Seleniunlibrary advanced workshop

Tatu Aalto

agenda

- Introduction, Browser configuration, debugging
- Break
- Parallel execution, Expanding library, EventFringWebDrier
- Lunch
- Page object, Python page objects vs SeleniumLibrary,
- Checking for time of tests
- Break
- Using JavaScript to interact with SUT

Introduction

- Tell bout you.
  - Where do work, what how does your normal workday look like?
  - Hobby projects?
- What you want to learn?
  - Why you are here?
  - What are the problematic areas in your work
- How do learn?
  - Do you read books, workshops, trainings or something else?
  - When do you earn, at work, somewhere else?

Browse configuration

- Create WebDriver vs Open Browser
- Create WebDriver is more powerful, but is somewhat harder to read
- Open Browser is more limited, but keyword has many options.
  - https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Open%20Browser
- Open Browser support Selenium grid
- Long expression can be hard to read in both keywords.
  - Move logic in Python side

DEBUGGING

- Debugging depends on the problem
  - Selector, library, SUT, keyword or some other problem
- Firefox and Chrome developer tools are good for xpath and ccs selectors
- Open browser console
  - Xpath: **$x('//div’)**
  - CSS: **$$('head')**

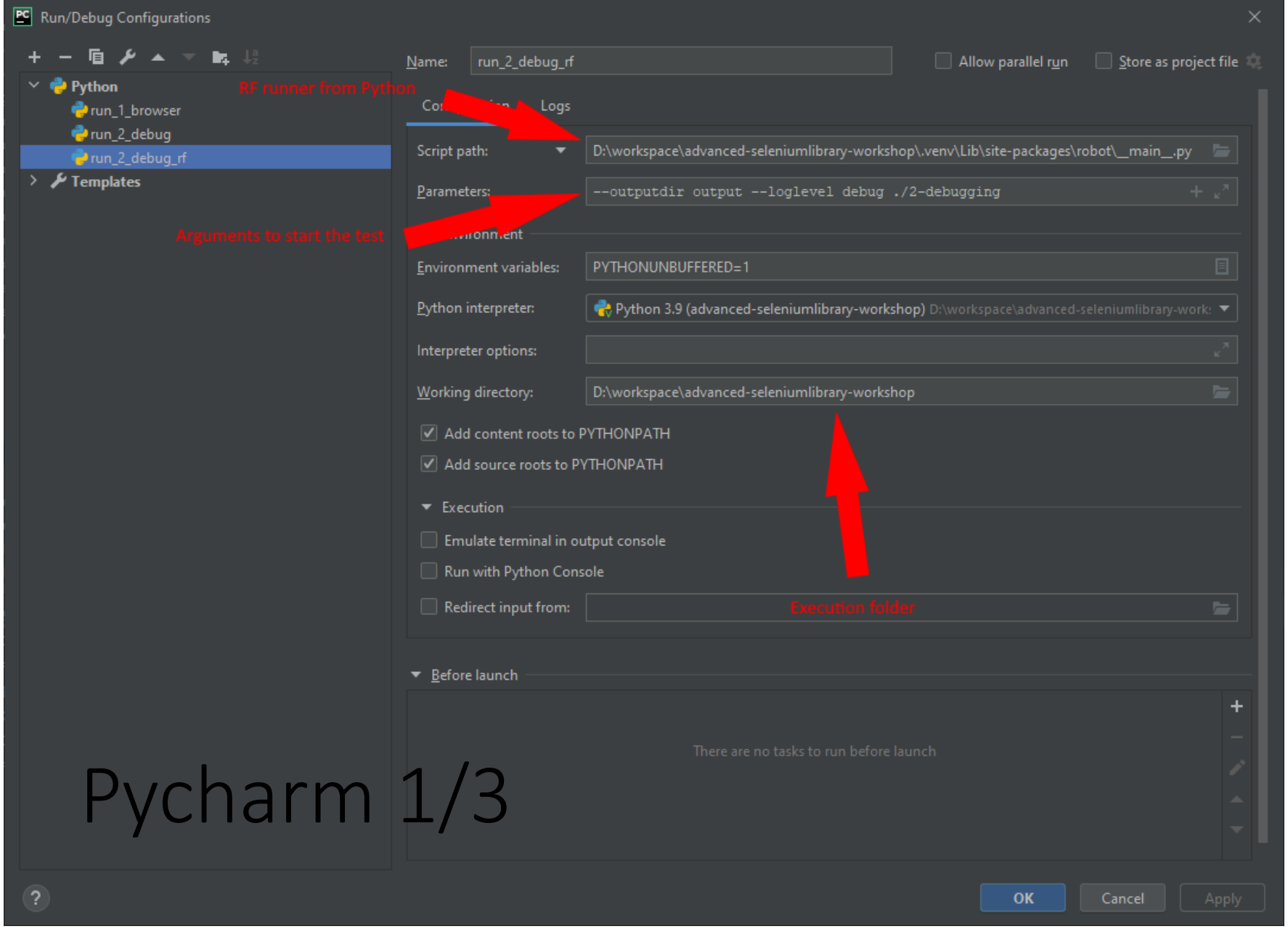
Debugging library

- Convert the keyword by using Python and Selenium

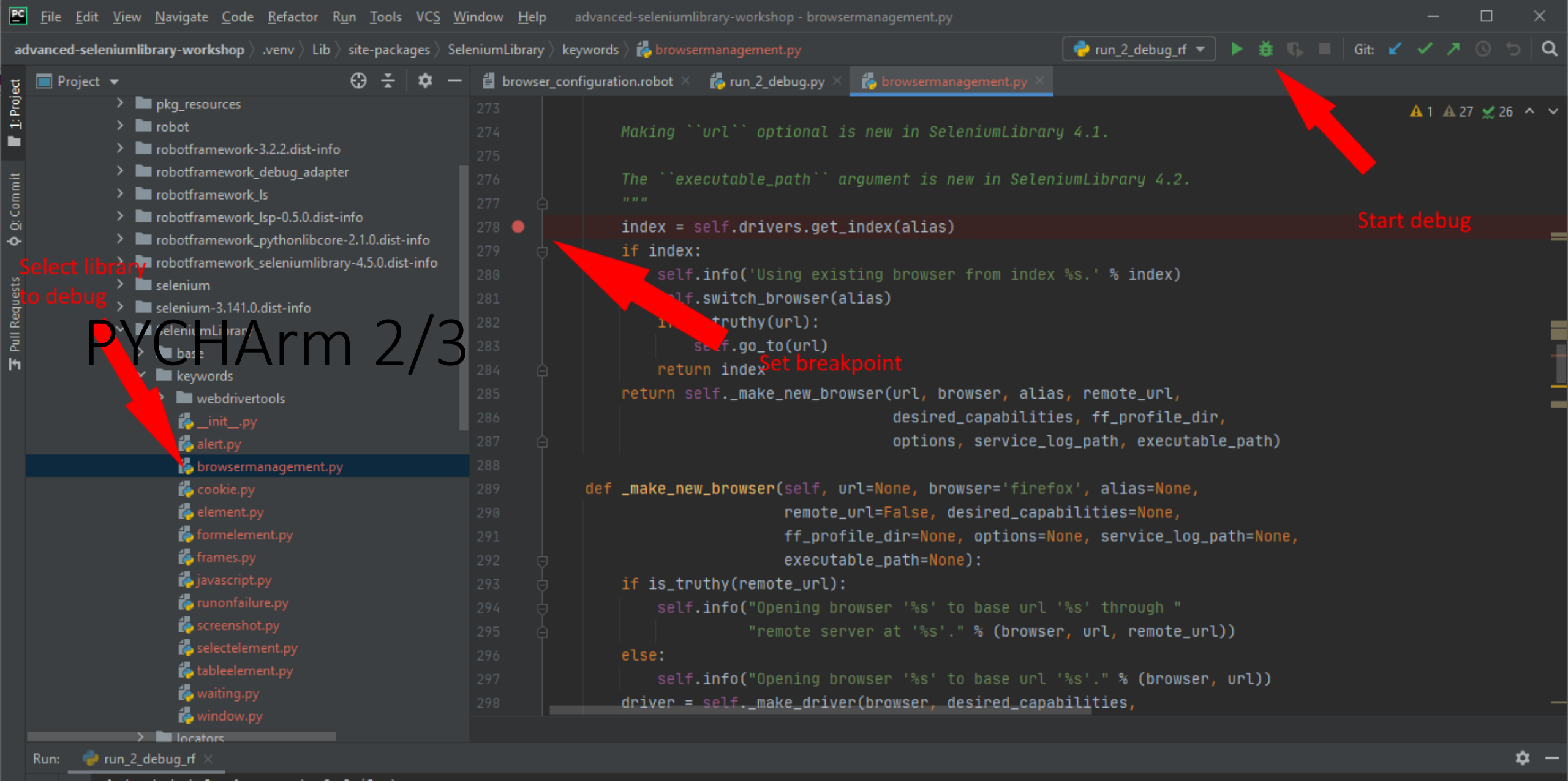
- Robot Framework user guide:
  - import sys, pdb; pdb.Pdb(stdout=sys.__stdout__).set_trace()
- Or use real IDE

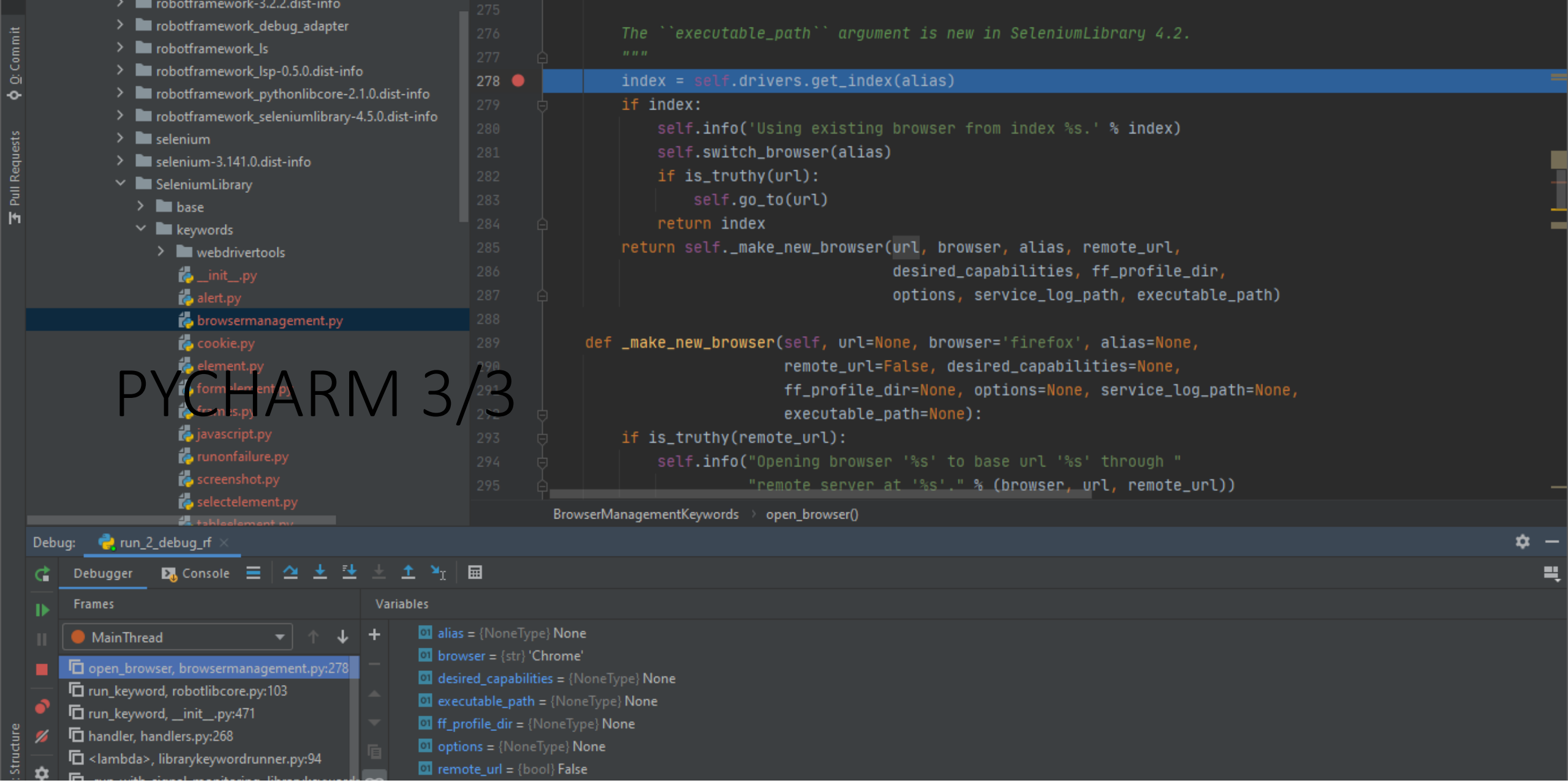
```python
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://robocon.io/")
# more code
driver.quit()
```


Run/Debug Configurations
Python
run_1_browser
run_2_debug
run_2_debug_rf
Templates
RF runner from Python
Arguments to start the test
Name: run_2_debug_rf
Allow parallel run
Store as project file
Logs
Script path: D:\workspace\advanced-seleniumlibrary-workshop\.venv\Lib\site-packages\robot__main__.py
Parameters: --outputdir output --loglevel debug ./2-debugging
Environment variables: PYTHONUNBUFFERED=1
Python interpreter: Python 3.9 (advanced-seleniumlibrary-workshop) D:\workspace\advanced-seleniumlibrary-work
Interpreter options:
Working directory: D:\workspace\advanced-seleniumlibrary-workshop
Add content roots to PYTHONPATH
Add source roots to PYTHONPATH
Execution
Emulate terminal in output console
Run with Python Console
Redirect input from:
Execution folder
Before launch
There are no tasks to run before launch
OK
Cancel
Apply

Pycharm 1/3


File Edit View Navigate Code Refactor Run Tools VCS Window Help
advanced-seleniumlibrary-workshop - browsermanagement.py
advanced-seleniumlibrary-workshop .venv Lib site-packages SeleniumLibrary keywords browsermanagement.py
run_2_debug_rf
Git:
Project
browser_configuration.robot
run_2_debug.py
browsermanagement.py
pkg_resources
robot
robotframework-3.2.2.dist-info
robotframework_debug_adapter
robotframework_ls
robotframework_lsp-0.5.0.dist-info
robotframework_pythonlibcore-2.1.0.dist-info
robotframework_seleniumlibrary-4.5.0.dist-info
selenium
selenium-3.141.0.dist-info
keywords
webdrivertools
__init__.py
alert.py
browsermanagement.py
cookie.py
element.py
formelement.py
frames.py
javascript.py
runonfailure.py
screenshot.py
selectelement.py
tableelement.py
waiting.py
window.py
locators
Making ``url`` optional is new in SeleniumLibrary 4.1.
The ``executable_path`` argument is new in SeleniumLibrary 4.2.
"""
index = self.drivers.get_index(alias)
if index:
self.info('Using existing browser from index %s.' % index)
self.switch_browser(alias)
return index
return self._make_new_browser(url, browser, alias, remote_url,
desired_capabilities, ff_profile_dir,
options, service_log_path, executable_path)
def _make_new_browser(self, url=None, browser='firefox', alias=None,
remote_url=False, desired_capabilities=None,
ff_profile_dir=None, options=None, service_log_path=None,
executable_path=None):
if is_truthy(remote_url):
self.info("Opening browser '%s' to base url '%s' through "
"remote server at '%s'." % (browser, url, remote_url))
else:
self.info("Opening browser '%s' to base url '%s'." % (browser, url))
driver = self._make_driver(browser, desired_capabilities,
Run: run_2_debug_rf
Start debug
Select library to debug
Set breakpoint
PYCHArm 2/3


robotframework-3.2.2.dist-info
robotframework_debug_adapter
robotframework_ls
robotframework_lsp-0.5.0.dist-info
robotframework_pythonlibcore-2.1.0.dist-info
robotframework_seleniumlibrary-4.5.0.dist-info
selenium
selenium-3.141.0.dist-info
SeleniumLibrary
base
keywords
webdrivertools
__init__.py
alert.py
browsermanagement.py
cookie.py
element.py
javascript.py
runonfailure.py
screenshot.py
selectelement.py
PYCHARM 3/3
The ``executable_path`` argument is new in SeleniumLibrary 4.2.
"""
index = self.drivers.get_index(alias)
if index:
self.info('Using existing browser from index %s.' % index)
self.switch_browser(alias)
if is_truthy(url):
self.go_to(url)
return index
return self._make_new_browser(url, browser, alias, remote_url,
desired_capabilities, ff_profile_dir,
options, service_log_path, executable_path)
def _make_new_browser(self, url=None, browser='firefox', alias=None,
remote_url=False, desired_capabilities=None,
ff_profile_dir=None, options=None, service_log_path=None,
executable_path=None):
if is_truthy(remote_url):
self.info("Opening browser '%s' to base url '%s' through "
"remote server at '%s'." % (browser, url, remote_url))
BrowserManagementKeywords › open_browser()
Debug: run_2_debug_rf
Debugger
Console
Frames
Variables
MainThread
open_browser, browsermanagement.py:278
run_keyword, robotlibcore.py:103
run_keyword, __init__.py:471
handler, handlers.py:268
<lambda>, librarykeywordrunner.py:94
alias = {NoneType} None
browser = {str} 'Chrome'
desired_capabilities = {NoneType} None
executable_path = {NoneType} None
ff_profile_dir = {NoneType} None
options = {NoneType} None
remote_url = {bool} False

More logging

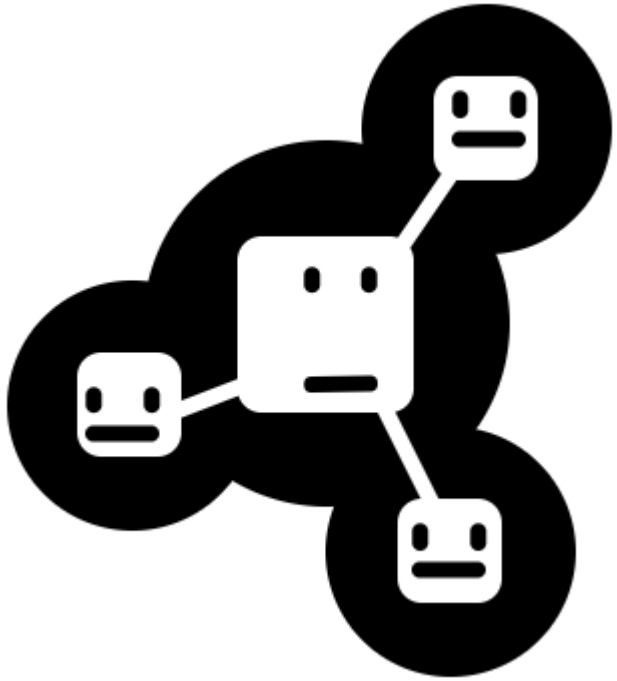
- Increate the loglevel: --loglevel trace

- Enable browser driver logs with service_log_path argument in Open Browser keyword

Robot Framework debuglibrary

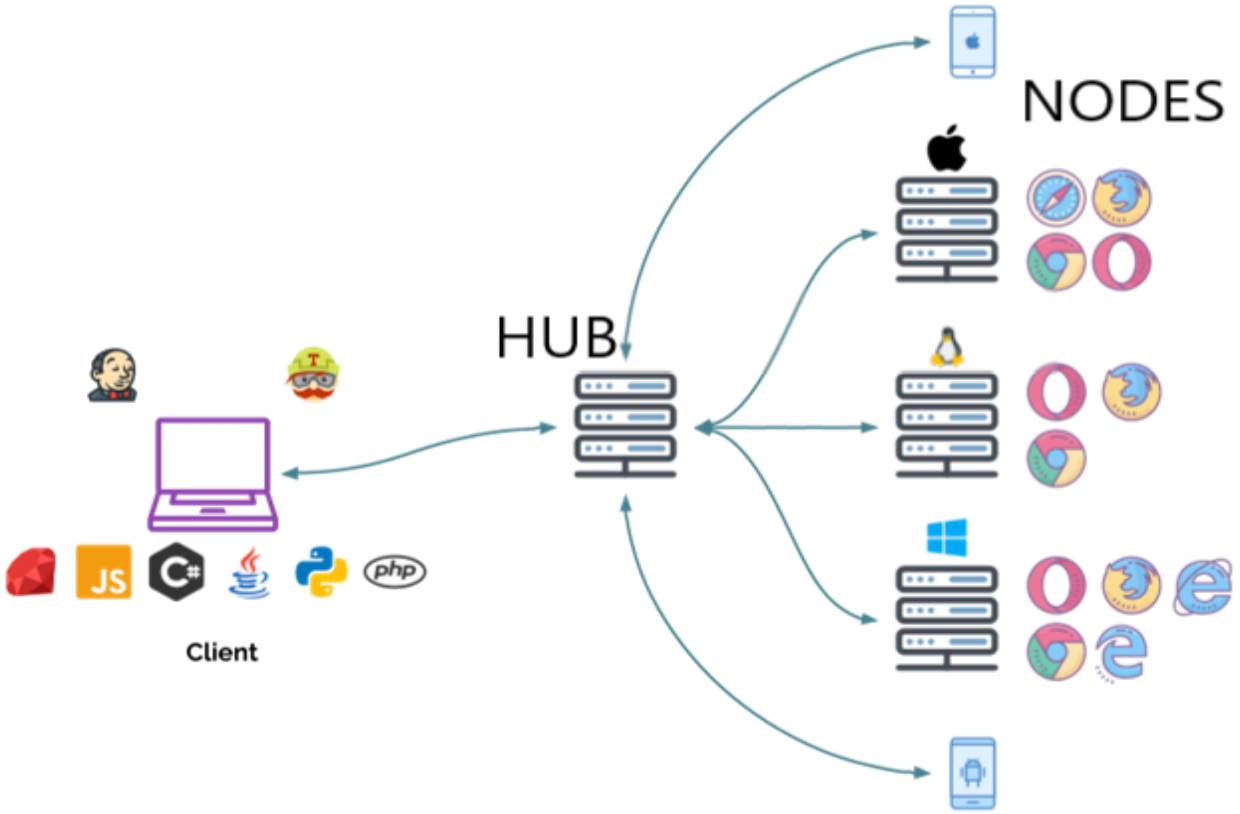
- Debuglibrary works with a **Debug** keyword
- From command line allows to:
  - Execute keywords
  - Create variables
  - Step in RF code
- Debug logic, try out selectors

Parallel EXECUTION: PABOT

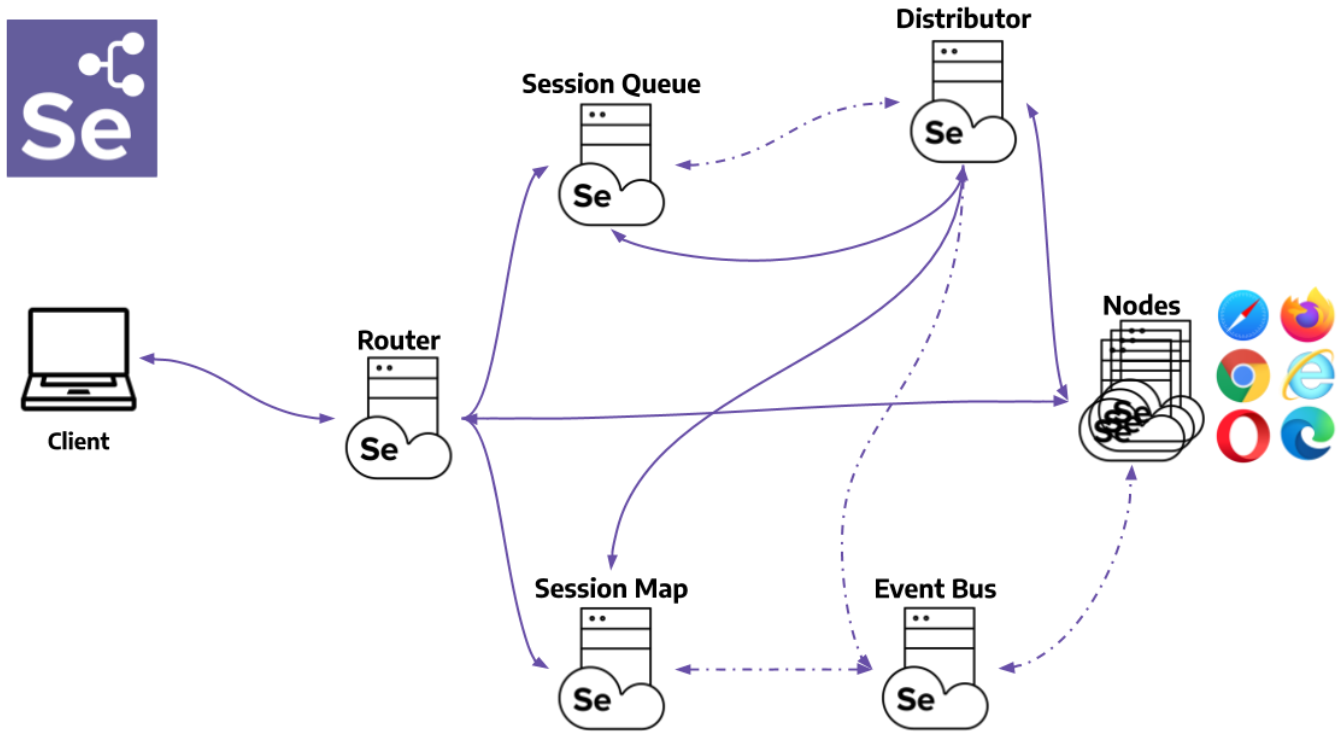


```
[PID:6] [0] EXECUTING Suites.Suite1   .
[PID:7] [1] EXECUTING Suites.Suite2   o-- suites
[PID:7] [1] PASSED Suites.Suite2          |
[PID:6] [0] PASSED Suites.Suite1          o-- suite1.robot
Output:  /output.xml                      |   |
Log:     /log.html                        |   o-- Test 1
Report:  /report.html                     |   |
Total testing: 16.20 seconds              |   o-- Test 2
Elapsed time:  9.10 seconds               |
                                          o-- suite2.robot
                                              |
                                              o-- Test 3
                                              |
                                              o-- Test 4
```

Parallel EXECUTION: grid 3



Parallel EXECUTION: grid 4



EXTENDING SELENIUMlibrary

- Two ways:
  1. Crete new library (There are multiple ways)
  2. Plugin API

- SeleniumLibrary has public API for extending:
  - https://github.com/robotframework/SeleniumLibrary/blob/master/docs/extending/extending.rst#public-api

Event fringing webdriver

- Listener interface for Selenium API
- Allows the perform actions before and after SeleniumLibrary calls Selenium API
  - Supports almost all Selenium API methods, expect when creating WebDriver
  - Grants access to the currently active WebDriver and WebElements (when feasible)

- Instead of using SeleniumLibrary keywords to wait that page is ready, Event Fringing WebDriver can do it automatically.
- More details: https://www.selenium.dev/selenium/docs/api/py/webdriver_support/selenium.webdriver.support.event_firing_webdriver.html#module-selenium.webdriver.support.event_firing_webdriver

Page object model

- What Page Object pattern tries to achieve in Selenium
  1. Clear separation between the test and code interacting the page (locators, methods, classes)
  2. Single repository/source for interactions in the page.

- Do I need Page Object library/model with Robot Framework?

Page object LIBRARY

- Do I need write a library or can it be done for Robot Framework data/code?
- Prior art: https://github.com/boakley/robotframework-pageobjectlibrary/
- Should which kind of library I should write for Page Object?

Checking time of the test

- Knowing how long your test take run:
  - Single test?
  - Single suite?
  - Singe test run in CI?
  - How long it takes to run all the test
- Knowing which test are run?
  - Are all test used?
  - Are all test run in regularly
  - How often you run all test?

Profiling single test run

- Gives you snapshot timing and their possible test runs:
  - https://bitbucket.org/robotframework/robottools/src/master/times2csv/
  - https://github.com/jzdunek/robot-profiler/blob/master/src/python/robot_profiler.py

- If problem happens randomly, looking at the problematic points is good idea.

Visualizing test executions

- Feed test result to a database and use a service to visualize the data
  - Redash, visualization service on top of database (somewhere in the cloud)
  - AWS, there are many ways
  - https://github.com/salabs/Epimetheus visualization service on top of database

JAVASCRIPT

- Useful for many things:
  - Polling
  - File uploads
  - Click and so

- Test without **Wait…** keyword is possible
  - Instead of waiting elements in the page.
  - Wait Until Page Contains Element + Wait Until Element Is Visible => Please do not that
  - Ask from page when it is ready

SELENIUM testablity

- Automatic detection of asynchronous events in the page
  - CSS animations, rest API calls and so on.

- Does waiting automatically. No more wait keywords

ANGULAR and primefaces

- Both uses JavaScript to poll application
- Angular library: https://github.com/Selenium2Library/robotframework-angularjs/blob/master/AngularJSLibrary/__init__.py
- Primafaces library: https://github.com/MarketSquare/robotframework-primefaces/blob/master/src/PrimeFacesLibrary/__init__.py
- Ask from the application when it is ready