Python 3 Cheatsheet: Trees & Graphs

Tree Traversals (DFS)

```
# Recursive PostOrder (Left-Right-Root)
def postOrder(node):
    if not node: return
    postOrder(node.left)
    postOrder(node.right)
    print(node.val)

# Iterative PreOrder (Root-Left-Right)
def preOrder(root):
    stack = [root]
    while stack:
        node = stack.pop()
        if node:
            print(node.val)
            stack.append(node.right) # Push right first
            stack.append(node.left)
```

BFS (Level Order)

```
from collections import deque
def bfs(root):
    q = deque([root])
    while q:
        node = q.popleft()
        if node:
            print(node.val)
            q.append(node.left)
            q.append(node.right)
```

Graphs

Adjacency List

```
graph = defaultdict(list)
for u, v in edges:
    graph[u].append(v)
    graph[v].append(u)
```

Topological Sort (Kahn's Algo)

```
# Create In-Degree
in_degree = {u: 0 for u in nodes}
for u in graph:
    for v in graph[u]:
        in_degree[v] += 1

# Queue 0 in-degree
q = deque([u for u in in_degree if in_degree[u] == 0])
res = []
while q:
    u = q.popleft()
    res.append(u)
    for v in graph[u]:
        in_degree[v] -= 1
        if in_degree[v] == 0:
            q.append(v)
```

Union Find (Disjoint Set Union)

```
class DSU:
    def __init__(self, N):
        self.parent = list(range(N))
    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x]) # Path compression
        return self.parent[x]
    def union(self, x, y):
        rootX, rootY = self.find(x), self.find(y)
        if rootX != rootY:
            self.parent[rootX] = rootY
            return True
        return False
```

Trie (Prefix Tree)

```
class Trie:
    def __init__(self):
        self.root = {}

    def insert(self, word):
        node = self.root
        for c in word:
            if c not in node: node[c] = {}
            node = node[c]
        node['#'] = True # End marker
```