**AUTOMATED UNIT TESTING**

**BROWSER TEST AUTOMATION**

**MOBILE TEST AUTOMATION**

**USING**

**PYTHON / PYTEST/ SELENIUM/ APPIUM AND**

**OTHER EXTENSIONS**

Table of Contents

[1 Introduction 3](#_Toc28016021)

[2 Comparison between manual and test automation 3](#_Toc28016022)

[3 Glossary 3](#_Toc28016023)

[4 References 3](#_Toc28016024)

[5 Ecosystem for the training course 3](#_Toc28016025)

[5.1 Mocking 3](#_Toc28016026)

[5.2 Pytest 4](#_Toc28016027)

[5.3 Choosing a test layout / import rules 4](#_Toc28016028)

[5.3.1 Choosing a test layout / import rules 4](#_Toc28016029)

[5.3.2 Good Practices 4](#_Toc28016030)

[5.3.3 Fixtures 4](#_Toc28016031)

[5.4 Selenium 4](#_Toc28016032)

[5.5 Web drivers 5](#_Toc28016033)

[5.5.1 Chrome 5](#_Toc28016034)

[5.5.2 Firefox 5](#_Toc28016035)

[5.5.3 Edge 5](#_Toc28016036)

[5.5.4 Safari 5](#_Toc28016037)

# Introduction

Over the years, testing applications has become very challenging due to the different browsers and devices that end users have at their disposal.

In addition to the plethora of devices/browsers, manual testing is slow, subjective and prone to errors.

The lifecycle in manual testing is a long and laborious process. Reporting defects and their status is another area that frequently suffers.

Over time, is has been observed that testers tend to lose focus due to fatigue, boredom and other factors.

Numerous test automation frameworks have been developed to address this problem. To start with, test automation is more difficult to get off the ground. There is a much steeper learning curve and requires immense discipline to get going.

The benefits of using test automation are numerous. OF course, there are many cases where test automation is not possible.

# Comparison between manual and test automation

# Glossary

# References

# Ecosystem for the training course

## Mocking

provides extremely powerful and useful means by which to mock and stub out these undesired side-effects.

example: perhaps we’re writing a finance app and want to test out our new ‘Post to Accounts feature’, but don’t want to actually post to Accounts every time we run our test suite

## Pytest

### Best Practices

### Choosing a test layout / import rules

<https://docs.pytest.org/en/latest/goodpractices.html#test-discovery>

|  |  |
| --- | --- |
| **Tests outside application code** | Putting tests into an extra directory outside your actual application code might be useful if you have many functional tests or for other reasons want to keep tests separate from actual application code (often a good idea): |
| **Tests as part of application code** | Inlining test directories into your application package is useful if you have direct relation between tests and application modules and want to distribute them along with your application: |

### Fixtures

|  |  |  |  |
| --- | --- | --- | --- |
|  | A software test fixture sets up the system for the testing process by providing it with all the necessary code to initialize it, thereby satisfying whatever preconditions there may be. An example could be loading up a database with known parameters from a customer site before running your test. |  |  |
|  | The purpose of test fixtures is to provide a fixed baseline upon which tests can reliably and repeatedly execute. |  |  |

### Running Pytest

<https://docs.pytest.org/en/latest/pythonpath.html#pytest-vs-python-m-pytest>

## TOX (advanced topic)

## Selenium

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

## Web drivers

### Chrome

### Firefox

### Edge

### Safari

| **MAIN TOPIC** | **SUB TOPIC** | **DETAILS**  **/**  **(LINKS FOR FURTHER STUDY)**  **/**  **(FEEDBACK)**  **/**  **(SAMPLE PROGRAMS)** | **(CLASSROOM EXERCISES)**  **/**  **(ASSIGNMENTS)** | **ETA** |
| --- | --- | --- | --- | --- |
| **OVERALL CONTEXT** | WHAT ARE YOU EXPECTING ? | <Update after feedback from the students> | **N/A** | DAY 1  (<=15 mins) |
|  | MY EXPECTATIONS FROM THE STUDENTS/YOU | * Be aware of the course content (*Have all of you gone through the course details [separate doc] ?)* * Do the class room exercises * Complete your assignments * Make notes *(I do it and it helps me)* * Don’t just nod your head to what I say. Digest it slowly. Stop me if I am going too fast | **N/A** | DAY 1  (<= 15 mins) |
|  |  |  |  |  |
| **PYTEST** | The Basics | # A really basic example 1  def test\_numbers\_3\_4():      assert( multiply(3,4) == 12 ) |  |  |
|  |  | *#* A really basic example 2  **def** reducto(x):  **return** x - 1  **def** test\_answer():  **assert** reducto (3) == 5 |  |  |
|  |  | The ASSERT keyword |  |  |
|  |  | Executing code on specific events |  |  |
|  |  | Executing a single test |  |  |
|  |  |  |  |  |
|  |  | Executing multiple tests |  |  |
|  |  |  |  |  |
|  |  | Assert that a specific exception is raised |  |  |
|  |  |  |  |  |
|  |  | Grouping multiple tests in a Class |  |  |
|  |  |  |  |  |
|  |  | Marking test functions with attributes |  |  |
|  |  | Stopping after the first (or N) failures |  |  |
|  |  |  |  |  |
|  |  | Specifying tests / selecting tests |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |