Conditional Logic

Lesson Duration: 60 minutes

Learning Objectives

- Understand what conditional logic is
- Understand why we use conditional logic in our code
- Understand how to check for equality
- Be able to use conditional operators
    - `==` (is equal to)
    - `<` (is less than)
    - `>` (is greater than)
    - `>=` (is greater than or equal to)
    - `<=` (is less than or equal to)

- Be able to use an `if` statement
- Be able to use an `else` clause in an `if` statement
- Be able to use an `elif` clause in an `if` statement
- Be able to check multiple conditions using `and` and `or`

Intro

Every day we make decisions which effect what we do. If we didn't then we'd do exactly the same thing every day, and in every situation, and life would get really repetitive.

Often, these decisions we make in our daily lives are made based on a 'yes' or 'no' question. When we think about these questions and come to a yes / no answer we are evaluating them. We take in a statement about the world and then give a yes / no answer. We then make a decision as to what to do based on that answer.

> Think about the actions you might do - for example, you decide your outfit based on weather conditions!

Computer programs work in exactly the same way. We can give them a "statement" just like what we have been doing, and it will "evaluate" it to true or false (yes or no). Our programs can then take different paths depending on the condition - this is why such statements are called *conditional statements*. If we could not do this, then all our programs would behave in exactly the same way, every time we run them.

We can try to convert question statements to boolean expressions. (true or false statements) Open-ended questions cannot be converted to booleans, we need to be making a yes or no decision.

> Try thinking about the following questions - can these be translated into boolean values?

```
1) Are you aged over 18?
2) Where were you born?
3) Is it sunny today?
4) Have you been to Rome?
5) Is the number 7 an even number?
6) What is your favourite colour?
```

```
1) I am aged over 18
2) Can't make it boolean
3) It is sunny today
4) I have been to Rome
7) 7 is an even number
6) Can't make boolean
```

Checking for Equality

One form of conditional statement in programming is to check if two values are the same. If they are the same, the result is true, otherwise it is false.

In Python we use a special operator to check if things are equal. We cannot use the `=` operator for this - if you remember, `=` on its own *assigns* a variable. Two `==` together checks if two things are equal.

Let's create a file in the same `session_1` folder called `conditionals.py`!

Type in the following, then try running it by pressing the play button!

```
print(4 == 2 + 2)
# => True

print(4 == 2 + 3)
# => False

print("cat" == "cat")
# => True

print("cat" == "dog")
# => False
```

Note that Python requires both the type *and* value to be the same for two things to be considered equal.

```
"1" == 1
# => False
```

If, however, we encounter this situation, and we have to compare 2 different types of data, we can do so by transforming one of them to match the others type!

```
print("1" == str(1))
print(int("1") == 1)
```

Conditional Operators

The equality or `==` operator belongs to a group of operators which are used for checking conditions. These are collectively known as conditional operators.

`>` - Is Greater Than

We use this operator to evaluate if the object on the left is greater than the object on the right.

```
print(5 > 4)
# => True
print(9 > 100)
# => False
```

`<` - Is Less Than

We use this operator to evaluate if the object on the left is less than the object on the right.

```
print(99 < 100000)
# => True

print(100 < 50)
# => False
```

`>=` and `<=`

These two operators, "greater than or equal to" (`>=`) and "less than or equal to" (`<=`) are variants of the greater than and less than operators, which also evaluate to `true` if the values are equal, unlike the greater than and less than operators themselves.

```
print(100 < 100)
# => False

print(100 <= 100)
# => True
```

Control Flow

Similarly, we might make decisions based on whether something is true or false.

> If I get a new job offer, I'll celebrate. If I feel exhausted, I will not drive.

Programming languages also do this to allow us to have different actions and paths our code can go down, otherwise our program could only do one thing in a linear fashion. This ability to control the flow of a program based on changing conditions is one of the most important, and most powerful features of programming languages.

`if` Statement

We are going to write a small program which asks the user to enter which animal is their favourite and the program will display a response depending on what the user enters. This way we can introduce you to gathering input from the user through the terminal!

Let's create another file called `favourite_animal.py` !

Pseudocode

Pseudocode is English-like language intended for human reading rather than machine reading. When we're solving a problem, often it helps us organise the steps involved by writing them out in a non-code language.

Here's an example for our favourite animal program:

```
# prompt the user the questions which one is their favourite animal?
# gather input from the user, save it into a variable
# if that value is the capybara, we print out an approving message
# end the code
```

So let's get the user to enter which animal they are.

```
animal = input("What is your favourite animal?")
```

We are now going to check the value of `animal` and if it is "capybara" we want the program to print out the message "This is my favourite animal!".

To do this we are going to use an `if` statement, which is the simplest form of Python control flow statement.

To create an `if` statement in Python we begin by writing the `if` keyword. This is followed by the condition, the statement we want to evaluate to true or false, followed by a colon (`:`);

On the next line(s) we write the code we want to execute if the condition is true. This code is indented so that we can distinguish this code from the rest of the code in the program.

```
animal = input("What is your favourite animal?")

if animal == "capybara":
    print("This is my favourite animal!")
```

A Note on Code Blocks and Indentation

Note that we indent the code as we take actions depending on the outcome of the `if`.

Python relies on indentation (whitespace at the beginning of a line) to define lines of code which are grouped together. Other programming languages often use curly-brackets for this purpose. To indicate a block of code in Python, you must indent(add whitespace to the start of the line) each line of the block by the same amount. The two blocks of code in our example `if` statement are both indented by four spaces, which is a typical amount of indentation for Python.

The `else` Clause

So our program now does something if our condition is true, but we might want to do something different if our condition is false. It's like saying:

```
IF the condition is true THEN do X
OTHERWISE do Y
```

In Python, rather than saying 'otherwise' we use the `else` keyword. The code inside the `else` section will be run if the `if` condition is `false`.

So in our program, if the user input is not "chicken" then we are going to get the program to display "Not bad, but not my favourite."

To add an `else` section we add an *unindented* line with the `else` keyword followed by a colon (`:`). On the next line(s) we write the code we want to execute for the 'else' section. Like the code in the `if` section, this code is indented.

Let's add to our pseudocode.

```
# prompt the user the questions which one is their favourite animal?
# gather input from the user, save it into a variable
# if that value is the capybara, we print out an approving message
# if its something else, we display a different message
# end the code
```

```
animal = input("What is your favourite animal?")

if animal == "capybara":
    print("This is my favourite animal!")
else:
    print("Not bad, but not my favourite")
```

Adding Other Conditions

At the moment we only check the input for one value, "chicken", but what if we want to print out a different message if the user enters "kitten"?

One way to do it would be to add another `if` statement to the code in the `else` section.

```
animal = input("What is your favourite animal?")

if animal == "capybara":
    print("This is my favourite animal!")
else:
    if animal == "kitten":
        print("That's pretty cool too!")
    else:
        print("Not bad, but not my favourite")
```

This can get very messy, especially if we want to add further conditions but thankfully Python has another clause which we can add to an `if` statement - the `elif` (short for 'else if') clause. We add this after the `if` section, but *before* the `else` section. Like `if`, `elif` is followed by the condition we want to check, then the code we want to run if the condition is true, so if we wanted to check that the user had entered `kitten` we could add an `elif` clause to our program:

```
animal = input("What is your favourite animal?")

if animal == "capybara":
    print("This is my favourite animal!")
elif animal == "kitten":
    print("That's pretty cool too!")
else:
    print("Not bad, but not my favourite")
```

We can add as many `elif` clauses as we need, as long as they are added before the `else` clause.

Let's make a little program that asks you to guess when Python was created. Take 10 minutes to try to build it, then continue the video or check the solution!

> Hint: `input()` is going to return to you a `str` type, so to compare it to an `int`, you have to use `int()`!

```
guess = input("When was Python created? ")

if int(guess) == 1991:
    print("Whoo you win!")
else:
    print("Nope")
```

Control Flow Nice to Haves

The Ternary Operator

Create a new file called `control_flow_extras.py`

If you have an `if...else` statement where you have only one statement to execute for `if` and only on statement to execute for `else`, we can simplify our code!

```
score = 6

if score > 5:
    result = "pass"
else:
    result = "fail"

print(result)
```

We can put the statement all on the same line

```
score = 6

result = "pass" if score > 5 else "fail"

print(result)
```

Checking Multiple Conditions

Sometimes we want to check if one or more conditons are true. To do this we can use **logical** operators.

```
zsolt_hungry = True
zsolt_tired = True
```

`and`

Let's say we want to compare *both* `zsolt_hungry` *and* `zsolt_tired`. If they are *both* `true` then we want to print the message "Don't talk to me!". We can attach more than one condition to the same `if` statement using the `and` operator. For the `and` operator to return `true` *both* conditions need to evaluate to `true`

```
zsolt_hungry = True
zsolt_tired = True

if zsolt_hungry and zsolt_tired:
    print("Don't talk to me!")
```

`or`

So rather than printing "Jarrod is hangry!" if both `jarrod_hungry` and `jarrod_tired` are true, let's say we want to print "Jarrod is grumpy!" if either `jarrod_hungry` is `true` OR `jarrod_tired` is `true` OR both `jarrod_hungry` and `jarrod_tired` are `true`. Rather than using the `and` operator, we can use the `or` operator:

```
zsolt_hungry = True
zsolt_tired = True

if zsolt_hungry and zsolt_tired:
    print("Don't talk to me!")

zsolt_tired = False # ADDED
if zsolt_tired or zsolt_hungry:
    print("I might need food, or I might need rest, but not both")
```

Conclusion

We have seen that in our programs we can create expressions that evaluate to `true` or `false`. We have seen that we can use these expressions in `if` statements to control the flow our programs.