Welcome Back!!....Trees continued

Arnamoy Bhattacharyya

Announcements

- Test1: returned. You have till Saturday for remark
  - Avg: 76% Median 80%
- Demo1: Almost ready to be returned
  - Avg 80%
- A2: Has been out for a while
  - Check Piazza, has been some good discussions
  - Submit whatever you have, then resubmit

Agenda

1. Revisit tree class
2. Module Level Functions vs Class Level Methods in Trees
3. Example of Tree in OS file system

Generic Tree Class

```
class Tree:
    """
    A bare-bones Tree ADT that identifies the root with the entire tree.
    """
    def __init__(self, value=None, children=None):
        """
        self.value = value
        # copy children if not None
        self.children = children.copy() if children else []
```

```
>>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(5)])
>>> tn3 = Tree(3, [Tree(6), Tree(7)])
>>> tn1 = Tree(1, [tn2, tn3])
```

Functions for trees

def height(tree):

def arity(tree):

def count_leaves(tree):

Module Level vs Class Level

Q1. How can you differentiate between a Module level **function** and a Class level **method**?

Module Level vs Class Level

Q1. How can you differentiate between a Module level **function** and a Class level **method**?

A. self argument

B. by looking at the indentation (module level functions will not have any indent before it)

Module Level vs Class Level

Q2. When to choose a Module level **function** over a Class level **method**?

Module Level vs Class Level

Q2. When to choose a Module level **function** over a Class level **method**?

A. When the “behaviour” is closely associated with the class, make it class level
   a. E.g get_salary() of an Employee, get_area() of a Shape, height() of a tree

It is your design decision

In A2, you have a module level function. What is it?

Exercise 1: Tree contains

```
def __contains__(self, v):
    """
    Return whether Tree self contains v.

    @param Tree self: this tree
    @param object v: value to search this tree for

    >>> t = Tree(17)
    >>> t.__contains__(17)
    True
    >>> t = descendants_from_list(Tree(19), [1, 2, 3, 4, 5, 6, 7], 3)
    >>> t.__contains__(5)
    True
    >>> t.__contains__(18)
    False
```

Exercise 2: Tree count nodes

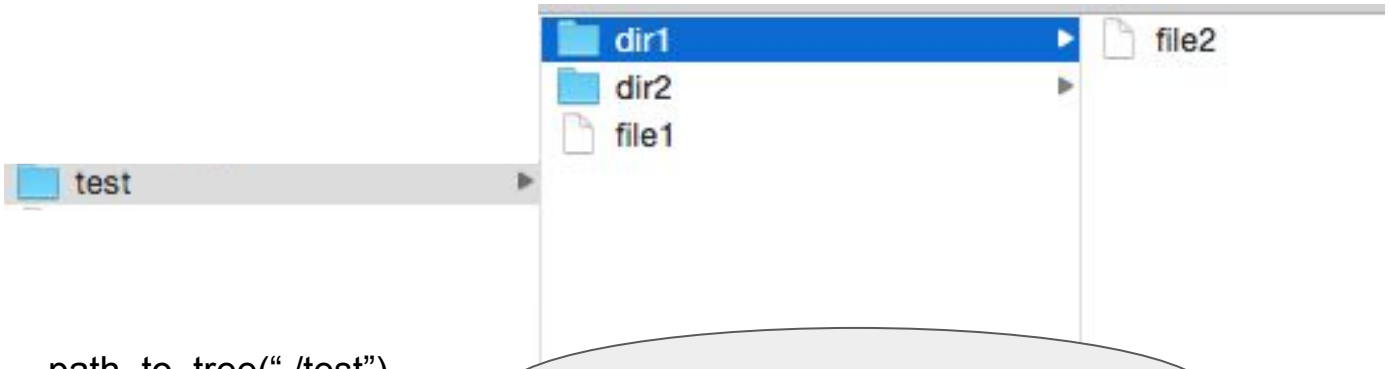
```
def count_nodes(self) -> int:
    """
    Return node count of Tree self.

    @param Tree self: this tree
    @rtype int

    >>> t = Tree(17)
    >>> t.count_nodes()
    1
    >>> t = descendants_from_list(Tree(19), [1, 2, 3, 4, 5, 6, 7], 3)
    >>> t.count_nodes()
    """
```

Use case: File system exploration using Trees

1. Explore the file system structure
2. Use our tree class methods to play around



path_to_tree(“./test”)