3.2.5 Data Structures: Lists

Now that you have been exposed to Python data types, Seth and Tom want you to learn how data types are stored and accessed. In the election audit, you may need to store or access data in a variety of formats, such as lists, tuples, and dictionaries. Accessing the data in these data structures as well as storing data in new files are common tasks that data analysts perform every day.

One of the basic data structures in Python is a list. A **list** is an array that contains multiple data items, like the following list of counties.

```
counties = ["Arapahoe", "Denver", "Jefferson"]
```

The items in a list can be data types such as integers, floating-point decimals, strings, and Boolean values, as well as other data structures like lists, tuples, and dictionaries. (We'll cover tuples and dictionaries later in this module.)

IMPORTANT

> When it comes to list, remember these three properties.
>
> 1. We can use **indexing** and **slicing** to retrieve specific items from the list.
> 2. We can add or remove items from a list, which makes lists a **dynamic** data structure.
> 3. We can change the contents in a list. For example, we can change "Jefferson" to "El Paso." This means lists are **mutable:** we can change one or more items in a list to something else.

Let's look at these three features in action.

Activate the Python interpreter. We'll first declare a list variable `counties` by typing the following code and then pressing Enter.

```
>>> counties = ["Arapahoe","Denver","Jefferson"]
```

NOTE

> When we want to add items to a list but the list has not been declared, we must first declare an empty list. An **empty list** can be declared with the following syntax: `my_list = [ ]`. Alternatively, you can use the built-in function `list()` to create an empty list: `my_list = list()`.

To confirm that this list is declared, type `counties` on the next line and press Enter. The output should be the following:

```
>>> counties
['Arapahoe', 'Denver', 'Jefferson']
```

Index Lists

In Python, we access items in a list using indexing.

REWIND

An **index** of a variable is its position in the array. Here are some general rules for indexing:

1. Each item in a list has an index that specifies its position in the list.
2. Indexing starts at 0. Therefore, the index of the first item is 0, the index of the second number is 1, and so on.
3. Because indexing begins at 0, the index of the last item in a list is 1 less than the number of items in the list.

To get the **first** item in the `counties` list, type the following in the command line:

```
>>> counties[0]
```

Press Enter to get the output, which is `'Arapahoe'`.

```
>>> counties[0]
'Arapahoe'
```

Retake

You can also use the `print` statement to print the county to the screen, like this:

```
>>> print(counties[2])
Jefferson
```

Negative indexes are used to identify a list item's position relative to the end of the list. For example, to find the **last** item in the `counties` list, we would enter the following:

```
>>> print(counties[-1])
Jefferson
```

To get the **second-to-last** item in a list, we would type `counties[-2]`, and so forth.

Find the Length of a List

To get the total number of items in a list, we use the `len()` function and then add the `list` inside the parentheses, like this: `len(counties)`. When we execute this script, the output will be the number of items in the list, which is 3.

```
>>> len(counties)
3
```

Slice Lists

Sometimes we'll need to retrieve certain items from the list. To do this, we can use the index values to slice a list. **Slicing** is used to get specific items from a list. The format for slicing a list is as follows: `list[start : end]`.

Let's break down how slicing works.

1. The `start` refers to the index of the first item in the slice.
2. The `end` is the index marking the end of the slice.
3. The expression `list[start : end]` returns a list containing a copy of the items in the list from the starting index value up to, but not including, the ending index value.

For example, to find the first and second items from the `counties` list, we type `counties[0:2]`, not `counties[0:1]`.

The output of `counties[0:2]` will be `Arapahoe` and `Denver`, the first and second items from the `counties` list.

```
>>> counties[0:2]
['Arapahoe', 'Denver']
```

The output of `counties[0:1]` will be `Arapahoe`, the first item from the `counties` list.

```
>>> counties[0:1]
['Arapahoe']
```

Alternatively, you can use `counties[:2]` to get the first and second items from the `counties` list.

```
>>> counties[:2]
['Arapahoe', 'Denver']
```

With `counties[:2]`, the beginning index is omitted with `[:`, so the slice contains the elements starting from 0 and ending at 2.

Retake

Another option to get `Denver` and `Jefferson` counties in a list is to use `counties[1:]`. Here, the ending index is omitted with `:]`; the slice contains the elements starting at 1 and ending at length of the list, or 3, since there are only three items in the `counties` list.

```
>>> counties[1:]
['Denver', 'Jefferson']
```

Add Items to a List

Items can be added to an empty list or a list that already exists by using the `append()` function and the syntax `list.append()`. In the parentheses, add the data you want, whether integers, floats, strings, or another data type or data structure. For example, let's add a fourth county, El Paso, to the counties list:

```
>>> counties.append("El Paso")
```

When you press Enter, you will notice there is no output—but that doesn't mean nothing happened. To check, type `counties` and press Enter. The output prints the list with El Paso added to the list as the last item.

```
>>> counties
['Arapahoe', 'Denver', 'Jefferson', 'El Paso']
```

NOTE

> Using the `append()` function on a list will always add the new item at the end of the list.

To specify where in a list to add a new item, select the location with an index by using the following syntax, `list.insert(index, obj)`.

Here, `index` represents where we would like the new item to be placed, and `obj` represents the item.

Let's look at an example. We'll add another instance of "El Paso" at index 2, which is the third position in the counties list, to the `counties` from the previous example.

When we execute this code and print the `counties` list, the output is the following:

```
>>> counties.insert(2, "El Paso")
>>> counties
['Arapahoe', 'Denver', 'El Paso', 'Jefferson', 'El Paso']
```

To remove an instance of "El Paso" from our list, we'll append the `.remove` method and specify the list item we're removing.

In the example below, `.remove("El Paso")` was appended to our `counties` variable. In the output, the only instance of "El Paso" remaining is located in the third index.

```
>>> counties.remove("El Paso")
>>> counties
['Arapahoe', 'Denver', 'Jefferson', 'El Paso']
>>>
```

Another method that can be used to remove items from a list is the `pop()` method. The `pop()` method removes the item at a given index from the list and then returns the removed item. This is a good way to check which item was removed.

Retake

To remove the last instance of "El Paso" using the `pop()` method, enter `counties.pop(3)`. "El Paso" is the fourth item in the list, which means its index is 3.

When we execute this code, the output will look like this:

```
>>> counties.pop(3)
'El Paso'
>>> counties
['Arapahoe', 'Denver', 'Jefferson']
>>>
```

Change an Element in a List

Python lists are mutable, which means we can change the elements inside the list. To change an item in a list, use the syntax `list[index]` on the left side of the equals sign; on the right side, you assign the index a new data value. The new value can be an integer, floating-point decimal number, string, Boolean value, another list, tuple, or dictionary.

If you want to change Jefferson county to El Paso, you would type the following: `counties[2] = "El Paso"`.

When you run the code, "Jefferson" is replaced by "El Paso."

```
>>> counties[2] = "El Paso"
>>> counties
['Arapahoe', 'Denver', 'El Paso']
>>>
```

Retake

NOTE

> For more information, see the **documentation on built-in Python functions like list()** **(https://docs.python.org/3.7/library/functions.html)**.