LAB-008-Exemplar_Create more functions

September 20, 2024

1 Modelo: Crea más funciones

Introducción

Las funciones integradas son funciones que existen dentro de Python y que se pueden llamar directamente. Ayudan a los analistas a completar tareas de forma eficiente. Python también admite funciones definidas por el usuario. Son funciones que los analistas escriben para sus necesidades específicas.

Por ejemplo, los patrones en los intentos de inicio de sesión podrían revelar actividades sospechosas. Las funciones de Python pueden ayudar a los analistas a trabajar eficazmente con listas de intentos de inicio de sesión. Tanto las funciones integradas como las funciones definidas por el usuario en Python pueden ayudar a los analistas de seguridad a analizar los intentos de inicio de sesión.

En este laboratorio, usarás funciones integradas para trabajar con una lista de intentos fallidos de inicio de sesión por mes para prepararla para un análisis posterior, y definirás una función que compara los intentos de inicio de sesión del usuario para el día actual con su número promedio de intentos de inicio de sesión.

Consejos para completar este laboratorio

Mientras recorres este laboratorio, ten en cuenta los siguientes consejos:

— `### TU CÓDIGO AQUÍ ###` indica dónde debes escribir el código. Asegúrate de reemplazarlo con tu propio código antes de ejecutar la celda de código. — Siéntete libre de abrir las pistas para obtener información adicional a medida que trabajas en cada tarea. — Para ingresar tu respuesta a una pregunta, haz doble clic en la celda de markdown para editar. Asegúrate de reemplazar "[Haz doble clic para ingresar aquí tus respuestas.]" con tu propia respuesta. — Puedes guardar tu trabajo manualmente haciendo clic en Archivo y luego en Guardar en la barra de menú en la parte superior del cuaderno. — Puedes descargar tu trabajo localmente haciendo un clic en Archivo y luego en Descargar, luego puedes especificar el formato de archivo que prefieras en la barra de menú en la parte superior del cuaderno.

1.1 Situación hipotética

En tu trabajo como analista de seguridad, eres responsable de trabajar con una lista que contiene la cantidad de intentos fallidos que se han producido cada mes. Identificarás cualquier patrón que pueda indicar actividad maliciosa. También eres responsable de definir una función que compare los inicios de sesión del día actual con un promedio y mejorarla añadiendo una sentencia `return`.

Tarea 1 En tu trabajo como analista, imagina que te proporcionan una lista con el número de intentos fallidos de inicio de sesión al mes, como la siguiente:

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

Esta lista está organizada por orden cronológico de meses (enero, febrero, marzo, abril, mayo, junio, julio, agosto, septiembre, octubre, noviembre y diciembre).

Esta lista se almacena en una variable llamada `failed_login_list`.

En esta tarea, usa una función integrada de Python para ordenar la lista. Pasarás la llamada a la función que ordena la lista directamente a la función `print()`. Esto te permitirá mostrar y examinar el resultado.

Asegúrate de reemplazar cada fragmento `### TU CÓDIGO AQUÍ ###` con tu propio código antes de ejecutar la siguiente celda.

[1]:
```
# Asigna a `failed_login_list` la lista con el número de intentos fallidos de␣
↪inicio de sesión por mes

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

# Ordena `failed_login_list` en orden numérico ascendente y muestra el resultado

print(sorted(failed_login_list))
```

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

Pista 1

Para ordenar `failed_login_list` en orden numérico ascendente, utiliza la función `sorted()`.

Se trata de una función integrada de Python que recibe una lista, ordena sus componentes y devuelve el resultado.

Pista 2

Para ordenar la `failed_login_list` en orden numérico ascendente, llama a la función `sorted()` y pásale la lista `failed_login_list` como argumento.

Para mostrar el resultado, asegúrate de colocar la llamada a `sorted()` dentro de la función `print()`.

Pregunta 1 ¿Qué observas de la salida anterior? ¿Notas algún número fuera de lo normal que indique un aumento en el número de intentos de inicio de sesión fallidos?

La salida anterior muestra que la llamada a la función `sorted()` ordenó la `failed_login_list` en orden numérico ascendente. La lista resultante comienza con el número más bajo de intentos de inicio de sesión fallidos y termina con el número más alto de intentos de inicio de sesión fallidos. Los dos últimos números de la lista ordenada (223 y 224) parecen ser números atípicos, lo que indica un aumento en el número de intentos de inicio de sesión fallidos.

1.2 Tarea 2

Ahora, querrás aislar el mayor número de intentos de inicio de sesión fallidos para poder investigar más tarde la información sobre el mes en que se produjo ese valor más alto.

Utilizarás la función que devuelve el mayor elemento numérico de una lista. Luego, pasarás esta función a la función `print()` para mostrar el resultado. Esto te permitirá determinar qué mes investigar más a fondo.

Asegúrate de reemplazar cada fragmento `### TU CÓDIGO AQUÍ ###` con tu propio código antes de ejecutar la siguiente celda.

[2]:
```
# Asigna a `failed_login_list` la lista con el número de intentos fallidos de␣
↪inicio de sesión por mes

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

# Determina el número más alto de intentos fallidos de inicio de sesión de␣
↪`failed_login_list` y muestra el resultado

print(max(failed_login_list))
```

```
264
```

Pista 1

Para determinar el número más alto de intentos fallidos de inicio de sesión de `failed_login_list`, utiliza la función `max()`.

Se trata de una función integrada de Python que recibe una secuencia, identifica su valor máximo y devuelve el resultado.

Pista 2

Para determinar el número más alto de intentos fallidos de inicio de sesión de `failed_login_list`, llama a la función `max()` con `failed_login_list` como argumento.

Para mostrar el resultado, asegúrate de colocar la llamada a `max()` dentro de la función `print()`.

Pregunta 2 ¿Qué observas de la salida anterior?

La salida anterior muestra que la llamada a la función `max()` aisló el número más alto de la `failed_login_list`. El número más alto de intentos fallidos de inicio de sesión fue 264.

Tarea 3

En tu trabajo como analista, primero definirás una función que muestre un mensaje sobre cuántos intentos de inicio de sesión ha realizado un usuario ese día.

En esta tarea, define una función llamada `analyze_logins()` que reciba dos parámetros, `username` y `current_day_logins`. Cada vez que se llame a esta función, debería mostrar un mensaje sobre el número de intentos de inicio de sesión que el usuario ha hecho ese día.

Asegúrate de reemplazar cada `### TU CÓDIGO AQUÍ ###` con tu propio código antes de ejecutar la siguiente celda. Ten en cuenta que la celda de código contendrá solo una definición de la función, por lo que su ejecución no producirá ninguna salida.

[3]:
```
# Define una función llamada `analyze_logins()` que recibe dos parámetros, ␣
↪`username` y `current_day_logins`

def analyze_logins(username, current_day_logins):

    # Muestra un mensaje sobre cuántos intentos de inicio de sesión ha hecho el␣
↪usuario ese día

    print("El número total de inicios de sesión del día de hoy para", username,␣
↪"es", current_day_logins)
```

Pista 1

Para escribir una cabecera de función en Python, comienza con la palabra clave `def`, seguida del nombre de la función y luego paréntesis.

Pista 2

En Python, para definir una función que recibe parámetros, coloca los nombres de los parámetros entre paréntesis en la cabecera de la función y usa una `,` entre cada parámetro y el siguiente.

Pista 3

Para definir una función llamada `analyze_logins()` que reciba dos parámetros, `username` y `current_day_logins`, empieza con la palabra clave `def`, seguida de `analyze_logins()`, y escribe `username, current_day_logins` dentro de los paréntesis. Asegúrate de escribir este código antes de `:`.

1.3 Tarea 4

Ahora que definiste la función `analyze_logins()`, llámala para comprobar cómo se comporta.

Llama a `analyze_logins()` con los argumentos `"ejones"` y `9`.

Asegúrate de reemplazar cada fragmento `### TU CÓDIGO AQUÍ ###` con tu propio código antes de ejecutar la siguiente celda.

[4]:
```
# Define una función llamada `analyze_logins()` que recibe dos parámetros, ␣
↪`username` y `current_day_logins`

def analyze_logins(username, current_day_logins):

    # Muestra un mensaje sobre cuántos intentos de inicio de sesión ha hecho el␣
↪usuario ese día
```

```
    print("El número total de inicios de sesión del día de hoy para", username,␣
 ↪"es", current_day_logins)

# Llama a `analyze_logins()`

analyze_logins("ejones", 9)
```

```
El número total de inicios de sesión del día de hoy para ejones es 9
```

Pista 1

Para llamar a la función `analyze_logins()` una vez definida, escribe `analyze_logins()`. Luego asegúrate de colocar los argumentos `"ejones"` y `9` entre paréntesis.

Pista 2

La llamada a la función debe escribirse como `analyze_logins("ejones", 9)`.

Pregunta 3 ¿Qué muestra esta función? ¿Varía la salida en función del usuario?

Esta función muestra el mensaje `"El número total de inicios de sesión del día de hoy para ejones es 9"` cuando los argumentos son `"ejones"` y `9`. La función se define de una manera específica para el usuario, por lo que produciría una salida diferente para un usuario diferente.

Tarea 5

Ahora, tendrás que ampliar esta función para que también proporcione el número promedio de intentos de inicio de sesión realizados por el usuario en ese día. Para ello será necesario incorporar un tercer parámetro a la definición de la función.

En esta tarea, agrega un parámetro llamado `average_day_logins`. El código utilizará este parámetro para mostrar un mensaje adicional. El mensaje adicional indicará el promedio de intentos de inicio de sesión realizados por el usuario en ese día. A continuación, llama a la función con los mismos argumentos primero y segundo utilizados en la Tarea 4 y un tercer argumento de `3`.

Asegúrate de reemplazar cada fragmento `### TU CÓDIGO AQUÍ ###` con tu propio código antes de ejecutar la siguiente celda.

[5]:
```
# Define una función llamada `analyze_logins()` que recibe tres parámetros,␣
 ↪`username`, `current_day_logins` y `average_day_logins`

def analyze_logins(username, current_day_logins, average_day_logins):

    # Muestra un mensaje sobre cuántos intentos de inicio de sesión ha hecho el␣
 ↪usuario ese día

    print("El número total de inicios de sesión del día de hoy para", username,␣
 ↪"es", current_day_logins)

    # Muestra un mensaje sobre el número promedio de intentos de inicio de␣
 ↪sesión que el usuario ha realizado ese día
```

```
    print("El promedio de inicios de sesión por día para", username, "es",␣
 ↪average_day_logins)

# Llama a `analyze_logins()`

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

```
El número total de inicios de sesión del día de hoy para ejones es 9
El promedio de inicios de sesión por día para ejones es 3
```

Pista 1

En Python, para definir una función que recibe parámetros, coloca los nombres de los parámetros entre paréntesis en la cabecera de la función, con una `,` entre cada parámetro y el siguiente.

Pista 2

Necesitas definir una función llamada `analyze_logins()` que reciba tres parámetros, `username`, `current_day_logins` y `average_day_logins`. Así que tendrás que escribir `username, current_day_logins, average_day_logins` entre paréntesis.

Pista 3

Para llamar a la función `analyze_logins()` una vez definida, escribe `analyze_logins()`. Luego asegúrate de colocar los argumentos `"ejones"`, `9` y `3` dentro de los paréntesis.

1.4 Tarea 6

En esta tarea, ampliarás aún más la función. Incluye un cálculo para obtener la relación entre los inicios de sesión realizados en el día actual y los realizados en un día promedio. Guárdalo en una nueva variable llamada `login_ratio`. La función muestra un mensaje adicional que utiliza esta variable.

Ten en cuenta que si `average_day_logins` es igual a `0`, dividir `current_day_logins` entre `average_day_logins` provocará un error. Debido al error, Python mostrará el siguiente mensaje: `ZeroDivisionError: division by zero`. Para esta actividad, supón que todos los usuarios habrán iniciado sesión al menos una vez antes. Esto significa que sus `average_day_logins` serán mayores que `0`, y la función no implicará dividir por cero.

Después de definir la función, llámala con los mismos argumentos que usaste en la tarea anterior.

Asegúrate de reemplazar cada fragmento `### TU CÓDIGO AQUÍ ###` con tu propio código antes de ejecutar la siguiente celda.

[6]:
```
# Define una función llamada `analyze_logins()` que recibe tres parámetros,␣
 ↪`username`, `current_day_logins` y `average_day_logins`

def analyze_logins(username, current_day_logins, average_day_logins):

```

```
    # Muestra un mensaje sobre cuántos intentos de inicio de sesión ha hecho el␣
↪usuario ese día

    print("El número total de inicios de sesión del día de hoy para", username,␣
↪"es", current_day_logins)

    # Muestra un mensaje sobre el número promedio de intentos de inicio de␣
↪sesión que el usuario ha realizado ese día

    print("El promedio de inicios de sesión por día para", username, "es",␣
↪average_day_logins)

    # Calcula la relación entre los inicios de sesión realizados en el día␣
↪actual y los realizados en un día promedio, almacenándola en una variable␣
↪llamada `login_ratio`

    login_ratio = current_day_logins / average_day_logins

    # Muestra un mensaje sobre la relación

    print(username, "inició sesión", login_ratio, "veces más que en un día␣
↪promedio.")

# Llama a `analyze_logins()`

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

```
El número total de inicios de sesión del día de hoy para ejones es 9
El promedio de inicios de sesión por día para ejones es 3
ejones inició sesión 3.0 veces más que en un día promedio.
```

Pista 1

Para calcular la relación entre los inicios de sesión realizados en el día actual y los realizados en un día promedio, divide `current_day_logins` entre `average_day_logins`. Asegúrate de escribir este cálculo a la derecha del `=`, para que el resultado se almacene en la variable `login_ratio`.

Pista 2

Llama a la función actualizada `analyze_logins()` y pasa `"ejones"`, `9` y `3` como los tres argumentos, en ese orden.

Pregunta 4 ¿Qué muestra esta versión de la función `analyze_logins()`? ¿Varía la salida en función del usuario?

Esta versión de la función `analyze_logins()` muestra el total de inicios de sesión del día actual de un usuario, el promedio de inicios de sesión por día y la relación entre el total de inicios de sesión del día actual y el promedio de inicios de sesión por día. Produce una salida distinta para cada nombre de usuario distinto que toma.

1.5 Tarea 7

Continuarás trabajando con la función `analyze_logins()` y le agregarás una sentencia return. Las sentencias return te permiten enviar información de vuelta a la llamada a la función.

En esta tarea, usa la palabra clave `return` para obtener el `login_ratio` de la función, de modo que puedas utilizarlo más adelante en tu trabajo.

Llamarás a la función con los mismos argumentos que usaste en la tarea anterior y guardarás la salida de la llamada a la función en una variable llamada `login_analysis`. A continuación, utilizarás una sentencia `print()` para mostrar la información guardada.

Asegúrate de reemplazar cada fragmento `### TU CÓDIGO AQUÍ ###` con tu propio código antes de ejecutar la siguiente celda.

[7]:
```
# Define una función llamada `analyze_logins()` que recibe tres parámetros, `username`, `current_day_logins` y `average_day_logins`

def analyze_logins(username, current_day_logins, average_day_logins):

    # Muestra un mensaje sobre cuántos intentos de inicio de sesión ha hecho el usuario ese día

    print("El número total de inicios de sesión del día de hoy para", username, "es", current_day_logins)

    # Muestra un mensaje sobre el número promedio de intentos de inicio de sesión que el usuario ha realizado ese día

    print("El promedio de inicios de sesión por día para", username, "es", average_day_logins)

    # Calcula la relación entre los inicios de sesión realizados en el día actual y los realizados en un día promedio, almacenándola en una variable llamada `login_ratio`

    login_ratio = current_day_logins / average_day_logins

    # Devuelve la relación

    return login_ratio

# Llama a `analyze_logins()` y guarda la salida en una variable llamada `login_analysis`

login_analysis = analyze_logins("ejones", 9, 3)

# Muestra un mensaje sobre el `login_analysis`
```

```
print("ejones", "inició sesión", "login_analysis", "veces más que en un día␣
↪promedio.")
```

```
El número total de inicios de sesión del día de hoy para ejones es 9
El promedio de inicios de sesión por día para ejones es 3
ejones inició sesión 3.0 veces más que en un día promedio.
```

Pista 1

Al definir la función `analyze_logins()` esta vez, coloca la palabra clave `return` delante de la salida que deseas que regrese la función.

Pista 2

Al definir la función `analyze_logins()` esta vez, escribe `return` delante de `login_ratio`. (No coloques paréntesis después de la palabra clave `return`. No es una función).

Pregunta 5 ¿Cómo se compara esta versión de la función `analyze_logins()` con las versiones anteriores?

Esta versión de la función `analyze_logins()` incluye una sentencia `return` en lugar de una sentencia `print()`, lo que permite almacenar la salida de la llamada a la función en una variable para su uso posterior.

1.6 Tarea 8

En esta tarea, usarás el valor de `login_analysis` en una sentencia condicional. Cuando el valor de `login_analysis` sea mayor o igual a `3`, la actividad de inicio de sesión requerirá una investigación adicional y se mostrará una alerta. Incorpora esta condición para completar la sentencia condicional en el código.

Asegúrate de reemplazar cada fragmento `### TU CÓDIGO AQUÍ ###` con tu propio código antes de ejecutar la siguiente celda.

[8]:
```
# Define una función llamada `analyze_logins()` que recibe tres parámetros,␣
↪`username`, `current_day_logins` y `average_day_logins`

def analyze_logins(username, current_day_logins, average_day_logins):

    # Muestra un mensaje sobre cuántos intentos de inicio de sesión ha hecho el␣
↪usuario ese día

    print("El número total de inicios de sesión del día de hoy para", username,␣
↪"es", current_day_logins)

    # Muestra un mensaje sobre el número promedio de intentos de inicio de␣
↪sesión que el usuario ha realizado ese día
```

```
    print("El promedio de inicios de sesión por día para", username, "es",␣
 ↪average_day_logins)

    # Calcula la relación entre los inicios de sesión realizados en el día␣
 ↪actual y los realizados en un día promedio, almacenándola en una variable␣
 ↪llamada `login_ratio`

    login_ratio = current_day_logins / average_day_logins

    # Devuelve la relación

    return login_ratio

# Llama a `analyze_logins()` y guarda la salida en una variable llamada␣
 ↪`login_analysis`

login_analysis = analyze_logins("ejones", 9, 3)

# Sentencia condicional que muestra una alerta sobre la actividad de inicio de␣
 ↪sesión si es superior a lo normal

if login_analysis >= 3:
    print("¡Cuidado! Esta cuenta tiene más actividad de inicio de sesión de lo␣
 ↪normal.")
```

```
El número total de inicios de sesión del día de hoy para ejones es 9
El promedio de inicios de sesión por día para ejones es 3
¡Cuidado! Esta cuenta tiene más actividad de inicio de sesión de lo normal.
```

Pista 1

Para calcular la relación entre los inicios de sesión realizados en el día actual y los realizados en un día promedio, divide `current_day_logins` entre `average_day_logins`.

Asigna una variable llamada `login_ratio` al resultado de este cálculo, usando el operador de asignación `=`.

Pista 2

Para asignar una variable llamada `login_ratio` al resultado del cálculo, utiliza el operador de asignación `=`. Escribe `login_ratio` a la izquierda de `=`, y coloca el cálculo a la derecha de `=`.

Pista 3

Llama a la función actualizada `analyze_logins()` y pasa `"ejones"`, `9` y `3` como los tres argumentos, en ese orden.

1.7 Conclusión

¿Qué conclusiones clave obtuviste de este laboratorio?

— Hay varias formas de escribir una función. — Se puede escribir para mostrar información en la pantalla o para devolver información que luego se puede guardar en una variable. — También se puede escribir para que reciba cualquier número de parámetros, utilizar los parámetros para ejecutar una serie de tareas, y luego devolver un resultado. — La función `sorted()` de Python es una función integrada que te ayuda a ordenar los componentes de una lista. — Por ejemplo, cuando llamas a `sorted()` con una lista de números, devuelve la lista con los elementos en orden numérico. — La función `max()` de Python es una función integrada que te ayuda a identificar el elemento con el valor máximo en una lista. — Por ejemplo, cuando llamas a `max()` con una lista de números, devuelve el número más grande de la lista. — La función `print()` en Python es una función integrada que ayuda a mostrar información. También se puede utilizar para mostrar directamente la salida de otra llamada a una función. — Para mostrar la salida de otra llamada a una función, asegúrate de colocarla dentro de la función `print()`.