https://learnpython.com/blog/create-dictionary-in-python/

|  |
| --- |
| **How to implement a graph in Python** |
| A **graph** is a data structure that consists of vertices that are connected ​via edges. It can be implemented with an: 1. Adjacency list For every vertex, its adjacent vertices are stored. In the case of a weighted graph, the edge weights are stored along with the vertices.​ 2. Adjacency matrix The row and column indices represent the vertices: *matrix*[*i*][*j*]=1 means that there is an edge from vertices *i* to *j*, *matrix*[*i*][*j*]=0 denotes that there is no edge between *i* and *j*.  For a weighted graph,the edge weight is usually written in place of 1. |
| Implementation1. Using an adjacency list The following code implements a graph using an adjacency list: add\_vertex(v) adds new vertex v to the graph, and add\_edge(v1, v2, e) adds an edge with weight e between vertices v1 and v2. |
| *Add a vertex to the dictionary*  def add\_vertex(v):    global graph    global vertices\_no    if v in graph:      print("Vertex ", v, " already exists.")    else:      vertices\_no = vertices\_no + 1      graph[v] = []  *# Add an edge between vertex v1 and v2 with edge weight e*  def add\_edge(v1, v2, e):    global graph  *# Check if vertex v1 is a valid vertex*    if v1 not in graph:      print("Vertex ", v1, " does not exist.")  *# Check if vertex v2 is a valid vertex*    elif v2 not in graph:      print("Vertex ", v2, " does not exist.")    else:  *# Since this code is not restricted to a directed or*  *# an undirected graph, an edge between v1 v2 does not*  *# imply that an edge exists between v2 and v1*      temp = [v2, e]      graph[v1].append(temp)  *# Print the graph*  def print\_graph():    global graph    for vertex in graph:      for edges in graph[vertex]:        print(vertex, " -> ", edges[0], " edge weight: ", edges[1])  *# driver code*  graph = {}  *# stores the number of vertices in the graph*  vertices\_no = 0  add\_vertex(1)  add\_vertex(2)  add\_vertex(3)  add\_vertex(4)  *# Add the edges between the vertices by specifying* |
| 2. Using an adjacency matrix **The following code implements a graph using an adjacency matrix: add\_vertex(v) adds new vertex v to the graph, and add\_edge(v1, v2, e) adds an edge with weight e between vertices v1 and v2.** |
| def add\_vertex(v):    global graph    global vertices\_no    global vertices    if v in vertices:      print("Vertex ", v, " already exists")    else:      vertices\_no = vertices\_no + 1      vertices.append(v)      if vertices\_no > 1:          for vertex in graph:              vertex.append(0)      temp = []      for i in range(vertices\_no):          temp.append(0)      graph.append(temp)  *# Add an edge between vertex v1 and v2 with edge weight e*  def add\_edge(v1, v2, e):      global graph      global vertices\_no      global vertices  *# Check if vertex v1 is a valid vertex*      if v1 not in vertices:          print("Vertex ", v1, " does not exist.")  *# Check if vertex v1 is a valid vertex*      elif v2 not in vertices:          print("Vertex ", v2, " does not exist.")  *# Since this code is not restricted to a directed or*  *# an undirected graph, an edge between v1 v2 does not*  *# imply that an edge exists between v2 and v1*      else:          index1 = vertices.index(v1)          index2 = vertices.index(v2)          graph[index1][index2] = e  *# Print the graph*  def print\_graph():    global graph    global vertices\_no    for i in range(vertices\_no):      for j in range(vertices\_no):        if graph[i][j] != 0:          print(vertices[i], " -> ", vertices[j], \          " edge weight: ", graph[i][j])  *# Driver code*  *# stores the vertices in the graph*  vertices = []  *# stores the number of vertices in the graph*  vertices\_no = 0  graph = []  *# Add vertices to the graph*  add\_vertex(1)  add\_vertex(2)  add\_vertex(3)  add\_vertex(4)  *# Add the edges between the vertices by specifying*  *# the from and to vertex along with the edge weights.*  add\_edge(1, 2, 1)  add\_edge(1, 3, 1)  add\_edge(2, 3, 3)  add\_edge(3, 4, 4)  add\_edge(4, 1, 5)  print\_graph()  print("Internal representation: ", graph) |
|  |