Activity_Create loops

April 8, 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, I 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.

1.2 Scenario

I'm working as a security analyst, and am 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

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

I will write an iterative statement that displays `Connection could not be established` three times. I will use the `for` keyword, the `range()` function, and a loop variable of `i`.

[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.")
```

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

1.4 Task

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

function within a `for` loop.

In my code that displays a network message connection, I'll incorporate a variable called `connection_attempts`. Assign the positive integer as the value of that variable and fill in the missing variable in the iterative statement.

[3]:
```
# Create a variable called `connection_attempts` that stores the number of␣
 ↪times the user has tried to connect to the network

connection_attempts = 5

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

1.5 Task

This task can also be achieved with a `while` loop.

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 one does not know how many times the specified action should be repeated, `while` loops are most appropriate.

[8]:
```
# 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 < 5:
    print("Connection could not be established.")
    connection_attempts = connection_attempts + 1
```

```
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.
```

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

[The while loop requires a condition to be specified. If the condition evaluates to True, the loop runs. The for loop does not require a condition.]

1.6 Task

Now I'll automate checking whether IP addresses are part of an allow list. I 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. I Used `i` as the loop variable in the `for` loop.

[9]:
```
# 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
```

1.7 Task

I am now given a list of IP addresses that are allowed to log in, stored in a variable called `allow_list`. I'll 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.

[14]:
```
# 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.
```

1.8 Task

The information the users are trying to access is now 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, I 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"`.

[17]:
```
# 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 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.
```

1.9 Task

The next task involves automating the creation of new employee IDs.

I 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.

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

[20]:
```
# 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("New ID", i)
    i = i + 5
```

```
New ID 5000
New ID 5005
New ID 5010
New ID 5015
New ID 5020
New ID 5025
New ID 5030
New ID 5035
New ID 5040
New ID 5045
New ID 5050
New ID 5055
New ID 5060
New ID 5065
New ID 5070
New ID 5075
New ID 5080
New ID 5085
New ID 5090
New ID 5095
New ID 5100
New ID 5105
New ID 5110
New ID 5115
New ID 5120
New ID 5125
New ID 5130
New ID 5135
New ID 5140
New ID 5145
```

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.

[22]:
```
# Assign the loop variable `i` to an initial value of 5000

i = 5000
```

```python
# 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
```

1.11 Conclusion

What are your key takeaways from this lab?

[I learnt that Iterative statements play a major role in automating security-related processes that need to be repeated. I learnt that you can use `for` loops allow you to repeat a process a specified number of times and `while` loops allow you to repeat a process until a specified condition has been met. Comparison operators are often used in these conditions.]