New Page

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.5/latest.js?config=TeX-
AMS_HTML"></script>
<!-- MathJax configuration -->
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
    tex2jax: {
        inlineMath: [ ['$','$'], ["\\(","\\)"] ],
        displayMath: [ ['$$','$$'], ["\\[","\\]"] ],
        processEscapes: true,
        processEnvironments: true
    },
    // Center justify equations in code and markdown cells. Elsewhere
    // we use CSS to left justify single line equations in code cells.
    displayAlign: 'center',
    "HTML-CSS": {
        styles: {'.MathJax_Display': {"margin": 0}},
        linebreaks: { automatic: true }
    }
});
</script>
<!-- End of mathjax configuration --></head>
```

Activity: Create loops¶

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.

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.

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.

In [1]:

```
# 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.")

```
</div>
```

```
<div class="prompt"></div>
```

```
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.

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.

In [2]:

```
# Create a variable called `connection_attempts` that stores the number of times the user has tr
```

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")

```
</div>
```

```
<div class="prompt"></div>
```

```
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.

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.

In [5]:

```
# Assign `connection_attempts` to an initial value of 0, to keep track of how many times the use
```

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: connection_attempts = connection_attempts + 1

```
<span class="c1"># Update `connection_attempts` (increment it by 1 at the end of each
iteration) </span>
<span class="nb">print</span><span class="p">(</span><span class="s2">&quot;Connection could
not be established&quot;</span><span class="p">)</span>
```

```
</div>
```

```
<div class="prompt"></div>
```

```
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 while loops go untill false, and for loops are for, for loops go untill it runs out of variables in the range

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.

In [10]:

```
# 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 ip_adress in ip_addresses: print(f"{ip_adress}")

```
</div>
```

```
<div class="prompt"></div>
```

```
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.

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.

In [11]:

```
# 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_adress in ip_addresses: if ip_adress in allow_list: print(f"{ip_adress} allowed") else: print(f"{ip_adress} not allowed")

```
</div>
```

```
<div class="prompt"></div>
```

```
192.168.142.245 not allowed
192.168.109.50 not allowed
192.168.86.232 allowed
192.168.131.147 not allowed
192.168.205.12 allowed
192.168.200.48 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.

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.

In [12]:

```
# 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_adress in ip_addresses: if ip_adress in allow_list: print(f"{ip_adress} allowed") else: print(f"{ ip_adress} not allowed")

```
</div>
```

```
<div class="prompt"></div>
```

```
192.168.142.245 not allowed
192.168.109.50 not allowed
192.168.86.232 allowed
192.168.131.147 not allowed
192.168.205.12 allowed
192.168.200.48 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`.

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.

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.

In [1]:

```
# 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

Generate employee IDs while i <= 5150: if i % 5 == 0: # Check divisibility by 5 print(i) i += 1

```
</div>
```

```
<div class="prompt"></div>
```

```
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.

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.

In [ ]:

```
# 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

i = 5000 while i <= 5150: print(i) if i == 5100: #the better thing is to know the remaining number print("Only 10 valid employee ids remaining") i += 5

```
</div>
```

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?

because in the loop we need to print the id before because its being used, then if that id is at 5100 it tells us that theres only 10 left not before. if we were to print it before printing the id we would have to say only 10 valid ids left then use up the spot and that doesnt make sense.

Conclusion¶

What are your key takeaways from this lab?

for loops are probs rly good for automating tasks

the the english way of saying how it works is, for variable in range: loop untill pass or variable = range.

additionally the while loops are used when something is true you loop untill its false

---

Revision #1
Created 2023-12-18 02:45:21 UTC by naruzkurai
Updated 2023-12-19 03:38:35 UTC by naruzkurai