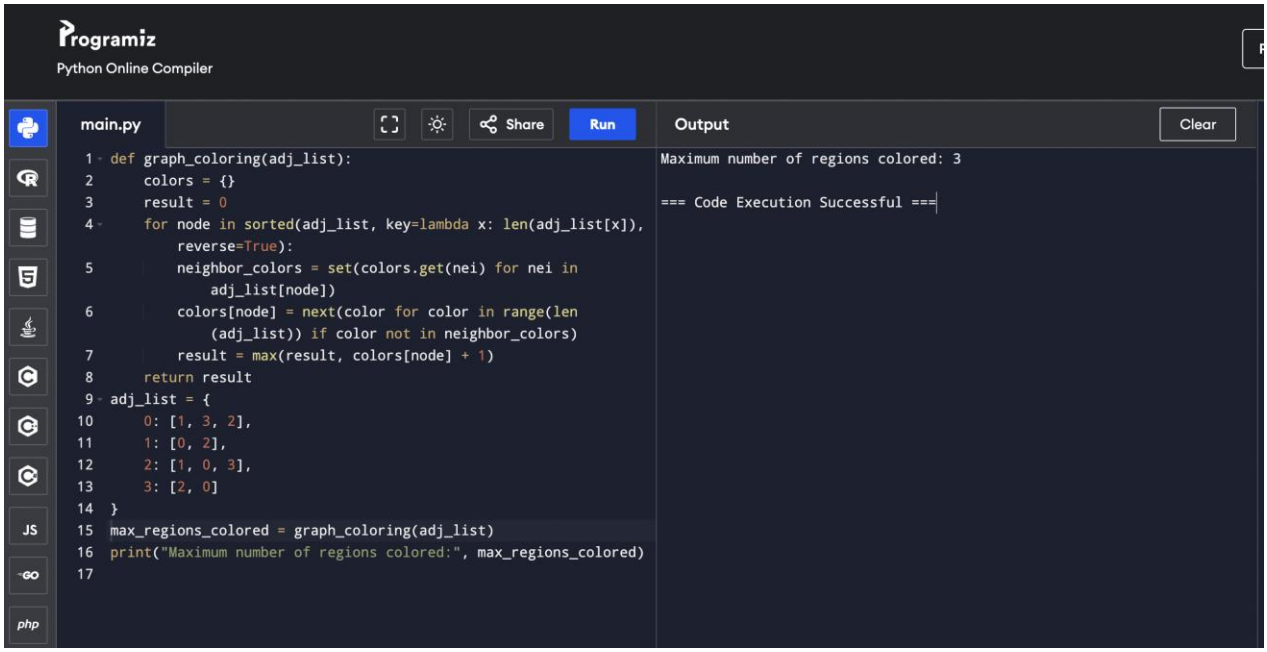
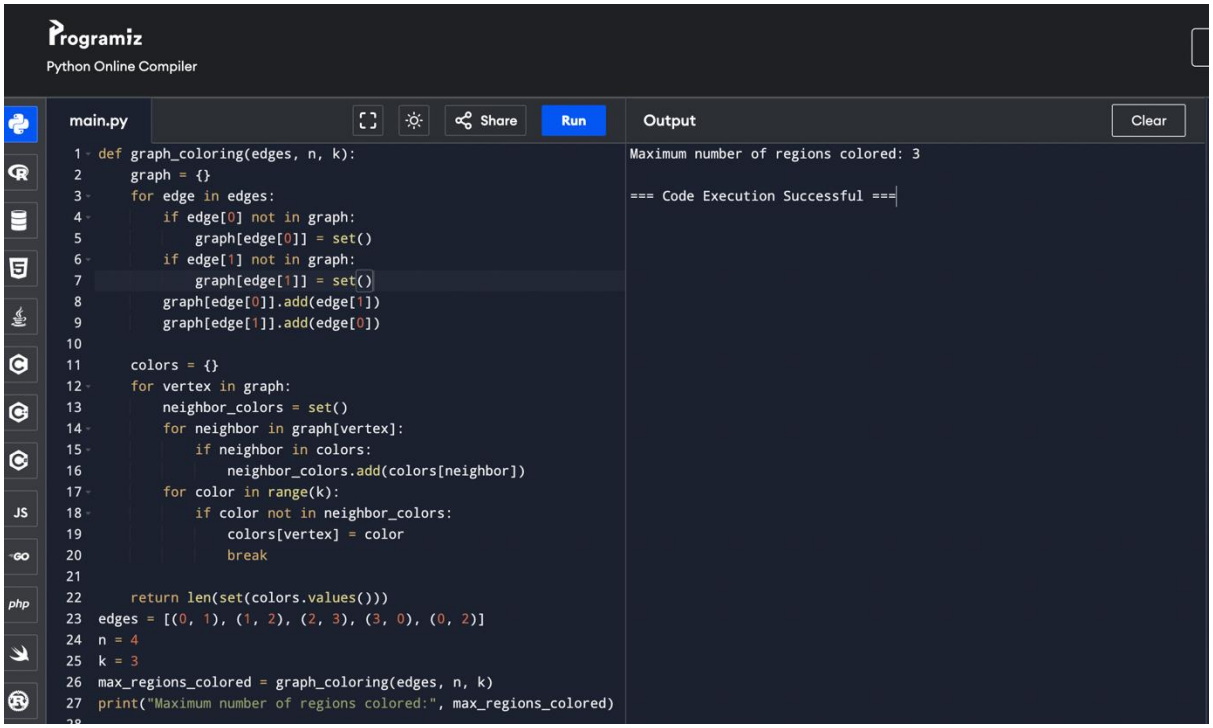
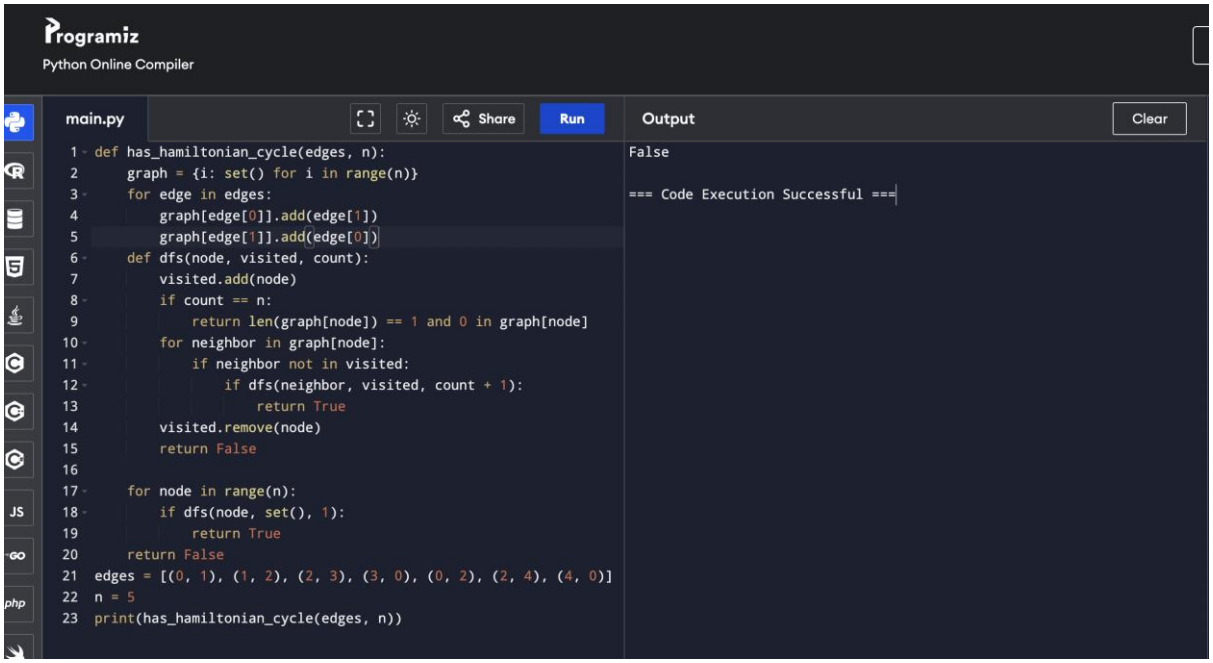
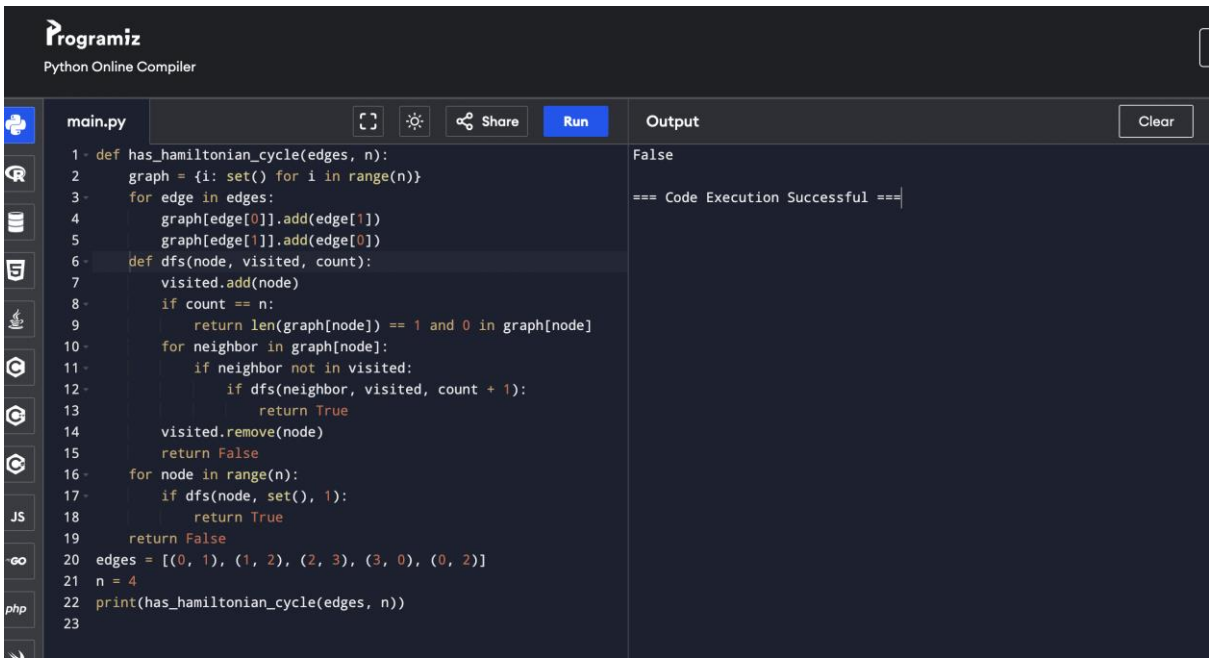

Programiz
Python Online Compiler
main.py
Share
Run
Output
Clear
def graph_coloring(adj_list):
colors = {}
result = 0
for node in sorted(adj_list, key=lambda x: len(adj_list[x]), reverse=True):
neighbor_colors = set(colors.get(nei) for nei in adj_list[node])
colors[node] = next(color for color in range(len(adj_list)) if color not in neighbor_colors)
result = max(result, colors[node] + 1)
return result
adj_list = {
0: [1, 3, 2],
1: [0, 2],
2: [1, 0, 3],
3: [2, 0]
}
max_regions_colored = graph_coloring(adj_list)
print("Maximum number of regions colored:", max_regions_colored)
Maximum number of regions colored: 3
=== Code Execution Successful ===


Programiz
Python Online Compiler
main.py
Share
Run
Output
Clear
def graph_coloring(edges, n, k):
graph = {}
for edge in edges:
if edge[0] not in graph:
graph[edge[0]] = set()
if edge[1] not in graph:
graph[edge[1]] = set()
graph[edge[0]].add(edge[1])
graph[edge[1]].add(edge[0])
colors = {}
for vertex in graph:
neighbor_colors = set()
for neighbor in graph[vertex]:
if neighbor in colors:
neighbor_colors.add(colors[neighbor])
for color in range(k):
if color not in neighbor_colors:
colors[vertex] = color
break
return len(set(colors.values()))
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n = 4
k = 3
max_regions_colored = graph_coloring(edges, n, k)
print("Maximum number of regions colored:", max_regions_colored)
Maximum number of regions colored: 3
=== Code Execution Successful ===


Programiz
Python Online Compiler
main.py
Share
Run
Output
Clear
def has_hamiltonian_cycle(edges, n):
graph = {i: set() for i in range(n)}
for edge in edges:
graph[edge[0]].add(edge[1])
graph[edge[1]].add(edge[0])
def dfs(node, visited, count):
visited.add(node)
if count == n:
return len(graph[node]) == 1 and 0 in graph[node]
for neighbor in graph[node]:
if neighbor not in visited:
if dfs(neighbor, visited, count + 1):
return True
visited.remove(node)
return False
for node in range(n):
if dfs(node, set(), 1):
return True
return False
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2), (2, 4), (4, 0)]
n = 5
print(has_hamiltonian_cycle(edges, n))
False
=== Code Execution Successful ===


Programiz
Python Online Compiler
main.py
Share
Run
Output
Clear
def has_hamiltonian_cycle(edges, n):
graph = {i: set() for i in range(n)}
for edge in edges:
graph[edge[0]].add(edge[1])
graph[edge[1]].add(edge[0])
def dfs(node, visited, count):
visited.add(node)
if count == n:
return len(graph[node]) == 1 and 0 in graph[node]
for neighbor in graph[node]:
if neighbor not in visited:
if dfs(neighbor, visited, count + 1):
return True
visited.remove(node)
return False
for node in range(n):
if dfs(node, set(), 1):
return True
return False
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n = 4
print(has_hamiltonian_cycle(edges, n))
False
=== Code Execution Successful ===

Programiz
Python Online Compiler

main.py

Share Run

```python
def subsets(S):
    S = sorted(set(S))
    result = []
    def backtrack(start, path):
        result.append(path)
        for i in range(start, len(S)):
            backtrack(i + 1, path + [S[i]])
    backtrack(0, [])
    return result
A = [1, 2, 3]
print(subsets(A))

```

Output

Clear

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

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