CIS 211

Spring 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.

```
class Listenable:
    def __init__(self):
        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(self, msg)

class Tile(Listenable):
    def __init__(self, label: str):
        super().__init__()
        self.label = label

    def relabel(self, label: str):
        self.notify_all(f"New label {label}")
        self.label = label

class Listener:
    def notify(self, tile: Tile, msg: str):
        print(f"Tile {tile.label}: {msg}")

row = [ Tile("alpha"), Tile("beta"), Tile("gamma") ]
row[1].add_listener(Listener())

for tile in row:
    tile.relabel("Modified")
```

What does it print?

(score)

Back of page is blank on purpose

2. [10 points] Recall that in Python regular expressions, we use \(and \) to treat treat parentheses as ordinary chracters rather than forming groups. Consider the following program:

```
import re

pat = re.compile(r"""
 \s* for \s+
 (?P<iter_var> [a-zA-Z]\w*)
 \s+ in \s+
 (  ( range\(len\( (?P<seq_i> [a-zA-Z]\w*) \)\) )
  | (  (?P<seq_e> [a-zA-Z]\w*) )
 ) \s*
 : \s*
""", re.VERBOSE)

def check(s: str):
    match = pat.fullmatch(s)
    if match:
        parts = match.groupdict()
        if parts["seq_i"] is not None:
            print(f"{parts['iter_var']} indexes through {parts['seq_i']}")
        else:
            print(f"{parts['iter_var']} loops directly through {parts['seq_e']}")
    else:
        print(f"for loop pattern not recognized in {s}")

check("for i in li:")
check("for i in range(len(li)):")
check ("x += li[17]")
```

What does it print?

(score)

Back of page is blank on purpose

3. [10 points] Programmers often use hexadecimal or "hex" (base 16) to represent binary values because each hex digit represents exactly four binary digits. But hex is not the only reasonable choice. Some Unix programs have traditionally used octal (base 8), in which each octal digit represents three binary digits. Finish the following functions for converting between strings of octal digits and positive integers. For full credit, do not use integer addition, multiplication, division, or exponentiation.

```
OCTAL_DIGITS = "01234567"    # indexable, e.g., OCTAL_DIGITS[3] == "3"
OCTAL_VAL = { "0": 0, "1": 1, "2": 2, "3": 3, "4": 4, "5": 5, "6": 6, "7": 7 }

def octal_to_int(oct: str) -> int:
    """Convert string of octal digits to positive integer.
    Examples: octal_to_int("377") == 255, octal_to_int("027") == 23
    Assume all characters in 'oct' are octal digits.
    """
```

```
def int_to_octal(w: int) -> str:
    """Concert positive integer w to string of octal digits.
    Examples:   int_to_octal(255) == "377", int_to_octal(23) == "27"
    """
```

```
assert octal_to_int("027") == 23
assert int_to_octal(255) == "377"
```

(score)

Back of page is blank on purpose

4. [10 points] Finish function `only_multiples` consistent with its docstring and the assertions at the bottom of this page.

```
from typing import List

def only_multiples(li: List[int]) -> List[int]:
    """Returns a list consisting of items in li
    that appear more than once, in the same order
    as they appeared in li.
    Example: only_multiples([1, 2, 3, 2, 1, 2, 4]) == [1, 2, 2, 1, 2]
             only_multiples([1, 2, 3, 4, 5]) == [ ]
    """
```

```
assert only_multiples([1, 2, 3, 2, 1, 2, 4]) == [1, 2, 2, 1, 2]
sample = [1, 2, 3, 4, 5]
assert only_multiples(sample) == [ ]
assert sample == [1, 2, 3, 4, 5]
```

(score)

Back of page is blank on purpose

5. [10 points] Finish the `free_of` method of classes `Composite_Food` and `Simple_Food`.

```
from typing import List

class Food:
    def free_of(self, allergens: List[str]):
        """True if it does NOT contain any of the allergens"""
        raise NotImplementedError

class Composite_Food(Food):
    def __init__(self, ingredients: List[Food]):
        self.ingredients = ingredients

    def free_of(self, allergens: List[str]) -> bool:
```

```
class Simple_Food(Food):
    def __init__(self, name: str):
        self.name = name

    def free_of(self, allergens: List[str]) -> bool:
```

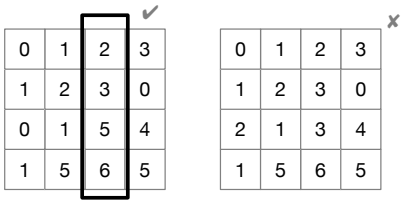
```
# Example
crust = Composite_Food([Simple_Food("wheat"), Simple_Food("yeast"), Simple_Food("water")])
sauce = Composite_Food([Simple_Food("tomato"), Simple_Food("garlic"), Simple_Food("onion")])
topping = Simple_Food("cheese")
pizza = Composite_Food([crust, sauce, topping])

assert not pizza.free_of(["wheat", "barley"])
assert pizza.free_of(["peanuts", "cashews", "walnuts"])
```

(score)

Back of page is blank on purpose

6. [10 points]

Finish function `some_column_increasing`. For full credit, do not construct additional lists.




```
from typing import List

def some_col_increasing(m: List[List[int]]) -> bool:
    """True if at least one column of the matrix is strictly increasing,
    e.g., [[1, 3], [0, 4], [8, 5]] has column 1 (3,4,5) increasing
    but [[1, 3], [0,4], [8, 4]] doesn't; (3,4,4) is not *strictly* increasing.
    """
```

```
assert some_col_increasing([[0, 1, 2, 3], [1, 2, 3, 0], [0, 1, 5, 4],[1, 5, 6, 5]])
assert not some_col_increasing([[0, 1, 2, 3],[1, 2, 3, 0],[2, 1, 3, 4],[1, 5, 6, 5]])
assert not some_col_increasing([])  # Vacuous case: No rows or columns, so none increasing
assert not some_col_increasing([[]]) # Vacuous case: No columns
```

(score)