2**. Trees are useful for many programming operations such as sorting and searching. There are a number of ways they can be stored including in tables, as objects, etc. What are some of the good ways you've found to store trees, and what are the advantages and disadvantages or each? Or asked another way, how to you pick when to use which kind of storage? Feel free to reference Python libraries.**

Ans. Storing Trees in programming depends on the use cases for a particular problem, there are several approaches below are few to name a few.

1. **Adjacency List:** Each node of the tree is represented as an object or a record containing references to its child nodes. They can be implemented using Lists, Arrays, Dictionaries or Linked Lists.

- **Advantages:** It is memory-efficient for sparse trees, as it stores only the specific things. It is easy to implement and works well for more tree operations.

- **Disadvantages:** Traversing the tree requires following pointers and it can be slower compared to other pointers. It may require additional storage to store the pointers.

2. Adjacency Matrix: The Matrix is used to store the connections between the nodes. if there is a connection then there is 1 or else 0. The Matrix can be stored using **numpy** Arrays.

- **Advantages:** It is used when the tree is done to check the relationship between the nodes. It is useful when edge operations are frequent.

- **Disadvantages:** Consumes More memory if it is Sparse trees.

3. Parent Pointer Tree: In this approach each node contains references to their parents, This will be useful where upward navigation of tree is common.

- **Advantages:** It easily enables for navigation from Child to Parent node. With this the operations like Moving up the tree is more efficient.

- **Disadvantages:** Searching the child nodes and requires additional storage for storing the references of parents.
- **Python libraries: anytree, treelib, networkx, pydot** libraries can be used.