LAB_014_Activity

December 27, 2024

1 Activity: Debug Python Code

1.1 Introduction

One of the biggest challenges faced by analysts is ensuring that automated processes run smoothly. Debugging is an important practice that security analysts incorporate in their work to identify errors in code and resolve them so that the code achieves the desired outcome.

Through a series of tasks in this lab, you'll develop and apply your debugging skills in Python.

Tips for completing this lab

As you navigate this lab, keep the following tips in mind:

- `### YOUR CODE HERE ###` indicates where you should write code. Be sure to replace this with your own code before running the code cell.
- Feel free to open the hints for additional guidance as you work on each task.
- To enter your answer to a question, double-click the markdown cell to edit. Be sure to replace the "[Double-click to enter your responses here.]" with your own answer.
- You can save your work manually by clicking File and then Save in the menu bar at the top of the notebook.
- You can download your work locally by clicking File and then Download and then specifying your preferred file format in the menu bar at the top of the notebook.

1.2 Scenario

In your work as a security analyst, you need to apply debugging strategies to ensure your code works properly.

Throughout this lab, you'll work with code that is similar to what you've written before, but now it has some errors that need to be fixed. You'll need to read code cells, run them, identify the errors, and adjust the code to resolve the errors.

1.3 Task 1

The following code cell contains a syntax error. In this task, you'll run the code, identify why the error is occuring, and modify the code to resolve it. (To ensure that it has been resolved, run the code again to check if it now functions properly.)

```
[2]: # For loop that iterates over a range of numbers
     # and displays a message each iteration

     for i in range(10)
         print("Connection cannot be established")
```

```
          File "<ipython-input-2-324a3968fb71>", line 4
        for i in range(10)
                          ^
    SyntaxError: invalid syntax
```

Hint 1

The header of a `for` loop in Python requires specific punctuation at the end.

Hint 2

The header of a `for` loop in Python requires a colon (`:`) at the end.

Question 1   What happens when you run the code before modifying it? How can you fix this?

Before correcting the code I receive a SyntaxError: invalid syntax on line 4. I fixed the Syntax error by adding a colon at the end of the header of the for loop.

1.4   Task 2

In the following code cell, you're provided a list of usernames. There is an issue with the syntax. In this task, you'll run the cell, observe what happens, and modify the code to fix the issue.

```
[4]: # Assign `usernames_list` to a list of usernames

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

     # Display `usernames_list`

     print(usernames_list)
```

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

Hint 1

Each element in `usernames_list` is a username and should be a string. In Python, a string should have quotation marks around it.

Hint 2

When creating a list in Python, the elements of the list should be separated with commas. There should be a comma between every two consecutive elements.

Question 2   What happens when you run the code before modifying it? How can you fix it?

Before editing the code I received a SyntaxError on Line 3. Seems to be one of the elements in the list is missing a quotation mark and a comma to split each element in the list.

1.5   Task 3

In the following code cell, there is a syntax error. Your task is to run the cell, identify what is causing the error, and fix it.

```
[7]: # Display a message in upper case

print("update needed".upper()
```

```
      File "<ipython-input-7-b640f5ee0427>", line 3
    print("update needed".upper()
                                 ^
SyntaxError: unexpected EOF while parsing
```

Hint 1

Calling a function in Python requires both opening and closing parantheses.

Hint 2

In the code above, check that each function call has both opening and closing parantheses.

Question 3   What happens when you run the code before modifying it?   What is causing the syntax error? How can you fix it?

Before editing the code I received a SyntaxError: unexpected EOF while parsing. There is a closing parenthesis that is missing from the print() function.

1.6   Task 4

In the following code cell, you're provided a `usernames_list`, a `username`, and code that determines whether the username is approved. There are two syntax errors and one exception. Your task is to find them and fix the code. A helpful debugging strategy is to focus on one error at a time and run the code after fixing each one.

```
[16]: # Assign `usernames_list` to a list of usernames that represent approved users

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

      # Assign `username` to a specific username

      username = "esmith"

      # For loop that iterates over the elements of `usernames_list` and determines
       ↪whether each element corresponds to an approved user

      for name in username_list:

          # Check if `name` matches `username`
          # If it does match, then display a message accordingly

          if name == username:
              print("The user is an approved user")
```

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

        NameError                                 Traceback (most recent call␣
    ↪last)

        <ipython-input-16-beb81205ed64> in <module>
          9 # For loop that iterates over the elements of `usernames_list` and␣
    ↪determines whether each element corresponds to an approved user
         10
    ---> 11 for name in username_list:
         12
         13     # Check if `name` matches `username`

        NameError: name 'username_list' is not defined
```

Hint 1

In Python, the = assignment operator allows you to assign or reassign a variable to a value, and the == comparison operator allows you to compare one value to another (or the value of one variable to the value of another).

Hint 2

Indentation is important in Python syntax. Check that the indentation inside the `for` loop and the indentation inside the `if` statement are correct.

Hint 3

Check that each time a variable is used, it's spelled in the same way it was spelled when it was assigned.

Question 4 What happens when you run the code before modifying it? What is causing the errors? How can you fix it?

Before editing the code I recieved a SyntaxError for Line 16 only 1 equal sign was used to compare 2 variables which requires a second equal sign and must look like "==". Next I received an IndentationError on Line 17 the print function inside of the for loop must be indented further forward after the for loop header. Finally a NameError on Line 11 the name "username_list" is not recognized because it's missing a "s" to match the assigned variable "usernames_list".

1.7 Task 5

In this task, you'll examine the following code and identify the type of error that occurs. Then, you'll adjust the code to fix the error.

```
[20]:  # Assign `usernames_list` to a list of usernames

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

       # Assign `username` to a specific username

       username = "eraab"

       # Determine whether `username` is the final username in `usernames_list`
       # If it is, then display a message accordingly

       if username == usernames_list[4]:
           print("This username is the final one in the list.")
```

This username is the final one in the list.

Hint 1

Recall that indexing in Python starts at 0.

Hint 2

Identify how many elements there are in the `usernames_list`.

Hint 3

Since indexing in Python starts at 0 and the `usernames_list` contains 5 elements, identify which index value corresponds to the final element in `usernames_list`.

Question 5 What happens when you run the code before modifying it? What type of error is this? How can you fix it?

Before editing the code I received an IndexError: list index out of range. Meaning the list index has a smaller amount of elements than 0-5. The list assigned only contains elements in the index positions of 0-4.

1.8 Task 6

In this task, you'll examine the following code. The code imports a text file into Python, reads its contents, and stores the contents as a list in a variable named `ip_addresses`. It then removes elements from `ip_addresses` if they are in `remove_list`. There are two errors in the code: first a syntax error and then an exception related to a string method. Your goal is to find these errors and fix them.

```python
# Assign `import_file` to the name of the text file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addressess that are no longer allowed to
 ↪access the network

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

# With statement that reads in the text file and stores its contents as a list
 ↪in `ip_addresses`

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

# Convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()

# For loop that iterates over the elements in `remove_list`,
# checks if each element is in `ip_addresses`,
# and removes each element that corresponds to an IP address that is no longer
 ↪allowed

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

# Display `ip_addresses` after the removal process

print(ip_addresses)
```

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.6.9',
'192.168.52.90', '192.168.90.124', '192.168.186.176', '192.168.133.188',
```

```
'192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224',
'192.168.60.153', '192.168.69.116']
```

Hint 1

A `with` statement in Python requires a colon (`:`) at the end of the header.

Hint 2

The `.split()` method in Python is used on strings to convert them to lists. To call the `.split()` method, place the string you want to split in front of the method call.

Question 6 What happens when you run the code before modifying it? What is causing the errors? How can you fix them?

Before debugging the code I received a SyntaxError on line 11. The end of the with statement requires a colon ":" that is missing. Next line 16 contains a NameError which is caused by the .split() function being out of place. It should be at the end of the variable not at the beginning.

1.9 Task 7

In this final task, there are three operating systems: OS 1, OS 2, and OS 3. Each operating system needs a security patch by a specific date. The patch date for OS 1 is `"March 1st"`, the patch date for OS 2 is `"April 1st"`, and the patch date for OS 3 is `"May 1st"`.

The following code stores one of these operating systems in a variable named `system`. Then, it uses conditionals to output the patch date for this operating system.

However, this code has logic errors. Your goal is to assign the `system` variable to different values, run the code to examine the output, identify the error, and fix it.

```python
[27]:  # Assign `system` to a specific operating system as a string

       system = "OS 2"

       # Assign `patch_schedule` to a list of patch dates in order of operating system

       patch_schedule = ["March 1st", "April 1st", "May 1st"]

       # Conditional statement that checks which operating system is stored in␣
        ↪`system` and displays a message showing the corresponding patch date

       if system == "OS 1":
           print("Patch date:", patch_schedule[0])

       elif system == "OS 2":
           print("Patch date:", patch_schedule[1])

       elif system == "OS 3":
           print("Patch date:", patch_schedule[2])
```

```
Patch date: April 1st
```

Hint 1

Recall that indexing in Python starts at `0`.

Hint 2

Note that the patch dates in `patch_schedule` are in order of operating system. The first patch date in `patch_schedule` corresponds to OS 1, the second patch date in `patch_schedule` corresponds to OS 2, and so on.

Hint 3

Since indexing in Python starts at `0` and `patch_schedule` is in order of operating system from OS 1 to OS 3, the index value `0` corresponds to the patch date for OS 1, the index value `1` corresponds to the patch date for OS 2, and so on.

Question 7 What happens when you run the code before modifying it? What is causing the logic errors? How can you fix them?

Before debugging the code the logic error occurs because the specified index values for the corresponding patch_schedules were incorrect. OS1 is index 0, OS2 is index 1 and OS3 is index 2.

1.10 Conclusion

What are your key takeaways from this lab?

Debugging or double checking code can be extremely important and helpful skill. Being able to recognize and address error types is a coveted skill for an analyst to have.

```
[ ]:
```