Activity_Create loops

February 20, 2024

1 Activity: 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.

[1]:
```
for i in range(3):
    print("Connection could not be established.") #code i used to implment the␣
 ↪print function to print the value of i 3 times
```

```
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.

[3]:
```
# Assign a positive integer value to the variable connection_attempts
connection_attempts = 5  # You can choose any positive integer value

# Iterative statement using `for`, `range()`, and a loop variable of `i`
# Display "Connection could not be established." based on the value of␣
 ↪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.
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.

[5]:
```
# 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

# Set the specified number of attempts
max_attempts = 3

# 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 < max_attempts:
    print("Connection could not be established.")

    # Update `connection_attempts` (increment it by 1 at the end of each␣
↪iteration)
    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 primary difference between the `for` loop and the `while` loop lies in their control flow and termination conditions.

1. **`for` Loop:**
   - The `for` loop is used when you know in advance how many times you want to iterate.
   - It operates over a sequence (e.g., `range()` in Python) and runs a specific number of times.
   - In the provided example, the `for` loop iterates a specified number of times (controlled by the `range()` function).
2. **`while` Loop:**
   - The `while` loop is used when you want to repeat a block of code until a certain condition is no longer true.
   - It keeps executing as long as the specified condition remains true.
   - In the provided example, the `while` loop continues to iterate until the `connection_attempts` variable reaches the specified `max_attempts`.

In summary, the `for` loop is suitable when the number of iterations is known in advance, and the `while` loop is more appropriate when you need to repeat a task until a specific condition is met, and you might not know the exact number of iterations beforehand. This helps the code run better to execute the operations faster.

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.

[6]:
```
# 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.

[7]:
```
# 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 ip_address in ip_addresses:
    if ip_address in allow_list:
        print(f"{ip_address} is allowed")
    else:
        print(f"{ip_address} is not allowed")
```

```
192.168.142.245 is not allowed
192.168.109.50 is not allowed
192.168.86.232 is allowed
192.168.131.147 is not allowed
```

```
192.168.205.12 is allowed
192.168.200.48 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 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.

[9]:
```
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"]

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. Further investigation of login␣
↪activity required"

for ip in ip_addresses:
    if ip in allow_list:
        print(f"{ip} is allowed")
    else:
        print(f"{ip} is not allowed. Further investigation of login activity␣
↪required")
```

```
        break
```

```
192.168.142.245 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, divisible 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.

[10]:
```
# 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:
    # Check if the current value of i is divisible by 5
    if i % 5 == 0:
        # Display the generated employee ID
        print(f"Generated Employee ID: {i}")
    # Increment the value of i for the next iteration
    i += 1
```

```
Generated Employee ID: 5000
Generated Employee ID: 5005
```

```
Generated Employee ID: 5010
Generated Employee ID: 5015
Generated Employee ID: 5020
Generated Employee ID: 5025
Generated Employee ID: 5030
Generated Employee ID: 5035
Generated Employee ID: 5040
Generated Employee ID: 5045
Generated Employee ID: 5050
Generated Employee ID: 5055
Generated Employee ID: 5060
Generated Employee ID: 5065
Generated Employee ID: 5070
Generated Employee ID: 5075
Generated Employee ID: 5080
Generated Employee ID: 5085
Generated Employee ID: 5090
Generated Employee ID: 5095
Generated Employee ID: 5100
Generated Employee ID: 5105
Generated Employee ID: 5110
Generated Employee ID: 5115
Generated Employee ID: 5120
Generated Employee ID: 5125
Generated Employee ID: 5130
Generated Employee ID: 5135
Generated Employee ID: 5140
Generated Employee ID: 5145
Generated Employee ID: 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.

[11]:
```
# 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)
    # Check if `i` is 5100 to display the alert
    if i == 5100:
        print("Only 10 valid employee IDs remaining. Hurry up!")
    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. Hurry up!
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?

The statement `print(i)` is written before the conditional in this specific code because the goal is to display the current value of `i` in every iteration of the loop, regardless of whether the condition is met or not.

In the given context, the loop is responsible for generating unique employee IDs and displaying them. The `print(i)` statement is placed before the conditional, ensuring that the employee ID is printed for every iteration. Then, the conditional is used to check if the current value of `i` is equal to 5100, and if so, an additional message is printed.

If `print(i)` were placed inside the conditional, it would only be executed when the condition is true (i.e., when `i` is 5100). This would result in printing the employee ID and the alert message only for that specific iteration, and not for the other iterations of the loop.

By placing `print(i)` outside the conditional, the code ensures that the employee ID is printed for every iteration, providing a clear display of the generated IDs as the loop progresses.

1.11 Conclusion

What are your key takeaways from this lab?

Here are the key takeaways from this lab:

1. **Looping Constructs:** Understanding how to use loops, such as `while` loops, is crucial for repetitive tasks. Loops allow you to iterate through a sequence of statements until a certain condition is met.
2. **Conditional Statements:** Using `if` statements inside loops allows you to control the flow of the program based on specific conditions. This is useful for executing different code blocks under different circumstances.
3. **Variable Manipulation:** In this lab, the loop variable `i` was used for iteration, and its value was manipulated inside the loop to achieve the desired behavior.
4. **Break Statement:** The `break` statement was introduced to terminate the loop when a certain condition was met. This is useful for stopping further iterations once a specific point is reached.
5. **Message Display:** Incorporating informative messages within the loop, such as displaying generated employee IDs and providing alerts, enhances the user experience and communication of program progress.
6. **Code Structure:** Proper indentation and code organization are essential for readability and maintaining code clarity. Consistent indentation helps in understanding the structure of loops and conditional blocks.
7. **Problem-Solving Skills:** The lab involved solving a problem statement related to generating unique employee IDs, requiring logical thinking and consideration of specific conditions.

Overall, this lab provided hands-on experience in writing Python code involving loops, conditionals, and variable manipulation to achieve a specific goal to understand situational means to solve complex issues for coding.

[ ]: