BULLETPROOF PYTHON

PROPERTY-BASED TESTING WITH HYPOTHESIS

Michael Seifert

ROADMAP

- What is property-based testing?
- Session 1 - Baby steps with Hypothesis
- Generating all the things
- Session 2 - Describing your data
- Approaches for writing property-based tests
- Session 3 - Practice
- Hypothesis in practice

WHAT IS PROPERTY-BASED TESTING?

```python
from typing import List, TypeVar

T = TypeVar("T", int, float)

def max(l: List[T]) -> T:
    current_max = None
    for element in l:
        if current_max is None or element > current_max:
            current_max = element
    return current_max
```

```python
def test_max_returns_maximum_int():
    values = [-3, 5, 1]
    assert max(values) == 5

def test_max_returns_maximum_float():
    values = [-3.0, 5.0, 1.0]
    assert max(values) == 5.0
```

```python
@pytest.mark.parametrize(
    "values, expected_max",
    (
        ([-3, 5, 1], 5),
        ([-3.0, 5.0, 1.0], 5.0),
    )
)
def test_max_returns_max(values, expected_max):
    assert max(values) == expected_max
```

```python
@pytest.mark.parametrize(
    "values",
    (
        [-3, 5, 1],
        [-3.0, 5.0, 1.0],
    )
)
def test_max_returns_max(values):
    assert max(values) == sorted(values)[-1]
```

```python
@given(
    st.one_of(
        st.lists(st.integers()),
        st.lists(st.floats()),
    )
)
def test_max_returns_max(values):
    assert max(values) == sorted(values)[-1]
```

```
@given(
    st.one_of(
        st.lists(st.integers()),
        st.lists(st.floats()),
    )
)
def test_max_returns_max(values):
    assert max(values) == sorted(values)[-1]
```

```
    def test_max_returns_max(values):
>       assert max(values) == sorted(values)[-1]
E       IndexError: list index out of range
```

```python
from typing import List, TypeVar

T = TypeVar("T", int, float)

def max(l: List[T]) -> T:
    current_max = None
    for element in l:
        if current_max is None or element > current_max:
            current_max = element
    return current_max
```

```python
from typing import List, TypeVar

T = TypeVar("T", int, float)

def max(l: List[T]) -> T:
    if not l:
        raise ValueError()
    current_max = None
    for element in l:
        if current_max is None or element > current_max:
            current_max = element
    return current_max
```

```python
from typing import List, TypeVar

T = TypeVar("T", int, float)

def max(l: List[T]) -> T:
    if not l:
        raise ValueError()
    current_max = None
    for element in l:
        if current_max is None or element > current_max:
            current_max = element
    return current_max
```

```python
def test_max_raises_when_input_is_empty():
    with pytest.raises(ValueError):
        max([])
```

```python
@given(
    st.one_of(
        st.lists(st.integers(), min_size=1),
        st.lists(st.floats(), min_size=1),
    )
)
def test_max_returns_max(values):
    assert max(values) == sorted(values)[-1]
```

```python
@given(
    st.one_of(
        st.lists(st.integers(), min_size=1),
        st.lists(st.floats(), min_size=1),
    )
)
def test_max_returns_max(values):
    assert max(values) == sorted(values)[-1]
```

```
    def test_max_returns_max(values):
>       assert max(values) == sorted(values)[-1]
E       assert nan == nan
E        +  where nan = max([nan])
```

```python
@given(
    st.one_of(
        st.lists(st.integers(), min_size=1),
        st.lists(st.floats(allow_nan=False), min_size=1),
    )
)
def test_max_returns_max(values):
    expected_max = sorted(values)[-1]
    assert max(values) == expected_max
```

TAKE AWAYS

TAKE AWAYS

- The people who write the code don't come up with good tests to break it.

TAKE AWAYS

- The people who write the code don't come up with good tests to break it.
- Describe how your data should look like. Let Hypothesis worry about the rest.

HANDS ON

BABY STEPS WITH HYPOTHESIS

Unix/MacOS (Bash)

```
$ git clone https://github.com/seifertm/hypothesis-workshop
$ cd hypothesis-workshop/exercises
$ python -m venv venv
$ . venv/bin/activate
$ pip install -r requirements.txt -c constraints.txt
$ pytest
```

Windows (PowerShell)

```
> git clone https://github.com/seifertm/hypothesis-workshop
> cd hypothesis-workshop\exercises
> python -m venv venv
> venv\Scripts\activate.ps1
> pip install -r requirements.txt -c constraints.txt
> pytest
```

GENERATING ALL THE THINGS

SPECIFIC EXAMPLES

```
@example(1)
@example(0)
@given(st.integers())
def test_my_func(i):
    ...
```

MANIPULATING STRATEGIES

```
@given(
    st.integers().map(str)
)
def test_map(int_as_string):
    ...
```

MANIPULATING STRATEGIES

```
@given(
    st.integers().map(str)
)
def test_map(int_as_string):
    ...
```

```
@given(
    st.integers().filter(lambda i: i % 7 != 0)
)
def test_filter(not_divisible_by_seven):
    ...
```

MULTI VALUE STRATEGIES

```
@given(st.sampled_from(["red", "green", "blue"]))
def test_question(color):
    ...
```

LISTS

```
@given(
    st.lists(
        st.integers(),
        min_size=1,
        max_size=10,
        # unique = True,
    )
)
def test_list_of_integers(ints):
    ...
```

TUPLES

```
@given(
    st.tuples(st.integers(), st.integers())
)
def test_pair(pair):
    ...
```

DICTIONARIES

```
@given(
    st.fixed_dictionaries(
        {
            "key1": st.binary(),
            "key2": st.text(),
        }
    )
)
def test_dict(my_map):
    ...
```

COMBINING STRATEGIES

```
@given(
    st.one_of(
        (st.integers(), st.floats())
    )
)
def test_float_or_int(number):
    ...
```

COMBINING STRATEGIES

```
@given(
    st.one_of(
        (st.integers(), st.floats())
    )
)
def test_float_or_int(number):
    ...
```

```
@given(st.integers() | st.floats())
def test_float_or_int(number):
    ...
```

COMPLEX STRATEGIES

```
@st.composite
def list_with_index(draw):
    int_list = draw(st.lists(st.integers(), min_size=1))
    list_index = draw(
        st.integers(min_value=0, max_value=len(int_list) - 1)
    )
    return int_list, list_index
```

CONSTRUCTING OBJECTS

```
@dataclass
class Point2d:
    x: float
    y: float
```

CONSTRUCTING OBJECTS

```
@dataclass
class Point2d:
    x: float
    y: float
```

```
@given(
    st.builds(Point2D)
)
def test_point(point):
    ...
```

CONSTRUCTING OBJECTS

```
@given(
    st.builds(
        Point2D,
        x=st.floats(allow_nan=False),
        y=st.floats(allow_nan=False)
    )
)
def test_point(point):
    ...
```

SINGLE VALUE STRATEGY

```
@given(st.just(42))
def test_question(answer_to_life):
    ...
```

SINGLE VALUE STRATEGY

```
@given(st.just(42))
def test_question(answer_to_life):
    ...
```

```
@given(
    st.builds(
        Point2D,
        x=st.just(42.0)
    )
)
```

TAKE AWAYS

Rather generate too much than too little

Study the Hypothesis docs

HANDS ON - DESCRIBING YOUR DATA

APPROACHES FOR WRITING PROPERTY-BASED TESTS

FUZZING

```
@given(
    st.lists(
        st.integers() | st.floats() | st.text()
    )
)
def test_sort(a_list):
    custom_sort(a_list)
```

DIFFERENTIAL TESTING

```
@given(
    st.lists(
        st.text()
    )
)
def test_my_custom_sort(l):
    assert custom_sort(l) == sorted(l)
```

ROUNDTRIPS

```
@given(st.binary())
def test_base64(binary):
    encoded = b64encode(binary)
    decoded = b64decode(encoded)
    assert decoded == binary
```

METAMORPHIC TESTS

```
@given(st.integers(min_value=0))
def test_negative_square_equals_square(n):
    assert square(n) == square(-n)
```

METAMORPHIC TESTS

```python
@given(st.integers(min_value=0))
def test_negative_square_equals_square(n):
    assert square(n) == square(-n)
```

```python
@given(st.integers(min_value=0))
def test_square_is_strictly_monotonic(n):
    assert square(n) < square(n + 1)
```

ALGEBRAIC PROPERTIES

```
@given(st.integers())
def test_sign_is_idempotent(n):
    assert sign(n) == sign(sign(n))
```

HANDS ON - PRACTICE

HYPOTHESIS IN PRACTICE

DEBUGGING

```
@given(st.lists(st.integers()))
def test_printing_failures(ints):
    note("Sum of list: {sum(ints)}")
    ...
```

REPRODUCING FAILURES

```
@settings(print_blob=True)
@given(st.integers())
def test_that_fails(n):
    ...
```

REPRODUCING FAILURES

```
@settings(print_blob=True)
@given(st.integers())
def test_that_fails(n):
    ...
```

```
@reproduce_failure(b"AAAAABBBB1111===")
@given(st.integers())
def test_that_fails(n):
    ...
```

PROFILES

```
settings.register_profile("dev")
settings.register_profile("ci", deadline=None)
settings.load_profile(
    os.getenv("HYPOTHESIS_PROFILE", "dev")
)
```

STATISTICS

```
$ pytest --hypothesis-show-statistics
```

```
test_max_returns_max:

- during reuse phase (0.00 seconds):
    - Typical runtimes: < 1ms, ~ 55% in data generation
    - 2 passing examples, 0 failing examples, 0 invalid exampl

- during generate phase (0.17 seconds):
    - Typical runtimes: 0-1 ms, ~ 74% in data generation
    - 98 passing examples, 0 failing examples, 10 invalid exam

- Stopped because settings.max_examples=100
```

THANK YOU!

www.seifertm.de

LinkedIn, GitHub: @seifertm

Twitter: @seifertm0