Introduction to Lists

by Sophia

WHAT'S COVERED

In this lesson, we will learn about using a data collection type called a list to store and access data. Specifically, this lesson covers:

1. Arrays and Lists
2. List Properties

1. Arrays and Lists

When working with code, there are times when we may only be working with one element at a time, such as a name or a single price. However, there are other instances when we may need to work with larger sets of data, such as a list of people's names or a list of products. An array is a special type of variable that represents a data collection that's available in most programming languages. A **data collection type** is exactly that, a collection of related values that are organized in a specific way so that they can be used effectively. Although Python doesn't directly support arrays, it supports lists that are very similar to an array.

Before we can get into the code behind lists, it's important that we understand what exactly a list is, how it works, and why we would want to use them.

What is a list?

The **list** is the simplest data collection type in Python and it can store multiple values in a single variable. An **element** is one of the values in a list (or other data collection type). Elements can also be called items.

EXAMPLE
Here is an example of a list:

```
['Cheddar', 'Swiss', 'Gouda', 'American', 'Mozzarella']
```
Looks like a list of cheese...anyone hungry?

How do lists work?

A list consists of elements that can be efficiently accessed using an index or position. The **index** (also known as position) is an integer value that indicates an element in a data collection type like a list. We would use the index to locate a certain element in the list, for example, if we wanted to only select Swiss cheese from the list of cheeses. We'll see how to do that in a minute.

How is a Python list different from an array?

The only key difference between an array and a list in Python is that an array must have all of the values in the array of the same data type. In a list, each element can be of a different data type.

EXAMPLE

For example, an array in Java would look like this:

```
javaArray = {"Ford", "Chevrolet", "Dodge", "GMC"};
```

But for Python, a list can look like this:

```
pythonList = ['Ford', 10, 'Chevrolet', -4.5]
```

Why do we use lists?

An easy explanation or analogy is to think of a list as a parking lot. We can look at the entire parking lot as a whole and think of it as a single object. However, inside the parking lot, there are parking spots that are uniquely identified by a number. There may be parking spots that have a car, a truck, a motorcycle, or even empty. Within a list, the first spot is identified as a 0. This means that if there are 10 elements in a list, the index values go from 0 to 9. If there are 5 spots, the index values of the spots go from 0 to 4.

EXAMPLE

Here is a visual example based on the code:

```
parkingLot = ['motorcycle', '', 'truck', 'car', 'car']
```

`parkingLot` **list**					
Spot	0	1	2	3	4
Vehicle	motorcycle	empty	truck	car	car

Just like a string, a list is a sequence of elements with values. In a string, the values are characters; in a list, they can be any type. There are several ways to create a new list; the simplest is to enclose the elements in square brackets ("[]"). If the elements are string values, make sure to use single or double quotes (just remember to keep them consistent).

EXAMPLE

```
[10, 20, 30, 40]
['hamster', 'rabbit', 'chipmunk']
```

The first example is a list of four integers. The second is a list of three strings. Notice that with the list of numbers, we did not quote the values, similar to how we don't quote individual integers. For the list of strings, we did have to quote each individual element. This can be tricky if our string values have commas as part of an element, like the following:

```
['hamster', 'rabbit, dog, cat', 'chipmunk']
```

In this case, there are still just three strings even though it looks like there could be five. However, we have to look at each start and ending quote. The second string in the list has the value of:

"rabbit, dog, cat"

The elements of a list in Python do not have to be the same type. The following list contains a string, a float, an integer, and another list. Note: a list within another list is called nested (more on that in a later lesson).

```
['spam', 2.0, 5, [10, 20]]
```

Python lists can hold all of these different data elements, which can be a bit confusing. Since everything is an object in Python, we can mix and match different kinds of data types and store them all in a single list.

All of the lists that we're using here are generally quite short as they make the examples easier to work through and manage. However, we could have hundreds or thousands of elements in a list as well.

TERMS TO KNOW

Data Collection Type
A data collection type is a collection of related values that are organized in a specific way so that they can be used effectively.

List
The list is the simplest data collection type in Python and it can store multiple values in a single variable.

Element
An element is one of the values in a list (or other data collection type); elements can also be called items.

Index
The index (also known as position) is an integer value that indicates an element in a data collection type like a list.

2. List Properties

As we discussed earlier, list elements are indexed, meaning the first element in the list has an index of 0, the second element has an index of 1, and so on. Each element in the list is ordered using this index. Each element in the list will follow this syntax.

EXAMPLE
listname[x]

In this example, we would replace `listname` with the name of the list that we want to access and replace the x with the index (position) number that we want. Since the first element is always 0 and not 1, this means that the elements in the list have a defined order and that order will not change unless we change the order. If we append new elements to a list, the new elements are placed at the end of the list.

A list that contains no elements is called an empty list; we can create one with empty brackets, [].

As we might expect, we can assign list values to variables using the variable and the assignment operator or the "=" sign.

EXAMPLE

```
petsList = ['dog', 'cat', 'fish']
numbersList = [17, 123]
emptyList = []
```

Lists are also changeable or **mutable**, meaning we're able to change elements, add elements, or delete elements from a list after it has been created. The syntax for accessing the elements of a list is the same as for accessing the characters of a string: the bracket operator. The expression inside the brackets specifies the index. This index is the position number. Remember that the indices start at 0.

EXAMPLE

```
petsList = ['dog', 'cat', 'fish']
print(petsList[0])
```

And the output would be "dog".

```
dog
```

TRY IT

Directions: Try it out yourself. Output the element in the list that has the value "fish".

What would the index need to be set to? Click the plus (expand) icon on the right to see if you are correct. +

Were you correct? The index would need to be 2 to retrieve "fish" from the list.

Lists are mutable because we can change the order of elements in a list or reassign an element in a list. When the bracket operator appears on the left side of the assignment operator, it identifies the element of the list that will be assigned.

EXAMPLE

```
numbersList = [17, 123]
numbersList[1] = 5
print(numbersList)
```
Here is that output.

```
[17, 5]
```

The element in index 1 (the second number) of the `numbersList` which used to be 123, is now 5.

KEY CONCEPT

Remember this when it comes to the location of list brackets and the assignment operator (“=” sign):

- When the brackets are on the right side of the assignment operator, we are assigning list elements (values) to variables.

```
petsList = ['dog', 'cat', 'fish']
```

- When the brackets are on the left side of the assignment operator, the index inside those brackets identifies an element of the list so you can do something with that element.

```
petsList = ['dog', 'cat', 'fish']
petsList[1] = 'kitten'
```
Now the list is:

```
petsList = ['dog', 'kitten', 'fish']
```
We can think of a list as a relationship between indices (index) and elements. This relationship is called a mapping; each index “maps to” one of the elements.

List indices work the same way as string indices:

1. Any integer expression can be used as an index. Here is an example of an unusual (but still functional) integer expression.

```
petsList = ['dog', 'cat', 'fish']
print(petsList[5-4])
```
Here is that output.

```
cat
```

In the code above, 5-4 is the same as if we had 1 so it would print out index 1 or "cat".

2. If we try to print an element that does not exist, we get an IndexError.

```
petsList = ['dog', 'cat', 'fish']
print(petsList[5])
```

Once run, we get that error.

```
Trackback (most recent call last):
    File "/home/main.py", line 2, in <module>
     print(petsList[5])
IndexError: list index out of range
```

In the example above, the index values go from 0 to 2 so index 5 gives an IndexError exception.

3. If an index has a negative value, it counts backward from the end of the list.

```
petsList = ['dog', 'cat', 'fish']
print(petsList[-1])
```

We still print an element.

```
fish
```

This is a new concept. When we use a negative index (like -1), we start at the end of the list, which is the first item from the last position. So, "fish" was the output. If we had used -2 for the index, the output would have been "cat".

TRY IT

Directions: Go ahead and try some of the ways list indices work like string indices (the examples above). Try a -3. What do you suppose you would get if you attempted a -4?

TERM TO KNOW

Mutable
Mutable means that we're able to change items, add items, or delete items from a data collection type after it has been created.

SUMMARY

In this lesson, we learned about the differences between **lists and arrays**. We found out that the list is the simplest data collection type in Python and it can store multiple values (called elements) in a single variable. We also discussed how to create a list, that a list can be accessed using an index, and that a list index starts at 0 for the first element. We also learned various **list properties**, including that lists are mutable because we can change the order of elements in a list or reassign an element in a list. And finally, we learned that list indices work much the same way as string indices.

Best of luck in your learning!

TERMS TO KNOW

Data Collection Type
A data collection type is a collection of related values that are organized in a specific way so that they can be used effectively.

Element
An element is one of the values in a list (or other data collection type); elements can also be called items.

Index
The index (also known as position) is an integer value that indicates an element in a data collection type like a list.

List
The list is the simplest data collection type in Python and it can store multiple values in a single variable.

Mutable
Mutable means that we’re able to change items, add items, or delete items from a data collection type after it has been created.