Activity_Define and call a function

October 27, 2025

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.

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

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

Hint 1

Take a look at the function above. `def alert():` defines a function named alert. This means you're creating a reusable block of code that can be called later in your 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 consol

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.

Based on the notes and the lab example, the alert() prints the string "Potential security issue. Investigate further." This is used to notify security issues in a system like if this function were called, the output would show Potential security issue. Also note that when a string is displayed, the quotes around the string do not appear in the output….

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.

[18]:
```
# 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?

Based on my past experinces and notes, if we place the code in a function, it helps me reuse the code by calling the alert() function, if I need it to print messages about a potential security issue and further investigation…. The second option would be to write out that print() statement every time, which would be tedious.

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.

[19]:
```
# 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

Task 3 is similar to Task 2 here. The difference is that a `for` loop is used to display the alert message multiple times.

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?

In the past one, it does not have the for loop to repeat the print statement. This one, the output shows "Potential security issue. Investigate further." three times due to the for loop where the messages appears eveyr new line. This loop iterates over a range of numbers (specified by range(3)) and executes a print() statement in each iteration. The only thing was similar was the print mehtod, not the for loop.

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.

[20]:
```
# Define a function named `list_to_string()`

def list_to_string():
```

```
  File "<ipython-input-20-f359e12ed06d>", line 3
    def list_to_string():
                         ^
SyntaxError: unexpected EOF while parsing
```

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.

[ ]:
```
# 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()
```

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?

Since it uses a for loop and printing i parts, it will print each element in the list. for i in username_list means you are accessing the list and then if you call i it means you want it to print out each element in the list. This will print out all the names in the list.

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.

```
[ ]: # 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()
```

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?

When i is placed into the sum_variable and added, it means it is concated which makes the output shows all the elements from username_list merged together in one line, but the output is difficult to read since I have a hard time knowing where the username ends and the next begins.

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.

```
[ ]: # 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()
```

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?

Based on this example, once the "," is added/concantated to to sum_variable, and then the function list_to_string() is called, the output shows all the elements from username_list in one line, but with a comma and a space after each username. I found this useful as preivously it was not easy to read it all mushed up together. And now, this format is much easier to read to know which one is the username from the next.

1.10 Conclusion

What are your key takeaways from this lab?

In this lab, I learned how to define and use functions in Python using the `def` keyword. Functions allow me to organize code into reusable blocks. For example, I created a function `alert()` that prints a message when called. To call a function, I simply use its name followed by parentheses, like this: `alert()`. I also worked with string concatenation, where I used the `+` operator to combine strings. For instance, I concatenated usernames from a list into one large string by adding each username to a variable inside a loop. I used a `for` loop to iterate over the list of usernames, which helped me process each item one by one. The benefit of using a `for` loop in this case is that it automates repetitive tasks, saving time and reducing errors. Overall, using functions, string concatenation, and loops made my code more efficient, organized, and easy to maintain.

[ ]: