Full Name:	Answer Key
Email Address (UW Net ID):	Answer Key @uw.edu
Section:	Answer Key

CSE 160 Autumn 2024 - Midterm Exam

Instructions:

- You have **40 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**
Question 1	Expressions
Question 2	File I/O
Question 3	Nested Loops
Question 4	Dictionaries
Question 5	Nested Lists

1. Given the table below, fill in the correct values and type for the matching expression. 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**
`{"a": 10, "b": 20}["c"]`	**Error**	
`[1, 2, 3][1] + (4, 5)[0]`	**6**	**int**
`"10" + 6 / 2`	**Error**	
`len([1, 2, 3, 4]) / 2`	**2.0**	**float**

`x = set([5, 6, 7, 8])`
`x.add([9, 10, 11, 12])`
	Error	

2. For this problem, you are given a file named `list_of_points.txt`. The contents of this file are shown below:

list_of_points.txt

```
Point1: 1 2 3 4
Point2: 5 6 7 8
Point3: 0 0 0 0
```

Predict the output of the following lines of code if it is given this file as input.

```
my_file = open("list_of_points.txt")
for line in my_file:
    words = line.split()
    point = []
    for coordinate in words[1:]:
        point.append(int(coordinate))
    print(words[0], point)
my_file.close()
```

```
Point1: [1, 2, 3, 4]
Point2: [5, 6, 7, 8]
Point3: [0, 0, 0, 0]
```

3. You are interested in whether ChatGPT overuses certain words (for example, "delve") in the answers it gives. Given a ChatGPT answer and a list of overused words, you decide to write a function to count the number of times it uses one of the overused words.

```
1    def count_overused(answer, overused):
2        total = 0
3        words = answer.split()
4        for word in range(len(words)):
5            for overused_word in range(len(overused)):
6                if word == overused_word:
7                    total += 1
8        return total
```

For example, if `overused = ["delve", "tapestry", "landscape", "dive"]` and `example_answer = "Let's delve into how the tapestry of the landscape was influenced by the war of 1812"`.

`count_overused(example_answer, overused)` should return 3.

a) Unfortunately, that is not what is returned if this code is run. Assuming that capitalization and spelling is correct, what number will be returned instead in the given example? Explain where that number is coming from.

This function returns 4 instead because the for loops are looping through indexes of the list / string rather than values / characters.

b) What small change (affecting 2 lines of code or less) would you implement in this function so that it would have the correct behavior? Your answer should cite specific line numbers and show the code you would replace that line with.

Lines 4 and 5 should remove the range(len(. For example:

```
for word in words:
    for overused_word in overused:
```

4. The sub-question (a) on the next page uses the following code.

```
1     def category_grade(earned, possible):
2         earned_points = 0
3         total_possible = 0
4         for i in range(len(earned)):
5             earned_points += earned[i]
6             total_possible += possible[i]
7         return earned_points / total_possible
8
9     def calculate_total_grade(earned, possible, weights):
10        total = 0
11        for i in range(len(weights)):
12            category_grade = category_grade(earned[i], possible[i])
13            total += category_grade * (weights[i] / 100)
14        return total * 100
```

These functions are designed to calculate the total grade for a student based on earned points, possible points, and weights for different categories of assignments.

This code was initially written assuming `weights` was a list and that `earned_points` and `possible_points` were nested lists. Now, they have been converted to dictionaries with keys `"homework"`, `"participation"`, and `"exams"`, but the code has **not** yet been updated to reflect this change. Below are examples of how these structures looked as lists and then after the change to use dictionaries.

BEFORE (as lists)

```
earned = [
    [8, 9, 7],
    [4, 3],
    [25]
]

possible = [
    [10, 10, 10],
    [5, 5],
    [30]
]

weights = [50, 20, 30]
```

AFTER (as dictionaries)

```
earned = {
    "homework": [8, 9, 7],
    "participation": [4, 3],
    "exams": [25]
}

possible = {
    "homework": [10, 10, 10],
    "participation": [5, 5],
    "exams": [30]
}

weights = {
    "homework": 50,
    "participation": 20,
    "exams": 30
}
```

a) Identify which line(s) in the `category_grade` and `calculate_total_grade` functions need to be changed in order to make it work with `earned` and `possible` as dictionaries. Then, describe the necessary change, written as the code to replace the identified line(s).

Line number(s): ______**11**______

Change(s) required:

Change `for i in range(len(weights))` to `for i in weights`.

Additionally, lines 11, 12, 13 could change to use key instead of `i`, which would be more clear.

b) (An alternate version of the exam also asked for the result of the function call `calculate_total_grade(earned, possible, weights)`.)

79.0

5. Doctors can use MRI (Magnetic Resonance Imaging) scans of parts of your body to diagnose certain diseases without having to perform surgery. Because our organs are 3-dimensional, these images are also 3D (unlike the 2D images that you were working with in Homework 3). If we wanted to represent this in Python, we would have to use 3D *voxels* rather than 2D pixels as shown below.

An example 2 x 2 pixel_grid:

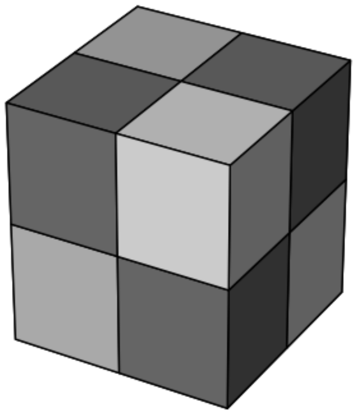
```
pixel_grid = [[1, 15],
              [20, 5]]
```

Rendered as a grid like so:

1	15
20	5

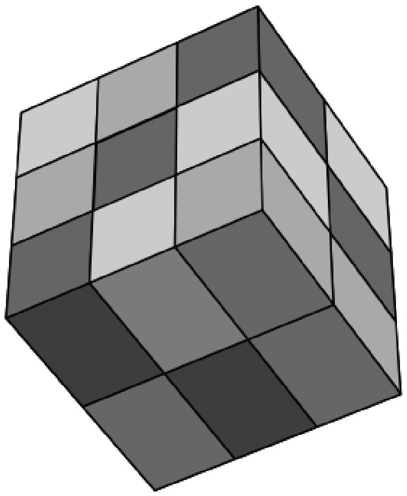
An example 2 x 2 x 2 voxel_grid:

```
voxel_grid = [
    [[3, 2],
     [2, 3]],

    [[2, 4],
     [4, 2]]
]
```



You can view these voxel grids as a series of 2D grids stacked together. For example, below is a 3 x 3 x 2 voxel grid and its rendering. (Each value in the grid for these renderings represents the color of an individual block, just as it might for a pixel in the 2D grids.)

```
voxel_grid = [
    [[3, 4, 2],
     [4, 2, 3],
     [2, 3, 4]],

    [[3, 2, 4],
     [2, 3, 2],
     [3, 4, 3]],
]
```



a) Implement a function called `remove_noise` which when given a `voxel_grid` returns a *new* voxel_grid that has 0s in place of all voxels with values < 10, with all other voxels being the same as the original. Altering the parameter `voxel_grid` will not give you full credit for this question.

```
def remove_noise(voxel_grid):

    new_grid = []
    for depth in voxel_grid:
        layer_1 = []
        for row in depth:
            layer_2 = []
            for col in row:
                if col < 10:
                    layer_2.append(0)
                else:
                    layer_2.append(col)
            layer_1.append(layer_2)
        new_grid.append(layer_1)
    return new_grid
```

Extra Credit (one point): Write a one-liner joke about programming, computers, python, or a similar topic of your choice. For example:

What do you call a snake that's good at math *and* programming? A Pi-thon!

(You may not use this one.🐍😎)