LAB_014_Exemplar

April 24, 2025

1 Exemplar: 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.)

[1]:
```python
# 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?

When the code is run before it's modified, the output shows `SyntaxError: invalid syntax`, which indicates that there is a syntax error. The syntax error is caused by the missing `:` at the end of the `for` loop header. To fix this, you can add `:` at that position.

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.

[2]:
```python
# 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 the code is run before it's modified, the output shows `SyntaxError: invalid syntax`. The issue occurred when assigning a value to `usernames_list`. The fourth username is missing a closing quotation mark, and there is a missing comma between the fourth and fifth usernames. Each username in the list should be a string, and commas should be used to separate one username from the next. To fix the syntax error, you can add a closing quotation mark to properly specify the fourth username as a string and then add a comma between the fourth and fifth usernames to separate them. So instead of `"zdutchma "esmith",`, it should say `"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.

```
[3]: # 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 the code is run before it's modified, the output shows `SyntaxError: unexpected EOF while parsing`. This is caused by the missing closing paranthesis at the end of the `print()` statement. To fix this, you can add `)` at the end of the line.

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.

[4]:
```python
# 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 the code is run before it's modified, the output shows `SyntaxError: invalid syntax`, as that's the first error that Python encounters in this code. There are three issues in the code: 1. In the `if` condition, the `=` assignment operator is used instead of the `==` comparison operator, causing a syntax error. To fix this, you can replace `=` with `==`. 2. Inside the `if` statement, indentation is missing, causing a syntax error. To fix this, you can add appropriate indentation before the `print()` statement. 3. The variable `usernames_list` is misspelled in the `for` loop condition. It's spelled as `username_list` there, causing an exception. To fix this, you can add the missing `s` in the appropriate spot.

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.

[5]:
```python
# 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 the code is run before it's modified, the output shows `IndexError: list index out of range`, which means that there is an index error, and it's caused by an invalid index value that is being used with a list. Note that an index error is a type of exception in Python. Also, recall that indexing in Python starts at `0` and the `usernames_list` has a length of `5`. So `4` is the index value corresponds to the final element in `usernames_list`. `5` is not a valid index in `usernames_list`. You can fix the error by replacing `5` with `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.

[1]:
```
# 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 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 the code is run before it's modified, the output shows `SyntaxError: invalid syntax`, as that's the first error that Python encounters in this code. There are two errors in the code: 1. There is a syntax error because the header of the `with` statement is missing a `:` at the end. To fix this, you can add `:` there. 2. There is an exception related to the string method `.split()`. To call this method, you must write the name of the variable that contains the string you want to use, followed by a `.`, and then the name of the method. So to fix, you can replace `split.ip_addresses()` with `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.

[7]:
```
# 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 the code is run before it's modified, the `system` variable is assigned to `"OS 2"`, but the output is `Patch date: March 1st`. This is not the correct patch date for OS 2.

When assigning `system` to `"OS 1"`, the output is `Patch date: May 1st`. This is not the correct patch date for OS 1.

These logic errors are due to the incorrect index values in the first and second `print()` statements in the code. Note that indexing in Python starts at `0` and `patch_schedule` is in order of operating system from OS 1 to OS 3. To fix the logic errors, you can use `patch_schedule[0]` to get the correct patch date for OS 1 and `patch_schedule[1]` to get the correct patch date for OS 2.

1.10 Conclusion

What are your key takeaways from this lab?

- Debugging is an essential practice that analysts use to identify errors in code and fix them to ensure that the code runs smoothly.
- Python executes code from top to bottom and stops once it encounters an error. So if there are multiple errors in a code cell, the outputted error message will typically show the first error.

- In Python, common types of errors include syntax errors, logic errors, and exceptions.
  - Syntax errors often involve punctuation such as a missing `:` at the end of a `with` statement header and a missing `,` between elements in a list.
  - Logic errors could involve incorrect indices when accessing elements from a list.
  - Exceptions could involve misspelled variable names or incorrectly called string methods.
- A key strategy for debugging is running code and examining if it produces the intended results. If the output isn't correct, or if it displays an the error message, use this to identify which line(s) of the code could be causing the issue. After fixing the code, it's important to run it again to ensure that everything works as expected.