

# 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

- 
- 
- 
- 
- 

## 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 [2]:

```
# 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

Hint 2

## 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 [3]:

```
# Create a variable called `connection_attempts` that stores the number of  
times the user has tried to connect to the network  
  
connection_attempts = 7  
  
# 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

Connection could not be established

Connection could not be established

Hint 1

Hint 2

## 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 [4]:

```
# 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**

**Hint 2**

**Hint 3**

**Question 1**

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

In the while loop, we have to explicitly define how the loop variable changes in the body of the loop.

## 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 [5]:

```
# 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

Hint 2

## 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 [6]:

```
# 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**

**Hint 2**

### Hint 3

## 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 [10]:  
  
# 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

Hint 2

Hint 3

## 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 [13]:

```
# 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:  
    if i >= 5000:  
        print(i)  
        i=(i+5)  
    else:  
        break
```

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**

**Hint 2**



### Hint 3

## 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 [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
# 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?**

So it also includes the result that doesn't match the condition.

## Conclusion

**What are your key takeaways from this lab?** distinction between for and while loop incorporate conditional statements and logical operators with loops

## Exemplar: 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

- 
- 
- 
- 
- 

### 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.")
```

Connection could not be established.

Connection could not be established.

Connection could not be established.

Hint 1

Hint 2

## 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 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

Hint 2

## 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 [3]:

```
# 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

Hint 2

Hint 3

### 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.

## 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 [4]:  
  
# 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

Hint 2

## 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 [5]:  
# 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**

**Hint 2**

**Hint 3**

## 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 [6]:

```
# 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

Hint 2

Hint 3

## 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 [7]:

```
# 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

Hint 2

Hint 3

## 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 [8]:

```
# 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

Hint 2

Hint 3

### Question 2

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

The goal is to display every employee ID number that's created, and the loop variable `i` represents the ID number created in each iteration of the loop. The statement `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. (Since the condition in the `if` statement is `i == 5100`.)

## 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.