Get started with the course

Introduction to Python programming in cybersecurity

Core Python components

Conditional and iterative statements

Video: Conditional statements in Python
8 min

Reading: More on conditionals in Python
20 min

Lab: Activity: Create a conditional statement
50 min

Lab: Exemplar: Create a conditional statement
20 min

Video: For loops
5 min

Video: While loops
5 min

Reading: More on loops in Python
30 min

Ungraded Plugin: Identify: Select the correct iterative statement
10 min

Lab: Activity: Create loops
40 min

Lab: Exemplar: Create loops
20 min

Practice Quiz: Test your knowledge: Conditional and iterative statements
4 questions

Review: Introduction to Python

More on conditionals in Python

Previously, you explored conditional statements and how they're useful in automating tasks in Python. So far, you've focused on the `if` and `else` keywords. In this reading, you'll review these and learn another keyword, `elif`. You'll also learn how you can apply the `and`, `or`, and `not` operators to your conditions.

How conditional statements work

A **conditional statement** is a statement that evaluates code to determine whether it meets a specific set of conditions. When a condition is met, it evaluates to a Boolean value of `True` and performs specified actions. When the condition isn't met, it evaluates a Boolean value of `False` and doesn't perform the specified actions.

In conditional statements, the condition is often based on a comparison of two values. This table summarizes common comparison operators used to compare numerical values.

operator	use
`>`	greater than
`<`	less than
`>=`	greater than or equal to
`<=`	less than or equal to
`==`	equal to
`!=`	not equal to

Note: The equal to (`==`) and not equal to (`!=`) operators are also commonly used to compare string data.

if statements

The keyword `if` starts a conditional statement. It's a necessary component of any conditional statement. In the following example, `if` begins a statement that tells Python to print an `"OK"` message when the HTTP response status code equals `200`:

```
if status == 200:

    print("OK")
```

This code consists of a header and a body.

The header of an if statement

The first line of this code is the header. In the header of an `if` statement, the keyword `if` is followed by the condition. Here, the condition is that the `status` variable is equal to a value of `200`. The condition can be placed in parentheses:

```
if (status == 200):

    print("OK")
```

In cases like this one, placing parentheses around conditions in Python is optional. You might want to include them if it helps you with code readability. However, this condition will be processed the same way if written without parentheses.

In other situations, because Python evaluates the conditions in parentheses first, parentheses can affect how Python processes conditions. You will read more about one of these in the section of this reading on `not`.

Note: You must always place a colon (`:`) at the end of the header. Without this syntax, the code will produce an error.

The body of an if statement

After the header of an `if` statement comes the body of the `if` statement. This tells Python what action or actions to perform when the condition evaluates to `True`. In this example, there is just one action, printing `"OK"` to the screen. In other cases, there might be more lines of code with additional actions.

Note: For the body of the `if` statement to execute as intended, it must be indented further than the header. Additionally, if there are multiple lines of code within the body, they must all be indented consistently.

Continuing conditionals with else and elif

In the previous example, if the HTTP status response code was not equal to `200`, the condition would evaluate to `False` and Python would continue with the rest of the program. However, it's also possible to specify alternative actions with `else` and `elif`.

else statements

The keyword `else` precedes a code section that only evaluates when all conditions that precede it within the conditional statement evaluate to `False`.

In the following example, when the HTTP response status code is not equal to `200`, it prints an alternative message of `"check other status"`:

```
if status == 200:

    print("OK")

else:

    print("check other status")
```

Note: Like with `if`, a colon (`:`) is required after `else`, and the body that follows the `else` header is indented.

elif statements

In some cases, you might have multiple alternative actions that depend on new conditions. In that case, you can use `elif`. The `elif` keyword precedes a condition that is only evaluated when previous conditions evaluate to `False`. Unlike with `else`, there can be multiple `elif` statements following `if`.

For example, you might want to print one message if the HTTP response status code is `200`, one message if it is `400`, and one if it is `500`. The following code demonstrates how you can use `elif` for this:

```
if status == 200:

    print("OK")

elif status == 400:

    print("Bad Request")

elif status == 500:

    print("Internal Server Error")
```

Python will first check if the value of `status` is `200`, and if this evaluates to `False`, it will go onto the first `elif` statement. There, it will check whether the value of `status` is `400`. If that evaluates to `True`, it will print `"Bad Request"`, but if it evaluates to `False`, it will go on to the next `elif` statement.

If you want the code to print another message when all conditions evaluate to `False`, then you can incorporate `else` after the last `elif`. In this example, if it reaches the `else` statement, it prints a message to check the status:

```
if status == 200:

    print("OK")

elif status == 400:

    print("Bad Request")

elif status == 500:

    print("Internal Server Error")

else:

    print("check other status")
```

Just like with `if` and `else`, it's important to place a colon (`:`) after the `elif` header and indent the code that follows this header.

Note: Python processes multiple `elif` statements differently than multiple `if` statements. When it reaches an `elif` statement that evaluates to `True`, it won't check the following `elif` statements. On the other hand, Python will run all `if` statements.

Logical operators for multiple conditions

In some cases, you might want Python to perform an action based on a more complex condition. You might require two conditions to evaluate to `True`. Or, you might require only one of two conditions to evaluate to `True`. Or, you might want Python to perform an action when a condition evaluates to `False`. The operators `and`, `or`, and `not` can be used in these cases.

and

The `and` operator requires both conditions on either side of the operator to evaluate to `True`. For example, all HTTP status response codes between `200` and `226` relate to successful responses. You can use `and` to join a condition of being greater than or equal to `200` with another condition of being less than or equal to `226`:

```
if status >= 200 and status <= 226:

    print("successful response")
```

When both conditions are `True`, then the `"successful response"` message will print.

or

The `or` operator requires only one of the conditions on either side of the operator to evaluate to `True`. For example, both a status code of `100` and a status code of `102` are informational responses. Using `or`, you could ask Python to print an `"informational response"` message when the code is either `100` or `102`:

```
if status == 100 or status == 102:

    print("informational response")
```

Only one of these conditions needs to be met for Python to print the message.

not

The `not` operator negates a given condition so that it evaluates to `False` if the condition is `True` and to `True` if it is `False`. For example, if you want to indicate that Python should check the status code when it's something outside of the successful range, you can use `not`:

```
if not(status >= 200 and status <= 226):

    print("check status")
```

Python first checks whether the value of status is greater than or equal to `200` and less than or equal to `226`, and then because of the operator `not`, it inverts this. This means it will print the message if `status` is less than `200` or greater than `226`.

Note: In this case, the parentheses are necessary for the code to apply `not` to both conditions. Python will evaluate the conditions within the parentheses first. This means it will first evaluate the conditions on either side of the `and` operator and then apply `not` to both of them.

Key takeaways

It's important for security analysts to be familiar with conditional statements. Conditional statements require the `if` keyword. You can also use `else` and `elif` when working with conditionals to specify additional actions to take. The logical operators `and`, `or`, and `not` are also useful when writing conditionals.

Mark as completed

Like Dislike Report an issue