Exemplar - Create more functions

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

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.

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.

```
# Assign `failed_login_list` to the list of the number of failed login attempts per month

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

Hint 2

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?
The output above shows that the function call to `sorted()` sorted the `failed_login_list` in ascending numerical order. The outputted list starts with the lowest number of failed login attempts and ends with the highest number of failed login attempts. The last two numbers in the sorted list (223 and 224) seem to be outlying numbers, indicating an increase in the number of failed login attempts.

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.

```
# Assign `failed_login_list` to the list of the number of failed login attempts per month

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

Hint 2

Question 2

What do you observe from the output above?

The output above shows that the function call to `max()` isolated the highest number from the `failed_login_list`. The highest number of failed login attempts was 264.

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.

```
# 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)
```

Hint 1

Hint 2

Hint 3

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.

```
# 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()`
```

```
analyze_logins("ejones", 9)
```

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

Hint 1

Hint 2

Question 3

What does this function display? Would the output vary for different users?
This function displays the message `"Current day login total for ejones is 9"` when the arguments are `"ejones"` and `9`. The function is defined in a way that's specific to the user, so it would produce a different output for a different user.

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.

```
# Define a function named `analyze_logins()` that takes in three parameters, `username`, `current_day_logins`, a
nd `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 average number of login attempts the user has made that day

  print("Average logins per day for", username, "is", average_day_logins)

# Call `analyze_logins()`

analyze_logins("ejones", 9, 3)
```

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

Hint 1

Hint 2

Hint 3

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.

```
# Define a function named `analyze_logins()` that takes in three parameters, `username`, `current_day_logins`, a
nd `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 average number of login attempts the user has made 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 v
ariable 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()`

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

Hint 2

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 a user's current day login total, average logins per day, and ratio of current day login total to average logins per day. It produces a distinct output for each distinct username that it takes in.

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.

```
# Define a function named `analyze_logins()` that takes in three parameters, `username`, `current_day_logins`, a
nd `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 average number of login attempts the user has made 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

Hint 2

Question 5

How does this version of the `analyze_logins()` function compare to the previous versions?
This version of the `analyze_logins()` function involves a `return` statement instead of a `print()` statement, which allows you to store the output from the function call in a variable for later usage.

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.

```
# 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 average number of login attempts the user has made 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 v
ariable 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

Hint 2

Hint 3

Conclusion

What are your key takeaways from this lab?

- There are a variety of ways a function can be written.
    - It can be written to display information to the screen, or return information that can then be saved in a variable.
    - Also it can be written to take in any number of parameters, use the parameters to execute a series of tasks, and then return a result.

- The `sorted()` function in Python is a built-in function that helps you sort the components of a list.
    - For example, when you call `sorted()` with a list of numbers, it returns the list with the elements in numerical order.
- The `max()` function in Python is a built-in function that helps you identify the element with the maximum value in a list.
    - For example, when you call `max()` with a list of numbers, it returns the largest number in the list.
- The `print()` function in Python is a built-in function that helps display information. It can also be used to directly display the output from another function call.
    - To display the output from another function call, make sure to place it inside a `print()` statement.