Create a Graph, print it

```
class Node:
    def __init__(self,src,des,weight):
        self.source = src
        self.destination = des
        self.weight = weight

class Graph:
    def __init__(self):
        self.vertices = {}

    def addEdge(self,src,des,weight=0):
        if src in self.vertices:
            self.vertices[src].append([des,weight])
        else:
            self.vertices[src] = []
            self.vertices[src].append([des,weight])

        # if des in self.vertices:
        #     self.vertices[des].append([src,weight])
        # else:
        #     self.vertices[des] = []
        #     self.vertices[des].append([src,weight])
    def neighbours(self,vertex):
        nbr = []
        for vert in self.vertices[vertex]:
            nbr.append(vert[0])
        return nbr
    def printgraph(self):
        for vert in self.vertices:
            print(vert, '->',self.vertices[vert])

    def edgeCost(self,src,nbr):
        for neigh in self.vertices[src]:
            if neigh[0]==nbr:
                return neigh[1]


graph = Graph()
graph.addEdge(0,1,10)
graph.addEdge(0,4,20)
graph.addEdge(4,0,30)
graph.addEdge(4,1,40)
graph.addEdge(4,3,50)
graph.addEdge(3,1,60)
graph.addEdge(3,2,70)
graph.addEdge(3,4,80)
graph.addEdge(1,0,90)
graph.addEdge(1,2,100)
graph.addEdge(1,4,110)
graph.addEdge(1,3,120)
graph.addEdge(2,1,130)
graph.addEdge(2,3,140)
graph.printgraph()
print(graph.neighbours(3))
print(graph.edgeCost(4,3))
```

```
class Graph:
    def __init__(self):
        self.vertexes = {}
        self.weightedEdges = {}

    def addVertex(self, vert):
        if vert not in self.vertexes:
            self.vertexes[vert] = []

    def addEdge(self, fromVert, toVert):
        if fromVert in self.vertexes:
            self.vertexes[fromVert].append(toVert)
        else:
            self.vertexes[fromVert] = []
            self.vertexes[fromVert].append(toVert)

    def addWeightedEdges(self, fromVert, toVert, weight):
        if fromVert not in self.weightedEdges:
            self.weightedEdges[fromVert] = []
            self.weightedEdges[fromVert].append((toVert, weight))
        else:
            self.weightedEdges[fromVert].append((toVert, weight))

    def getVertex(self, vertKey):
        if vertKey in self.vertexes:
            return self.vertexes[vertKey]
```

```python
    def getVertices(self):
        for vert in self.vertexes:
            print(vert, '->', self.vertexes[vert])

    def getWeightedVertices(self):
        for vert in self.weightedEdges:
            print(vert, '->', self.weightedEdges[vert])

    def inGraph(self, vert):
        if vert in self.vertexes:
            return True
        else:
            return False


graph = Graph()
graph.addVertex(0)
graph.addVertex(1)
graph.addVertex(2)
graph.addVertex(3)
graph.addVertex(4)
graph.addVertex(5)
# graph.getVertices()
graph.addEdge(0, 1)
graph.addEdge(1, 2)
graph.addEdge(2, 3)
graph.addEdge(3, 4)
graph.addEdge(4, 0)
graph.addEdge(0, 5)
graph.addEdge(3, 5)
graph.addEdge(5, 2)
graph.addEdge(5, 4)
# graph.getVertices()

print(graph.inGraph(4))

graph.addWeightedEdges(0, 1, 5)
graph.addWeightedEdges(1, 2, 4)
graph.addWeightedEdges(2, 3, 9)
graph.addWeightedEdges(3, 4, 7)
graph.addWeightedEdges(4, 0, 1)
graph.addWeightedEdges(0, 5, 2)
graph.addWeightedEdges(3, 5, 3)
graph.addWeightedEdges(5, 2, 1)
graph.addWeightedEdges(5, 4, 8)

graph.getWeightedVertices()
```