Set 10: Two-Dimensional Lists

Skill 10.1: Create a two-dimensional list in Python

Skill 10.1 Concepts

We've seen that the items in a list can be numbers or strings. Lists can contain other lists! We will commonly refer to these as *two-dimensional (2D)* lists.

Let's look at a class height example,

- Noelle is 61 inches tall
- Ava is 70 inches tall
- Sam is 67 inches tall
- Mia is 64 inches tall

Previously, we saw that we could create a list representing both Noelle's name and height,

```
noelle = ["Noelle", 61]
```

Conversely, we could create parallel lists, where the name of each person maps to their height in a separate list.

Code	Output

`names = ["Noelle", "Ava", "Sam", "Mia"]`
`heights = [61, 70, 67, 64]`
`print(names[0], "is", heights[0], "inches tall")`
`print(names[1], "is", heights[1], "inches tall")`
`print(names[2], "is", heights[2], "inches tall")`
`print(names[3], "is", heights[3], "inches tall")`
	Noelle is 61 inches tall
	Ava is 70 inches tall
	Sam is 67 inches tall
	Mia is 64 inches tall

Another way to manage this data is to create a list for each person and store all the individual lists in one bigger list,

```
heights = [["Noelle", 61], ["Ava", 70], ["Sam", 67], ["Mia", 64]]
```

We will often find that a two-dimensional list is a very good structure for representing grids such as games like tic-tac-toe.

```
tic_tac_toe = [
            ["X","O","X"],
            ["O","X","O"],
            ["O","O","X"]
]
```

Skill 10.1 Exercise 1

Skill 10.2: Access two-dimensional lists elements

Skill 10.2 Concepts

To understand how to access elements in a two-dimensional list, let's consider the gradebook below,

gradebook		Columns →				
		0	1	2	3	4
		assignment 1	assignment 2	assignment 3	assignment 4	assignment 5
Rows ↓ 0	Bart	5	4.5	3	4	3.5
1	Kyle	2	3	2.5	4	R
2	Bugs	4	1	3.5	5	4
3	Marvin	5	2	3.5	3	4.5

The gradebook can be visualized as a collection of rows and columns. From the diagram above we see that Bart's grades are in row 0, Kyle's grades are in row 1, etc. The grades for each student are stored in sub-lists represented by the columns. For example, Bart's grade on assignment 1 is stored in column 0, his grade on assignment 2 is stored in column 1.

We can access any data member in the 2D list above by referencing the appropriate row and col.

location	data member
row = 1 col = 3	4
row = 0 col = 4	3.5
row = 3 col = 1	2
row = 2 col = 4	4
row = 1 col = 5	out of bounds

The gradebook can be represented as a two-dimensional list in Python,

```
grade_book = [
    [5, 4.5, 3, 4, 3,5],
    [2, 3, 2.5, 4, "R"],
    [4, 1, 3.5, 5, 4],
    [5, 2, 3.5, 3, 4.5]
    ]
```

To access elements in the two-dimensional list, we can use the following convention,

`list[row_number][col_number]`

For example,

Code	Output
`result = grade_book[1][4]` `print(result)`	R

Skill 10.2 Exercises 1 thru 2

Skill 10.3: Modify elements in a two-dimensional list

Skill 10.3 Concepts

Now that we know how to access two-dimensional lists, modifying the elements should come naturally.
Let's return to a classroom example, but now instead of heights or test scores, our list stores the student's favorite hobby!

```
class_name_hobbies = [["Jenny", "Breakdancing"], ["Alexus", "Photography"], ["Grace",
"Soccer"]]
```

"Jenny" changed their mind and is now more interested in "Meditation".

We will need to modify the list to accommodate the change to our class_name_hobbies list. To change a value in a two-dimensional list, reassign the value using the specific index.

Code

```
class_name_hobbies = [["Jenny", "Breakdancing"], ["Alexus", "Photography"], ["Grace", "Soccer"]]
class_name_hobbies[0][1] = "Meditation"
print(class_name_hobbies)
```

Output

```
[['Jenny', 'Meditation'], ['Alexus', 'Photography'], ['Grace', 'Soccer']]
```

Negative indices will work as well.

Code

```
class_name_hobbies = [["Jenny", "Breakdancing"], ["Alexus", "Photography"], ["Grace", "Soccer"]]
class_name_hobbies[-1][-1] = "Football"
print(class_name_hobbies)
```

Output

```
[['Jenny', 'Breakdancing'], ['Alexus', 'Photography'], ['Grace', 'Football']]
```

Skill 10.3 Exercise 1

Skill 10.4: Apply the *zip()* function to combine lists

Skill 10.4 Concepts

In Python, we have an assortment of built-in functions that allow us to build our programs faster and cleaner. One of those functions is zip().

The *zip()* function allows us to quickly combine associated data-sets without needing to rely on multi-dimensional lists. While *zip()* can work with many different scenarios, we are going to explore only a single one in this article.

Let's use a list of student names and associated heights as our example data set:

- Jenny is 61 inches tall
- Alexus is 70 inches tall
- Sam is 67 inches tall
- Grace is 64 inches tall
-

Suppose that we already had a list of names and a list of heights,

```
names = ["Jenny", "Alexus", "Sam", "Grace"]
heights = [61, 70, 67, 64]
```

If we wanted to create a nested list that paired each name with a height, we could use the built-in function *zip().*

The *zip()* function takes two (or more) lists as inputs and returns an *object* that contains a list of pairs. Each pair contains one element from each of the inputs. This is how we would do it for our names and heights lists:

Code

```
names = ["Jenny", "Alexus", "Sam", "Grace"]
heights = [61, 70, 67, 64]
names_and_heights = zip(names, heights)
print(names_and_heights)
```

Output

```
<zip object at 0x000001E7A66D4C80>
```

This *zip object* contains the location of this variable in our computer's memory. Don't worry though, it is fairly simple to convert this object into a useable list by using the built-in function *list()*,

Code

```
converted_list = list(names_and_heights)
print(converted_list)
```

Output

```
[('Jenny', 61), ('Alexus', 70), ('Sam', 67), ('Grace', 64)]
```

Notice two things:

1. Our data set has been converted from a zip memory object to an actual list (denoted by [ ])
2. Our inner lists don't use square brackets [ ] around the values. This is because they have been converted into tuples (an immutable type of list that we will learn more about!)

Skill 10.4 Exercise 1