Full Name: _____

Email Address (UW Net ID): _____@uw.edu

Section: _____

CSE 160 Autumn 2024 - Final Exam

Instructions:

- You have **110 minutes** to complete this exam.
- The exam is **closed book**, including no calculators, computers, phones, watches or other electronics.
- You are allowed a single sheet of notes for yourself.
- We also provide a syntax reference sheet.
- Turn in *all sheets* of this exam, together and in the same order when you are finished.
- Be sure to read the entire question prompt before answering.
- When time has been called, you must put down your pencil and stop writing.
    - **Points will be deducted if you are still writing after time has been called.**
- You may only use parts and features of Python that have been covered in class up to this point.
- You may ask questions by raising your hand, and a TA will come over to you.

Good luck!

Question	**Topic**
Question 1	Errors
Question 2	Files and CSV
Question 3	Data Structures
Question 4	Functions
Question 5	Graphs and Testing
Question 6	Classes

Question 1) For each of the following snippets of code, draw a line from the snippet to the TypeError message that it would result in. An example of code that would cause each specific error is also shown for your information. You do not have to match the specific line of code.

```
data = [
  {"test_score": 89, "study_strategy": "flash cards"},
  {"test_score": 70, "study_strategy": "flash cards"},
  {"test_score": 85, "study_strategy": "active recall"}
]
headers = set()
for row in data:
    for key, val in row.items():
        if 0 >= val >= 100:
            headers.add(key)
        else:
             headers.add(val)
```

Matches with 3

```
animals = ["husky", "duck", "cougar"]
first_letters = {}
for word in range(len(animals)):
    letter = word[0]
    if letter not in first_letters:
        first_letters[letter] = 0
    first_letters[letter] += 1
```

Matches with 1

```
race_finishes = [8, 9, 11]
teams = ["A", "B", "C"]
leaderboard = {}
for time in race_finishes:
    for team in time:
        leaderboard[team] = time
```

Matches with 4

```
race_finishes = [8.1, 9.5, 10.0]
teams = ["A", "B", "C"]
leaderboard = {}
for time in race_finishes:
    for team in teams:
        leaderboard[team] = race_finishes[time]
```

Matches with 2

1

TypeError: 'int' object is not subscriptable

(not subscriptable means not able to be indexed or accessed with brackets)

Example: `1[1]`

2

TypeError: list indices must be integers or slices, not float

Example: `[1, 2][1.5]`

3

TypeError: '>=' not supported between instances of 'int' and 'str'

Example: `"2" >= 1`

4

TypeError: 'int' object is not iterable

(not iterable means unable to be looped over)

Example: `for i in 3:`

Question 2) For this problem, you will write a function called `parse_grocery_list` that takes a single string argument that is the path to a CSV file and returns a dictionary. An example of the contents of the file is on the left, and an example of the returned dictionary is on the right:

```
code,category,name,amount,price
1234,fruit,apple,5,2
4321,fruit,banana,3,1
5678,vegetable,carrot,6,1
8765,fruit,orange,1,2
1357,vegetable,broccoli,3,1
```

```
{
  1234: {
    "category": "fruit",
    "name": "apple",
    "amount": 5,
    "price": 2
  },
  ...
}
```

Each line represents a single grocery store item. The `code` value is expected to be a unique integer (no two items will ever have the same code). And the others (`category`, `name`, `amount`, and `price`) are non-unique.

Notes and hints:

- `csv.DictReader` takes a file (not a file path!) and returns a list of dictionaries.
- `code`, `amount`, and `price` should be converted to integers.

```
import csv
def parse_grocery_list(filepath):
    # Write your solution to question 2 here!

    d = {}
    with open(filepath) as f:
        reader = csv.DictReader(f)
        for row in reader:
            d[int(row["code"])] = {
                "name": row["name"],
                "price": int(row["price"]),
                "amount": int(row["amount"]),
                "category": row["category"]
            }
    return d
```

Question 3) Given the following descriptions of data, determine a data structure that best represents it from the choices given, and write down the Python representation of the data.

Example 1: given "A groceries list for someone who wants to buy apples, bananas, and eggs," you would choose a list and write `["apples", "bananas", "eggs"]`

A. A 2D pixel grid that looks like the following:

2	1
1	0

Circle one: list of lists list of dictionaries

dictionary with lists as values dictionary of dictionaries

Write the Python data representation: [[2, 1], [1, 0]]

B. A survey on favorite pizza topping combinations: 2 people liked pineapples and olives, 1 person liked pineapples, 1 person liked olives, and no one liked either.

	`likes_olives`	`dislikes_olives`
`likes_pineapple`	2	1
`dislikes_pineapple`	1	0

Circle one: list of lists list of dictionaries

dictionary with lists as values dictionary of dictionaries

Write the Python data representation: {“likes_pineapple”: {“likes_olives”: 2, “dislikes_olives”: 1}
“dislikes_pineapple”: {“likes_olives”: 2, “dislikes_olives”: 1}}

C. Two people were interviewed on how often they brush or floss their teeth daily. It does not matter who the people are.

brush	floss
2	1
1	0

Circle one: list of lists list of dictionaries

dictionary with lists as values dictionary of dictionaries

Write the Python data representation: Two possible answers:
[{“Brush”: 2, “Floss”: 1}, {“Brush”: 1, “Floss”: 0}]

OR {“Brush”: [2, 1], “Floss”: [1, 0]}

Question 4) Consider these three functions and then for each of the function calls below, write what the function would return

```
def mystery(a, b):
    n = 0
    for i in range(-1 * b):
        n += a
    return n


def another(m, n):
    for i in range(len(m) + 1):
        m.append(n)
```

```
def final(j, k):
    n = mystery(j, k)
    if n > 0:
        m = []
        another(m, k)
        return m
    else:
        return k - n
```

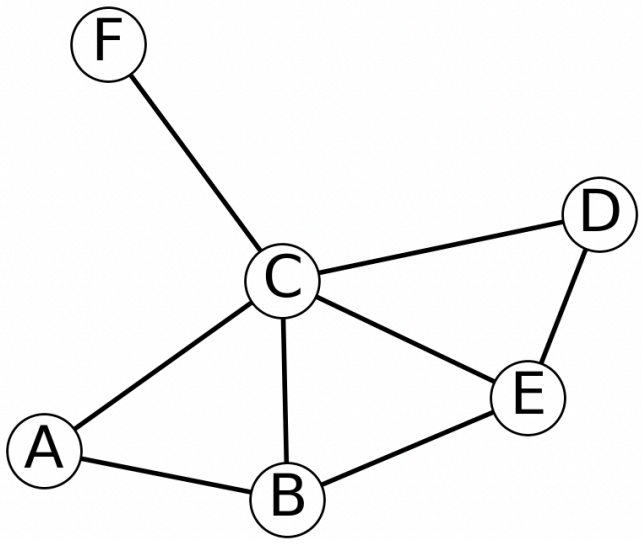
Function Call	**Return**
`final(-2, -3)`	**2**
`final(-3, 2)`	**3**
`final(3, -1)`	**[-1]**

Question 5) Think back to Homework 5, where you were asked to write code to find various combinations of friends in a graph. You're now asked to write some tests for those functions!

Consider the graph on the right.

For each of the tests below, fill in the missing input or expected result, as appropriate.

As reference, the function docstrings are included as a separate page.



Function	Input	Expected Result
`friends`	**`'B'`**	`{'A', 'E', 'C'}`
`friends_of_friends`	`'A'`	**`{'D', 'F', 'E'}`**
`num_common_friends_map`	**`'B'`**	`{'D': 2, 'F': 1}`

Question 6) Read the class written on the following two pages. It is missing some necessary code, indicated by underscores (_____). Using the code already written, the docstrings, and the print output (see separate page), fill in the blank lines of code to finish the class. Hint: read through the entire class before starting to fill in the missing code!

An example usage of this class, with the matching output, is given as a separate page.

```
class FlyLab:
    """
    Grow your own flies! The FlyLab class starts a new home for your
    flies which you can multiply by feeding them food.
    """
    def __init__(self, flies):
        """
        Initializes the fields for the FlyLab class. The number of
        flies to start out must be specified whenever an instance is
        created.
        """
        self.flies = flies
        self.budget = 0
        self.food = []
        self.prices = {"compost": 1, "syrup": 2, "fruit": 5}

    def add_budget(self, amount):
        """Adds money to the budget and prints out a message."""
        self.budget += amount
        print("You added", amount, "dollars to your fly lab! You now
              have", self.budget, "dollars")
```

```
def buy_food(self, item):
    """
    Adds a food to the FlyLab while subtracting from the budget,
    if the food is able to be bought. If the food does not exist
    in the prices dictionary or there is not enough money in
    the budget, prints out an appropriate message.
    """
    if item not in self.prices:
        print("Your requested item does not exist")
    elif self.budget < self.prices[item]:
        print("You don't have enough money to buy this")
    else:
        self.food.append(item)
        self.budget -= self.prices[item]
        print("You added one", item, "to your fly lab! Your
              current food inventory is:", self.food)

def feed_flies(self):
    """
    Increases the number of flies by a factor of two while
    subtracting the first food added. If there is no food, prints
    out a message
    """
    if len(self.food) > 0:
        eaten_food = self.food.pop(0)
        self.flies *= 2
        print("You fed your flies", eaten_food)
        print("Your flies multiplied and you now have",
              self.flies, "flies!")
    else:
        print("You don't have enough food.")
```

Extra Credit (1pt):

Now that you know programming, write a poem or limerick about how your life will be forever changed. 🙂

Function documentation strings for Question 5:

friends(user):

```
    """Returns a set of the friends of the given user in the given graph."""
```

friends_of_friends(user):

```
    """
    Find and return the friends of friends of the given user.

    Arguments:
        user: a unique identifier for a node (user) in the graph

    Returns: a set containing the names of all of the friends of
    friends of the user. The set should not contain the user itself
    or their immediate friends.
    """
```

num_common_friends_map(user):

```
    """
    Returns a map (a dictionary), mapping a person to the number of friends
    that person has in common with the given user. The map keys are the
    people who have at least one friend in common with the given user,
    and are neither the given user nor one of the given user's friends.

    Example graph:
        - "X" and "Y" have two friends in common
        - "X" and "Z" have one friend in common
        - "X" and "W" have one friend in common
        - "X" and "V" have no friends in common
        - "X" is friends with "W" (but not with "Y" or "Z")

    Here is what should be returned:
      number_of_common_friends_map("X") => { 'Y':2, 'Z':1 }

    Arguments:
        user: a unique identifier for a node (user) in the graph

    Returns: a dictionary mapping each person to the number of (non-zero)
    friends they have in common with the user
    """
```

Prompts and example output for Question 6:

An example usage of the class is listed below. Each line of code (indicated by >>>) was run one line at a time and the output (if there was any) was printed immediately below it.

```
>>> my_flies = FlyLab(100)

>>> my_flies.add_budget(7)
You added 7 dollars to your fly lab! You now have 7 dollars

>>> my_flies.feed_flies()
You don't have enough food

>>> my_flies.buy_food("fruit")
You added one fruit to your fly lab! Your current food inventory is: ['fruit']

>>> my_flies.buy_food("caviar")
Your requested item doesn’t exist

>>> my_flies.buy_food("syrup")
You added one syrup to your fly lab! Your current food inventory is: ['fruit', 'syrup']

>>> my_flies.buy_food("compost")
You don’t have enough money to buy this

>>> my_flies.feed_flies()
You fed your flies fruit
Your flies multiplied and you now have 200 flies!

>>> my_flies.feed_flies()
You fed your flies syrup
Your flies multiplied and you now have 400 flies!
```