1. 创建一个树（Tree）的抽象基类，内包含一个抽象基类 Position，要求如下：
   Position 类函数：
   element()：返回 position 存储的元素，在抽象基类内，暂时先 raise NotImplementedError
   __eq__()：等于，函数体暂时先定义 raise NotImplementedError
   __ne__()：不等于，函数体暂时先定义 raise NotImplementedError

   Tree 类函数：
   root()：返回树的根节点，函数体暂时先定义 raise NotImplementedError
   parent(p)：返回 p 的父结点，函数体暂时先定义 raise NotImplementedError
   num_children(p)：返回 p 的子结点的数量，函数体暂时先定义 raise NotImplementedError
   children(p)：返回 p 的子结点的集合，函数体暂时先定义 raise NotImplementedError
   __len__()：函数体暂时先定义 raise NotImplementedError
   is_root(p)：确定 p 是否是树的根结点
   is_leaf(p)：确定 p 是否是一个叶子结点
   is_empty()：确定树是否是空树
   depth(p)：计算一个结点的深度
   height()：计算树的高度
2. 基于树，创建一个二叉树的抽象基类，并定义以下函数
   T.left(p)：返回 p 的左子结点，函数体暂时先定义 raise NotImplementedError
   T.right(p)：返回 p 的右子结点，函数体暂时先定义 raise NotImplementedError
   T.sibling(p)：返回 p 的兄弟结点

3. 创建一个 2 中描述类的子类，通过使用链式结构为底层数据结构，实现二叉树和 1、2 内所有函数，要求如下：
   创建一个 Node 内建类，包含 element、parent、left、right 四个属性
   创建 1 里面 Position 类的子类，包含 container 和 node 两个属性
   实现_validate(p)函数，以验证 position 是否有效
   实现_make_position(node)函数，将一个 node 封装成一个 position
   实现_add_root(e)函数，根据 e 创建一个 node，然后把此结点变成树的根结点，并返回此结点的 position（通过_make_position）
   实现_add_left(p, e), _add_right(p, e)，给一个 position 添加左侧结点和右侧结点
   _replace(p, e)：用 e 替换 p 现在的 element，返回旧的 element 的值
   _delete(p)：删除 p 处的结点，如果 p 有两个子结点，raise error，如果 p 有一个子结点，将子结点挪到 p 处
   _attach(p, t1, t2)：将 t1，t2 两个树分别设置成 p 的左子树和右子树，如果 p 不是叶子结点，raise error，如果 t1，t2 不是树，raise error。
4. 创建一个 2 中描述类的子类，通过使用数组为底层数据结构，实现二叉树和 1、2 内所有函数。