Testing

CS 691 / SWE 699

Fall 2025

GEORGE MASON UNIVERSITY

Logistics

- Lecture 11 reading questions due today at 4:30pm
- Lecture 11 activity (in class today), due by 11/13 at 4:30pm
- Project presentations in 3 weeks

Today

- Discussion: Experiences from Lecture 10
- Discussion: Reading questions for Lecture 11
- Lecture
    - Testing
- In-Class Activity

Discussion: Experiences from Lecture 10 Activity

- How did you use LLM to review code?
- What types of issues did you try?
- What was it good or bad it?
- How did experience of using LLM compare to not using an LLM?

Discussion: Reading questions for Lecture 10

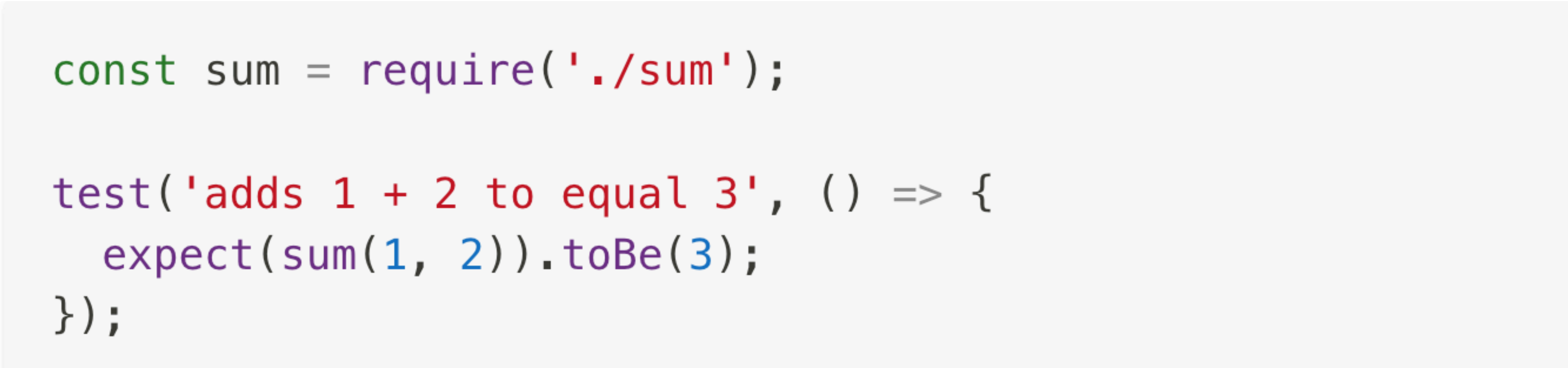
- What questions did you have from readings for Lecture 10
    - Discuss questions & possible answers in group of 3 or 4
    - Come back with 1 question you want to discuss w/ whole class

Testing

Testing

- Techniques to increase confidence that code behaves as expected
- Requirements
  - Inputs
  - Expected outputs
- Some challenges
  - Space of inputs is vast. How do you navigate it? How do you know when you've written enough tests?
  - Oracle problem: how do you know if the output is correct?
- Underlying goal: build an (ideally compact) model of how code should behave across all potential scenarios

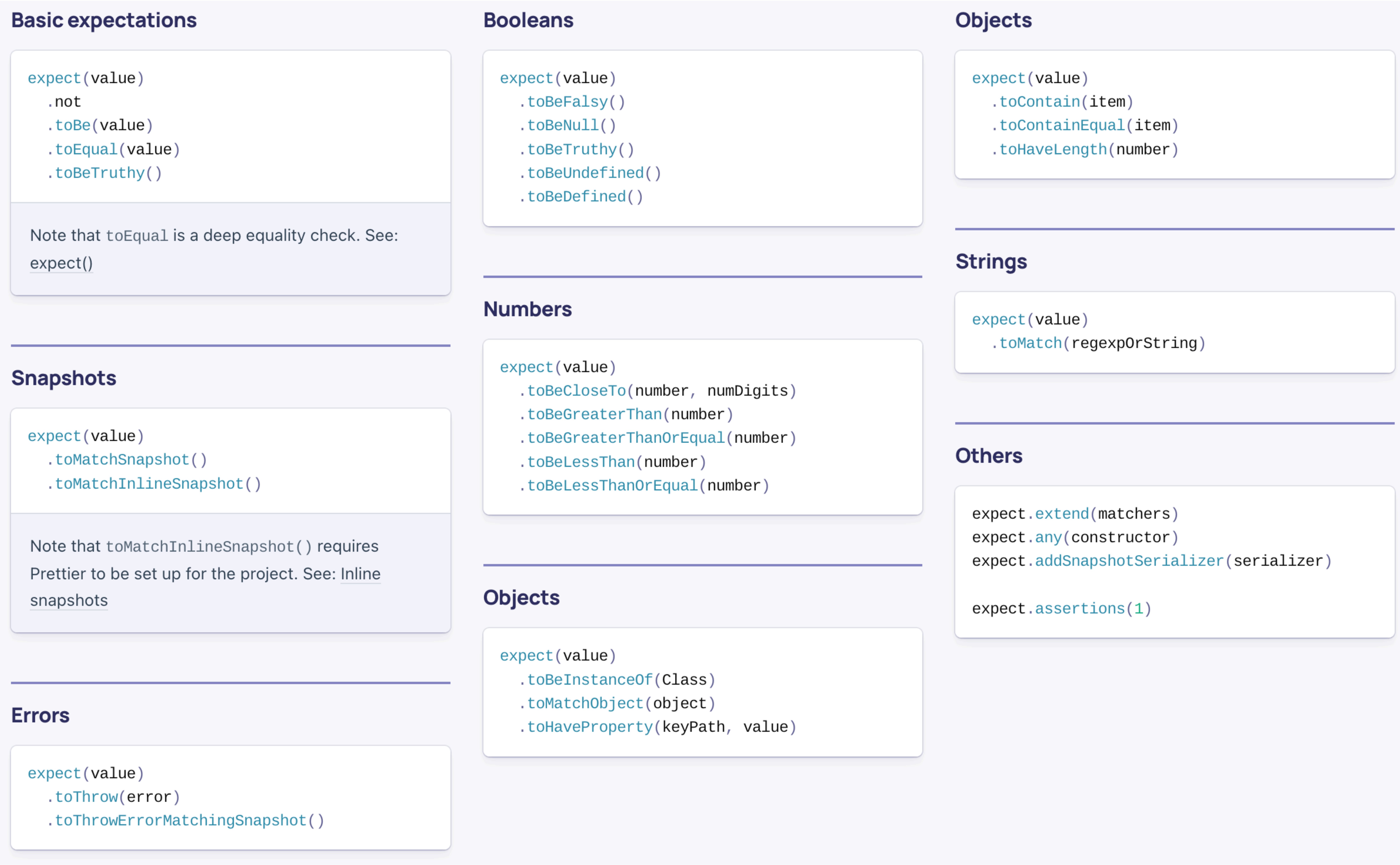
Traditional unit testing



```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

- Developers writes unit tests
- Exercises part of the program relevant to behavior of interest
- Adds assertions on return values or other outputs to ensure that code works as expected

Many ways to write assertions



Basic expectations

```
expect(value)
  .not
  .toBe(value)
  .toEqual(value)
  .toBeTruthy()
```

Note that `toEqual` is a deep equality check. See: expect()

Snapshots

```
expect(value)
  .toMatchSnapshot()
  .toMatchInlineSnapshot()
```

Note that `toMatchInlineSnapshot()` requires Prettier to be set up for the project. See: Inline snapshots

Errors

```
expect(value)
  .toThrow(error)
  .toThrowErrorMatchingSnapshot()
```

Booleans

```
expect(value)
  .toBeFalsy()
  .toBeNull()
  .toBeTruthy()
  .toBeUndefined()
  .toBeDefined()
```

Numbers

```
expect(value)
  .toBeCloseTo(number, numDigits)
  .toBeGreaterThan(number)
  .toBeGreaterThanOrEqual(number)
  .toBeLessThan(number)
  .toBeLessThanOrEqual(number)
```

Objects

```
expect(value)
  .toBeInstanceOf(Class)
  .toMatchObject(object)
  .toHaveProperty(keyPath, value)
```

Objects

```
expect(value)
  .toContain(item)
  .toContainEqual(item)
  .toHaveLength(number)
```

Strings

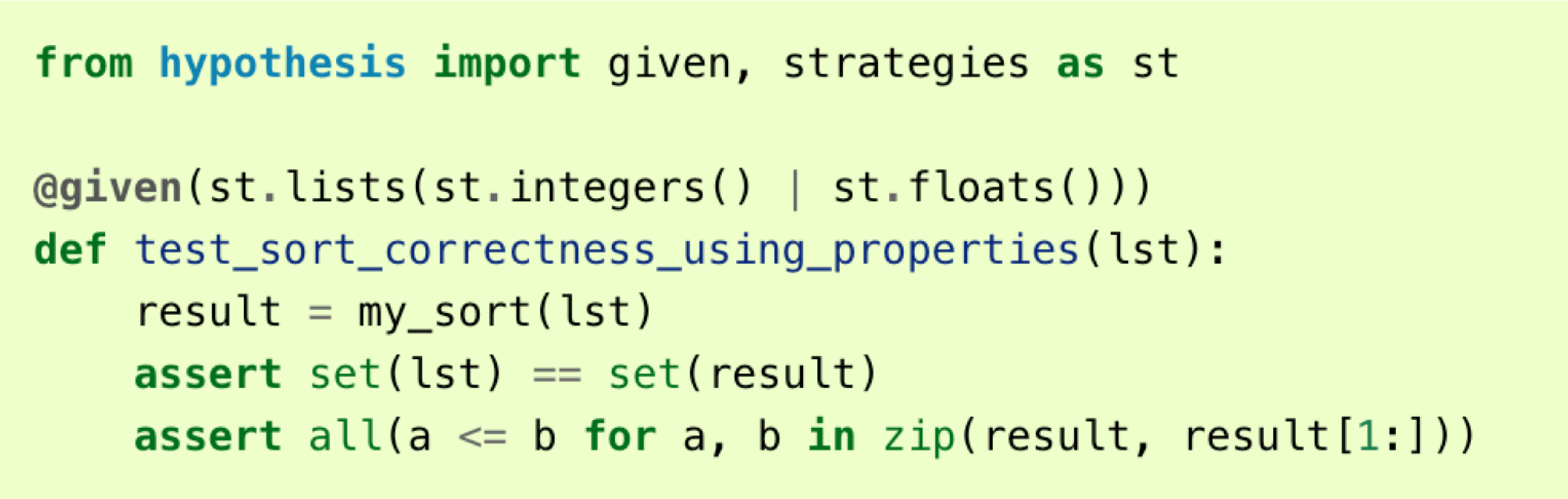
```
expect(value)
  .toMatch(regexpOrString)
```

Others

```
expect.extend(matchers)
expect.any(constructor)
expect.addSnapshotSerializer(serializer)

expect.assertions(1)
```

https://devhints.io/jest

Property-Based Testing



```python
from hypothesis import given, strategies as st

@given(st.lists(st.integers() | st.floats()))
def test_sort_correctness_using_properties(lst):
    result = my_sort(lst)
    assert set(lst) == set(result)
    assert all(a <= b for a, b in zip(result, result[1:]))
```

https://hypothesis.readthedocs.io/en/latest/index.html

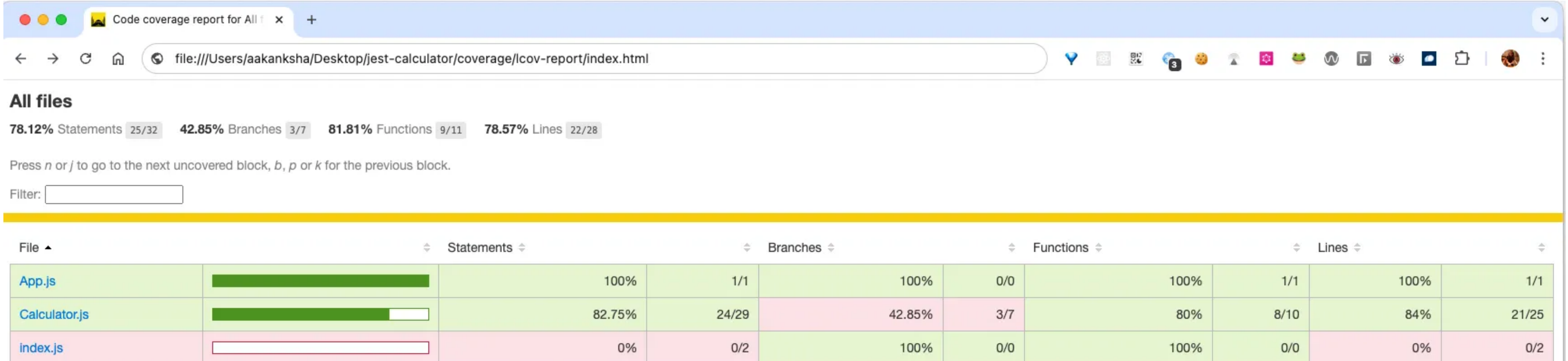
- Rather than manually select individual outputs, instead describe a *property* that should always hold for all inputs
    - output list is always sorted
    - output list contains the same elements as the input list
- Testing framework that explores input space to try to find counterexamples, using property as oracle

Related activities

- Debugging -- what happens if a test fails unexpectedly?
- Software documentation -- can generate information about expected behavior of code
- Program comprehension -- may need to understand program to understand what to test
- Maintenance -- may need to update tests when code changes

Measuring how complete tests are with coverage

- Statement coverage: has each statement executed?
- Branch coverage: have both branches of statements with branches (e.g., if, while) executed?
- Path coverage: have all distinct paths through the program been executed?



Code coverage report for All f

file:///Users/aakanksha/Desktop/jest-calculator/coverage/lcov-report/index.html

All files

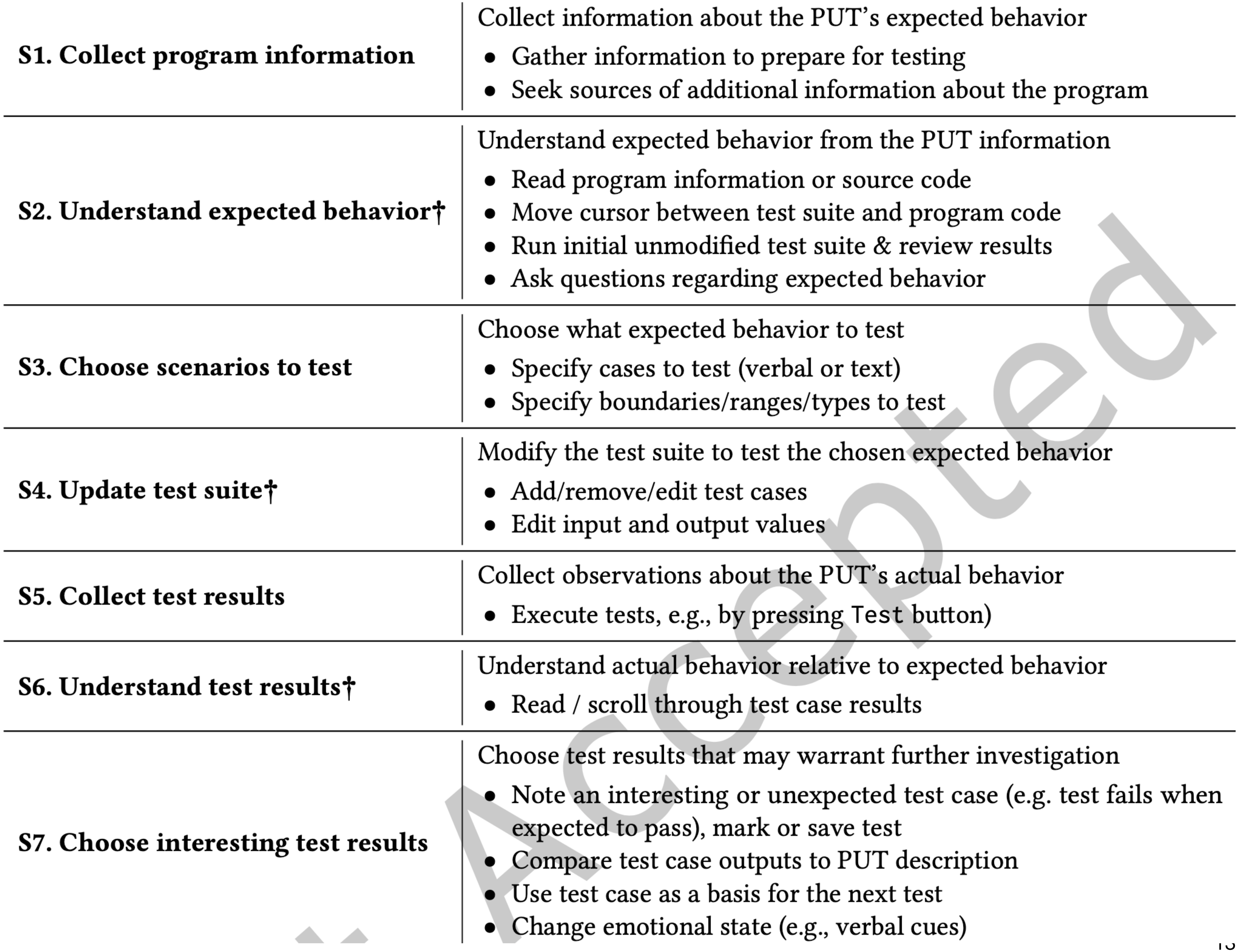
78.12% Statements 25/32 **42.85%** Branches 3/7 **81.81%** Functions 9/11 **78.57%** Lines 22/28

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements		Branches		Functions		Lines	
App.js		100%	1/1	100%	0/0	100%	1/1	100%	1/1
Calculator.js		82.75%	24/29	42.85%	3/7	80%	8/10	84%	21/25
index.js		0%	0/2	100%	0/0	100%	0/0	0%	0/2

Process of testing



S1. Collect program information	Collect information about the PUT's expected behavior • Gather information to prepare for testing • Seek sources of additional information about the program
S2. Understand expected behavior†	Understand expected behavior from the PUT information • Read program information or source code • Move cursor between test suite and program code • Run initial unmodified test suite & review results • Ask questions regarding expected behavior
S3. Choose scenarios to test	Choose what expected behavior to test • Specify cases to test (verbal or text) • Specify boundaries/ranges/types to test
S4. Update test suite†	Modify the test suite to test the chosen expected behavior • Add/remove/edit test cases • Edit input and output values
S5. Collect test results	Collect observations about the PUT's actual behavior • Execute tests, e.g., by pressing `Test` button)
S6. Understand test results†	Understand actual behavior relative to expected behavior • Read / scroll through test case results
S7. Choose interesting test results	Choose test results that may warrant further investigation • Note an interesting or unexpected test case (e.g. test fails when expected to pass), mark or save test • Compare test case outputs to PUT description • Use test case as a basis for the next test • Change emotional state (e.g., verbal cues)

Matthew C. Davis, Sangheon Choi, Amy Wei, Sam Estep, Brad A. Myers, and Joshua Sunshine. 2025. TestLoop: A Process Model Describing Human-in-the-Loop Software Test Suite Generation. ACM Trans. Softw. Eng. Methodol. Just Accepted (September 2025). https://doi.org/10.1145/3765754

Discussion: How do you use testing?

Challenges with testing

- May take from 25% or more of engineering time
- Still most often a very manual process of writing tests
- Not very exciting or popular activity, so often neglected

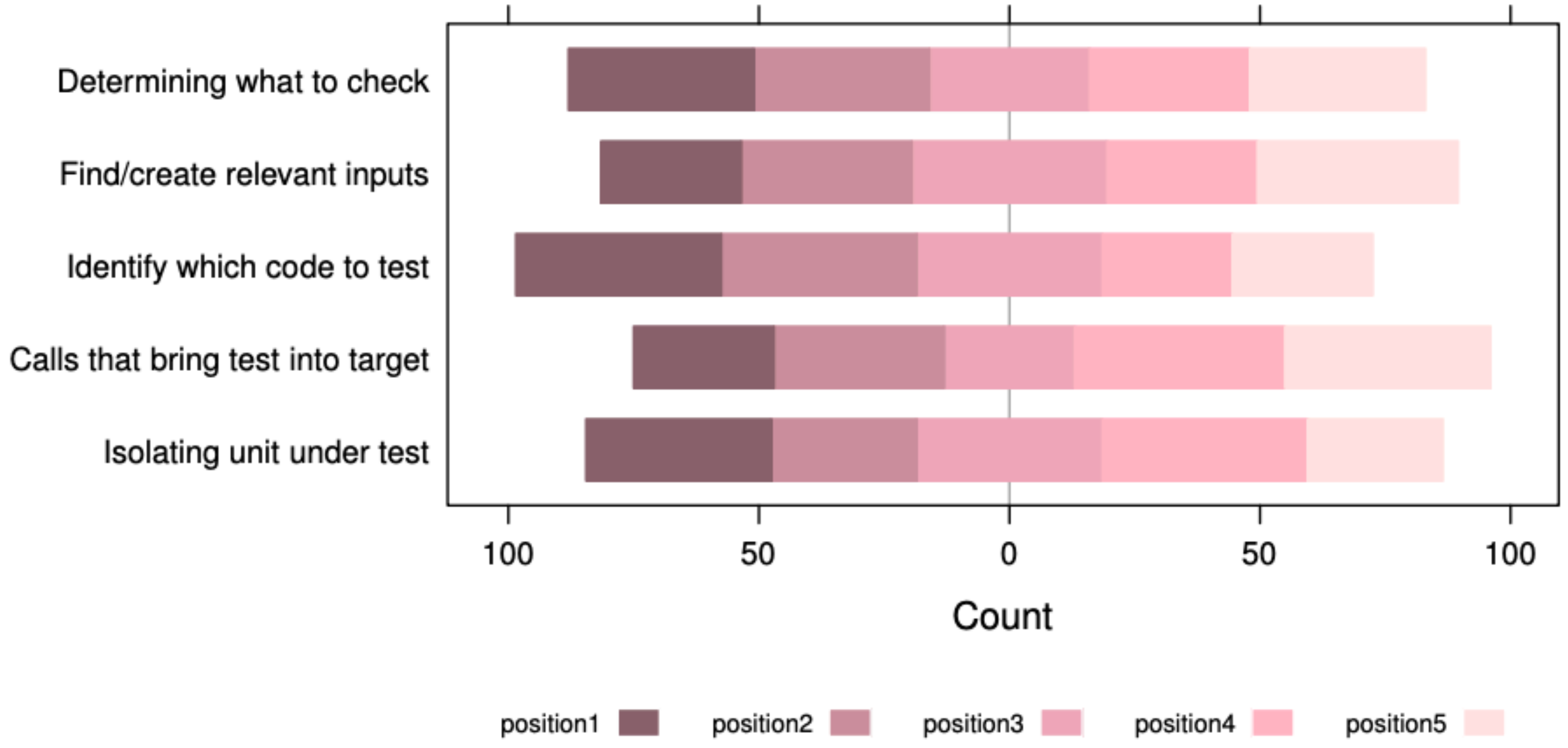


Fig. 10. What is most difficult about writing unit tests?

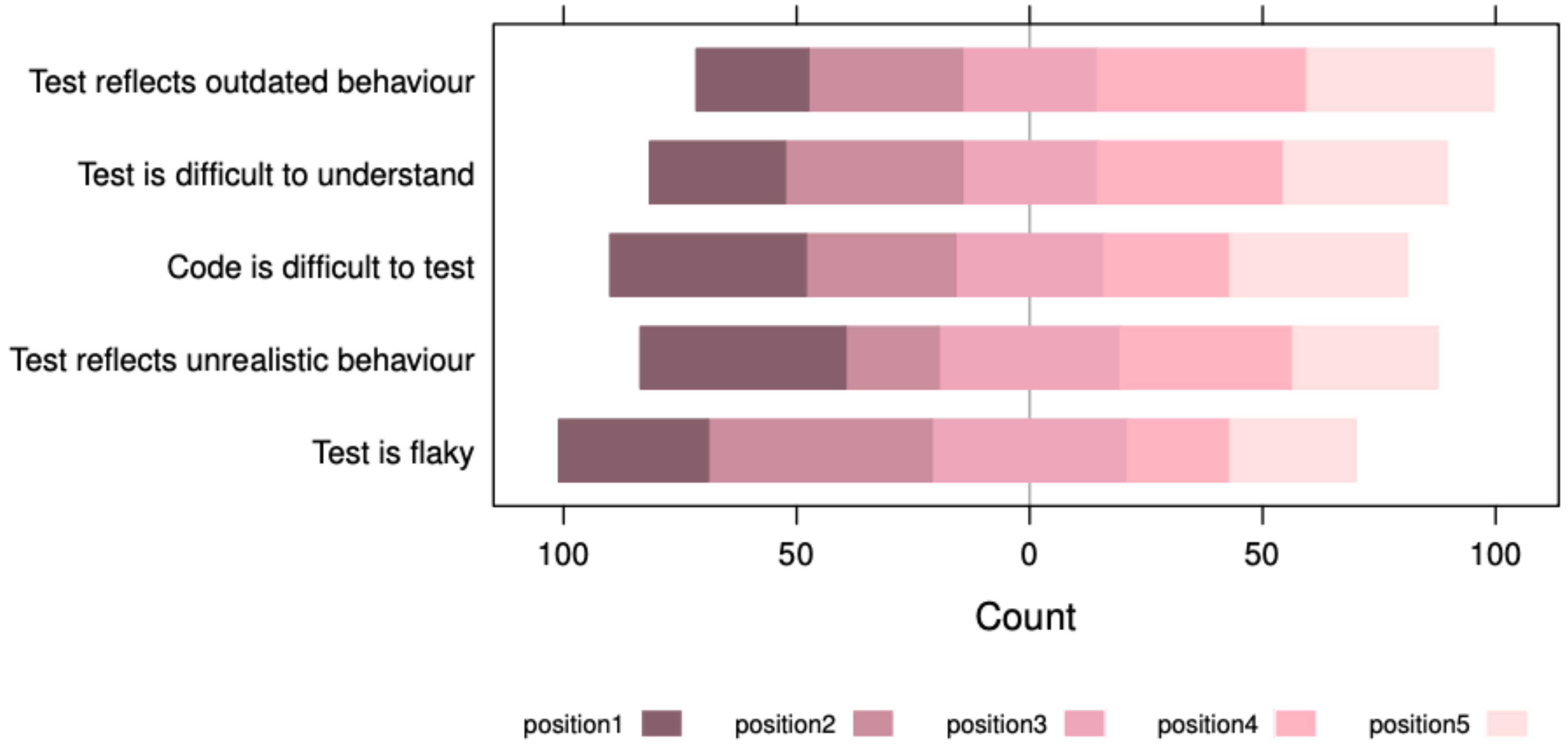
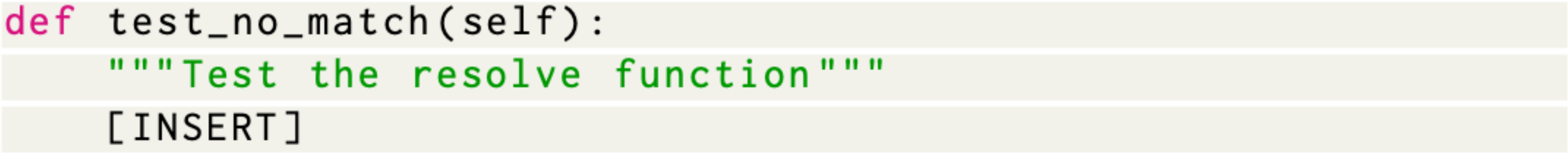


Fig. 11. What is most difficult about fixing unit tests?

E. Daka and G. Fraser, "A Survey on Unit Testing Practices and Problems," *2014 IEEE 25th International Symposium on Software Reliability Engineering*, Naples, Italy, 2014, pp. 201-211, doi: 10.1109/ISSRE.2014.11.

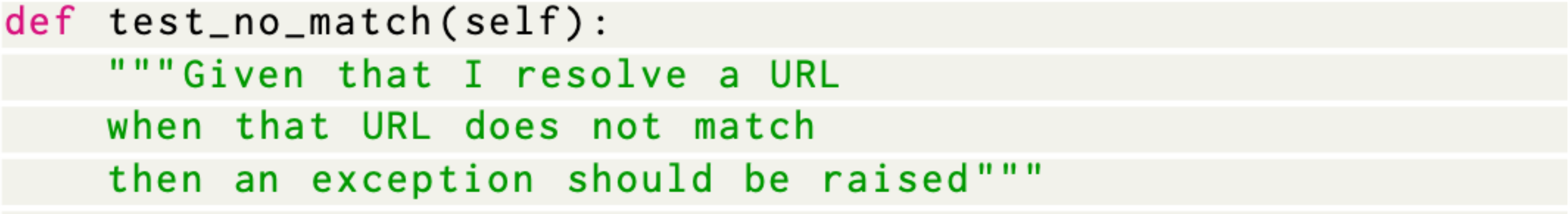
Testing with LLMs

- Let's ask an LLM to generate unit tests for a function!
- What context should we give it?
  - method description



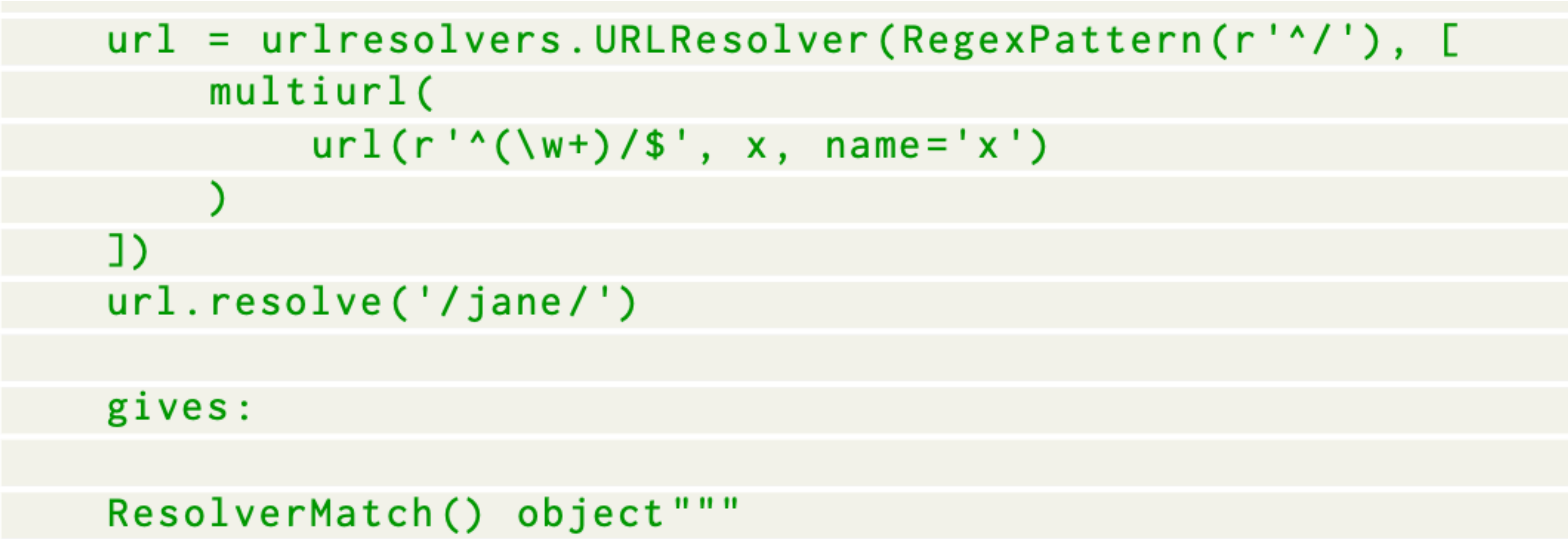
```
def test_no_match(self):
    """Test the resolve function"""
    [INSERT]
```

  - scenario description



```
def test_no_match(self):
    """Given that I resolve a URL
    when that URL does not match
    then an exception should be raised"""
```

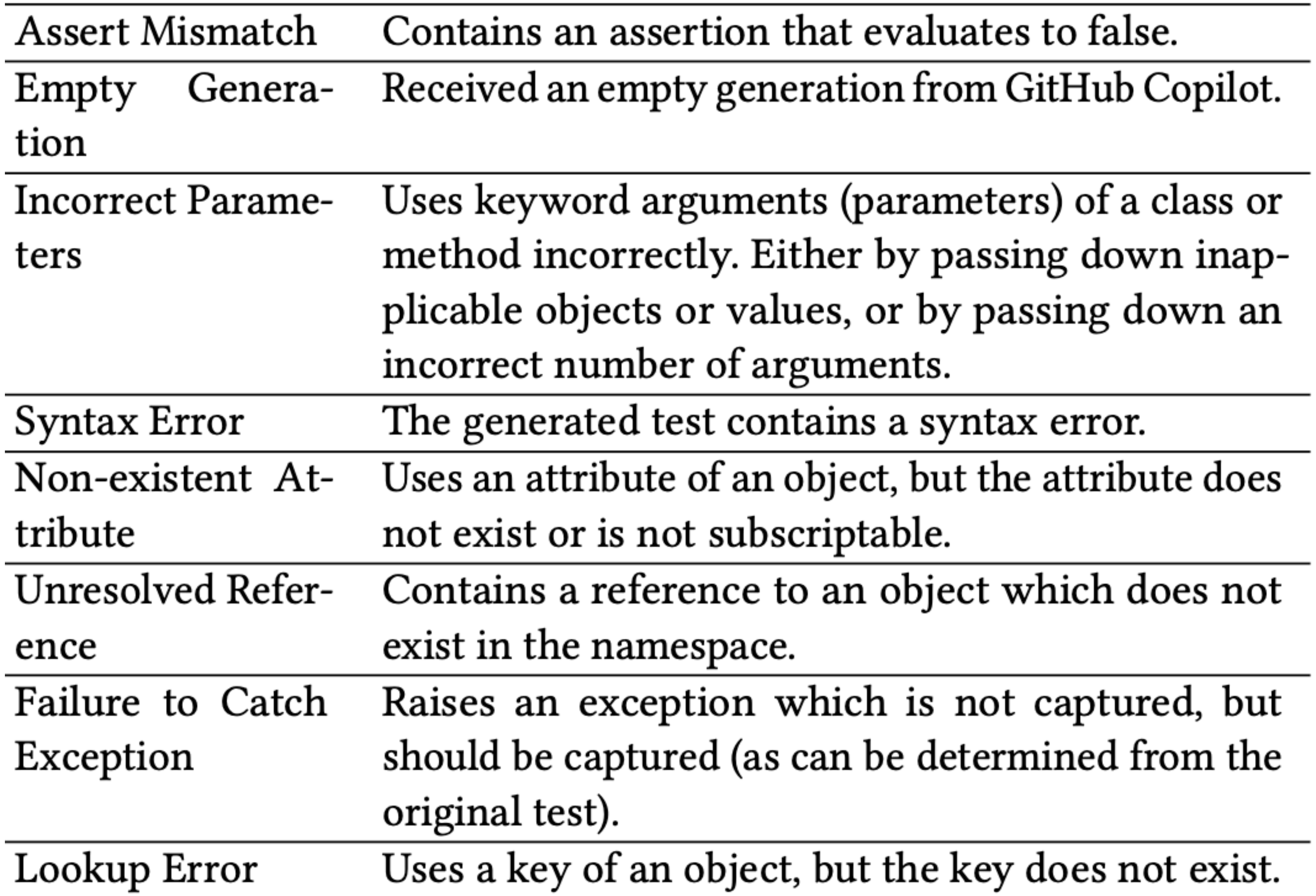
  - detailed usage spec



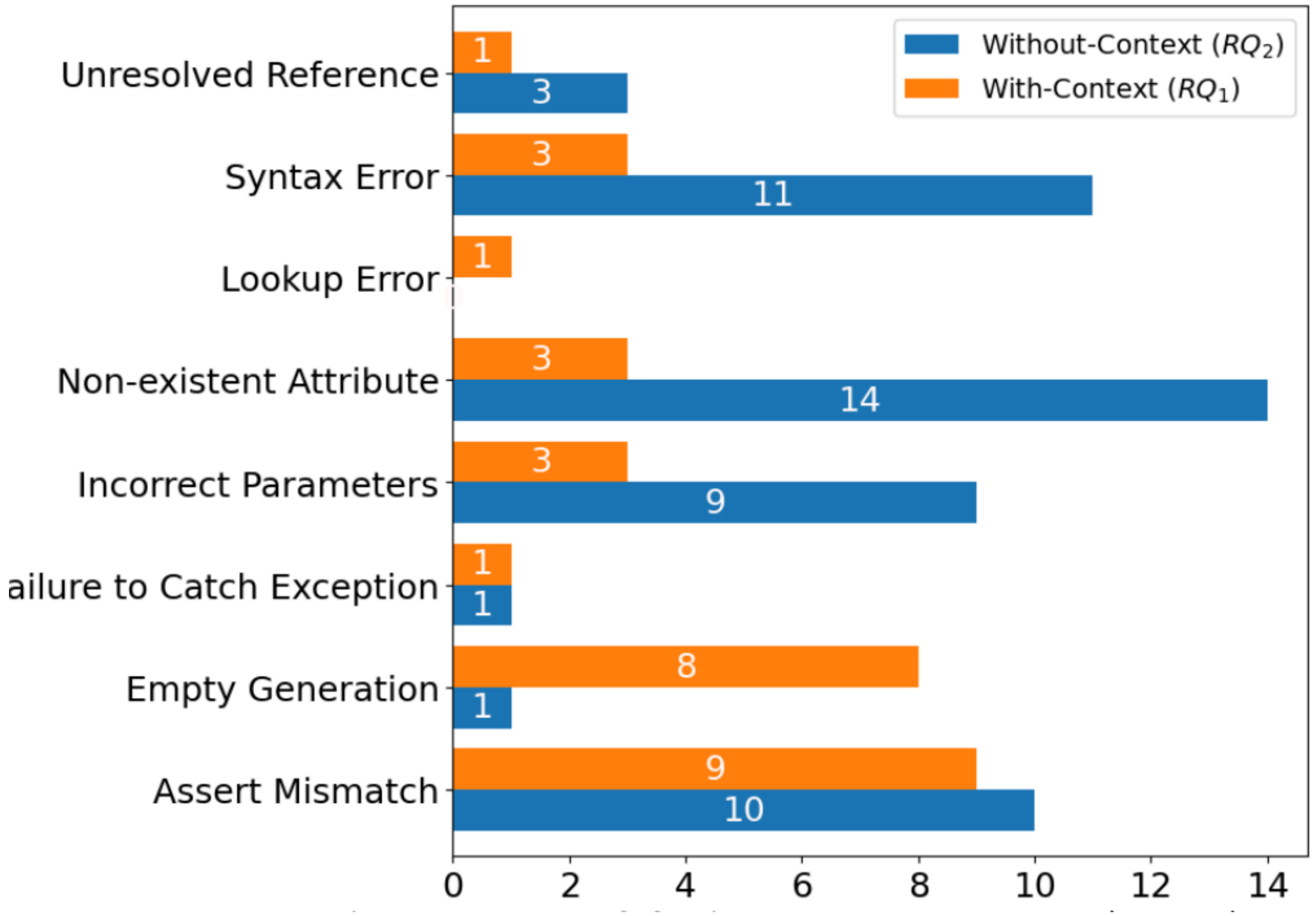
```
    url = urlresolvers.URLResolver(RegexPattern(r'^/'), [
        multiurl(
            url(r'^(\w+)/$', x, name='x')
        )
    ])
    url.resolve('/jane/')

    gives:

    ResolverMatch() object"""
```

Khalid El Haji, Carolin Brandt, and Andy Zaidman. 2024. Using GitHub Copilot for Test Generation in Python: An Empirical Study. In Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024) (AST '24). Association for Computing Machinery, New York, NY, USA, 45–55. https://doi.org/10.1145/3644032.3644443

What can go wrong with LLM generated tests



Assert Mismatch	Contains an assertion that evaluates to false.
Empty Generation	Received an empty generation from GitHub Copilot.
Incorrect Parameters	Uses keyword arguments (parameters) of a class or method incorrectly. Either by passing down inapplicable objects or values, or by passing down an incorrect number of arguments.
Syntax Error	The generated test contains a syntax error.
Non-existent Attribute	Uses an attribute of an object, but the attribute does not exist or is not subscriptable.
Unresolved Reference	Contains a reference to an object which does not exist in the namespace.
Failure to Catch Exception	Raises an exception which is not captured, but should be captured (as can be determined from the original test).
Lookup Error	Uses a key of an object, but the key does not exist.



Khalid El Haji, Carolin Brandt, and Andy Zaidman. 2024. Using GitHub Copilot for Test Generation in Python: An Empirical Study. In Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024) (AST '24). Association for Computing Machinery, New York, NY, USA, 45–55. https://doi.org/10.1145/3644032.3644443

10 min break

In-Class Activity

- In groups of 2, try using Cursor to write tests to maximize branch coverage
    - Start with one of your own repos (e.g., your city simulator)
    - Work together with Cursor to create tests, focusing on maximizing branch coverage
        - If using JS, can use built in Jest report on coverage
    - Fix issues found by the tests as they arise
- Deliverables
    - Screen recording through Kaltura
        - Upload to OneDrive, turn on link sharing, share link in Lecture 10 activity submission on Canvas
        - Submit answers to questions on your experiences on Canvas (next slide)
- Aim to finish by 7:10pm today; Due tomorrow at 4:30pm

Questions to answer

- How did you work with Cursor to create unit tests?
- How effectively was Cursor able to test different behaviors?
- What, if anything, was Cursor not able to test?
- In what ways did you provide guidance to Cursor to help in creating tests?
- What overall test coverage were you able to achieve?

- **Deliverable**: Submit through Canvas, at least a page