# Trees

### Let's first understand: 1) What is a Tree data structure? and 2) How to create a Tree?

### 1) A Tree is a data structure consisting of nodes arranged in Tree like structure, where each node will contain some data and references to it children nodes. The top most node is called the root node. To access the complete Tree, reference of only root node is required.

### 2)Therefore, to create a Tree, we need to define Node class. Example code is as follows:

In [2]:
```
#Here, we will take an example of binary tree i.e. each node can have maximum two children nodes (left child and right child)
#and the data for that node
class Node:
    #constructor to declare and initialize node properties
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
#Say, we have a root node containing a list of numbers and we want to keep on splitting the list around mid index
#until we reach the leaf node
#For this, let's create a recursive function, which will take a list of numbers and return the root node of the tree
def build_tree_example(data):
    #base case
    if len(data) == 1:
        #only one number in the list so make a node having no children and return it
        return Node(data)
    #otherwise, make two lists of data for left branch and right branch by splitting around mid index
    mid_index = len(data) // 2
    data_left = data[0: mid_index]
    data_right = data[mid_index:]
    #create root node with the data list
    root = Node(data)
    #recursively build the left subtree and right subtree
    root.left = build_tree_example(data_left)
    root.right = build_tree_example(data_right)
    #return root node of the tree
    return root
#function to print the tree recursively
def print_tree(root):
    #check if root is empty
    if root == None:
        return
    #check if it's a leaf node
    if root.left == None and root.right == None:
        print(root.data)
        return
    #otherwise, print the root node data and recursively call print function on left and right child
    print(root.data)
    print_tree(root.left)
    print_tree(root.right)

#Let's test it
data = [2, 3, 1, 4, 8, 7, 6, 10]
root = build_tree_example(data)
print_tree(root)
```

```
[2, 3, 1, 4, 8, 7, 6, 10]
[2, 3, 1, 4]
[2, 3]
[2]
[3]
[1, 4]
[1]
[4]
[8, 7, 6, 10]
[8, 7]
[8]
[7]
[6, 10]
[6]
[10]
```

**Now that we are comfortable with how to build and print a tree in python, it's time to build the decision tree. You can take help of the hints provided below:**

In [3]:
```
#The task for this project is that given some training data, we need to build a tree structure, where each node will make a d
ecision based on
#value of a particular feature.
```

In [4]:
```
#first, we need to define some helper functions and classes, which will be used while building the tree
```

In [5]:
```
def class_counts(rows):
    """This function will take some rows of dataset and
        will return a dictionary mapping each label with number of rows for that label
    """
```

In [6]:
```
class Question:
    """This class will help in creating Question objects having two properties:
        (i) column/feature and (ii) value for that feature.
        we can have a method in this class to check whether given example matches the value for the feature in this Question
 or not.
        It will be useful in partitioning the data based on a particular value for a particular feature/column.
    """
```

In [7]:
```
def partition(rows, question):
    """
    This function takes list of rows (examples) and after checking for each example, whether it matches the feature value in
    question or not, it will return two separate list of rows
    """
```

In [8]:
```
def entropy(rows):
    """
    This function will take some rows of the dataset and return the entropy of these set of rows.
    It will be used in calculating the info gain for a split based on a question.
    """
```

In [9]:
```
def info_gain(true_rows, false_rows, current_entropy):
    """
    This function will take the current entropy of the dataset and two different sets of rows after split
    It will calculate the entropy for these two set of rows using entropy function above
    and will return the info gain for that split
    It will be useful in finding the best split for dataset/ given rows.
    """
```

In [10]:
```
def find_best_split(rows):
    """
    This function will take a set of examples i.e. rows as input
    and return the max info gain and corresponding question.
    """
    #Iterate over each column
        #Iterate over each of the unique values fo the column
            #find and return the max gain and best question
```

In [11]:
```
class Decision_Node:
    """
    It will have three properties
    (i) question
    (ii) child node representing true branch
    (iii) child node representing false branch
    """
```

In [12]:
```
class Leaf:
    """
    Unlike Decision Node, it just gives label for the example that reaches this leaf node.
    It just holds a dictionary mapping as obtained from class_counts function above for given set of rows.
    """
```

In [13]:
```
def build_tree(rows):
    """
    This function will build the tree recursively and return the root node of the tree
    """
    #find gain and question using find_best_split function
    #check for base case i.e. no further split is possible (gain = 0), then return Leaf
    #otherwise
    #do the partition using partition function above and get two list of rows say true_rows and false_rows
    #Build true_branch and false_branch by recursively calling on true_rows and false_rows
    #return the Decision Node with question, true_branch and false_branch
```