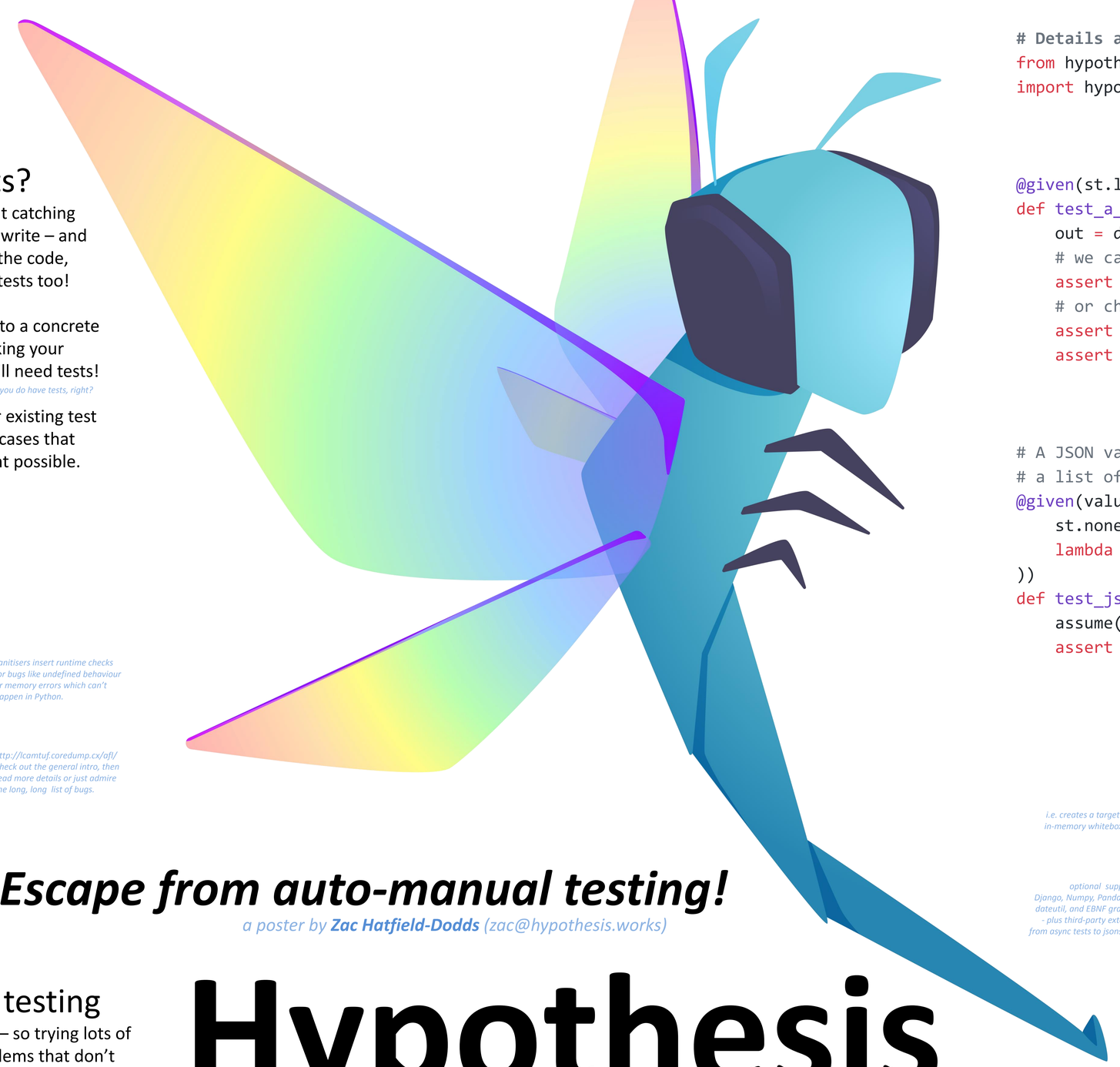

Hypothesis

Escape from auto-manual testing!

*a poster by **Zac Hatfield-Dodds** (zac@hypothesis.works)*

Why generate tests?

Hand-written unit tests are good at catching regressions, but can be tedious to write – and if you forgot an edge case writing the code, you probably forgot it writing the tests too!

there are some cool tricks here, like checking that a trace from a test or prod logs is allowed by your formal model.

Formal methods are hard to scale to a concrete implementation – and while checking your design is a big advantage, you'll still need tests!

…you do have tests, right?

testing is much more fun when you're not sure what will happen, and might even learn something!

Generating tests fits well into your existing test suite and can often produce edge cases that you hadn't considered – or thought possible.

Fuzzing

is the art and science of passing arbitrary inputs to programs, to see if they crash or hang. Using sanitisers and assertions to crash on logic errors is even more useful!

sanitisers insert runtime checks for bugs like undefined behaviour or memory errors which can't happen in Python.

Fuzzers like *American Fuzzy Lop* are **very** effective –they find lots of vulnerabilities in important and widely used software!

http://lcamtuf.coredump.cx/afl/ Check out the general intro, then read more details or just admire the long, long list of bugs.

Property-based testing

there are lots of definitions – we discuss this one at https://hypothesis.works/articles/what-is-property-based-testing/

is testing using fuzzable tests – so trying lots of different inputs can find problems that don't necessarily crash.

- does this multi-core algorithm always get the same answer as a slow and simple version?
- given auto-formatted Python code, does running the formatter again change it?

https://danluu.com/testing/ might change the way you think about testing code or hardware

PBT was invented by QuickCheck, in Haskell, but recent tools are equally inspired by fuzzing.

this is great for web APIs, the SymPy algebra system, distributed schedulers, etc. Shrinking is indispensable!

`hypothesis.stateful` can generate *actions* as easily as inputs – exploring test programs that no human would every try to write by hand.

```python
# Details at https://hypothesis.readthedocs.io
from hypothesis import assume, given
import hypothesis.strategies as st


@given(st.lists(st.integers(), min_size=2))
def test_a_sort_function(ls):
    out = dubious_sort(ls)
    # we can compare to a trusted implementation,
    assert out == sorted(ls)
    # or check the properties we need directly.
    assert Counter(out) == Counter(ls)
    assert all(a <= b for a, b in zip(out, out[1:]))


# A JSON value may be none, a bool, a number, a string;
# a list of JSON values, or a string-to-JSON-value dict.
@given(value = st.recursive(
    st.none() | st.booleans() | st.floats() | st.text(),
    lambda s: st.lists(s) | st.dictionaries(st.text(), s)
))
def test_json_values_roundtrip(value):
    assume(value == value)
    assert value == json.loads(json.dumps(value))
```

actual test output below ↓

Unpacking the code sample

i.e. creates a target for our in-memory whitebox fuzzer

`@given` wraps your test function, and runs it many times with arguments drawn from the strategies.

optional support for Django, Numpy, Pandas, pytz, dateutil, and EBNF grammars - plus third-party extensions from async tests to jsonschema

Strategies can be defined directly, combined, and composed with arbitrary code – or inferred from regular expressions, arbitrary type hints, and more. You can even draw more values within the test.

When a test case raises an error, `@given` searches for a minimal input that causes the same error. This makes debugging as easy as possible!

Who else uses Hypothesis?
4% of all Python users, CPython, PyPy, >200 packages, >1700 repos; AWS, Google, Stripe, Ethereum, schoolteachers, … and you?

Do my functions have to be pure?
Nope! So long as it does the same thing each time it's passed the same arguments and cleans up after, your test can do anything.

Isn't random testing flaky?
Every failing example is cached and retried , and we print the random seed too. You can even commit the cache!

How fast is Hypothesis?
The overhead is small, but running your test function 100 times (by default) takes a while.

What test runners can I use?
Anything that calls functions - tested against unittest, nose, pytest, etc. Async tests work with a shim.

```
$ pytest example.py
======================= FAILURES ========================
_________________ test_json_roundtrips _________________
Traceback (most recent call last):
  ...
AssertionError: assert [[nan]] == [[nan]]
  At index 0 diff: [nan] != [nan]
---------------------- Hypothesis ----------------------
Falsifying example: test_json_roundtrips(value=[[nan]])
================= 1 failed, 1 passed ==================
```

Hypothesis reports a minimal example of each bug it finds!

value compares to itself by list identity, but to the deserialised list by element values!