`else` with `for` or `while` loop

- Executed if the loop exits normally (i.e. not via `break`)

```
for item in item-list:
    # code for the for loop

    if condition:
        break

    # more code for the for loop

else:
    # code block for when loop completes normally

# 'break' takes us here, skipping the 'else' clause
```

When a loop exits, it is sometimes important to know whether the loop ended because the iteration was complete, or whether the loop exited as a result of a `break` statement. With other languages, you can set a flag to let you know if exiting was as a result of finding something or not; with Python, there can be an optional `else` clause at the end of the `for` or `while` loop, specifically for this purpose.

The code shows the structure of an `else` statement associated with a `for` loop.

```
for item in item-list:
   # code for the for loop
   if condition:
      break
   # more code for the for loop
else:
   # code block for when loop completes normally
```

The indentation of the `else` shows that it is a clause at the end of the `for` loop. The behavior is:

- If the `for` loop ran to completion, the `else` code block is executed.
- If the `for` loop exited for some reason via a `break` statement, the `else` code block is not executed.

The following example shows a use of this `else` clause:

```
for device in devices_list:
   if name == device.name:
      print 'Found device:', name
      break
else:
   print 'Did not find device:', name
```

In the example, the code is iterating across devices in a list of devices, looking for a device with the specified name. If the `for` loop goes through all devices in the list, it will execute the `else` clause, printing an indication that no device with that name was found. If the device with the specified name is found, the `break` statement will be executed and the `else` clause is not executed.

The code structure and behavior for a `while-else` is the same as with the `for` loop.