Testing with purpose

HOW DO WE *IDENTIFY* PROBLEMS WITH OUR CODE?

Doctests: for understanding

```
def insert_after(lst: List[int], n1: int, n2: int) -> None:
    """After each occurrence of <n1> in <lst>, insert <n2>.

    >>> lst = [5, 1, 2, 1, 6]
    >>> insert_after(lst, 1, 99)
    >>> lst
    [5, 1, 99, 2, 1, 99, 6]
    """
```

Unit tests: for assessing “units” of code

“unit” = one function, usually

Unit tests are typically written in a separate file, enabling us to write a comprehensive set of tests without impacting readability of the code itself.

The key technical tools are:

- the *assertion* (Python: `assert`)
- the *test case* (Python: a function named `test_*`)

`pytest`: for automating unit testing

```
def test_simple() -> None:
    input_list = [5, 1, 2, 1, 6]
    insert_after(input_list, 1, 99)
    expected = [5, 1, 99, 2, 1, 99, 6]
    assert input_list == expected
```

Choosing test cases

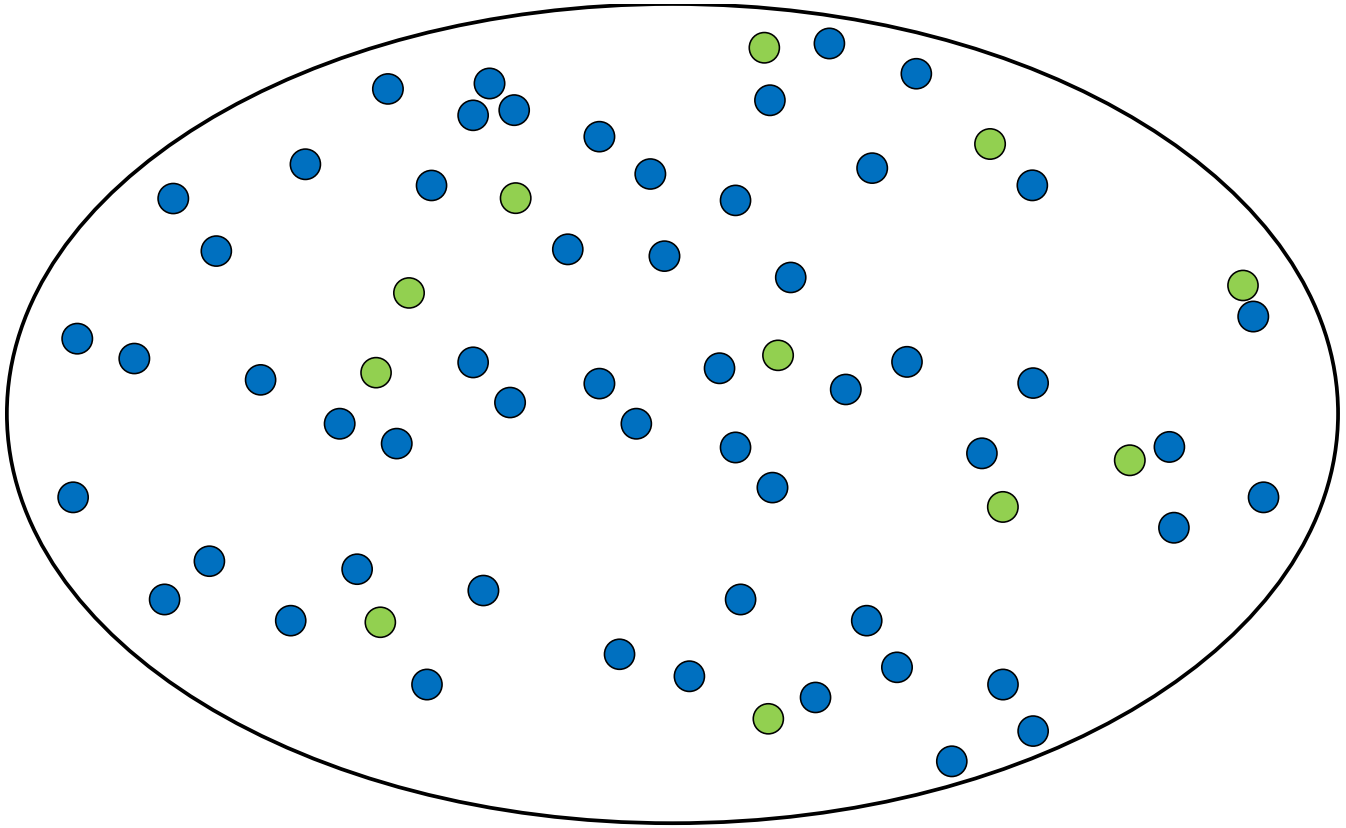
This is the hard part (and the interesting part).

You are making an argument to *yourself* and any other reader.

You want to conclude that the code works on any valid input.

tested

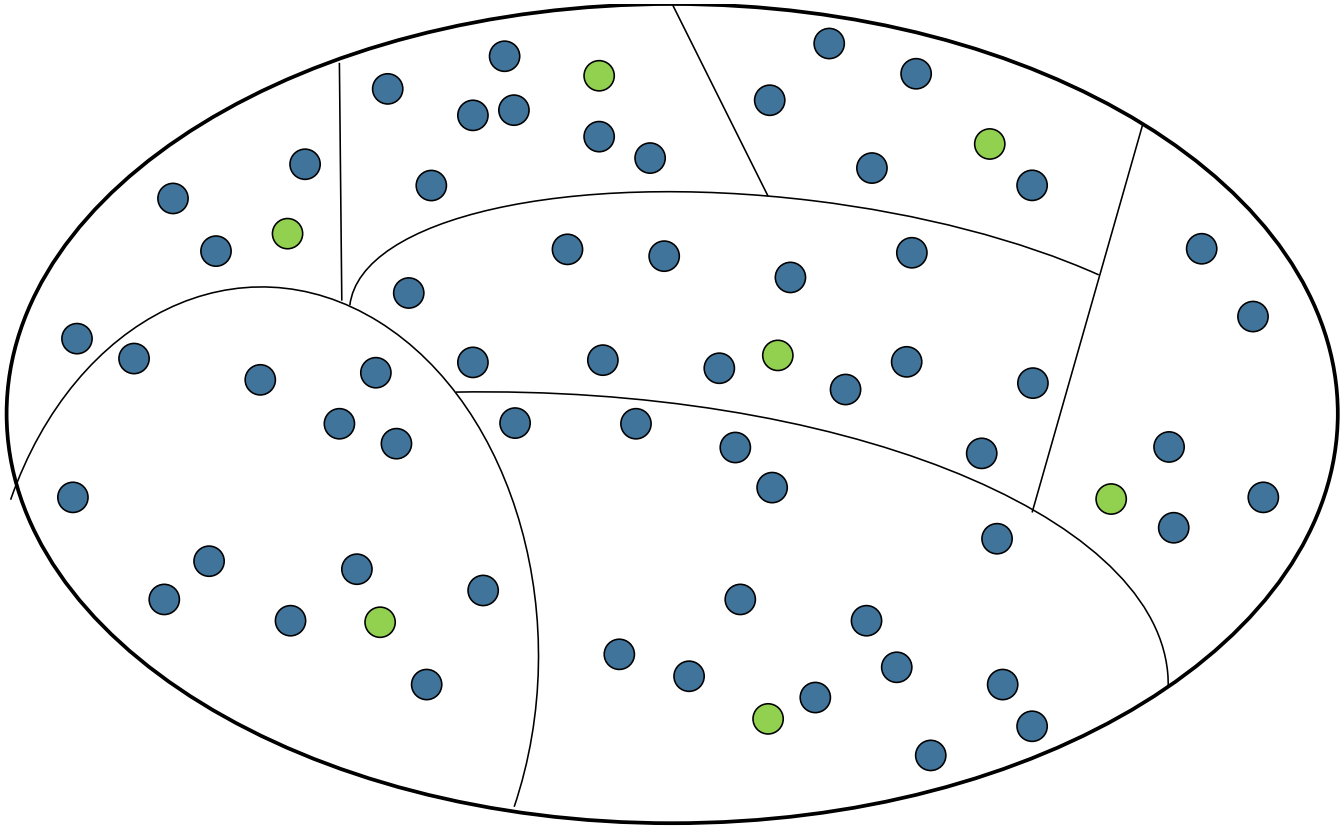
untested



Choosing test cases randomly is futile.

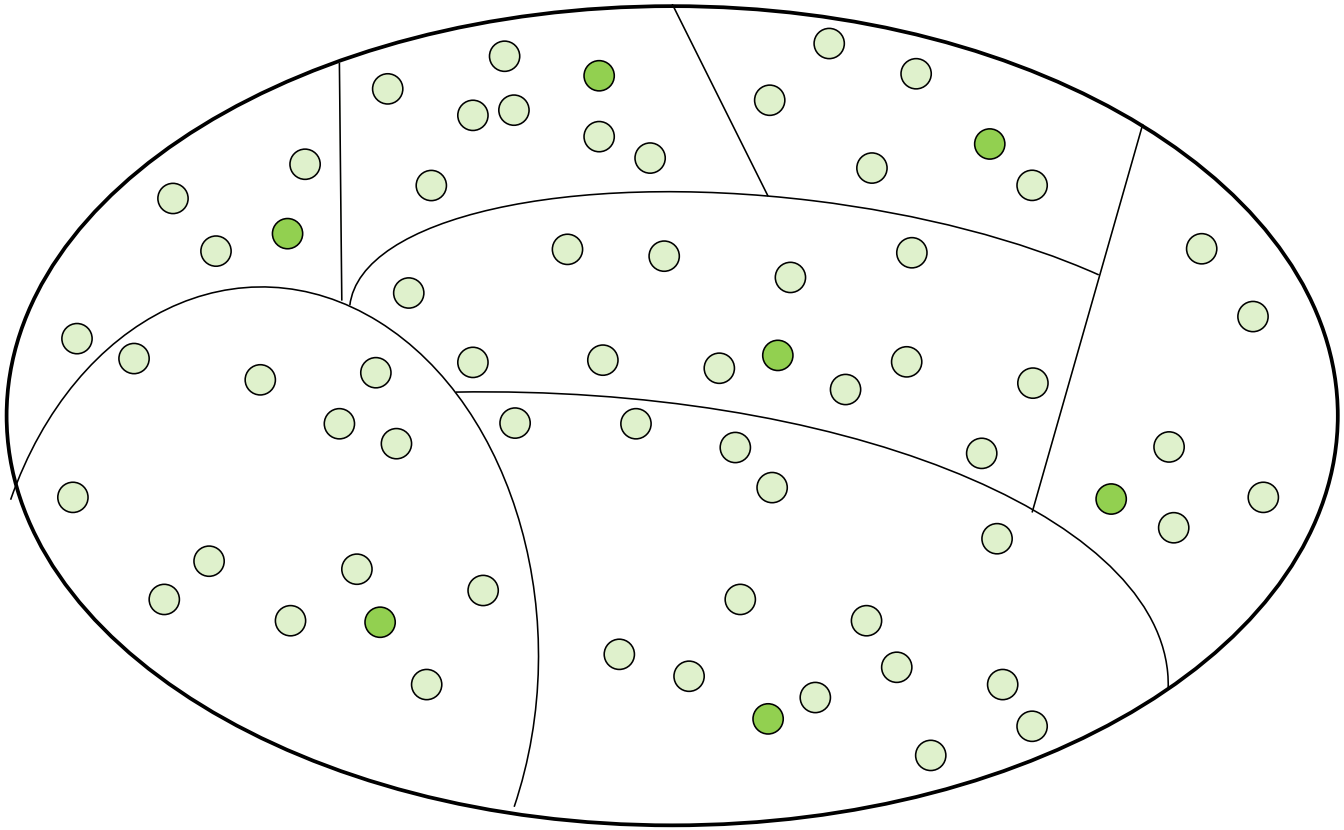
tested

untested



Instead, divide cases into categories of “equivalent” inputs and test one of each.

- tested
- untested but reasonable to extrapolate



If well-chosen, it is reasonable to extrapolate.

Property tests

An alternate strategy rather than giving a specific input and expected output.

Suppose we are testing a function to find the max in a list.

	Instead of specifying this:	**We specify this:**
Input	[3, 62, 4, 53, 9]	A list of integers
Output	62	An element of the list, or None

Inputs (many!) are generated and tested automatically.

`hypothesis`: for property-based testing

Demonstration