Lab 18: Tree Recursion in Python

A supplementary guide with a question walkthrough and hints can be found here.

Instructions:

This worksheet serves as a guide and set of instructions to complete the lab.

- You must use the starter file, found here, to get credit for the lab.
- File is named "tree-lab-starter.py"
- There is no workbook for this lab. Review the Lab 8 workbook for review on recursive functions.

Notes:

All functions for this lab must use recursion.

Submitting:

You will need to complete Exercises 1-3 and submit this to Gradescope. To receive full credit, you will need to complete the required functions, and the required functions must pass all tests from the autograder in Gradescope. For instructions on how to submit to labs to Gradescope, please see this document.

Please note, you must use the starter file, and you must NOT edit the name of any of the required functions. Failing to do either for these will result in the autograder failing.

Running Doctests:

- To run doctests embedded within the docstring of a function, you can use the command:
- ** make sure to replace "foo" with the title of your starter file
  ```
  python3 -m doctest -v foo.py
  ```

- To run all doctests you can run:
  ```
  python3 -m doctest foo.py
  ```

Objectives:

Recursion is arguably the second most important topic in this course (after abstraction of course). You've seen and worked with many examples of recursion in Snap, although it's time we practice in Python. In today's lab you will implement what you know about recursion into Python functions. By the end of the lab, you will:

- Understand how to implement tree recursion in Python.
- Practice planning and coding recursive functions.
- Implement higher-order functions in non-iterative solutions.
- Write tree recursive functions with helper functions
- Use OOP and recursion to write functions for tree data structures

Required Functions:

- make_subsets(lst)
- fibonacci(n)
- find_height(node)

Exercise 1: make_subsets(lst)

- Objective:
    - Create a function that finds all subsets of a given list. A subset is any combination of elements from the original list, including the empty set and the list itself.
- Inputs:
    - `lst (list)`: The list from which to find subsets.
- Outputs:
    - `list of lists`: A list containing all subsets of the original list.
- Examples:

```python
>>> make_subsets([])
[[]]
>>> make_subsets([1])
[[], [1]]
>>> make_subsets([1, 2])
[[], [1], [2], [1, 2]]
>>> make_subsets([1, 2, 3])
[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]
>>> make_subsets(["a", "b", "c", "d"])  # Example provided
[[], ['a'], ['b'], ['a', 'b'], ['c'], ['a', 'c'], ['b', 'c'], ['a', 'b', 'c'], ['d'], ['a', 'd'], ['b', 'd'], ['a', 'b', 'd'], ['c', 'd'], ['a', 'c', 'd'], ['b', 'c', 'd'], ['a', 'b', 'c', 'd']]
```

Exercise 2: fibonacci(n)

- Objective:

    - Create a function that calculates the n-th Fibonacci number using tree recursion with memoization. Memoization is an optimization technique that stores the results of expensive function calls and reuses them when the same inputs occur again.
    - You MUST use the helper function provided.
- Inputs:
    - `n (int)`: The position in the Fibonacci sequence.
- Outputs:
    - `int`: The n-th Fibonacci number.
- Examples:

```python
>>> fibonacci(0)
0
>>> fibonacci(1)
1
>>> fibonacci(2)
1
>>> fibonacci(5)
5
>>> fibonacci(10)  # Example provided
55
>>> fibonacci(30)  # Example provided
832040
```

Exercise 3: find_height(node)

- Objective:
    - Create a function that finds the height of a k-tree using tree recursion. The height of a tree is the number of edges from the root to the furthest leaf node.
    - You will be using the KTreeNode class and an instance of this class will be what is passed in to the function. You will need to use one of the attributes of this class.
- Inputs:
    - `node (KTreeNode)`: The root node of the tree.
- Outputs:
    - `int`: The height of the tree.
- Examples:

```python
>>> root = KTreeNode(1)
>>> find_height(root)
0
>>> child1 = KTreeNode(2)
>>> root.children.append(child1)
>>> find_height(root)
1
>>> child2 = KTreeNode(3)
>>> root.children.append(child2)
>>> child3 = KTreeNode(4)
>>> child1.children.append(child3)
>>> find_height(root)
2
>>> child4 = KTreeNode(5)
>>> child3.children.append(child4)
>>> find_height(root)  # Example provided
3
```

You can always check the validity of your solutions by using the local autograder. Remember to submit on Gradescope!