Unit 5 - Basic Data Structures - Study Guide

Tuples and Lists

- A **Data Structure** is used in programming to store more than 1 piece of information in a single variable.
- We learned about 2 types of **Data Structures** in this unit: Tuples and Lists.
- Tuples
  - An **immutable** data structure. This means that its contents cannot be changed once it's been initialized.
  - Created using **parentheses ( )** around multiple pieces of data, separated by commas
- Lists
  - A **mutable** data structure. Unlike a tuple, its contents **can** be changed after initialization.
  - Created by **square brackets [ ]** around multiple pieces of data, separated by commas
- Indexing, Slicing, Concatenation
  - All list and tuple variables store multiple pieces of data within them. Each of these pieces of data is referred to as an **element**.
  - Each element within a list or tuple has an **index** - this is the numerical address where the element is located within its structure.
    - The first index is always 0, then counting increases by 1 for each subsequent element.
    - In Python, you can also access a list or tuple from a **negative** index, which will begin accessing from the **end**. -1 is the last element, -2 next, etc.
  - An element can be accessed from a list or tuple variable by putting that element's index in between square brackets after the variable's name.
    - Example: `my_list[0]`
  - A **slice** of a list or tuple can be accessed in a similar way to how indexing is performed: the desired indices are placed in square brackets after the variable's name, separated by a colon. The first index is **included**, and the second index is **excluded**.
    - Example: `my_list[0:6]`
    - A slice of a list will be a list. A slice of a tuple will be a tuple!
  - Lists can be concatenated together with other lists, and tuples can be concatenated together with other tuples!
    - If you want to add a single element to a list, it's best to use the `append()` function *(see below)*
    - If you want to append a single element to a tuple, you must turn that value into a tuple itself. This can be done by putting it inside of parentheses with a comma after it, like so: `my_tuple + (3, )`

Using Loops with Lists

- Since the indices within a list start at 0 and the `range()` function gives a sequence of numbers starting at 0, we can use this to loop over every value in a list
- We can use the `len()` function to determine how many elements are present within a list!
  - ```
    for i in range(len(my_list)):
        print(my_list[i])
    ```
- The alternative style of for loop will have the **iterator** take the value of every element within a list, rather than each of the indices.
  - ```
    for element in my_list:
        print(element)
    ```
- It's possible to create a For Loop that combines the powers of both of these types of loops, using the enumerate() function. This will create a loop with 2 loop variables - one for the indices, and one for the elements.
  - ```
    for i, element in enumerate(my_list):
        print(str(i) + ": " + str(element))
    ```
    ```

List Functions

- There are many functions built into lists that can affect their lengths. Here are the important ones for the quiz:
  - `my_list.append(element)`
    - This will add the parameter `element` to the **end** of `my_list`
  - `my_list.extend(list_two)`
    - This will add the elements in the parameter `list_two` to the **end** of `my_list`
  - `my_list.sort()`
    - This will sort the elements in `my_list` into **ascending order**
  - `my_list.reverse()`
    - This will flip the elements in `my_list` around so that it ends up backwards
  - `my_list.remove(element)`
    - This will find the first place where `element` can be found in `my_list`, then remove it
      - If `element` isn't found in `my_list`, it will break your program!
- These functions are not called on lists, but are very useful in converting lists to and from string values!
  - `list(my_string)`
    - This will return a list where every element is one of the characters in `my_string`.
      - `list("hey hey")` will return `["h", "e", "y", " ", "h", "e", "y"]`
  - `my_string.split()`
    - This will return a list where every element is one of the *words* in `my_string`.
      - `list("hey hey")` will return `["hey", "hey"]`
  - `str.join(my_list)`
    - This will return a string where `str` is concatenated between every element of `my_list`.
      - `"m".join(["h", "e", "y", " ", "h", "e", "y"])` will return `"hmemeym mhmemy"`