Full Name: [Edited for CSE 160 26wi]

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

Section:

CSE 160 Autumn 2025 - Final Exam

Instructions:
- You have **the exam period (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.
- 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.
- You will only be graded on what is written in the answer boxes unless otherwise made clear that something outside of the box is also part of your answer.

Good luck!

Question	Topic	Points
Question 1	Expressions	10
Question 2	Code Tracing	5
Question 3	Nested Structures	10
Question 4	File I/O	8
Question 5	Classes: Tracing	7
Question 6	Classes: Writing	10

1 (10 pts). Given the table below, fill in the correct values and type for the matching expression. In cells with multiple lines of code, we ask that you evaluate the last line of code after the lines above are run. In other words, what will be outputted if this code is run in the Python interpreter. If there is an error, write "Error" in the value column. (You may leave the type column blank and you do *not* have to explain the error.)

Expression	Value	Type

`str1 = 'racecar'`
`str2 = 'tacocat'`
`str2[1:3] + str1[4:]`		

`x = (1,2)`
`x[0] = 2`
`x`		

`x = [1, 2, 3, 2, 1]`
`y = x.append(5)`
`y[3]`		

`list1 = ['x', 'y', 'z']`
`list2 = [9, 8, 7]`
`list2.append(list1)`
`len(list2) - len(list1)`		

2 (5 pts) Consider the following code:

```python
def mystery(a, b):
    lst = []
    for i in range(0, len(a), 2):
        val = ""
        for j in range(len(b[i])):
            val += "*" * j
            lst.insert(0, val)
    return sorted(lst)
```

What will the following function calls return? Write `"Error"` if the function call will produce an error. You do not have to specify why the error occurs.

Function Call	Output/Returned Value
`mystery("CSE160")`	
`mystery(["hello", "world", "this"], ["is", "a", "..."])`	
`mystery([["bibbity", "bobbity", "boo"]], ["!!"])`	
`mystery([1, 1, 1], [2, 1, 3])`	
`mystery("Python", "CSE 160")`	

3 (10 pts). Suppose you are given the following dictionary that represents different baseball players and the positions they play. The key is the name of a baseball player (`string`) and the value associated with each key is a list of positions (`list` of `strings`). An example of a dictionary is shown below.

```
roster = {"Cal Raleigh": ["C"],
          "Leo Rivas": ["2B", "3B", "SS"],
          "Eugenio Suarez": ["3B", "SS"]}
```

Write a function called `roster_analysis` that takes in a `roster` dictionary and will **return** a tuple with two elements: the first element being a collection of all the positions covered by your roster and the second element a collection of all positions NOT covered by your roster.

These collections can be any data structure. Assume that some positions can appear more than once across the dictionary (e.g. `3B` and `SS`). Even if a position appears more than once, your result should only show that position once (i.e. no duplicates). You can assume only non-empty dictionaries will be passed in. An example of a function call is shown below.

```
roster_analysis(roster)
```
→ (["C", "2B", "3B", "SS"], ["SP", "1B", "LF", "RF", "CF", "DH"])

Notice that although `"3B"` and `"SS"` appeared more than once in our `roster` dictionary across different players, each position only appears one time in the resulting tuple. You have been provided a list of all required team positions `(p)` for those who are unfamiliar with baseball.

```python
# Write your code here

def roster_analysis(roster):

    p = ["SP", "C", "1B", "2B", "3B", "SS", "LF", "RF", "CF", "DH"]
```

4 (8 points) Suppose you are given a file `songs.csv`, as shown below. Each data row has an artist, song, and the year of release for that song.

```
Olivia Rodrigo,Vampire,2023
Olivia Rodrigo,Deja Vu,2021
ABBA,Dancing Queen,1976
ABBA,Mamma Mia,1975
ABBA,SOS,1975
Carly Rae Jepsen,Call Me Maybe,2012
Taylor Swift,Red,2012
```

Write a function called `find_songs` that takes in two parameters: `filename` and an integer `min_year`. The function should read the csv file `filename` and return a dictionary where the keys are the artists and the values are lists of songs released on or after `min_year`.

An example of what your function should generate is shown below:

```
find_songs('songs.csv', 2012) → {'Olivia Rodrigo': ['Vampire', 'Deja Vu'],
                                  'Carly Rae Jepsen': ['Call Me Maybe'],
                                  'Taylor Swift': ['Red']}
```

Continued on the next page…

```
# Write your code here
```

5 (7 pts). Consider the following MovieTheater class

```python
class MovieTheater():
    """
    This class creates a movie theater object.
    """

    def __init__(self, name, num):
        """
        Initialized object with a given name, number of theaters, and creates
        empty movie showings dictionary.
        """
        self.name = name
        self.theaters = num
        self.showings = {}
        for i in range(1, num + 1):
            self.showings[i] = []

    def add_movie(self, movie, time, num):
        """
        Adds the given movie and showing time as a tuple to the given
        theater.
        """
        self.showings[num].append((movie, time))

    def display_movies(self):
        """
        Prints a list of all movies showing at the movie theater.
        """
        movies = []
        for movie in self.showings.values():
            if movie[0] not in movies:
                movies.append(movie[0])
        print(movies)

    def buy_ticket(self, movie):
        """
        Given a movie title, the function prints the theater and time of the
        movie showing.
        """
        for theater in self.showings.keys():
            if len(self.showings[theater]) > 0:
                for showing in self.showings[theater]:
                    if showing[0] == movie:
                        return("Ticket bought for " + movie +
                               " at " + showing[1] + " in Theater "
                               + str(theater))
        return ("No showings for " + movie)
```

What is the output of the following code?

```
varsity_uw = MovieTheater("Varsity", 4)

movies = [("Wicked: For Good", "8:45 PM", 3), ("Elf", "4:30 PM", 2), ("Princess
Bride", "7:15 PM", 1)]

for movie in movies:
     varsity_uw.add_movie(movie[0], movie[1], movie[2])

print(varsity_uw.buy_ticket("Wicked: For Good"))

varsity_uw.add_movie("Wicked: For Good", "9:00 PM", 4)

varsity_uw.display_movies()
```

```
# Write your solution here
```

6 (10 pts). Write a class that represents a Stadium. An outline for the class along with descriptors of each attribute and method are provided below. Fill in the attributes and methods so that the class works as intended. There is some example code at the end of this problem that demonstrates how the class should work.

```
class Stadium:
    """
    This Stadium class will keep track of important features
    of a stadium including name, location, capacity,
    list of vendors, and events (as a dictionary). The class
    requires only a name, location, and capacity to create the object.
    Everything else should be initialized to 0 or be empty.
    """

    def __init__(self, name, location, capacity):
```
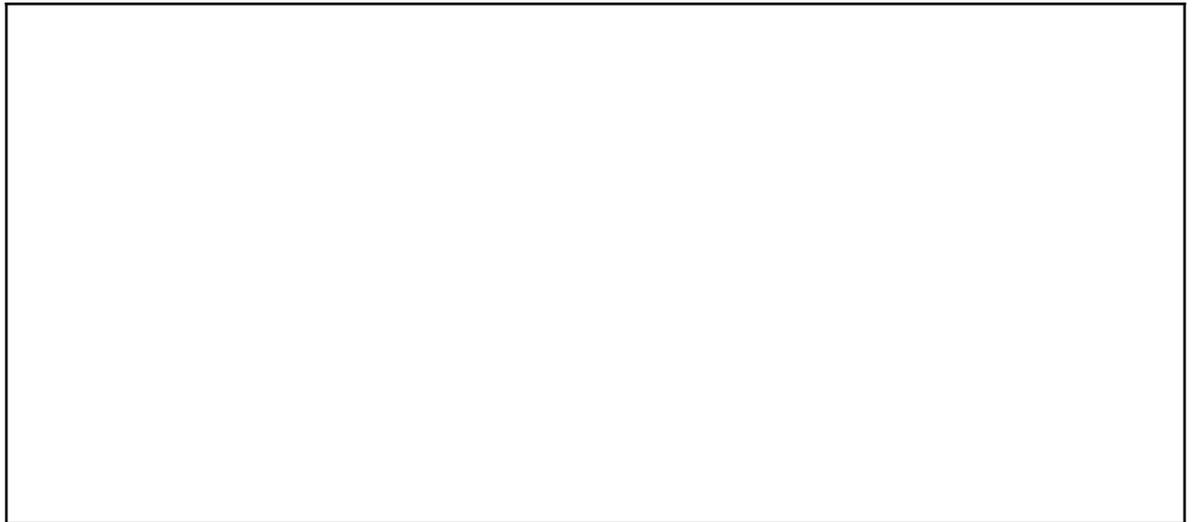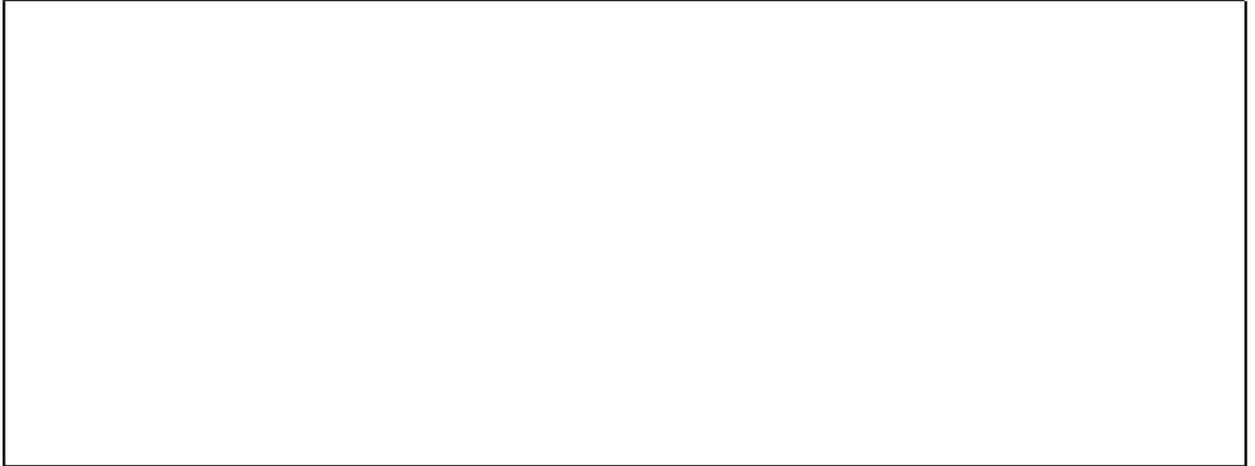
```
    def expand(self, size):
        """
        Expands the stadium capacity by the number specified by
        size. You can assume this value will always be greater
        than 0.
        """
```

```python
def add_vendor(self, vendor_name):
    """
    Adds the vendor name to the list of vendors.
    If the vendor already exists in the list,
    return the message: "Already selling here!"
    """
```

```python
def add_event(self, event, expected_capacity):
    """
    Adds the event to the events dictionary where the key
    is the name of the event and the value is the expected capacity.
    Return the message: "Find another venue!" if the expected
    capacity exceeds the stadium capacity.
    """
```
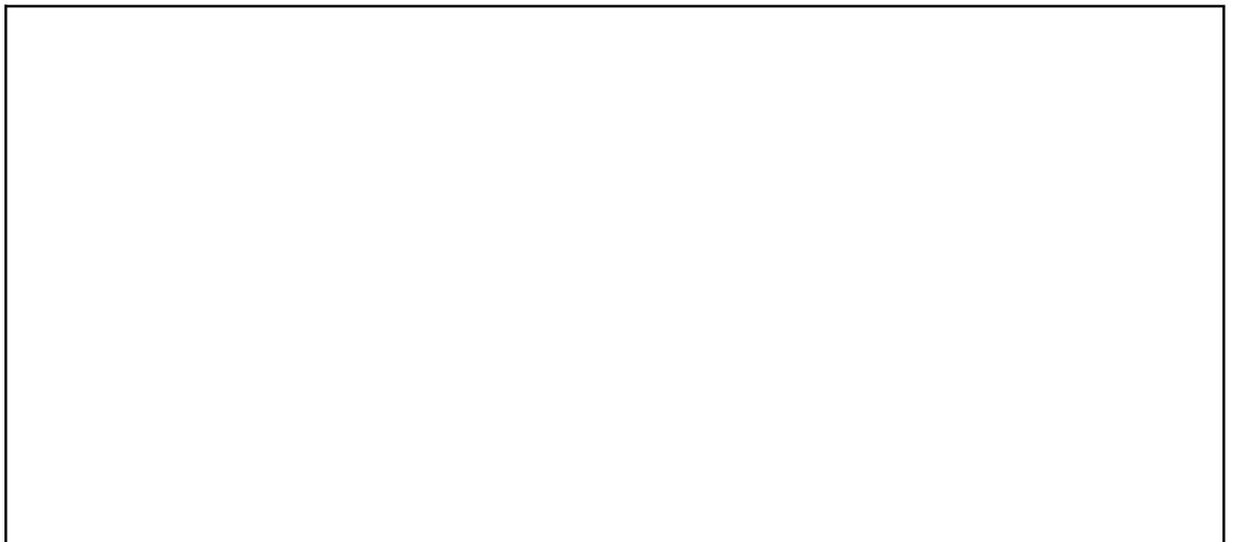
```python
def find_largest_event(self):
    """
    Returns the name of the event with the largest expected
    attendance. If no events are scheduled,
    return the message: "No events scheduled"
    """
```
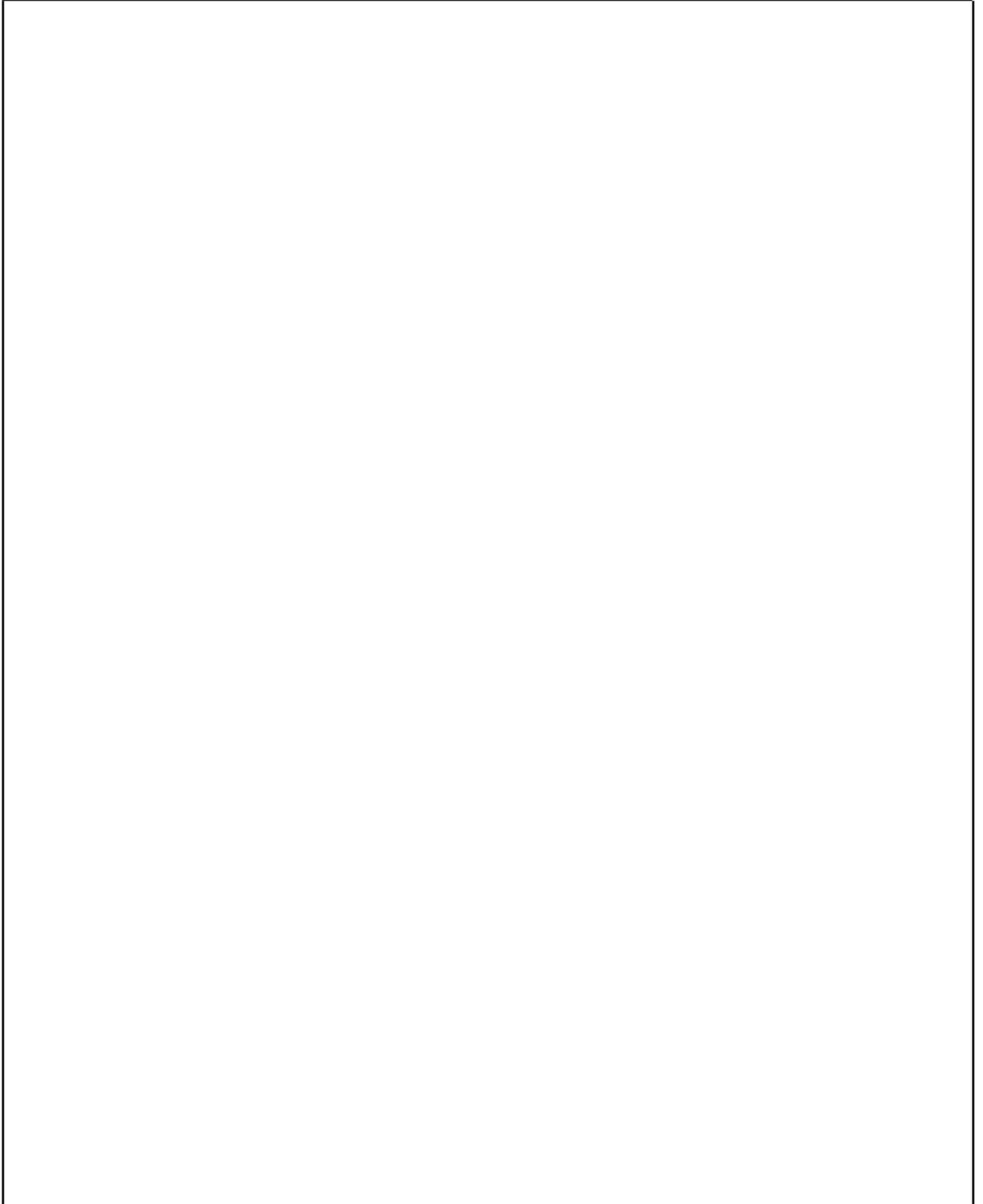
```python
def summary(self):
    """
    Prints information related to the Stadium object.
    An example is as shown below:

    Stadium: T-Mobile Park
    Location: Seattle
    Vendors: ["Moto Pizza", "Salt & Straw"]
    # Events Scheduled: 2
    Events Scheduled: {"Mariners Game": 30000, "Metallica": 25000}
    """
```
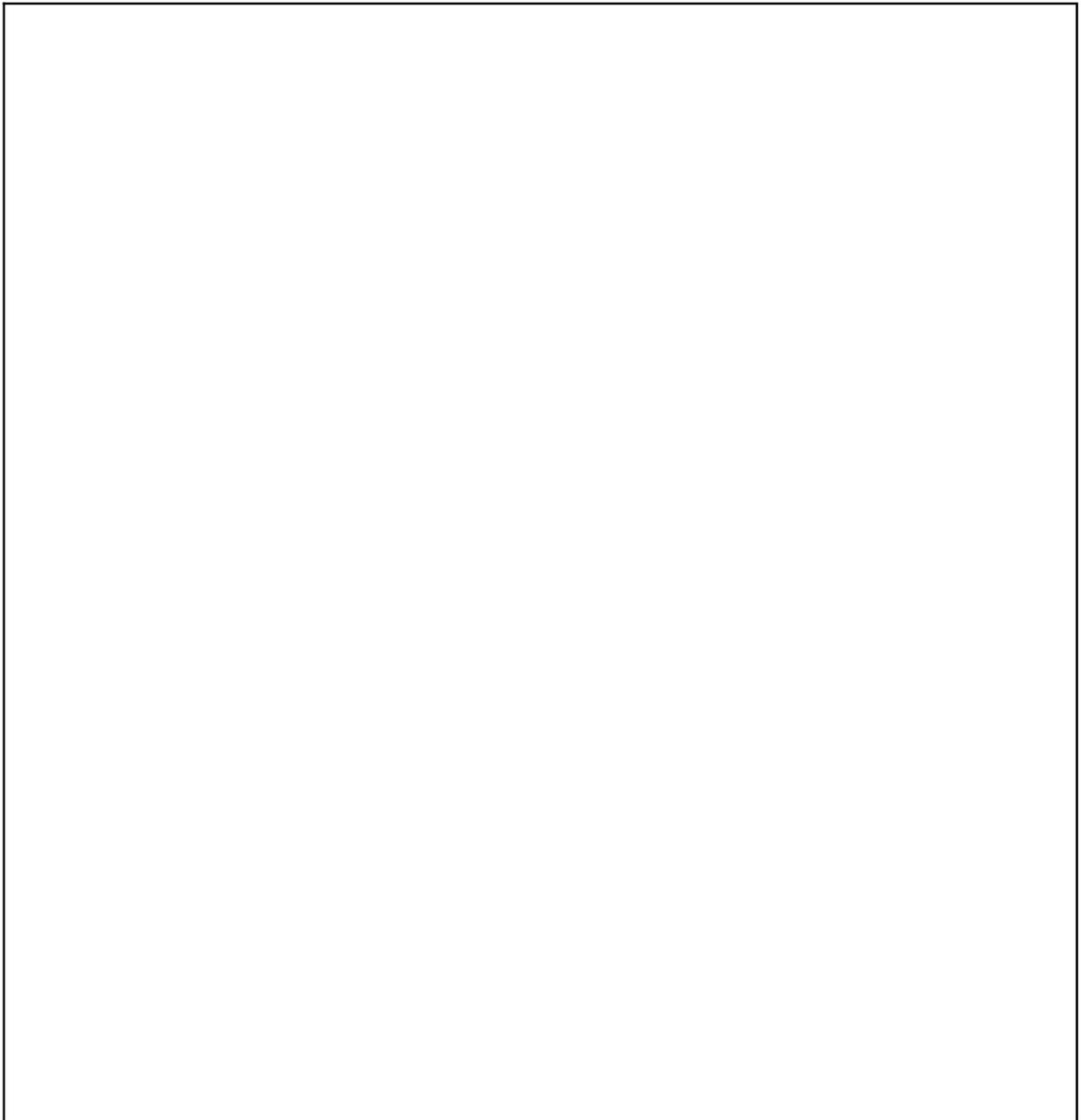
Prompts and example output for Question 6

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) is printed immediately below.

```
>>> tmobile = Stadium("T-Mobile Park", "Seattle", 48000)


>>> tmobile.add_event("Savannah Bananas Game", 1000000)
"Find another venue!"


>>> tmobile.expand(5000)


>>> tmobile.add_event("Mariners Game", 50000)


>>> tmobile.add_event("Metallica", 30000)


>>> tmobile.add_vendor("Moto Pizza")


>>> tmobile.add_vendor("Moto Pizza")
"Already selling here!"


>>> tmobile.find_largest_event()
"Mariners Game"


>>> tmobile.add_vendor("Salt & Straw")


>>> tmobile.summary()
Stadium: T-Mobile Park
Location: Seattle
Vendors: ["Moto Pizza", "Salt & Straw"]
# Events Scheduled: 2
Events Scheduled: {"Mariners Game", 50000, "Metallica": 30000}
```

Extra Credit (1 point): Name or draw your favorite character from any media (fiction or non fiction). Please keep it appropriate.

Good luck with any remaining finals and have a good Winter Break! ⛄ ❄️

CSE 160 25au Final Exam Cheat Sheet

```
# if/elif/else syntax
if condition1:
    # statements
elif condition2:
    # other statements
else:
    # more statements
```

```
# for loop syntax
for i in sequence:
    # statements

# function definition syntax
def function_name(param1, param2, …):
    # statements
```

Function	Description
range([start,] stop [, step])	Returns a sequence of numbers from *start* (inclusive) to *stop* (exclusive) incremented by *step*
len(lst) len(dict)	Returns the number of elements in *lst/dict*

Lists

Function	Description
lst = []	Creates an empty list
lst[idx]	Returns the element in *lst* at index *idx*
lst[start : end]	Returns a sublist of *lst* from index *start* to index *end* (exclusive)
lst[start : end : step]	Returns a sublist of *lst* from index *start* (default 0) to index *end* (exclusive, default *len(lst)*), incrementing by *step*
lst.append(elmt)	Adds the element *elmt* to the end of *lst*. Returns **None**.
lst.extend(other)	Adds each of the elements in the list *other* to the end of *lst*. Returns **None**.
lst.index(elmt)	Returns index of the first occurrence of *elmt* in *lst,* error if *elmt* is not in lst
lst.count(elmt)	Returns the number of times *elmt* occurs in *lst*
lst.remove(elmt)	Removes first occurrence of *elmt* from *lst*, error if *elmt* is not in *lst*. Returns **None**.
lst.pop(idx) lst.pop()	Removes and returns the element at index *idx* in *lst*. With no parameter, removes the last element in *lst*
lst.insert(idx, elmt)	Inserts an element *elmt* in *lst* at index *idx.* Returns **None**.
lst.sort()	Sorts the given list *lst*. Returns **None**.

File I/O

Function	Description
`my_file = open(`*`filepath`*`)`	Opens the file with given *`filepath`* for reading, returns a file object
`my_file.close()`	Closes file `my_file`
`with open(`*`filepath`*`) as `*`f`*`:` `    # read file`	Opens the file with given *`filepath`* for reading via the file object *`f`* in the body of the "with" statement.

`reader = csv.DictReader(`*`f`*`)`
`for record in reader:`
`    # process each record`
	(Assumes "`import csv`" at the top of the Python file and that the file is open.) Reads the file as the comma-separated-value format, returning a list of dictionaries.

"string".strip()	Given a string **"string"**, remove any trailing or ending whitespace.
"string".split([delimiter])	Returns a list where the elements are the result of separating **"string"** by **[delimiter]** (which defaults to all whitespace if not given).

```
# Process one line at a time:          # Process entire file at once
 for line_of_text in my_file:          all_data_as_a_big_string = my_file.read()
     # process line_of_text
```

Dictionaries

Function	Description
`my_dict`` = {}` *`my_dict`*` = dict()`	Creates a new, empty dictionary
`my_dict``[`*`key`*`]`	Returns the value associated with the given *`key`* in *`my_dict`*
`del my_dict[`*`key`*`]`	Removes the *`key`* (and its associated value) from *`my_dict`*
`list(`*`my_dict`*`.keys())`	Returns a list of keys in *`my_dict`*
`list(`*`my_dict`*`.values())`	Returns a list of values in *`my_dict`*
`list(`*`my_dict`*`.items())`	Returns a list of tuples of the form *`(key, value)`*
`sorted(`*`my_dict`*`)`	Returns a sorted list of the keys in *`my_dict`*

```
# Process each key-value pair together:    # Process one key at a time
for key, value in my_dict.items():         for key in my_dict:
    # process key and value                    # use dictionary's key
```

Classes

Function	Description

`class Name:`
    `# class methods, for example:`
    `def method(self, [args]):`
        `# method body`
	Defines a new class named *Name* with the subsequently defined methods.

`def __init__(self):` `# method body`	The function that is called during class construction/creation, as in *Name()*.
self	Required parameter for all class methods (functions). Refers to the specific instance of the class. Can hold any number of arbitrary variables, as in *self.name*
n = `Name()`	Instantiates (creates/constructs) a new instance of the *Name* class and assigns a reference to it in the variable *n*.
n.`method(`*[args]*`)`	Calls the *method* function on the instance defined in *n*, optionally passing in any required arguments.

Common Errors

IndexError – Index out of range

AssertionError – Assert failed

KeyError – Key not found in dictionary

IndentationError – Invalid indentation

TypeError – Operation applied to invalid combination of types

ValueError – Function gets properly typed argument, but invalid value

SyntaxError – Invalid Python syntax

NameError – Variable name not found

FloatingPointError – Floating point operation fails

RuntimeError – Otherwise Unknown Error