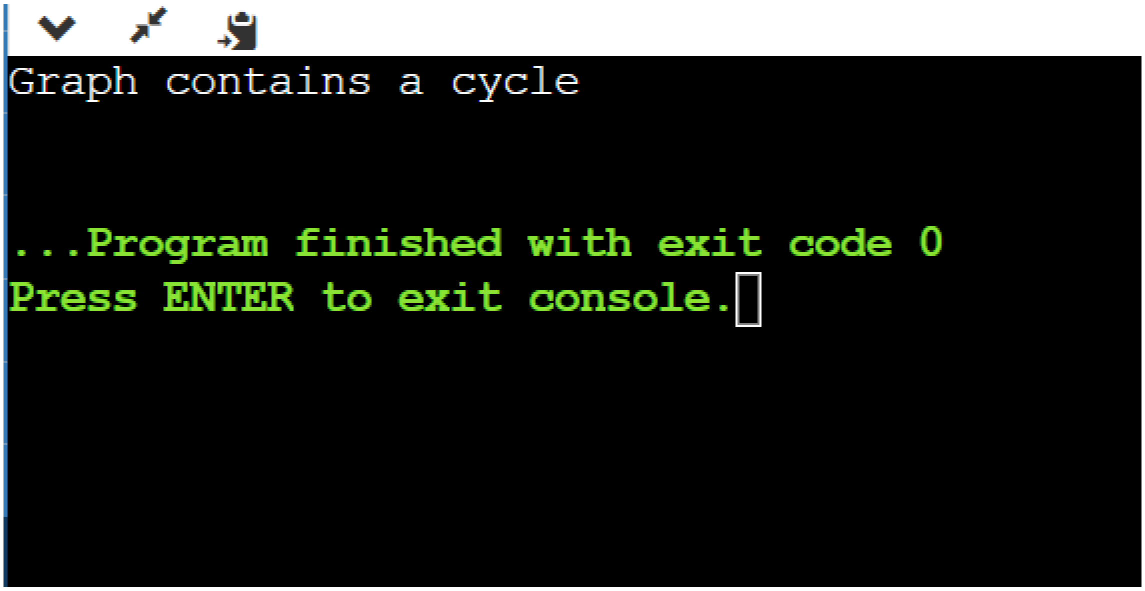
CODE:

```
from collections import defaultdict, deque

class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adjacency_list = defaultdict(list)

    def add_edge(self, u, v):
        self.adjacency_list[u].append(v)

    def topological_sort(self):
        in_degree = [0] * self.vertices
        for vertex in range(self.vertices):
            for neighbor in self.adjacency_list[vertex]:
                in_degree[neighbor] += 1

        queue = deque()
        for vertex in range(self.vertices):
            if in_degree[vertex] == 0:
                queue.append(vertex)

        visited_count = 0
        while queue:
            current_vertex = queue.popleft()
            visited_count += 1

            for neighbor in self.adjacency_list[current_vertex]:
                in_degree[neighbor] -= 1
                if in_degree[neighbor] == 0:
                    queue.append(neighbor)

        return visited_count == self.vertices  # True if DAG, False if cycle

# Example usage
g = Graph(6)
g.add_edge(0, 1)
g.add_edge(1, 2)
g.add_edge(2, 3)
```

```
g.add_edge(3, 4)
g.add_edge(4, 1)  # Introducing a cycle

if g.topological_sort():
    print("Graph is a DAG (no cycle)")
else:
    print("Graph contains a cycle")
```



```
Graph contains a cycle


...Program finished with exit code 0
Press ENTER to exit console.
```