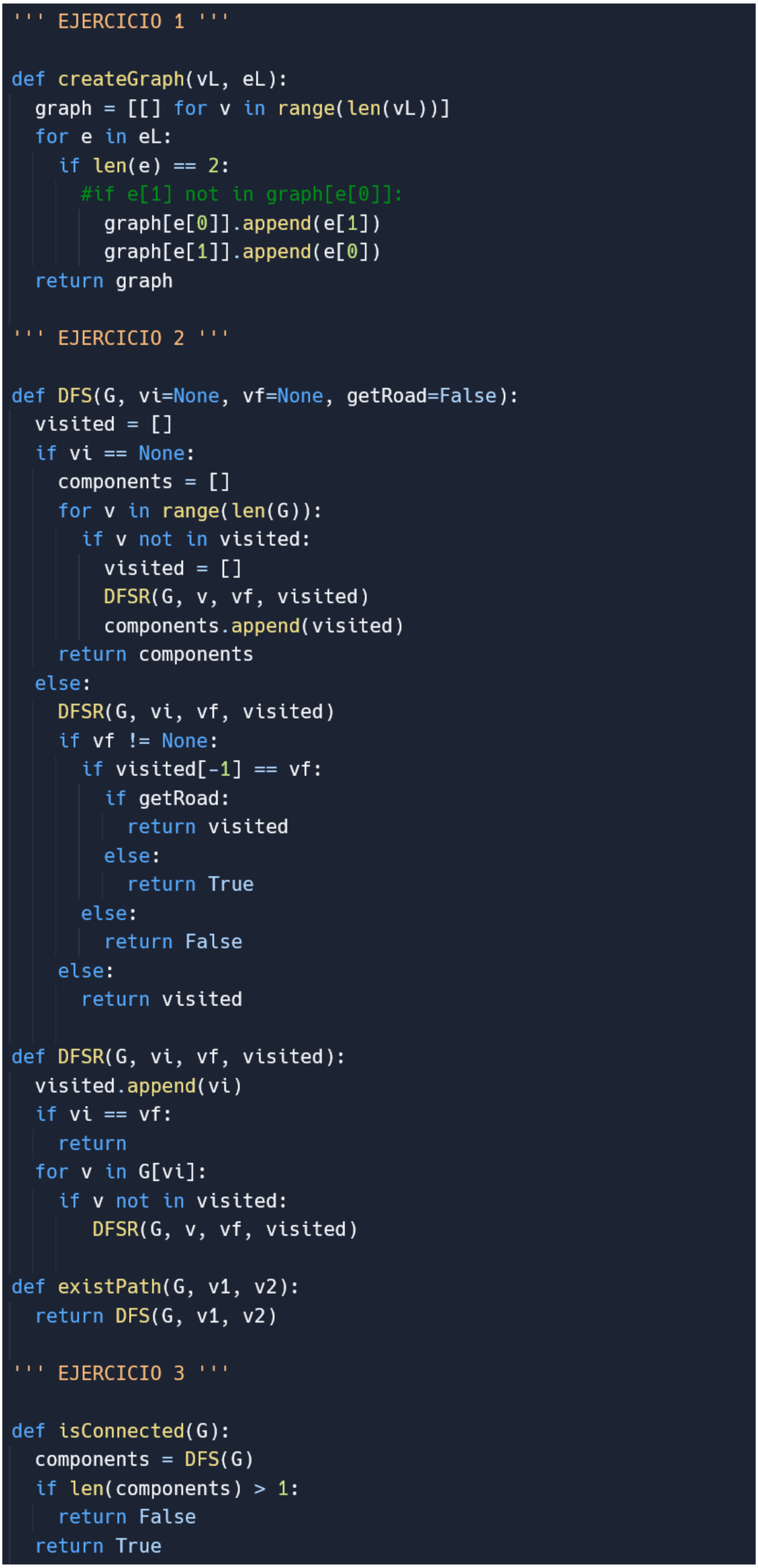
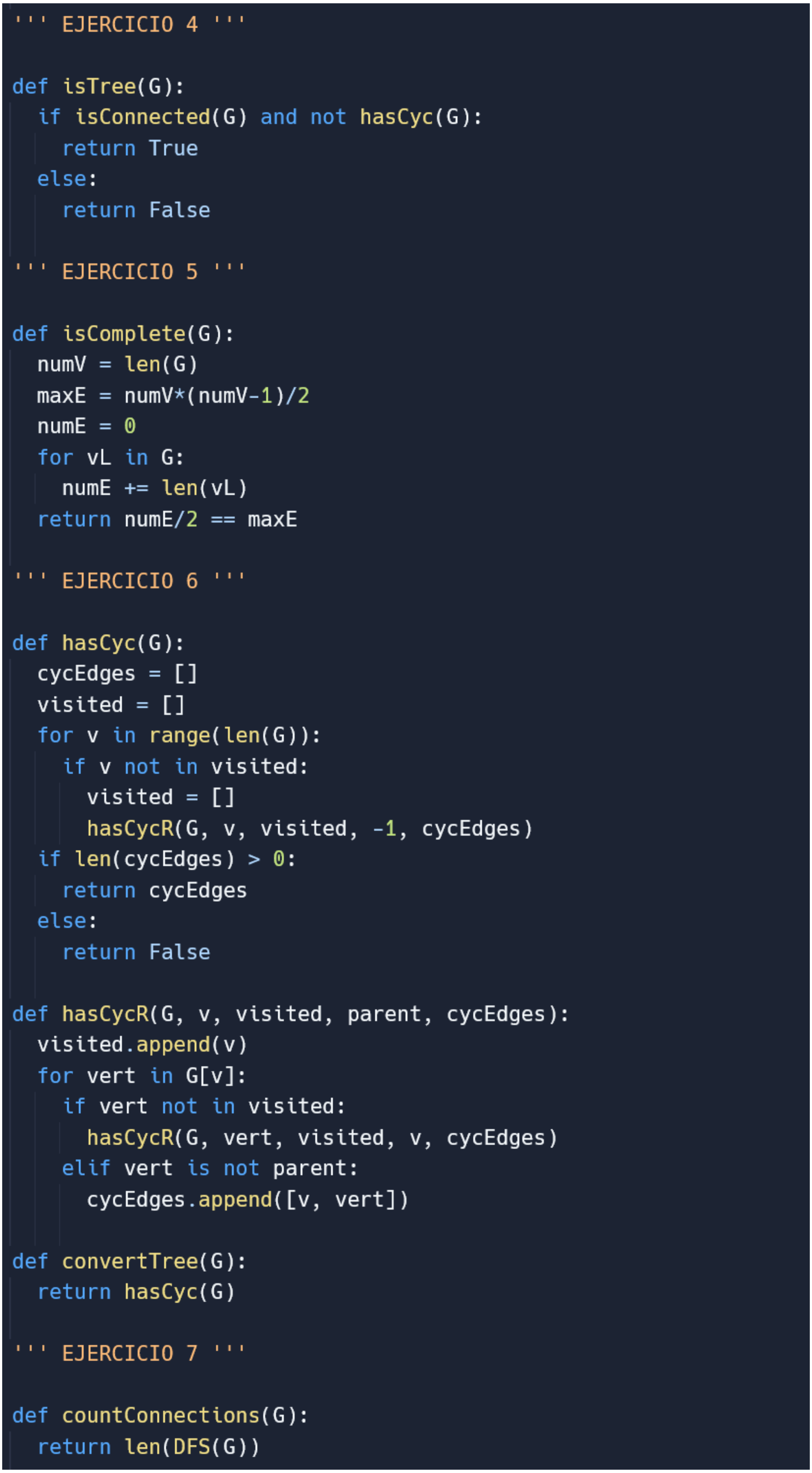
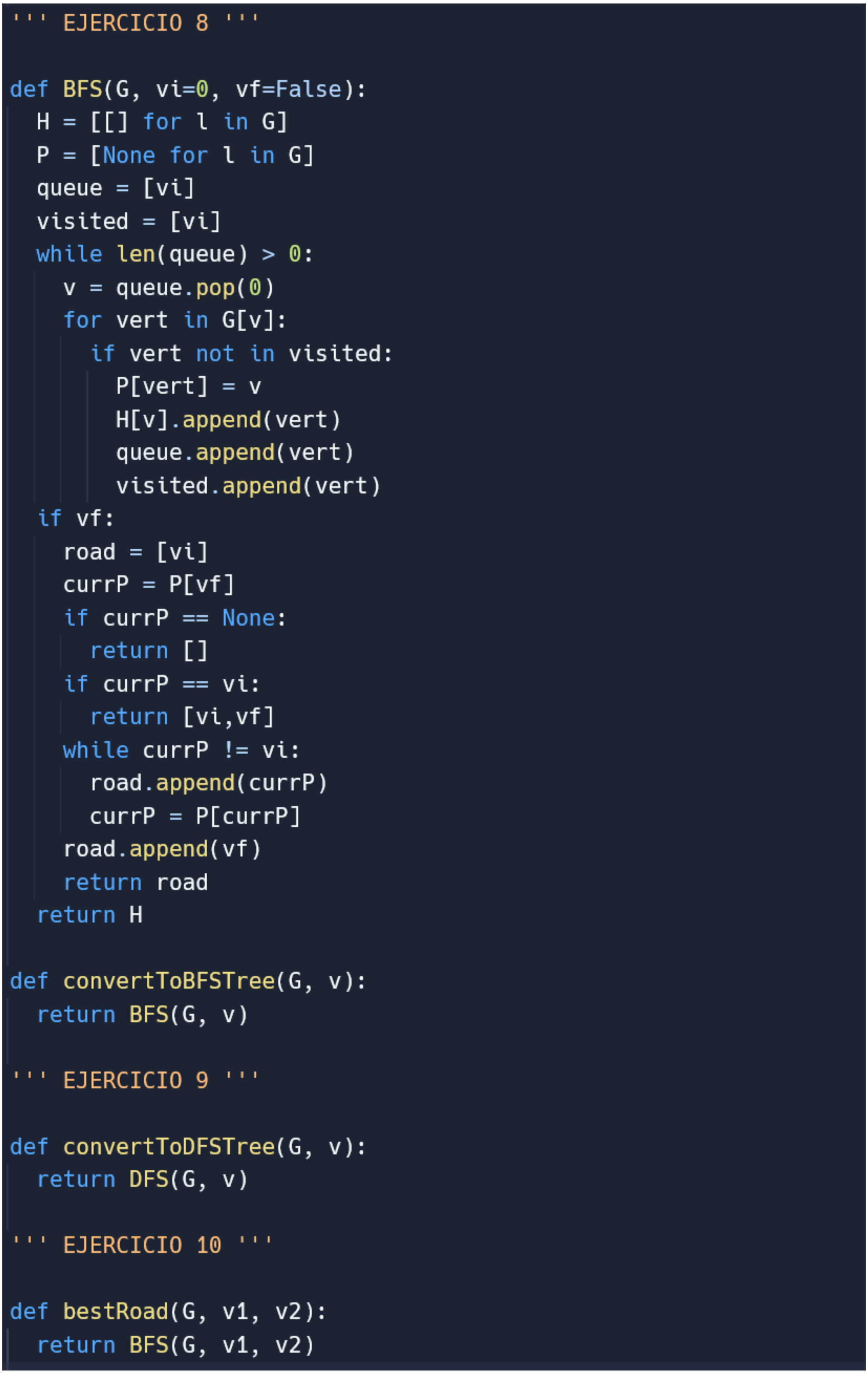


```python
''' EJERCICIO 1 '''

def createGraph(vL, eL):
  graph = [[] for v in range(len(vL))]
  for e in eL:
    if len(e) == 2:
      #if e[1] not in graph[e[0]]:
        graph[e[0]].append(e[1])
        graph[e[1]].append(e[0])
  return graph

''' EJERCICIO 2 '''

def DFS(G, vi=None, vf=None, getRoad=False):
  visited = []
  if vi == None:
    components = []
    for v in range(len(G)):
      if v not in visited:
        visited = []
        DFSR(G, v, vf, visited)
        components.append(visited)
    return components
  else:
    DFSR(G, vi, vf, visited)
    if vf != None:
      if visited[-1] == vf:
        if getRoad:
          return visited
        else:
          return True
      else:
        return False
    else:
      return visited

def DFSR(G, vi, vf, visited):
  visited.append(vi)
  if vi == vf:
    return
  for v in G[vi]:
    if v not in visited:
       DFSR(G, v, vf, visited)

def existPath(G, v1, v2):
  return DFS(G, v1, v2)

''' EJERCICIO 3 '''

def isConnected(G):
  components = DFS(G)
  if len(components) > 1:
    return False
  return True
```



```
''' EJERCICIO 4 '''

def isTree(G):
  if isConnected(G) and not hasCyc(G):
    return True
  else:
    return False

''' EJERCICIO 5 '''

def isComplete(G):
  numV = len(G)
  maxE = numV*(numV-1)/2
  numE = 0
  for vL in G:
    numE += len(vL)
  return numE/2 == maxE

''' EJERCICIO 6 '''

def hasCyc(G):
  cycEdges = []
  visited = []
  for v in range(len(G)):
    if v not in visited:
      visited = []
      hasCycR(G, v, visited, -1, cycEdges)
  if len(cycEdges) > 0:
    return cycEdges
  else:
    return False

def hasCycR(G, v, visited, parent, cycEdges):
  visited.append(v)
  for vert in G[v]:
    if vert not in visited:
      hasCycR(G, vert, visited, v, cycEdges)
    elif vert is not parent:
      cycEdges.append([v, vert])

def convertTree(G):
  return hasCyc(G)

''' EJERCICIO 7 '''

def countConnections(G):
  return len(DFS(G))
```



```python
''' EJERCICIO 8 '''

def BFS(G, vi=0, vf=False):
  H = [[] for l in G]
  P = [None for l in G]
  queue = [vi]
  visited = [vi]
  while len(queue) > 0:
    v = queue.pop(0)
    for vert in G[v]:
      if vert not in visited:
        P[vert] = v
        H[v].append(vert)
        queue.append(vert)
        visited.append(vert)
  if vf:
    road = [vi]
    currP = P[vf]
    if currP == None:
      return []
    if currP == vi:
      return [vi,vf]
    while currP != vi:
      road.append(currP)
      currP = P[currP]
    road.append(vf)
    return road
  return H

def convertToBFSTree(G, v):
  return BFS(G, v)

''' EJERCICIO 9 '''

def convertToDFSTree(G, v):
  return DFS(G, v)

''' EJERCICIO 10 '''

def bestRoad(G, v1, v2):
  return BFS(G, v1, v2)
```