Exemplar_Create loops

October 27, 2025

1 Exemplar: Create loops

1.1 Introduction

As a security analyst, some of the measures you take to protect a system will involve repetition. As an example, you might need to investigate multiple IP addresses that have attempted to connect to the network. In Python, iterative statements can help automate repetitive processes like these to make them more efficient.

In this lab, you will practice writing iterative statements 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 that 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

You're working as a security analyst, and you're writing programs in Python to automate displaying messages regarding network connection attempts, detecting IP addresses that are attempting to access restricted data, and generating employee ID numbers for a Sales department.

1.3 Task 1

In this task, you'll create a loop related to connecting to a network.

Write an iterative statement that displays `Connection could not be established` three times. Use the `for` keyword, the `range()` function, and a loop variable of `i`. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[19]:
```
# Iterative statement using `for`, `range()`, and a loop variable of `i`
# Display "Connection could not be established." three times

for i in range(3):
    print("Connection could not be established.")
```

```
Connection could not be established.
Connection could not be established.
Connection could not be established.
```

Hint 1

Use `i` as the loop variable and then place the `in` operator after `i`.

Hint 2

After the `in` operator, pass in the appropriate number to the `range()` function so that it instructs Python to repeat the specified action three times.

1.4 Task 2

The `range()` function can also take in a variable. To repeat a specified action a certain number of times, you can first assign an integer value to a variable. Then, you can pass that variable into the `range()` function within a `for` loop.

In your code that displays a network message connection, incorporate a variable called `connection_attempts`. Assign the positive integer of your choice as the value of that variable and fill in the missing variable in the iterative statement. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell. Test out the code with different values for `connection_attempts` and observe what happens.

[20]:
```
# Create a variable called `connection_attempts` that stores the number of␣
 ↪times the user has tried to connect to the network

connection_attempts = 3

# Iterative statement using `for`, `range()`, a loop variable of `i`, and␣
 ↪`connection_attempts`
# Display "Connection could not be established." as many times as specified by␣
 ↪`connection_attempts`

for i in range(connection_attempts):
    print("Connection could not be established")
```

```
Connection could not be established
Connection could not be established
Connection could not be established
```

Hint 1

Assign the `connection_attempts` variable to a number that represents how many times the user will try to connect to the network.

Hint 2

Pass in the appropriate variable to the `range()` function so that it instructs Python to repeat the specified action the specified number of times.

1.5 Task 3

This task can also be achieved with a `while` loop. Complete the `while` loop with the correct code to instruct it to display `"Connection could not be established."` three times.

In this task, a `for` loop and a `while` loop will produce similar results, but each is based on a different approach. (In other words, the underlying logic is different in each.) A `for` loop terminates after a certain number of iterations have completed, whereas a `while` loop terminates once it reaches a certain condition. In situations where you do not know how many times the specified action should be repeated, `while` loops are most appropriate.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[21]:
```
# Assign `connection_attempts` to an initial value of 0, to keep track of how␣
↪many times the user has tried to connect to the network

connection_attempts = 0

# Iterative statement using `while` and `connection_attempts`
# Display "Connection could not be established." every iteration, until␣
↪connection_attempts reaches a specified number

while connection_attempts < 3:
    print("Connection could not be established")

    # Update `connection_attempts` (increment it by 1 at the end of each␣
↪iteration)
    connection_attempts = connection_attempts + 1
```

```
Connection could not be established
Connection could not be established
Connection could not be established
```

Hint 1

In the condition, use a comparison operator to check whether `connection_attempts` has reached a specific number. This number represents the number of times the message will be displayed.

Hint 2

In the condition, use the `<` comparison operator to check whether `connection_attempts` is less than a specific number. This number represents the number of times the message will be displayed.

Hint 3

Use the `print()` function to display the appropriate message to the user.

Question 1 What do you observe about the differences between the `for` loop and the `while` loop that you wrote?

The messages outputted from both loops were identical. The logic is what differed between the two loops. In the `for` loop, the loop variable `i` was automatically defined in the loop header, and it was updated automatically in each iteration. In the `while` loop, the loop variable `connection_attempts` had to be defined before the loop header, and it had to be explicitly updated inside the loop body.

1.6 Task 4

Now, you'll move onto your next task. You'll automate checking whether IP addresses are part of an allow list. You will start with a list of IP addresses from which users have tried to log in, stored in a variable called `ip_addresses`. Write a `for` loop that displays the elements of this list one at a time. Use `i` as the loop variable in the `for` loop.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[22]:
```
# Assign `ip_addresses` to a list of IP addresses from which users have tried␣
↪to log in

ip_addresses = ["192.168.142.245", "192.168.109.50", "192.168.86.232", "192.168.
↪131.147",
                "192.168.205.12", "192.168.200.48"]

# For loop that displays the elements of `ip_addresses` one at a time

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

```
192.168.142.245
192.168.109.50
192.168.86.232
192.168.131.147
192.168.205.12
192.168.200.48
```

Hint 1

Use `i` as the loop variable and the `in` operator to convey that the specified action should repeat for each element that's in the list `ip_addresses`.

Hint 2

To display the loop variable in every iteration, use the `print()` function inside the `for` loop.

1.7 Task 5

You are now given a list of IP addresses that are allowed to log in, stored in a variable called `allow_list`. Write an `if` statement inside of the `for` loop. For each IP address in the list of IP addresses from which users have tried to log in, display `"IP address is allowed"` if it is among the allowed addresses and display `"IP address is not allowed"` otherwise.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[23]:
```
# Assign `allow_list` to a list of IP addresses that are allowed to log in

allow_list = ["192.168.243.140", "192.168.205.12", "192.168.151.162", "192.168.
 ↪178.71",
              "192.168.86.232", "192.168.3.24", "192.168.170.243", "192.168.119.
 ↪173"]

# Assign `ip_addresses` to a list of IP addresses from which users have tried␣
 ↪to log in

ip_addresses = ["192.168.142.245", "192.168.109.50", "192.168.86.232", "192.168.
 ↪131.147",
                "192.168.205.12", "192.168.200.48"]

# For each IP address in the list of IP addresses from which users have tried␣
 ↪to log in,
# If it is among the allowed addresses, then display "IP address is allowed"
# Otherwise, display "IP address is not allowed"

for i in ip_addresses:
        if i in allow_list:
                print("IP address is allowed")
        else:
                print("IP address is not allowed")
```

```
IP address is not allowed
IP address is not allowed
IP address is allowed
IP address is not allowed
IP address is allowed
IP address is not allowed
```

Hint 1

Use `i` as the loop variable and the `in` operator to convey that the specified action should repeat for each element that's in the list `ip_addresses`.

Hint 2

Make sure that the `if` statement checks whether the user's IP address is in the list of allowed IP

addresses.

Hint 3

Use the `print()` function to display the messages.

1.8 Task 6

Imagine now that the information the users are trying to access is restricted, and if an IP address outside the list of allowed IP addresses attempts access, the loop should terminate because further investigation would be needed to assess whether this activity poses a threat. To achieve this, use the `break` keyword and expand the message that is displayed to the user when their IP address is not in `allow_list` to provide more specifics. Instead of `"IP address is not allowed"`, display `"IP address is not allowed. Further investigation of login activity required"`.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[24]:
```
# Assign `allow_list` to a list of IP addresses that are allowed to log in

allow_list = ["192.168.243.140", "192.168.205.12", "192.168.151.162", "192.168.
 ↪178.71",
              "192.168.86.232", "192.168.3.24", "192.168.170.243", "192.168.119.
 ↪173"]

# Assign `ip_addresses` to a list of IP addresses from which users have tried␣
 ↪to log in

ip_addresses = ["192.168.142.245", "192.168.109.50", "192.168.86.232", "192.168.
 ↪131.147",
                "192.168.205.12", "192.168.200.48"]

# For each IP address in the list of IP addresses from which users have tried␣
 ↪to log in,
# If it is among the allowed addresses, then display "IP address is allowed"
# Otherwise, display "IP address is not allowed"

for i in ip_addresses:
        if i in allow_list:
                print("IP address is allowed")
        else:
                print("IP address is not allowed. Further investigation of␣
 ↪login activity required")
                break
```

```
IP address is not allowed. Further investigation of login activity required
```

Hint 1

Use `i` as the loop variable and the `in` operator to convey that the specified action should repeat for each element that's in the list `ip_addresses`.

Make sure that the `if` statement checks whether the user's IP address is in the list of allowed IP addresses.

Use the `break` keyword to terminate the loop at the appropriate time.

Hint 2

Use the `break` keyword inside the `else` statement after the appropriate message is displayed.

Hint 3

Use the `print()` function to display the messages.

1.9 Task 7

You'll now complete another task. This involves automating the creation of new employee IDs.

You have been asked to create employee IDs for a Sales department, with the criteria that the employee IDs should all be numbers that are unique, divisble by 5, and falling between 5000 and 5150. The employee IDs can include both 5000 and 5150.

Write a `while` loop that generates unique employee IDs for the Sales department by iterating through numbers and displays each ID created.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[25]:
```
# Assign the loop variable `i` to an initial value of 5000

i = 5000

# While loop that generates unique employee IDs for the Sales department by␣
 ↪iterating through numbers
# and displays each ID created

while i <= 5150:
    print(i)
    i = i + 5
```

```
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
```

```
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
```

Hint 1

Use a comparison operator to check whether `i` has reached the upper bound (which is the highest employee ID number allowed). Remember that the employee IDs need to fall between 5000 and 5150.

Make sure to update the value of the loop variable `i` at the end of the loop.

Hint 2

Use the `<=` comparison operator to check whether `i` has reached the upper bound, since the employee IDs need to fall between 5000 and 5150.

At the end of the loop, increment the loop variable by 5. This is because the employee IDs need to be divisble by 5 and the first employee ID is set to 5000.

Hint 3

Use the `<=` comparison operator to check whether `i` has reached 5150, since the employee IDs need to fall between 5000 and 5150.

Use the `print()` function to display the loop variable `i` in each iteration.

Use the `=` assignment operator and the `+` addition operator to increment the value of the loop variable at the end of each iteration.

1.10 Task 8

You would like to incorporate a message that displays `Only 10 valid employee ids remaining` as a helpful alert once the loop variable reaches `5100`.

To do so, include an `if` statement in your code.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

[26]:
```
# Assign the loop variable `i` to an initial value of 5000

i = 5000

# While loop that generates unique employee IDs for the Sales department by␣
 ↪iterating through numbers
# and displays each ID created
# This loop displays "Only 10 valid employee ids remaining" once `i` reaches␣
 ↪5100

while i <= 5150:
    print(i)
    if i == 5100:
        print("Only 10 valid employee ids remaining")
    i = i + 5
```

```
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
Only 10 valid employee ids remaining
5105
5110
5115
5120
5125
```

```
5130
5135
5140
5145
5150
```

Hint 1

Use a comparison operator to check whether `i` has reached `5100`.

Hint 2

Use the `==` comparison operator to check whether `i` has reached `5100`.

Hint 3

Use the `print()` function to display the message.

Question 2 Why do you think the statement `print(i)` is written before the conditional rather than inside the conditional?

In this code, I learned how that it must print out every employee ID number with a for loop, where variable `i` represents the ID number created in each iteration of the loop. The `print(i)` is written before the conditional, so that the loop is displayed in every iteration, otherwise, if `print(i)` was written inside the conditional, the loop variable would only be printed out when it's equal to 5100. This is beucase the condition in the `if` statement is `i == 5100` then the output is dmeosntrated of what I am tyring to explain…

1.11 Conclusion

What are your key takeaways from this lab? * Iterative statements play a major role in automating security-related processes that need to be repeated.
* You can `for` loops allow you to repeat a process a specified number of times. * You can use `while` loops allow you to repeat a process until a specified condition has been met. Comparison operators are often used in these conditions. * The `<` comparison operator allows you to check whether one value is less than another. * The `<=` comparison operator allows you to check whether one value is less than or equal to another. * The `==` comparison operator allows you to check whether one value is equal to another.

In this lab, I learned the importance of using iterative statements, like `for` and `while` loops, to automate repetitive tasks in programming. These loops are especially helpful in security analysis, as they allow you to automate tasks like checking multiple network connection attempts or verifying IP addresses against a list of allowed addresses. The `for` loop is useful when you know the exact number of iterations, while the `while` loop is better suited for situations where the loop runs until a specific condition is met. The lab also highlighted the use of comparison operators in controlling the flow of these loops, such as `<`, `<=`, and `==`. Lastly, using `break` to exit a loop early, as well as incorporating useful messages (like alerting when only 10 employee IDs remain), helped me understand how to add more meaningful interactions within the loop logic. Overall, these concepts make it easier to handle repetitive tasks efficiently and apply logic to automate real-world scenarios.

[ ]: