PROPERTY-BASED TESTING WITH PYTEST & HYPOTHESIS

```
import pytest
import hypothesis

...
```

https://hypothesis.readthedocs.io

DEMO

01_basics.py

DEMO

01_basics.py

- pytest makes testing easy
- hypothesis integrates nicely with pytest
- We write tests which test properties of our code and don't compare results with expected results
- hypothesis will try to shrink input that makes your tests fail
- hypothesis keeps previous inputs around in a local database so the error is reproducible

PROPERTY BASED TESTING VS FUZZING

hypothesis.works/.../what-is-property-based-testing

*" Fuzzing is **feeding** a piece of code (function, program, etc.) **data from a large corpus**, possibly dynamically generated, possibly dependent on the results of execution on previous data, in order **to see whether it fails**. "*

hypothesis.works/.../what-is-property-based-testing

*" Property based testing is the **construction of tests** such that, when these tests are fuzzed, failures in the test reveal problems with the system under test that **could not have been revealed by direct fuzzing** of that system. "*

hypothesis.works/.../what-is-property-based-testing

" [...] property-based testing is what you do, not what the computer does. The part that the computer does is “just fuzzing”. "

" A property-based testing library:

- *A fuzzer.*
- *A library of tools for making it easy to construct property-based tests using that fuzzer.*

"

DEMO

02_gen_basic.py

DEMO

02_gen_basic.py

- @given() is the main entry point
- It provides various strategies for generating values of common types
- It provides functions for modifying/mixing strategies and generating collections
- All strategies have various options to adjust/limit/extend the range of values

DEMO

03_gen_more.py

DEMO

04_gen_composite.py

FUZZING

- Lots of stuff happening recently because of Google's OSS-FUZZ project
- OSS-FUZZ Process Overview
- Integrated projects: https://github.com/google/oss-fuzz/tree/master/projects
- Yesterday they added FuzzBench: report
- Uses libFuzzer, hongfuzz, .. and AFL

AFL

wikipedia.org/American_fuzzy_lop_(fuzzer)
http://lcamtuf.coredump.cx/afl/

- Instrumentation-guided genetic fuzzer -> Mutates input and tries to change the code flow in the program to reach full coverage
- Via compile time instrumentation: `CC=afl-gcc ./configure; make`
- .. or QEMU for binary only programs
- .. or custom through shared memory -> Python

PYTHON-AFL

https://github.com/jwilk/python-afl

Shared memory:
https://robertheaton.com/2019/07/08/how-to-write-an-afl-wrapper-for-any-language/

Instrumentation via `os.settrace()`

DEMO: PYTHON-AFL + SETTRACE()

PYTHON-AFL

- Fast because written in C, instrumentation through settrace() in Cython
- Good for a file parser etc.
- Depends on AFL (`apt install afl++`) and Cython
- Compared to hypothesis no pytest integration
- (but I haven't used it for anything real yet..)

REAL WORLD HYPOTHESIS EXAMPLES

- pycairo / pycairo
- mutagen
- pypy / pypy
- attrs / attrs
- hyper-h2 / hyper-h2
- brotlipy

GOOD - IMHO

- Also usable for TDD
- Good integrated with pytest, easy to integrate into an existing test suite, few dependencies
- Easy to generate complex Python types
- Custom strategies can be created and shared

BAD - IMHO

- Tests are harder to write
- Tests are slower
- Mainly for preventing errors with edge cases, doesn't mean your implementation is correct
- Unit tests are also documentation, hypothesis tests are harder to read
- No coverage based fuzzing like other fuzzers

END

https://docs.pytest.org
https://hypothesis.readthedocs.io
https://github.com/jwilk/python-afl