LAB-007-Exemplar_Define and call a function

September 20, 2024

1 Modelo: Define y llama a una función

Introducción

Como analista de seguridad, cuando escribes código Python para automatizar una determinada tarea, a menudo te encontrarás con la necesidad de reutilizar el mismo bloque de código más de una vez. Por eso son importantes las funciones. Puedes llamar a esa función siempre que necesites que la computadora ejecute esos pasos. Python no solo tiene funciones integradas que ya se han definido, sino que también proporciona las herramientas para que los usuarios definan sus propias funciones. Los analistas de seguridad a menudo definen y llaman a funciones en Python para automatizar series de tareas.

En este laboratorio, practicarás la definición y llamada de funciones en Python.

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

Escribir funciones en Python es una habilidad útil en tu trabajo como analista de seguridad. En este laboratorio, definirás y llamarás a una función que muestra una alerta sobre un posible problema de seguridad. Además, trabajarás con una lista de nombres de usuario de empleados, creando una función que convierte la lista en una cadena.

Tarea 1

La siguiente celda de código contiene una función definida por el usuario llamada `alert()`.

Para esta tarea, analiza la definición de la función y toma nota de tus observaciones.

No necesitarás ejecutar la celda para responder a la pregunta que sigue. Pero si ejecutas la celda, ten en cuenta que no producirá una salida porque la función solo se está definiendo aquí.

[1]:
```
# Define una función llamada `alert()`

def alert():
    print("Posible problema de seguridad. Investiga más a fondo.")
```

Pista 1

Al analizar la definición de la función, asegúrate de observar el cuerpo de la función, que es el bloque de código con sangría que aparece después de la cabecera de la función. El cuerpo de la función te dice lo que hace la función.

Pregunta 1 Resume lo que hace la función definida por el usuario anterior con tus propias palabras. Piensa en cuál sería la salida si se llamara a esta función.

La función `alert()` muestra la cadena `"Posible problema de seguridad. Investiga más a fondo."` en la pantalla. Esta función se puede utilizar para informar a los analistas de seguridad sobre posibles problemas de seguridad en un sistema. Si se llamara a esta función, la salida mostraría `Posible problema de seguridad. Investiga más a fondo.`. Recuerda que cuando se muestra una cadena, las comillas que encierran la cadena no aparecen en la salida.

1.2 Tarea 2

Para esta tarea, llama a la función `alert()` que se definió anteriormente y analiza la salida.

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

[2]:
```
# Define una función llamada `alert()`

def alert():
    print("Posible problema de seguridad. Investiga más a fondo.")

# Llama a la función `alert()`

alert()
```

```
Posible problema de seguridad. Investiga más a fondo.
```

Pista 1

Para llamar a la función, escribe `alert()` después de la definición de la función. Ten en cuenta que la función se puede llamar solo después de que se defina.

Pregunta 2 ¿Cuáles son las ventajas de colocar este código en una función en lugar de ejecutarlo directamente?

Colocar el código en una función permite reutilizarlo de forma eficaz. Siempre que necesites mostrar los mensajes sobre un posible problema de seguridad e investigar más a fondo, solo tienes que llamar a la función `alert()`. La alternativa sería escribir esa sentencia `print()` cada vez, lo que sería tedioso.

Tarea 3

Las funciones pueden incluir otros componentes con los que ya has trabajado. La siguiente celda de código contiene una variación de la función `alert()` que ahora utiliza un bucle `for` para mostrar el mensaje de alerta varias veces.

Para esta tarea, llama a la nueva función `alert()` y observa la salida.

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

[3]:
```
# Define una función llamada `alert()`

def alert():
    for i in range(3):
    print("Posible problema de seguridad. Investiga más a fondo.")

# Llama a la función `alert()`

alert()
```

```
Posible problema de seguridad. Investiga más a fondo.
Posible problema de seguridad. Investiga más a fondo.
Posible problema de seguridad. Investiga más a fondo.
```

Pista 1

Para llamar a la función, escribe `alert()` después de la definición de la función. Ten en cuenta que la función se puede llamar solo después de que se defina.

Pregunta 3 ¿Cómo se compara la salida anterior con la salida de llamar a la versión anterior de la función `alert()`? ¿En qué se diferencian las dos definiciones de la función?

La salida muestra `Posible problema de seguridad. Investiga más a fondo.` tres veces, cada vez en una línea nueva. Mientras tanto, la salida de llamar a la versión anterior de `alert()` muestra el mensaje solo una vez. La diferencia de comportamiento se debe al bucle `for` utilizado en la segunda versión. Este bucle itera sobre un rango de números (especificado por `range(3)` y ejecuta una sentencia `print()` en cada iteración. x Esta sentencia `print()` es la misma que la de la definición de la función anterior.

1.3 Tarea 4

En la siguiente parte de tu trabajo, trabajarás con una lista de nombres de usuario autorizados, que representan a los usuarios que pueden ingresar a un sistema. Desarrollarás una función que te ayudará a convertir la lista de nombres de usuario autorizados en una gran cadena. Estructurar

estos datos de manera diferente te permite trabajar con ellos de distintas maneras. Por ejemplo, estructurar los nombres de usuario como una lista te permite agregar o eliminar fácilmente un nombre de usuario de ella. En cambio, estructurarlos como una cadena te permite colocar fácilmente su contenido en un archivo de texto.

Para esta tarea, empieza definiendo una función llamada `list_to_string()`. Escribe la cabecera de la función.

Asegúrate de reemplazar el fragmento `### TU CÓDIGO AQUÍ ###` con tu propio código. Ten en cuenta que la ejecución de esta celda producirá un error, ya que esta celda solo contendrá la cabecera de la función; escribirás el cuerpo de la función y completarás la definición de la función en una tarea posterior.

[ ]:
```
# Define una función llamada `list_to_string()`

def list_to_string():
```

Pista 1

Para escribir la cabecera de la función, comienza con la palabra clave `def`, seguida del nombre de la función, los paréntesis y los dos puntos.

Tarea 5

Ahora comenzarás a desarrollar el cuerpo de la función `list_to_string()`.

En la siguiente celda de código, se te proporciona una lista de nombres de usuario autorizados, almacenados en una variable llamada `username_list`. Tu tarea es completar el cuerpo de la función `list_to_string()`. Recuerda que el cuerpo de una función debe tener sangría. Para completar el cuerpo de la función, escribe un bucle que itere a través de los elementos de la `username_list` y muestre cada elemento. A continuación, llama a la función y ejecuta la celda para observar lo que ocurre.

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

[5]:
```
# Define una función llamada `list_to_string()`

def list_to_string():

  # Almacena la lista de nombres de usuario autorizados en una variable llamada␣
↪`username_list`

  username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab",␣
↪"gesparza", "alevitsk", "wjaffrey"]

  # Escribe un bucle for (para) que itere a través de los elementos de␣
↪`username_list` y muestre cada elemento

  for i in username_list:
    print(i)
```

```
# Llama a la función `list_to_string()`

list_to_string()
```

```
elarson
bmoreno
tshah
sgilmore
eraab
gesparza
alevitsk
wjaffrey
```

Pista 1

El bucle `for` en el cuerpo de la función `list_to_string()` debe iterar a través de los elementos de `username_list`. Por lo tanto, utiliza la variable `username_list` para completar la condición del bucle `for`.

Pista 2

En cada iteración del bucle `for`, se mostrará un elemento de `username_list`. La variable de bucle `i` representa cada elemento de `username_list`. Para completar la sentencia `print()` dentro del bucle `for`, pasa `i` a la llamada a la función `print()`.

Pista 3

Para llamar a la función, escribe `list_to_string()` después de la definición de la función. Recuerda que la función se puede llamar solo después de que se defina.

Pregunta 4 ¿Qué observas de la salida anterior?

La salida muestra cada elemento de `username_list` en una nueva línea.

1.4 Tarea 6

La concatenación de cadenas es un potente concepto de codificación. Te permite combinar varias cadenas para formar una cadena grande, utilizando el operador de suma (`+`). A veces, los analistas necesitan combinar datos individuales en un único valor de cadena. En esta tarea, usarás la concatenación de cadenas para modificar la definición de la función `list_to_string()`.

En la siguiente celda de código, se te proporciona una variable llamada `sum_variable` que inicialmente contiene una cadena vacía. Tu tarea es utilizar la concatenación de cadenas para combinar los nombres de usuario de la `username_list` y almacenar el resultado en `sum_variable`.

En cada iteración del bucle `for`, agrega el elemento actual de `username_list` a `sum_variable`. Al final de la definición de la función, escribe una sentencia `print()` para mostrar el valor de `sum_variable` en esa etapa del proceso. A continuación, ejecuta la celda para llamar a la función `list_to_string()` y examina su salida.

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

[6]:
```
# Define una función llamada `list_to_string()`

def list_to_string():

  # Almacena la lista de nombres de usuario autorizados en una variable llamada␣
↪`username_list`

  username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab",␣
↪"gesparza", "alevitsk", "wjaffrey"]

  # Asigna una cadena vacía a `sum_variable`

  sum_variable = ""

  # Escribe un bucle for (para) que itere a través de los elementos de␣
↪`username_list` y muestre cada elemento

  for i in username_list:
    sum_variable = sum_variable + i

  # Muestra el valor de `sum_variable`

  print(sum_variable)

# Llama a la función `list_to_string()`

list_to_string()
```

```
elarsonbmorenotshahsgilmoreeraabgesparzaalevitskwjaffrey
```

Pista 1

Dentro del bucle `for`, completa la línea que actualiza la `sum_variable` en cada iteración. La variable de bucle `i` representa cada elemento de `username_list`. Como necesitas sumar el elemento actual al valor actual de `sum_variable`, coloca `i` después del operador de suma `(+)`.

Pista 2

Utiliza la función `print()` para mostrar el valor de `sum_variable`. Asegúrate de pasar `sum_variable` a la llamada a `print()`.

Pregunta 5 ¿Qué observas de la salida anterior?

La salida muestra todos los elementos de `username_list` fusionados en una línea. En su formato actual, la salida es difícil de leer. Es difícil descifrar dónde termina un nombre de usuario y comienza el siguiente.

1.5 Tarea 7

En esta tarea final, modificarás el código que escribiste anteriormente para mejorar la legibilidad de la salida.

Esta vez, en la definición de la función `list_to_string()`, agrega una coma y un espacio después de cada nombre de usuario. Esto evitará que todos los nombres de usuario se crucen en la salida. Agregar una coma ayuda a separar claramente un nombre de usuario del siguiente en la salida. Agregar un espacio después de la coma como separador adicional entre un nombre de usuario y el siguiente facilita la lectura de la salida. A continuación, llama a la función y ejecuta la celda para observar la salida.

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

[7]:
```
# Define una función llamada `list_to_string()`

def list_to_string():

  # Almacena la lista de nombres de usuario autorizados en una variable llamada␣
↪`username_list`

  username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab",␣
↪"gesparza", "alevitsk", "wjaffrey"]

  # Asigna una cadena vacía a `sum_variable`

  sum_variable = ""

  # Escribe un bucle for (para) que itere a través de los elementos de␣
↪`username_list` y muestre cada elemento

  for i in username_list:
    sum_variable = sum_variable + i + ", "

  # Muestra el valor de `sum_variable`

  print(sum_variable)

# Llama a la función `list_to_string()`

list_to_string()
```

```
elarson, bmoreno, tshah, sgilmore, eraab, gesparza, alevitsk, wjaffrey,
```

Pista 1

Dentro del bucle `for`, completa la línea que actualiza la `sum_variable` en cada iteración. La variable de bucle `i` representa cada elemento de `username_list`. Después de agregar el elemento

actual al valor actual de `sum_variable`, agrega una cadena que contenga una coma seguida de un espacio.

Para completar este paso, coloca `" , "` después del último operador de suma (`+`).

Pista 2

Para llamar a la función, escribe `list_to_string()` después de la definición de la función. Ten en cuenta que la función se puede llamar solo después de que se defina.

Pregunta 6 ¿Qué observas esta vez en la salida de la llamada a la función?

La salida muestra todos los elementos de `username_list` en una línea. Esta vez, hay una coma y un espacio después de cada nombre de usuario. Este formato es mucho más fácil de leer. Es más fácil distinguir un nombre de usuario de otro.

1.6 Conclusión

¿Qué conclusiones clave obtuviste de este laboratorio?

— Python te permite definir y llamar a las funciones que creas. — Los componentes principales de una cabecera de definición de función incluyen la cabecera de la función y el cuerpo de la función. — La cabecera de la función incluye la palabra clave `def`, seguida del nombre de la función, seguida de paréntesis, seguida de dos puntos. — El cuerpo de la función incluye un bloque de código con sangría que indica a la computadora qué hacer cuando se llama a la función. — La concatenación de cadenas consiste en utilizar el operador de suma (`+`) para combinar varias cadenas. — Un caso de uso para la concatenación de cadenas es combinar las cadenas de una lista en una cadena grande.