Robot Framework

Hands-on workshop 11.9.2019

Trainers

Henrik Vesterinen (M.Sc.)

Software and RPA developer with specialization in Python and Robot Framework. Background in the physical sciences from the University of Helsinki and CERN.

Sanna Elomaa (M.Sc.)

RPA developer with a focus on Python and Robot Framework. Background in mathematics and the insurance industry.

Trainers

Siili Solutions

- Siili Solutions specializes in design, technology and data.
- Software development, test automation, robotic process automation
- Offices in Finland, Germany, Poland, USA
- Siili **Intelligent Automation** offers open source RPA solutions with Robot Framework.

Introduction

Welcome to the course!

- Introduction: What is Robot Framework?
- Getting started
- Best Practices
- Automating the Web

What is Robot Framework?

- Keyword-driven test automation framework
- Tests written using natural language
- Open source, Apache License 2.0
- Implemented with Python

https://robotframework.org/

What is Robot Framework?

- The first version was developed at Nokia Networks in 2005 (by Pekka Klärck)
- Version 2.0 was released as open source in 2008
- Nowadays sponsored by Robot Framework Foundation
  - A non-profit consortium of approximately 30 member companies

Users of Robot Framework

- Nokia
- Cisco
- ABB
- Kone
- U.S. Naval Research Laboratory
- ...and more

Active development and a growing community!

https://robotframework.org/#users


Getting started

Basic tools

We'll be working with the command line and an editor.

- Install Python 3 (newest version)
  https://www.python.org/downloads/

NB: On Windows, add Python scripts directory to PATH.

Good code editor

Recommended:

- PyCharm
- Atom
- Sublime Text
- VSCode

Development environment

You should always use a virtual environment when developing Python software. Python 3 has built in `venv`:

Linux/MacOS:

```
python3 -m venv venv
. venv/bin/activate
```

Windows:

```
python -m venv venv
call venv\Scripts\activate.bat
```

For more sophisticated usage, see *virtualenvwrapper*:
https://virtualenvwrapper.readthedocs.io/en/latest/

Installing Robot Framework

Robot Framework is a Python library so you can install it with `pip`:

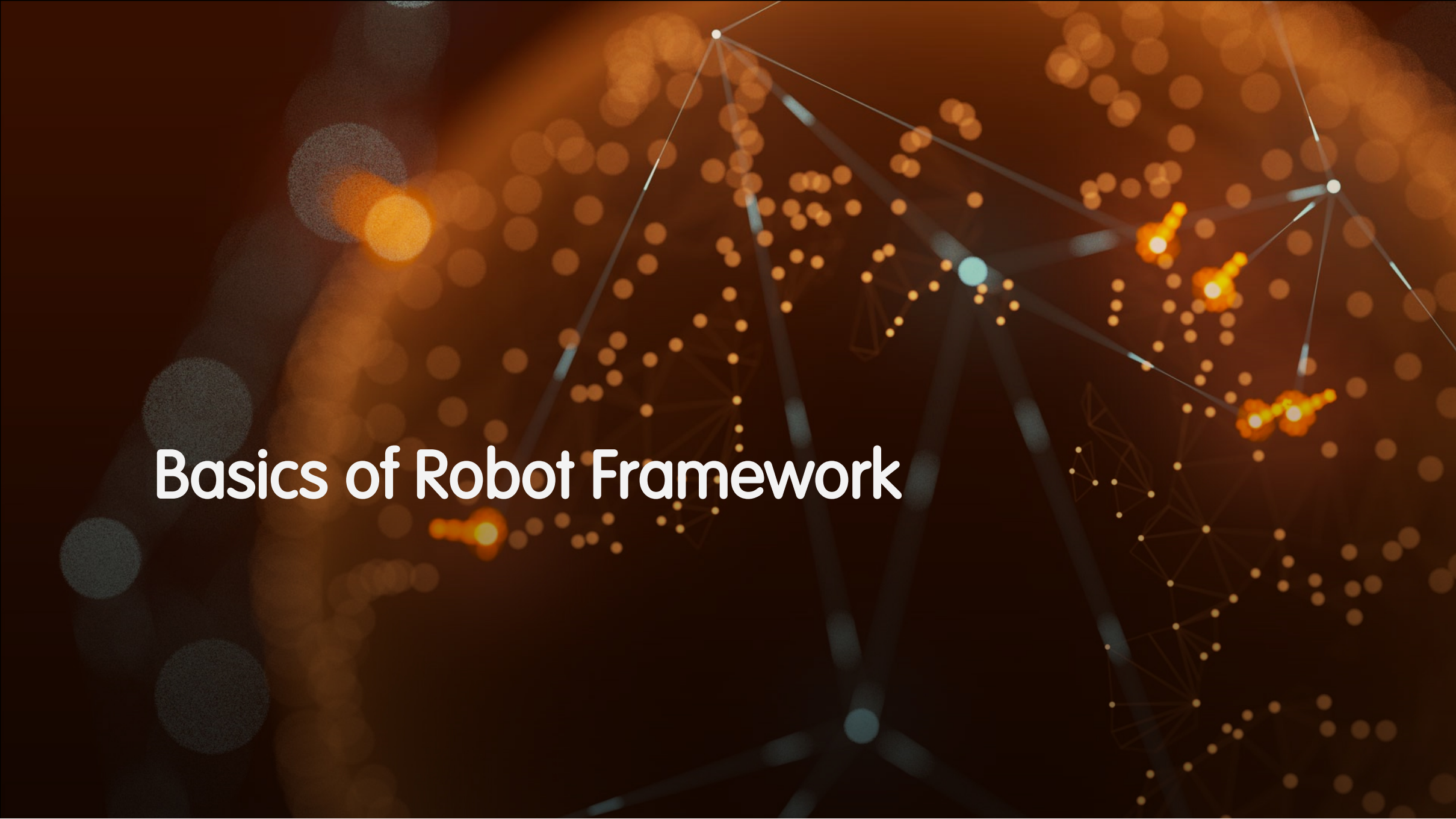
```
pip install robotframework
```

Exercise

Set up your Python environment

1. Make sure your editor is set up
2. Create a project directory
3. Create a new virtual environment and activate it
4. Install Robot Framework with pip into your virtual environment
5. Test your installation by running

```
robot --version
```


Basics of Robot Framework

Terminology

- Test Suite
- Test Case
- Keyword
- Libraries
- Resources

Syntax

- Plain text
- File extension `.robot`
- Space separated (min. 2)
- Keyword and test case blocks are defined with indents
- Code editor support

```
Open Login Page
    Log Variables
    Open Browser    www.example.com    chrome
```

Syntax: suite example

```
*** Settings ***
Library                   SeleniumLibrary


*** Variables ***
${USERNAME}               admin
${PASSWORD}               nimda
${URL}                    http://www.example.com


*** Test Cases ***
Valid Login
    Open Login Page
    Input Username        ${USERNAME}
    Input Password        ${PASSWORD}

*** Keywords ***
Open Login Page
    Open Browser          ${URL}          chrome
```

Available libraries

Standard libraries, e.g.

- BuiltIn
- Collections
- Dialogs
- Screenshot
- String

External libraries, e.g.

- AppiumLibrary
- SapGuiLibrary
- SeleniumLibrary
- WhiteLibrary
- TOSLibrary

BuiltIn library

The BuiltIn library includes some often needed keywords, e.g.

- Log
- Set Variable
- Should Be Equal
- Sleep
- Run Keyword If
- Fail
- Convert To Integer

Always available without importing.

About the Robot Framework parser

The keyword libraries are not magic, they are Python code!

The following Python function

```
def print_sum(a, b):
    print(a + b)
```

can be called from Robot Framework like:

```
Print Sum    1    5
```

Exercise

We'll build a simple test suite in the project directory.

1. Create a new file `my_first_test_suite.robot`
2. Import the `Screenshot` library
3. Create a keyword `Take A Screenshot And Log Text` that uses the following standard library keywords:
   - `Take Screenshot`
   - `Log`
4. Create a test case `Logging Test` which calls your own keyword

Exercise

Your solution should look something like this:

```
*** Settings ***
Library                   Screenshot


*** Test Cases ***
Logging Test
    Take Screenshot And Log Text


*** Keywords ***
Take Screenshot And Log Text
    Take Screenshot
    Log                   Something worth logging...
```

Running Robot Framework tests

```
robot <test_suite_file>
```

Running the tests generates three output files:

- report.html
- log.html
- output.xml

Exercise

Run your previously created test suite and look at the output files.

Exercise

Run your previously created test suite and look at the output files.

Once the run has completed you should be seeing something like this:

```
==============================================================================
First Test Suite
==============================================================================
Logging Test                                                          | PASS |
------------------------------------------------------------------------------
First Test Suite                                                      | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\exercise\output.xml
Log:     C:\exercise\log.html
Report:  C:\exercise\report.html
```

Variables

- String: `${MY_VARIABLE}`
- List: `@{MY_VARIABLES}`
- Dictionary: `&{MY_MAPPING}`

```
*** Variables ***
${VARIABLE}             An example string
${ANOTHER VARIABLE}     Another example string
${INTEGER}              ${42}
@{STRINGS}              one           two            three          four
@{NUMBERS}              ${1}          ${INTEGER}     ${3.14}
&{MAPPING}              one=${1}      two=${2}       three=${3}
&{FINNISH}              cat=kissa     dog=koira
```

Keyword arguments

Defining arguments for user keywords:

```
*** Keywords ***
Input Username
    [Arguments]         ${username}
    Input Text          username_field      ${username}
```

You can then pass variables as arguments when calling the keyword:

```
Input Username          ${VALID_USER_ID}
```

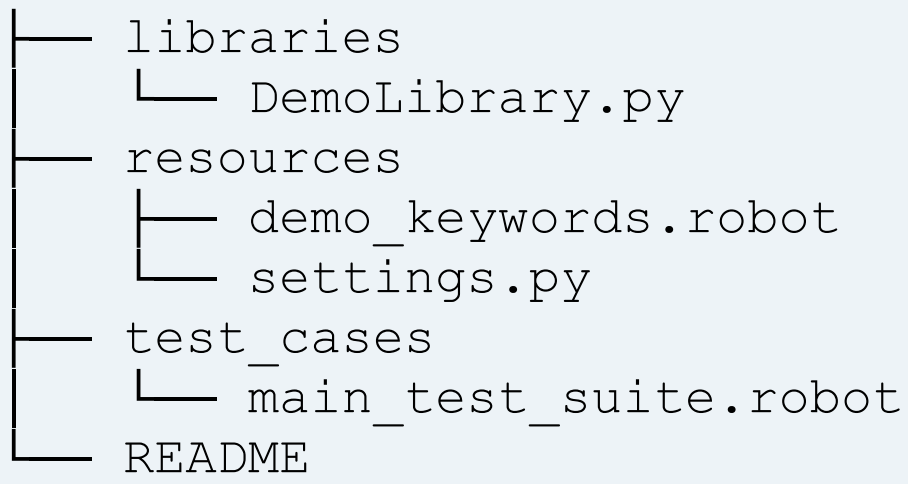
Exercise

Modify your previous test suite:

1. Create a variable for your text in the Variables table
2. Define an argument for the keyword `Take A Screenshot And Log Text`
3. Pass the variable as an argument when calling the keyword in your test case
4. Run your tests and check the log


Best practices

Ideal project structure



Ideal project structure

```
├── test_cases
│   └── main_test_suite.robot
```

This is the high level test suite:

```
*** Settings ***
Variables           ../resources/settings.py
Library             ../libraries/DemoLibrary.py
Resource            ../resources/demo_keywords.robot

*** Test Cases ***
Assert Input Data Valid
    Read Excel File
```

There should be no 'coding' on this level of abstraction.

Test cases should be atomic!

Robot Framework Keywords

```
├── resources
│   ├── demo_keywords.robot
```

```
*** Settings ***
Library             ../libraries/InputExcelReader.py

*** Keywords ***
Read Excel File
    Read Excel                  ${excel_path}
    Load Worksheet              ${sheet_name}
    ${raw_data}=                Extract Data
    Set Suite Variable          ${RAW_DATA}     ${raw_data}
```

Name the keywords with natural language.

Configuration

```
├── resources
│   ├── settings.py
```

Put configurable settings inside `resources/settings.py`:

```
db_server = "lolcathost"
db_port = 27017
db_name = "demorobot"

excel_path = "test_input.xlsx"
sheet_name = "Sheet1"
```

And import them like

```
*** Settings ***
Variables           ../resources/settings.py
```

Python libraries

```
├── libraries
│   └── InputExcelReader.py
```

More complex keywords should be written in Python:

```python
import openpyxl

class InputExcelReader:

    def read_excel_file(self, excel_path):
        self.wb = openpyxl.load_workbook(excel_path, data_only=True)
```

Debugging in Robot Framework

There is no real working debugger for Robot Framework. However, you can pause the execution with `Pause Execution` keyword. For this to work, you have to import library `Dialogs`:

```
*** Settings ***
Library    Dialogs

*** Test Cases ***
Test Dialogs
    Pause Execution         Personalized popup message
```

And you can use `Log Variables` to log all variable values to RF log.


Automating the Web

SeleniumLibrary

Web automation is the most common and best supported by Robot Framework.

Install `SeleniumLibrary` into your virtual environment:

```
pip install robotframework-seleniumlibrary
```

For keyword documentation, see http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

Webdrivers

You also need a browser and a web driver. Recommended are Google Chrome and chromedriver.

Use `webdrivermanager` to install the driver inside the virtualenv:

```
pip install webdrivermanager
```

and then

```
webdrivermanager chrome
```

Locators

Locators specify the GUI elements used in keywords.

In Chrome you can find locators for elements by right clicking on them and selecting Inspect.

Locators

Locators specify the GUI elements used in keywords.

In Chrome you can find locators for elements by right clicking on them and selecting Inspect.

Different locator strategies:

- id
- name
- class
- xpath
- ...

Exercise

Create a test case which does the following using SeleniumLibrary:

- Perform a search on DuckDuckGo (https://duckduckgo.com/) for "Robot Framework"
- Verify that the first search result is for robotframework.org

Exercise: Example solution (1/2)

```
*** Settings ***
Documentation       SeleniumLibrary demo
Library             SeleniumLibrary


*** Variables ***
${rf_url}=          robotframework.org
${duck_url}=        https://duckduckgo.com/


*** Test Cases ***
Test RF on search engine
    [Setup]         Open duckduckgo
    Search For Robot Framework
    Assert First Result is RF Homepage
    [Teardown]      Close all browsers
```

Exercise: Example solution (2/2)

```
*** Keywords ***
Open duckduckgo
    [Documentation]     Open Duck Duck Go Search engine
    ...                 with Chrome browser.
    Open Browser        ${duck_url}         gc

Search For Robot Framework
    Input query
    Click Search

Input query
    Input text       id:search_form_input_homepage      Robot Framework

Click Search
    Click Element    id:search_button_homepage

Assert First Result is RF Homepage
    Wait Until Page Contains Element    id:r1-0
    ${url}=    Get Element Attribute    id:r1-0         data-hostname
    Should Be Equal                     ${url}          ${rf_url}
```


More Robot Framework

Setup & Teardown

Setup specifies the actions to be performed before test execution, and teardown specifies the actions after the test execution.

Two possible levels:

- Test Suite: executed once per test run
- Test Case: executed once per test case

Teardown is executed despite the test result (pass, fail, interrupted, ...)

Setup & Teardown

```
*** Settings ***
Suite Teardown          Clean Up


*** Test Cases ***
Test Something
    [Setup]             Open Application
    Do Something
    [Teardown]          Close Application
```

Tags

Test cases can be labeled with tags:

```
*** Test Cases ***
First Example
    [Tags]          smoke
    Do Something

Second Example
    [Tags]          smoke   not_ready
    Do Something Else

Third Example
    [Tags]          dummy
    Do Something Completely Different
```

Tags can be used to include or exclude cases when running the tests:

```
robot --include smoke --exclude not_ready tests.robot
```

Data Driver Testing

Robot Framework supports test templates which provide an easy way to move from keyword-driven test cases to data-driven tests.

```
*** Test Cases ***
Templated test case
    [Template]          Example keyword
    <situation_1__arg_1>        <situation_1__arg_2>
    <situation_2__arg_1>        <situation_2__arg_2>
```

And much more...

- https://robotframework.org/
- Robot Framework User Guide

And much more...

- https://robotframework.org/
- Robot Framework User Guide

- Robot Framework Conference: https://robocon.io/
- Meetups all over the world
  - Helsinki, Berlin, Stockholm, San Francisco, Sydney, Utrecht...


Thanks!