CIS 211

Winter 2019 Midterm Exam

Your name: _____

Student ID#: _____

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. [5 points] Consider this program:

```
class Room:
    """A room in a building."""
    def __init__(self, room_num: str, capacity: int):
        self.num = room_num
        self.capacity = capacity

    def big_enough(self, required: int) -> bool:
        """Is this room big enough?"""
        return self.capacity >= required

    def __str__(self) -> str:
        return self.num


class Building:
    """A building may include several rooms"""
    def __init__(self, name: str):
        self.name = name
        self.rooms = [ ]

    def add_room(self, room: Room):
        self.rooms.append(room)

    def select(self, capacity: int) -> list:
        """Which rooms have sufficient capacity for an event?"""
        result = []
        for room in self.rooms:
            if room.big_enough(capacity):
                result.append(room)
        return result

###
deschutes = Building("Deschutes Hall")
deschutes.add_room(Room("Rm 100", 25))
deschutes.add_room(Room("Rm 120", 5))
deschutes.add_room(Room("Rm 200", 20))
deschutes.add_room(Room("Rm 201", 12))

for room in deschutes.select(15):
    print(room)
```

What does it print?

(score)

2. [10 points] The program below has a bug due to aliasing.

```
from typing import List

class Ingredient:
    def __init__(self, name: str, amount: float, unit: str):
        self.name = name
        self.amount = amount
        self.unit = unit

    def scale(self, proportion: float):
        self.amount = proportion * self.amount

    def __str__(self) -> str:
        return f"{self.amount} {self.unit} {self.name}"

def pr(recipe: List[Ingredient]):
    for ingredient in recipe:
        print(str(ingredient))

def double(recipe: List[Ingredient]):
    for ingredient in recipe:
        ingredient.scale(2.0)

hot_water = Ingredient("water", 100, "ml")
hot_coffee = [hot_water, Ingredient("coffee", 21, "grams")]
hot_tea = [hot_water, Ingredient("tea", 5, "grams")]
double(hot_coffee)

pr(hot_coffee)
print()
pr(hot_tea)
```

The programmer was hoping it would print

```
200.0 ml water
42.0 grams coffee

100.0 ml water
5 grams tea
```

But it doesn't! What does it actually print?

(score)

3. [10 points] Consider:

```
class Listener:
    def notify(self, msg: str):
        print(f"***> {msg}")

class Box:
    def __init__(self, capacity):
        self.capacity = capacity
        self.listeners = [ ]

    def add_listener(self, listener: Listener):
        self.listeners.append(listener)

    def notify_all(self, msg: str):
        for listener in self.listeners:
            listener.notify(msg)

    def scale(self, sf: int):
        self.capacity = sf * self.capacity
        self.notify_all(f"New capacity {self.capacity}")

boxes = [Box(1), Box(2), Box(3), Box(4)]

for box in boxes:
    box.scale(2)

for box in boxes:
    box.add_listener(Listener())

for box in boxes:
    box.scale(2)
```

What does this program print?

(score)

4. [10 points] Finish function `select_in_range` consistent with its docstring comment examples.

```
from typing import List

def select_in_range(v: List[int], min_val: int, max_val: int):
    """Returns a a new list containing the elements x of v
    that are at least min_val and at most max_val.
    Examples:
        select_in_range([5, 10, -2, 1, 20], 1, 10) == [5, 10, 1]
        select_in_range([5, 10, -2, 1, 20], -10, 0) == [-2]
        select_in_range([], -3, 4) == []
        select_in_range([5, 10, -2, 1, 20], 100, 200) == []
    """
```


(score)

5. [10 points] In this question, we will represent a 4 by 4 matrix of integers as a list of lists of int.

I want a function `every_column_zero` that determines whether each *column* of the matrix has at least one zero element. Complete the function.

```
def every_column_zero(matrix: List[List[int]]) -> bool:
    """matrix is 4 lists of 4 integers (4 rows of 4 columns each).
    Return True iff each column has at least one element with value 0.
    """
```

```
assert every_column_zero([[1, 0, 1, 0], [0, 1, 1, 1], [1, 1, 1, 1], [1, 1, 0, 1]])
assert not every_column_zero([[1, 0, 1, 0], [1, 1, 1, 0], [1, 0, 1, 0], [1, 0, 0, 0]])
```


(score)

6. [10 points] Complete classes Interior and Leaf so that `all_positive` returns True if all the leaves of the tree hold numbers greater than zero and False otherwise.

```
class Tree:
    """Abstract base class for Interior and Leaf"""

    def all_positive(self) -> bool:
        """Should return true if all leaves are positive integers"""
        raise NotImplementedError("All concrete subclasses must implement all_positive")

class Interior(Tree):
    """An interior Tree node."""

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

    def all_positive(self) -> bool:









class Leaf(Tree):
    """A leaf node of the tree"""

    def __init__(self, value: int):
        self.value = value

    def all_positive(self) -> bool:
```


(score)