CIS 211

Winter 2019 Final 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: _____

Back of page is blank on purpose

1. [10 points] Consider the following program.

```
from typing import List

class SummaryHook:
    """Abstract base class"""
    def __init__(self, initial: int = 0):
        self.summary = initial

    def get(self) -> int:
        return self.summary

    def visit(self, cell: int):
        raise NotImplementedError("Oops")


class Summarizable:
    def __init__(self, elements: List[int] = []):
        self.elements = elements

    def foreach(self, hook: SummaryHook) -> int:
        for cell in self.elements:
            hook.visit(cell)
        return hook.get()

class Sm(SummaryHook):
    def visit(self, cell: int):
        self.summary += cell

class Mx(SummaryHook):
    def visit(self, cell: int):
        if cell > self.summary:
            self.summary = cell

s = Summarizable([10, 30, 20])

summary = s.foreach(Sm())
print(f"Summarized by Sm to {summary}")

summary = s.foreach(Mx())
print(f"Summarized by Mx to {summary}")
```

What does it print?

(score)

Back of page is blank on purpose

2. [10 points] Recall that in Python regular expressions, "." is the wild-card that matches any character and "\s" matches a whitespace character. We use \(and \) to treat treat parentheses as ordinary chracters rather than forming groups. Consider the following program:

```
import re
from typing import Pattern

pat = re.compile(r"""
    \s* def \s*
    (?P<f> [a-zA-Z_]+)  \s*
    \(  (?P<a> .* )  \) \s*
    ( -> \s*  (?P<r> .+) )?   \s*
    : \s*
    """, re.VERBOSE)

def check(p: Pattern, s: str):
    """Check how p matches a string"""
    match = pat.fullmatch(s)
    if not match:
        print(f"No match of '{pat.pattern}' on {s}")
    else:
        d = match.groupdict()
        print(f"Matched {d}")

check(pat, "def add(x: int, y: int) -> int:")
check(pat, "def shout():")
```

What does it print?

(score)

Back of page is blank on purpose

3. [10 points] Sometimes we compactly encode a path such as movement of a pen or movement of a monster in a video game as a series of small moves in a grid. In this problem, we will pack one of 8 compass directions into bits 5..7 of a byte and use the remaining bits 0..4 to record a distance of up to 31 units. Finish the three functions so that they correctly pack and extract the information. Recall that you can convert a Direction `d` to an integer as `d.value` and you can convert an integer `i` in the range 0..7 to a Direction as `Direction(i)`. You may assume that values provided to `to_dir` and `to_dist` were produced by `encode`.

```
from enum import Enum

class Direction(Enum):
    North = 0
    NorthEast = 1
    East = 2
    SouthEast = 3
    South = 4
    SouthWest = 5
    West = 6
    NorthWest = 7


def encode(dir: Direction, steps: int) -> int:
    """Return 8-bit encoding of movement"""




def to_dir(word: int) -> Direction:
    """Extract direction value from encoded movement"""




def to_steps(word: int) -> int:
    """Extract number of steps from movement"""



east13 = encode(Direction.East, 13)
assert to_dir(east13) == Direction.East
assert to_steps(east13) == 13
```


(score)

Back of page is blank on purpose

4. [10 points] Professor X gives several quizzes in a term. If a student misses a quiz, a score of 0.0 is recorded. However, if a student misses up to three quizzes, those scores will be replaced with the average of the other quiz scores. If a student misses more than 3 quizzes, *none* of the missing quiz scores will be replaced. Canvas doesn't support this grading method ... help Professor X by writing a Python function to replace the missing scores.

```
from typing import List

def replace_missing_quizzes(quizzes: List[float]):
    ""If no more than 3 quizzes are 0.0, replace them  by the average
    of the remaining scores. Otherwise the list is not modified.
    """
```

```
scores = [15.0, 0.0, 3.0, 12.0]
replace_missing_quizzes(scores)
assert scores == [15.0, 10.0, 3.0, 12.0 ]

scores = [15.0, 0.0, 0.0, 3.0, 12.0, 0.0, 7.0, 0.0]
replace_missing_quizzes(scores)
assert scores == [15.0, 0.0, 0.0, 3.0, 12.0, 0.0, 7.0, 0.0]
```


(score)

Back of page is blank on purpose

5. [10 points] Finish the Inner and Leaf classes below so that `t.sum_in_range(min, max)` returns the sum of the leaves whose values are between *min* and *max* inclusive.

```
class Tree:
    def sum_in_range(self, min_val: int, max_val: int) -> int:
        """Sum of leaf values in range min_val .. max_val"""
        raise NotImplementedError("Hey!  You forgot!")


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

    def sum_in_range(self, min_val: int, max_val: int) -> int:
        """Sum of leaf values in range min_val .. max_val"""









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

    def sum_in_range(self, min_val: int, max_val: int) -> int:
        """Sum of leaf values in range min_val .. max_val"""









t = Inner(Leaf(5), Inner( Inner(Leaf(8), Leaf(3)),  Leaf(6)))

assert t.sum_in_range(4,7) == 11
assert t.sum_in_range(-3, -5) == 0
assert t.sum_in_range(0,10) == 22
```


(score)

Back of page is blank on purpose

6. [10 points] Finish the following function:

```
def some_col_all_pos(m: List[List[int]]) -> bool:
    """True iff some column is all greater than zero.
    m is assumed to be a rectangular matrix (all rows are the same length)
    Examples: some_col_all_pos([]) == False  because it has no columns
              some_col_all_pos([[]]) == False because it has no columns
              some_col_all_pos([[42]]) == True but some_col_all_pos([[0]])) == False
              some_col_all_pos([[1, 0], [2, -3]]) == True because [1,2] is all positive
              some_col_all_pos([[0, 0, 1, 1],[1, 2, 3, 4], [8, 7, 1, 0]])
                    == True because [1, 3, 1] is all positive
    """
```


(score)