Directed Graphs

```
# Previously I talked about storing airport locations along
# with connecting flights in a Graph. It didn't however
# make sense to store that information because we couldn't
# define directions from one airport to another. That isn't
# true now with Directed Graphs!

# With Directed Graphs you can only go in one direction on
# an edge between vertices. That fits perfectly with our
# airport model.

# To define one our Adjacency Matrix will contain a 1 if the
# Row connects to a Column. In our example Pitt connects to
# the Det, DC, Tor and Nash columns.

# To make this work we'll start and then look for a vertex with
# no successors. If we find it we delete that vertex from the
# graph and insert it into a list.

# Model a Vertex
class Vertex:
    def __init__(self, name):
        self.name = name


class Graph:
    def __init__(self):
        self.max_vertices = 10
        self.vertex_list = [0]*10
        self.adjacency_matrix = [[0] * self.max_vertices for i in range(self.max_vertices)]
        self.vertex_count = 0

        # NEW We need a place to store the vertices in order
        self.sorted_list = [0]*10

    def add_vertex(self, name):
        self.vertex_list[self.vertex_count] = Vertex(name)
        self.vertex_count += 1

    # NEW We only store a 1 where the row connects to a column
    def add_edge(self, first, last):
        self.adjacency_matrix[first][last] = 1

    def print_vertex(self, index):
        print(self.vertex_list[index].name, end="")

    # NEW Move row up in adjacency matrix
    def move_row_up(self, row, length):
        for col in range(0, length):
            self.adjacency_matrix[row][col] = self.adjacency_matrix[row+1][col]

    # NEW Moves a column to the left in the adjacency matrix
```

```
def move_col_left(self, col, length):
    for row in range(0, length):
        self.adjacency_matrix[row][col] = self.adjacency_matrix[row][col+1]

# NEW Deletes a vertex
def delete_vertex(self, index):
    # If not the last vertex
    if index != self.vertex_count - 1:
        # Delete from vertex list
        for i in range(index, self.vertex_count - 1):
            self.vertex_list[i] = self.vertex_list[i+1]

        # Delete row from adjacency matrix
        for row in range(index, self.vertex_count-1):
            self.move_row_up(row, self.vertex_count)

        # Delete column from adjacency matrix
        for col in range(index, self.vertex_count-1):
            self.move_col_left(col, self.vertex_count-1)

    # Delete vertex count
    self.vertex_count -= 1

# NEW Returns vertices with no successors or -1 if none
# apply
def get_vert_with_no_successors(self):
    # Boolean that checks for edge
    is_edge = False

    for row in range(0, self.vertex_count):
        is_edge = False
        for col in range(0, self.vertex_count):

            # Check if vertex has a successor and if
            # it does try the next vertex
            if self.adjacency_matrix[row][col] > 0:
                is_edge = True
                break

        # If no edges are found it has no successors
        if not is_edge:
            return row

    # If here there are no more vertices
    return -1


# NEW Create the ordered list
def sort(self):
    # Get the original number of vertices
    start_number_verts = self.vertex_count

    # Cycle while vertices remain
    while self.vertex_count > 0:
            # Get vertex with no successors
            curr_vertex = self.get_vert_with_no_successors()

            # If -1 then a cycle exists meaning vertices
            # are not following the rule that there must be
            # an ending point
            if curr_vertex == -1:
                print("Error : Your Graph has a Cycle")
                return

            # Insert vertex name in the sorted list
            self.sorted_list[self.vertex_count - 1] = self.vertex_list[curr_vertex].name

            # Delete the vertex
            self.delete_vertex(curr_vertex)

        # Display the sorted list
        print(self.sorted_list)


graph = Graph()
graph.add_vertex('Pitt')
graph.add_vertex('Det')
graph.add_vertex('DC')
graph.add_vertex('Tor')
graph.add_vertex('Nash')
graph.add_vertex('Madi')
graph.add_vertex('Lans')
graph.add_vertex('Buff')
graph.add_vertex('Knox')
graph.add_edge(0, 1)
graph.add_edge(0, 2)
graph.add_edge(0, 3)
graph.add_edge(0, 4)
graph.add_edge(1, 5)
graph.add_edge(2, 6)
graph.add_edge(3, 7)
graph.add_edge(4, 8)
graph.sort()
```