Spencer Olson **February 11**

Lab for Chapter 3 – from Kaefer pp. 38-52 (Lists, Strings, Tuples)

3.2. Use the input() function learned in Chapter 2 to input three strings (a first name, a middle initial, and a last name). Concatenate these strings together with a space between each, storing the result in a new variable. Print out the concatenated string.

a. Do this using the "+" concatenation operator. Explain how Python knows this is different than adding two numbers together.

```
full_name = first_name + " " + middle_initial + " " + last_name
print("Full name:", full_name)
```

Its combining variables that have input strings. Where nothing has been defined as numbers.

b. Do this using an f-string

```
full_name = f"{first_name} {middle_initial} {last_name}"
print(f"Your full name is ", full_name)
```

c. Do this using the % Operator

```
full_name = "%s %s %s" % (first_name, middle_initial, last_name)
print("Your full name is {full_name}.")
```

d. Do this using the `format()` method for a string

```
full_name = format(first_name + middle_initial + last_name)
print(full_name)
```

e. Do this using a `join()` method of a list

```
full_name = " ".join([first_name, middle_initial, last_name])
print(full_name)
```

f. Do this using the `format()` method for a string but unpacking the list as the argument

```
name_parts = [first_name, middle_initial, last_name]
full_name = "{} {} {}".format(*name_parts)
print(full_name)
```

3.3. Define a list of survey response values (5, 7, 3, and 8) and store them in a variable. Next define a tuple of respondent ID values (1012, 1035, 1021, and 1053). Use the .append() method to append the tuple to the list. Print out the list.
Code here:

```
survey_responses = [5 , 7 , 3 , 8]
respondent_ids = (1012, 1035, 1021, 1053)

survey_responses.append(respondent_ids)

print(survey_responses)
```

a. Explain the output.

The output is [5, 7, 3, 8, (1012, 1035, 1021, 1053)] Basically includes all items appended into a single output of both items.

b. Discuss how you might make use of the list and what problems you might have using it.

You can use the list to generate a linkage between a string and a list. This would connect them together in their positioning. It assists in organizing large data sets of information.

c. What might be a better way to store survey response values and their respective respondent ID values?

For this case scenario a dictionary may work better to store the information as we can attach responses to the id's themselves.

d. Compare and contrast a List object with a String object
Similarities
Indexing and slicing : Integration : Concatenation : Containment checking
Key Differences
Mutability: Lists are mutable, while strings are immutable.
Data Storage: Lists store multiple elements of different types, whereas strings store only text.
Modification: List elements can be modified individually, but string modifications require creating a new string.

e. Look up some documentation or tutorial on Arrays in Python. Compare and contrast with a List object. Explain why you might want to use an array instead of a list.

Use an array in Python when you need memory efficiency, faster performance, or type safety for homogeneous data, especially for large datasets of the same type, like numbers. Arrays are more efficient than lists for numerical operations and are commonly used with libraries like NumPy. Choose a list when you need flexibility, as they can store elements of different types and provide more built-in methods for manipulation. Lists are more general-purpose and are easier to work with in most everyday scenarios.

3.4. Create the list of responses from Exercise 3.3 (values 5, 7, 3, 8). Next add the response "0" to the end of the list using the .append() method. Next add the response "6" to the list between the values 7 and 3 (in what will be the third position in the list) using the .insert() method. Print out the list to verify that you made the changes correctly.

a. Code here:

```
responses = [5,7,3,8]
responses.append(0)
responses.insert(2,6)
print(responses)
```

b. Do the same operations using list slicing (e.g. [:2]) and the "+" operator rather than .insert(). Explain what the "+" operator does on lists compared to strings and numbers.

```
responses = responses[:2] + [6] + responses[2:]
```

c. Why might you use list slicing over .insert()?
List slicing is useful when inserting multiple elements or when you want to preserve the original list, as it creates a new list rather than modifying the original. It also offers clearer syntax for concatenating parts of the list. .insert() **is more straightforward for inserting a single element at a specific index, making it simpler for small modifications. However, list slicing may use more memory since it creates new lists, while** .insert() **modifies the list in place.**

3.5. Create the tuple of survey respondent IDs from Exercise 3.3 (1012, 1035, 1021, and 1053). Next attempt to add the respondent ID "1011" to the end of the tuple using the .append() method.

Code here:

```
respondent_ids = (1012, 1035, 1021, 1053)
respondent_ids.append(1011)
```

a. What happens when you attempt to run the code? Explain the problem.

The AttributeError will occur because tuples do not have the .append() method.

b. Address the problem above using the "+" operator with the tuple (1011,). Why could you not just use 1011 or (1011)?
Because you are adding a string into the other strings. If you just type 1011 or (1011) you will not be adding anything into anything. You need the operator "+" as an inserting mechanism.

c. Compare and contrast tuples and lists

Tuples and lists both store ordered collections of items, but tuples are immutable **(cannot be changed after creation), while lists are** mutable **(can be modified). Lists support operations like .append(), .insert(), and .remove(), whereas tuples do not allow element modification. Tuples are typically used for fixed data structures, making them faster and more memory-efficient than lists. Lists, on the other hand, are better for dynamic data that needs frequent updates.**

d. Why do you think tuples are not changeable (immutable)?
For reliability tuples are a fixed set, this increases speed and data integrity.