Title: Mythical Creature Binary Tree Algorithm

Author: Shaun Clarke

Goal: This program allows us to create a binary tree and access its descendants or print the entire tree.

Steps:

```
1. Define a class Creature:
    a. This class has the basins needed to creates a creature.
    b. The constructor takes the creature’s name and initializes the following:
        i. The creature’s name.
        ii. Left set to none.
        iii. Right set to none.
    c. Define a dunder method __str__(self):
        i. This method prints the name when the class is printed.
        ii. It returns the creature name.
2. Define a class CreatureTree:
    a. This class creates and prints the creature tree.
    b. The constructor takes no parameters and initializes the following:
        i. Root set to None.
    c. Define a method add_root(self, name: str) -> str:
        i. This method adds the root node to the tree
        ii. Checks if root node already exists.
        iii. If it doesn’t exits add it and return confirmation to user.
    d. Define a method find(self, node: Creature, name: str) -> object:
        i. This method finds a specified node in the creature tree, using recursion.
        ii. If the node doesn’t exist:
            1. Return none.
        iii. If the node in question is found:
            1. Return that node.
        iv. If the node was not found, recursively check the left subtree.
        v. If the node was found on the left:
            1. Return it.
        vi. Else, recursively check the right subtree.
        vii. Return if found on right
    e. Define a method add_creature(self, parent_name: str, side: str, child_name: str)
       -> str:
        i. This method adds a creature to a parent node using recursion.
        ii. Call the find method to get the parent node.
        iii. If the parent node was not found, let the user know.
        iv. If the creature is being added to the left side:
            1. If the paren.left doesn’t exist, add the creature there.
            2. Else, call add_creature to recursively check the parent.left
               subtree to find an available spot to add the creature.
       v. If the creature is being added to the right side:
            1. If the parent.right  doesn’t exist, add the creature there.
            2. Else, call add_creature to recursively check the parent.left
               subtree to find an available spot to add the creature.
      vi. If left or right tree was not selected, let the user know their choice was
          invalid.
  f. Define a method print_tree(self):
       i. This method prints the complete tree structure.
      ii. If there Is no root node:
            1. Let the user know.
     iii. Otherwise, print the root node.
      iv. Check if left and right child nodes exist and return a Boolean value for
          each.
       v. If left is true:
            1. Assign self.root.left.name to a vriable called left.
      vi. else assign “” to the left variable.
     vii. If right is true:
            1. Assign self.root.right.name to a vriable called right.
    viii. else assign “” to the right variable.
      ix. If there is a left node print the tree connector
       x. Else, fill the space so the tree is aligned.
      xi. If there is a right node:
            1. print the tree connector
     xii. Else:
            1. fill the space so the tree is aligned.
    xiii. Print the left and right node below the connectors.
     xiv. Set the following variables to empty quotations:
            1. left_left = ""
            2.  left_right = ""
            3.   right_left = ""
            4.   right_right = ""
      xv. If self.root.left exists:
            1. If self.root.left has a left node:
                 a. Add the name to the left_left variable
            2. If self.root.left has a right node:
                 a. Add the name to the left_right variable
       i.   If self.root.right exists:
            3. If self.root.right has a left node:
                 a. Add the name to the right_left variable
```

```
                4.  If self.root.right has a right node:
                        a.  Add the name to the right_right variable
            ii.     If any of these nodes exist left_left, left_right, right_left, right_right
                    a.  print a new line.
                    b.  If left_left or left_right node exist:
                            i.  Print the connectors.
                    c.  Else, print an empty space to keep tree aligned.
                    d.  If right_left or right_right exists:
                            i.  Print the connectors
                    e.  Else:
                            i.  Print a new line
                    f.  Print left_left, left_right, right_left, right_right with the appropriate indentations.
        g.  Define a method get_ancestors(self, node, target, path):
            i.    This method uses recursion to get the ancestors for a specific creature.
            ii.   If the Node doesn't exist:
                    1.  Return false
            iii.  If the node name matches the target name.
                    1.  We found a creature, return True.
            iv.   Call get_ancestor to search the left and right children.
            v.    If an ancestor was found on the left or right:
                    1.  Append that ancestor to the path list and return True.
            vi.   If none of the above conditions were satisfied, return false.
        h.  Define a method print_ancestors(self, name):
            i.    This method calls get_ancestors and formats the specific ancestor lineage into a sentence.
            ii.   Create an empty list to hold found ancestors.
            iii.  Call get_ancestors to find the ancestors of the creature the user entered.
            iv.   If no ancetors was found:
                    1.  Let the user know no ancestors was found in tree
            v.    Otherwise:
                    1.  Construct the text to be outputted and add it to the variable "sentence".
                    2.  Loop through the path list:
                            a.  Get ancestor, dynamically add it to sentence.
                            b.  Add sentence to previous sentences
                    3.  Return sentence
    3.  Define a function display_menu(root_exists):
        a.  Print the menu header
        b.  If no root node exists:
            i.    Print the add root creature menu option.
        c.  Else:
            i.  Display the full menu.
        d.  Define a method main():
            i.  Instantiate a CreatureTree object
            ii.  While loop:
                    1.  Call the display menu method to display the menu.
                    2.  Get user menu selection.
                    3.  If the user selected option 0:
                            a.  Ask the user to enter the parent creature name.
                            b.  Call the tree.add_root method to create the parent
                                creature node.
                    4.  If the user selected option 1:
                            a.  Print header creatures.
                            b.  Display the present tree structure.
                            c.  Ask user to enter the name of the parent node.
                            d.  Ask user to enter L or R side where node will be added.
                            e.  Ask use to enter the name of the child node.
                            f.  Add the creature and print confirmation.
                    5.  If the user selected option 2:
                            a.  Print the header creature tree.
                            b.  Display the present tree structure.
                    6.  If the user selected option 3.
                            a.  Ask the user to enter the name of creature they want to
                                lookup.
                            b.  Call the print_ancestor function to print the lineage as a
                                sentence.
                    7.  If the user selects option 4:
                            a.  Print goodbye and break the while loop to end the
                                program.
    4.  Call the main function to run the program.
```