

## Solutions, Week 3, Day 2

1. Give the graph class, from the previous lesson, an iterator over all nodes.

Give it the method:

```
class Graph:
    [...]

    def __iter__(self):
        for item in self._nodeDict:
            yield item
```

2. Give the graph class an iterator over all neighbours of a given node, e.g. `myGraph.iterNeighbours(myNode)`.

Give it the method:

```
class Graph:
    [...]

    def iterNeighbours(self, n):
        return _iterGraphNeighbours(self, n)

    def _iterGraphNeighbours(g, n):
        for item in g._nodeDict[n]:
            yield item
```

3. Give the graph class a “random walk” iterator, that starts from a random node and moves to a joining random node. (You’ll need the `random` module, in particular the `choice` method.)

Give it the method:

```
import random

class Graph:
    [...]

    def iterRandom(self, n):
        theNodes = self._nodeDict.keys()
        theStartingNode = random.choice(theNodes)
        return _iterRandom(self, theStartingNode)

    def _iterRandom(g, curr_node):
        yield curr_node
        theNodes = self._nodeDict[curr_node]
        if (theNodes):
            theNextNode = random.choice(theNodes)
        else:
            # what if this node has no neighbours?
```

```
        theNextNode = curr_node
return _iterRandom (self, theNextNode)
```