pytest

- `unittest`:
  - included in standard python
  - widely used
  - requires lots of boilerplate code
- `pytest`:
  - not part of standard python (need to install package)
  - allows smaller test cases without overhead

pytest

- doesn't require classes -> can have standalone test functions
- even class based tests don't need to be subclasses
- uses built-in `assert` statement rather than custom methods
- to run: type `pytest` in terminal

pytest - finding tests

- startes in current folder, searches for:
  - `tests` directory (folder) or any modules/subpackages with names beginning with `test_`
  - in each, looks for:
    - any functions starting with `test`
    - any classes starting with `Test`
- often structure project with:
  - `src` folder for project source code
  - `tests` folder for unit tests

pytest - test function

```
def foo(x,y):
    return x/y

def test_div_even():
    assert foo(4,2) == 2
```

pytest - exception

```
def foo(x,y):
    if y == 0:
        raise ValueError
    return x/y

def test_div_by0():
    with pytest.raises(ValueError):
        foo(6,0)
```

pytest - exception with message

```
def foo(x,y):
    if y == 0:
        raise ValueError("cannot divide by 0")
    return x/y

def test_div_by0():
    with pytest.raises(ValueError, match="cannot divide by 0"):
        foo(6,0)
```

pytest - class

```
class TestFoo:
    def test_div_even():
        assert foo(4,2) == 2

    def test_div_by0():
        with pytest.raises(ValueError):
            foo(6,0)
```

testing - common setup (unittest)

- often have common code that should run before each test
  - add `setUp()` method to class
- often have common code that should run after each test
  - add `tearDown()` method to class
- also versions for whole class
  - `setUpClass()`
  - `tearDownClass()`