

## Lab 2

### Purpose

The lab intends to introduce the concept of unit testing and the test framework “Pytest” from Python. Mockups, overriding objects, and how to structure test cases is covered. Finally you will learn how to generate test reports on code coverage from your test runs.

### Task description

A Python class and corresponding methods are supplied:

```
class DataChecker:
    def __init__(self):
        ...

    def check_valid_age(self, age):
        ...

    def check_valid_text_field(self, input):
        ...

    def customer_has_equipment_attached(self, customerID):
        ...
```

The implementation for the class and methods can be found in the test system under:

`/home/pft/restapi/point-of-sale/datachecker.py`

Write test cases covering the methods in the class. There will be bugs present in the code, if encountered via test case, then write a corresponding fix and re-run test to make it pass. If you encounter the bug via code inspection, then first write a failing test case to trigger the defect, then implement the fix and re-run your test case.

Also you should produce a code coverage report of the class under test (datachecker.py) after having executed the test cases.

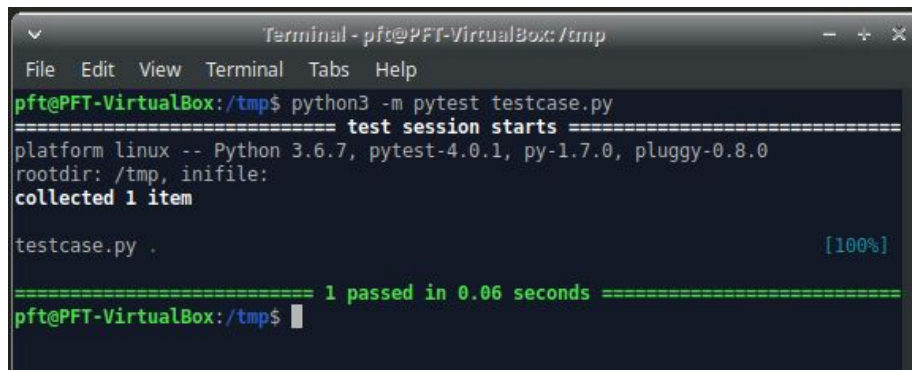
## How to write and run test cases using PyTest in Python

The pytest framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries. In order to write a

test case, you only need to implement a method that starts with “test\_” in its method name.

```
def test_addition():  
    assert 1 + 1 == 2
```

Next to execute the test we run the PyTest module from the terminal, either a file name is specified or PyTest will recursively look for any files containing “test\_” methods in the current working directory.

A screenshot of a terminal window titled "Terminal - pft@PFT-VirtualBox: /tmp". The terminal shows the command "python3 -m pytest testcase.py" being executed. The output indicates a successful test session with one item collected and passed in 0.06 seconds. The terminal text is as follows:

```
Terminal - pft@PFT-VirtualBox: /tmp  
File Edit View Terminal Tabs Help  
pft@PFT-VirtualBox:/tmp$ python3 -m pytest testcase.py  
===== test session starts =====  
platform linux -- Python 3.6.7, pytest-4.0.1, py-1.7.0, pluggy-0.8.0  
rootdir: /tmp, inifile:  
collected 1 item  
  
testcase.py . [100%]  
  
===== 1 passed in 0.06 seconds =====  
pft@PFT-VirtualBox:/tmp$
```

## Group multiple tests in a class

Once you develop multiple tests, you may want to group them into a class. pytest makes it easy to create a class containing more than one test:

```
class TestClass(object):  
    def test_one(self):  
        x = "this"  
        assert 'h' in x  
  
    def test_two(self):  
        x = "hello"  
        assert hasattr(x, 'check')
```

Pytest supports multiple ways of specifying and selecting which test cases, specifically executing the testclass above would be:

```
python3 -m pytest testfile.py::TestClass
```

For other options see: <https://docs.pytest.org/en/latest/usage.html>

## Mocking and overriding objects in Python

One method will have a dependency on the underlying database, since we’re writing unit tests we want to avoid dependencies on external components. In

this case you will need to override/mock the method in question. Luckily in Python this is as simple as assigning your own method to the method you want to mock.

What are the downsides of mocking things when testing code? What are the benefits, why do we do it?

Below is an example of a simplified class to send emails. The external dependency to communicate with the SMTP server is something we want to avoid in a unit test, so that will need to be overridden.

```
import smtplib

class SendEmail:

    def send_email(self, email, text, subject):

        # create a message
        msg = MIMEMultipart()
        setup the parameters of the message
        msg['From']=MY_ADDRESS
        msg['To']=email
        msg['Subject']=

        # add in the message body
        msg.attach(MIMEText(message, text))

        setup_connection_and_send(msg)

    def setup_connection_and_send(self, msg):

        # set up the SMTP server
        s = smtplib.SMTP(host='smtp-mail.bth.se', port=587)
        s.starttls()
        s.login(MY_ADDRESS, PASSWORD)

        # send the message via the server set up earlier.
        try:
            s.send_message(msg)
        except Exception as e:
            raise

        return True
```

Below is how you could write a test case overriding the external dependency in the `setup_connection_and_send()` method. Everything in Python is an object, even methods. That makes it very easy to override any object with our own.

```

import smtplib

def setup_connection_and_send_override(self, msg):
    return True

def test_send_email():
    se = SendEmail()
    se.setup_connection_and_send = setup_connection_and_send_override

    assert se.send_email("teacher@bth.se", "Greetings from PA1417", "Hello teacher!") == True

```

## How to measure the code coverage of code in Python

There are several code coverage modules available for Python. We’re going to use “Coverage”. Coverage is the most common choice when measuring code coverage in Python, and Pytest provides a plugin “*pytest-cov*” to ease the integration of running test cases and generating a code coverage report.

Just like “Requests” it’s a 3rdparty module we need to install, however since *pytest-cov* depends in coverage we simply need to install the former:

```
pip install pytest-cov
```

Next thing to do is to run our test cases and to instruct *pytest-cov* where the source code for the system that’s being tested resides:

```
python3 -m pytest --cov=src/ tests/
```

This will result in a printout to terminal of the coverage, and to generate an html report, simply run:

```
coverage html
```

Now a folder is created called *htmlcov* which will contain “index.html” and all the other classes that were invoked with corresponding html-files. Open index.html in a browser.

## References and further reading

For more information on Pytest, see: <https://docs.pytest.org/en/latest/contents.html>

For more information on Coverage, see: <https://coverage.readthedocs.io/en/v4.5.x/>

For more information on pytest-cov, see: <https://pypi.org/project/pytest-cov/>  
Thursday, 28. March 2019 09:26PM