Computing Algorithms 12th Week Project

- Done By :
- Abdalrahman Sherif Ibrahim – 20107171
- Nadine Mohammed – 20107088
- Sara Ashraf - 20105551

Project Idea: (6)

- Project 6 Write a code that constructs a graph. After creating the graph, the code should check whether the graph is: a. Connected. b. acyclic. **The above check will be done using Depth-First-Search (DFS) and implemented by both adjacency matrix and adjacency list.**

Rubric

		Subject	Maximum mark	Actual mark
Project Assessment		Presentation	2	
		Hardcopy	2	
		The code is implemented using both adjacency matrix and adjacency list, and the check is implemented properly	4	
		Data entry is implemented properly	2	
			5	
			5	
			5	
			5	

Objectives:

Explain the difference between Acyclic and Cyclic graphs.

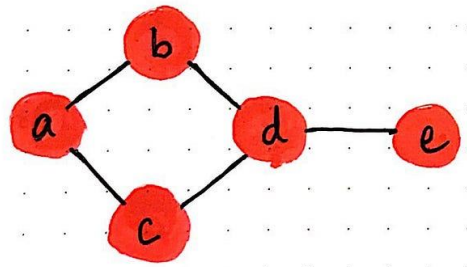
Illustrate the difference between connected and disconnected graphs.

Explain the code implementation and demonstrate graph visualization.

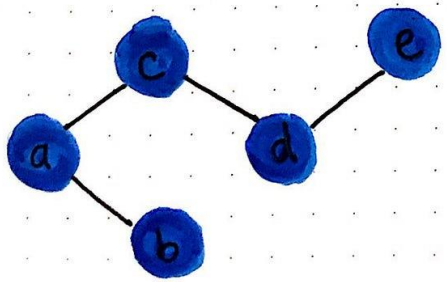
Present and discuss different inputs to showcase the differences.

What is the difference between Cyclic & Acyclic graphs?

- A graph that contains at least one cycle is known as a cyclic graph. Conversely, a graph that contains zero cycles is known as an acyclic graph.

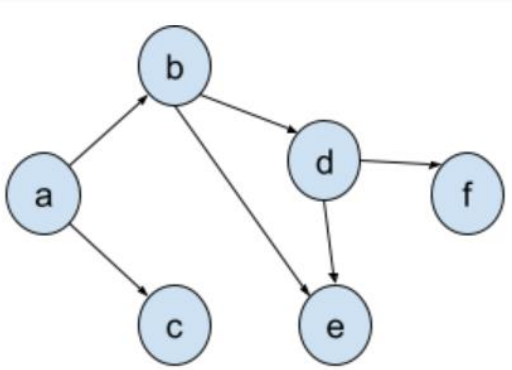


A graph with at least one cycle is known as a cyclic graph.

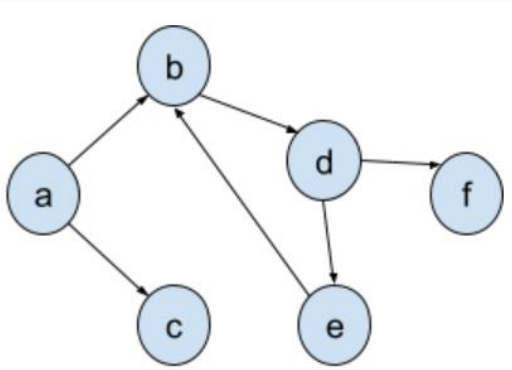


A graph with no cycles in it is known as an acyclic graph.

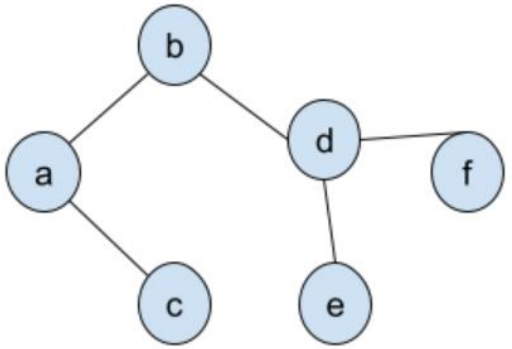
Additional examples for illustration.



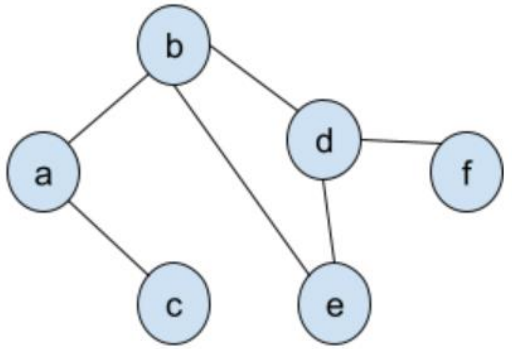
Acyclic (Directed) Graph



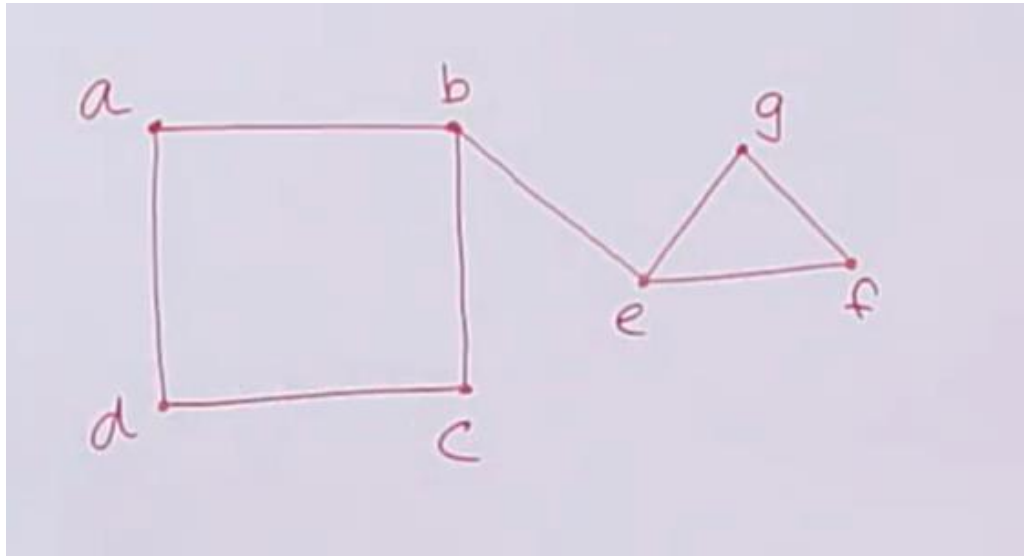
Cyclic (Directed) Graph

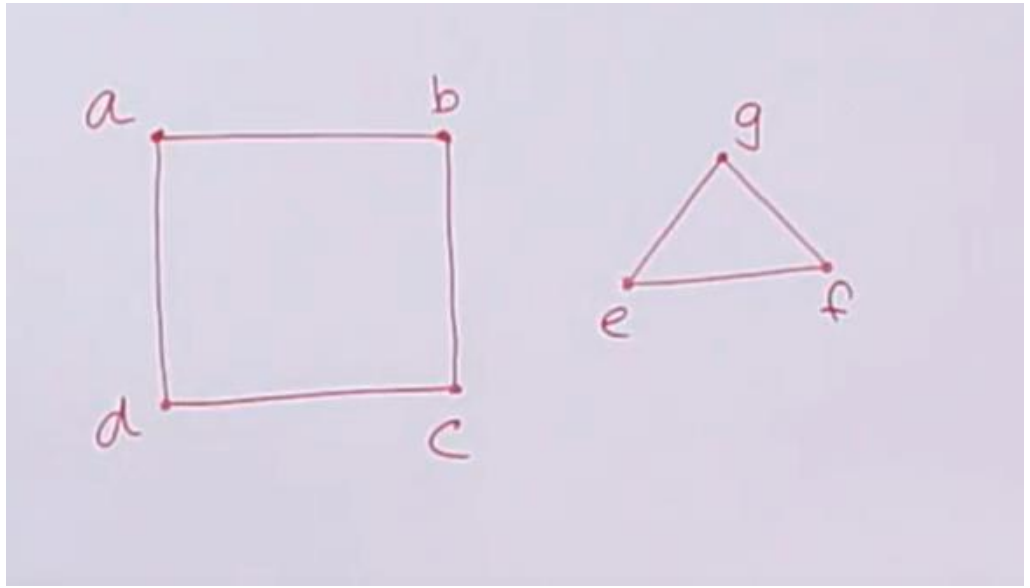


Acyclic (Undirected) Graph



Cyclic (Undirected) Graph

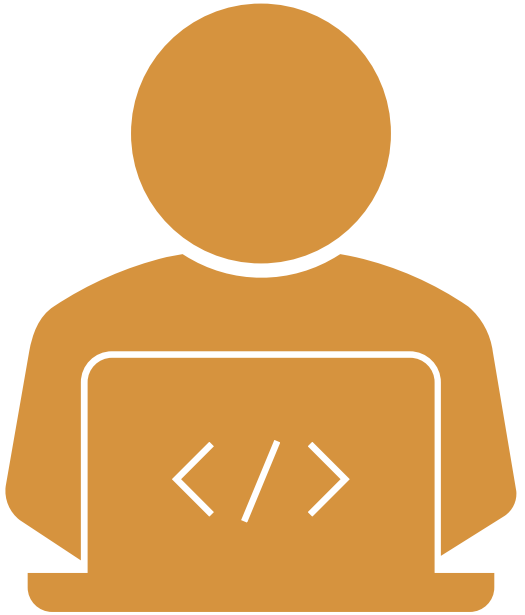




2. Illustrate the difference between connected & disconnected graphs.

- A graph is said to be connected graph if there is a path between every pair of vertex. From every vertex to any other vertex there must be some path to traverse. This is called the connectivity of a graph. A graph is said to be disconnected, if there exists multiple disconnected vertices and edges.

3. Code Implementation (step-by-step) using Python.



```python
import networkx as nx
import matplotlib.pyplot as plt

class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.adjList = [[] for _ in range(self.V)]
```

1. Import libraries
2. Define the graph class
3. Initialize the graph.

Importing libraries , Defining & initializing the graph class.

- 4. Add edges to the graph
- 5. Check if the graph is connected
- 6. Check if the graph is acyclic

Complexity of:Adding edges: O(1)
isConnected: O(V + E)
isAcyclic: O(V + E)

```
    def addEdge(self, src, dest):
        self.adjList[src].append(dest)
        self.adjList[dest].append(src)

    def isConnected(self):
        visited = [False] * self.V
        self.dfs(0, visited)
        return all(visited)

    def isAcyclic(self):
        visited = [False] * self.V
        for v in range(self.V):
            if not visited[v] and self.isCyclicUtil(v, visited, -1):
                return False
        return True
```

- 7. Performing Depth-First Search (DFS):

8.Checking for Cycles using DFS:

9.Visualizing the Graph:

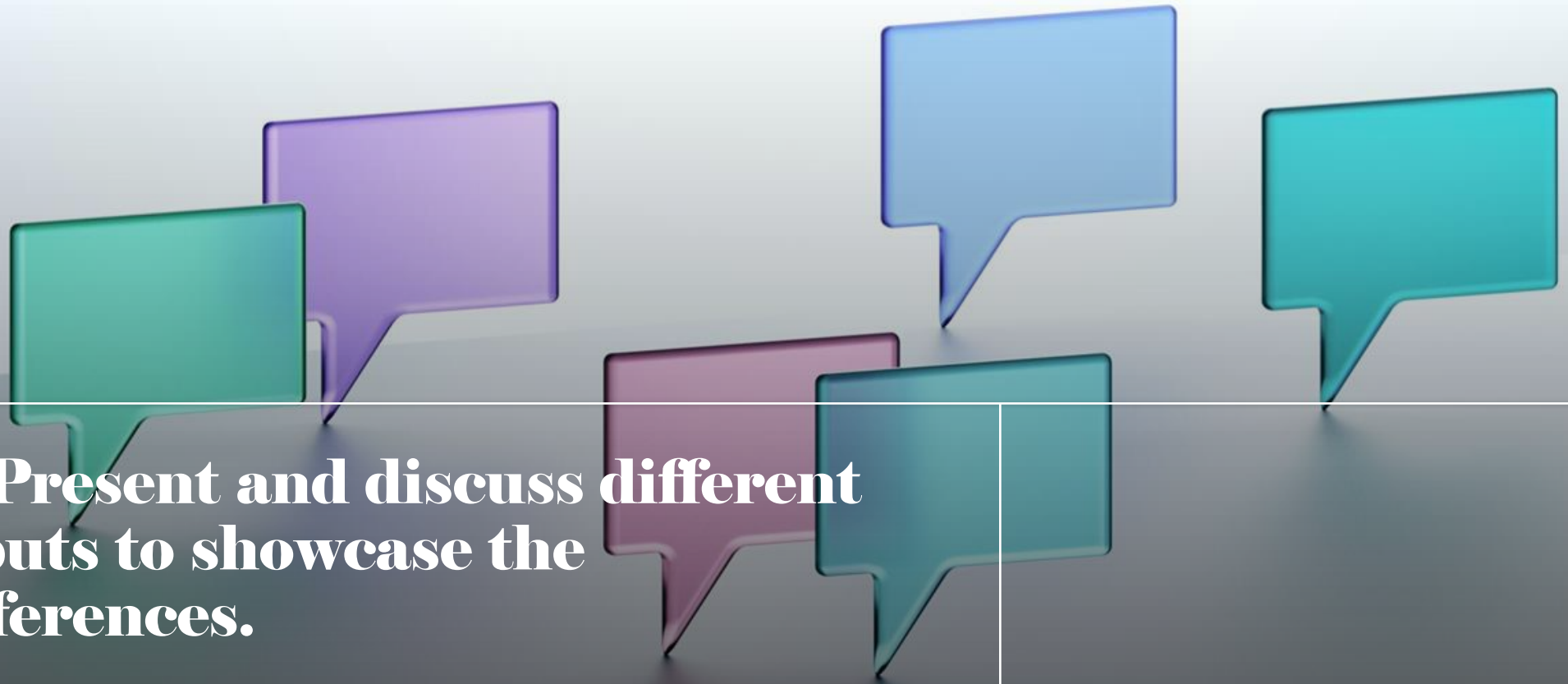
```
    def dfs(self, v, visited):
        visited[v] = True
        for i in self.adjList[v]:
            if not visited[i]:
                self.dfs(i, visited)

    def isCyclicUtil(self, v, visited, parent):
        visited[v] = True
        for i in self.adjList[v]:
            if not visited[i]:
                if self.isCyclicUtil(i, visited, v):
                    return True
            elif i != parent:
                return True
        return False

    def visualize(self):
        G = nx.Graph()
        for v in range(self.V):
            G.add_node(v)
            for i in self.adjList[v]:
                G.add_edge(v, i)
        pos = nx.spring_layout(G)
        nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=500, edge_color='black', linewidths=1)
        plt.show()
```

Main function

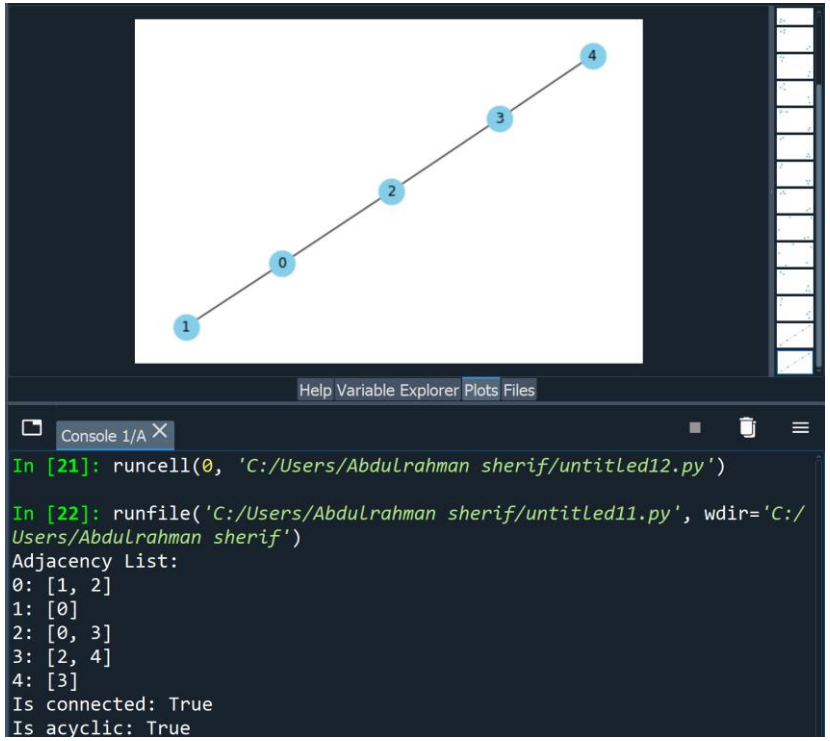
```
51  if __name__ == '__main__':
52      graph = Graph(5)
53      graph.addEdge(0, 1)
54      graph.addEdge(0, 2)
55      graph.addEdge(2, 3)
56      graph.addEdge(3, 4)
57
58      print("Adjacency List:")
59      for i in range(graph.V):
60          print(f"{i}: {graph.adjList[i]}")
61
62      print("Is connected:", graph.isConnected())
63      print("Is acyclic:", graph.isAcyclic())
64
65      graph.visualize()
66
```



4. Present and discuss different inputs to showcase the differences.

In that case the graph is acylic & connected.

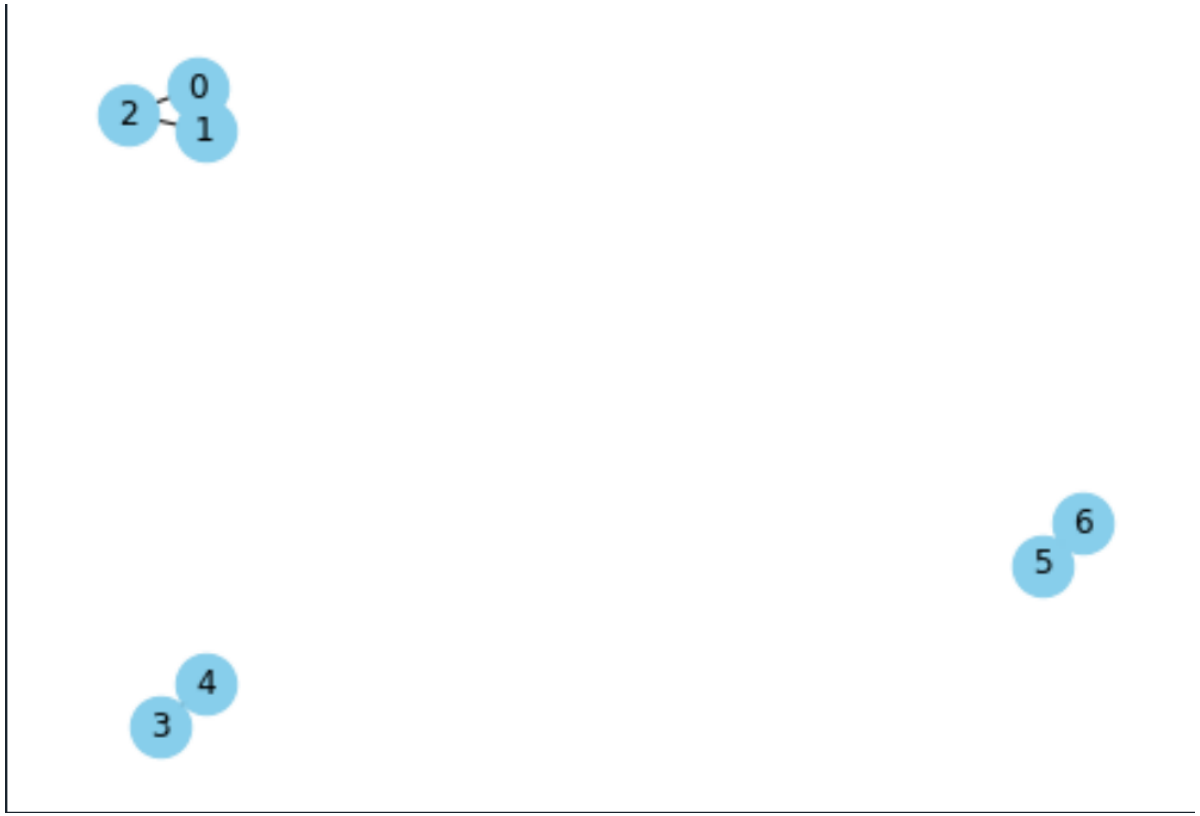
1




```
if __name__ == '__main__':
    graph = Graph(5)
    graph.addEdge(0, 1)
    graph.addEdge(0, 2)
    graph.addEdge(2, 3)
    graph.addEdge(3, 4)
```

In that case the graph is cyclic & disconnected.

2




```
Adjacency List:
0: [1, 2]
1: [0, 2]
2: [0, 1]
3: [4]
4: [3]
5: [6]
6: [5]
Is connected: False
Is acyclic: False
```

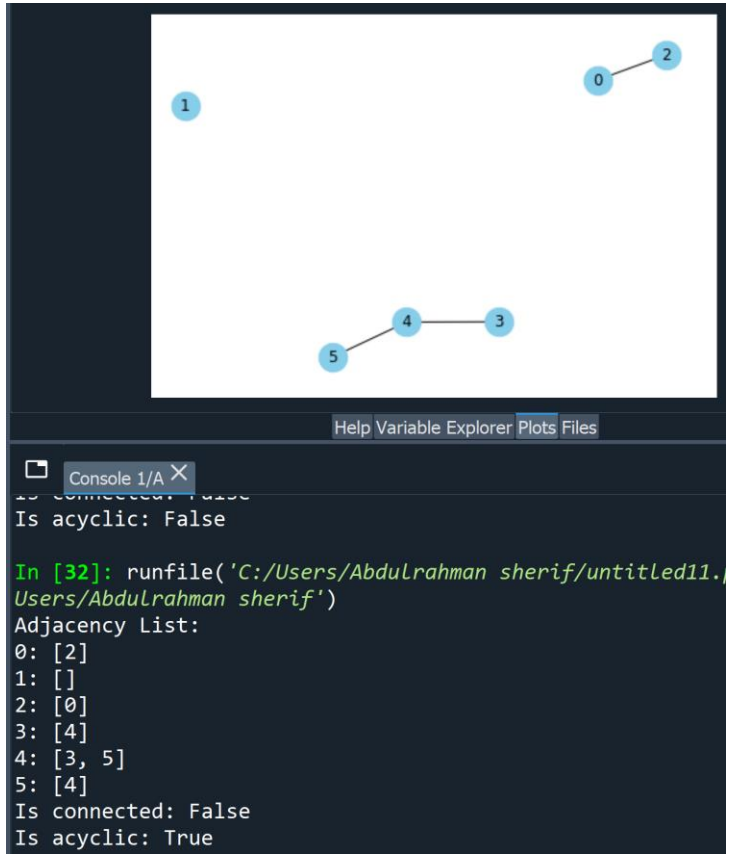
```
if __name__ == '__main__':

    graph = Graph(7)
    graph.addEdge(0, 1)
    graph.addEdge(0, 2)
    graph.addEdge(1, 2)

    graph.addEdge(3, 4)
    graph.addEdge(5, 6)
```

In that case the graph is acylic & disconnected.

3



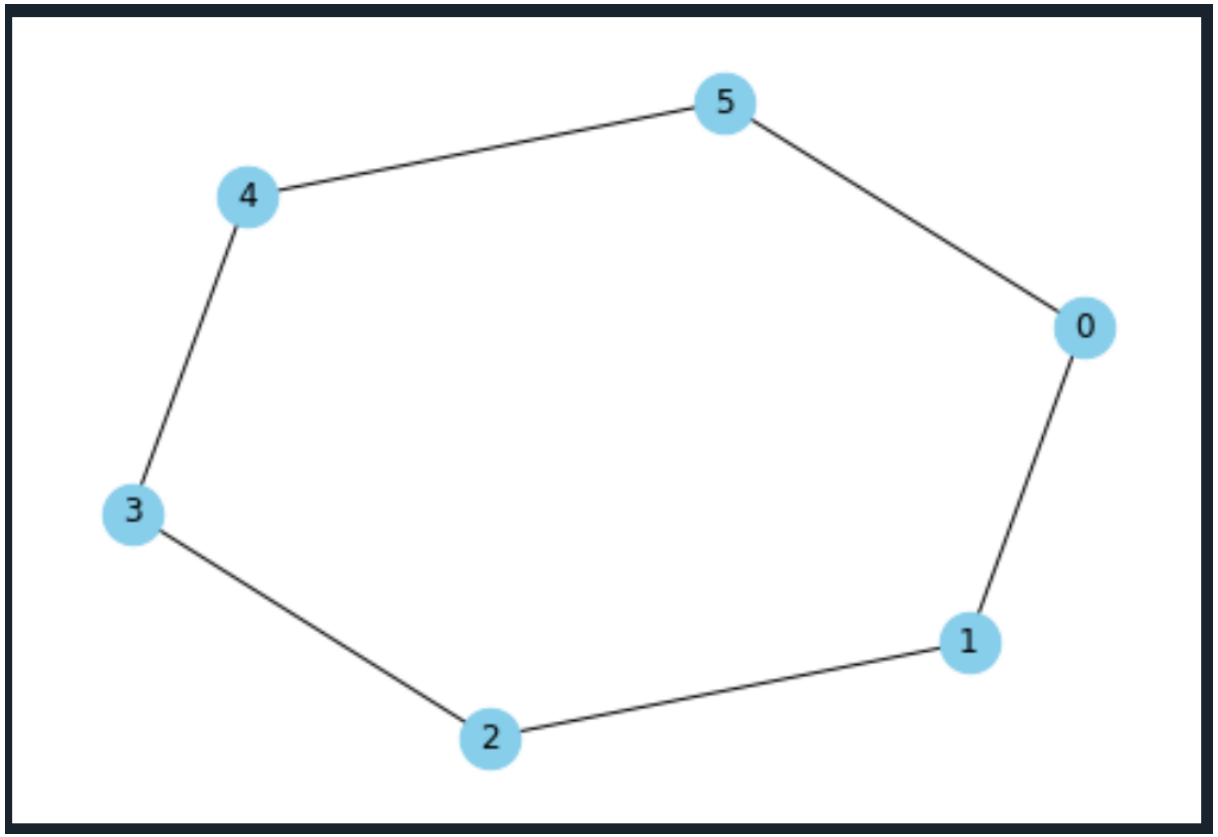

```
if __name__ == '__main__':

    graph = Graph(6)
    graph.addEdge(0, 2)
    graph.addEdge(3, 4)

    graph.addEdge(4, 5)
```

In that case the graph is Cyclic & connected.

4

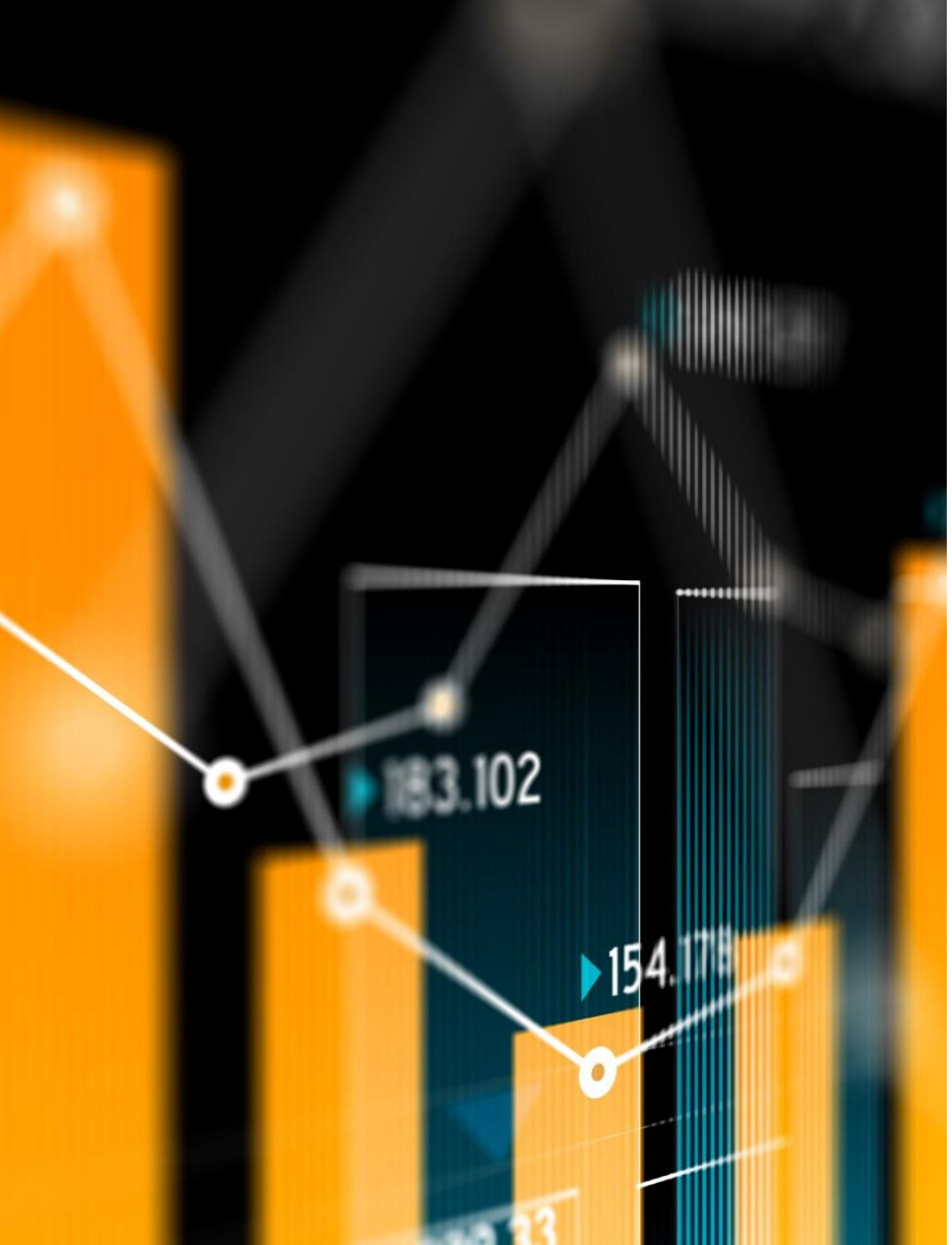



```python
if __name__ == '__main__':

    graph = Graph(6)
    graph.addEdge(0, 1)
    graph.addEdge(1, 2)
    graph.addEdge(2, 3)

    graph.addEdge(3, 4)
    graph.addEdge(4, 5)
    graph.addEdge(5, 0)
```

```
Adjacency List:
0: [1, 5]
1: [0, 2]
2: [1, 3]
3: [2, 4]
4: [3, 5]
5: [4, 0]
Is connected: True
Is acyclic: False
```



Conclusion

In this project, we learned about graphs and their properties. We implemented code to determine if a graph is cyclic or acyclic, as well as connected or disconnected. The visualization feature helped us better understand the graphs by displaying them visually. Overall, this project provided an introduction to graph theory concepts and allowed us to explore different types of graphs.