Don't write tests, Generate them!

Introduction

Property-based testing, anyone?

A typical test-suite

```
def test_strip_whitespace_with_no_argument():
    assert strip('  foo ') == 'foo'

def test_should_strip_whitespace_with_argument():
    assert strip('  foo ', ' ') == 'foo'

def test_should_strip_non_whitespace():
    assert strip('foo', 'fo') == ''

...
```

Example based tests

Given
Setup some **example** data

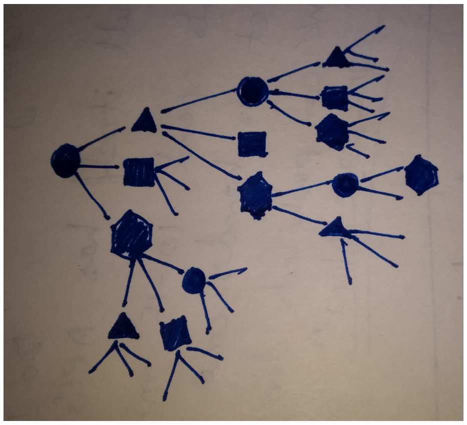
When
Perform actions

Then
`assert output == expected`

Problems?

- Combinatorial explosion
- Biases carry-over to tests
- Tedious

State in a website



Enter Generative testing

(Property-based testing)

Property-based test, the hard way

```
def test_strip_random():
    for _ in range(200):
        s = random_string()
        strip_chars = random_string()
        S = strip(s, strip_chars)
        assert is_stripped(S, s, strip_chars)

def is_stripped(S, s, strip_chars):
    assert len(S) <= len(s)
    if len(S) > 0:
        assert S[0] not in set(strip_chars)
        assert S[-1] not in set(strip_chars)
    return True

random_string = [
    random.choice(string.ascii_letters)
    for _ in range(10)
]
```

Property based tests

Given
For **random** data matching a spec

When
Perform actions

Then
`assert property(output)`

Hypothesis - Property based testing for Python

Hypothesized test

```
from hypothesis import given, strategies as st

@given(st.text(), st.text())
def test_strip_hypothesis(s, strip_chars):
    S = strip(s, strip_chars)
    assert is_stripped(S, s, strip_chars)
# Ran 1 test in 0.159s
```

Failing output

```
strip = lambda x, y: x.lstrip(y)
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

output = '01', input_ = '01', strip_chars = '1'

    def is_stripped(output, input_, strip_chars):
        assert len(output) <= len(input_)
        if len(output) > 0:
            assert output[0] not in set(strip_chars)
>           assert output[-1] not in set(strip_chars)
E           assert '1' not in {'1'}
E            +  where {'1'} = set('1')

test_code.py:113: AssertionError
----------------------------------------------------- Hypothesis ------------------------------------------------------
Falsifying example: test_strip(s='01', strip_chars='1')
```

Shrinking

- Random data has lots of noise
- Try to find the "simplest" failing case

To learn more, see Designing a better simplifier

Data generation

Generators for built-ins

```
from hypothesis import strategies as st

def sample(strategy, n=3):
    return [strategy.example() for _ in range(n)]

print(sample(st.integers()))
print(sample(st.floats()))
print(sample(st.complex_numbers()))
print(sample(st.text(max_size=3)))
print(sample(st.lists(st.integers())))
```

```
[-7435755662106, -49, -1295624]
[-9.266256382731017e+17, -0.19780830243100944, -2.4010523231296193e+61]
[(-0.99999-0.99999j), (-2.220446049250313e-16+nanj), (0.0035546080693361136-1.923176004582495e-275j)]
['', '\U000ded7f9', '']
[[52647858669059, -31758544979, 71365626], [0], []]
```

Extra generators

- Django models
- Numpy arrays
- Dates & times
- Faker generators

Composable strategies

```
from hypothesis import strategies as st

st.recursive?
st.one_of?
st.builds?
st.streaming?

.map, .filter, .flatmap
```

Composing strategies - Example

```
rows = [('John', 'Adams', 90), (...), (...)]
headers = ['first_name', 'last_name', 'gpa']
print(tablib.Dataset(*rows, headers=headers))
```

```
first_name|last_name |gpa
----------|----------|---
John      |Adams     |90
George    |Washington|67
Thomas    |Jefferson |50
```

Generate Rows & Header

```
from hypothesis import strategies as st; import string

n = 3
alphabet = string.ascii_letters
generate_row = st.tuples(
    st.text(alphabet, min_size=1),
    st.text(alphabet, min_size=1),
    st.integers(min_value=0, max_value=100)
)
generate_table = st.lists(generate_row, min_size=3, max_size=3)
generate_headers = st.lists(
    st.text(alphabet, min_size=1),
    unique=True,
    min_size=n,
    max_size=n
)
```

Putting it together

```
def create_dataset(rows, headers):
    return tablib.Dataset(*rows, headers=headers)

def generate_dataset():
    return st.builds(create_dataset, generate_data, headers=generate_headers)

print(generate_dataset().example())
```

```
znefubbdv     |wpclcf|ouc
--------------|------|---
aecpjxzwfqosmu|krlmfh|55
htq           |jid   |87
lwbfboxyifre  |oqdha |83
```

Simple `tablib` test

```
def test_add_column():
    rows = [['kenneth'], ['bessie']]
    data = tablib.Dataset(*rows, headers=['fname'])
    new_col = ['reitz', 'monke']
    data.append_col(new_col, header='lname')

    assert data[0] == ('kenneth', 'reitz'))
    assert data.width == 2
```

to a property based test

```
@given(data=generate_dataset(),
       new_col=st.lists(st.text(min_size=1), min_size=3, max_size=3),
       header=st.text(min_size=3))
def test_hyp_add_column(data, new_col, header):
    first_row = data[0]
    data.append_col(new_col, header=header)

    assert data[0] == first_row + (new_col[0],)
    assert data.width == 4
```

Test transpose

```
@given(generate_dataset())
def test_transpose(self, data):
    data_ = data.transpose()

    self.assertEqual(data.width, data_.height+1)
    self.assertEqual(data.height, data_.width-1)
```

Round trip transpose

```
@given(generate_dataset())
def test_two_transposes(self, data):
    data_ = data.transpose().transpose()

    self.assertEqual(data.width, data_.width)
    self.assertEqual(data.height, data_.height)
```

```
    self.assertEqual(data.width, data_.height)
E   AssertionError: 3 != 2
------------------------------------------------ Captured stdout call -------------------------------------------------
Falsifying example:
a|b|c
-|-|-
a|a|0
a|a|0
a|a|0
```

Round trip to json

```
@given(generate_dataset())
def test_json_export_import_works(data):
    json_ = data.json
    data_ = tablib.import_set(json_)

    self.assertEqual(data.width, data_.width)
    self.assertEqual(data.height, data_.height)
    self.assertEqual(data[0], data_[0]))
```

```
    self.assertEqual(data[0], data_[0])
E   AssertionError: Tuples differ: ('a', 'a', 0) != ('a', 0, 'a')
```

Verification

`strip` tests from before

Sorting actually returns a sorted list

Computing the mean

```
from hypothesis import given, strategies as st

@given(st.lists(st.floats(allow_nan=False, allow_infinity=False)), min_size=1)
def test_mean_is_within_reasonable_bounds(ls):
    assert min(ls) <= mean(ls) <= max(ls)
```

Going by definition …

```
def mean(xs):
    return sum(xs) / len(xs)
```

```
ls = [8.988465674311579e+307, 8.98846567431158e+307]

    @given(st.lists(st.floats(allow_nan=False, allow_infinity=False), min_size=1))
    def test_mean_is_within_reasonable_bounds(ls):
>       assert min(ls) <= mean(ls) <= max(ls)
E       assert inf <= 8.98846567431158e+307
E        +  where inf = mean([8.988465674311579e+307, 8.98846567431158e+307])
E        +  and   8.98846567431158e+307 = max([8.988465674311579e+307, 8.98846567431158e+307])
```

Avoiding overflow

```
def mean(xs):
    n = len(xs)
    return sum(x / n  for x in xs)
```

```
ls = [1.390671161567e-309, 1.390671161567e-309, 1.390671161567e-309]

    @given(st.lists(st.floats(allow_nan=False, allow_infinity=False), min_size=1))
    def test_mean_is_within_reasonable_bounds(ls):
>       assert min(ls) <= mean(ls) <= max(ls)
E       assert 1.390671161567e-309 <= 1.390671161566996e-309
E        +  where 1.390671161567e-309 = min([1.390671161567e-309, 1.390671161567e-309, 1.390671161567e-309])
E        +  and   1.390671161566996e-309 = mean([1.390671161567e-309, 1.390671161567e-309, 1.390671161567e-309])
```

For instance, `numpy`

```
import numpy as np
def mean(xs):
    return np.array(xs).mean()
```

```
ls = [8.988465674311579e+307, 8.98846567431158e+307]
```

```
    @given(st.lists(st.floats(allow_nan=False, allow_infinity=False), min_size=1))
    def test_mean_is_within_reasonable_bounds(ls):
>       assert min(ls) <= mean(ls) <= max(ls)
E       assert inf <= 8.98846567431158e+307
E        +  where inf = mean([8.988465674311579e+307, 8.98846567431158e+307])
E        +  and   8.98846567431158e+307 = max([8.988465674311579e+307, 8.98846567431158e+307])
```

Read this 30 page paper, to see how to do it right!

Test Oracle

```
from hypothesis import strategies as st, given
from my_lib import my_sort

@given(st.lists(st.integers()))
def test_my_sort(xs):
    assert sorted(xs) == my_sort(xs)
```

More patterns

See talk by Jeremy Thurgood

- Induction
- Transformation
- Invariance
- Idempotence

Keep in mind

- Fast data generation
- Fast assertions
- Simple looking, yet powerful
- Re-use?

Stateful testing

```
def test_website():
    assert login(credentials)
    assert go_to_homepage()
    assert follow_friend()
    assert logout()
```

Pseudocode example

```
class WebSiteStateMachine(RuleBasedStateMachine):
    def __init__(self):
        super(WebSiteStateMachine, self).__init__()

    def login(self):
        """Login using credentials and assert success."""

    @rule()
    def logout(self):
        """Logout and assert it worksn."""

    @rule(user=st.sampled_from(USERS))
    def follow_user(self, user):
        """Assert that following a user works."""

WebSiteTestCase = WebSiteStateMachine.TestCase
```

Problems with Generative Testing?

- Performance
- Debugging CI failures
- Rare branches?

Conclusion

Property based tests

- Concise
- Overcome developer biases
- Assert general things

Hypothesis

- Generate data, given a requirement
- Check that a **property** holds true
- Shrink failed cases to simplest case

Some interesting case studies

- John Hughes: Testing the hard stuff and staying sane
- Ashton Kemerling: Generative Integration Testing
- Sean Grove: Generating and Running 1M tests

Pairing anyone?

Thank you

@punchagan

http://tinyurl.com/pygentest

Author: Puneeth Chaganti
Validate