Full Name: _____

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

Section: _____

CSE 160 Autumn 2025 - Midterm Exam

Instructions:

- You have **the entire class period (50 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.

Good luck!

Question	**Topic**	**Points**
Question 1	Expressions	10
Question 2	Functions	8
Question 3	File I/O	8
Question 4	Lists	12
Question 5	Nested Structures	12

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**

`x = 3`
`y = 7`
`x > y`		

`3 / 2 + len("hello")`		
`[1, 3, 5][1] + {"1": 1, "2": 2, "3": 3}["1"]`		
`"I am taking " + 160`		
`str(3456) + "boom"`		

2 (8 points). Consider the following functions.

```
def bibbity(a, b):
      result = ""
      for i in range(a, b):
         if i % 2 == 0:
            result += "*"
      return result

def bobitty(pumpkin):
      c = pumpkin[0]
      d = pumpkin[len(pumpkin) - 1]
      return bibbity(c, d)

def boo(mice, pumpkin):
      wand = mice + "!"
      carriage = bobitty(pumpkin)
      if len(wand) >= len(carriage):
            return("Dress!")
      else:
            return("midnight")
```

What is the output of the following function calls? If there is an error, write “Error”. You do not need to specify the error.

Input	Output
`boo("***", [1])`	
`boo("*******", [])`	
`boo("*", [1, 2])`	
`boo("*****", [2, 8, 7, 12, 15])`	

3 (8 points). Consider the file `shops.csv` below. The file contains the name of a chain restaurant or shop along with all the reviews each individual store has received. You can assume each shop name only appears once.

`shops.csv:`

```
McDonalds,1,4,2,3,3
Taco Bell,1,2,3,3,3
Chik-fil-A,4,3,5,2,1
Starbucks,2,3,3,2,2
Victrola,4,5,4,5,5
Tim Hortons,4,3,3,2,1
```

Your friend tried to write code that would read in the file and store the contents in a dictionary where the key is the store name and the value associated with the key is a list of all its reviews as integers. The intended output is shown below:

```
{'McDonalds': [1, 4, 2, 3, 3], 'Taco Bell': [1, 2, 3, 3, 3],
 'Chik-fil-A': [4, 3, 5, 2, 1], 'Starbucks': [2, 3, 3, 2, 2],
 'Victrola': [4, 5, 4, 5, 5], 'Tim Hortons': [4, 3, 3, 2, 1]}
```

However, your friend has tried but can't seem to figure out why their code is not working correctly. Their code is below:

```
1   file = open(shops.csv)
2   d = {}
3
4   for line in file:
5      data = line.strip().split()
6      d[data] = []
7      for r in data:
8          d[data[0]].append(r)
9    file.close()
```

Continued on the next page…

Name the line number(s) that contain errors. You do not need to specify what the error is. For each line you identify as containing an error, rewrite the line so that it's now correct. Fixing all errors should lead to the code achieving the correct output. You cannot add or remove lines of code, you can only update the existing lines. You may structure your answer like the following:

```
Line X has an error. The correct version is <correct version of the code>
```

```
// Write your answers here
```

4 (12 points). Suppose you are given a nested list where each list represents a student along with their score on 3 assignments. Write a function `gradebook()` that takes in the nested list and returns a dictionary where the key is the student name and the value is their average score across those three assignments. You **cannot** use any built in `sum()` or `avg()` Python methods.

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

```
grades =[ ['Trixie', 70, 100, 88],
          ['Ricky', 77, 86, 83],
          ['Steve', 75, 62, 94] ]

gradebook(grades) → {'Trixie': 86.0, 'Ricky': 82.0, 'Steve': 72.0}
```

```
# Write your code here
```

5 (12 pts)

5a (6 pts). Write a function `get_diagonal` that takes in a nested list and returns a new list of items that form a diagonal across the nested list. You can assume the nested list makes a square, i.e. the nested list contains n lists that each have n elements. Your solution should work for a squared nested list of any size. For example:

```
nested_list  =  [["I", 5, "hello", 4.20, "Nailing"],
                [77, "love", 87.23, "my", 2],
                [2000, 5241, "CSE", "?", 4],
                ["wow", "exam", 69, 160, False],
                ["rn", 1, "pumpkin", True, "!"]]
```

`get_diagonal(nested_list)` should return
`["I", "love", "CSE", 160, "!"]`

```
# Write your solution here
```

5b (6 pts). Now rewrite this function as `reverse_diagonal` to return the diagonal in the opposite direction. Do not use `.reverse()`. Your solution should work for a squared nested list of any size. The code should return:

```
["Nailing", 'my', 'CSE', 'exam', "rn"]
```

```
# Write your solution here
```

Extra Credit (1 point): Draw anything (keep it PG and appropriate).

CSE 160 25au Midterm 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*)	Returns the number of elements in *lst*

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**.
lst`.reverse()`	Reverses the order of elements in the 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.
`str.strip()`	Given a string `str`, remove any trailing or ending whitespace.
`str.split([separator])`	Given a string `str`, splits the string on the optional separator (will split on spaces if no separator is given) and returns a list of separated strings.

```
# Process one line at a time:
for line_of_text in my_file:
    # process line_of_text
```

```
# Process entire file at once
all_data_as_a_big_string = my_file.read()
```

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)`***

```
# Process each key-value pair together:
for key, value in my_dict.items():
    # process key and value
```

```
# Process one key at a time
for key in my_dict:
    # use dictionary's key
```

Common Error Names

IndexError – Index out of range
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