Testing PySide/PyQt Code

Using the pytest framework and pytest-qt

Florian Bruhin
"The Compiler"

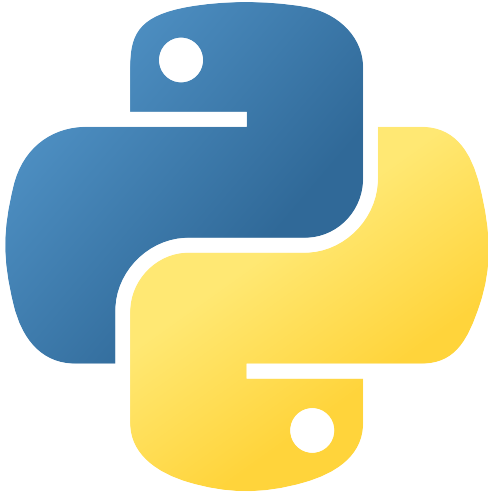
Bruhin Software

06. November 2019
Qt World Summit, Berlin

About me

- 2011: Started using Python
- 2013: Started using PyQt and developing qutebrowser
- 2015: Switched to pytest, ended up as a maintainer
- 2017: qutebrowser v1.0.0, QtWebEngine by default
- 2019: 40% employed, 60% open-source and freelancing (Bruhin Software)

Giving trainings and talks at various conferences and companies!



Relevant Python features

Decorators

```
registered_functions: List[Callable] = []

def register(f: Callable) -> Callable:
    registered_functions.append(f)
    return f

@register
def func() -> None:
    ....
```

Relevant Python features

Context Managers

```
def write_file() -> None:
    with open("test.txt", "w") as f:
        f.write("Hello World")
```

Defining your own:
Object with special `__enter__` and `__exit__` methods.

Relevant Python features

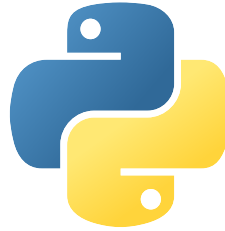
Generators/yield

```
def gen_values() -> Iterable[int]
    for i in range(4):
        yield i

print(gen_values())
# <generator object gen_values at 0x...>

print(list(gen_values()))
# [0, 1, 2, 3]
```


Qt

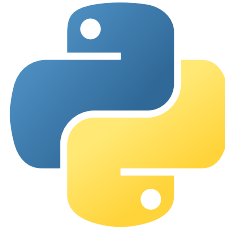
PyQt

- Started in 1998 (!) by Riverbank Computing
- GPL/commercial
- Qt4 $\leftrightarrow$ PyQt4
  Qt5 $\leftrightarrow$ PyQt5



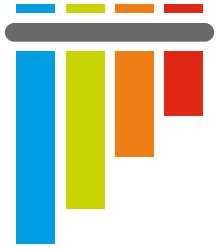
PySide / Qt for Python

- Started in 2009 by Nokia
- Unmaintained for a long time
- Since 2016: Officially maintained by the Qt Company again
- LGPL/commercial
- Qt4 $\leftrightarrow$ PySide
  Qt5 $\leftrightarrow$ PySide2 (Qt for Python)



Qt and Python

```
import sys

from PySide2.QtWidgets import QApplication, QWidget, QPushButton

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = QWidget()
    button = QPushButton("Don't press me!", window)
    button.clicked.connect(app.quit)
    window.show()
    app.exec_()
```



pytest

Why pytest?

Example code

```
def add_two(val: int) -> int:
    return val + 2
```

Why pytest?

Test with unittest.py

```
import unittest

class AddTwoTests(unittest.TestCase):

    def testAddTwo(self):
        self.assertEqual(add_two(2), 4)

if __name__ == '__main__':
    unittest.main()
```

Why pytest?

Test with pytest

```
def test_add_two():
    assert add_two(2) == 4
```

Why pytest?

Parametrizing

```
import pytest

@pytest.mark.parametrize('inp, out', [
    (2, 4),
    (3, 5),
])
def test_add_two(inp, out):
    assert add_two(inp) == out
```

pytest fixtures

Basic example

```
import pytest

@pytest.fixture
def answer():
    return 42

def test_answer(answer):
    assert answer == 42
```

pytest fixtures

Basic example

```python
import pytest

@pytest.fixture
def answer():
    return 42

def test_answer(answer):
    assert answer == 42
```

pytest fixtures

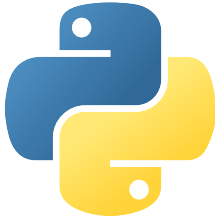
Fixtures using fixtures

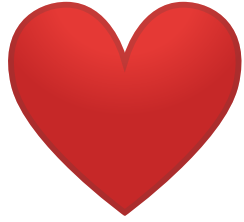
```
import pytest

@pytest.fixture
def half():
    return 21

@pytest.fixture
def answer( half ):
    return half * 2

def test_answer(answer):
    assert answer == 42
```

pytest fixtures

Setup and teardown

```
import pytest

@pytest.fixture
def database():
    db = Database()
    db.connect()
    yield db
    db.rollback()

def test_database(database):
    ...
```





pytest

The pytest-qt plugin

- Install: `pip install pytest-qt`
- Compatible with PyQt4, PyQt5, PySide, PySide2
- Main interaction via the `qtbot` fixture

Basic example

```
@pytest.fixture
def label( qtbot ):
    qlabel = QLabel()
    qlabel.setText("Hello World")
    qtbot.addWidget(qlabel)
    return qlabel

def test_label(label):
    assert label.text() == "Hello World"
```

Exception handling

- Unhandled Python exceptions in virtual C++ methods are problematic!
- What are you going to do?
  - Either return a default-constructed value to C++ and log the exception.
  - Or call `abort()` and kill the process.

Exception handling

- Unhandled Python exceptions in virtual C++ methods are problematic!
- What are you going to do?
  - Either return a default-constructed value to C++ and log the exception.
  - Or call `abort()` and kill the process.

```
============================== FAILURES ==============================
_____________________________ test_exc ______________________________
CALL ERROR: Qt exceptions in virtual methods:
-------------------------------------------
Traceback (most recent call last):
  File "...", line 7, in mouseReleaseEvent
    raise RuntimeError('unexpected error')
RuntimeError: unexpected error
```

Qt Logging Capture

Code

```
def do_something():
    qWarning("this is a WARNING message")

def test_warning():
    do_something()
    assert False
```

Qt Logging Capture

Output

```
============================ FAILURES ============================
_________________________ test_warning _________________________

    def test_warning():
        do_something()
>       assert False
E       assert False

test.py:8: AssertionError
----------------------- Captured Qt messages -----------------------
.../test_warning.py:4:

   QtWarningMsg: this is a WARNING message

========================= 1 failed in 0.04s =========================
```

Qt Logging Capture

Checking messages

```
def test_warning(qtlog):
    do_something()

    messages = [(m.type, m.message.strip()) for m in qtlog.records]
    assert messages == [(QtWarningMsg, "this is a WARNING message")]
```

QTest integration

```
def test_click(qtbot, window):
    qtbot.mouseClick(window.button_ok, Qt.LeftButton)
```

Waiting for windows

```
def test_start(qtbot, app):
    with qtbot.waitExposed(app.splash):
        app.start()
```

Waiting for signals

Basic example

```
def test_computation(qtbot, worker):
    with qtbot.waitSignal(worker.finished, timeout=1000):
        worker.start()
```

Waiting for signals

Basic example

```
def test_computation(qtbot, worker):
    with qtbot.waitSignal(worker.finished, timeout=1000):
        worker.start()
```

```
pytestqt.exceptions.TimeoutError:
Signal finished(QString) not emitted after 1000 ms
```

Waiting for signals

Checking results

```
def test_computation(qtbot, worker):
    with qtbot.waitSignal(worker.finished) as blocker:
        worker.start()

    assert blocker.args == ["result"]
```

Waiting for signals

Multiple signals

```
def test_computation(qtbot, worker):
    with qtbot.waitSignals([worker.success, worker.finished]):
        worker.start()
```

Waiting for signals

Making sure signals are *not* emitted

```
def test_computation(qtbot, worker):
    with qtbot.assertNotEmitted(worker.error):
        worker.start()
```

Waiting for conditions

```
def test_validate(qtbot, window):
    # Status label gets updated after focusing
    window.edit.setFocus()

    def check_label():
        assert window.status.text() == "Please input a number"

    qtbot.waitUntil(check_label)
```

Waiting for callbacks

```
def test_js(qtbot):
    page = QWebEnginePage()

    with qtbot.waitCallback() as callback:
        page.runJavaScript("1 + 1", callback)

    assert callback.args == [2]
```

Model tester

Code

```
class BrokenTypeModel(QAbstractListModel):

    ...

    def data(self, index=QModelIndex(), role=Qt.DisplayRole):
        return object()  # will fail the type check for any role


def test_model(qtmodeltester):
    model = BrokenTypeModel()
    qtmodeltester.check(model)
```

Model tester

Output

```
============================== FAILURES ==============================
_____________________________ test_model _____________________________
test_model.py:14: Qt modeltester errors
------------------------ Captured Qt messages ------------------------
QtWarningMsg: FAIL! variant.canConvert<QString>() () returned FALSE
  (qabstractitemmodeltester.cpp:589)
------------------------ Captured stdout call ------------------------
modeltest: Using Qt C++ tester
======================== 1 failed in 0.05s =========================
```

Other plugins

- pytest-xvfb: Run a virtual X server
- pytest-bdd: Gherkin-Language for tests (“Given ... When ... Then ...”)
- pytest-cov: Coverage recording
- hypothesis: Property-based testing
- ... >700 more: `http://plugincompat.herokuapp.com/`

Contact and resources

Florian Bruhin
florian@bruhin.software
https://bruhin.software
Twitter: @the-compiler

Are you interested in customized trainings, development or consulting? Let's talk!

Slides: https://bruhin.software/talks/qtws19.pdf

https://www.pytest.org/
https://pytest-qt.rtfd.org/