CIS 211

Spring 2022 Mini-exam 3 KEY

Your name: I.M.Key

Student ID#: 0x38a30732

Please do not write your name on other pages.

Unsolicited advice: Read the whole exam once before you start. Work the easy ones first. Think before you write. Draft and revise code on scratch paper before copying to the exam. If you get stuck on a problem, move on to others, then return to it. Take a deep slow breath.

Total: _____

1. [3 points] Fill in two lines of code in the following program so that it prints "Bing! Boom! Boom! Bing!", one word per line.

```
class Button:
    """Abstract base class for buttons."""
    def press(self):
        """Takes some action that depends on subclass"""
        raise NotImplementedError("This button is broken")

class ControlPanel:
    def __init__(self, buttons: list[Button]):
        self.buttons = buttons

    def press_all(self):
        """Press ALL the buttons!"""
        for button in self.buttons:
            button.press()

class BingButton(Button):
    def press(self):
        print("Bing!")

class BoomButton(Button):
    def press(self):
        print("Boom!")

# Fill in two lines of code here ...
green_button = BingButton()
blue_button = BoomButton()









panel = ControlPanel([ green_button, blue_button, blue_button, green_button])
panel.press_all()


#
# Expected output:
# Bing!
# Boom!
# Boom!
# Bing!
```

(score)

2. [7 points]

Complete the following code so that `t.count_gt(n)` returns the number of leaves in `t` that contain a value greater than `n`. The test cases at the end should all pass.

```
class Tree:
    """Abstract base class for a binary tree holding
    integer values in the leaves.
    """

    def count_gt(self, n: int) -> int:
        """How many leaves in this subtree hold values greater than n?"""
        raise NotImplementedError("Subclass should implement count_gt")



class Leaf(Tree):
    def __init__(self, val: int):
        self.val = val

    def count_gt(self, n: int) -> int:
        if self.val > n:
            return 1
        else:
            return 0




class Inner(Tree):
    def __init__(self, left: Tree, right: Tree):
        self.left = left
        self.right = right

    def count_gt(self, n: int) -> int:
        return self.left.count_gt(n) + self.right.count_gt(n)




# Simple test cases
t = Inner(Inner(Leaf(5), Leaf(3)), Inner(Leaf(2), Inner(Leaf(1), Leaf(6))))
assert t.count_gt(3) == 2
assert t.count_gt(5) == 1
assert t.count_gt(0) == 5
assert Leaf(3).count_gt(3) == 0
assert Leaf(3).count_gt(2) == 1
```

(score)

3. [10 points]

Complete the following function, which takes a rectangular matrix of strings as input argument. It should return True if the matrix contains at least one column in which all the elements of that column are equal. Otherwise it should return False.

Note the examples at the bottom of the page.

```
def some_col_all_same(m: list[list[str]])-> bool:
    """True if m contains at least one column in
    which all the elements are equal
    """
    if len(m) == 0:
        return False
    for col_i in range(len(m[0])):
        sample = m[0][col_i]
        this_col_all_same = True
        for row in m:
            if row[col_i] != sample:
                this_col_all_same = False
        if this_col_all_same:
            return True
    return False
```

```
assert not some_col_all_same([])   # Has no rows or columns
assert not some_col_all_same([[]]) # Has a row, but no columns
assert some_col_all_same([["a"]])
assert some_col_all_same([["a"], ["a"], ["a"]])
assert not some_col_all_same([["a", "b"], ["a", "c"], ["b", "c"]])
assert some_col_all_same([["a", "b", "c"],
                          ["b", "b", "c"],
                          ["b", "b", "a"]])
```

(score)