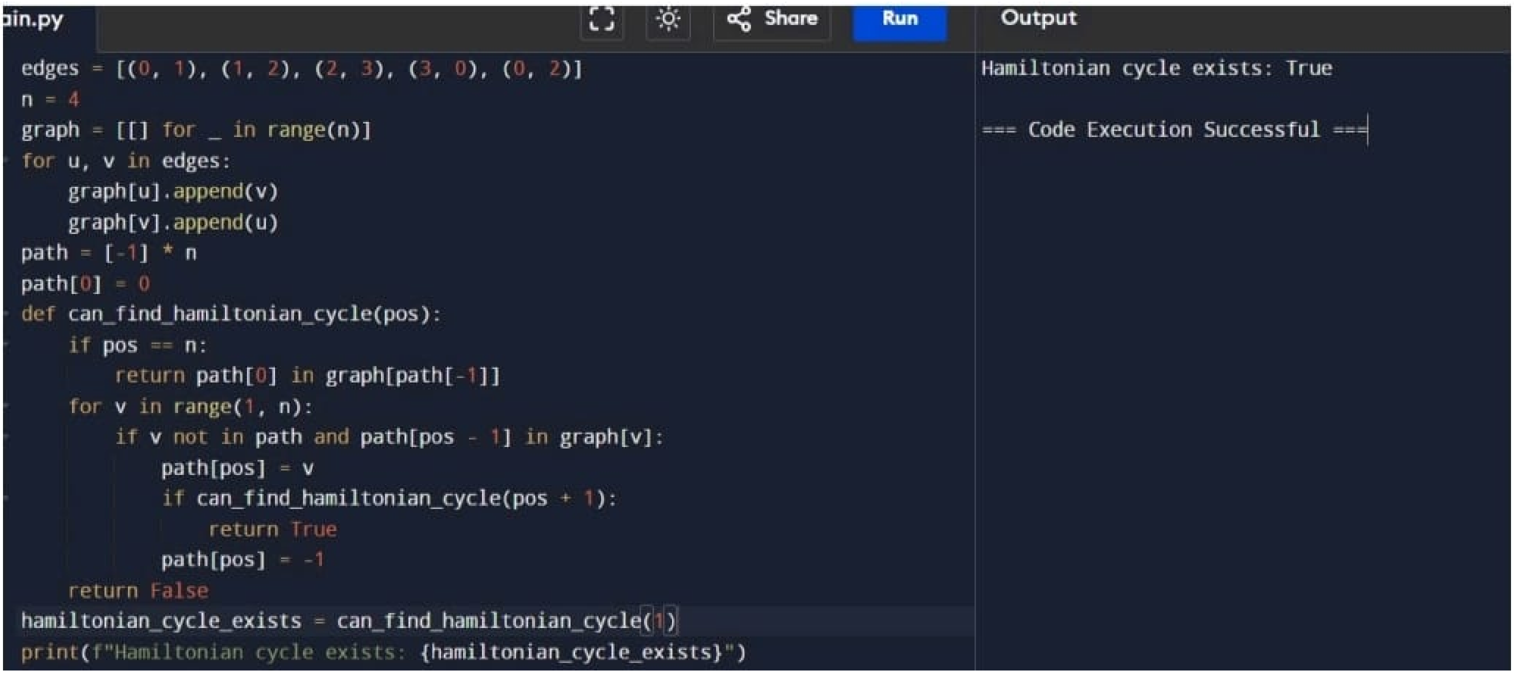


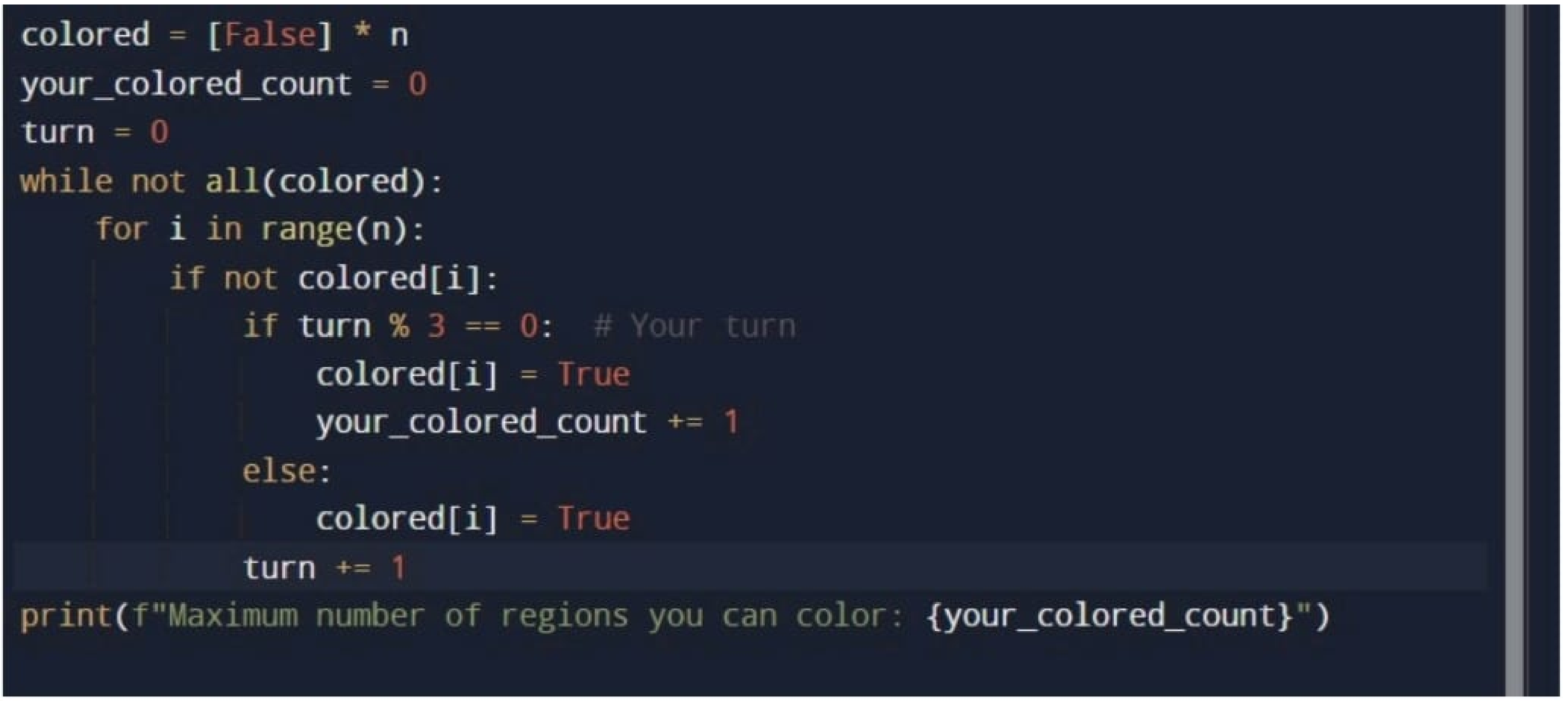
```python
from itertools import chain, combinations
nums = [1, 2, 3]
all_subsets = list(chain.from_iterable(combinations(nums, r) for r in range(len
    (nums) + 1)))
all_subsets = [list(subset) for subset in all_subsets]
print("All subsets:", all_subsets)
```

```
All subsets: [[], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]

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

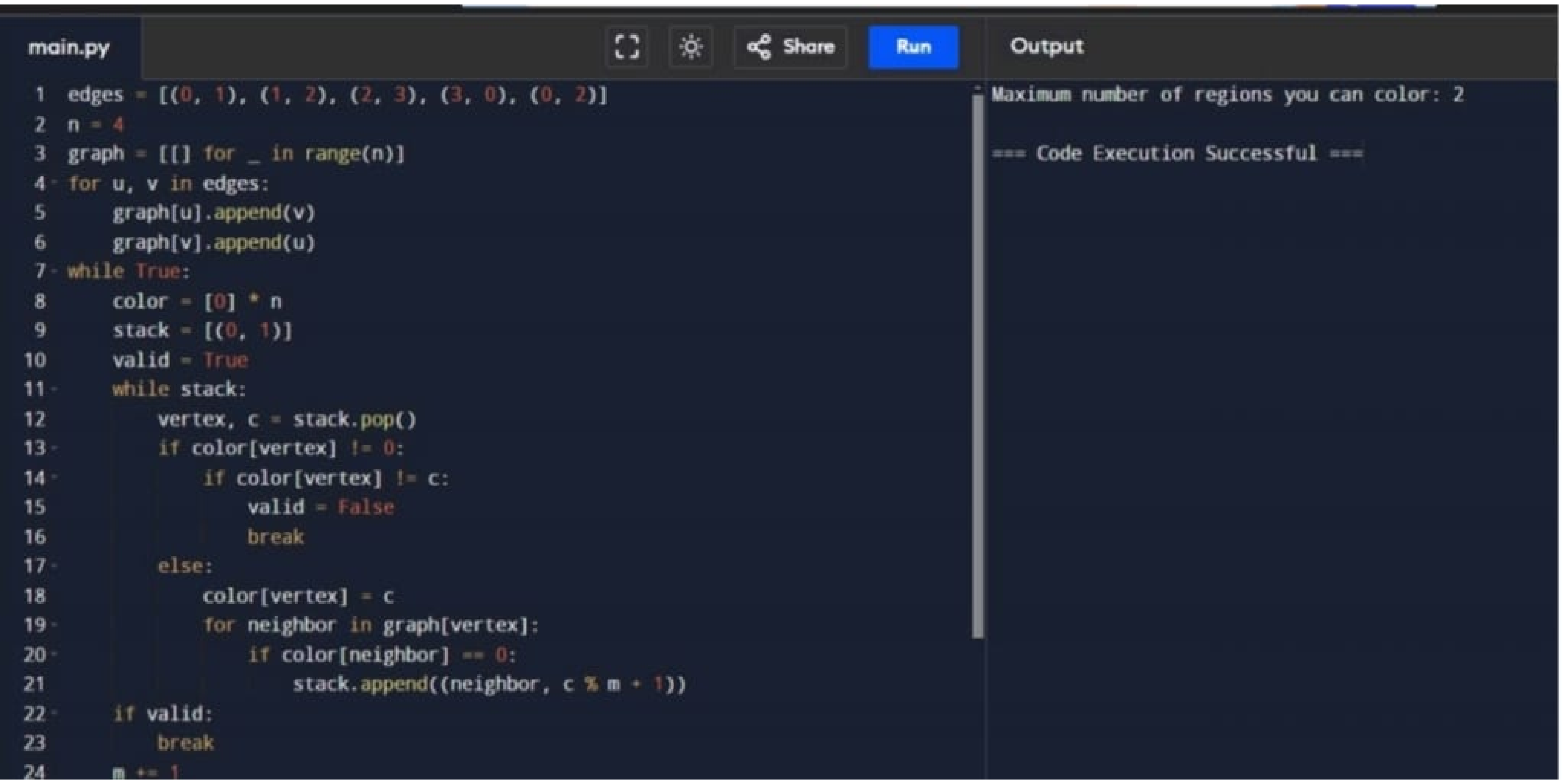


```
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n = 4
graph = [[] for _ in range(n)]
for u, v in edges:
    graph[u].append(v)
    graph[v].append(u)
path = [-1] * n
path[0] = 0
def can_find_hamiltonian_cycle(pos):
    if pos == n:
        return path[0] in graph[path[-1]]
    for v in range(1, n):
        if v not in path and path[pos - 1] in graph[v]:
            path[pos] = v
            if can_find_hamiltonian_cycle(pos + 1):
                return True
            path[pos] = -1
    return False
hamiltonian_cycle_exists = can_find_hamiltonian_cycle(1)
print(f"Hamiltonian cycle exists: {hamiltonian_cycle_exists}")
```

```
Hamiltonian cycle exists: True

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

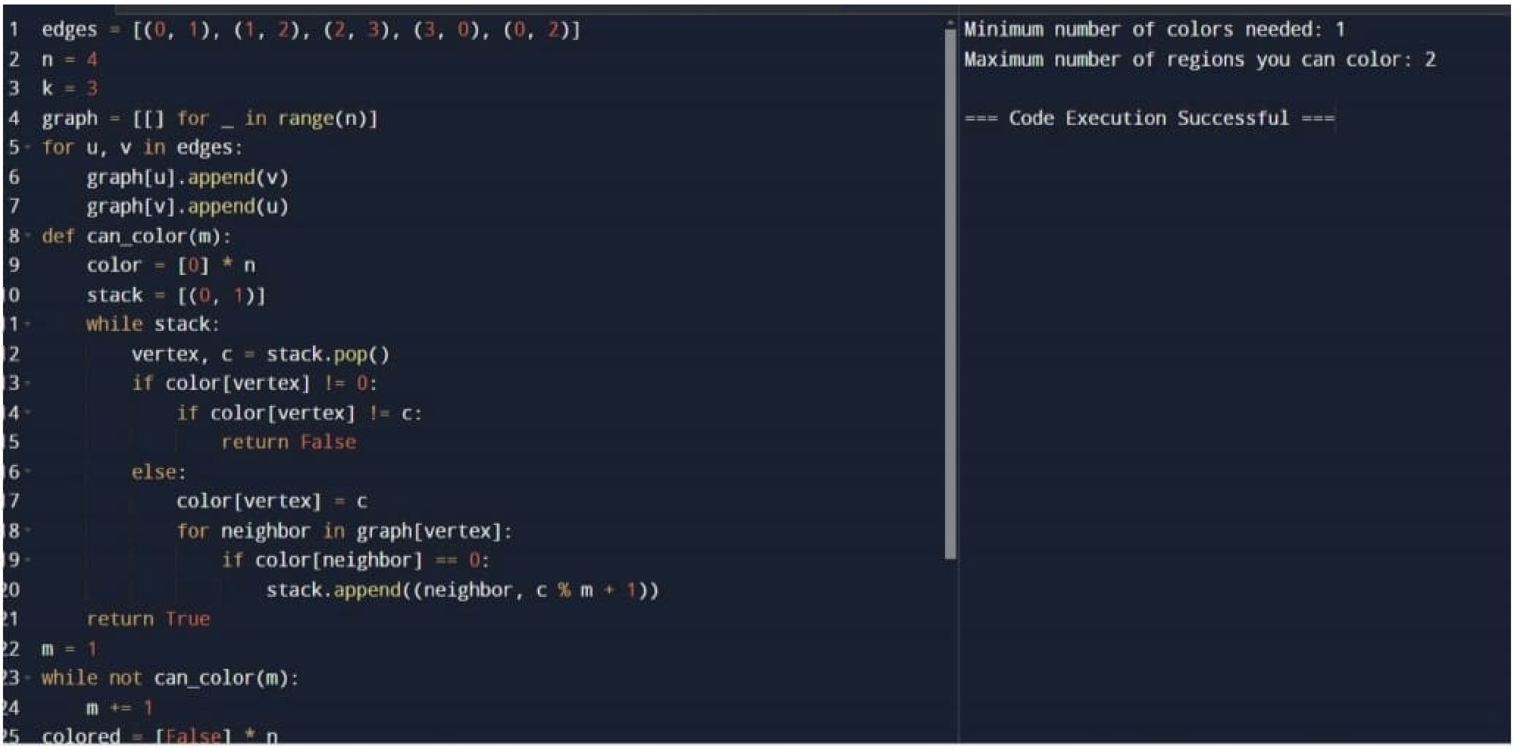


```
colored = [False] * n
your_colored_count = 0
turn = 0
while not all(colored):
    for i in range(n):
        if not colored[i]:
            if turn % 3 == 0:  # Your turn
                colored[i] = True
                your_colored_count += 1
            else:
                colored[i] = True
            turn += 1
print(f"Maximum number of regions you can color: {your_colored_count}")
```



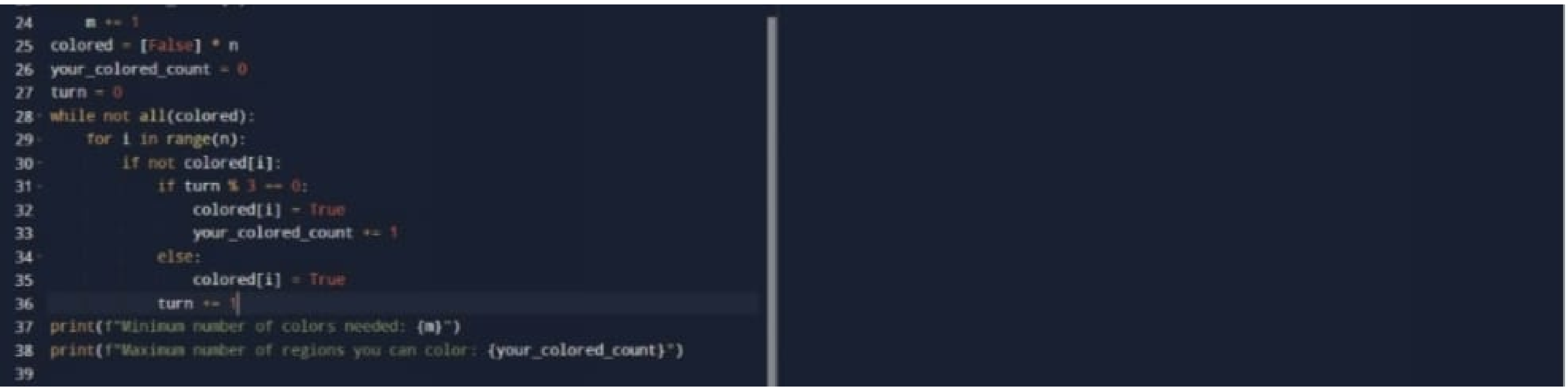
```python
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n = 4
graph = [[] for _ in range(n)]
for u, v in edges:
    graph[u].append(v)
    graph[v].append(u)
while True:
    color = [0] * n
    stack = [(0, 1)]
    valid = True
    while stack:
        vertex, c = stack.pop()
        if color[vertex] != 0:
            if color[vertex] != c:
                valid = False
                break
        else:
            color[vertex] = c
            for neighbor in graph[vertex]:
                if color[neighbor] == 0:
                    stack.append((neighbor, c % m + 1))
    if valid:
        break
    m += 1
```

```
Maximum number of regions you can color: 2

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

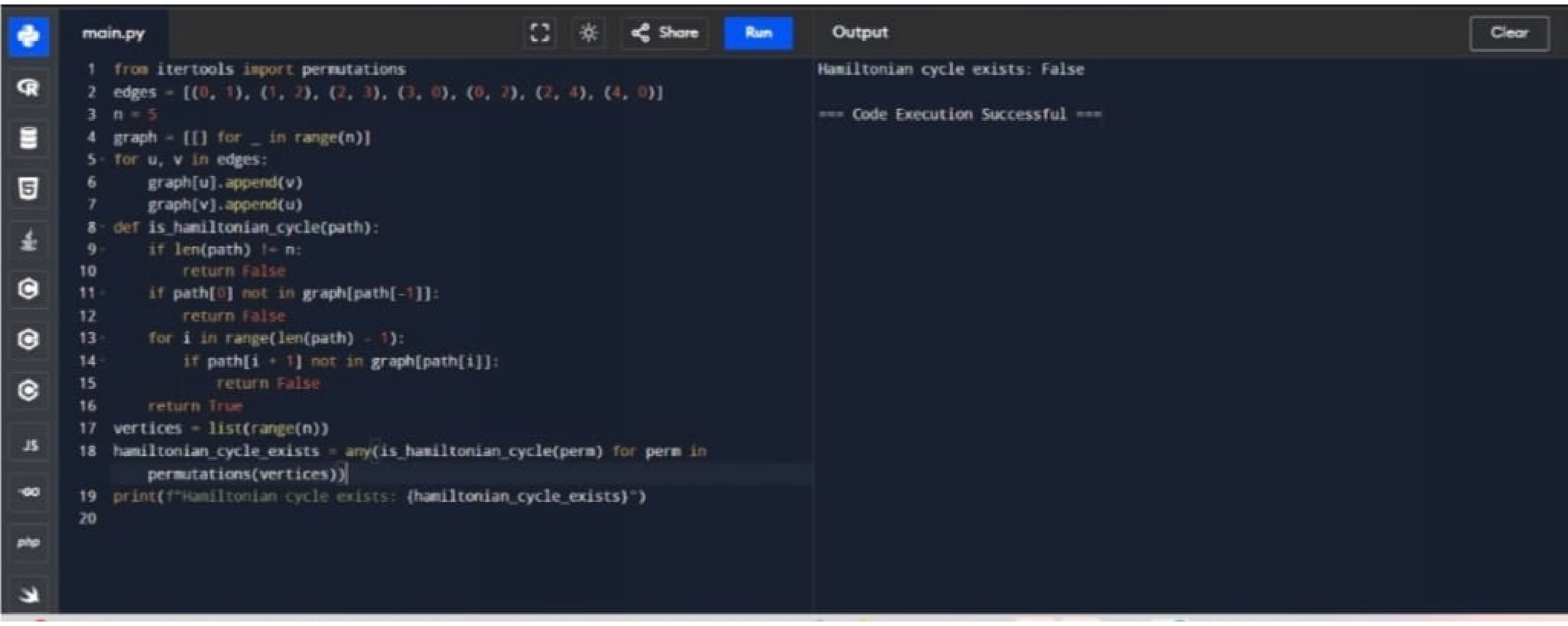


```python
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n = 4
k = 3
graph = [[] for _ in range(n)]
for u, v in edges:
    graph[u].append(v)
    graph[v].append(u)
def can_color(m):
    color = [0] * n
    stack = [(0, 1)]
    while stack:
        vertex, c = stack.pop()
        if color[vertex] != 0:
            if color[vertex] != c:
                return False
        else:
            color[vertex] = c
            for neighbor in graph[vertex]:
                if color[neighbor] == 0:
                    stack.append((neighbor, c % m + 1))
    return True
m = 1
while not can_color(m):
    m += 1
colored = [False] * n
```

```
Minimum number of colors needed: 1
Maximum number of regions you can color: 2

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

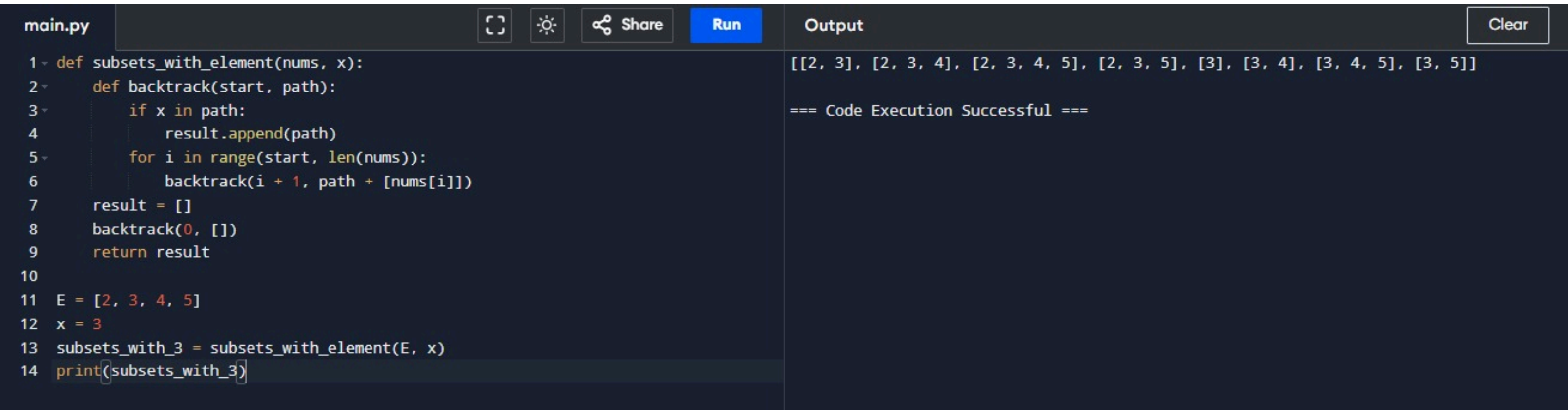


```
    m += 1
colored = [False] * n
your_colored_count = 0
turn = 0
while not all(colored):
    for i in range(n):
        if not colored[i]:
            if turn % 3 == 0:
                colored[i] = True
                your_colored_count += 1
            else:
                colored[i] = True
            turn += 1
print(f"Minimum number of colors needed: {m}")
print(f"Maximum number of regions you can color: {your_colored_count}")
```



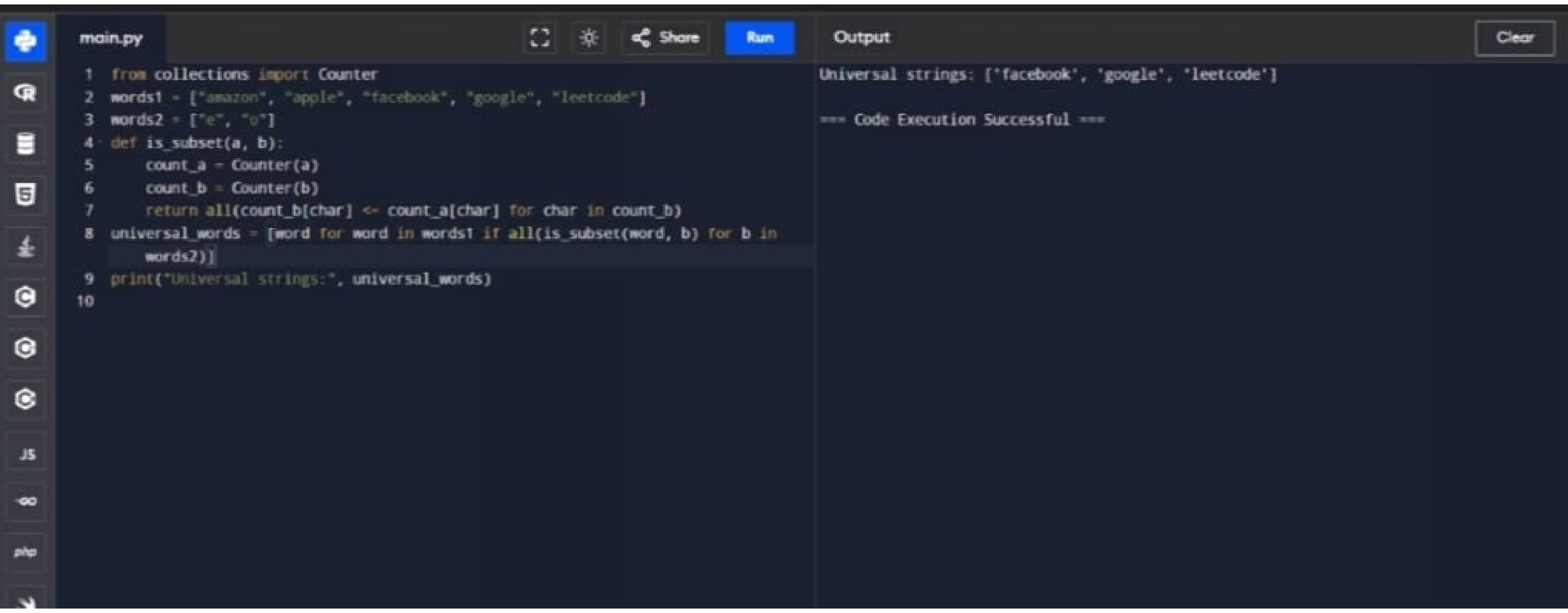
```python
from itertools import permutations
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2), (2, 4), (4, 0)]
n = 5
graph = [[] for _ in range(n)]
for u, v in edges:
    graph[u].append(v)
    graph[v].append(u)
def is_hamiltonian_cycle(path):
    if len(path) != n:
        return False
    if path[0] not in graph[path[-1]]:
        return False
    for i in range(len(path) - 1):
        if path[i + 1] not in graph[path[i]]:
            return False
    return True
vertices = list(range(n))
hamiltonian_cycle_exists = any(is_hamiltonian_cycle(perm) for perm in
    permutations(vertices))
print(f"Hamiltonian cycle exists: {hamiltonian_cycle_exists}")
```

```
Hamiltonian cycle exists: False

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



```python
def subsets_with_element(nums, x):
    def backtrack(start, path):
        if x in path:
            result.append(path)
        for i in range(start, len(nums)):
            backtrack(i + 1, path + [nums[i]])
    result = []
    backtrack(0, [])
    return result

E = [2, 3, 4, 5]
x = 3
subsets_with_3 = subsets_with_element(E, x)
print(subsets_with_3)
```

```
[[2, 3], [2, 3, 4], [2, 3, 4, 5], [2, 3, 5], [3], [3, 4], [3, 4, 5], [3, 5]]

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



```python
from collections import Counter
words1 = ["amazon", "apple", "facebook", "google", "leetcode"]
words2 = ["e", "o"]
def is_subset(a, b):
    count_a = Counter(a)
    count_b = Counter(b)
    return all(count_b[char] <= count_a[char] for char in count_b)
universal_words = [word for word in words1 if all(is_subset(word, b) for b in words2)]
print("Universal strings:", universal_words)
```

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

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