Activity_Define and call a function

April 9, 2024

1 Activity: Define and call a function

1.1 Introduction

As a security analyst, when writing out Python code to automate a certain task, one often finds themself needing to reuse the same block of code more than once. This is why functions are important, they allow us to automate repetitive parts of the code. One can call that function whenever one needs the computer to execute those steps. Python not only has built-in functions that have already been defined, but also provides the tools for users to define their own functions. Security analysts often define and call functions in Python to automate series of tasks.

1.2 Scenario

Writing functions in Python is a useful skill as a security analyst. In this lab, I'll define and a call a function that displays an alert about a potential security issue. Also, I'll work with a list of employee usernames, creating a function that converts the list into one string(string concatenation).

1.3 Task

The following code cell contains a user-defined function named `alert()`.

```
[ ]: # Define a function named `alert()`

     def alert():
         print("Potential security issue. Investigate further.")
```

To create a funtion we use the 'def' command then the function which ends in parenthesis and a colon. `def alert():` defines a function named alert. This means you're creating a reusable block of code that can be called later in the program to execute its instructions. The body of the function is `print("Potential security issue. Investigate further.")` It contains the code that will be executed when the `alert()` function is called. In this case, the print statement displays the message "Potential security issue. Investigate further." to the console

1.4 Task

I will now call the `alert()` function that was defined earlier.

[1]:
```python
# Define a function named `alert()`

def alert():
    print("Potential security issue. Investigate further.")

# Call the `alert()` function

alert()
```

```
Potential security issue. Investigate further.
```

[This makes us able to call that function anywhere in the code and have the same output hence removing the need to repeat the statement elsewhere.]

1.5 Task

Functions can include other components that you've already worked with. The following code cell contains a variation of the `alert()` function that now uses a `for` loop to display the alert message multiple times.

[2]:
```python
# Define a function named `alert()`

def alert():
    for i in range(3):
        print("Potential security issue. Investigate further.")

# Call the `alert()` function
alert()
```

```
Potential security issue. Investigate further.
Potential security issue. Investigate further.
Potential security issue. Investigate further.
```

1.6 Task

In the next part I'm working with a list of approved usernames, representing users who can enter a system. I'll be developing a function that helps convert the list of approved usernames into one big string. Structuring this data differently enables one to work with it in different ways. For example, structuring the usernames as a list allows one to easily add or remove a username from it. In contrast, structuring it as a string allows me to easily place its contents into a text file.

[3]:
```python
# Define a function named `list_to_string()`

def list_to_string():
```

```
  File "<ipython-input-3-f359e12ed06d>", line 3
```

```
    def list_to_string():
                         ^
SyntaxError: unexpected EOF while parsing
```

1.7 Task

In the following code cell, there is a list of approved usernames, stored in a variable named `username_list`. The task is to complete the body of the `list_to_string()` function. To complete the function body, I'll write a loop that iterates through the elements of the `username_list` and displays each element. Then, call the function and run the cell.

```
[5]: # Define a function named `list_to_string()`

def list_to_string():

  # Store the list of approved usernames in a variable named `username_list`

  username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab",␣
 ↪"gesparza", "alevitsk", "wjaffrey"]

  # Write a for loop that iterates through the elements of `username_list` and␣
 ↪displays each element

  for i in username_list:
    print(i)

# Call the `list_to_string()` function

list_to_string()
```

```
elarson
bmoreno
tshah
sgilmore
eraab
gesparza
alevitsk
wjaffrey
```

1.8 Task

String concatenation is a powerful concept in coding. It allows one to combine multiple strings together to form one large string, using the addition operator (+). Sometimes analysts need to merge individual pieces of data into a single string value. In this task, I'll use string concatenation to modify how the `list_to_string()` function is defined.

In the following code cell, there is a variable named `sum_variable` that initially contains an empty string. The task is to use string concatenation to combine the usernames from the `username_list` and store the result in `sum_variable`.

In each iteration of the `for` loop, I'll add the current element of `username_list` to `sum_variable`. At the end of the function definition, I'll write a `print()` statement to display the value of `sum_variable` at that stage of the process. Then, run the cell to call the `list_to_string()` function.

[10]:
```
# Define a function named `list_to_string()`

def list_to_string():

  # Store the list of approved usernames in a variable named `username_list`

  username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab",␣
 ↪"gesparza", "alevitsk", "wjaffrey"]

  # Assign `sum_variable` to an empty string

  sum_variable = ""

  # Write a for loop that iterates through the elements of `username_list` and␣
 ↪displays each element

  for i in username_list:
    sum_variable = sum_variable + i

  # Display the value of `sum_variable`

  print(sum_variable)

# Call the `list_to_string()` function

list_to_string()
```

```
elarsonbmorenotshahsgilmoreeraabgesparzaalevitskwjaffrey
```

Inside the `for` loop, there is a line that updates the `sum_variable` in each iteration. The loop variable `i` represents each element of `username_list`.

1.9 Task

In this final task, I'll modify the code written previously to improve the readability of the output.

This time, in the definition of the `list_to_string()` function, I'll add a comma and a space (`",`

") after each username. This will prevent all the usernames from running into each other in the output. Adding a comma helps clearly separate one username from the next in the output. Adding a space following the comma as an additional separator between one username and the next makes it easier to read the output.

[17]:
```
# Define a function named `list_to_string()`

def list_to_string():

  # Store the list of approved usernames in a variable named `username_list`

  username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab",␣
 ↪"gesparza", "alevitsk", "wjaffrey"]

  # Assign `sum_variable` to an empty string

  sum_variable = ""

  # Write a for loop that iterates through the elements of `username_list` and␣
 ↪displays each element

  for i in username_list:
    sum_variable = sum_variable + i + ", "

  # Display the value of `sum_variable`

  print(sum_variable)

# Call the `list_to_string()` function

list_to_string()
```

```
elarson, bmoreno, tshah, sgilmore, eraab, gesparza, alevitsk, wjaffrey,
```

1.10 Conclusion

What are your key takeaways from this lab?

[Python allows one to define and call functions. The main components of a function definition header include the function header and the function body. The function header includes the `def` keyword, followed by the name of the function, followed by parantheses, followed by a colon. The function body includes an indented block of code that instructs the computer on what to do when the function is called.String concatenation involves using the addition operator (`+`) to combine multiple strings together. One use case for string concatenation is combining the strings from a list into one large string.]