REPORT:

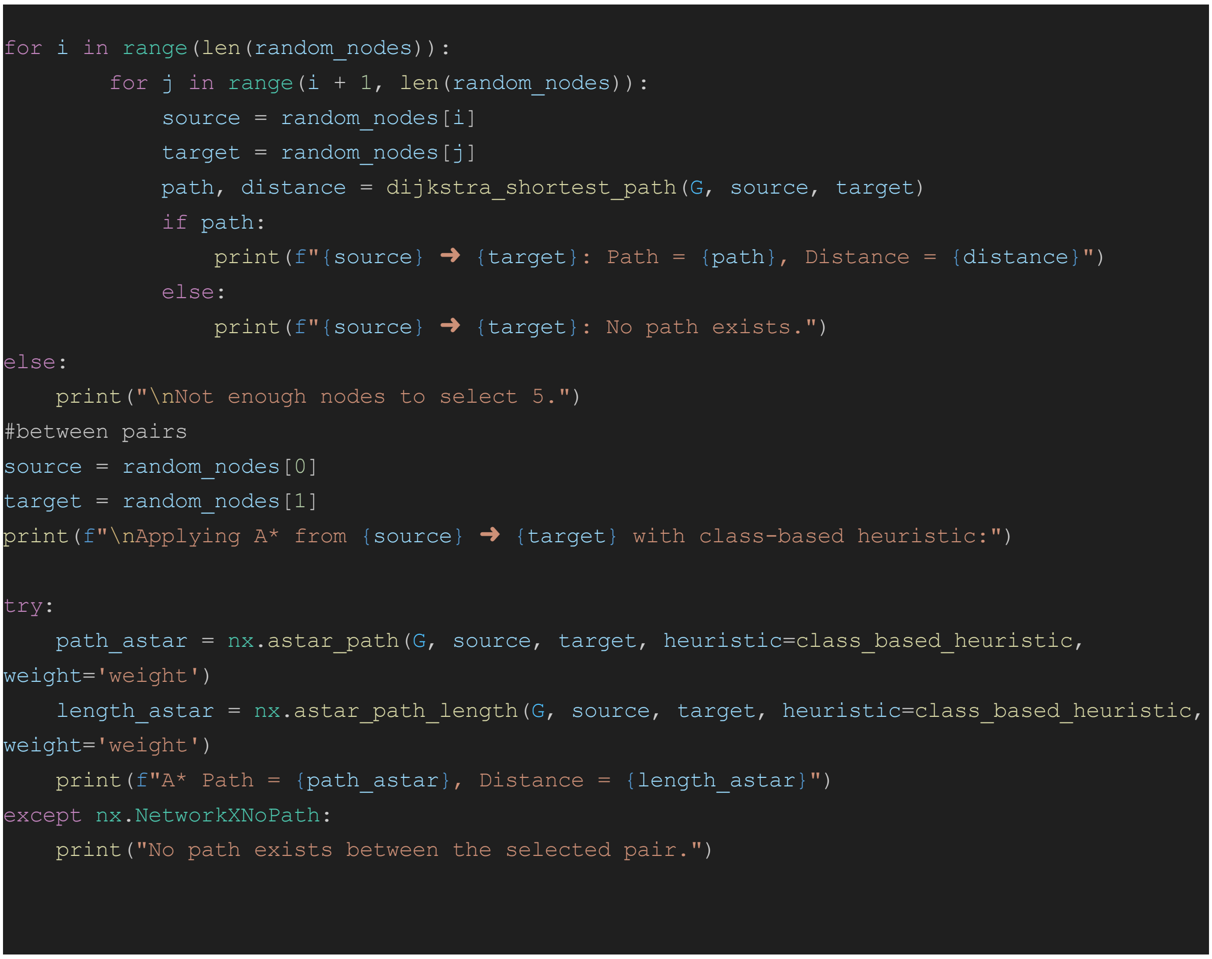
1] Network generation logic:

```
G = nx.Graph()

#  Step 1: Enter number of nodes and take node names
n = int(input("Enter number of nodes: "))
nodes = []

for i in range(n):
    node = input(f"Enter name of node {i+1}: ")
    nodes.append(node)

G.add_nodes_from(nodes)

#  Step 2: Enter number of edges and take edge pairs
e = int(input("Enter number of edges: "))
edges = []

for i in range(e):
    u = input(f"Enter start_node of edge {i+1}: ")
    v = input(f"Enter end_node of edge {i+1}: ")
    w = float(input(f"Enter weight({u},{v}):"))
    edges.append((u, v, {'weight':w}))

G.add_edges_from(edges)
```

2]

```
visited = {node: False for node in G
Friends_group = []

for node in G:
    if not visited[node]:
        group = []
        dfs_collect(node, visited, G.adj, group)
        Friends_group.append(group)
print(f"Number of Friends group :{len(Friends_group)}")

#number of people in the friends group and names of the people
for i, group in enumerate(Friends_group):
    print(f"Friends group {i+1}: {group}")
    print(f"     Size of group = {len(group)}")

#group with max and min people
     sizes = [len(group) for group in Friends_group]
max_size = max(sizes)
min_size = min(sizes)

print(f"\nMaximum size of a friends group: {max_size}")
print(f"Minimum size of a friends group: {min_size}")
```

3]sample shortest path:



```
for i in range(len(random_nodes)):
        for j in range(i + 1, len(random_nodes)):
            source = random_nodes[i]
            target = random_nodes[j]
            path, distance = dijkstra_shortest_path(G, source, target)
            if path:
                print(f"{source} ➜ {target}: Path = {path}, Distance = {distance}")
            else:
                print(f"{source} ➜ {target}: No path exists.")
else:
    print("\nNot enough nodes to select 5.")
#between pairs
source = random_nodes[0]
target = random_nodes[1]
print(f"\nApplying A* from {source} ➜ {target} with class-based heuristic:")

try:
    path_astar = nx.astar_path(G, source, target, heuristic=class_based_heuristic,
weight='weight')
    length_astar = nx.astar_path_length(G, source, target, heuristic=class_based_heuristic,
weight='weight')
    print(f"A* Path = {path_astar}, Distance = {length_astar}")
except nx.NetworkXNoPath:
    print("No path exists between the selected pair.")
```

5] Final Reflection: final_reflection