Activity_Work with strings in Python

April 11, 2024

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.

1.2 Scenario

I'm working as a security analyst, and am 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

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

[1]:
```
# 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(4186)

# Display the data type of `employee_id` now

```

```
print(type(employee_id))
```

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

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

1.4 Task

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

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

[4]:
```
# 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.
```

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

1.5 Task

In this task, I'll build upon the previous code. If an employee ID is only four digits, I'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.

I will 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.

[12]:
```
# 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 = employee_id + "E"

# Display the `employee_id` after the update

print(employee_id)
```

```
4186
4186E
```

1.6 Task

I now imagine that the characters in a device ID convey technical information about the device. I'll need to extract characters in specific positions from the device ID. The variable `device_id` represents a device ID containing alphanumeric characters.

[14]:
```
# 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[4])
```

```
c
```

1.7 Task

Now I will also need to extract the first through the third characters in the device ID. So I'll take a slice of the device ID using bracket notation in Python.

[16]:
```
# 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
```

I used 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, I used a `:` to separate the first index value (the starting index value) and second index value (ending index value).

1.8 Task

This next task involves extracting components of a URL.

I'll work with string indices to display various components of a URL that's stored in the URL variable. First, I'll extract and display the protocol of the URL and the `://` characters that follow it using string slicing.

[21]:
```
# 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://
```

1.9 Task 7

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

[25]:
```
# 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
```

Iapplied 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.

1.10 Task

It's a good idea to save important data in variables when programming. This allows for quick and easy tracking and reuse of information. So, I'll store the output of the `.index()` method in a variable called `ind`. This index represents the position where the domain extension `".com"` starts in the `url`.

[27]:
```
# 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")
```

1.11 Task

I can use string slicing to also extract the domain extension of a URL. To do so, I'll 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).

[28]:
```
# 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
```

[This code outputs 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

Finally, I'll extract the website name from the given URL using string slicing and the `ind` variable that I defined earlier. In the given URL, the website name is `"exampleURL1"`.

[29]:
```
# 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
```

1.13 Conclusion

What are your key takeaways from this lab?

[Strings are instrumental in storing important, security-related data, such as device IDs and URLs. String concatenation allows one to easily combine information in a string with the information stored in another string. String slicing is a powerful technique that enables one to extract any subsection of a string. Python has many functions and methods that help analysts work with string values, as well as data that they want to convert to string format. The type() function returns the data type of its input. The str() function converts the input object into a string. F The len() function returns the number of elements in an object. The .index() method finds the first occurrence of the input in a string and returns its location. It provides the index where the substring begins.]