Lists and the security analyst

Previously, you examined how to use bracket notation to access and change elements in a list and some fundamental methods for working with lists. This reading will review these concepts with new examples, introduce the `.index()` method as it applies to lists, and highlight how lists are used in a cybersecurity context.

List data in a security setting

As a security analyst, you'll frequently work with lists in Python. **List data** is a data structure that consists of a collection of data in sequential form. You can use lists to store multiple elements in a single variable. A single list can contain multiple data types.

In a cybersecurity context, lists might be used to store usernames, IP addresses, URLs, device IDs, and data.

Placing data within a list allows you to work with it in a variety of ways. For example, you might iterate through a list of device IDs using a `for` loop to perform the same actions for all items in the list. You could incorporate a conditional statement to only perform these actions if the device IDs meet certain conditions.

Working with indices in lists

Indices

Like strings, you can work with lists through their indices, and indices start at `0`. In a list, an index is assigned to every element in the list.

This table contains the index for each element in the list `["elarson", "fgarcia", "tshah", "sgilmore"]`:

element	index
`"elarson"`	`0`
`"fgarcia"`	`1`
`"tshah"`	`2`
`"sgilmore"`	`3`

Bracket notation

Similar to strings, you can use bracket notation to extract elements or slices in a list. To extract an element from a list, after the list or the variable that contains a list, add square brackets that contain the index of the element. The following example extracts the element with an index of `2` from the variable `username_list` and prints it. You can run this code to examine what it outputs:

```
username_list = ["elarson", "fgarcia", "tshah", "sgilmore"]
print(username_list[2])
```

Run
Reset

This example extracts the element at index `2` directly from the list:

```
print(["elarson", "fgarcia", "tshah", "sgilmore"][2])
```

Run
Reset

Extracting a slice from a list

Just like with strings, it's also possible to use bracket notation to take a slice from a list. With lists, this means extracting more than one element from the list.

When you extract a slice from a list, the result is another list. This extracted list is called a sublist because it is part of the original, larger list.

To extract a sublist using bracket notation, you need to include two indices. You can run the following code that takes a slice from a list and explore the sublist it returns:

```
username_list = ["elarson", "fgarcia", "tshah", "sgilmore"]
print(username_list[0:2])
```

Run
Reset

The code returns a sublist of `["elarson", "fgarcia"]`. This is because the element at index `0`, `"elarson"`, is included in the slice, but the element at index `2`, `"tshah"`, is excluded. The slice ends one element before this index.

Changing the elements in a list

Unlike strings, you can also use bracket notation to change elements in a list. This is because a string is **immutable** and cannot be changed after it is created and assigned a value, but lists are not immutable.

To change a list element, use similar syntax as you would use when reassigning a variable, but place the specific element to change in bracket notation after the variable name. For example, the following code changes element at index `1` of the `username_list` variable to `"bmoreno"`.

```
username_list = ["elarson", "fgarcia", "tshah", "sgilmore"]
print("Before changing an element:", username_list)
username_list[1] = "bmoreno"
print("After changing an element:", username_list)
```

Run
Reset

This code has updated the element at index `1` from `"fgarcia"` to `"bmoreno"`.

List methods

List methods are functions that are specific to the list data type. These include the `.insert()`, `.remove()`, `.append()` and `.index()`.

.insert()

The `.insert()` method adds an element in a specific position inside a list. It has two parameters. The first is the index where you will insert the new element, and the second is the element you want to insert.

You can run the following code to explore how this method can be used to insert a new username into a username list:

```
username_list = ["elarson", "bmoreno", "tshah", "sgilmore"]
print("Before inserting an element:", username_list)
username_list.insert(2,"wjaffrey")
print("After inserting an element:", username_list)
```

Run
Reset

Because the first parameter is `2` and the second parameter is `"wjaffrey"`, `"wjaffrey"` is inserted at index `2`, which is the third position. The other list elements are shifted one position in the list. For example, `"tshah"` was originally located at index `2` and now is located at index `3`.

.remove()

The `.remove()` method removes the first occurrence of a specific element in a list. It has only one parameter, the element you want to remove.

The following code removes `"elarson"` from the `username_list`:

```
username_list = ["elarson", "bmoreno", "wjaffrey", "tshah", "sgilmore"]
print("Before removing an element:", username_list)
username_list.remove("elarson")
print("After removing an element:", username_list)
```

Run
Reset

This code removes `"elarson"` from the list. The elements that follow `"elarson"` are all shifted one position closer to the beginning of the list.

Note: If there are two of the same element in a list, the `.remove()` method only removes the first instance of that element and not all occurrences.

.append()

The `.append()` method adds input to the end of a list. Its one parameter is the element you want to add to the end of the list.

For example, you could use `.append()` to add `"btang"` to the end of the `username_list`:

```
username_list = ["bmoreno", "wjaffrey", "tshah", "sgilmore"]
print("Before appending an element:", username_list)
username_list.append("btang")
print("After appending an element:", username_list)
```

Run
Reset

This code places `"btang"` at the end of the `username_list`, and all other elements remain in their original positions.

The `.append()` method is often used with `for` loops to populate an empty list with elements. You can explore how this works with the following code:

```
numbers_list = []
print("Before appending a sequence of numbers:", numbers_list)
for i in range(10):
    numbers_list.append(i)
print("After appending a sequence of numbers:", numbers_list)
```

Run
Reset

Before the `for` loop, the `numbers_list` variable does not contain any elements. When it is printed, the empty list is displayed. Then, the `for` loop iterates through a sequence of numbers and uses the `.append()` method to add each of these numbers to `numbers_list`. After the loop, when the `numbers_list` variable is printed, it displays these numbers.

.index()

Similar to the `.index()` method used for strings, the `.index()` method used for lists finds the first occurrence of an element in a list and returns its index. It takes the element you're searching for as an input.

Note: Although it has the same name and use as the `.index()` method used for strings, the `.index()` method used for lists is not the same method. Methods are defined when defining a data type, and because strings and lists are defined differently, the methods are also different.

Using the `username_list` variable, you can use the `.index()` method to find the index of the username `"tshah"`:

```
username_list = ["bmoreno", "wjaffrey", "tshah", "sgilmore", "btang"]
username_index = username_list.index("tshah")
print(username_index)
```

Run
Reset

Because the index of `"tshah"` is `2`, it outputs this number.

Similar to the `.index()` method used for strings, it only returns the index of the first occurrence of a list item. So if the username `"tshah"` were repeated twice, it would return the index of the first instance, and not the second.

Key takeaways

Python offers a lot of ways to work with lists. Bracket notation allows you to extract elements and slices from lists and also to alter them. List methods allow you to alter lists in a variety of ways. The `.insert()` and `.append()` methods add elements to lists while the `.remove()` method allows you to remove them. The `.index()` method allows you to find the index of an element in a list.

Mark as completed

Like Dislike Report an issue