Activity_Define and call a function

February 20, 2024

1 Activity: Define and call a function

1.1 Introduction

As a security analyst, when you're writing out Python code to automate a certain task, you'll often find yourself needing to reuse the same block of code more than once. This is why functions are important. You can call that function whenever you need 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.

In this lab, you'll practice defining and calling functions in Python.

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 this 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

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

1.3 Task 1

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

For this task, analyze the function definition, and make note of your observations.

You won't need to run the cell in order to answer the question that follows. But if you do run the cell, note that it will not produce an output because the function is just being defined here.

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

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

Hint 1

When analyzing the function definition, make sure to observe the function body, which is the indented block of code after the function header. The function body tells you what the function does.

Question 1 Summarize what the user-defined function above does in your own words. Think about what the output would be if this function were called.

The user-defined function `alert()` serves the purpose of providing a standardized message for alerting about a potential security issue. When this function is called, it prints the message "Potential security issue. Investigate further." to the console. It acts as a simple alert mechanism, prompting the user or developer to investigate the mentioned security issue in more detail.

1.4 Task 2

For this task, call the `alert()` function that was defined earlier and analyze the output.

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

[4]:
```
# Define a function named `alert()`
def alert():
    print("Potential security issue. Investigate further.")

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

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

Hint 1

To call the function, write `alert()` after the function definition. Note that the function can be called only after it's defined.

Question 2 What are the advantages of placing this code in a function rather than running it directly?

There are many advantages to placing code inside a function rather than running it directly:

1. **Reusability:** By encapsulating the code in a function, you can reuse it at different points in your program. In this case, you can call the `alert()` function whenever you need to display the security alert message.
2. **Modularity:** Functions promote modularity in your code. Each function can represent a specific task or functionality, making the code easier to understand, maintain, and update. If you decide to change the alert message or its behavior, you only need to modify the function.
3. **Readability:** Functions improve code readability by providing a clear structure. Instead of having the alert message code scattered throughout your program, placing it in a function makes the main part of your code cleaner and more organized.
4. **Encapsulation:** Functions encapsulate a specific piece of functionality, keeping the implementation details hidden from the rest of the program. This can help prevent unintended modifications to the code and reduce the chances of errors.
5. **Testing and Debugging:** Code inside functions is easier to test and debug. You can focus on one function at a time, making it simpler to identify and fix issues. This becomes particularly beneficial as your codebase grows.
6. **Parameterization:** Functions can accept parameters, allowing you to customize their behavior based on input values. In this case, you might enhance the `alert()` function to accept a custom message or other parameters.
7. **Namespace Management:** Functions have their own local namespace, preventing potential naming conflicts with variables in the global scope. This helps avoid unintended variable overwrites and improves code reliability.

Overall, placing code in functions adheres to good coding practices, promoting code organization, maintainability, and reusability. It facilitates the development and management of more complex programs to use for future project tasks.

1.5 Task 3

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.

For this task, call the new `alert()` function and observe the output.

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

[5]:
```
# 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.
```

Hint 1

To call the function, write `alert()` after the function definition. Note that the function can be called only after it's defined.

Question 3 How does the output above compare to the output from calling the previous version of the `alert()` function? How are the two definitions of the function different?

The previous version of alert() function displayed the security alert message once, while the updated version with a for loop repeats the alert message three times. The key difference is that the updated version introduces a loop for a more flexible and customizable way to display the alert multiple times.

1.6 Task 4

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

For this task, start defining a function named `list_to_string()`. Write the function header.

Be sure to replace the `### YOUR CODE HERE ###` with your own code. Note that running this cell will produce an error since this cell will just contain the function header; you'll write the function body and complete the function definition in a later task.

[7]:
```
def list_to_string(username_list):
    # Your code here
    string_representation = ", ".join(username_list)
    return string_representation
```

Hint 1

To write the function header, start with the `def` keyword, followed by the name of the function, parentheses, and a colon.

1.7 Task 5

Now you'll begin to develop the body of the `list_to_string()` function.

In the following code cell, you're provided a list of approved usernames, stored in a variable named `username_list`. Your task is to complete the body of the `list_to_string()` function. Recall that the body of a function must be indented. To complete the function body, write a loop that iterates

through the elements of the `username_list` and displays each element. Then, call the function and run the cell to observe what happens.

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

[12]:
```
# 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 username in username_list:
        print(username)

# Call the `list_to_string()` function
list_to_string()
```

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

Hint 1

The `for` loop in the body of the `list_to_string()` function must iterate through the elements of `username_list`. So, use the `username_list` variable to complete the `for` loop condition.

Hint 2

In each iteration of the `for` loop, an element of `username_list` should be displayed. The loop variable `i` represents each element of `username_list`. To complete the `print()` statement inside the `for` loop, passi to the `print()` function call.

Hint 3

To call the function, write `list_to_string()` after the function definition. Recall that the function can be called only after it's defined.

Question 4 What do you observe from the output above?

The output displays each username from the `username_list` on a new line. Each iteration of the for loop prints one username, resulting in a vertical list of usernames. The function successfully iterates through the elements of `username_list` and displays each element in the specified format.

1.8 Task 6

String concatenation is a powerful concept in coding. It allows you 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, you'll use string concatenation to modify how the `list_to_string()` function is defined.

In the following code cell, you're provided a variable named `sum_variable` that initially contains an empty string. Your 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, add the current element of `username_list` to `sum_variable`. At the end of the function definition, 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 and examine its output.

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

[13]:
```
# 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`
    # In each iteration, add the current element to `sum_variable` using string␣
 ↪concatenation
    for username in username_list:
        sum_variable = sum_variable + username

        # Display the value of `sum_variable` at each stage
        print(sum_variable)

# Call the `list_to_string()` function
list_to_string()
```

```
elarson
elarsonbmoreno
elarsonbmorenotshah
elarsonbmorenotshahsgilmore
elarsonbmorenotshahsgilmoreeraab
elarsonbmorenotshahsgilmoreeraabgesparza
elarsonbmorenotshahsgilmoreeraabgesparzaalevitsk
elarsonbmorenotshahsgilmoreeraabgesparzaalevitskwjaffrey
```

Hint 1

Inside the `for` loop, complete the line that updates the `sum_variable` in each iteration. The loop variable `i` represents each element of `username_list`. Since you need to add the current element to the current value of `sum_variable`, place `i` after the addition operator `(+)`.

Hint 2

Use the `print()` function to display the value of `sum_variable`. Make sure to pass in `sum_variable` to the call to `print()`.

Question 5 What do you observe from the output above?

[Double-click to edit this markdown cell and write something here.]

1.9 Task 7

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

This time, in the definition of the `list_to_string()` function, 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. Then, call the function and run the cell to observe the output.

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

[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`
    # In each iteration, add the current element and a comma with space to␣
 ↪`sum_variable`
    for username in username_list:
        sum_variable = sum_variable + username + ", "

    # 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,
```

Hint 1

Inside the `for` loop, complete the line that updates the `sum_variable` in each iteration. The loop variable `i` represents each element of `username_list`. After the current element is added to the current value of `sum_variable`, add a string that contains a comma followed by a space.

To complete this step, place `", "` after the last addition operator (`+`).

Hint 2

To call the function, write `list_to_string()` after the function definition. Note that the function can be called only after it's defined.

Question 6 What do you notice about the output from the function call this time?

The output displays the gradual accumulation of usernames in the `sum_variable` string. In each iteration of the for loop, the current username is added to `sum_variable` using string concatenation. The print statement inside the loop shows the value of `sum_variable` at each stage of the process. This demonstrates how string concatenation allows the usernames to be progressively combined into a single string in the block of code.

1.10 Conclusion

What are your key takeaways from this lab?

The key takeaways from this lab i learned were:

1. **Functions:** Understanding how to define and call functions is fundamental in programming. Functions provide a way to encapsulate reusable pieces of code, promoting modularity and maintainability.
2. **Loops:** The lab involved working with loops, specifically `for` loops, to iterate through elements in a list. Loops are essential for repetitive tasks and allow efficient handling of collections of data.
3. **String Concatenation:** String concatenation is a powerful concept for combining multiple strings into one. It involves using the `+` operator to concatenate strings, enabling the creation of larger strings from smaller components.
4. **Code Structure:** Proper indentation and organization of code contribute to readability and maintainability. Consistent indentation helps in understanding the structure of loops, functions, and conditional blocks.
5. **Parameterization:** Functions can take parameters, allowing them to be more flexible and accept input values. In this lab, a function parameter was used to pass a list of usernames to the function.
6. **Printing Output:** The `print()` function is valuable for displaying information during program execution. It was used to output messages and intermediate results for observation.
7. **Variable Manipulation:** The lab involved working with variables to store and manipulate data. String variables, such as `sum_variable`, were used to accumulate and display information.

8. **Problem-Solving Skills:** The lab tasks required logical thinking and problem-solving skills, particularly in structuring code to achieve specific goals.

Overall, the lab provided hands-on experience in Python programming, emphasizing functions, loops, string manipulation, and code organization to manage code fundamentals easier.

[ ]:

[ ]:

[ ]: