LAB_014_Exemplar_Updated

September 20, 2024

1 Ejemplo: Depura código Python

1.1 Introducción

Uno de los mayores desafíos que enfrentan los analistas es garantizar que los procesos automatizados se ejecuten sin problemas. La depuración es una práctica importante que los analistas de seguridad incorporan en su trabajo para identificar errores en el código y resolverlos, para que este logre el resultado deseado.

En este laboratorio, completarás una serie de tareas para desarrollar y aplicar tus habilidades de depuración 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 por tu propio código antes de ejecutar la celda de código.
- Siéntete libre de abrir las pistas para obtener información adicional mientras 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.]" por tu 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 clic en Archivo y luego en Descargar. Después, puedes especificar el formato de archivo que prefieras en la barra de menú en la parte superior del cuaderno.

1.2 Escenario

En tu trabajo como analista de seguridad, debes aplicar estrategias de depuración para asegurarte de que tu código funcione correctamente.

A lo largo de este laboratorio, trabajarás con código similar a lo que escribiste antes, pero que ahora tiene algunos errores que deben corregirse. Tendrás que leer celdas de código, ejecutarlas, identificar los errores y ajustar el código para resolverlos.

1.3 Tarea 1

La siguiente celda de código contiene un error de sintaxis. En esta tarea, ejecutarás el código, identificarás por qué se produce el error y modificarás el código para resolverlo. (Para asegurarte de que se haya resuelto, vuelve a ejecutar el código y comprueba que ahora funcione correctamente).

[1]:
```
# Bucle for que itera en un rango de números
# y muestra un mensaje en cada iteración

for i in range(10)
    print("No se puede establecer conexión")
```

```
  File "<ipython-input-1-717a54e88184>", line 4
    for i in range(10)
                      ^
SyntaxError: invalid syntax
```

Pista 1

Para el encabezado de un bucle `for` en Python, se debe usar una puntuación específica al final.

Pista 2

Para el encabezado de un bucle `for` en Python, se deben usar dos puntos (`:`) al final.

Pregunta 1 ¿Qué ocurre cuando se ejecuta el código antes de modificarlo? ¿Cómo puede solucionar esto?

Cuando el código se ejecuta antes de modificarlo, el resultado muestra `SyntaxError: invalid syntax`. Esto indica que hay un error de sintaxis. Este se debe a la falta de `:` al final del encabezado del bucle `for`. Para solucionar esto, puedes agregar `:` en esa posición.

1.4 Tarea 2

En la siguiente celda de código, se te proporciona una lista de nombres de usuario. Hay un problema con la sintaxis. En esta tarea, ejecutarás la celda, observarás lo que sucede y modificarás el código para solucionar el problema.

[2]:
```
# Asigna `usernames_list` a una lista de nombres de usuario

usernames_list = ["djames", "jpark", "tbailey", "zdutchma", "esmith",␣
 ↪"srobinso", "dcoleman", "fbautist"]

# Muestra `usernames_list`

print(usernames_list)
```

```
['djames', 'jpark', 'tbailey', 'zdutchma', 'esmith', 'srobinso', 'dcoleman',
'fbautist']
```

Pista 1

Cada elemento en `usernames_list` es un nombre de usuario y debe ser una cadena. En Python, una cadena debe estar encerrada entre comillas.

Pista 2

Al crear una lista en Python, los elementos de la lista deben estar separados por comas. Debe haber una coma entre cada par de elementos consecutivos.

Pregunta 2 ¿Qué ocurre cuando se ejecuta el código antes de modificarlo? ¿Cómo se podría solucionar esto?

Cuando el código se ejecuta antes de modificarlo, el resultado muestra `SyntaxError: invalid syntax`. El problema se produjo al asignar un valor a `usernames_list`. Al cuarto nombre de usuario le falta una comilla de cierre. Además, falta una coma entre el cuarto y el quinto nombre de usuario. Cada nombre de usuario en la lista debe ser una cadena y se deben usar comas para separar un nombre de usuario del siguiente. Para corregir el error de sintaxis, puedes agregar una comilla de cierre para especificar correctamente el cuarto nombre de usuario como una cadena y, luego, agregar una coma entre el cuarto y el quinto nombre de usuario para separarlos. Por lo tanto, en lugar de `"zdutchma "esmith",`, debería ser `"zdutchma", "esmith",`.

1.5 Tarea 3

En la siguiente celda de código hay un error de sintaxis. Tu tarea es ejecutar la celda, identificar qué está causando el error y corregirlo.

[3]:
```
# Muestra un mensaje en mayúsculas

print("actualización necesaria".upper())
```

```
ACTUALIZACIÓN NECESARIA
```

Pista 1

Para llamar a una función en Python, se deben usar paréntesis tanto de apertura como de cierre.

Pista 2

En el código anterior, revisa que cada llamada a la función tenga paréntesis tanto de apertura como de cierre.

Pregunta 3 ¿Qué ocurre cuando se ejecuta el código antes de modificarlo? ¿Qué está causando el error de sintaxis? ¿Cómo se podría solucionar?

Cuando el código se ejecuta antes de modificarlo, el resultado muestra `SyntaxError: unexpected EOF while parsing` (final de archivo inesperado durante el análisis). Esto se debe a la falta de

paréntesis de cierre al final de la sentencia `print()`. Para solucionar esto, puedes agregar `)` al final de la línea.

1.6 Tarea 4

En la siguiente celda de código, se te proporciona una variable llamada `usernames_list`, otra llamada `username` y un código que determina si el nombre de usuario está aprobado. Hay dos errores de sintaxis y una excepción. Tu tarea es encontrarlos y arreglar el código. Una estrategia de depuración útil es enfocarse en un error a la vez y ejecutar el código después de corregir cada uno.

[4]:
```
# Asigna `usernames_list` a una lista de nombres de usuario que representan␣
↪usuarios aprobados

usernames_list = ["djames", "jpark", "tbailey", "zdutchma", "esmith",␣
↪"srobinso", "dcoleman", "fbautist"]

# Asigna `username` a un nombre de usuario específico

username = "esmith"

# Bucle for que itera sobre los elementos de `usernames_list` y determina si␣
↪cada elemento corresponde a un usuario aprobado

for name in usernames_list:

    # Comprueba si `name` coincide con `username`
    # Si coincide, muestra un mensaje en consecuencia

    if name == username:
        print("El usuario es un usuario aprobado")
```

```
El usuario es un usuario aprobado
```

Pista 1

En Python, el operador de asignación `=` te permite asignar o reasignar un valor a una variable y el operador de comparación `==` te permite comparar un valor con otro (o el valor de una variable con el de otra).

Pista 2

La sangría es importante en la sintaxis de Python. Comprueba que la sangría dentro del bucle `for` y la sangría dentro de la sentencia `if` sean correctas.

Pista 3

Revisa que, cada vez que se usa una variable, se escriba de la misma manera que se escribió cuando se asignó.

Pregunta 4 ¿Qué ocurre cuando se ejecuta el código antes de modificarlo? ¿Qué está causando los errores? ¿Cómo se podría solucionar?

Cuando el código se ejecuta antes de modificarlo, el resultado muestra `SyntaxError: invalid syntax`, ya que es el primer error que encuentra Python en este código. Hay tres problemas en el código: 1. En la condición `if`, se usa el operador de asignación `=` en lugar del operador de comparación `==`, lo cual causa un error de sintaxis. Para solucionar esto, puedes reemplazar `=` por `==`. 2. Dentro de la sentencia `if`, falta la sangría. Esto causa un error de sintaxis. Para solucionarlo, puedes agregar la sangría adecuada antes de la sentencia `print()`. 3. La variable `usernames_list` está mal escrita en la condición del bucle `for`. Allí está escrita como `username_list`, lo que causa una excepción. Para solucionarlo, puedes agregar la `s` faltante en el lugar correcto.

Tarea 5

En esta tarea, examinarás el siguiente código e identificarás el tipo de error que se produce. Luego, ajustarás el código para corregir el error.

[5]:
```
# Asigna `usernames_list` a una lista de nombres de usuario

usernames_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab"]

# Asigna `username` a un nombre de usuario específico

username = "eraab"

# Determina si `username` es el nombre de usuario final en `usernames_list`
# Si lo es, muestra un mensaje en consecuencia

if username == usernames_list[4]:
    print("Este nombre de usuario es el último de la lista.")
```

```
Este nombre de usuario es el último de la lista.
```

Pista 1

Recuerda que la indexación en Python comienza en `0`.

Pista 2

Identifica cuántos elementos hay en `usernames_list`.

Pista 3

Dado que la indexación en Python comienza en `0` y `usernames_list` contiene `5` elementos, identifica qué valor del índice corresponde al elemento final en `usernames_list`.

Pregunta 5 ¿Qué ocurre cuando se ejecuta el código antes de modificarlo? ¿De qué tipo de error se trata? ¿Cómo se podría solucionar?

Cuando el código se ejecuta antes de modificarlo, el resultado muestra `IndexError: list index out of range`. Esto significa que hay un error de índice, causado por un valor de índice no válido que se está utilizando con una lista. Un error de índice es un tipo de excepción en Python. Además,

recuerda que la indexación en Python comienza en `0` y `usernames_list` tiene una longitud de `5`. Así que `4` es el valor del índice correspondiente al elemento final en `usernames_list`, ya que `5` no es un índice válido en `usernames_list`. Para corregir el error, reemplaza `5` por `4`.

1.7 Tarea 6

En esta tarea, examinarás el siguiente código. El código importa un archivo de texto a Python, lee el contenido y lo almacena como una lista en una variable llamada `ip_addresses`. Luego, elimina elementos de `ip_addresses` si están en `remove_list`. Hay tres errores en total en el código: primero un error de sintaxis, luego una excepción relacionada con un método de cadena y, por último, un error lógico. Tu objetivo es encontrar estos errores y corregirlos.

[6]:
```
# Asigna a `import_file` el nombre del archivo de texto

import_file = "allow_list.txt"

# Asigna a `remove_list` una lista de direcciones IP a las que ya no se les␣
↪permite acceder a la red

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.
↪58.57"]

# Sentencia with que lee el archivo de texto y almacena su contenido en␣
↪`ip_addresses`

with open(import_file, "r") as file:
    ip_addresses = file.read()

# Convierte `ip_addresses` de una cadena a una lista

ip_addresses = ip_addresses.split()

# Bucle for que itera sobre los elementos en `ip_addresses`
# y elimina cada elemento que corresponde a una dirección IP a la que ya no se␣
↪le permite el acceso

for element in ip_addresses:
    if element in remove_list:
        ip_addresses.remove(element)

# Muestra `ip_addresses` después del proceso de eliminación

print(ip_addresses)
```

```
␣
↪--------------------------------------------------------------------------
```

```
       FileNotFoundError                         Traceback (most recent call␣
↪last)

       <ipython-input-6-0d86553a6452> in <module>
         9 # Sentencia with que lee el archivo de texto y almacena su contenido␣
↪en `ip_addresses`
        10
   ---> 11 with open(import_file, "r") as file:
        12     ip_addresses = file.read()
        13


       FileNotFoundError: [Errno 2] No such file or directory: 'allow_list.txt'
```

Pista 1

Para una sentencia `with` en Python, deben usarse dos puntos (`:`) al final del encabezado.

Pista 2

El método `.split()` en Python se usa en cadenas para convertirlas en listas. Para llamar al método `.split()`, coloca la cadena que quieras dividir delante de la llamada al método.

Pista 3

Durante cada iteración del bucle `for`, la sentencia `if` debe determinar si el elemento está en la `remove_list`. Verifica a qué lista se hace referencia en la condición `if`.

Pregunta 6 ¿Qué ocurre cuando se ejecuta el código antes de modificarlo? ¿Qué está causando los errores? ¿Cómo se podrían solucionar?

Cuando el código se ejecuta antes de modificarlo, el resultado muestra `SyntaxError: invalid syntax`, ya que es el primer error que encuentra Python en este código. Hay tres errores en el código: 1. Hay un error de sintaxis, ya que al encabezado de la instrucción `with` le falta un `:` al final. Para solucionarlo, puedes agregar `:` allí. 2. Hay una excepción relacionada con el método de cadena `.split()`. Para llamar a este método, debes escribir el nombre de la variable que contiene la cadena que quieres usar, seguida de `.` y, luego, el nombre del método. Para solucionarlo, puedes reemplazar `split.ip_addresses()` por `ip_addresses.split()`. 3. Hay un error lógico en la sentencia `if`. La sentencia `if` debe verificar si cada elemento está en `remove_list`, no en `ip_addresses`. Si un elemento está en `remove_list`, así es como Python sabe cómo eliminar ese elemento de `ip_addresses`. Para solucionarlo, puedes reemplazar `if element in ip_addresses` por `if element in remove_list`.

1.8 Tarea 7

En esta tarea final, hay tres sistemas operativos: OS 1, OS 2 y OS 3. Cada sistema operativo necesita un parche de seguridad para una fecha específica. La fecha del parche para OS 1 es `"1 de`

`marzo"`, la fecha del parche para OS 2 es `"1 de abril"` y la fecha del parche para OS 3 es `"1 de mayo"`.

El siguiente código almacena uno de estos sistemas operativos en una variable llamada `system`. Luego, usa condicionales para generar la fecha del parche para este sistema operativo.

Sin embargo, este código tiene errores lógicos. Tu objetivo es asignar diferentes valores a la variable `system`, ejecutar el código para observar el resultado, identificar el error y corregirlo.

[ ]:
```
# Asigna a `system` un sistema operativo específico como una cadena

system = "OS 2"

# Asigna a `patch_schedule` una lista de fechas de parches en orden de sistema␣
↪operativo

patch_schedule = ["1 de marzo", "1 de abril", "1 de mayo"]

# Sentencia condicional que comprueba qué sistema operativo está almacenado en␣
↪`system` y muestra un mensaje donde figura la fecha del parche␣
↪correspondiente

if system == "OS 1":
    print("Fecha del parche:", patch_schedule[0])

elif system == "OS 2":
    print("Fecha del parche:", patch_schedule[1])

elif system == "OS 3":
    print("Fecha del parche:", patch_schedule[2])
```

Pista 1

Recuerda que la indexación en Python comienza en `0`.

Pista 2

Ten en cuenta que las fechas de parche en `patch_schedule` están en orden de sistema operativo. La primera fecha de parche en `patch_schedule` corresponde al OS 1, la segunda fecha de parche en `patch_schedule` corresponde al OS 2, y así sucesivamente.

Pista 3

Dado que la indexación en Python comienza en `0` y `patch_schedule` está en orden del sistema operativo del OS 1 al OS 3, el valor de índice `0` corresponde a la fecha de parche para el OS 1, el valor de índice `1` corresponde a la fecha de parche para el OS 2, y así sucesivamente.

Pregunta 7 ¿Qué ocurre cuando se ejecuta el código antes de modificarlo? ¿Qué está causando los errores en la lógica? ¿Cómo se podrían solucionar?

Cuando el código se ejecuta antes de modificarlo, se le asigna `"OS 2"` a la variable `system` pero el resultado es `Fecha del parche: 1 de marzo`. Esta no es la fecha correcta del parche para OS 2.

Cuando se le asigna `"OS 1"` a `system`, el resultado es `Fecha del parche: 1 de mayo`. Esta no es la fecha del parche correcta para OS 1.

Estos errores lógicos se deben a los valores de índice incorrectos en la primera y la segunda sentencia `print()` en el código. Ten en cuenta que la indexación en Python comienza en `0` y `patch_schedule` está en orden del sistema operativo del OS 1 al OS 3. Puedes corregir los errores lógicos usando `patch_schedule[0]` para obtener la fecha de parche correcta para OS 1 y `patch_schedule[1]` para obtener la fecha de parche correcta para OS 2.

1.9 Conclusión

¿Qué conclusiones clave obtuviste de este laboratorio?

- La depuración es una práctica esencial que los analistas utilizan para identificar errores en el código y corregirlos, con el fin de garantizar que el código se ejecute sin problemas.
- Python ejecuta el código de arriba hacia abajo y se detiene una vez que se encuentra con un error. Por lo tanto, si hay múltiples errores en una celda de código, el mensaje de error resultante generalmente mostrará el primer error.

- En Python, los tipos más comunes de errores incluyen errores de sintaxis, errores lógicos y excepciones.
  - Los errores de sintaxis se suelen deber a la puntuación, como la falta de `:` al final del encabezado de una sentencia `with` o la falta de `,` entre los elementos de una lista.
  - Los errores lógicos podrían implicar índices incorrectos al acceder a elementos de una lista o hacer referencia a una lista incorrecta en una condición `if`.
  - Las excepciones pueden incluir nombres de variables mal escritos o errores al llamar a los métodos de cadena.
- Una estrategia clave para la depuración es ejecutar el código y observar si produce los resultados previstos. Si el resultado no es correcto o muestra un mensaje de error, úsalo para identificar qué línea o líneas del código podrían estar causando el problema. Después de arreglar el código, es importante que vuelvas a ejecutarlo para asegurarte de que todo funcione según lo previsto.