LAB_014_Activity

August 5, 2023

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.)

[4]:
```
# 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")
```

```
Connection cannot be established
Connection cannot be established
Connection cannot be established
Connection cannot be established
Connection cannot be established
Connection cannot be established
Connection cannot be established
Connection cannot be established
Connection cannot be established
Connection cannot be established
```

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 modifying the code I received a SyntaxError:

```
File "<ipython-input-3-324a3968fb71>", line 4     for i in range(10)
^ SyntaxError: invalid syntax
```

I fixed this code by added a colon, `:` to the end 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.

[8]:
```
# 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?

When I ran the code before modifying it, I received a SyntaxError:

```
File "<ipython-input-7-8a568c7729fd>", line 3     usernames_list = ["djames",
"jpark", "tbailey", "zdutchma "esmith", "srobinso", "dcoleman", "fbautist"]
^ SyntaxError: invalid syntax
```

The issue here was a missing comma, `,`, and quotation mark, `"`, between the usernames `"zudtchma"` and `"esmith"`. I fixed this by changing the code from `"zdutchma "esmith",` to `"zdutchma", "esmith",`.

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.

[10]:
```
# Display a message in upper case

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

```
UPDATE NEEDED
```

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?

When I ran the code before modifying it, I received a SyntaxError:

```
File "<ipython-input-9-b640f5ee0427>", line 3     print("update needed".upper()
^ SyntaxError: unexpected EOF while parsing
```

The cause of this syntax error is a closing paranthesis, `)`. To fix this code, I added the paranthesis to the end of 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 usernames_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")
```

```
The user is an approved user
```

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?

When i run the code before modifying it, I receive a SyntaxError:

```
File "<ipython-input-13-8f65398e07e0>", line 16     if name = username:
^ SyntaxError: invalid syntax
```

The error is caused by a missing an equal sign, `=`, in the `if` statement. The statment should be `if name == username:`

The next error I receive is a IndentationError:

```
File "<ipython-input-14-4d910d42e5a5>", line 17     print("The user is an
approved user")         ^ IndentationError: expected an indented block
```

The error can be fixed by indenting the `print()` function that is below the `if` statement.

Another error I received after fixing the first two is a NameError:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-beb81205ed64> in <module>       9 # For loop that iterates over
the elements ofusernames_listand determines whether each element corresponds to an
approved user      10  ---> 11 for name in username_list:      12       13     #
Check ifnamematchesusername‘
```

NameError: name ‘username_list’ is not defined‘

This error is caused by a missing `s` in the `usernames_list` variable in the `for` loop. The typo was `username_list`, fixing this by adding the `s` resolved all errors.

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.

[21]:
```
# 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?

When I run the code before modifying it, I receive a IndexError:

`IndexError: list index out of range`

This error is an exception error as it is not possible to pull the `username` from the 5th spot in the `usernames_list`. There are 5 usernames in the variable, but Python counts start at 0, so the 5th would be index 4. To correct this issue, I changed the `if` statement from `if username == usernames_list[5]:` to `if username == usernames_list[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.

[26]:
```
# 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?

When I run the code before modifying it, I receive a SyntaxError:

```
File "<ipython-input-22-e9bdcbfcb5b3>", line 11     with open(import_file, "r")
as file                                            ^ SyntaxError: invalid syntax
```

The cause of this error is a missing colon, `:`, at the end of the `with` module after the `file` variable.

The next error I receive is a NameError:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-90a1df47b631> in <module>      14 # Convertip_addressesfrom a
string to a list      15  ---> 16 ip_addresses = split.ip_addresses()      17
18 # For loop that iterates over the elements inremove_list‘,
```

NameError: name ‘split’ is not defined‘

The cause of this error is the `split` method is in the `ip_address` variable spot and the `ip_address` variable is in the `split` method spot. The name of the variable must come first, then a period `.`, then the method. To fix this error, I swapped the variable and the method, `ip_addresses = ip_addresses.split()`.

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.

[28]:
```
# 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?

When I run the code before modifying it, I receive output:

```
Patch date: March 1st
```

Upon review, it appears the output is a LogicError. The patch date for OS 1 should be "March 1st", the patch date for OS 2 should be "April 1st", and the patch date for OS 3 is "May 1st". The `"OS 1" if` statement, and `"OS 2" elif` statement have incorrect index values. To fix this logic error, I changed the index value for `print()` function in the `"OS 1" if` statement to `print("Patch date:", patch_schedule[0])` and the `"OS 2" elif` statement to `print("Patch date:", patch_schedule[1])`. This corresponds the correct OS versions with the correct patch schedule, and the new output is:

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

1.10 Conclusion

What are your key takeaways from this lab?

My key takeaway from this lab is that debugging is just as crucial for analysts as it is to know how to code. I have learned so much from this lab, it really makes me think the way that Python reads code. I gained a better understanding in how Python runs code sequentially. When it hits an error, it halts and provides an output error message. These messages will be extremely useful in my career as I will continue to learn from mistakes with instant feedback. I learned from this lab how to identify the three error types in Python: syntax errors (e.g. a missing colon `:`), logic errors (e.g. code works but does not provide expected output), and exception errors (e.g. impossible index). I also learned a good practice in debugging code is to include `print()` lines throughout the code to analyze the output and determine where the code failed. Overall, this was a fun lab and course. I learned a lot of useful and practical information that I know will benefit my career.