Write a naïve or brute-force algorithm for finding a Hamiltonian cycle in a graph. Your solution must include the followings:

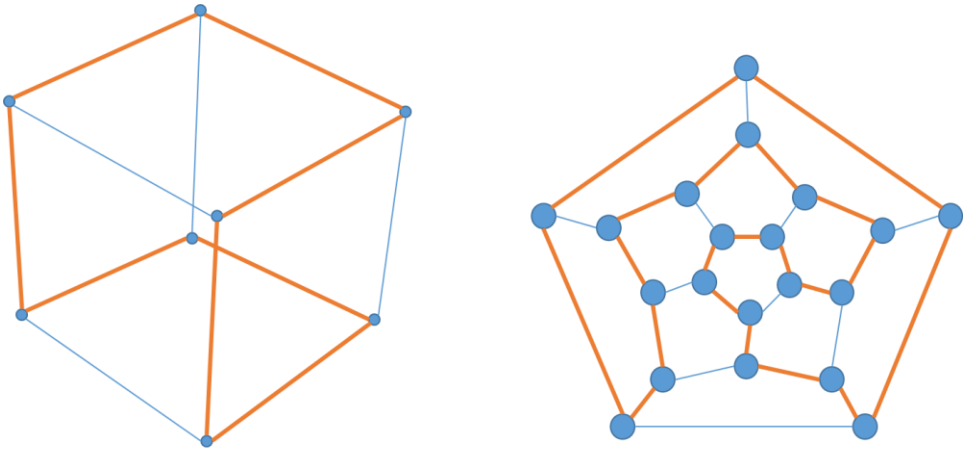
Pseudo code for your logic.

```
Def hamiltonianCycle(Graph, currentNode):
    If currentNode has not been visited:
        visit the node: visited.append(currentNode)
    if all nodes have been visited:
        return visited list.
    for nextNode in Graph:
        check next nodes recursively for Hamiltonian cycles.
```

Implementation in Python.

See .py file.

Sample test case corresponding to the example given in the Hamiltonian cycle problem description.



(a) (b)

For my sample test case I will use figure (a). I've numbered the nodes started at the point with 2 orange lines going into it as 1, then the top square counterclockwise is 1,2,3,4. The below square (starting at node 5 directly below 1) is 5,6,7,8

1. The algorithm will start at node 1. This has not been visited, so the node will be placed in the visited list. Not all nodes have been visited yet. The next node will be calculated as node 5 (directly below node 1), recursively call the algorithm with node 5 as the starting point.
2. The algorithm is on node 5. This has not been visited, so the node will be placed in the visited list. Not all nodes have been visited yet. The next node will be calculated as node 6 (to the right of node 5).
3. The algorithm is on node 6. This has not been visited, so the node will be placed in the visited list. Not all nodes have been visited yet. The next node will be calculated as node 7 (to the left of node 6).
4. The algorithm is on node 7. This has not been visited, so the node will be placed in the visited list. Not all nodes have been visited yet. The next node will be calculated as node 8 (to the left of node 7).
5. The algorithm is on node 8. This has not been visited, so the node will be placed in the visited list. Not all nodes have been visited yet. The next node will be calculated as node 4 (directly above node 8).
6. The algorithm is on node 4. This has not been visited, so the node will be placed in the visited list. Not all nodes have been visited yet. The next node will be calculated as node 3 (to the right of node 4).
7. The algorithm is on node 3. This has not been visited, so the node will be placed in the visited list. Not all nodes have been visited yet. The next node will be calculated as node 2 (to the right of node 3).
8. The algorithm is on node 2. This has not been visited, so the node will be placed in the visited list. Not all nodes have been visited yet. The next node will be calculated as node 1 (to the left of node 6).
9. The algorithm is on node 1. This node has been visited and the number of nodes visited is equal to the total number of nodes. The algorithm returns the visited array.

Note: The algorithm will visit all the neighbors of the node it is on. However, I have omitted putting in the steps of the nodes that were already visited as this test case description would be lengthy. So, keep in mind that when the algorithm is on a node it's also checking some nodes that have been visited. For example, node 4 will also visit node 1, but that node has already been visited so the algorithm will do nothing. To save time, I've only put the more interesting steps in the description.