recursion, natural and otherwise
structure to organize information
patriarchal family tree...
terminology

- set of nodes (possibly with values or labels), with directed edges between some pairs of nodes

- One node is distinguished as root

- Each non-root node has exactly one parent.

- A path is a sequence of nodes $n_1, n_2, \ldots, n_k$, where there is an edge from $n_i$ to $n_{i+1}$. The length of a path is the number of edges in it

- There is a unique path from the root to each node. In the case of the root itself this is just $n_1$, if the root is node $n_1$.

- There are no cycles — no paths that form loops.
more terminology

- **leaf**: node with no children
- **internal node**: node with one or more children
- **subtree**: tree formed by any tree node together with its descendants and the edges leading to them.
- **height**: $1 + \text{the maximum path length in a tree}$. A node also has a height, which is $1 + \text{the maximum path length of the tree rooted at that node}$
- **depth**: Height of the entire tree minus the height of a node is the depth of the node.
- **arity, branching factor**: maximum number of children for any node.
class Tree:
    
    A bare-bones Tree ADT that identifies the root with the entire tree.
    
    def __init__(self, value=None, children=None):
        
        Create Tree self with content value and 0 or more children

        @param Tree self: this tree
        @param object value: value contained in this tree
        @param list[Tree] children: possibly-empty list of children
        @rtype: None
        
        self.value = value
        # copy children if not None
        self.children = children.copy() if children else []
def height(t):
    """
    Return 1 + length of longest path of t.
    @param Tree t: tree to find height of
    @rtype: int
    >>> t = Tree(13)
    >>> height(t)
    1
    >>> t = descendants_from_list(Tree(13), [0, 1, 3, 5, 7, 9, 11, 13]
    >>> height(t)
    3
    """
    # 1 more edge than the maximum height of a child, except
    # what do we do if there are no children?
general form of recursion:

if (condition to detect a base case):

    (do something without recursion)

else: # (general case)

    (do something that involves recursive call(s))
def leaf_count(t):
    """
    Return the number of leaves in Tree t.
    @param Tree t: tree to count number of leaves of
    @rtype: int
    >>> t = Tree(7)
    >>> leaf_count(t)
    1
    >>> t = descendants_from_list(Tree(7), [0, 1, 3, 5, 7, 9, 11, 13], 3)
    >>> leaf_count(t)
    6
    """
def arity(t):
    
    """
    Return the maximum branching factor (arity) of Tree t.
    
    @param Tree t: tree to find the arity of
    @rtype: int
    """

    >>> t = Tree(23)
    >>> arity(t)
    0
    >>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(5), Tree(5.75)])
    >>> tn3 = Tree(3, [Tree(6), Tree(7)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> arity(tn1)
    4
    """