Artificial Intelligence Lab# 2

Ques: 1 Define the following terms:

- **Regular Graph**
  A regular graph is a graph where each vertex has the same number of neighbors; i.e. every vertex has the same degree or valency.

- **Null Graph**
  A null graph is a graph which has no vertices, i.e. it is a graph of order zero.

- **Trivial Graph**
  A graph having only one vertex in it is called a trivial graph. It is the smallest possible graph.

- **Simple Graph**
  A graph having no self-loops and no parallel edges in it is called as a simple graph.

- **Connected Graph**
  A graph in which we can visit from any one vertex to any other vertex is called as a connected graph.

- **Disconnected Graph**
  A graph in which there does not exist any path between at least one pair of vertices is called as a disconnected graph.

- **Complete Graph**
  A graph in which exactly one edge is present between every pair of vertices is called as complete graph.

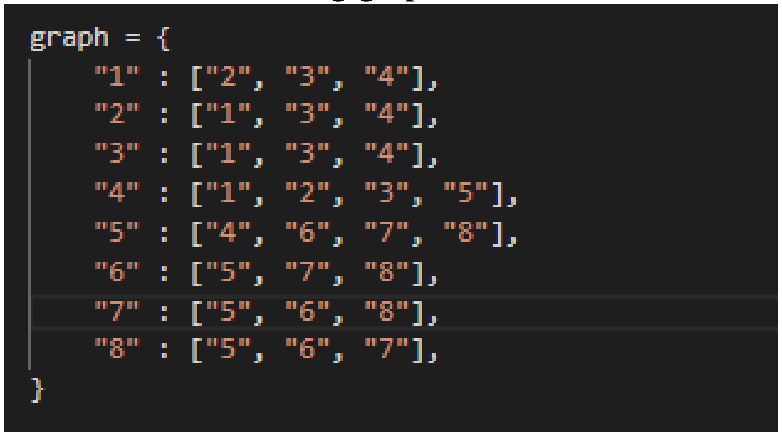
- **Cyclic Graph**
  A simple graph of 'n' vertices (n >= 3) and n edges forming a cycle of length 'n' with all its edges is called as cycle graph.

- **Degree of Vertex**
  The degree of vertex of a graph is the number of edges incident to the vertex.

- **Loop**
  A loop is an edge that connects a vertex to itself.

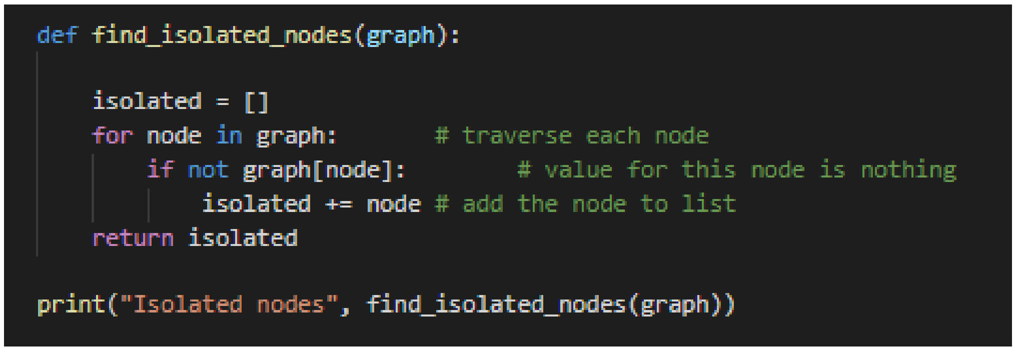
- **Parallel Edges**
  Parallel edges are two or more edges that are incident to the same two vertices.

Ques: 2 Consider the following graph:



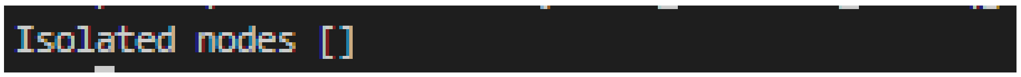
```
graph = {
    "1" : ["2", "3", "4"],
    "2" : ["1", "3", "4"],
    "3" : ["1", "3", "4"],
    "4" : ["1", "2", "3", "5"],
    "5" : ["4", "6", "7", "8"],
    "6" : ["5", "7", "8"],
    "7" : ["5", "6", "8"],
    "8" : ["5", "6", "7"],
}
```

a) Find isolated nodes
Code:



```
def find_isolated_nodes(graph):

    isolated = []
    for node in graph:       # traverse each node
        if not graph[node]:        # value for this node is nothing
            isolated += node # add the node to list
    return isolated

print("Isolated nodes", find_isolated_nodes(graph))
```

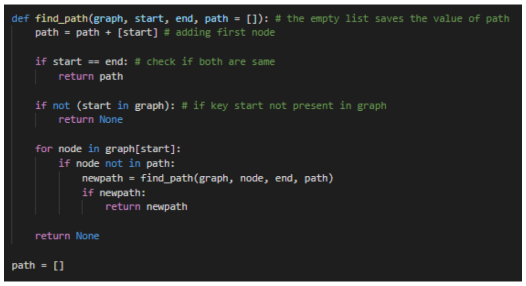
Answer:



```
Isolated nodes []
```

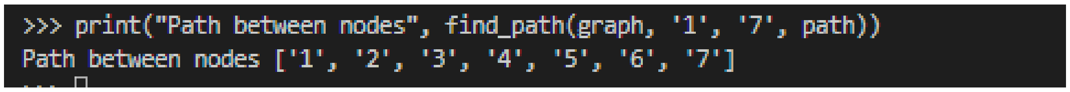
b) Find path between two vertex / node 1 and 7

Code:



```python
def find_path(graph, start, end, path = []): # the empty list saves the value of path
    path = path + [start] # adding first node

    if start == end: # check if both are same
        return path

    if not (start in graph): # if key start not present in graph
        return None

    for node in graph[start]:
        if node not in path:
            newpath = find_path(graph, node, end, path)
            if newpath:
                return newpath

    return None

path = []
```

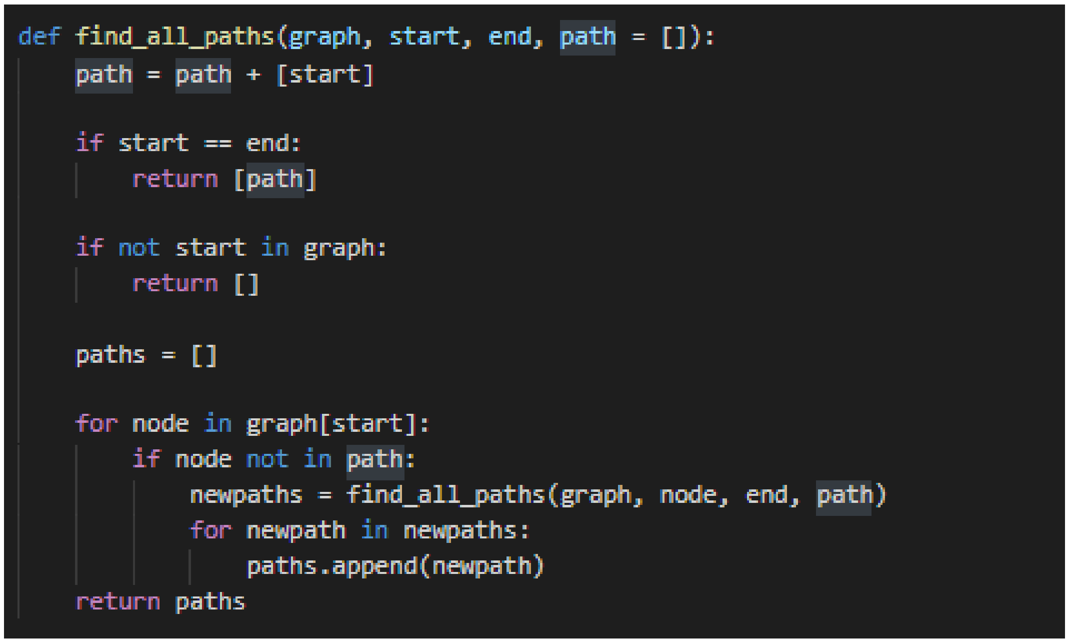
Answer:



```
>>> print("Path between nodes", find_path(graph, '1', '7', path))
Path between nodes ['1', '2', '3', '4', '5', '6', '7']
```

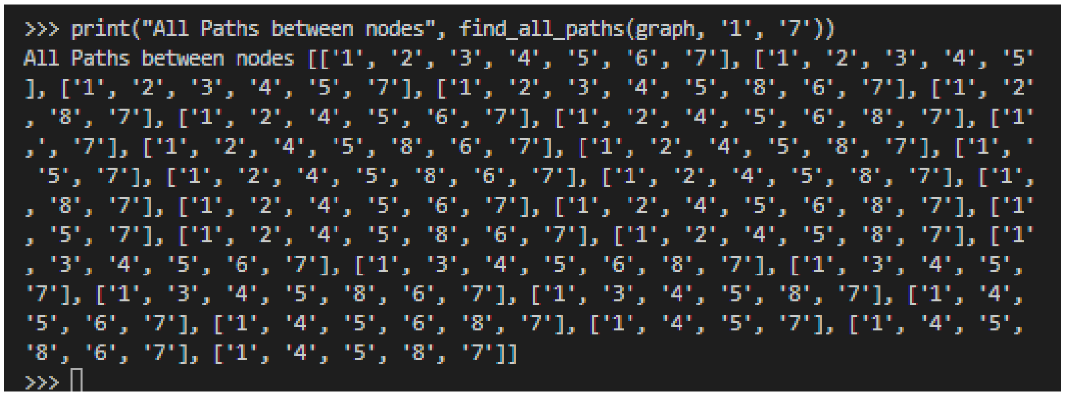
c) Find all paths in graph.

Code:



```python
def find_all_paths(graph, start, end, path = []):
    path = path + [start]

    if start == end:
        return [path]

    if not start in graph:
        return []

    paths = []

    for node in graph[start]:
        if node not in path:
            newpaths = find_all_paths(graph, node, end, path)
            for newpath in newpaths:
                paths.append(newpath)
    return paths
```

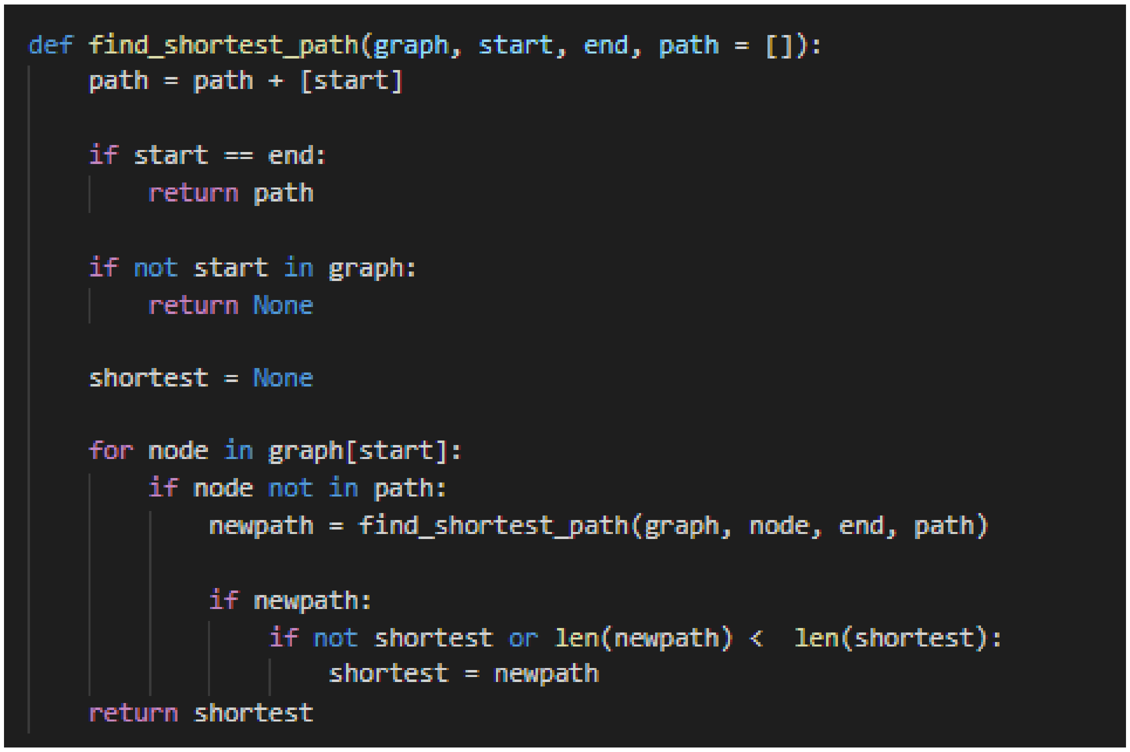
Answer:



```
>>> print("All Paths between nodes", find_all_paths(graph, '1', '7'))
All Paths between nodes [['1', '2', '3', '4', '5', '6', '7'], ['1', '2', '3', '4', '5'
], ['1', '2', '3', '4', '5', '7'], ['1', '2', '3', '4', '5', '8', '6', '7'], ['1', '2'
, '8', '7'], ['1', '2', '4', '5', '6', '7'], ['1', '2', '4', '5', '6', '8', '7'], ['1'
,', '7'], ['1', '2', '4', '5', '8', '6', '7'], ['1', '2', '4', '5', '8', '7'], ['1', '
 '5', '7'], ['1', '2', '4', '5', '8', '6', '7'], ['1', '2', '4', '5', '8', '7'], ['1',
, '8', '7'], ['1', '2', '4', '5', '6', '7'], ['1', '2', '4', '5', '6', '8', '7'], ['1'
, '5', '7'], ['1', '2', '4', '5', '8', '6', '7'], ['1', '2', '4', '5', '8', '7'], ['1'
, '3', '4', '5', '6', '7'], ['1', '3', '4', '5', '6', '8', '7'], ['1', '3', '4', '5',
'7'], ['1', '3', '4', '5', '8', '6', '7'], ['1', '3', '4', '5', '8', '7'], ['1', '4',
'5', '6', '7'], ['1', '4', '5', '6', '8', '7'], ['1', '4', '5', '7'], ['1', '4', '5',
'8', '6', '7'], ['1', '4', '5', '8', '7']]
>>> 
```

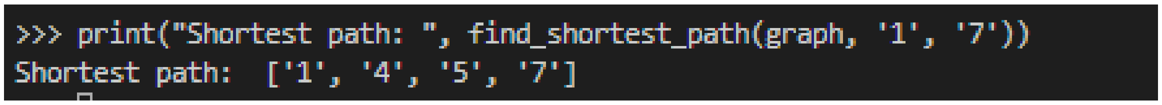
d) Find shortest path between node 1 and 7

Code:



```python
def find_shortest_path(graph, start, end, path = []):
    path = path + [start]

    if start == end:
        return path

    if not start in graph:
        return None

    shortest = None

    for node in graph[start]:
        if node not in path:
            newpath = find_shortest_path(graph, node, end, path)

            if newpath:
                if not shortest or len(newpath) <  len(shortest):
                    shortest = newpath
    return shortest
```

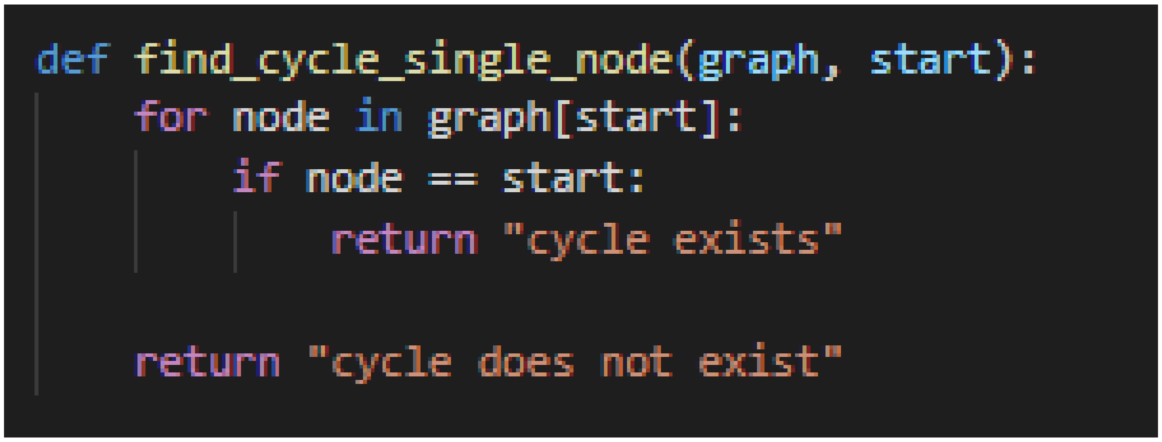
Answer:



```
>>> print("Shortest path: ", find_shortest_path(graph, '1', '7'))
Shortest path:  ['1', '4', '5', '7']
```

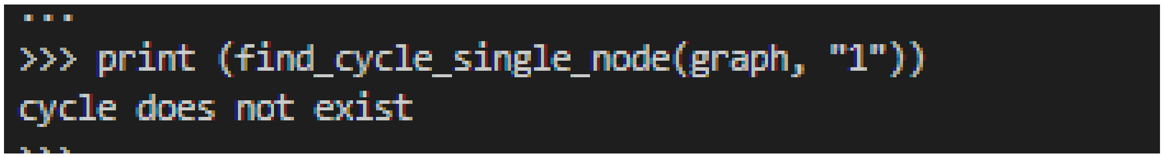
e) Determine cycles in graphs.

Code:



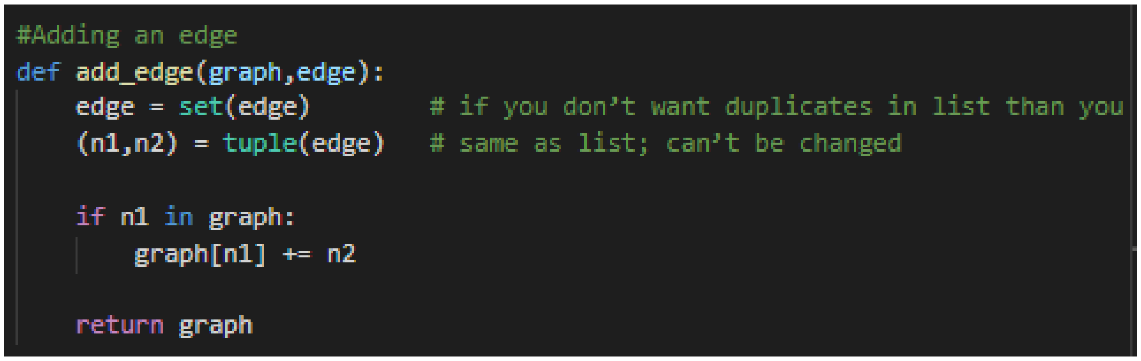
```python
def find_cycle_single_node(graph, start):
    for node in graph[start]:
        if node == start:
            return "cycle exists"

    return "cycle does not exist"
```

Answer:



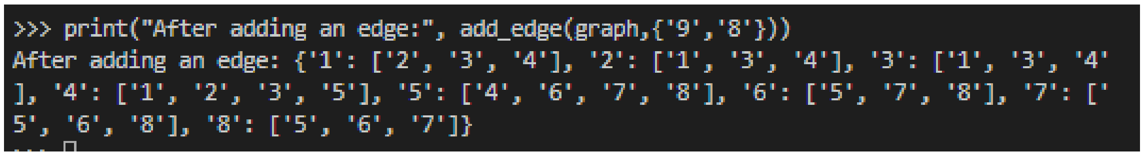
```
...
>>> print (find_cycle_single_node(graph, "1"))
cycle does not exist
```

f) Add an edge named 9.
Code:



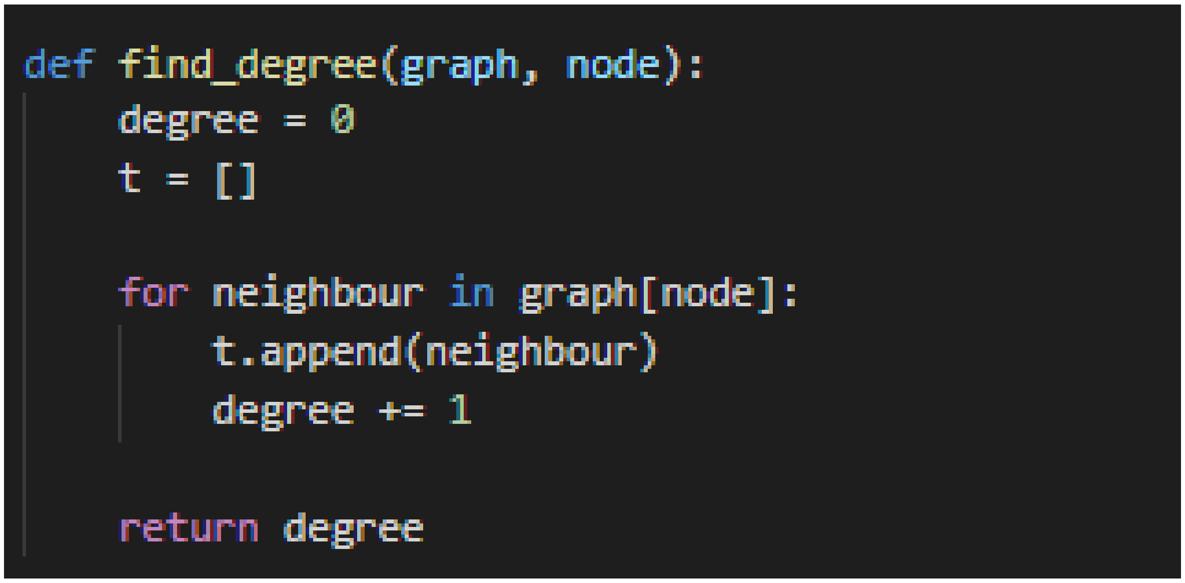
```
#Adding an edge
def add_edge(graph,edge):
    edge = set(edge)         # if you don’t want duplicates in list than you
    (n1,n2) = tuple(edge)    # same as list; can’t be changed

    if n1 in graph:
        graph[n1] += n2

    return graph
```

Answer:



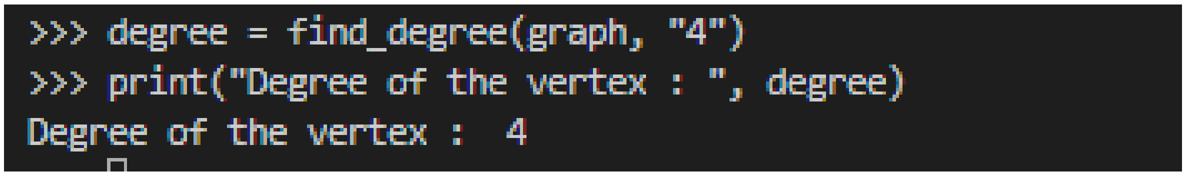
```
>>> print("After adding an edge:", add_edge(graph,{'9','8'}))
After adding an edge: {'1': ['2', '3', '4'], '2': ['1', '3', '4'], '3': ['1', '3', '4'
], '4': ['1', '2', '3', '5'], '5': ['4', '6', '7', '8'], '6': ['5', '7', '8'], '7': ['
5', '6', '8'], '8': ['5', '6', '7']}
```

g) Find degree of vertex 4.
Code:



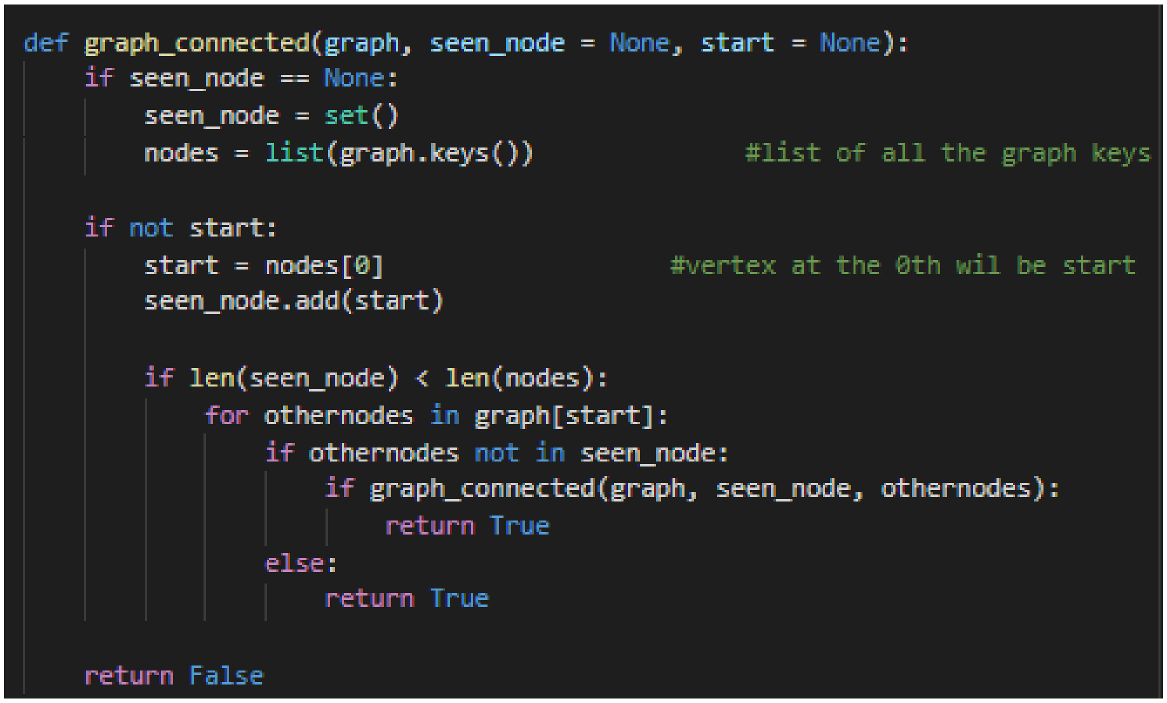
```
def find_degree(graph, node):
    degree = 0
    t = []

    for neighbour in graph[node]:
        t.append(neighbour)
        degree += 1

    return degree
```

Answer:



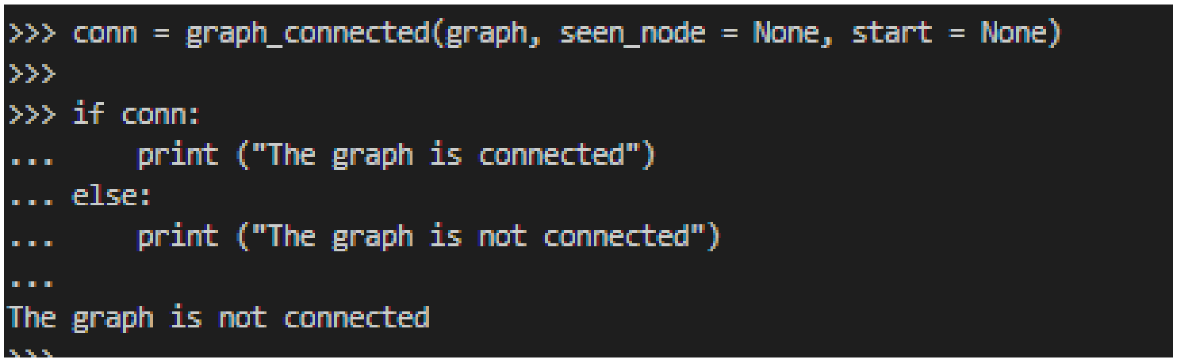
```
>>> degree = find_degree(graph, "4")
>>> print("Degree of the vertex : ", degree)
Degree of the vertex :  4
```

h) Find if the graph is connected.
Code:



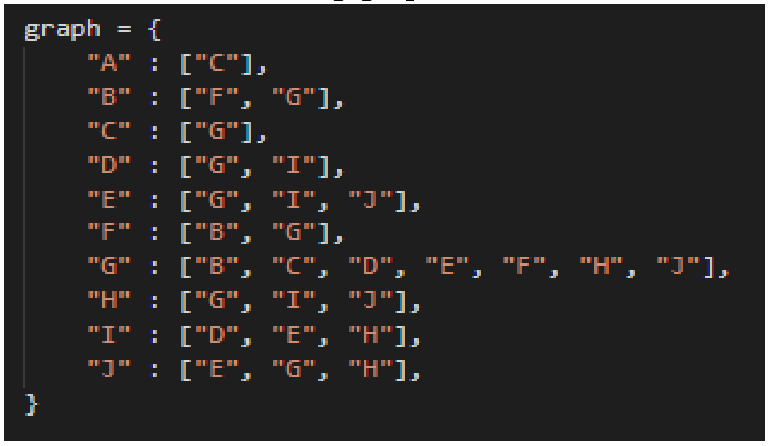
```
def graph_connected(graph, seen_node = None, start = None):
    if seen_node == None:
        seen_node = set()
        nodes = list(graph.keys())              #list of all the graph keys

    if not start:
        start = nodes[0]                     #vertex at the 0th wil be start
        seen_node.add(start)

        if len(seen_node) < len(nodes):
            for othernodes in graph[start]:
                if othernodes not in seen_node:
                    if graph_connected(graph, seen_node, othernodes):
                        return True
                else:
                    return True

    return False
```

Answer:



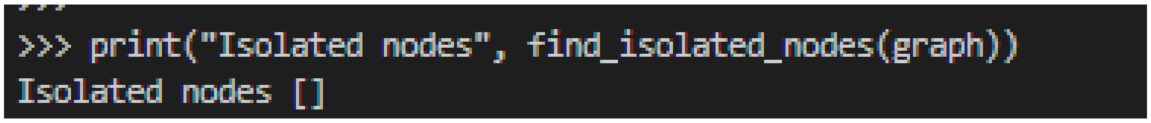
```
>>> conn = graph_connected(graph, seen_node = None, start = None)
>>>
>>> if conn:
...     print ("The graph is connected")
... else:
...     print ("The graph is not connected")
...
The graph is not connected
```

Ques: 3 Consider the following graph:



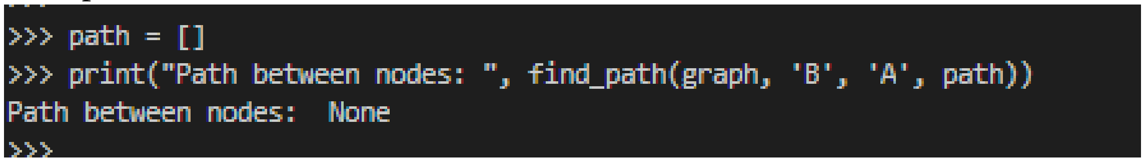
```
graph = {
    "A" : ["C"],
    "B" : ["F", "G"],
    "C" : ["G"],
    "D" : ["G", "I"],
    "E" : ["G", "I", "J"],
    "F" : ["B", "G"],
    "G" : ["B", "C", "D", "E", "F", "H", "J"],
    "H" : ["G", "I", "J"],
    "I" : ["D", "E", "H"],
    "J" : ["E", "G", "H"],
}
```

a) Find isolated nodes.



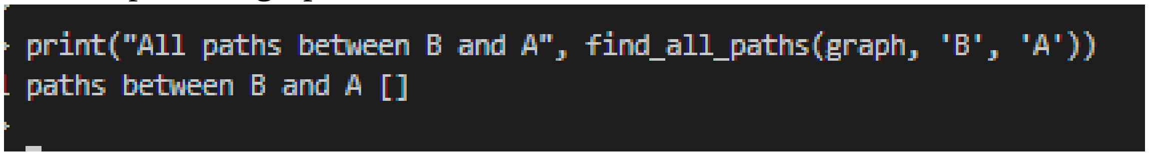
```
>>> print("Isolated nodes", find_isolated_nodes(graph))
Isolated nodes []
```

b) Find path between two vertex/node B and A.



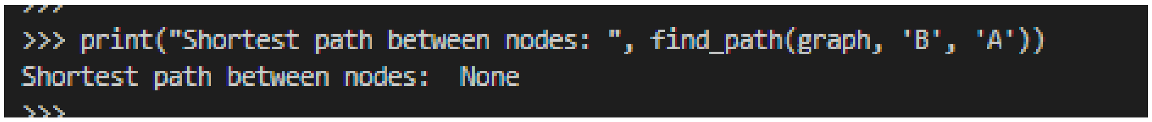
```
>>> path = []
>>> print("Path between nodes: ", find_path(graph, 'B', 'A', path))
Path between nodes:  None
>>>
```

c) Find all paths in graphs.



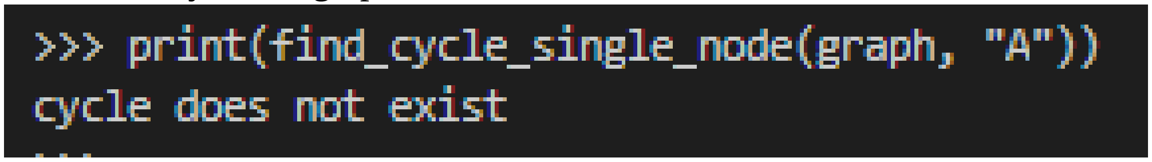
```
 print("All paths between B and A", find_all_paths(graph, 'B', 'A'))
 paths between B and A []
```

d) Find shortest between node B and A.



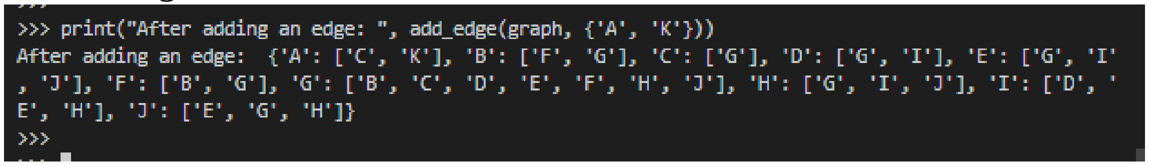
```
>>> print("Shortest path between nodes: ", find_path(graph, 'B', 'A'))
Shortest path between nodes:  None
>>>
```

e) Determine cycles in graphs.



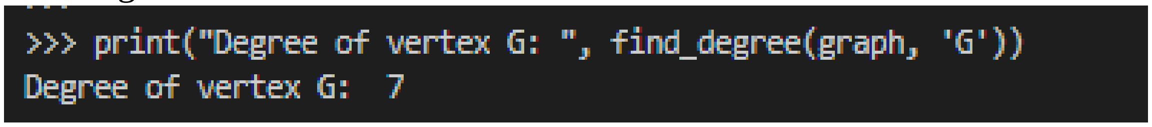
```
>>> print(find_cycle_single_node(graph, "A"))
cycle does not exist
```

f) Add an edge named K.



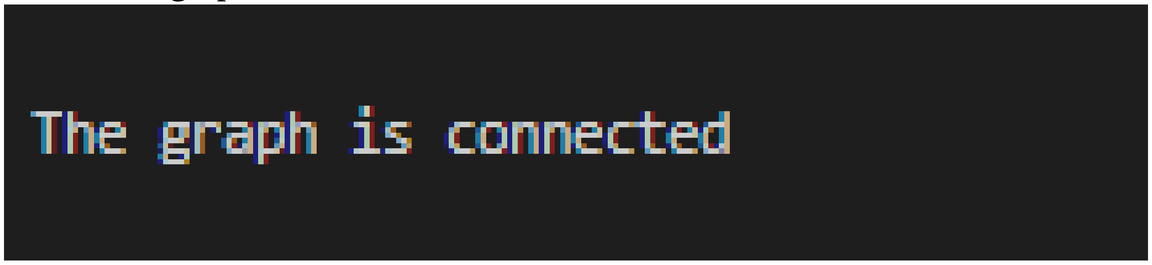
```
>>> print("After adding an edge: ", add_edge(graph, {'A', 'K'}))
After adding an edge:  {'A': ['C', 'K'], 'B': ['F', 'G'], 'C': ['G'], 'D': ['G', 'I'], 'E': ['G', 'I'
, 'J'], 'F': ['B', 'G'], 'G': ['B', 'C', 'D', 'E', 'F', 'H', 'J'], 'H': ['G', 'I', 'J'], 'I': ['D', '
E', 'H'], 'J': ['E', 'G', 'H']}
>>>
```

g) Find degree of vertex G.



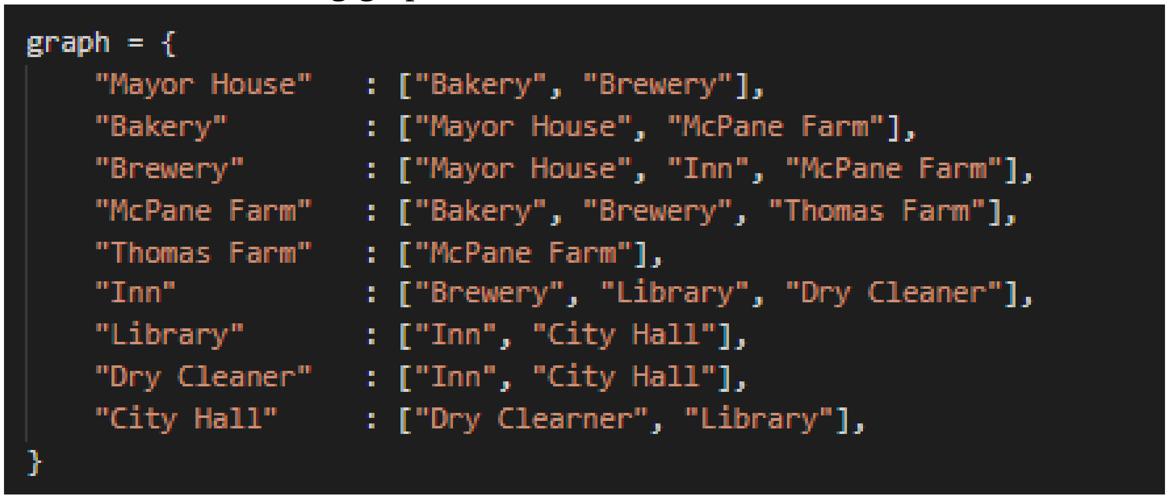
```
>>> print("Degree of vertex G: ", find_degree(graph, 'G'))
Degree of vertex G:  7
```

h) Find if the graph is connected.



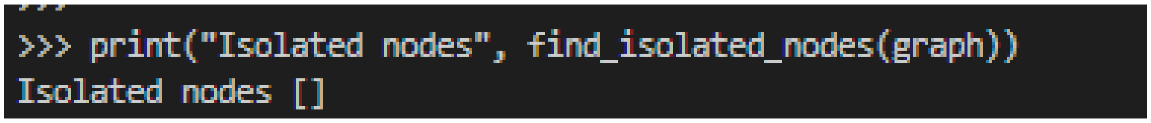
```
The graph is connected
```

Ques: 4 Consider the following graph:



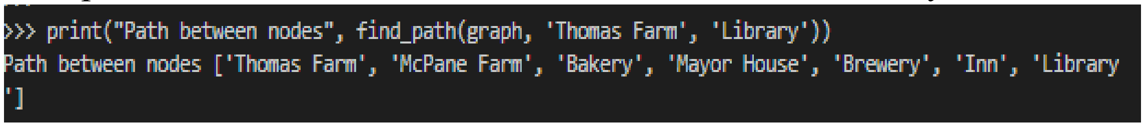
```
graph = {
    "Mayor House"   : ["Bakery", "Brewery"],
    "Bakery"        : ["Mayor House", "McPane Farm"],
    "Brewery"       : ["Mayor House", "Inn", "McPane Farm"],
    "McPane Farm"   : ["Bakery", "Brewery", "Thomas Farm"],
    "Thomas Farm"   : ["McPane Farm"],
    "Inn"           : ["Brewery", "Library", "Dry Cleaner"],
    "Library"       : ["Inn", "City Hall"],
    "Dry Cleaner"   : ["Inn", "City Hall"],
    "City Hall"     : ["Dry Clearner", "Library"],
}
```

a) Find isolated nodes.



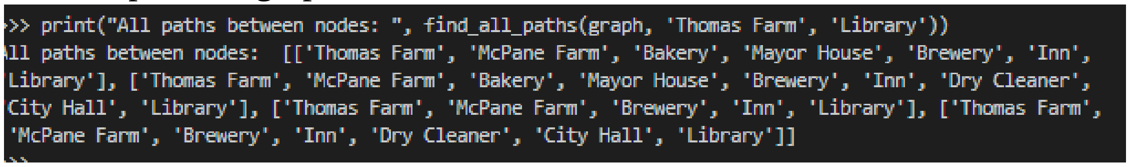
```
>>> print("Isolated nodes", find_isolated_nodes(graph))
Isolated nodes []
```

b) Find path between two vertex/node Thomas' Farm and Library.



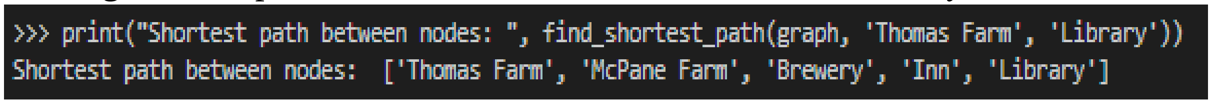
```
>>> print("Path between nodes", find_path(graph, 'Thomas Farm', 'Library'))
Path between nodes ['Thomas Farm', 'McPane Farm', 'Bakery', 'Mayor House', 'Brewery', 'Inn', 'Library
']
```

c) Find all paths in graph.



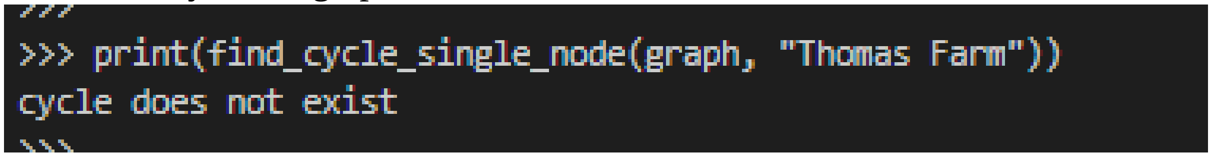
```
>> print("All paths between nodes: ", find_all_paths(graph, 'Thomas Farm', 'Library'))
ll paths between nodes:  [['Thomas Farm', 'McPane Farm', 'Bakery', 'Mayor House', 'Brewery', 'Inn',
Library'], ['Thomas Farm', 'McPane Farm', 'Bakery', 'Mayor House', 'Brewery', 'Inn', 'Dry Cleaner',
City Hall', 'Library'], ['Thomas Farm', 'McPane Farm', 'Brewery', 'Inn', 'Library'], ['Thomas Farm',
'McPane Farm', 'Brewery', 'Inn', 'Dry Cleaner', 'City Hall', 'Library']]
```

d) Finding shortest path between nodes Thomas' Farm and Library.



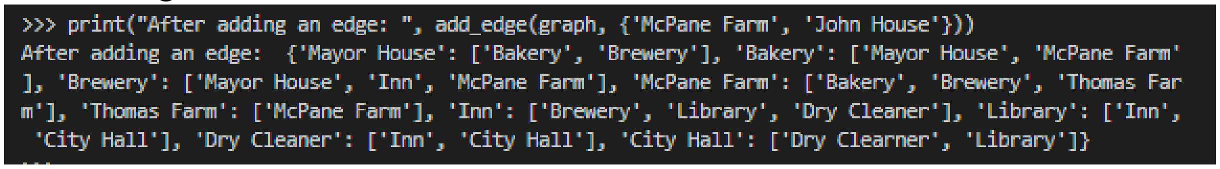
```
>>> print("Shortest path between nodes: ", find_shortest_path(graph, 'Thomas Farm', 'Library'))
Shortest path between nodes:  ['Thomas Farm', 'McPane Farm', 'Brewery', 'Inn', 'Library']
```

e) Determine cycles in graphs.



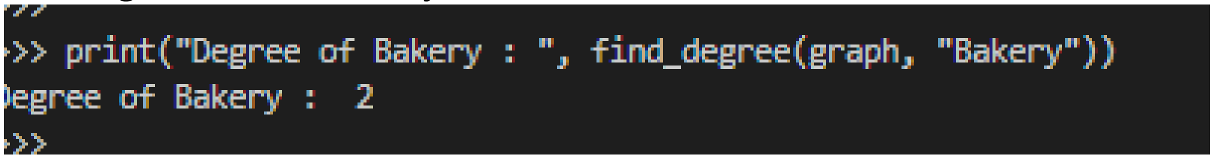
```
>>> print(find_cycle_single_node(graph, "Thomas Farm"))
cycle does not exist
```

f) Add an edge named John's House.



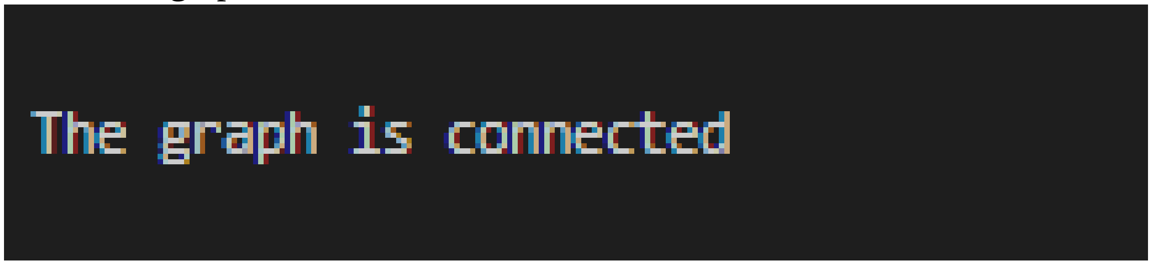
```
>>> print("After adding an edge: ", add_edge(graph, {'McPane Farm', 'John House'}))
After adding an edge:  {'Mayor House': ['Bakery', 'Brewery'], 'Bakery': ['Mayor House', 'McPane Farm'
], 'Brewery': ['Mayor House', 'Inn', 'McPane Farm'], 'McPane Farm': ['Bakery', 'Brewery', 'Thomas Far
m'], 'Thomas Farm': ['McPane Farm'], 'Inn': ['Brewery', 'Library', 'Dry Cleaner'], 'Library': ['Inn',
 'City Hall'], 'Dry Cleaner': ['Inn', 'City Hall'], 'City Hall': ['Dry Clearner', 'Library']}
```

g) Find degree of vertex Bakery.



```
>> print("Degree of Bakery : ", find_degree(graph, "Bakery"))
egree of Bakery :  2
>>
```

h) Find if the graph is connected.



```
The graph is connected
```