More about libraries

- Library names are case and space sensitive
- Libraries can take arguments
  - Arguments are listed after the library name
  - Both library name and arguments can be set using variables
- Libraries can be imported in the *Settings* section or using *Import Library* keyword from built-in library

```
*** Settings ***
Library     OperatingSystem
Library     my.package.TestLibrary
Library     MyLibrary       arg1      arg2
Library     ${LIBRARY}
```

```
*** Test Cases ***
Example
    Do Something
    Import Library    MyLibrary     arg1     arg2
    KW From MyLibrary
```

- If the library is a file it must have an extension (.py on this course)
- You can also specify path to the library
  - The path is relative to the directory where the current test file resides
  - You can also specify absolute path

```
*** Settings ***
Library     PythonLibrary.py
Library     /absolute/path/JavaLibrary.java
Library     relative/path/PythonDirLib/    possible    arguments
Library     ${RESOURCES}/Example.class
```

TIP: Set path to your own libraries from the command line

Resource files

- Resource files are imported using the Resource keyword in the Setting section (similar to #include in C/C++)
  - Use .resource extension
  - You can specify path in the same way as with libraries

```
*** Settings ***
Resource      example.resource
Resource      ../data/resources.robot
Resource      ${RESOURCES}/common.resource
```

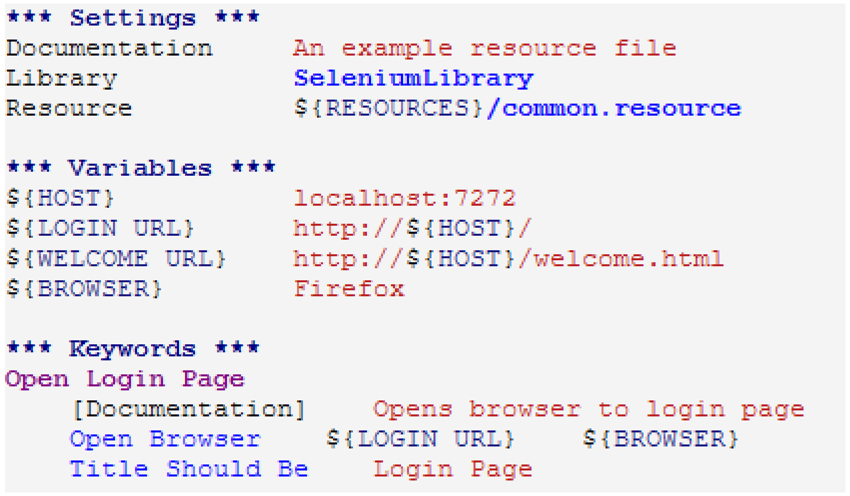
- Same structure as test case files with a few exceptions
  - Resource file can't contain test cases
  - In Settings you can only import settings (Library, Resources, Variables) and Documentation
- Variables and Keywords work in the same way as with test case file
- Variables can also be imported from a file

```
*** Settings ***
Variables     myvariables.py
Variables     ../data/variables.py
Variables     ${RESOURCES}/common.py
Variables     taking_arguments.py     arg1     ${ARG2}
```

```
VARIABLE = "An example string"
ANOTHER_VARIABLE = "This is pretty easy!"
INTEGER = 42
STRINGS = ["one", "two", "kolme", "four"]
NUMBERS = [1, INTEGER, 3.14]
MAPPING = {"one": 1, "two": 2, "three": 3}
```

Resource files

- Example:



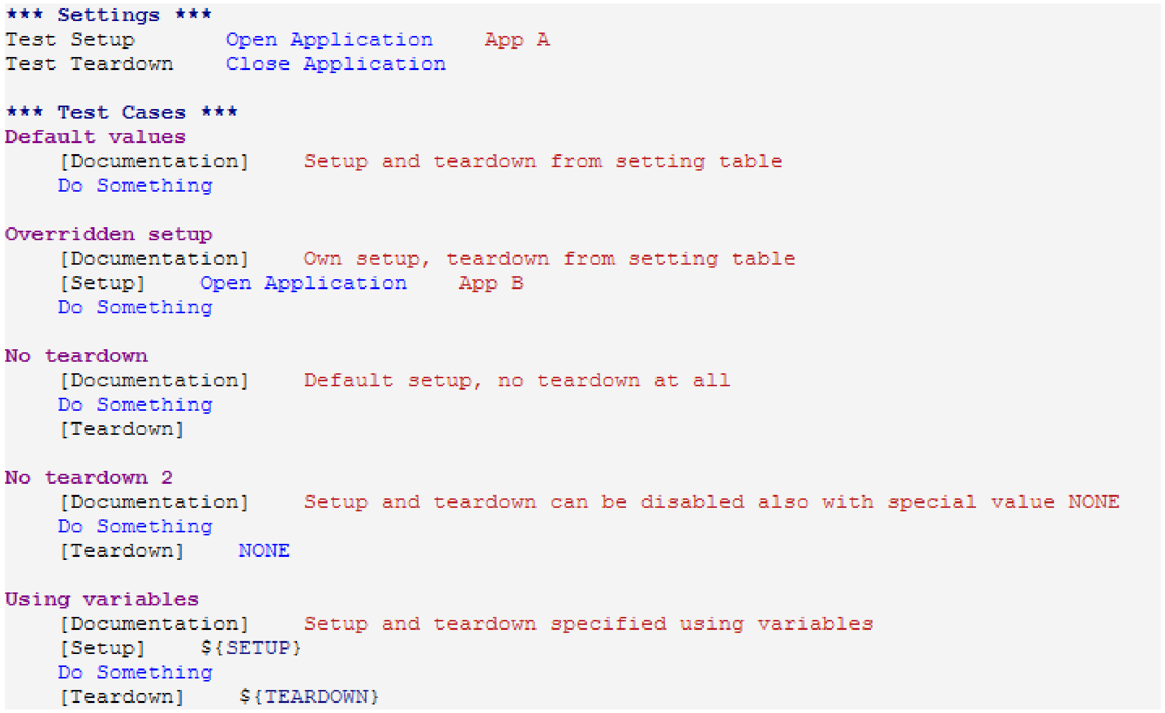
```
*** Settings ***
Documentation     An example resource file
Library           SeleniumLibrary
Resource          ${RESOURCES}/common.resource

*** Variables ***
${HOST}           localhost:7272
${LOGIN URL}      http://${HOST}/
${WELCOME URL}    http://${HOST}/welcome.html
${BROWSER}        Firefox

*** Keywords ***
Open Login Page
    [Documentation]    Opens browser to login page
    Open Browser    ${LOGIN URL}    ${BROWSER}
    Title Should Be    Login Page
```

Test setup and teardown

- Test setup is a single keyword that is executed before a test case runs and teardown is executed after the test case has run
  - Teardown is executed even if the test case fails
  - All the keywords in the teardown are executed even if one of them fails
- It is possible to override setup and teardown in individual test cases



```
*** Settings ***
Test Setup       Open Application    App A
Test Teardown    Close Application

*** Test Cases ***
Default values
    [Documentation]    Setup and teardown from setting table
    Do Something

Overridden setup
    [Documentation]    Own setup, teardown from setting table
    [Setup]    Open Application    App B
    Do Something

No teardown
    [Documentation]    Default setup, no teardown at all
    Do Something
    [Teardown]

No teardown 2
    [Documentation]    Setup and teardown can be disabled also with special value NONE
    Do Something
    [Teardown]    NONE

Using variables
    [Documentation]    Setup and teardown specified using variables
    [Setup]    ${SETUP}
    Do Something
    [Teardown]    ${TEARDOWN}
```

Suite setup and teardown

- A suite setup is executed before running any of the suite's test cases or child test suites, and a test teardown is executed after them
- They are defined in the Setting table with *Suite Setup* and *Suite Teardown* settings
- A test suite created from a directory can have similar settings as a suite created from a test case file
    - Setting of a suite created from a directory must be placed into a special test suite initialization file. An initialization file name must always be of the format __init__.ext, where the extension must be one of the supported file formats (typically __init__.robot).