MTH 3300 STRA Final Exam

May 22, 2025

Name	

	Points	**Total Points**
Problem 1		10
Problem 2		10
Problem 3		15
Problem 4		15
Problem 5		10
Problem 6		10
Problem 7		10
Problem 8		20
Total		100

Problem 1

Given two strings `ransom_note` and `magazine`, return `True` if `ransom_note` can be constructed using the letters from `magazine` and `False` otherwise.

Example:

```
can_construct_ransom_note("aa", "aba") outputs True
can_construct_ransom_note("aa", "ab") outputs False
```

NOTE: Don't assume that the examples above are the only test cases your function will be tested on.

```python
def can_construct_ransom_note(ransom_note: str, magazine: str) -> bool:
    # Write your answer below
```

Problem 2

Remember that a factorial is defined as $n! = n \times n - 1 \times ... \times 1$.

Write a function that finds the sum of the digits of a factorial. For example, $4! = 4 \times 3 \times 2 \times 1 = 24$, so `sum_factorial_digits(4)` would produce 6 since $4! = 24$ and $2 + 4$ would give us 6.

```
def sum_factorial_digits(n: int) -> int:
    # Write your answer below
```

Problem 3

Write a function `generate_pascal(n: int)` that generates the first n rows of Pascal's Triangle. Each number in the triangle is the sum of the two numbers directly above it.

Example of Pascal's Triangle:

```
Row 0:     1
Row 1:    1 1
Row 2:   1 2 1
Row 3:  1 3 3 1
Row 4: 1 4 6 4 1
```

Your function should return a list of lists, where each inner list represents a row.

For example, `generate_pascal(3)` should return:

```
[
    [1],
    [1, 1],
    [1, 2, 1]
]
```

Note: This triangle has many interesting mathematical properties:

- Each row starts and ends with 1
- Each number is the sum of the two numbers above it
- Row n contains the coefficients of the expansion of $(x + y)^n$

```python
def generate_pascal(n: int) -> list[list[int]]:
    # Write your solution here
```

Problem 4

Each question in this section is worth 3 points.

i) Consider the following code:

```
class A:
    def __init__(self):
        self._x = 1
        self.__y = 2

    def get_y(self):
        return self.__y

class B(A):
    def __init__(self):
        super().__init__()
        self._x = 3
        self.__y = 4

b = B()
print(b._x, b.get_y())
```

What is the output?

a) 3 2
b) 3 4
c) 1 2
d) 1 4

ii) Consider this code:

```
class A:
    def __init__(self):
        self.value = 0

    def __add__(self, other):
        return self.value + other

    def __sub__(self, other):
        return other - self.value

a = A()
print(a + 5)
print(10 - a)
```

What is the output?

a) 5 10
b) 5 TypeError
c) TypeError 10
d) TypeError TypeError

iii) Consider this code:

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    def __str__(self):
        return f"({self.x}, {self.y})"


v1 = Vector(1, 2)
v2 = Vector(3, 4)
v3 = v1 + v2
print(v3)
```

What is the output?

a) (4, 6)
b) (1, 2)
c) (3, 4)
d) (1, 3)

iv) Given:

```
class A:
    @classmethod
    def method(cls, x):
        return x

class B(A):
    @staticmethod
    def method(x):
        return x + 1

b = B()
print(b.method(1))
print(B.method(1))
```

What is the output?

a) 2 2
b) 1 2
c) 2 1
d) 1 1

v) Given the following code:

```
class Base:
    multiplier = 2

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

    @classmethod
    def update_multiplier(cls, new_value):
        cls.multiplier = new_value

    def calculate(self):
        return self.value * self.multiplier

class Derived(Base):
    multiplier = 3

    def calculate(self):
        return super().calculate() + 5

base_obj = Base(10)
derived_obj = Derived(10)

Base.update_multiplier(4)
print(f"Result 1: {base_obj.calculate()}")
print(f"Result 2: {derived_obj.calculate()}")

Derived.update_multiplier(6)
print(f"Result 3: {base_obj.calculate()}")
print(f"Result 4: {derived_obj.calculate()}")
```

What is the output?

a) Result 1: 40 Result 2: 45 Result 3: 40 Result 4: 65
b) Result 1: 40 Result 2: 35 Result 3: 40 Result 4: 65
c) Result 1: 40 Result 2: 45 Result 3: 24 Result 4: 65
d) Result 1: 20 Result 2: 35 Result 3: 40 Result 4: 65

Problem 5

The greatest common divisor of two or more integers which are not all zero is the largest positive integer that divides each of the integers. For example, the GCD of 8 and 12 is 4.

(a) (5 points) Write a **recursive** function `gcd(x, y)` that computes the GCD of two integers x and y using the Euclidean algorithm. The function should return the GCD of the two numbers. Note that we can use the following piecewise function to represent the GCD.

$$f(x, y) := \begin{cases} x \text{ if } y = 0 \\ f(y, x \bmod y) \text{ otherwise} \end{cases}$$

(b) (5 points) Implement the same method above, but **iteratively**, i.e. use a loop

Problem 6

Suppose that you have the following data in a CSV file:

```
name,age,species,health_status
Max,3,Dog,Healthy
Luna,2,Cat,Needs Medication
Charlie,5,Dog,Recovering
Bella,1,Cat,Healthy
Rocky,4,Dog,Needs Medication
Milo,2,Cat,Healthy
Shadow,6,Dog,Recovering
Lucy,3,Cat,Healthy
Cooper,2,Dog,Healthy
Lily,4,Cat,Needs Medication
```

Given the following class definition:

```python
class Animal:
    name: str
    age: int
    species: str
    health_status: str

    def __init__(self, name: str, age: int, species: str, health_status: str):
        self.name = name
        self.age = age
        self.species = species
        self.health_status = health_status
```

Add a `classmethod` to the above class called `from_csv(cls, filepath: str) -> list[Animal]` that reads the CSV data from `filepath` and creates an `Animal` instance for each row

Problem 7

Given a list of `int` numbers `nums`, determine if the sequence of numbers is a geometric progression. By definition, a geometric progression is a sequence of numbers that have the form:

$$a, ar, ar^2, ar^3, ar^4 ...$$

where a is the initial value and r is the common ratio.

For example, $2, 6, 18, 54, ...$ is a geometric progression with a common ratio of 3.

```
def is_geometric_progression(nums: list[int]) -> bool:
    # write your work here
```

Problem 8

You are tasked with implementing a basic inventory system for an RPG game. The system should handle items, stacks of items, and inventory management.

Part 1: Item Class (3 points)

Create a class `Item` with:

- Attributes: `name` (str), `weight` (int), `value` (int), `rarity` (str)

Part 2: Stack Class (7 points)

Create a class `Stack` that represents multiple items of the same type:

- Attributes:
  - `item` (`Item` instance)
  - `quantity` (int, cannot be negative)
- Properties:
  - `total_weight`
  - `total_value`
- Methods:
  - `add(self, n: int)`: increases quantity by n
  - `remove(self, n: int)`: decreases quantity by n (minimum quantity is 0)

Part 3: Inventory Class (10 points)

Create a class `Inventory` that manages collections of Stacks:

- Attributes:
  - `maximum_weight` (int)
  - `items` (list of `Stack` instances, internal use only)
- Property:
  - `total_value`: returns sum of all `stack` values
- Methods:
  - `add_item(self, item: Item, quantity: int = 1)`:
    - If item exists, increase its stack quantity
    - If item is new, create new stack
  - `get_items_by_rarity(self, rarity: str) -> list[Stack]`:
    - Returns list of all stacks with matching rarity
  - `transfer_to(other: 'Inventory', item_name: str, quantity: int)`:
    - Moves specified quantity of named item to other inventory

Example usage:

```
>>> sword = Item("Steel Sword", weight=5, value=100, rarity="Common")
>>> inv = Inventory(maximum_weight=50)
>>> inv.add_item(sword, 2)
>>> print(inv.total_value)  # Should print: 200
>>> rare_items = inv.get_items_by_rarity("Rare")  # Should return empty list
```