

INF1-IP

Introduction to Programming

Lecture 3 – Conditions and logic



> Comparison operators and Booleans

So far, we have seen:

- *Assignment* operators: =, +=, etc.
- *Arithmetic* operators: +, -, etc.

There are also *comparison* operators which return *Boolean* values (i.e., **true** or **false**):

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

> Comparison operators and Booleans

Note that in maths, `=` is a *comparator* which checks whether two things are equal, whereas in Python (and most languages) it indicates assignment.

To check if two things are equal, we use `==`

```
x = 3 # assigns the value 3 to the variable x
x == 3 # checks that x has value 3 and returns either True or False
```

> String comparison

Python uses Unicode to translate *characters* to *numbers*

https://en.wikipedia.org/wiki/List_of_Unicode_characters

Scotland = [83, 99, 111, 116, 108, 97, 110, 100]

England = [69, 110, 103, 108, 97, 110, 100]

Hence

“Scotland” > “England”

> String comparison

Case matters – so

```
"ten" > "eleven"    # evaluates to True
```

```
ten = [116, 101, 110]
```

```
eleven = [101, 108, 101, 118, 101, 110]
```

But

```
"Ten" > "eleven"    # evaluates to False
```

```
Ten = [84, 101, 110]
```

> Logical operators

There are also three logical operators: and, or, and not

```
x and y    # True if both x and y are True
x or y     # True if at least one of x or y are True
not x      # True if x is False
```

> Values that count as false

Build-in types in Python have values that are evaluated to False:

Type	Falsy value	Example	Example
bool	False	<code>bool(False) → False</code>	<code>bool(True) → True</code>
int	0	<code>bool(0) → False</code>	<code>bool(7) → True</code>
float	0.0	<code>bool(0.0) → False</code>	<code>bool(3.5) → True</code>
str	""	<code>bool("") → False</code>	<code>bool("hello") → True</code>
list	[]	<code>bool([]) → False</code>	<code>bool([1, 2, 3]) → True</code>

You can check this with `bool()`.

`bool(x)` where `x` is not a falsy value evaluates to `True`.

> Conditionals: if, elif and else

Now that we can check whether things are true or not, we can create different behaviour depending on context.

```
if condition:  
    do_something()
```

- The end of the if condition is marked with a colon
- Indentation is meaningful in Python
 - In some languages indentation is only used to make things readable – meaning is fixed by punctuation, etc.
 - You can do indentation how you like (spaces, tabs) but it must be consistent

> Conditionals: if, elif and else

When we use if statements, either something happens or it doesn't.

```
if weather = "rainy":  
    activity = "watch telly"
```

We can make this more complex by using else: either something happens or something else does.

```
if weather = "rainy":  
    activity = "watch telly"  
else:  
    activity = "play football"
```

> Conditionals: if, elif and else

If there are more than two potential outcomes, we use elif – short for else if

```
if weather = "rainy":  
    activity = "watch telly"  
elif weather = "windy":  
    activity = "go walking"  
else:  
    activity = "play football"
```

If and elif need a condition – what are we checking the truth of?
Else doesn't need a condition – this is what happens if not of the conditions are true.

> Indentation in Python

We have already said that indentation in Python is meaningful. That means that if you change it, you get a different output.

```
if lecture_theatre = "G07_F21":  
    print("this is too small!")  
print("I've checked the size!")
```

The second print statement is *outside* the if statement, so it prints whether or not the condition is true.

```
if lecture_theatre = "G07_F21":  
    print("this is too small!")  
    print("I've checked the size!")
```

Now, the second print statement is *inside* the if statement, so it only prints if the condition is true.