Activity_Work with strings in Python

October 28, 2025

1 Activity: Work with strings in Python

1.1 Introduction

Security analysts work with a lot of string data. For example, some security analysts work on creating and updating IDs such as employee IDs and device IDs, which are commonly represented as strings. As another example, certain network activity will be stored as string data. Becoming comfortable working with strings in Python is essential for the work of a security analyst.

In this lab, you'll practice creating Python code and working with strings. You'll work with an employee ID, a device ID, and a URL, all represented as string data.

Tips for completing this lab

As you navigate this lab, keep the following tips in mind:

- `### YOUR CODE HERE ###` indicates where you should write code. Be sure to replace that with your own code before running the code cell.
- Feel free to open the hints for additional guidance as you work on each task.
- To enter your answer to a question, double-click the markdown cell to edit. Be sure to replace the "[Double-click to enter your responses here.]" with your own answer.
- You can save your work manually by clicking File and then Save in the menu bar at the top of the notebook.
- You can download your work locally by clicking File and then Download and then specifying your preferred file format in the menu bar at the top of the notebook.

1.2 Scenario

You're working as a security analyst, and you are responsible for writing programs in Python to automate updating employee IDs, extracting characters from a device ID, and extracting components from a URL.

1.3 Task 1

In your organization, employee IDs are currently either four digits or five digits in length. In this task, you're given a four-digit numeric employee ID stored in a variable called `employee_id`. Convert this to a string format and store the result in the same variable. Later, you'll update this employee ID string so that it complies with a new standardized format.

Complete the following code. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[9]:
```
# Assign `employee_id` to a four digit number as an initial value

employee_id = 4186

# Display the data type of `employee_id`

print(type(employee_id))

# Reassign `employee_id` to the same value but in the form of a string

employee_id = str(employee_id)

# Display the data type of `employee_id` now

print(type(employee_id))
```

```
<class 'int'>
<class 'str'>
```

Hint 1

Use the `str()` function in Python to convert the initial value of the `employee_id` variable into a string.

Hint 2

Pass `employee_id` into the `str()` function.

Question 1 What do you observe about the data type of `employee_id` the first time it's displayed? What do you observe about the data type of `employee_id` the second time it's displayed (after the variable is reassigned)?

This is similar to adding the type(argument_to_find_type) like previously. But for this case, the first time the value is printed, the data type of employee_id is integer and in the second time second time, after the variable is reassigned and printed, the data type of employee_id is string.

1.4 Task 2

Imagine that you have just been informed of a new criteria for employee IDs. They must all be five digits long for standardization purposes.

In this task, you will write a conditional statement that displays a message if the length of the employee ID is less than five digits.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[10]:
```
# Assign `employee_id` to a four digit number as an initial value

employee_id = 4186

# Reassign `employee_id` to the same value but in the form of a string

employee_id = str(employee_id)

# Conditional statement that displays a message if the length of `employee_id`␣
↪is less than five digits

if len(employee_id) < 5:
    print("This employee ID has less than five digits. It does not meet length␣
↪requirements.")
```

```
This employee ID has less than five digits. It does not meet length
requirements.
```

Hint 1

The `len()` function in Python can be used to get the length of `employee_id`.

Hint 2

Start the conditional statement with the `if` keyword.

Hint 3

To write the condition in the conditional statement, use the `<` comparison operator to check whether the length of `employee_id` is less than `5`. Make sure to place the condition between the `if` and the `:`.

1.5 Task 3

In this task, you'll build upon the previous code. If an employee ID is only four digits, you'll use concatenation to create a five-digit employee ID number.

Concatenation is a process that allows you to merge strings together. The addition operator (`+`) in Python allows you to concatenate two strings.

Write an `if` statement that evaluates whether the length of `employee_id` is less than `5`. When the condition evaluates to `True`, reassign `employee_id` by concatenating `"E"` in front of the four-digit employee ID to create a five character employee ID. Then, display `employee_id` again. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

[11]:
```
# Assign `employee_id` to a four digit number as an initial value

employee_id = 4186

# Reassign `employee_id` to the same value but in the form of a string
```

```
employee_id = str(employee_id)

# Display the `employee_id` as it currently stands

print(employee_id)

# Conditional statement that updates the `employee_id` if its length is less␣
↪than 5 digits

if len(employee_id) < 5:
    employee_id = "E" + employee_id

# Display the `employee_id` after the update

print(employee_id)
```

```
4186
E4186
```

Hint 1

To complete the header of the conditional statement, use the `if` keyword to start, the `len()` function to get the length of `employee_id`, and the `<` comparison operator to check whether the length is less than `5`. Make sure to write this before the `:`.

Hint 2

Use the `=` assignment operator to update the value of the `employee_id` variable. Update the value of `employee_id` to the concatenation of `"E"` with the variable's current value. The `"E"` should appear to the left of the current value.

Hint 3

Use the `print()` function to display `employee_id` after the update.

1.6 Task 4

Now you'll move on to the next part of your task. Imagine that the characters in a device ID convey technical information about the device. You'll need to extract characters in specific positions from the device ID. Start off by extracting the fourth character.

The variable `device_id` represents a device ID containing alphanumeric characters; it's already stored as a string.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[12]:
```
# Assign `device_id` to a string that contains alphanumeric characters

device_id = "r262c36"
```

```
# Extract the fourth character in `device_id` and display it

print(device_id[3])
```

```
2
```

Hint 1

Use a pair of square brackets, passing in the appropriate index value, in order to extract the fourth character in `device_id`.

Hint 2

In Python, index values start at `0`.

Hint 3

Given that index values start at `0` in Python, an index value of `3` corresponds to the fourth character in a sequence.

1.7 Task 5

Now you will also need to extract the first through the third characters in the device ID. So take a slice of the device ID. You can achieve this using bracket notation in Python. Then, display the slice to examine the result.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[13]:
```
# Assign `device_id` to a string that contains alphanumeric characters

device_id = "r262c36"

# Extract the first through the third characters in `device_id` and display the␣
 ↪result

print(device_id[0:3])
```

```
r26
```

Hint 1

Use a pair of square brackets, passing in the appropriate index values, in order to extract the first through the third characters in `device_id`.

Inside the square brackets, use a `:` to separate the first index value (the starting index value) and second index value (ending index value).

Hint 2

Keep in mind that the second index value passed into bracket notation is exclusive. In other words, the index value passed into the square brackets after the `:` is not included when the string is sliced. The resulting slice will not include the character at that index.

Hint 3

Recall that the second index value passed into bracket notation is exclusive and indexing in Python starts at `0`. The first index value should be `0` and the second index value should be `3`, in order to extract the first through the third characters in `device_id`.

1.8 Task 6

You'll now proceed to the last part of your task. This involves extracting components of a URL.

You'll work with string indices to display various components of a URL that's stored in the URL variable. First, you'll extract and display the protocol of the URL and the `://` characters that follow it using string slicing. Consider that the protocol is in the secure format of `https` when determining the indices for your slice.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[14]:
```
# Assign `url` to a specific URL

url = "https://exampleURL1.com"

# Extract the protocol of `url` along with the syntax following it, display the␣
 ↪result

print(url[0:8])
```

```
https://
```

Hint 1

Note that `https://` is eight characters long.

Hint 2

Use a pair of square brackets to slice the string stored in `url`, passing in two index values separated by `:`. Keep in mind that the second index value is exclusive and that indexing in Python starts at `0`.

Hint 3

Use the `print()` function to display the slice.

1.9 Task 7

Later in this lab, you'll extract the domain extension. To prepare for this, use the `.index()` method to identify the index where the domain extension `.com` is located in the given URL.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[15]:
```
# Assign `url` to a specific URL

url = "https://exampleURL1.com"

# Display the index where the domain extension ".com" is located in `url`

print(url.index(".com"))
```

```
19
```

Hint 1

Apply the `.index()` method to `url` in order to get the appropriate index. The `.index()` method takes in a substring, and if that substring is located in the original string, it returns the index where that substring starts to occur in the original string.

Hint 2

Call `url.index()`, and inside the parentheses, pass in the targeted domain extension as a string.

1.10 Task 8

It's a good idea to save important data in variables when programming. This allows for quick and easy tracking and reuse of information.

Store the output of the `.index()` method in a variable called `ind`, which is short for index. This index represents the position where the domain extension `".com"` starts in the `url`. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell. Note that running this cell will not produce an output.

[16]:
```
# Assign `url` to a specific URL

url = "https://exampleURL1.com"

# Assign `ind` to the output of applying `.index()` to `url` in order to␣
↪extract the starting index of ".com" in `url`

ind = url.index(".com")
```

Hint 1

To assign the extracted index to the `ind` variable, use the `.index()` method to the right of the `=` assignment operator.

Hint 2

Call `url.index()`, and inside the parantheses, pass in the targeted domain extension as a string. Be sure to place this to the right of the `=` assignment operator, so that the result is assigned to `ind`.

1.11 Task 9

You can use string slicing to also extract the domain extension of a URL. To do so, you can create a slice. The starting index should be the `ind` variable. This contains the index where the domain extension begins. The ending index should be `ind + 4` (since `".com"` is four characters long). Sometimes, like in this situation, it's easier to express the ending index in relation to the starting index. Examine the following code, run it as is, and observe the output.

[17]:
```
# Assign `url` to a specific URL

url = "https://exampleURL1.com"

# Assign `ind` to the output of applying `.index()` to `url` in order to␣
 ↪extract the starting index of ".com" in `url`

ind = url.index(".com")

# Extract the domain extension in `url` and display it

print(url[ind:ind+4])
```

```
.com
```

Question 2 What does this code output and why?

After inputing the code url[idnd:ind+4], this tells the machine which ones to print which is the last parts of the ends of ".com".. Once it prints out, it shows the domain name .com by first saving the starting position of the domain name in the ind variable and then extracting 4 consecutive characters from url starting from the saved position.

1.12 Task 10

Finally, extract the website name from the given URL using string slicing and the `ind` variable that you defined earlier. In the given URL, the website name is `"exampleURL1"`. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[18]:
```
# Assign `url` to a specific URL

url = "https://exampleURL1.com"

# Assign `ind` to the output of applying `.index()` to `url` in order to␣
 ↪extract the starting index of ".com" in `url`

ind = url.index(".com")

# Extract the website name in `url` and display it

```

```
print(url[8:ind])
```

```
exampleURL1
```

Hint 1

In order to extract the website name in the given URL, use a pair of square brackets to create a slice of `url`, passing in a start index and an ending index, separating the two with `:`.

Hint 2

The starting index should be set to `8`, since this is the position after the protocol and the `://` syntax ends and where the website name begins. The ending index should be set to the position where the `.com` domain name begins.

1.13 Conclusion

What are your key takeaways from this lab?

I believe strings are essential for storing important data, particularly in security-related contexts like device IDs and URLs. String manipulation techniques, such as concatenation and slicing, are powerful tools for combining and extracting specific parts of a string. In Python, several built-in functions and methods make working with strings easier. For example, the `type()` function reveals the data type of an object, while `str()` converts other data types into strings. The `len()` function helps determine the number of characters in a string, and `.index()` locates the first occurrence of a substring within a string. These tools are vital for analysts dealing with string values or converting data to string format.

[ ]: