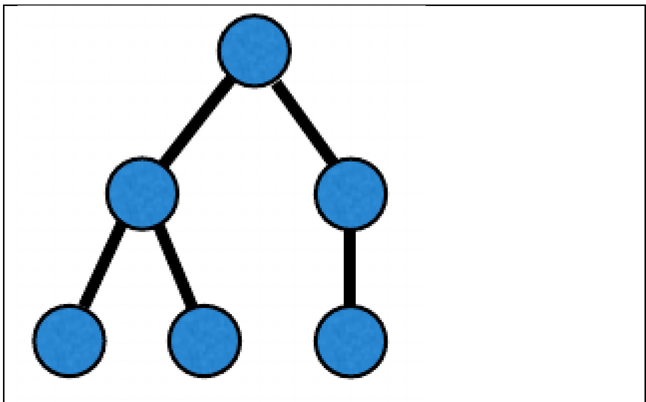
1. Discuss the differences between trees and graphs.

A graph is any ordered pair that can be defined as G = (V, E) where V is a set of vertices, also called nodes, and E is a set of edges, also called links, that represents the relationships between the vertices, and E=(u, v) represents the edge from vertex u to vertex V (Carmen et al., 2022). This is a very broad category that includes undirected graphs, where edges have no orientation making the edge (u, v) the same as edge (v, u), directed graphs where edges do have an orientation resulting in the edge (u, v) is the edge from node u to node v, and weighted graphs where edges have a weight or value i.e. edge (u, v) having a weight of n. In addition, graphs have paths, which can be described as either a sequence of vertices connected by an edge or simply a sequence of edges (Carmen et al., 2022). A cycle is a subset of paths which initiate and terminate at the same node, and a graph is connected if there is a path between any two of its nodes (Carmen et al., 2022). Therefore, trees comprise a subset of graphs that are undirected and connected that have no cycles. This means that they will have *n* vertices and *n* - 1 edges.

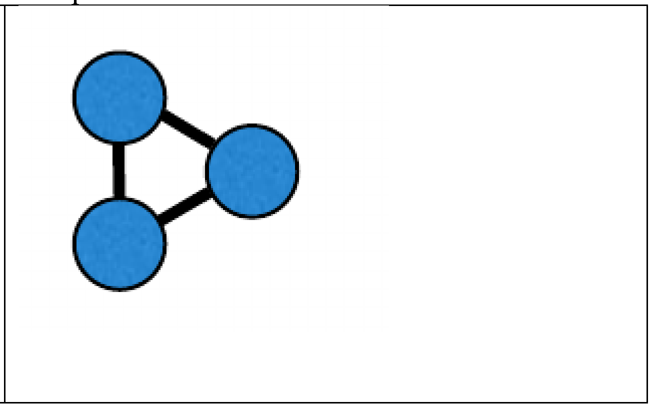
2. How would you represent tree and graph structure? I would use an adjacency list to represent a graph as it maps the graph's nodes to its edges. I would use an edge list to represent trees because their being connected and undirected with no cycles allows one to take advantage of the edge list's simplicity and practicality without experiencing its downsides.

3. Show an example of a tree and a graph.

Tree:



Graph:



4. Write down sample code for both tree and graph representation and initialize it with the previously provided examples.

```
#Use dictionary for adjacency list graph G=(V,E) and E=(u,V)
G = {} #create empty dictionary for graph

def _V(G, V): # function to create node
  if V not in G: # check to see if node already exists
        G[V] = [] #if not add node as key with an empty list value for edges

def _E(G, u, V, directed="True"): #function to add edges to nodes
  _V(G, u)          #add node u to graph if not present
  _V(G, V)          #add node V if not present
  G[u].append(V)        #add edge between u and V

  if not directed: #if undirected, add edge in opposite direction
    G[V].append(u)         #add edge between V and u

#The graph example is a complete K3 graph, making it undirected.
#Initialized as follows to fit example:
```

```
_E(G, 'A', 'B', False)
_E(G, 'A', 'C', False)
_E(G, 'B', 'C', False)
print(G)

#An edge list tree can be made using tuples for edges as items in a list.
#E = (u, V) for trees has V the child and u the parent.

_tree = [] #creates empty list

def make_tree(u, V):                #function to add edges to the tree
  _tree.append((u, V))    #adds the edge as a tuple to the list

#initialize to fit example
make_tree(0, 1)                          #adds edge between node 0 and node 1
make_tree(0, 2)                          #node 0 and node 2 edge
make_tree(1, 3)                          #node 1 and node 3 edge
make_tree(1, 4)                          #node 1 and node 4 edge
make_tree(2, 5)                          #node 2 and node 5 edge
print(_tree)
```

## References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. The MIT Press.