Full Name: ANSWER KEY!

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`
	False	boolean

`3 / 2 + len("hello")`	6.5	float
`[1, 3, 5][1] + {"1": 1, "2": 2, "3": 3}["1"]`	4	int
`"I am taking " + 160`	Error!	Error!
`str(3456) + "boom"`	'3456boom'	string

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 what the error.

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

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


Line 1 has an error, should be file = open("shops.csv")

Line 5 has an error, should be data = line.strip().split(",")

Line 6 has an error, should be d[data[0]] = []

Line 7 has an error, should be for r in data[1:] (we only want to iterate over
the scores)

Line 8 has an error, should be d[data[0]].append(int(r))
```

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



One possible solution

def gradebook(grades):
   result = {}
   for student in grades:
      name = student[0]
      total = 0
      for score in student[1:]:
         total+=score
      average = total/(len(student)-1)
      result[name] = average
   return result
```

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

def get_diagonal(nested_list):
    result = []
    for i in range(len(nested_list)):
        result.append(nested_list[i][i])
    return result
```

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

def reverse_diagonal(nested_list):
    result = []
    for i in range(len(nested_list)):
        result.append(nested_list[i][len(nested_list)-1-i])
    return result



// Using negative indices
def reverse_diagonal(nested_list):
    result = []
    for i in range(len(nested_list)):
        result.append(nested_list[i][-1-i])
    return result
```

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

Anything as long as something appropriate was drawn or written