THE NEXT STEP IN PYTHON: TESTING WITH PYTEST

Yangyang Li

2025-05-05

pytest

Next Steps in Python

Workshop starts at 12:02

Materials for today

To work on your computer.

- https://github.com/nuitrcs/testing_with_pytest

To work on the cloud.

- <Item 1>

Follow @Northwestern_IT on Instagram for the latest workshop and bootcamp updates.

This workshop is brought to you by:

Northwestern IT
Research Computing and Data Services

Need help?

- AI, Machine Learning, Data Science
- Statistics
- Visualization
- Collecting web data (scraping, APIs), text analysis, extracting information from text
- Cleaning, transforming, reformatting, and wrangling data
- Automating repetitive research tasks
- Research reproducibility and replicability
- Programming, computing, data management, etc.
- R, Python, SQL, MATLAB, Stata, SPSS, SAS, etc.

Request a **FREE** consultation at bit.ly/rcdsconsult.

Python Testing with pytest

Making your code more reliable in 1 hour

pytest

WORKSHOP AGENDA

1. **Introduction to Testing** (10 min)
   - Why test?
   - Types of tests
   - Testing terminology
2. **Getting Started with pytest** (15 min)
   - Basic structure
   - Running tests
   - Understanding results
3. **Writing Effective Tests** (15 min)
   - Test design patterns
   - Parameterization
4. **Test-Driven Development** (15 min)
   - Red-Green-Refactor cycle
   - TDD in practice
   - Benefits & challenges
5. **Q&A and Wrap-up** (5 min)

INTRODUCTION TO TESTING

WHY TEST YOUR CODE?

- **Find bugs early** — Before your users do
- **Refactor with confidence** — Change code without fear
- **Documentation** — Tests show how code should work
- **Design improvement** — Testing forces better architecture
- **Professional practice** — Industry standard skill

Test Early, Test Often

"Code without tests is broken by design." — Jacob Kaplan-Moss

pytest

POLL QUESTION

How much experience do you have with testing Python code?

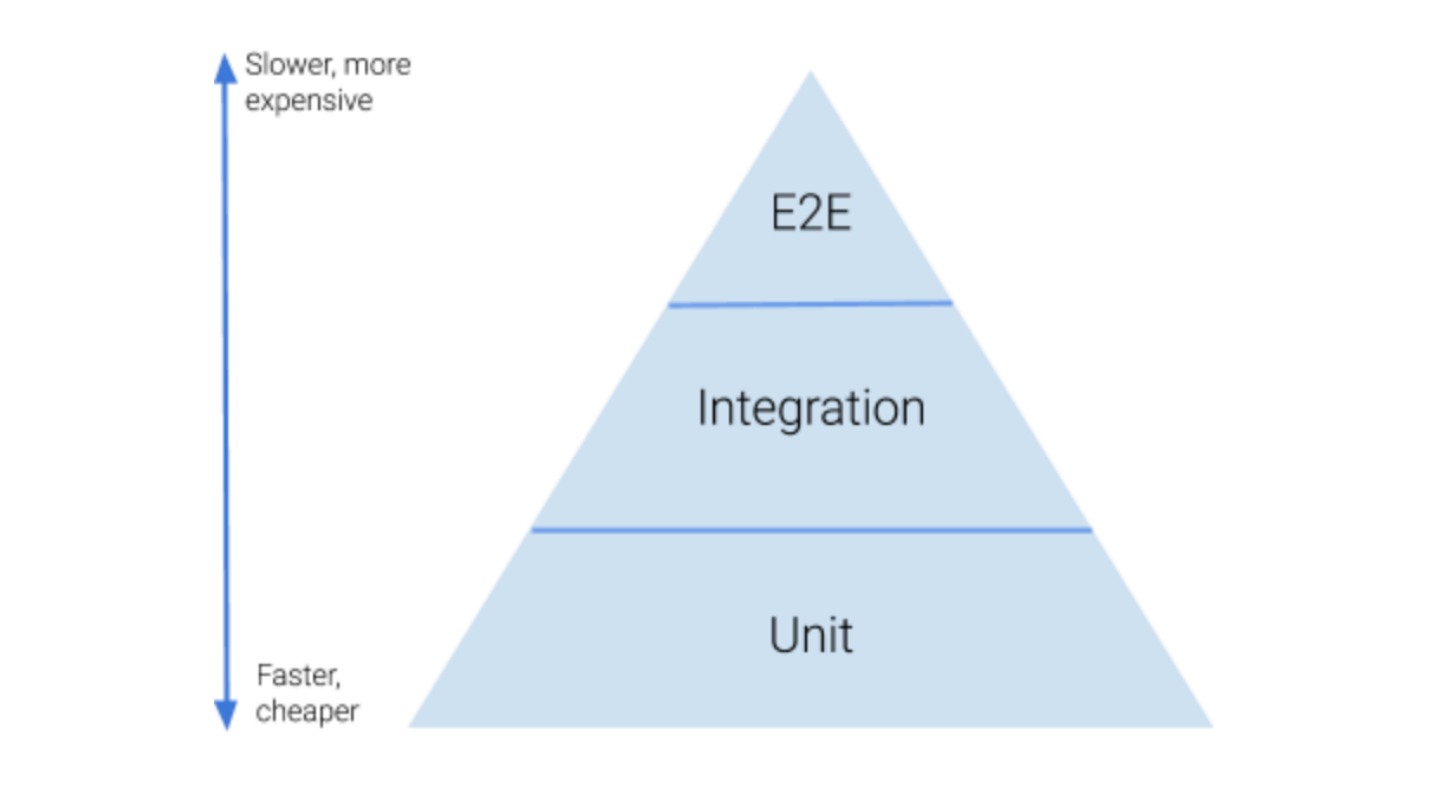
A. None - I'm completely new to testing

B. Minimal - I've written a few basic tests

C. Moderate - I use pytest occasionally

D. Experienced - I practice TDD regularly

TESTING PYRAMID



Unit tests
- Test individual functions
- Fast, isolated
- Many tests

Integration tests
- Test component interaction
- Fewer, more complex

pytest

GETTING STARTED WITH PYTEST

WHY PYTEST?

Advantages

- Simple, Pythonic syntax
- Rich assertion messages
- Powerful fixtures & plugins
- Easy parameterization

vs. unittest

```
# unittest
self.assertEqual(1 + 1, 2)

# pytest — simpler!
assert 1 + 1 == 2
```

```
# Installation
pip install pytest
```


pytest

BASIC PYTEST STRUCTURE

```python
# test_example.py
def test_addition():
    # Simple assertions
    assert 1 + 1 == 2


def test_string_methods():
    # pytest shows values on failure
    assert "hello".upper() == "HELLO"
    assert "world".capitalize() == "World"
```

```python
# Run with:
pytest test_example.py -v

# Output:
# test_example.py::test_addition PASSED
# test_example.py::test_string_methods PASSED
```


pytest

LIVE EXERCISE

ⓘ Let's write a test together!

1. Create `calculator.py` with basic functions:

```
def add(a, b):
    return a + b
```

2. Write `test_calculator.py`:

```
from calculator import add


def test_add():
    assert add(1, 2) == 3
    assert add(-1, 1) == 0
```

3. Run: `pytest test_calculator.py -v`

pytest

WRITING EFFECTIVE TESTS

PYTHON CLASSES REFRESHER

Classes in Python

- Blueprint for creating objects
- Encapsulate data (attributes) and behavior (methods)
- Create with the `class` keyword
- Instantiate with `object = ClassName()`
- `self` refers to the instance

```
class Calculator:
    def __init__(self, initial_value=0):
        self.result = initial_value

    def add(self, value):
        self.result += value
        return self.result
```

Using a Class

```
# Create an instance
calc = Calculator(10)

# Call methods
calc.add(5)  # returns 15
calc.add(3)  # returns 18

# Access attributes
print(calc.result)  # 18
```


pytest

THE AAA PATTERN

Arrange → Act → Assert

THE AAA PATTERN

```python
# Testing a User class
def test_valid_email():
    # Arrange
    user = User('Test', 'test@example.com')

    # Act
    result = user.is_valid_email()

    # Assert
    assert result is True
```

```python
class User:
    def __init__(self, name, email):
        self.name = name
        self.email = email

    def is_valid_email(self):
        return ('@' in self.email
                and '.' in self.email)
```


pytest

PARAMETERIZED TESTS

Before

```python
def test_email_valid():
    user = User('Test', 'test@example.com')
    assert user.is_valid_email() is True

def test_email_invalid_no_at():
    user = User('Test', 'invalid-email')
    assert user.is_valid_email() is False

def test_email_invalid_no_dot():
    user = User('Test', 'invalid@nodotatall')
    assert user.is_valid_email() is False
```

After

```python
import pytest

@pytest.mark.parametrize("email,expected", [
    ("test@example.com", True),
    ("invalid-email", False),
    ("another@test.org", True),
    ("missing-dot@com", False),
])
def test_email_validation(email, expected):
    user = User('Test', email)
    assert user.is_valid_email() is expected
```


pytest

POLL QUESTION

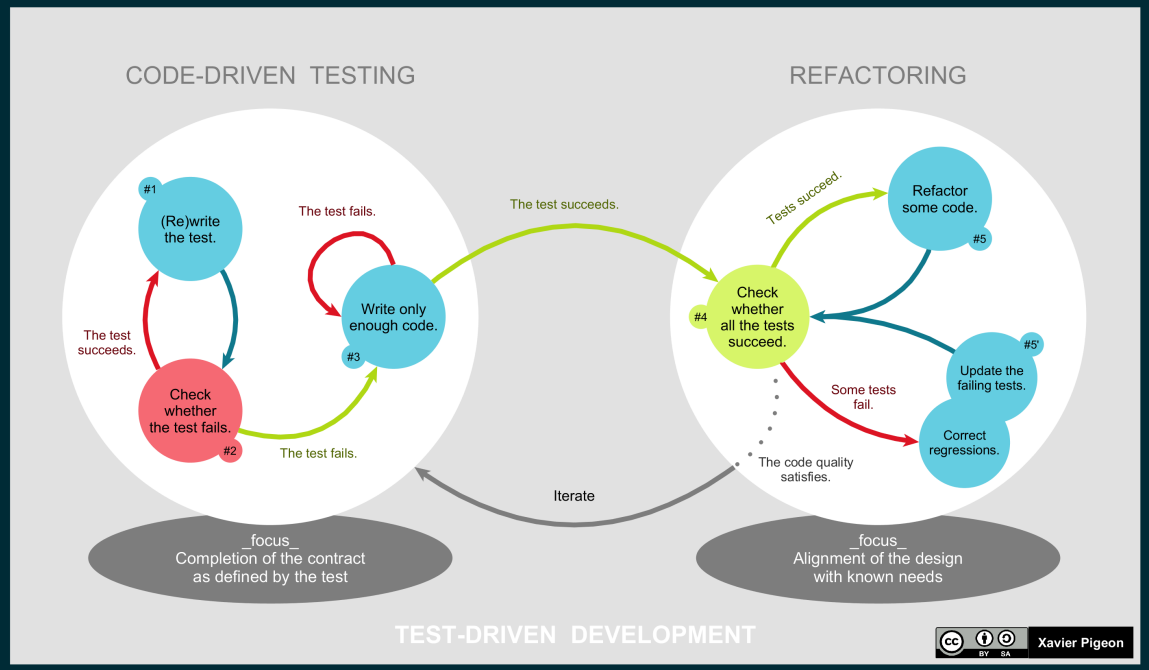
What are you most interested in testing?

A. Functions with calculations
B. Data validation logic
C. API interactions
D. Database operations

pytest

TEST-DRIVEN DEVELOPMENT

THE TDD CYCLE



1. **Red:** Write a failing test
2. **Green:** Write minimal code to pass
3. **Refactor:** Improve without breaking tests

pytest

TDD DEMO: SHOPPING CART

Red Phase | Green Phase | Next Test | Implementation

```python
def test_add_item():
    cart = ShoppingCart()
    cart.add_item("apple", 1.0)
    assert cart.total() == 1.0
```


pytest

STATISTICS FUNCTION EXAMPLE

Test First | Implementation | Edge Cases

```python
def test_calculate_statistics():
    numbers = [1, 2, 3, 4, 5]
    stats = calculate_statistics(numbers)
    assert stats["min"] == 1
    assert stats["max"] == 5
    assert stats["average"] == 3.0
```

TDD BENEFITS & CHALLENGES

Benefits
- Forces clear requirements
- Prevents over-engineering
- Built-in regression testing
- Improves API design
- Documentation by example

Challenges
- Learning curve
- Requires discipline
- Can slow initial development
- Test maintenance
- “Test-induced design damage”

Tip

Start small and build your testing habit incrementally!

POLL QUESTION

How might TDD change your workflow?

A. Initial slowdown but long-term gain

B. Clarify requirements before coding

C. Improve my code architecture

D. Still not convinced it's worth it

WRAP-UP

KEY TAKEAWAYS

1. **Testing is an investment** in code quality and maintainability
2. **pytest makes testing approachable** with simple syntax and powerful features
3. **Start with simple unit tests** and build from there
4. **TDD can guide development** and improve your software design
5. **Practice is essential** to get comfortable with testing

RESOURCES

- pytest Documentation
- Python Testing with pytest (Brian Okken)
- Real Python Testing Guides
- Workshop materials: https://github.com/nuitrcs/testing_with_pytest

THANK YOU!

Questions?

Contact: yangyang.li@northwestern.edu
GitHub: @cauliyang

pytest