The Graph Abstract Data Type

The graph abstract data type (ADT) is defined as follows:

- `Graph()` creates a new, empty graph.
- `addVertex(vert)` adds an instance of `Vertex` to the graph.
- `addEdge(fromVert, toVert)` Adds a new, directed edge to the graph that connects two vertices.
- `addEdge(fromVert, toVert, weight)` Adds a new, weighted, directed edge to the graph that connects two vertices.
- `getVertex(vertKey)` finds the vertex in the graph named `vertKey`.
- `getVertices()` returns the list of all vertices in the graph.
- `in` returns `True` for a statement of the form `vertex in graph`, if the given vertex is in the graph, `False` otherwise.

Beginning with the formal definition for a graph there are several ways we can implement the graph ADT in Python. We will see that there are trade-offs in using different representations to implement the ADT described above. There are two well-known implementations of a graph, the **adjacency matrix** and the **adjacency list**. We will explain both of these options, and then implement one as a Python class.

Source: Problem Solving and Algorithms in Python (http://interactivepython.org/runestone/static/pythonds/index.html#) from Bradley Miller on www.interactivepython.org (http://interactivepython.org/runestone/static/pythonds/index.html#).