CIS 211

Spring 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. [10 points] Consider the following program, which uses aliasing intentionally.

```
from typing import List

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

class Matrix:
    def __init__(self, elements: List[List[Tile]]):
        self.tiles = elements
        self.columns = self.group_by_columns()

    def group_by_columns(self):
        cols = [ ]
        for col_i in range(len(self.tiles[0])):
            group = [ ]
            for row_i in range(len(self.tiles)):
                group.append(self.tiles[row_i][col_i])
            cols.append(group)
        return cols

    def magnify_columns(self):
        for col_i in range(len(self.columns)):
            col = self.columns[col_i]
            for tile in col:
                tile.value = tile.value * col_i

    def pr(self):
        for row in self.tiles:
            for tile in row:
                print(f"{tile.value} ", end="")
            print()

m = Matrix([[Tile(1), Tile(1), Tile(1), Tile(1)],
            [Tile(1), Tile(1), Tile(1), Tile(1)],
            [Tile(1), Tile(1), Tile(1), Tile(1)],
            ])

m.magnify_columns()
m.pr()
```

What does it print?

(score)

2. [10 points] Consider this program:

```
class CourseRecord:
    def __init__(self, title: str, group: str, grade: int ):
        self.title = title
        self.group = group
        self.grade = grade

    def __str__(self):
        return f"{self.title} ({self.group}): {self.grade}"

class Selector:
    def select(self, item: CourseRecord) -> bool:
        raise NotImplementedError("Concrete selector needs 'select' method")

class Transcript(list):
    def select(self, selector: Selector):
        result = [ ]
        for item in self:
            if selector.select(item):
                result.append(item)
        return result

class SelectByGroup(Selector):
    def __init__(self, group: str):
        self.group = group

    def select(self, item: CourseRecord) -> bool:
        return item.group == self.group

my_grades = Transcript()
my_grades.append(CourseRecord("Ancient Greek Cinema", "AL", 3))
my_grades.append(CourseRecord("History of Pasta", "SS", 4))
my_grades.append(CourseRecord("Folk Songs of Austria and Australia", "AL", 2))

for  course in my_grades.select(SelectByGroup("AL")):
    print(course)
```

What does it print?

(score)

3. [10 points] Consider this program:

```
class Listener:
    def notify(self, event_name: str):
        raise NotImplementedError("Listener subclass needs notify method")

class Door:
    def __init__(self):
        self.listeners = []

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

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

    def open(self):
        print("Door opening")
        self.notify_all("open")

    def close(self):
        print("Door closing")
        self.notify_all("close")

class Spy(Listener):
    def __init__(self):
        self.open_count = 0

    def notify(self, event_name: str):
        if event_name == "open":
            self.open_count += 1

monitor = Spy()
front_door = Door()
front_door.open()
front_door.close()
front_door.add_listener(monitor)
front_door.open()
front_door.close()
print(f"Opened {monitor.open_count} times")
```

What does it print?

(score)

4. [10 points] Finish method `region_cloud` so that it returns a new PointCloud that contains exactly the points within the given rect, consistent with the example at the bottom of the page.

```
from typing import Tuple, List

class Pt:
    def __init__(self, x: int, y: int):
        self.x = x
        self.y = y

    def __repr__(self) -> str:
        return f"Pt({self.x}, {self.y})"

    def __ge__(self, other: "Pt") -> bool:
        return self.x >= other.x and self.y >= other.y


class Rect:
    def __init__(self, ll: Pt, ur: Pt):
        assert ur >= ll, "ur must be upper right corner"
        self.ll = ll
        self.ur = ur

    def contains(self, pt: Pt) -> bool:
        return pt >= self.ll and self.ur >= pt


class PointCloud(list):

    def region_cloud(self, rect: Rect) -> "PointCloud":
        """Returns PointCloud with points contained in rect"""
```

```
pc = PointCloud()
pc.append(Pt(1,1))
pc.append(Pt(3,3))
pc.append(Pt(4,4))
pc.append(Pt(8,8))
selected = pc.region_cloud(Rect(Pt(2,2), Pt(5,5)))
print(selected)
# Expecting:  [Pt(3,3), Pt(4,4)]
```

(score)

5. [10 points]

We will represent a matrix of integers as a list of lists of int. Each row (inner list of int) will be the same length. I want a function that determines whether a matrix of integers contains at least one column that contains only positive integers. Code it without creating any additional lists.

```
from typing import List

def some_column_all_positive(m: List[List[int]]) -> bool:
    """At least one column is all positive integers."""
```

```
assert some_column_all_positive([[-1, 2, -3]])  # because [2] is all positive
assert not some_column_all_positive([])
assert some_column_all_positive([[2, 3, -2], [-2, 2, 3]]) # because [3, 2]
assert not some_column_all_positive([[2, 3, -2], [-1, -1, 2], [8, 9, 7]])
```

(score)

6. [10 points] Complete the `select_text` methods so that it returns the concatenation of content with matching tags, as in the example at the bottom of this page.

```
from typing import List

class Node:
    """Abstract base class --- tree of text"""
    def select_text(self, tag: str) -> str:
        """Should return concatenation of all text in nodes matching a particular tag"""
        raise NotImplementedError("Each node type should define select_text")

class Block(Node):
    def __init__(self,  content: List[Node]):
        self.content = content

    def select_text(self, tag: str) -> str:
```

```
class Text(Node):
    def __init__(self, tag: str, content: str):
        self.tag = tag
        self.content = content

    def select_text(self, tag: str) -> str:
```

```
doc = Block([ Text("par", "When you just need to "),
              Block([Text("em", "get some sleep and then"),
                     Text("par", "get to work")])])

selected = doc.select_text("par")
print(selected)
# Expected output: "When you just need to get to work"
```

(score)