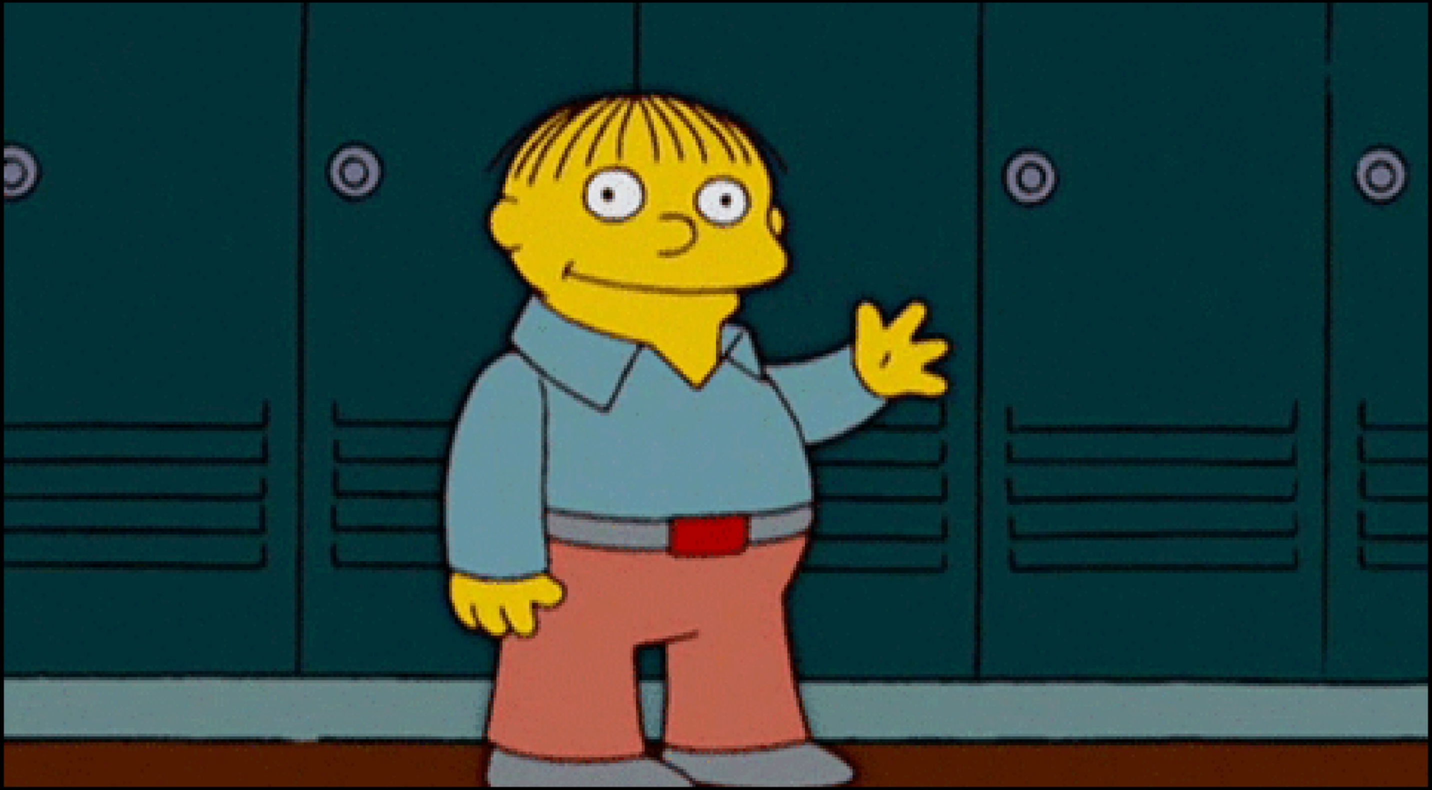
Testing, testing everywhere!

Toni Robres Turón

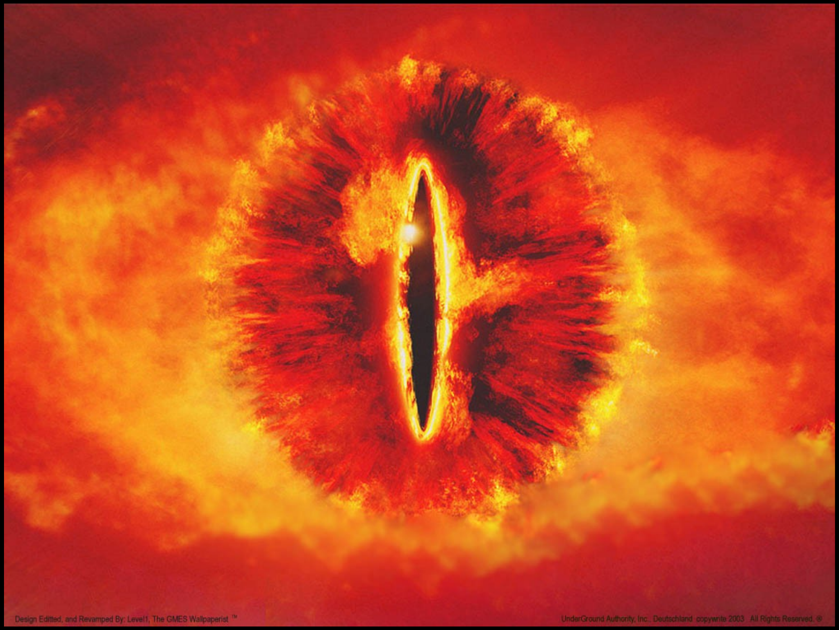
@twiindan

{name: Toni Robres}



{role: QA}

Three tools for the frontend testers under UI
Seven for the backend testers in their APIs
Nine for the Performance testers doomed to kill systems.


Design Editted, and Revamped By: Level1, The GMES Wallpaperist ™
UnderGround Authority, Inc. Deutschland copywrite 2003 All Rights Reserved. ®

One for the dark load of software enginering
One tool to rule them all, One tool never found by testers...
Untill now

Sometimes diversity is not good

Sometimes diversity is not good

Confrontation

Which characteristics should have the perfect testing tool?

REUSE

REDUCE

RECYCLE

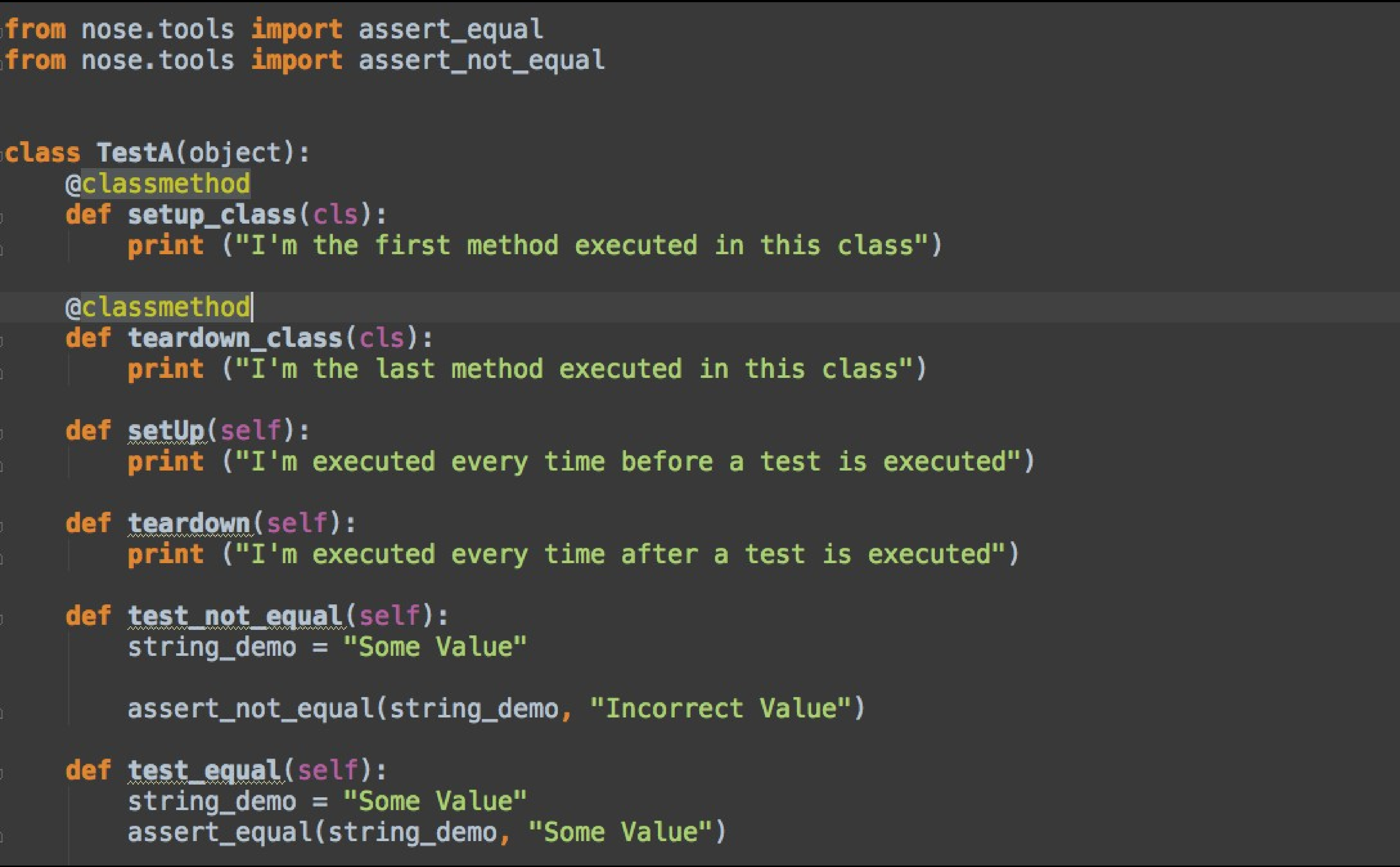
python™

Testing Activities

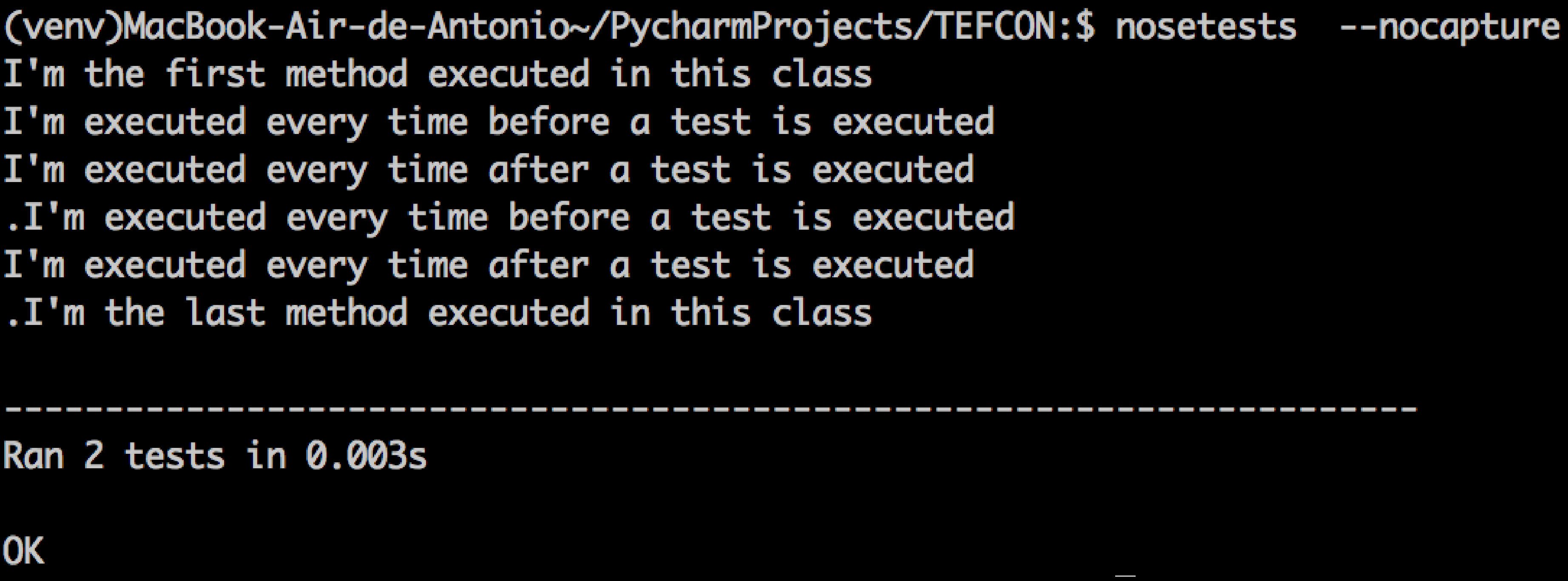
- Unit Testing
- Acceptance Testing
- Web Testing
- Performance Testing
- Test Definition

Unit Testing

- Nose
  - Extended framework for python unit testing
  - Easy to write and run tests
  - Provides coverage
  - Provides profiler
  - Test can be organized
  - Include tools for testing

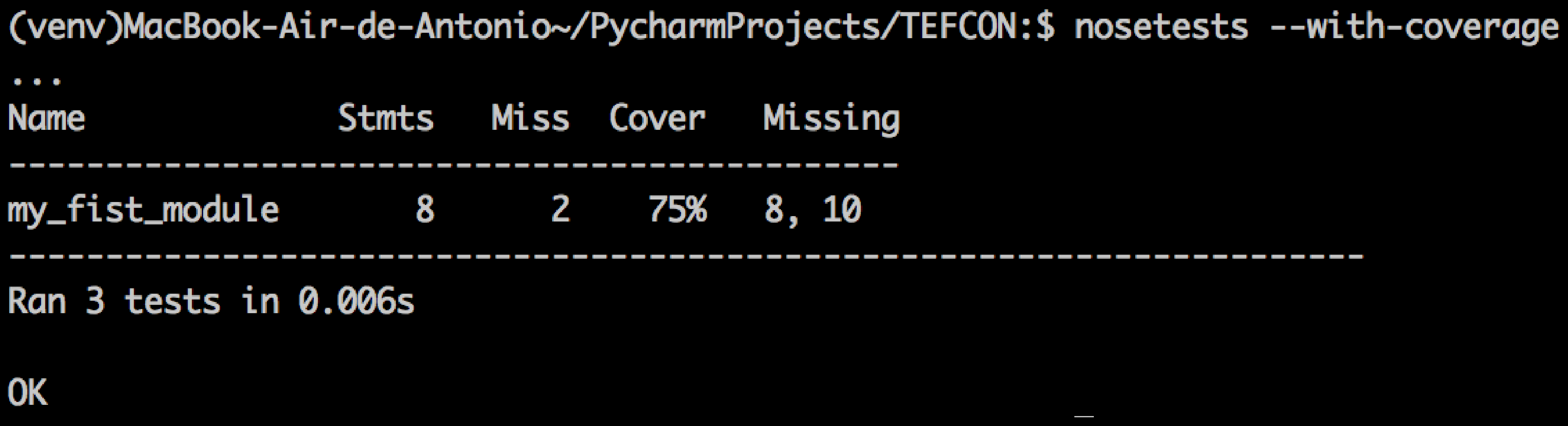

```python
from nose.tools import assert_equal
from nose.tools import assert_not_equal


class TestA(object):
    @classmethod
    def setup_class(cls):
        print ("I'm the first method executed in this class")

    @classmethod
    def teardown_class(cls):
        print ("I'm the last method executed in this class")

    def setUp(self):
        print ("I'm executed every time before a test is executed")

    def teardown(self):
        print ("I'm executed every time after a test is executed")

    def test_not_equal(self):
        string_demo = "Some Value"

        assert_not_equal(string_demo, "Incorrect Value")

    def test_equal(self):
        string_demo = "Some Value"
        assert_equal(string_demo, "Some Value")
```

EXECUTE THE TESTS



```
(venv)MacBook-Air-de-Antonio~/PycharmProjects/TEFCON:$ nosetests  --nocapture
I'm the first method executed in this class
I'm executed every time before a test is executed
I'm executed every time after a test is executed
.I'm executed every time before a test is executed
I'm executed every time after a test is executed
.I'm the last method executed in this class

----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

Coverage



```
(venv)MacBook-Air-de-Antonio~/PycharmProjects/TEFCON:$ nosetests --with-coverage
...
Name                 Stmts   Miss  Cover   Missing
----------------------------------------------
my_fist_module           8      2    75%   8, 10
----------------------------------------------------------------------
Ran 3 tests in 0.006s

OK
```

API REST Testing

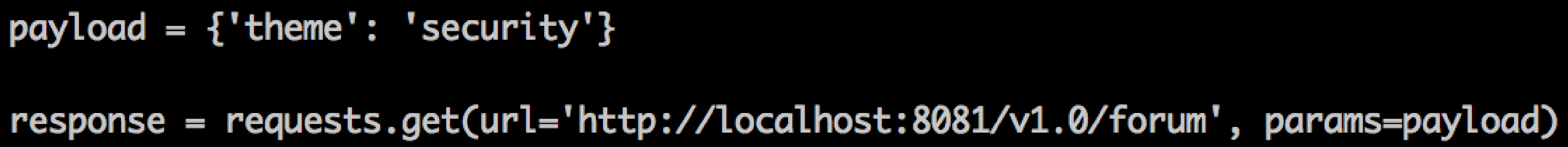
- Request: HTTP For humans
  - Library to perform API REST requests
  - Easy to use
  - Basic and Oauth Authentication
  - Cookies support
  - Multipart Files Upload
  - Session objects
  - Verify SSL Certificates
  - Proxies
  - Can be integrated with nose and lettuce

Basic usage

```
In [2]: import requests

In [3]: response = requests.get('http://localhost:8081/v1.0')

In [4]: response.ok
Out[4]: True

In [5]: response.status_code
Out[5]: 200

In [6]: response.content
Out[6]: '{"product": "forum", "version": "0.2.0"}'

In [7]: body = response.json()

In [8]: body['product']
Out[8]: u'forum'

In [9]: response_header = response.headers

In [10]: response_header['content-type']
Out[10]: 'application/json'
```

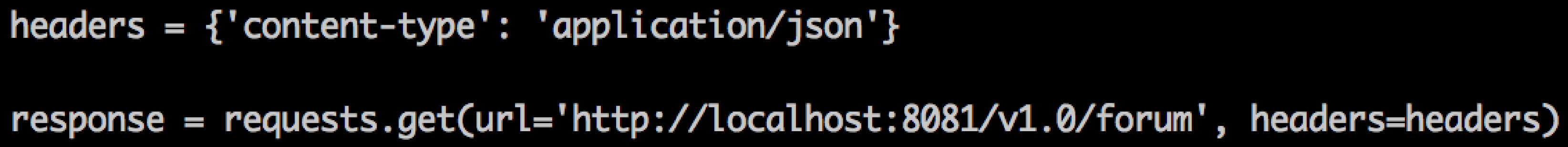
Usage

- Query Parameters defined as Python Dict:



```
payload = {'theme': 'security'}

response = requests.get(url='http://localhost:8081/v1.0/forum', params=payload)
```

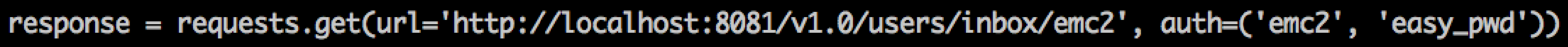
- Custom headers defined as Python Dict



```
headers = {'content-type': 'application/json'}

response = requests.get(url='http://localhost:8081/v1.0/forum', headers=headers)
```

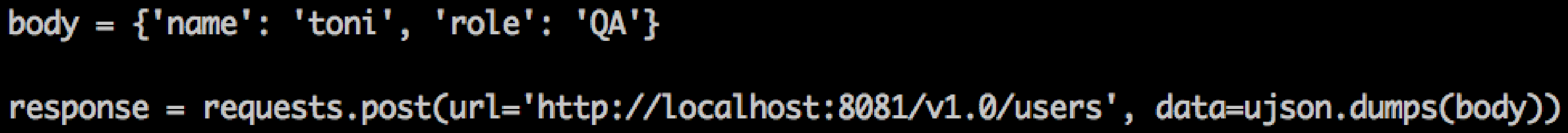
Usage

Basic authentication



```
response = requests.get(url='http://localhost:8081/v1.0/users/inbox/emc2', auth=('emc2', 'easy_pwd'))
```

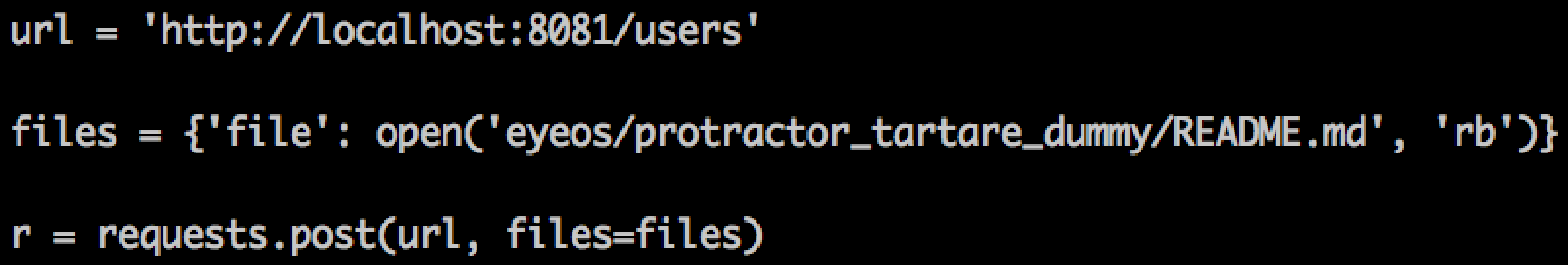
Content body defined as Python Dict



```
body = {'name': 'toni', 'role': 'QA'}

response = requests.post(url='http://localhost:8081/v1.0/users', data=ujson.dumps(body))
```

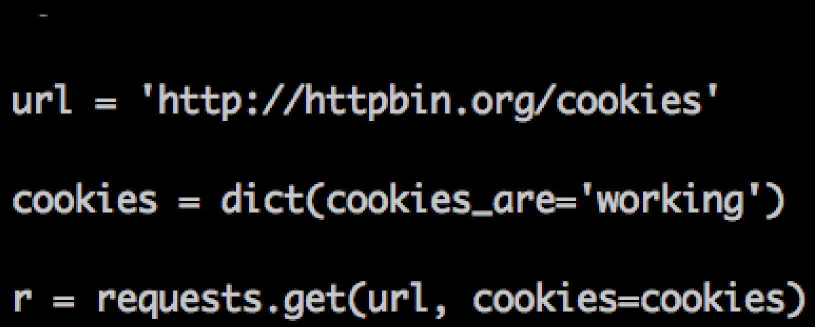
Usage

- Upload a file:



```
url = 'http://localhost:8081/users'

files = {'file': open('eyeos/protractor_tartare_dummy/README.md', 'rb')}

r = requests.post(url, files=files)
```

- Cookies



```
url = 'http://httpbin.org/cookies'

cookies = dict(cookies_are='working')

r = requests.get(url, cookies=cookies)
```

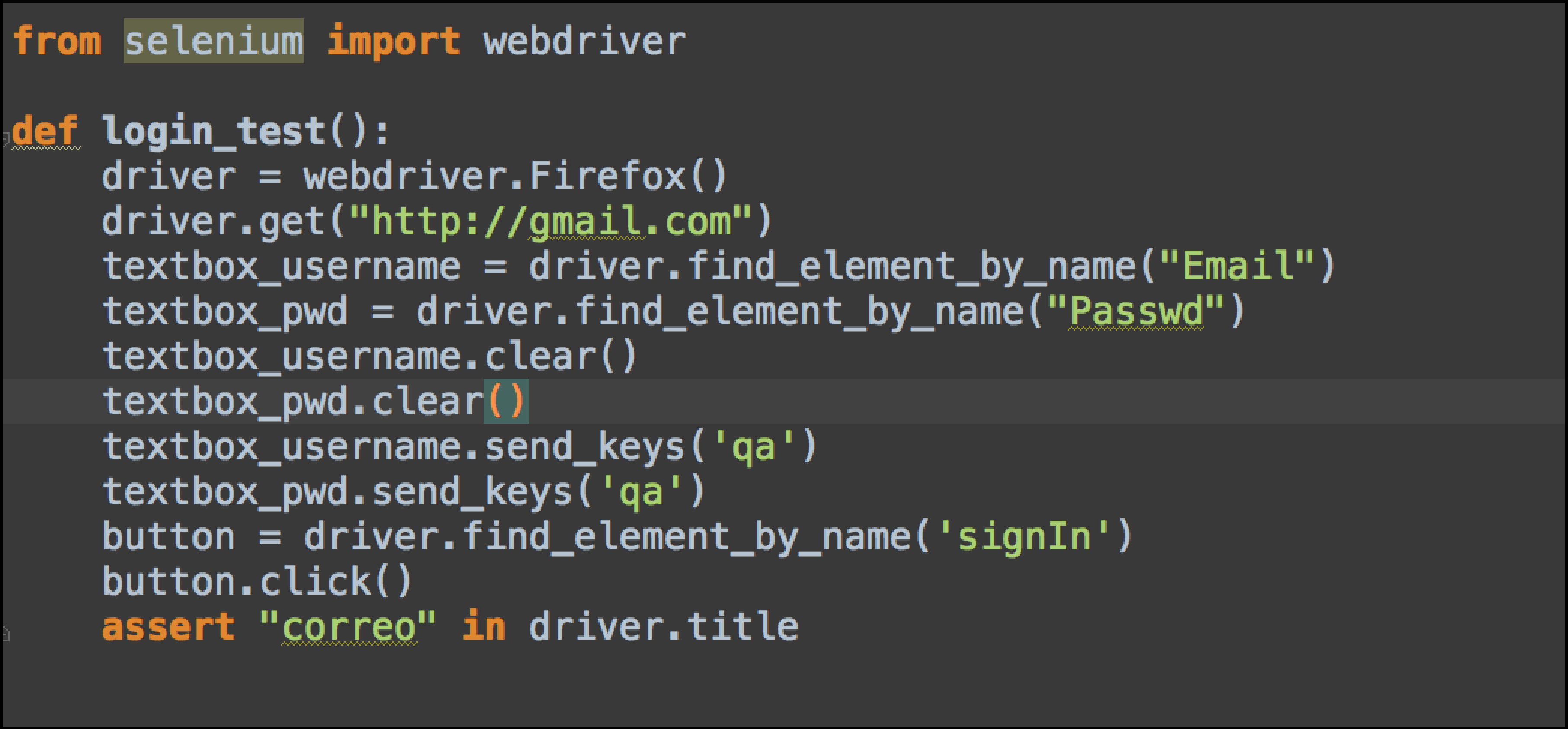
Web Testing

- Selenium
  - Most extended library to test Web GUI
  - Suport Firefox, Chrome and Internet Explorer
  - Can be integrated with nose and lettuce
  - Integrated with CI
  - Grid support
  - Cookies support

Selenium

- How it works?
  - Locate the Elements
    - By id, CSS, XPATH, name, Class…
  - Select Elements
    - Assert properties
  - Interact
    - Send keys
    - Click

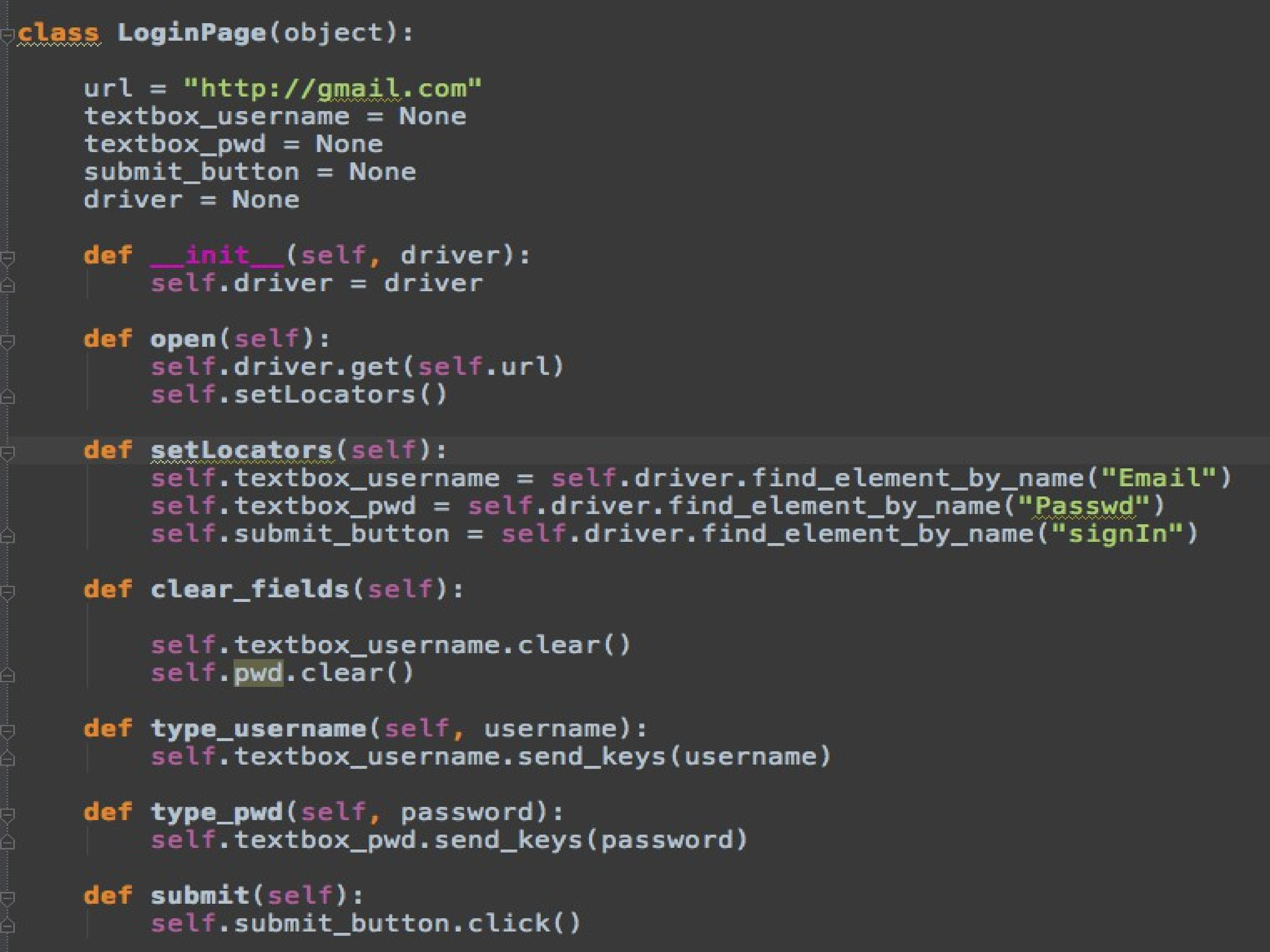
Basic Example



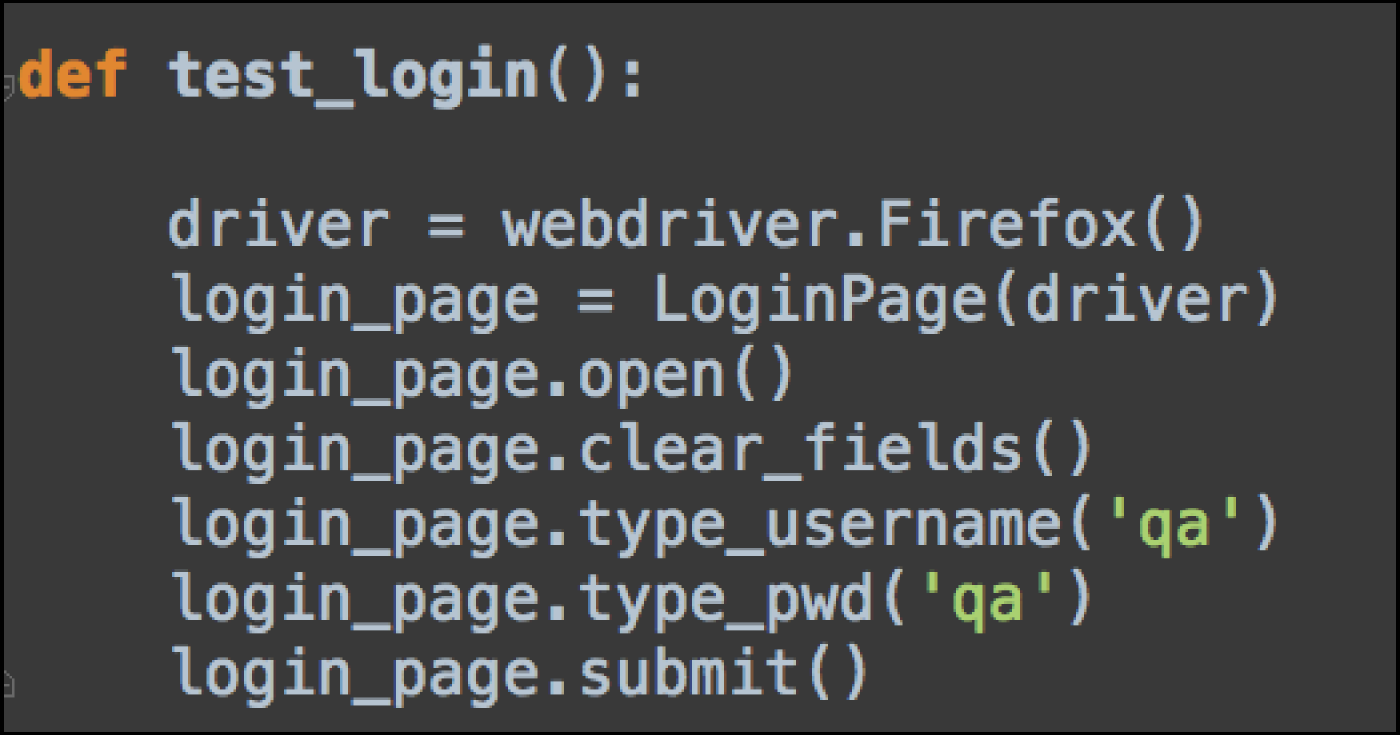
```
from selenium import webdriver

def login_test():
    driver = webdriver.Firefox()
    driver.get("http://gmail.com")
    textbox_username = driver.find_element_by_name("Email")
    textbox_pwd = driver.find_element_by_name("Passwd")
    textbox_username.clear()
    textbox_pwd.clear()
    textbox_username.send_keys('qa')
    textbox_pwd.send_keys('qa')
    button = driver.find_element_by_name('signIn')
    button.click()
    assert "correo" in driver.title
```

Page Object Pattern

- Language Neutral Pattern for representing a web page in an Object Oriented manner
- Necessary for survive in Selenium
  - Increase maintanability
  - Increase readability
  - Abstract web page logical from tests



```python
class LoginPage(object):

    url = "http://gmail.com"
    textbox_username = None
    textbox_pwd = None
    submit_button = None
    driver = None

    def __init__(self, driver):
        self.driver = driver

    def open(self):
        self.driver.get(self.url)
        self.setLocators()

    def setLocators(self):
        self.textbox_username = self.driver.find_element_by_name("Email")
        self.textbox_pwd = self.driver.find_element_by_name("Passwd")
        self.submit_button = self.driver.find_element_by_name("signIn")

    def clear_fields(self):

        self.textbox_username.clear()
        self.pwd.clear()

    def type_username(self, username):
        self.textbox_username.send_keys(username)

    def type_pwd(self, password):
        self.textbox_pwd.send_keys(password)

    def submit(self):
        self.submit_button.click()
```



```python
def test_login():

    driver = webdriver.Firefox()
    login_page = LoginPage(driver)
    login_page.open()
    login_page.clear_fields()
    login_page.type_username('qa')
    login_page.type_pwd('qa')
    login_page.submit()
```

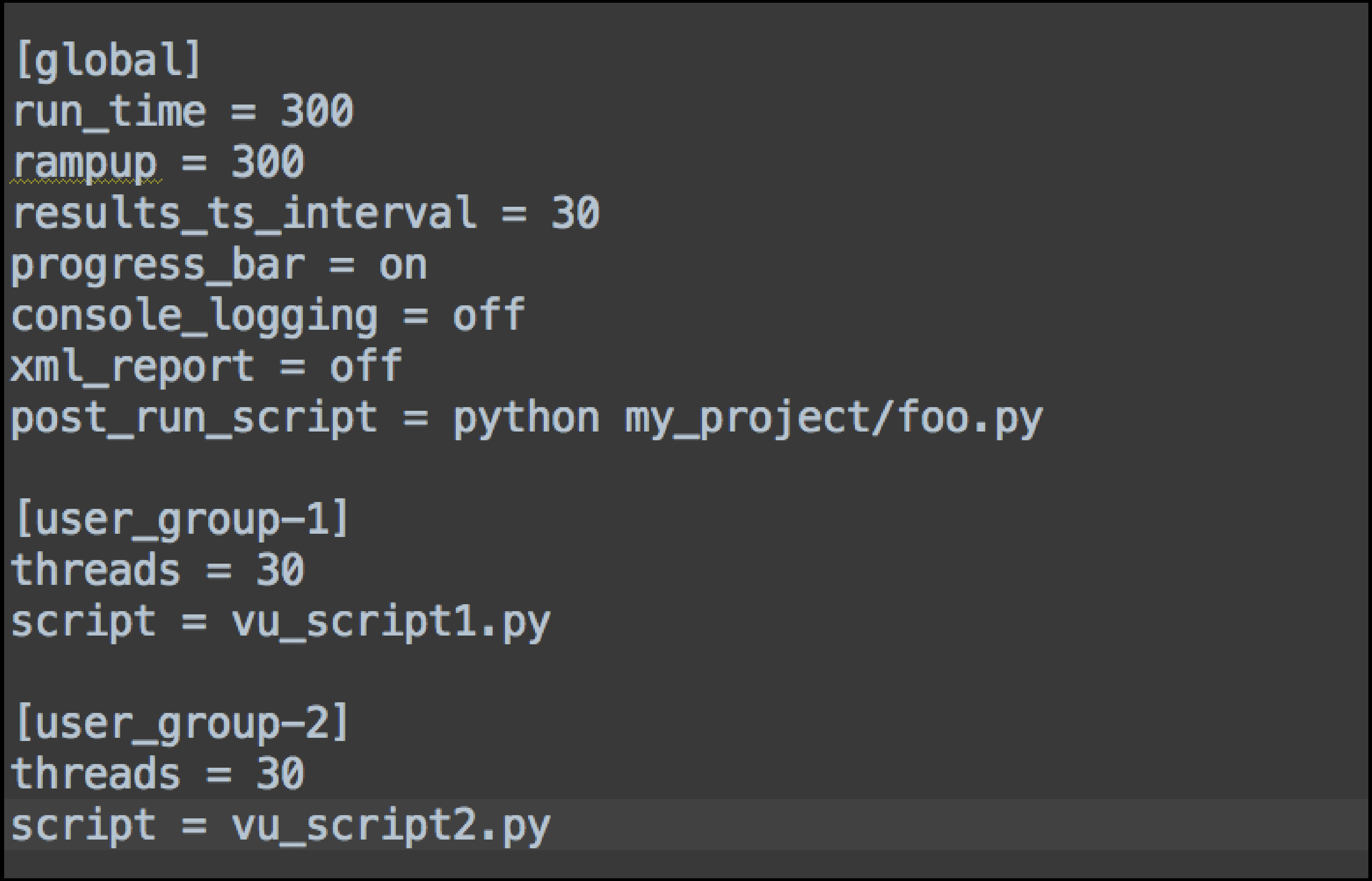
Web Testing

- What happen with selenium IDE?

Perfomance Testing

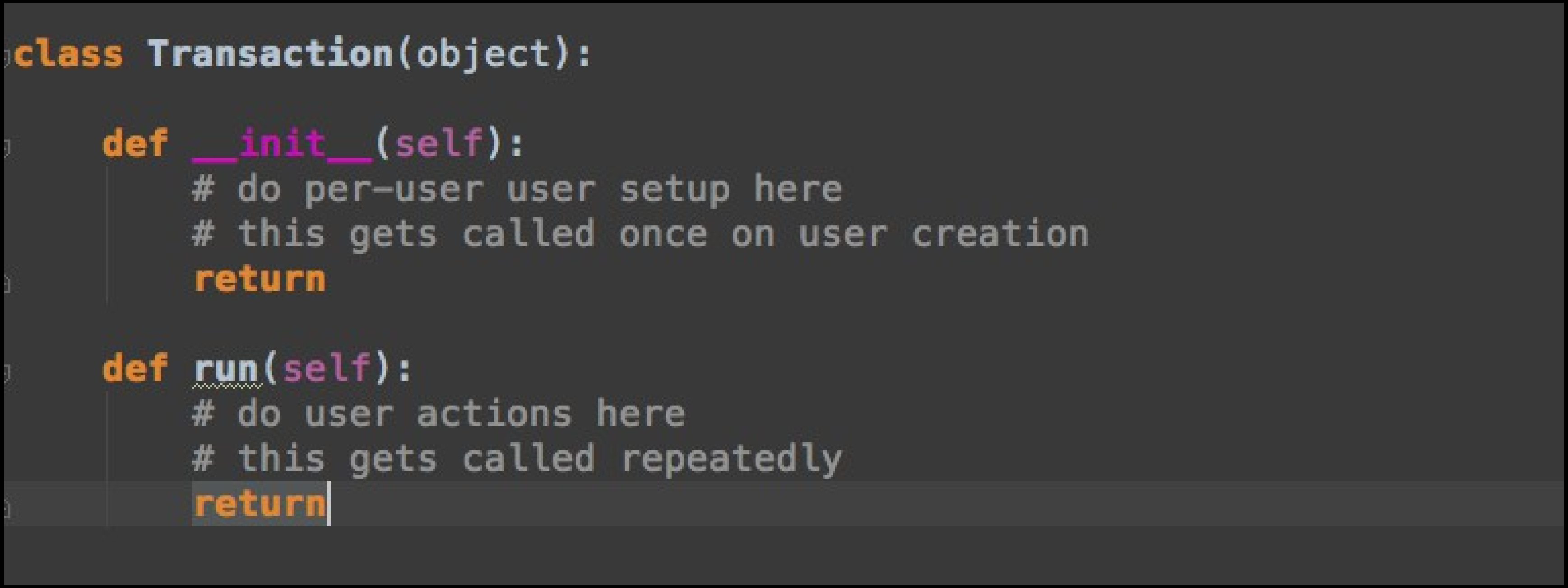
- MultiMechanize
  - Runs concurrent Python scripts to generate load against service
  - Reporting Jmeter compatible
  - Easy configuration
  - Can reuse Custom Request library
  - Multithreading and multiprocessing
  - Distributed

Config File



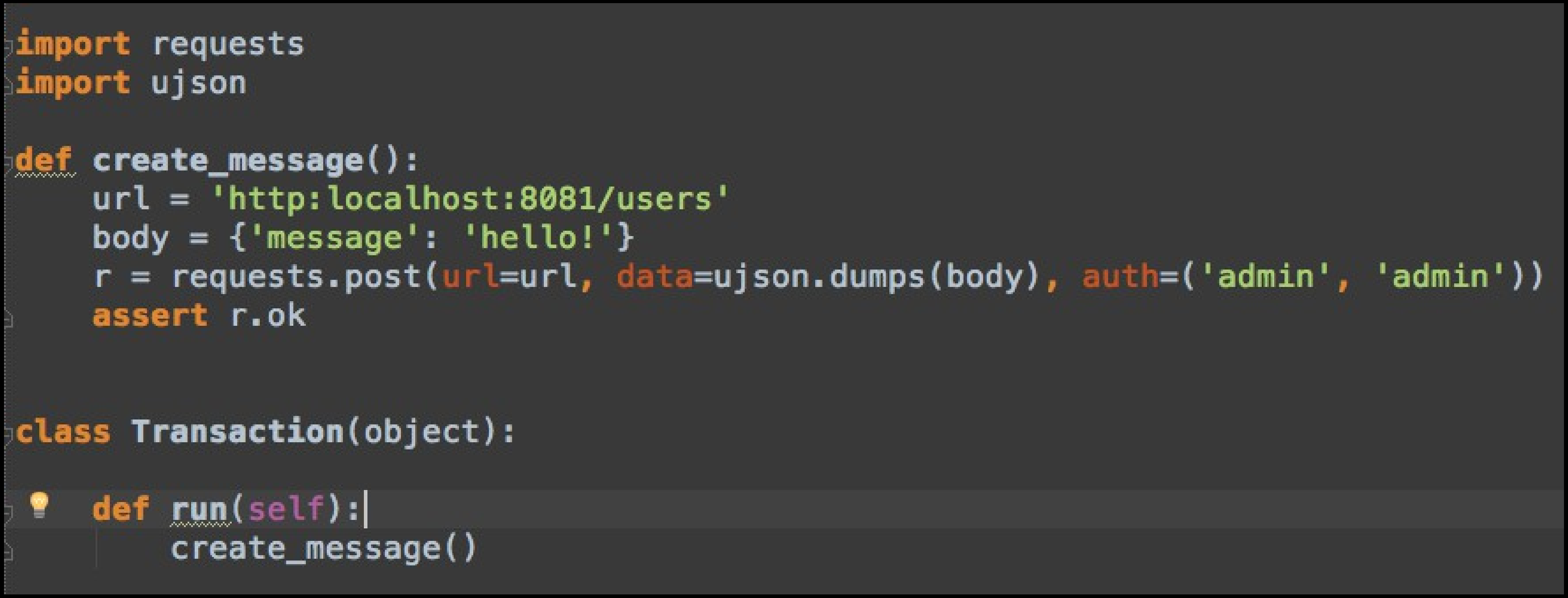
```
[global]
run_time = 300
rampup = 300
results_ts_interval = 30
progress_bar = on
console_logging = off
xml_report = off
post_run_script = python my_project/foo.py

[user_group-1]
threads = 30
script = vu_script1.py

[user_group-2]
threads = 30
script = vu_script2.py
```

Script File



```python
class Transaction(object):

    def __init__(self):
        # do per-user user setup here
        # this gets called once on user creation
        return

    def run(self):
        # do user actions here
        # this gets called repeatedly
        return
```

Example script File

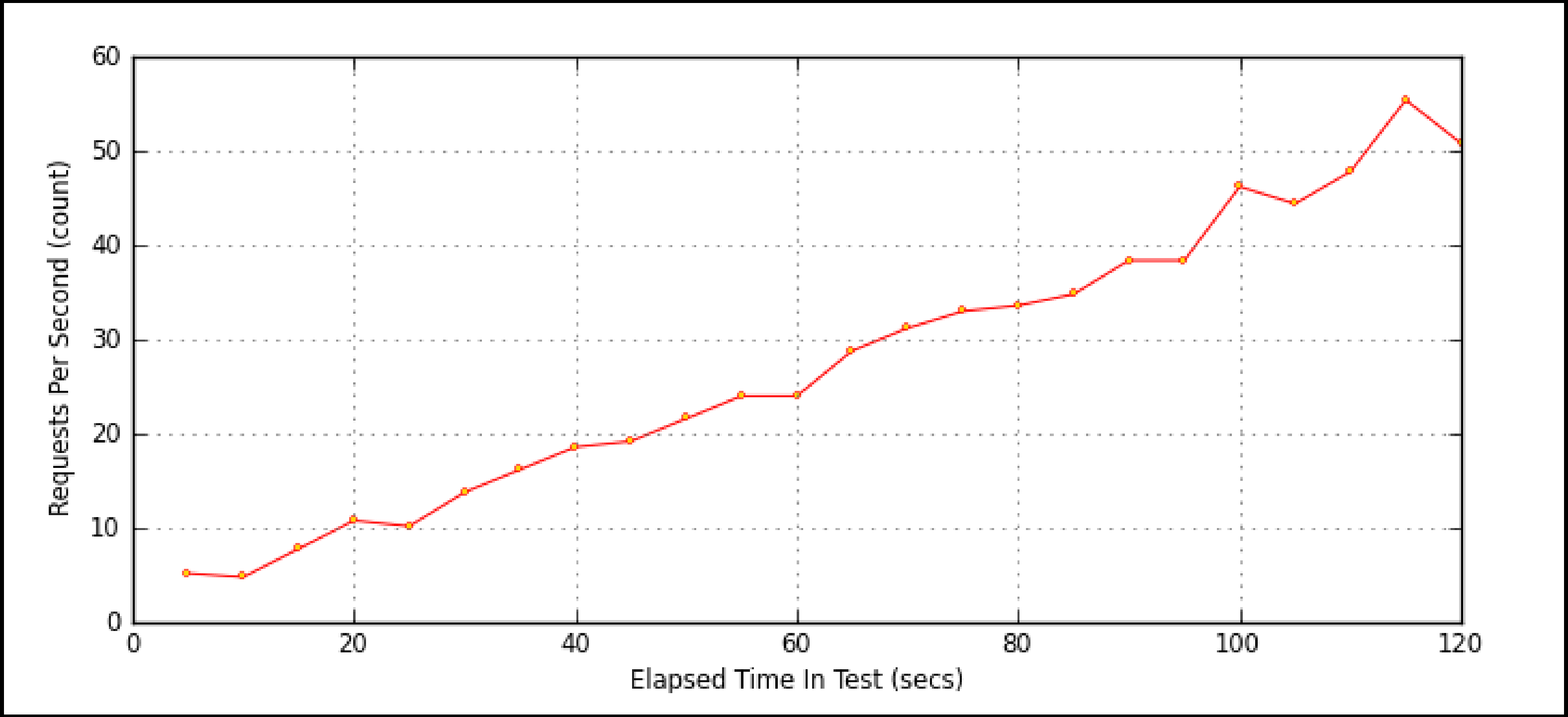


```
import requests
import ujson

def create_message():
    url = 'http:localhost:8081/users'
    body = {'message': 'hello!'}
    r = requests.post(url=url, data=ujson.dumps(body), auth=('admin', 'admin'))
    assert r.ok


class Transaction(object):

    def run(self):
        create_message()
```

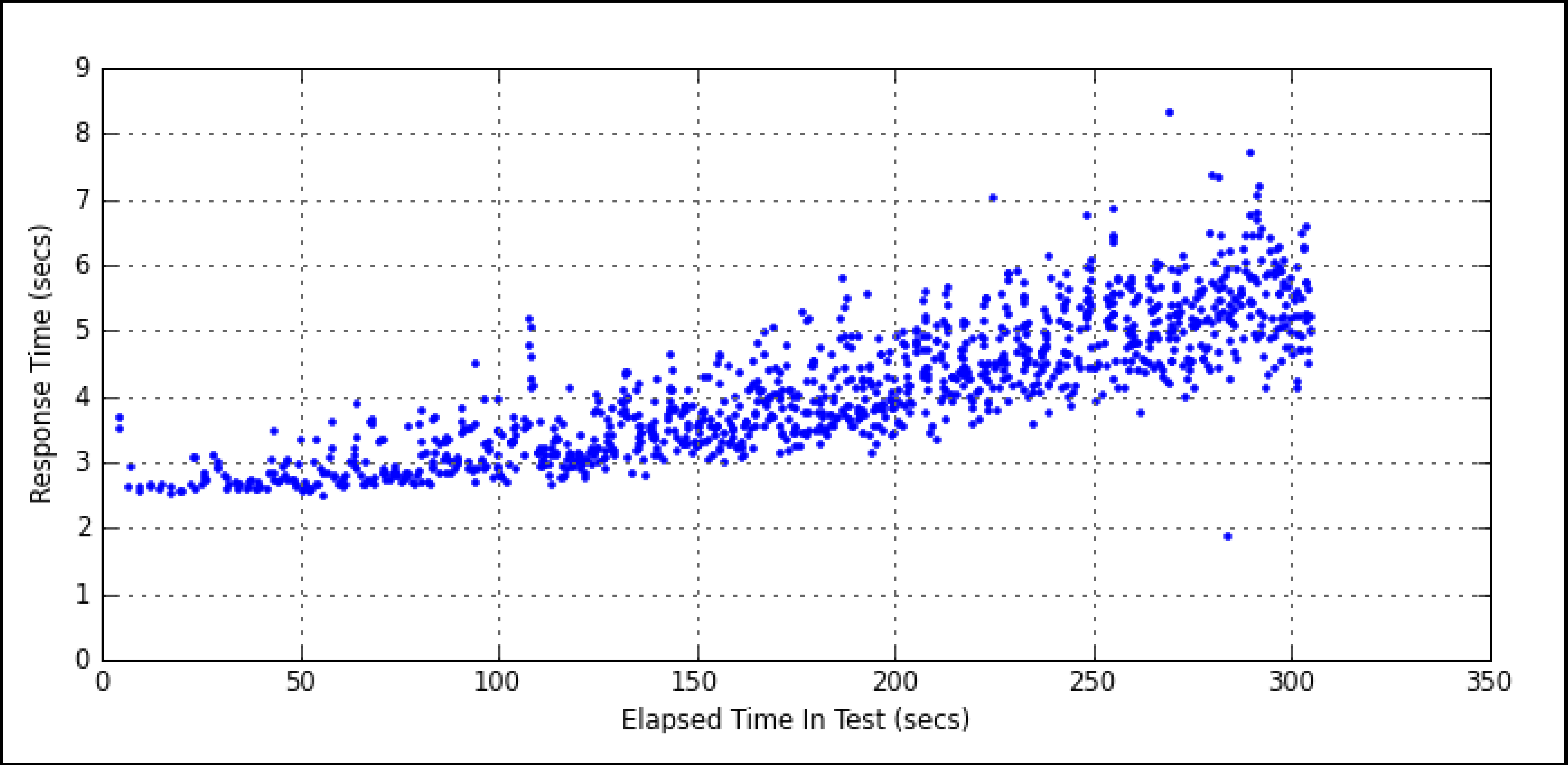
Multi Mechanize Stats

- test summary
- transaction timers
- custom timers (from instrumented client code)
- time-series/interval data
- counts
- rate/throughput
- response times
- average, min, max, stdev
- percentiles (80th, 90th, 95th)

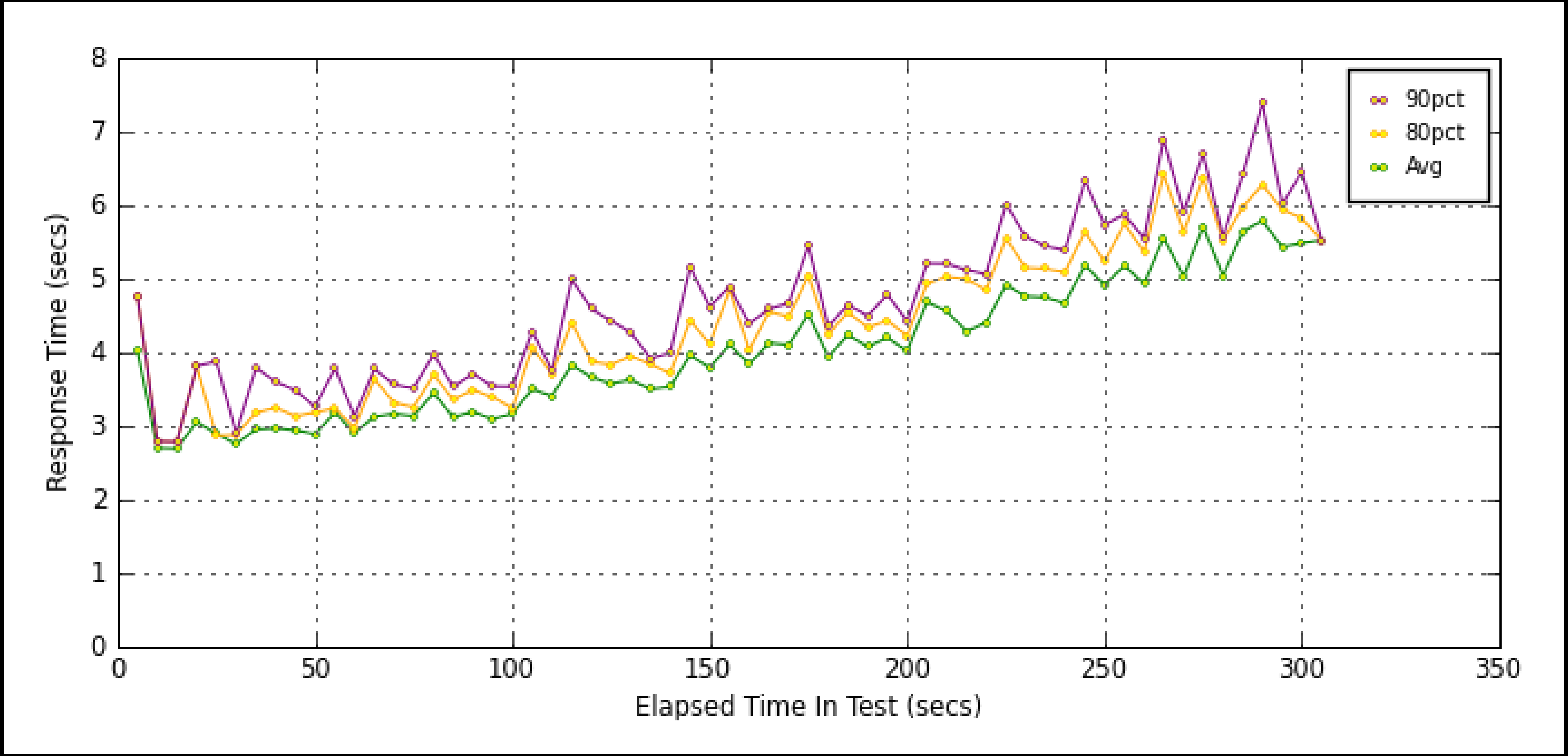
Graphs


Requests Per Second (count)
Elapsed Time In Test (secs)
0
10
20
30
40
50
60
0
20
40
60
80
100
120

Graphs


Response Time (secs)
Elapsed Time In Test (secs)
0
1
2
3
4
5
6
7
8
9
0
50
100
150
200
250
300
350

Graphs



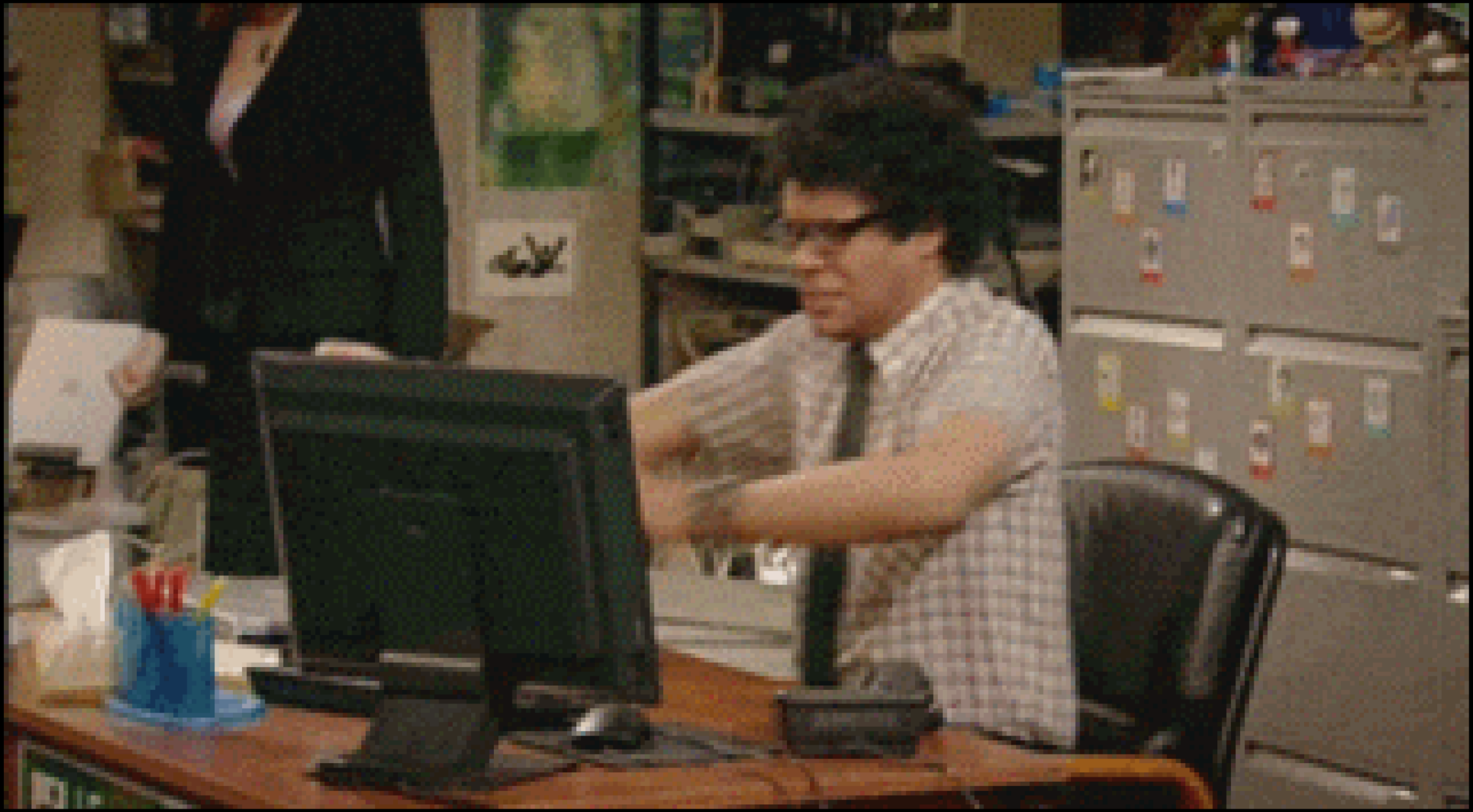
90pct
80pct
Avg
Response Time (secs)
Elapsed Time In Test (secs)
0
1
2
3
4
5
6
7
8
50
100
150
200
250
300
350

Summarizing

- I can do tests in all Levels:
  - Web
  - API
  - Performance

- What happen with test Definition and test Execution stats?

Jira / TestLink / IBM



BDD

- Using examples to create a shared understanding and surface uncertainly to deliver software that matters.
- Define the software behaviour:
  - Given (Preconditions)
  - When (actions)
  - Then (Post conditions)

Lettuce

- BDD Tool for Python
- Easy to integrate with tests developed with Request and Webdriver
- Data driven
- Using decorators to execute functions that describes the software behaviour

Feature Example

Scenario Outline: Retrieve the geolocation with city name given

Given a <city> name

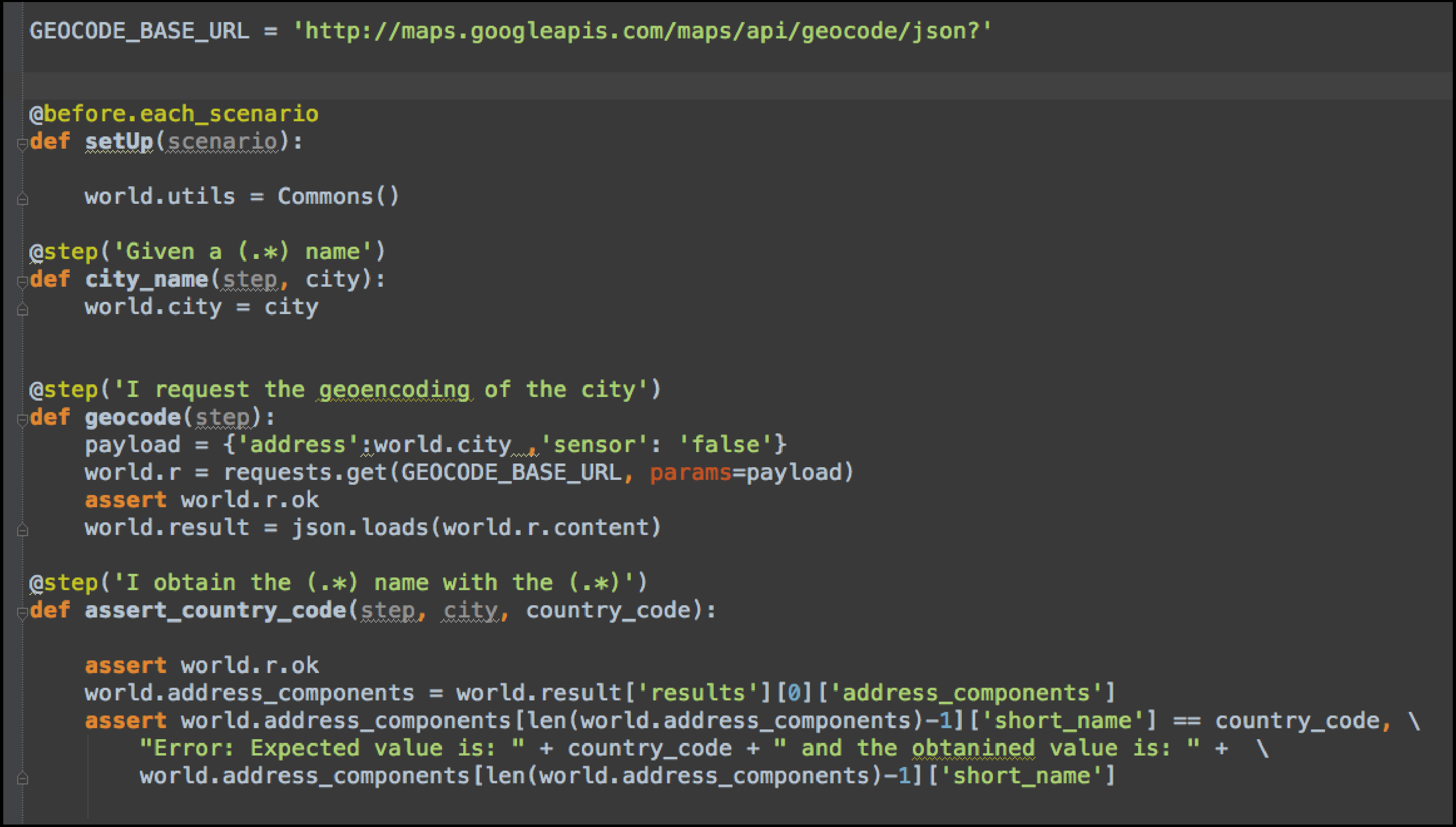
When I request the geoencoding of the city

Then I obtain the <city> name with the <country_code>

Examples:

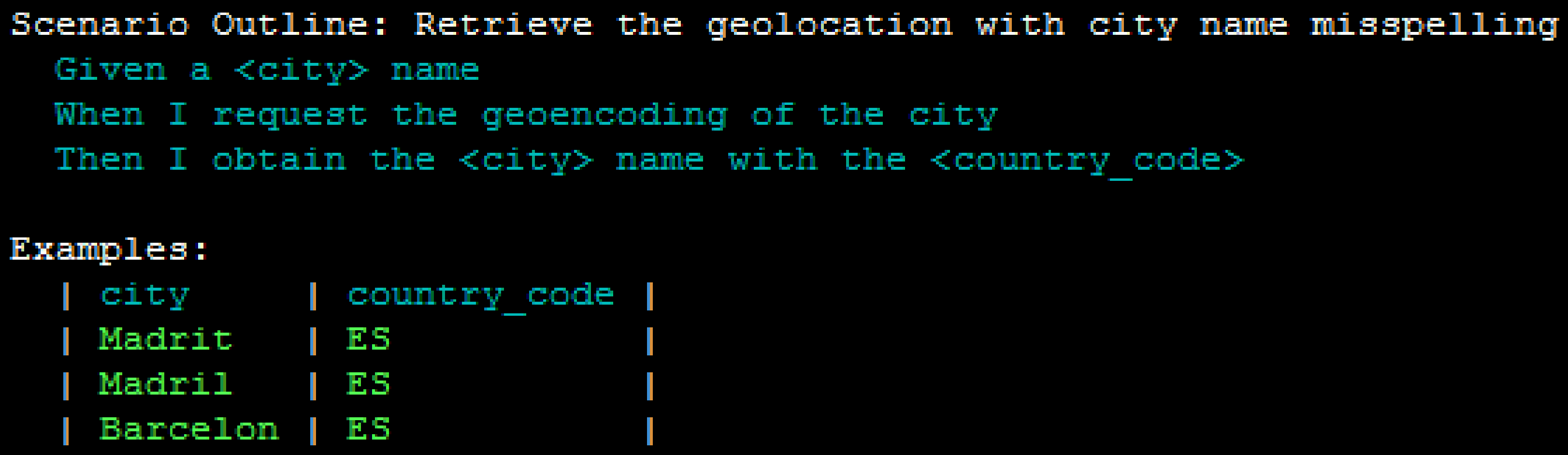
city	country_code
Barcelona	ES
Paris	FR
San+Francisco	US

Coding example

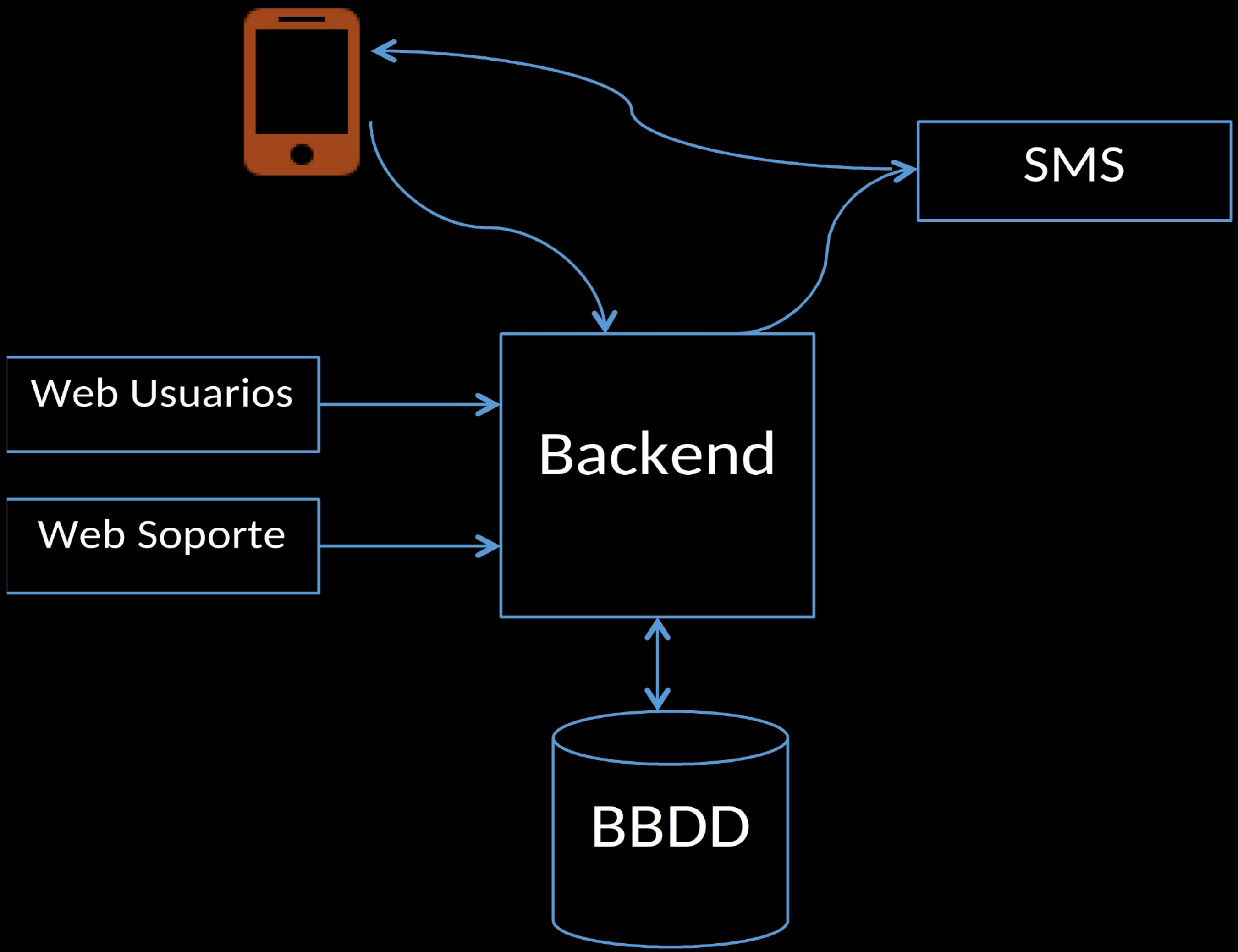


```python
GEOCODE_BASE_URL = 'http://maps.googleapis.com/maps/api/geocode/json?'


@before.each_scenario
def setUp(scenario):

    world.utils = Commons()

@step('Given a (.*) name')
def city_name(step, city):
    world.city = city


@step('I request the geoencoding of the city')
def geocode(step):
    payload = {'address':world.city ,'sensor': 'false'}
    world.r = requests.get(GEOCODE_BASE_URL, params=payload)
    assert world.r.ok
    world.result = json.loads(world.r.content)

@step('I obtain the (.*) name with the (.*)')
def assert_country_code(step, city, country_code):

    assert world.r.ok
    world.address_components = world.result['results'][0]['address_components']
    assert world.address_components[len(world.address_components)-1]['short_name'] == country_code, \
        "Error: Expected value is: " + country_code + " and the obtanined value is: " +  \
        world.address_components[len(world.address_components)-1]['short_name']
```

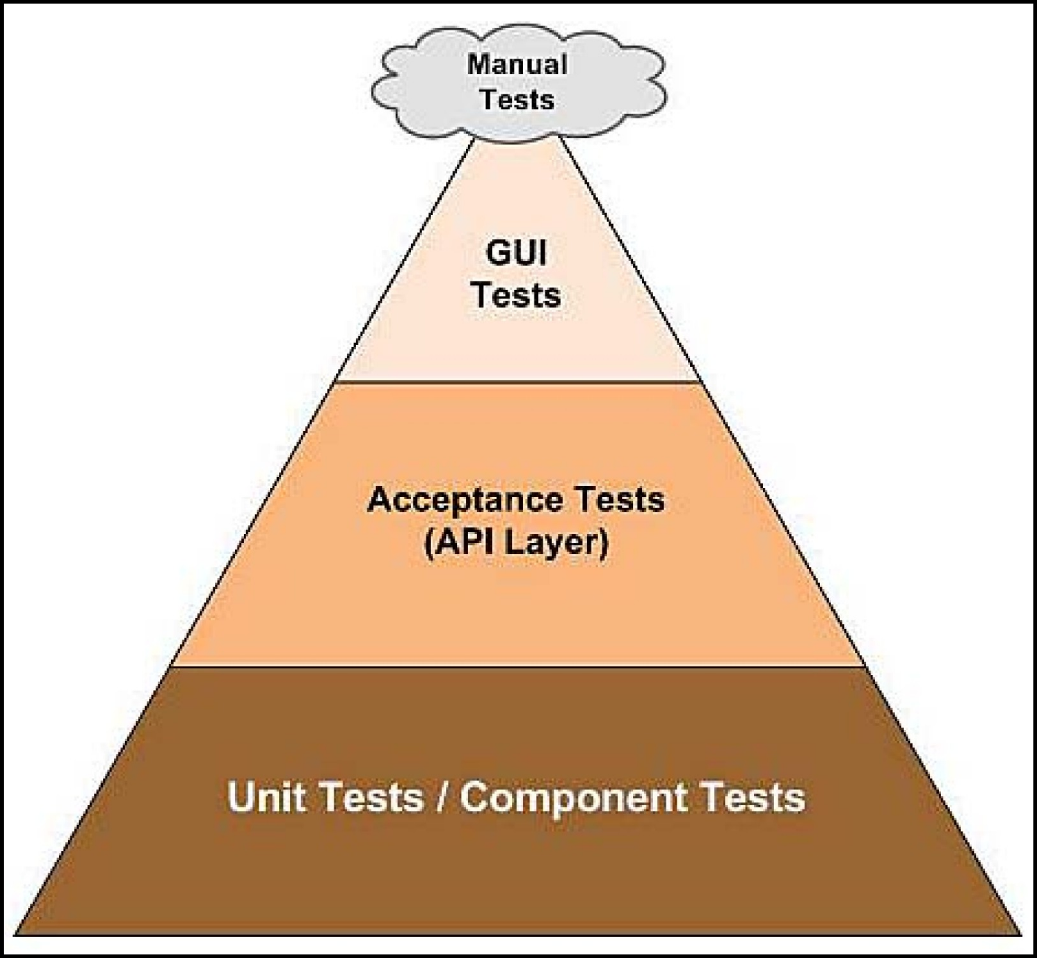
Test Runner and report



```
Scenario Outline: Retrieve the geolocation with city name misspelling
  Given a <city> name
  When I request the geoencoding of the city
  Then I obtain the <city> name with the <country_code>

Examples:
   | city     | country_code |
   | Madrit   | ES           |
   | Madril   | ES           |
   | Barcelon | ES           |
```


SMS
Web Usuarios
Backend
Web Soporte
BBDD

Que probar y con que?



Que probar y con que?

- Unit testing all components
- Component test:
  - Backend Requests
  - Webs Webdriver mocking the backend
  - Mobile Appium mocking the backend
- Integration:
  - Webs and backend
  - Mobile and backend
  - Backend with SMS plattform
- E2E

Bonus Track

- What happen if my component has different interface than API REST?
  - All the components always have an input
  - For example
    - Rabbit ▯ Pika, Kombu
    - MongoDB ▯ Pymongo
    - Redis ▯ Python Redis client
    - MySQL ▯ sqlite, sqlalchemy

Overview

- Using Python for all testing activities
  - Easy to integrate
  - Can reuse common libraries
  - Only needs learn one tool
  - Collaboration between development and testing
  - Community

Result

Questions?