Scenerio

I’m working on a project as a security analyst where I manage a list tracking the number of failed login attempts per month. My task is to examine this data for any patterns that could suggest malicious activity. Additionally, I’m developing a function to compare the current day’s login attempts against an average, and I’ll enhance it by incorporating a return statement for better functionality.

In my work as an analyst, I’m tackling a project where I’ve been given a list of failed login attempts for each month, recorded as follows: 119, 101, 99, 91, 92, 105, 108, 85, 88, 90, 264, and 223. These numbers correspond to the months from January through December in chronological order and are stored in a variable called failed_login_list. For this task, I’m using a built-in Python function to sort the list, and I’ll pass that function call directly into a print() statement so I can display and analyze the sorted results.

In [9]:

```python
# 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]
```

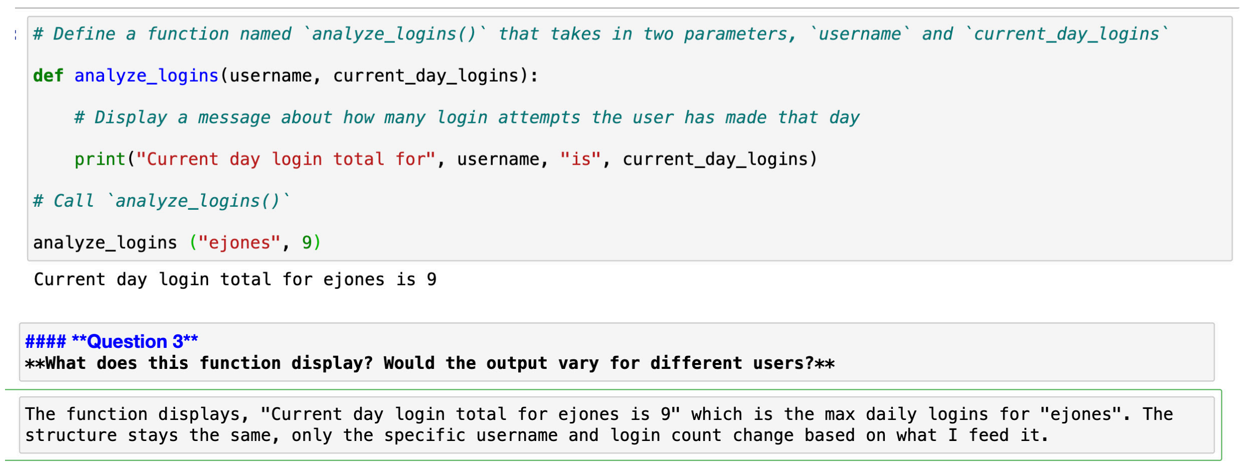
What do you observe from the output above?

```
223 and 264 stand out because they are much higher than the others. These correspond to November (264) and December
(223), indicating a sharp rise late in the year.
```

In this task, I’m working on defining a function called analyze_logins() that tracks daily login attempts. For this task, I’ll create the function to accept two parameters—username and current_day_logins—and each time it’s called, it will show a message detailing the number of login attempts the user made that day.

```python
# 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)
```

Now that I’ve defined the analyze_logins() function, I’ll test its behavior by calling it with the arguments "ejones" and 9.



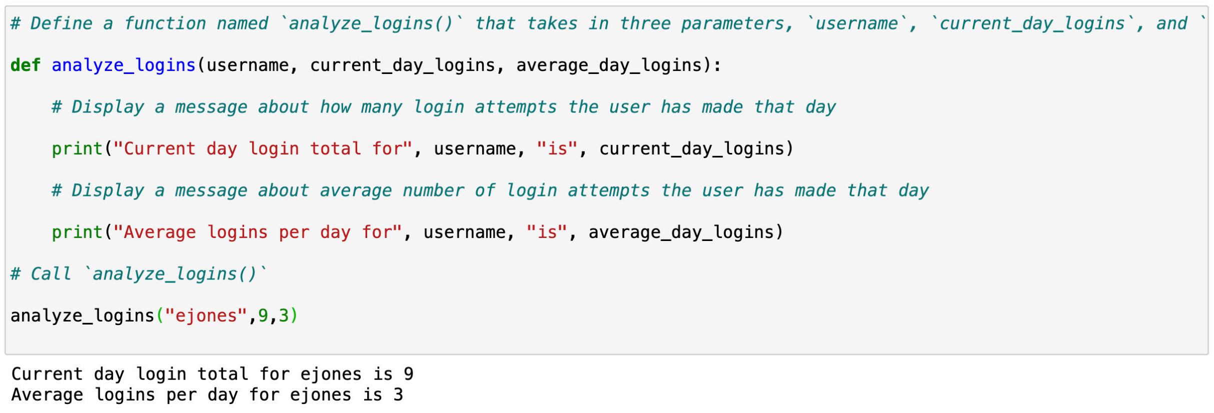
```python
# Define a function named `analyze_logins()` that takes in two parameters, `username` and `current_day_logins`

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
```

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

```
The function displays, "Current day login total for ejones is 9" which is the max daily logins for "ejones". The
structure stays the same, only the specific username and login count change based on what I feed it.
```

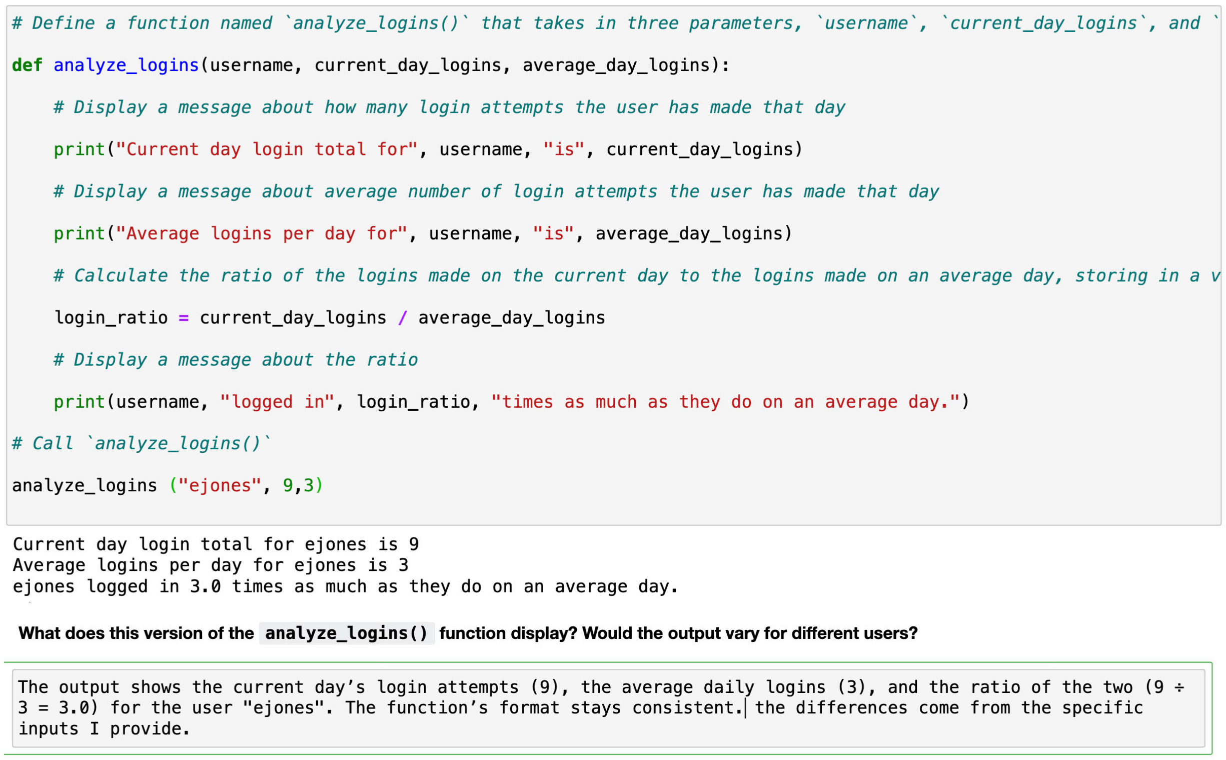
Next, I’m enhancing the analyze_logins() function to include the average number of login attempts for a user on that day, which means adding a third parameter to the function. For this task, I’ll introduce a parameter named average_day_logins and use it to display an extra message about the user’s average login attempts. After updating the function, I’ll call it with the same first two arguments from Task 4—"ejones" and 9—plus a third argument of 3.



```python
# Define a function named `analyze_logins()` that takes in three parameters, `username`, `current_day_logins`, and `

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
```

In this task, I’m further refining the analyze_logins() function by adding a calculation to determine the ratio of current day logins to the average day logins, storing it in a new variable called login_ratio. The function will then display an extra message using this variable. I need to note that if average_day_logins were 0, dividing current_day_logins by it would trigger a ZeroDivisionError: division by zero. However, for this project, I’ll assume all users have logged in at least once before, ensuring average_day_logins is greater than 0 and avoiding that issue. Once the function is updated, I’ll call it with the same arguments as last time—"ejones", 9, and 3.



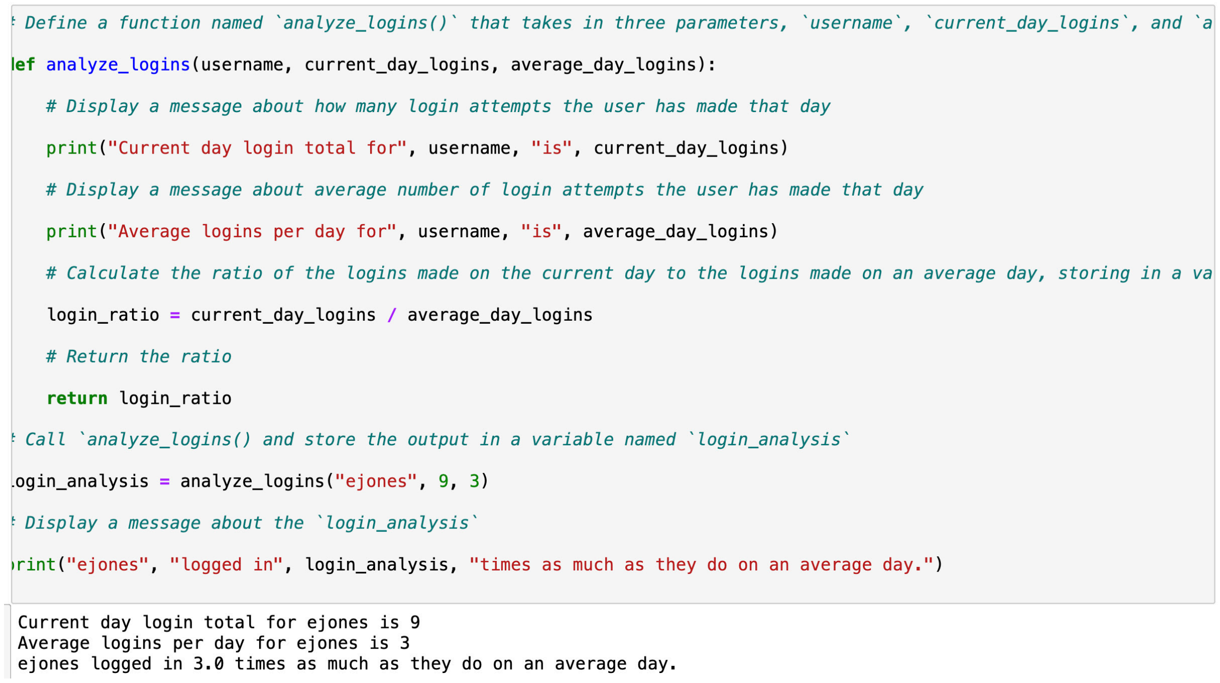
```python
# Define a function named `analyze_logins()` that takes in three parameters, `username`, `current_day_logins`, and `

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

    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.
```

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

```
The output shows the current day’s login attempts (9), the average daily logins (3), and the ratio of the two (9 ÷
3 = 3.0) for the user "ejones". The function’s format stays consistent. the differences come from the specific
inputs I provide.
```

In this task, I’m modifying the analyze_logins() function to use the return keyword to output the login_ratio, making it available for later use in my work. I’ll call the function with the same arguments as before—"ejones", 9, and 3—and store the returned value in a variable called login_analysis. Then, I’ll use a print() statement to display that stored result.



```python
 Define a function named `analyze_logins()` that takes in three parameters, `username`, `current_day_logins`, and `a

lef 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 va

   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`

.ogin_analysis = analyze_logins("ejones", 9, 3)

 Display a message about the `login_analysis`

rint("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.
```

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

```
The previous version foes not return anything, while this version returns the login ratio. Additionally, version
one prints the ratio message within the function itself, whereas version 2 prints it outside after calling the
function. Version 2 is more flexible because it allows the caller to handle the output as needed.
```

In this task, I will use the value of login_analysis in a conditional statement. When login_analysis is greater than or equal to 3, I will consider the login activity as requiring further investigation and display an alert. My task is to complete this conditional statement and include the appropriate alert message.

```python
# Define a function named `analyze_logins()` that takes in three parameters, `username`, `current_day_logins`, and `a

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 va

    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 > 1:
    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.
```