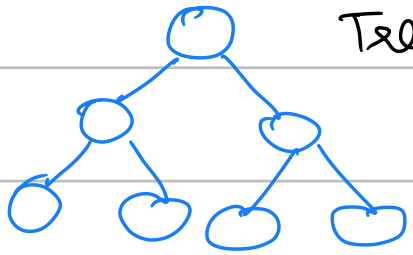
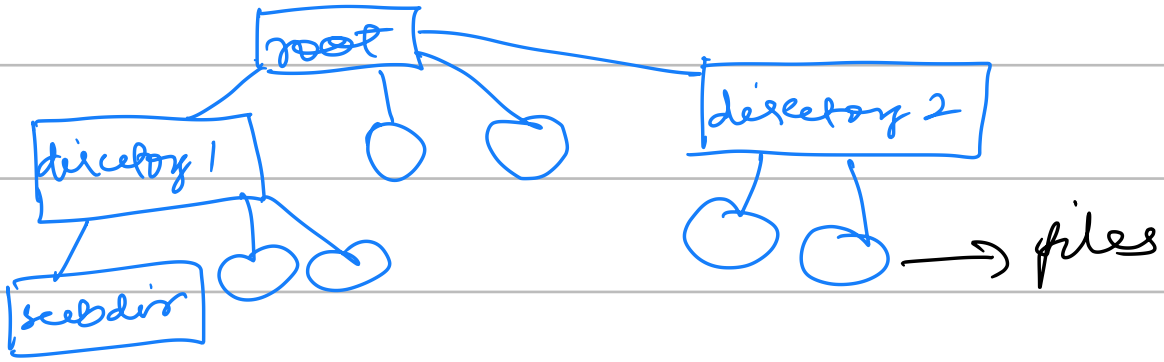
Trees in Python



Tree structure

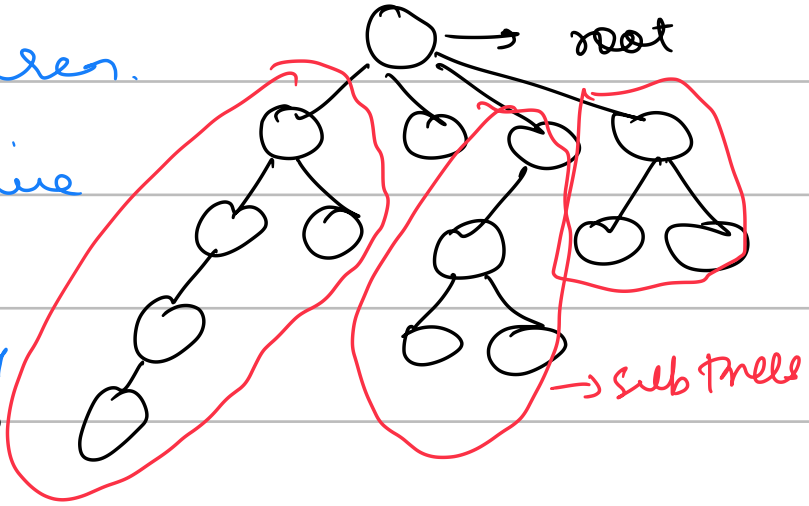
Mostly we see use of tree in our file structure



Another example is a companies hierarchy

What exactly is a tree structure?

There exist a node, root node main node, this node can have more nodes as its children. Now each of these children can further have more children.



Trees a highly recursive structure and we mostly use recursion to solve trees.

* Binary trees

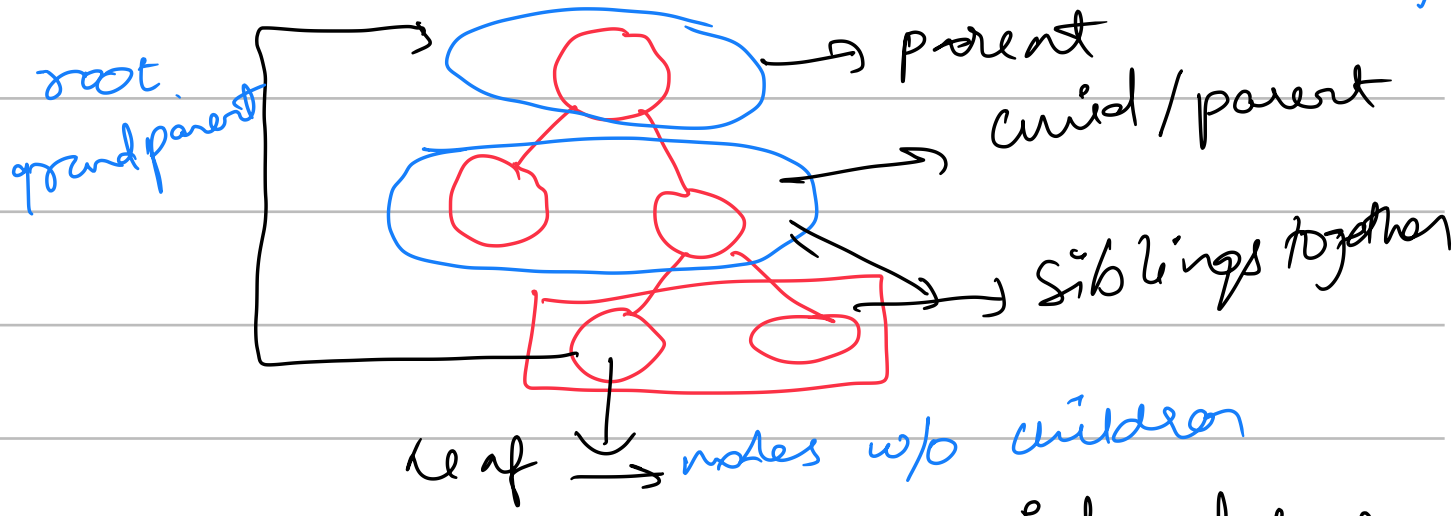
In binary trees each node can only have either 0 or 1 or 2 nodes.

* Terminology of trees

Node → individual data point in tree

Root → top most node is called root.

Its like "Head" in LinkedList. We can call the whole tree or traverse the whole tree through root.



Imp terms → node, root, parent, child and leaf.

* So what will a node store

⇒

This way Node will have 3 attributes

Address of child 1

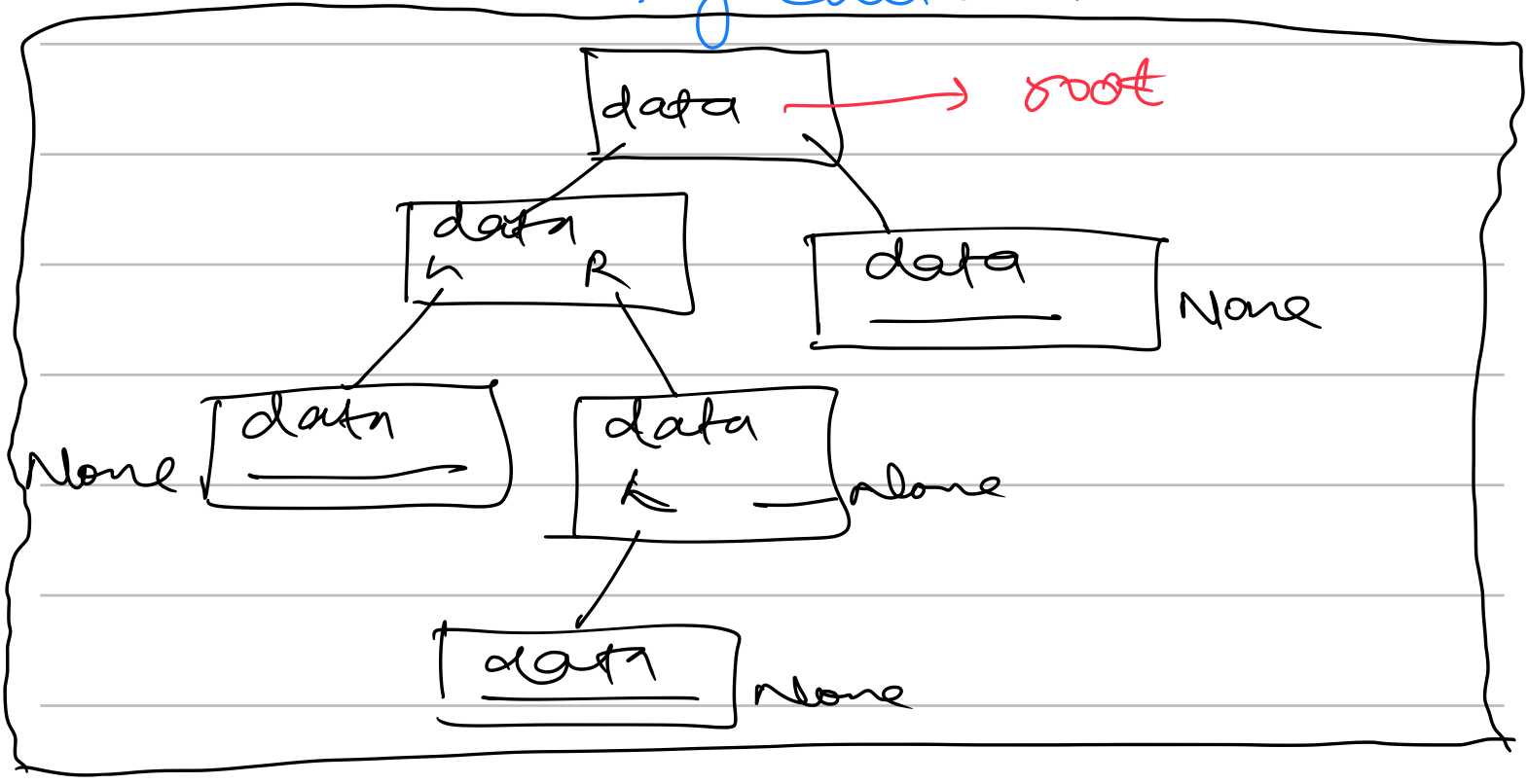
Address of child 2

1.) Data
2.) Address of child 1
3.) Address of child 2

In case of binary trees these children are called left & Right.

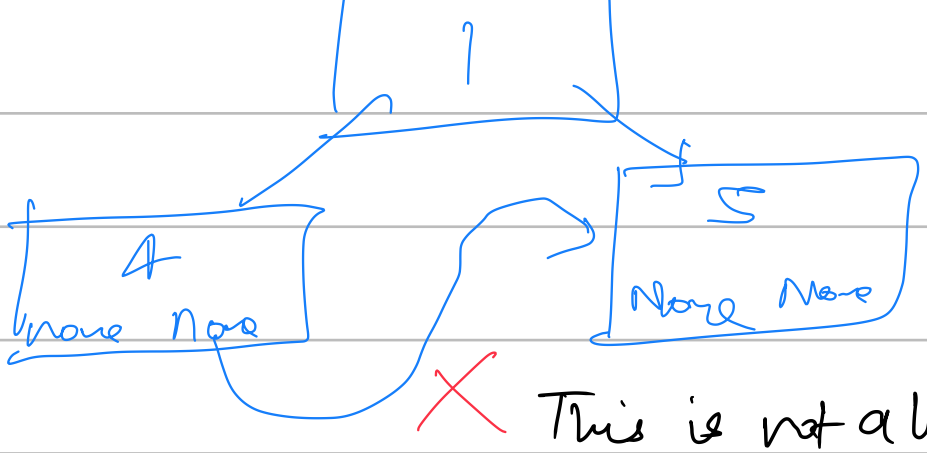
↓

Keep Address of the particular nodes children. The addresses can also be None meaning that particular node doesn't have any children.



Note - Check github for code for BT

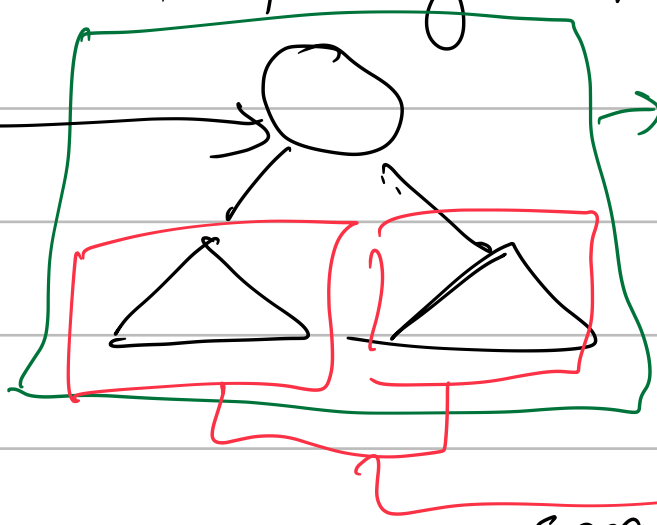
Note → Trees doesn't have cycles



This is not allowed in trees

```python
class BinaryTreeNode:

    def __init__(self,data):
        self.data = data  #our node should have some data
        self.left = None
        self.right = None   #two address holders for left node and right node

rootNode = BinaryTreeNode(2)
node1 = BinaryTreeNode(4)
node2 = BinaryTreeNode(8)

#connected children to parent node
rootNode.left = node1
rootNode.right = node2

print(rootNode.data)
print(rootNode.left.data)
print(rootNode.right.data)
```

Note → most of the tree problems gets solved through recursion.



we will print roots data and then use recursion on the remaining left & right subtree

what what will be the base case?

(1) Is root a leaf node, then there is no point on calling on the left node & right node.

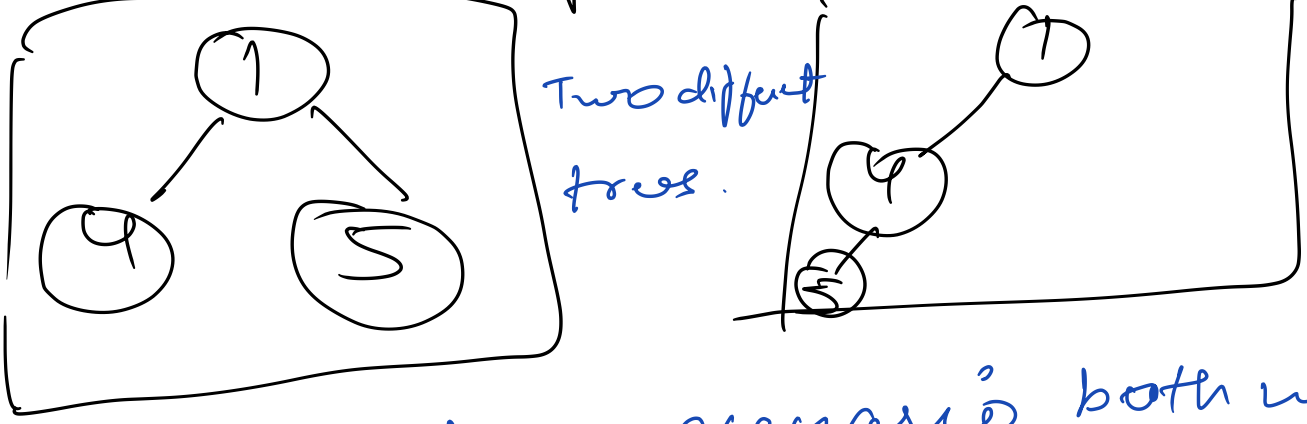
Now there are two problems with this approach

I) what if root is "None" i.e tree is empty.

II) we will have a overhead of checking if left is None then don't call, if right is "none" then don't call.

So in most of our cases we take the base case that "If root is None"

But this still does not tell us the complete picture



Using the above scenario both will print 1, 4, 5. But there is confusion b/w the parent child relationship.

So what to do?....

⇒ need to print parent child relation. we need to print in a way like

root: L, R or 1 : 24, R5

So to print left nodes data we can do root.left.data ⇒ this print left childs data.

" Now this can only be root.left != not None "

or else we will get an error.

Taking user input for Binary Tree?

We take input recursively as well user gives root and take input for

left subtree and right subtree.
All this input for left & right subtree will be through recursion.

Note :- while taking input the whole first subtree whether be it left subtree or right subtree has to be completed before moving to the other subtree.

---

given root find number of nodes in tree?

Base case if root is None return 0. Otherwise call recursion on left tree, then on the right. Add 1 + leftcount + rightcount and you are done.

Believe in Recursion, just think About Root

Traversals on Binary Tree

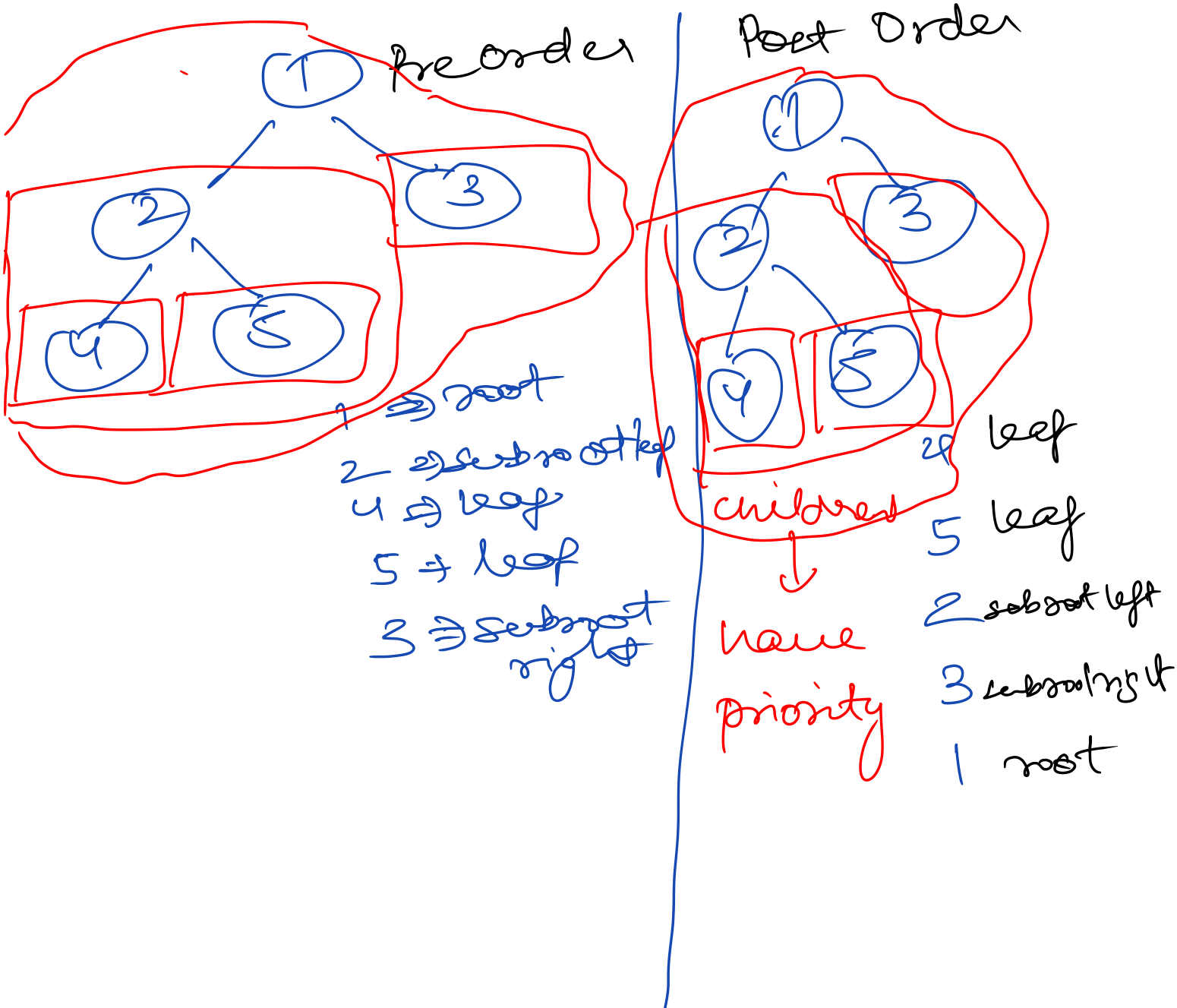
a) Pre order

Root left Right

when root takes for himself before children

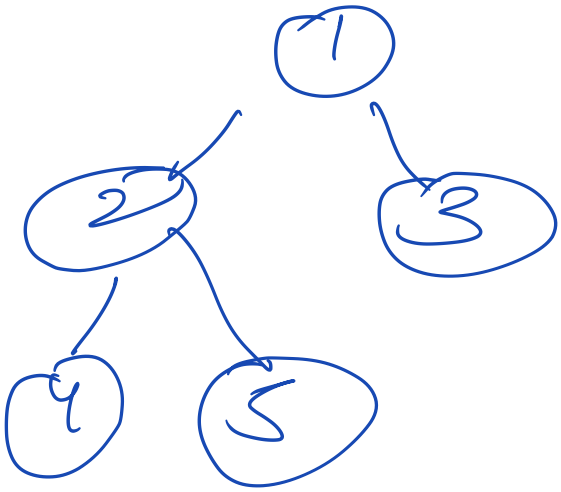
b) Post Order

First children are taken care of before the root



In order Traversal

left ⇒ root ⇒ right



4 → left leaf

2 ⇒ subroot left

5 ⇒ right leaf

1 ⇒ root

3 ⇒ subroot right

Time Complexity of Trees.

```
def numNodes(root):
    base case
    a = numNode(left)
    b = numNodes(right)
    return 1 + a + b
```