Activity_Create more functions

February 20, 2024

1 Activity: Create more functions

1.1 Introduction

Built-in functions are functions that exist within Python and can be called directly. They help analysts efficiently complete tasks. Python also supports user-defined functions. These are functions that analysts write for their specific needs.

For example, patterns in login attempts could reveal suspicious activity. Python functions can help analysts work efficiently with lists of login attempts. Both built-in functions and user-defined functions in Python can help security analysts analyze login attempts.

In this lab, you'll use built-in functions to work with a list of failed login attempts per month to prepare it for further analysis, and you'll define a function that compares the user's login attempts for the current day to their average number of login attempts.

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

In your work as a security analyst, you're responsible for working with a list that contains the number of failed attempts that occurred each month. You'll identify any patterns that might indicate malicious activity. You're also responsible for defining a function that compares the logins for the current day to an average and improving it by adding a `return` statement.

1.3 Task 1

In your work as an analyst, imagine that you're provided a list of the number of failed login attempts per month, as follows:

`119`, `101`, `99`, `91`, `92`, `105`, `108`, `85`, `88`, `90`, `264`, and `223`.

This list is organized in chronological order of months (January, February, March, April, May, June, July, August, September, October, November, and December).

This list is stored in a variable named `failed_login_list`.

In this task, use a built-in Python function to order the list. You'll pass the call to the function that sorts the list directly into the `print()` function. This will allow you to display and examine the result.

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

[1]:
```
failed_login_list = [119, 101, 99, 91, 92, 105, 108, 85, 88, 90, 264, 223]

# Sort `failed_login_list` in ascending numerical order and display the result
print(sorted(failed_login_list))
```

```
[85, 88, 90, 91, 92, 99, 101, 105, 108, 119, 223, 264]
```

Hint 1

To order the `failed_login_list` in ascending numerical order, use the `sorted()` function.

This is a built-in Python function that takes in a list, sorts its components, and returns the result.

Hint 2

To order the `failed_login_list` in ascending numerical order, call the `sorted()` function and pass in `failed_login_list`.

To display the result, make sure to place the call to `sorted()` inside the `print()` statement.

Question 1 What do you observe from the output above? Do you notice any outlying numbers that indicate an increase in the failed number of login attempts?

Looking at the sorted output, it appears that the majority of the values are very low, with the exception of two higher values: 223 and 264. These two values are larger than the rest, suggesting potential outliers or months with a significantly increased number of failed login attempts compared to the others. Further analysis may be needed to understand the reasons behind these higher values and whether they represent any security concerns or anomalies.

1.4 Task 2

Now, you'll want to isolate the highest number of failed login attempts so you can later investigate information about the month when that highest value occurred.

You'll use the function that returns the largest numeric element from a list. Then, you'll pass this function into the `print()` function to display the result. This will allow you to determine which month to investigate further.

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

[2]:
```
failed_login_list = [119, 101, 99, 91, 92, 105, 108, 85, 88, 90, 264, 223]

# Determine the highest number of failed login attempts from␣
 ↪`failed_login_list` and display the result
print(max(failed_login_list))
```

```
264
```

Hint 1

To determine the highest number of failed login attempts from `failed_login_list`, use the `max()` function.

This is a built-in Python function that takes in a sequence, identifies the maximum value from the sequence and returns the result.

Hint 2

To determine the highest number of failed login attempts from `failed_login_list`, call the `max()` function and pass in `failed_login_list`.

To display the result, make sure to place the call to `max()` inside the `print()` statement.

Question 2 What do you observe from the output above?

The output shows the highest number of failed login attempts, which is 264. This shows that the month corresponding to this value experienced the most significant increase in failed login attempts compared to the other months in the provided list. Further investigation into the circumstances surrounding this particular month may be necessary to address any potential security concerns or issues.

1.5 Task 3

In your work as an analyst, you'll first define a function that displays a message about how many login attempts a user has made that day.

In this task, define a function named `analyze_logins()` that takes in two parameters, `username` and `current_day_logins`. Every time this function is called, it should display a message about the number of login attempts the user has made that day.

Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell. Note that the code cell will contain only a function definition, so running it will not produce an output.

[5]:
```
def analyze_logins(username, current_day_logins):
    """
    Display a message about the number of login attempts the user has made that␣
 ↪day.

    Parameters:
    - username (str): The username of the user.
    - current_day_logins (int): The number of login attempts made by the user␣
 ↪on the current day.
    """
    print("Current day login total for", username, "is", current_day_logins)

# Example usage:
username_example = "JohnDoe"
current_day_logins_example = 3
analyze_logins(username_example, current_day_logins_example)
```

```
Current day login total for JohnDoe is 3
```

Hint 1

To write a function header in Python, start with the `def` keyword, followed by the function name and then parantheses.

Hint 2

In Python, to define a function that takes in parameters, place the names of the parameters inside of the parantheses at the function header, and use a `,` between each parameter and the next.

Hint 3

To define a function named `analyze_logins()` that takes in two parameters, `username` and `current_day_logins`, start with the `def` keyword, followed by `analyze_logins()`, and write `username, current_day_logins` inside the parantheses. Be sure to write this code before the `:`.

1.6 Task 4

Now that you've defined the `analyze_logins()` function, call it to test out how it behaves.

Call `analyze_logins()` with the arguments `"ejones"` and `9`.

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

[6]:
```
# Define a function named `analyze_logins()` that takes in two parameters,␣
 ↪`username` and `current_day_logins`

def analyze_logins(username, current_day_logins):
    # Display a message about how many login attempts the user has made that day
    print("Current day login total for", username, "is", current_day_logins)
```

```
# Call `analyze_logins()` with the arguments "ejones" and 9
analyze_logins("ejones", 9)
```

```
Current day login total for ejones is 9
```

Hint 1

To call the `analyze_logins()` function after it's defined, write `analyze_logins()`. Then make sure to place the arguments `"ejones"` and `9` inside the parantheses.

Hint 2

The function call should be written as `analyze_logins("ejones", 9)`.

Question 3 What does this function display? Would the output vary for different users?

The analyze_logins() function, as defined in the provided code, displays a message about the number of login attempts made by a user on the current day. The output includes the username and the corresponding number of login attempts.

For the specific call in the code (analyze_logins("ejones", 9)), the output would be:"Current day login total for ejones is 9"

The output would indeed vary for different users because the function takes the username as a parameter. If you call the function with different usernames and login attempt values, the output will reflect the specific user and the number of login attempts for that user on the current day.

1.7 Task 5

Now, you'll need to expand this function so that it also provides the average number of login attempts made by the user on that day. Doing this will require incorporating a third parameter into the function definition.

In this task, add a parameter called `average_day_logins`. The code will use this parameter to display an additional message. The additional message will convey the average login attemps made by the user on that day. Then, call the function with the same first and second arguments as used in Task 4 and a third argument of `3`.

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

[8]:
```
# Define a function named `analyze_logins()` that takes in three parameters:␣
↪`username`, `current_day_logins`, and `average_day_logins`

def analyze_logins(username, current_day_logins, average_day_logins):
    # Display a message about how many login attempts the user has made that day
    print("Current day login total for", username, "is", current_day_logins)
```

```
    # Display a message about the average login attempts made by the user on␣
 ↪that day
    print("Average logins per day for", username, "is", average_day_logins)

# Call `analyze_logins()` with the arguments "ejones", 9, and 3
analyze_logins("ejones", 9, 3)
```

```
Current day login total for ejones is 9
Average logins per day for ejones is 3
```

Hint 1

In Python, to define a function that takes in parameters, place the names of the parameter inside the parantheses at the function header, with a `,` between each parameter and the next.

Hint 2

You need to define a function named `analyze_logins()` that takes in three parameters, `username`, `current_day_logins`, and `average_day_logins`. So you'll need to write `username, current_day_logins, average_day_logins` inside the parantheses.

Hint 3

To call the `analyze_logins()` function after it's defined, write `analyze_logins()`. Then make sure to place the arguments `"ejones"`, `9`, and `3` inside the parantheses.

1.8 Task 6

In this task, you'll further expand the function. Include a calculation to get the ratio of the logins made on the current day to the logins made on an average day. Store this in a new variable named `login_ratio`. The function displays an additional message that uses this variable.

Note that if `average_day_logins` is equal to `0`, then dividing `current_day_logins` by `average_day_logins` will cause an error. Due to the error, Python will display the following message: `ZeroDivisionError: division by zero`. For this activity, assume that all users will have logged in at least once before. This means that their `average_day_logins` will be greater than `0`, and the function will not involve dividing by zero.

After defining the function, call the function with the same arguments that you used in the previous task.

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

[9]:
```
# Define a function named `analyze_logins()` that takes in three parameters:␣
 ↪`username`, `current_day_logins`, and `average_day_logins`

def analyze_logins(username, current_day_logins, average_day_logins):
    # Display a message about how many login attempts the user has made that day
    print("Current day login total for", username, "is", current_day_logins)
```

```
    # Display a message about the average login attempts made by the user on␣
↪that day
    print("Average logins per day for", username, "is", average_day_logins)

    # Calculate the ratio of the logins made on the current day to the logins␣
↪made on an average day, storing in a variable named `login_ratio`
    login_ratio = current_day_logins / average_day_logins

    # Display a message about the ratio
    print(username, "logged in", login_ratio, "times as much as they do on an␣
↪average day.")

# Call `analyze_logins()` with the arguments "ejones", 9, and 3
analyze_logins("ejones", 9, 3)
```

```
Current day login total for ejones is 9
Average logins per day for ejones is 3
ejones logged in 3.0 times as much as they do on an average day.
```

Hint 1

To calculate the ratio of the logins made on the current day to the logins made on an average day, divide `current_day_logins` by `average_day_logins`.

Assign a variable named `login_ratio` to the result of this calculation, using the `=` assignment operator.

Hint 2

To assign a variable named `login_ratio` to the result of the calculation, use the `=` assignment operator. Write `login_ratio` to the left of `=`, and place the calculation to the right of `=`.

Hint 3

Call the updated `analyze_logins()` function and pass in `"ejones"`, `9`, and `3` as the three arguments, in that order.

Question 4 What does this version of the `analyze_logins()` function display? Would the output vary for different users?

This version of the `analyze_logins()` function displays three messages which are:

1. The current day login total for the given username.
2. The average logins per day for the given username.
3. A message about the login ratio, indicating how many times the user logged in compared to an average day.

The output would vary for different users because the function takes the `username`, `current_day_logins`, and `average_day_logins` as parameters. Calling the function with different usernames and corresponding login attempt values will result in personalized output for each user, reflecting their specific login activity on the current day and the average login attempts on other days in the code.

1.9 Task 7

You'll continue working with the `analyze_logins()` function and add a return statement to it. Return statements allow you to send information back to the function call.

In this task, use the `return` keyword to output the `login_ratio` from the function, so that it can be used later in your work.

You'll call the function with the same arguments used in the previous task and store the output from the function call in a variable named `login_analysis`. You'll then use a `print()` statement to display the saved information.

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

[10]:
```
# Define a function named `analyze_logins()` that takes in three parameters:␣
↪`username`, `current_day_logins`, and `average_day_logins`

def analyze_logins(username, current_day_logins, average_day_logins):
    # Display a message about how many login attempts the user has made that day
    print("Current day login total for", username, "is", current_day_logins)

    # Display a message about the average login attempts made by the user on␣
↪that day
    print("Average logins per day for", username, "is", average_day_logins)

    # Calculate the ratio of the logins made on the current day to the logins␣
↪made on an average day, storing in a variable named `login_ratio`
    login_ratio = current_day_logins / average_day_logins

    # Return the ratio
    return login_ratio

# Call `analyze_logins()` and store the output in a variable named␣
↪`login_analysis`
login_analysis = analyze_logins("ejones", 9, 3)

# Display a message about the `login_analysis`
print("ejones logged in", login_analysis, "times as much as they do on an␣
↪average day.")
```

```
Current day login total for ejones is 9
Average logins per day for ejones is 3
ejones logged in 3.0 times as much as they do on an average day.
```

Hint 1

When defining the `analyze_logins()` function this time, place the `return` keyword in front of the output that you want the function to return.

Hint 2

When defining the `analyze_logins()` function this time, write `return` in front of `login_ratio`. (Do not place parentheses after the `return` keyword. It is not a function.)

Question 5 How does this version of the `analyze_logins()` function compare to the previous versions?

This version of the `analyze_logins()` function is an extension of the previous versions. The key difference is the addition of the return statement that returns the calculated `login_ratio`. This allows the function to produce a value that can be stored and used outside the function.

In the previous versions, the function displayed information using print statements but did not provide a value that could be used in subsequent calculations or operations. With the addition of the return statement in this version, the function becomes more versatile as it can now be used in a broader context, and the calculated `login_ratio` can be accessed and utilized in other parts of the code to run functions easier.

1.10 Task 8

In this task, you'll use the value of `login_analysis` in a conditional statement. When the value of `login_analysis` is greater than or equal to `3`, then the login activity will require further investigation, and an alert will be displayed. Incorporate this condition to complete the conditional statement in the code.

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

[11]:
```
# Define a function named `analyze_logins()` that takes in three parameters: `username`, `current_day_logins`, and `average_day_logins`

def analyze_logins(username, current_day_logins, average_day_logins):
    # Display a message about how many login attempts the user has made that day
    print("Current day login total for", username, "is", current_day_logins)

    # Display a message about the average login attempts made by the user on that day
    print("Average logins per day for", username, "is", average_day_logins)

    # Calculate the ratio of the logins made on the current day to the logins made on an average day, storing in a variable named `login_ratio`
    login_ratio = current_day_logins / average_day_logins

    # Return the ratio
    return login_ratio

# Call `analyze_logins()` and store the output in a variable named `login_analysis`
login_analysis = analyze_logins("ejones", 9, 3)
```

```
# Conditional statement that displays an alert about the login activity if it's␣
 ↪more than normal
if login_analysis >= 3:
    print("Alert! This account has more login activity than normal.")
```

```
Current day login total for ejones is 9
Average logins per day for ejones is 3
Alert! This account has more login activity than normal.
```

Hint 1

To calculate the ratio of the logins made on the current day to the logins made on an average day, divide `current_day_logins` by `average_day_logins`.

Assign a variable named `login_ratio` to the result of this calculation, using the `=` assignment operator.

Hint 2

To assign a variable named `login_ratio` to the result of the calculation, use the `=` assignment operator. Write `login_ratio` to the left of `=`, and place the calculation to the right of `=`.

Hint 3

Call the updated `analyze_logins()` function and pass in `"ejones"`, `9`, and `3` as the three arguments, in that order.

1.11 Conclusion

What are your key takeaways from this lab?

The main key takeaways learned from this lab was:

Function Definition: I've learned how to define functions in Python, specifying parameters and utilizing return statements.

Function Invocation: The process of calling functions with specific arguments and capturing the results they return.

Conditional Statements: I've seen how to use conditional statements to execute different actions based on specified conditions.

Data Analysis: The application of functions to analyze data, calculate ratios, and make decisions based on the outcomes.

Modular Code: Writing modular code by encapsulating functionality within functions, contributing to code readability and reusability.

Error Handling: Awareness of potential errors, such as division by zero, and considering appropriate mechanisms for handling them.

In summary, this lab has provided hands-on experience with functions, data analysis, and conditional statements in Python, highlighting the practical aspects of coding and problem-solving.This was very helpful to develop my skills stronger to practice getting better coding with it.

[ ]: