TESTING MULTI-OS PYTHON SOFTWARE

TESTING MULTI-OS PYTHON SOFTWARE

INTRODUCTION

- Introduction
- Unit tests in python
- Automating tests
- General tips and techniques

ABOUT ME

- Developer job at open source software company uib GmbH, see https://uib.de
- Part of backend team at opsi Project, see https://opsi.org
- I love to automate things and make workflows simple, reproducible and robust

WHY SOFTWARE TESTING

- To find bugs (obviously)
- To not break things when refactoring code
- To be robust against malicious or stupid user input

WHY SOFTWARE TESTING

- To find bugs (obviously)
- To not break things when refactoring code
- To be robust against malicious or stupid user input

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."
- Rick Cook

TYPES OF TESTING

- Unit testing: testing single components and core functionalities
- Integration testing: testing interfaces and collaboration of different components
- End-to-end testing: testing complete workflow over all involved components

TYPES OF TESTING

- Unit testing: testing single components and core functionalities
- Integration testing: testing interfaces and collaboration of different components
- End-to-end testing: testing complete workflow over all involved components

This talk focuses on unit testing python applications through python frameworks.
Most concepts may be generalized.

UNIT TESTS IN PYTHON

PYTEST CONCEPT

- Command line tool to collect test scenarios from directory - see https://docs.pytest.org
- Execute tests one by one and evaluate results
- Human-readable summary in output (info of failed tests, percentage of test coverage per file)
- Generates line-by-line status file for tools

PYTEST IS USEFUL...

- For conveniently testing functionality of a whole project or part of it
- To parameterize tests (multiple inputs for the same function)
- When working with different environments (markers control which tests run where and how)

PYTEST EXAMPLE CODE

Given a function you want to test

```
ROT13 = str.maketrans(
  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz",
  "NOPQRSTUVWXYZABCDEFGHIJKLMnopqrstuvwxyzabcdefghijklm"
)

def obfuscate(data: str) -> str:
  return data.translate(ROT13)
```

(It obfuscates a string by shifting each letter by 13 positions)

PYTEST EXAMPLE TEST 1

```python
from testproject import obfuscate

def test_example() -> None:
  assert obfuscate("Hello World") == "Uryyb Jbeyq"
  assert obfuscate(obfuscate("teststring")) == "teststring"
```

PYTEST EXAMPLE TEST 1

```python
from testproject import obfuscate

def test_example() -> None:
  assert obfuscate("Hello World") == "Uryyb Jbeyq"
  assert obfuscate(obfuscate("teststring")) == "teststring"
```

```
========================================= test session starts ==========================================
platform linux -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: /home/doerrer/Documents/uib/testproject
plugins: hypothesis-6.81.1
collected 1 item

tests/test_pytest.py .                                                                          [100%]

========================================== 1 passed in 0.01s ===========================================
```

PYTEST EXAMPLE TEST 2

```python
from testproject import obfuscate

import pytest

@pytest.mark.parametrize("data, result", (
  ("Hello World", "Uryyb Jbeyq"),
  ("The quick brown fox jumps over the lazy dog", "Gur dhvpx oebja sbk whzcf bire gur ynml qbt"),
  ("See you later!", "Frr lbh yngre!"),
))
def test_example2(data, result) -> None:
  assert obfuscate(data) == result
```

PYTEST EXAMPLE TEST 2

```python
from testproject import obfuscate

import pytest

@pytest.mark.parametrize("data, result", (
  ("Hello World", "Uryyb Jbeyq"),
  ("The quick brown fox jumps over the lazy dog", "Gur dhvpx oebja sbk whzcf bire gur ynml qbt"),
  ("See you later!", "Frr lbh yngre!"),
))
def test_example2(data, result) -> None:
  assert obfuscate(data) == result
```

```
========================================= test session starts ==========================================
platform linux -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: /home/doerrer/Documents/uib/testproject
plugins: hypothesis-6.81.1
collected 3 items

tests/test_pytest.py ...                                                                       [100%]

========================================== 3 passed in 0.02s ===========================================
```

PYTEST EXAMPLE TEST 3

```python
from testproject import obfuscate

import pytest

@pytest.mark.parametrize("data", (
  "Let's meet at the park.",
  "My number is 0176123456789"
))
def test_example3(data) -> None:
  obfuscated = obfuscate(data)
  for word in data.split(" "):
    assert word not in obfuscated
```

PYTEST EXAMPLE TEST 3

```
========================================= test session starts ==========================================
platform linux -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: /home/doerrer/Documents/uib/testproject
plugins: hypothesis-6.81.1
collected 2 items

tests/test_pytest.py .F                                                                    [100%]

=============================================== FAILURES ===============================================
_____________________________ test_example3[My number is 0176123456789] _____________________________

data = 'My number is 0176123456789'

    @pytest.mark.parametrize("data", (
        "Let's meet at the park.",
        "My number is 0176123456789"
    ))
    def test_example3(data) -> None:
        obfuscated = obfuscate(data)
        for word in data.split(" "):
>               assert word not in obfuscated
E     AssertionError: assert '0176123456789' not in 'Zl ahzore vf 0176123456789'
E       '0176123456789' is contained here:
E         Zl ahzore vf 0176123456789

tests/test_pytest.py:24: AssertionError
======================================= short test summary info ========================================
FAILED tests/test_pytest.py::test_example3[My number is 0176123456789] - AssertionError: assert '0176123
456789' not in 'Zl ahzore vf 0176123456789'
====================================== 1 failed, 1 passed in 0.02s =====================================
```

HYPOTHESIS CONCEPT

A software tester walks into a bar.
Crawls into a bar.
Dances into a bar.
Flies into a bar.
And orders:
a beer.
99999999 beers.
0 beers.
a lizard in a beer glass.
-1 beer.
"qwertyuiop" beers.

HYPOTHESIS CONCEPT

A software tester walks into a bar.
Crawls into a bar.
Dances into a bar.
Flies into a bar.
And orders:
a beer.
99999999 beers.
0 beers.
a lizard in a beer glass.
-1 beer.
"qwertyuiop" beers.

A real customer walks into the bar and asks where the bathroom is.
The bar goes up in flames.

HYPOTHESIS CONCEPT

- Define what types of input data are allowed for a function
- Check against arbitrary patterns following the scheme

Official documentation at https://hypothesis.readthedocs.io

HYPOTHESIS IS USEFUL...

- For identifying problematic input values
- For checking against extreme instances of some types
  - Integer-like: zero, negative numbers, extremely small and big numbers, …
  - String-like: empty strings, long strings, special characters, …
  - Path-like: relative and absolute paths, spaces, backslash and other atrocities in file names…
  - …

HYPOTHESIS EXAMPLE TEST 1

```
from hypothesis import given, strategies

from testproject import obfuscate

@given(strategies.characters())
def test_obfuscate_hyp(chars):
  assert obfuscate(obfuscate(chars)) == chars
  assert obfuscate(chars) != chars
```

HYPOTHESIS EXAMPLE TEST 1

```
============================= test session starts ==============================
platform linux -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: /home/doerrer/Documents/uib/testproject
plugins: hypothesis-6.81.1
collected 1 item

tests/test_hypothesis.py F                                               [100%]

=================================== FAILURES ===================================
_____________________________ test_obfuscate_hyp ______________________________

    @given(strategies.characters())
>   def test_obfuscate_hyp(chars):

tests/test_hypothesis.py:14:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

chars = '0'

    @given(strategies.characters())
    def test_obfuscate_hyp(chars):
        assert obfuscate(obfuscate(chars)) == chars
>       assert obfuscate(chars) != chars
E    AssertionError: assert '0' != '0'
E     +  where '0' = obfuscate('0')
E    Falsifying example: test_obfuscate_hyp(
E        chars='0',
E    )

tests/test_hypothesis.py:16: AssertionError
=========================== short test summary info ============================
FAILED tests/test_hypothesis.py::test_obfuscate_hyp - AssertionError: assert '0' != '0'
============================== 1 failed in 0.15s ===============================
```

HYPOTHESIS EXAMPLE TEST 2

An example from official site https://hypothesis.readthedocs.io

```
from hypothesis import given, strategies

@given(strategies.integers(), strategies.integers())
def test_ints_are_commutative(x, y):
  assert x + y == y + x
```

HYPOTHESIS EXAMPLE TEST 2

An example from official site https://hypothesis.readthedocs.io

```python
from hypothesis import given, strategies

@given(strategies.integers(), strategies.integers())
def test_ints_are_commutative(x, y):
  assert x + y == y + x
```

```
========================================= test session starts ==========================================
platform linux -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: /home/doerrer/Documents/uib/testproject
plugins: hypothesis-6.81.1
collected 1 item

tests/test_hypothesis.py .                                                                      [100%]

========================================== 1 passed in 0.17s ===========================================
```

HYPOTHESIS EXAMPLE TEST 3

```
from hypothesis import given, strategies

@given(strategies.floats(), strategies.floats())
def test_floats_are_commutative(x, y):
  assert x + y == y + x
```

HYPOTHESIS EXAMPLE TEST 3

```
========================================= test session starts ==========================================
platform linux -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: /home/doerrer/Documents/uib/testproject
plugins: hypothesis-6.81.1
collected 1 item

tests/test_hypothesis.py F                                                                       [100%]

=============================================== FAILURES ===============================================
___________________________________ test_floats_are_commutative ____________________________________

    @given(strategies.floats(), strategies.floats())
>   def test_floats_are_commutative(x, y):

tests/test_hypothesis.py:10:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

x = inf, y = -inf

    @given(strategies.floats(), strategies.floats())
    def test_floats_are_commutative(x, y):
>       assert x + y == y + x
E    assert (inf + -inf) == (-inf + inf)
E    Falsifying example: test_floats_are_commutative(
E        x=inf,
E        y=-inf,
E    )

tests/test_hypothesis.py:11: AssertionError
======================================= short test summary info ========================================
FAILED tests/test_hypothesis.py::test_floats_are_commutative - assert (inf + -inf) == (-inf + inf)
========================================== 1 failed in 0.21s ===========================================
```

AUTOMATING TESTS

GITLAB-CI

- Git is a powerful tool!
- Great open source software development platform gitlab - see https://about.gitlab.com
- Feature gitlab-ci performs pipeline of defined jobs at each "git push"
- Job consists of actions - executed consicutively as long as exit code is 0 (OK)
- Talk of Erol Ülükmen at FrOSCon 2022: https://media.ccc.de/v/froscon2022-2795-ci_first

GITLAB-CI

- Pytest has exit code 1 if one test fails.
- So gitlab-ci job running pytest automatically fails on a test fail.
- Further steps are not triggered (shipping to internal end-to-end testing environment or even production)
- Author of commit is notified

MULTI-PLATFORM PYTHON APPS

- Easy to build multi-platform apps with python
- Development on one platform, then shipped to many
- Fundamental differences between platforms
  - Config files vs. registry
  - Handling of network shares and other resources
  - Users, file permissions, …
  - …

TESTING ON MULTIPLE PLATFORMS

- Some functionality can be mocked - e.g. simulating a web server
- Certainty only if testing behavior on (all!) target platforms
- Linux-like systems available in docker
- Windows, MacOS systems as VMs

COLLECTING TEST RESULTS

- One gitlab-ci job per platform (running in container/VM)
- Each pytest creates coverage file (xml-format) and stores it as pipeline artifact
- Final job collects artifacts and combines coverage files (by using "coverage" tool - see https://coverage.readthedocs.io)
- Result is human-readable summary and machine-readable coverage file

COLLECTING TEST RESULTS

Example summary for "opsi-cli"

Name	Stmts	Miss	Cover	Miss	Cover (Windows only)
opsicli/__init__.py	11	0	100%	0	100%
opsicli/__main__.py	105	9	91%	9	91%
opsicli/cache.py	61	4	93%	7	89%
opsicli/config.py	310	35	89%	41	87%
opsicli/io.py	272	44	84%	45	83%
opsicli/messagebus.py	136	35	74%	106	22%
opsicli/opsiservice.py	75	29	61%	58	23%
opsicli/plugin.py	181	10	94%	10	94%
opsicli/types.py	110	10	91%	12	89%
opsicli/utils.py	62	21	66%	23	63%
plugins/client-action/python/__init__.py	37	0	100%	4	89%
plugins/client-action/python/client_action_worker.py	32	3	91%	25	22%
plugins/client-action/python/set_action_request_worker.py	125	19	85%	111	11%
plugins/config/python/__init__.py	115	27	77%	27	77%
plugins/jsonrpc/python/__init__.py	76	22	71%	47	38%
plugins/manage-repo/python/__init__.py	106	4	96%	4	96%
plugins/plugin/python/__init__.py	164	27	84%	27	84%
plugins/self/python/__init__.py	152	36	76%	63	59%
plugins/support/python/__init__.py	42	12	71%	16	62%
plugins/support/python/worker.py	10	0	100%	7	30%
plugins/terminal/python/__init__.py	22	3	86%	3	86%
TOTAL	2260	340	85%	646	71%

GENERAL TIPS AND TECHNIQUES

TEST DRIVEN DEVELOPMENT

Reverse classical testing workflow:

1. Think about tasks and how to wrap them in classes and functions
2. Write tests for each class and function
3. Implement the actual classes and functions so that tests do not fail

USE GIT BRANCHES

- Main branch always ready for release
- Individual feature- and fix- branches, merge request to main once complete and tested
- Merge request reviewed and accepted/denied by another developer (principle of 4 eyes)

USE VIRTUAL ENVIRONMENTS

- Virtual python environments are isolated from host system
- Handle dependency management tailored to specific application
- If you break it - you dont break your host system
- Some frameworks assisting with that:
  - venv: very basic
  - conda environments: convenient
  - poetry: comfortable and flexible

USEFUL IDE FEATURES

- Dev-container: isolated environment (tests do not affect host)
- Go-to-definition (e.g. CTRL + click): shows origin of selected component
- Coverage gutter: pytest coverage line by line
- Git marker: added, changed, deleted lines visualized

```
111  class File(type(Path())):  # type: ignore[misc] # pylint: disable=to
112      click_type = click.Path(dir_okay=False)
113
114      def __new__(cls: Type[File], *args: Any, **kwargs: Any) -> File:
115          path = super().__new__(cls, *args, **kwargs)
116          if str(path) != "-":
117              path = path.expanduser().absolute()
118              if path.exists() and not path.is_file():
119                  raise ValueError(f"Not a file: {path!r}")
120          return path
```

```
396  [.column.is-half]
397
398  * dev-container: isolated environment (tests do not affect host)
399  * Go-to-definition (e.g. CTRL + click): shows origin of selected component
400  * coverage gutter: pytest coverage line by line
401  * git marker: added, changed, deleted lines visualized
402  [.column.is-half]
403
404  --
405  image::images/coverage_gutter.png[pytest cover,width=920,align=right]
406  image::images/git_marker.png[git markers,width=920,align=right]
407  --
```

USE LINTERS

- Linters highlight potential problems of all sorts
- Some may even automatically fix small issues
  - pylint and ruff: check general python conventions
  - flake8: checks code style
  - mypy: checks data types

USE TYPE HINTS

- Python is dynamically typed - variables can point to objects of different types.
- Type hints help to keep types in mind

```
def get_length(value):
  return len(value)


print(get_length("teststring"))  # ok
print(get_length("1234"))  # ok
print(get_length(1234))  # exception at runtime
```

USE TYPE HINTS

- Python is dynamically typed - variables can point to objects of different types.
- Type hints help to keep types in mind

```python
def get_length(value):
  return len(value)


print(get_length("teststring"))  # ok
print(get_length("1234"))  # ok
print(get_length(1234))  # exception at runtime
```

```python
def get_length(value: str) -> int:
  return len(value)


print(get_length("teststring"))  # ok
print(get_length("1234"))  # ok
print(get_length(1234))  # IDE will probably warn you
```

DO NOT REINVENT THE WHEEL

- Use recent python release for newest python standard library features
- Python has a large community with thousands of cool libraries
- Catalogue at https://pypi.org
- If you spot a problem: open an issue! Don’t just workaround

THANK YOU FOR YOUR ATTENTION!

Questions?

ADDITIONAL MATERIAL: SETTING UP A NEW PROJECT WITH POETRY

- poetry is a python dependency manager - see https://python-poetry.org/
- `poetry new <name>` creates
  - pyproject.toml with general project info and dependency list
  - <name> directory with empty `__init__.py` file (python package)
  - `tests directory with empty `__init__.py` file (package for the tests)
- `poetry install` creates virtual environment

ADDITIONAL MATERIAL: JENKINS

- Open source automation system https://jenkins.io
- We use it for end-to-end tests:
  - Triggered by gitlab-ci or manual action
  - Jobs run on VMs, packages collected from development repository
  - Script controls how components interact