Unit testing with pytest

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

What is a unit test

Unit - the smallest working part that can be tested.

- Examples of units: functions, methods, classes, modules, etc.

Unit testing - software testing method.

- Unit testing allows one to scrutinize the correctness of a unit.

Test case - a set of unit inputs and expected outputs.

- Test case summarizes a particular piece of the problem.

Why to use unit tests

Unit tests - are a foundation for testing "the bigger picture" of the software.

Use cases:

- When bugs found
- During development
- After implemented changes

How to create a unit test

Step-by-step:

1. **Decide which units to test**
2. **Define test cases** (the creative part):
   - "What are the possible unit outcomes?"
   - "How can one use the unit?"
   - "How should the unit behave in all those cases?"
3. **Write code for each test case**
4. **Run the tests** and analyze the results

Creating a unit test: example

Unit to test:

```
# Function for a sum of elements
def sum_of_arr(array:list) -> int:
    return sum(array)
```

Test cases:

1. Input is a list of numbers (as expected) - should return the `sum`
2. Input is an empty list - should return `0`
3. Input is a list containing one `number` - should return the `number`

Creating a unit test: code

```
# Test Case 1: regular array
def test_regular():
    assert sum_of_arr([1, 2, 3]) == 6
    assert sum_of_arr([100, 150]) == 250
```

```
# Test Case 2: empty list
def test_empty():
    assert sum_of_arr([]) == 0
```

```
# Test Case 3: one number
def test_one_number():
    assert sum_of_arr([10]) == 10
    assert sum_of_arr([0]) == 0
```

Summary

Unit test - a test that verifies that a unit works as expected.

Test case - inputs and outputs summarizing a particular piece of the problem.

Use cases:

- When bugs found
- During development
- After implemented changes

Defining test cases - involves creativity.

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

Feature testing with pytest

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

What is a feature test

Feature

- a software system functionality.
- satisfies a particular user's requirement.

Features

- are wider that **units**.
- End-user can use features.

Feature testing

- software testing method.
- verify the behavior of a specific feature.

- **Examples:**
  - Data distribution check
  - Report preparation

Units vs. features: personal computer

Units:

- Each individual button
- Pixels on the screen
- LED diodes
- Blocks on the disk

Features:

- Keyboard
- Screen
- Illumination
- File system

Why do we use feature tests

Feature testing helps:

- To test things at the scope of the user interaction with a system.

The scope is wider than units:

- One unit does not work - does not mean the feature is NOT OK.

Vice versa:

- All units work as expected - does not mean the feature is OK.

Feature test example: setup

Setup and defining the feature code:

```
# Setup
import pandas as pd
import pytest

df = pd.read_csv('laptops.csv')

# Filter feature
def filter_data_by_manuf(df, manufacturer_name):
    filtered_df = df\
        [df["Manufacturer"] == manufacturer_name]
    return filtered_df
```

Feature test example: testing

Testing code:

```
# Feature test function
def test_unique():
    manuf_name = 'Apple'
    filtered = filter_data_by_manuf(df, manuf_name)
    assert filtered\
        ['Manufacturer'].nunique() == 1
    assert filtered\
        ['Manufacturer'].unique() == [manuf_name]
```

Summary

Feature testing - software testing method, allows to verify the behavior of a specific feature.

Features are wider that **units:**

- If a button is a unit, then the keyboard is a feature.

Feature testing helps to ensure that users get exactly what they expect.

To create feature tests one has to create test cases.

The key to design feature tests - to understand what the features are in a specific system.

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

Integration testing with pytest

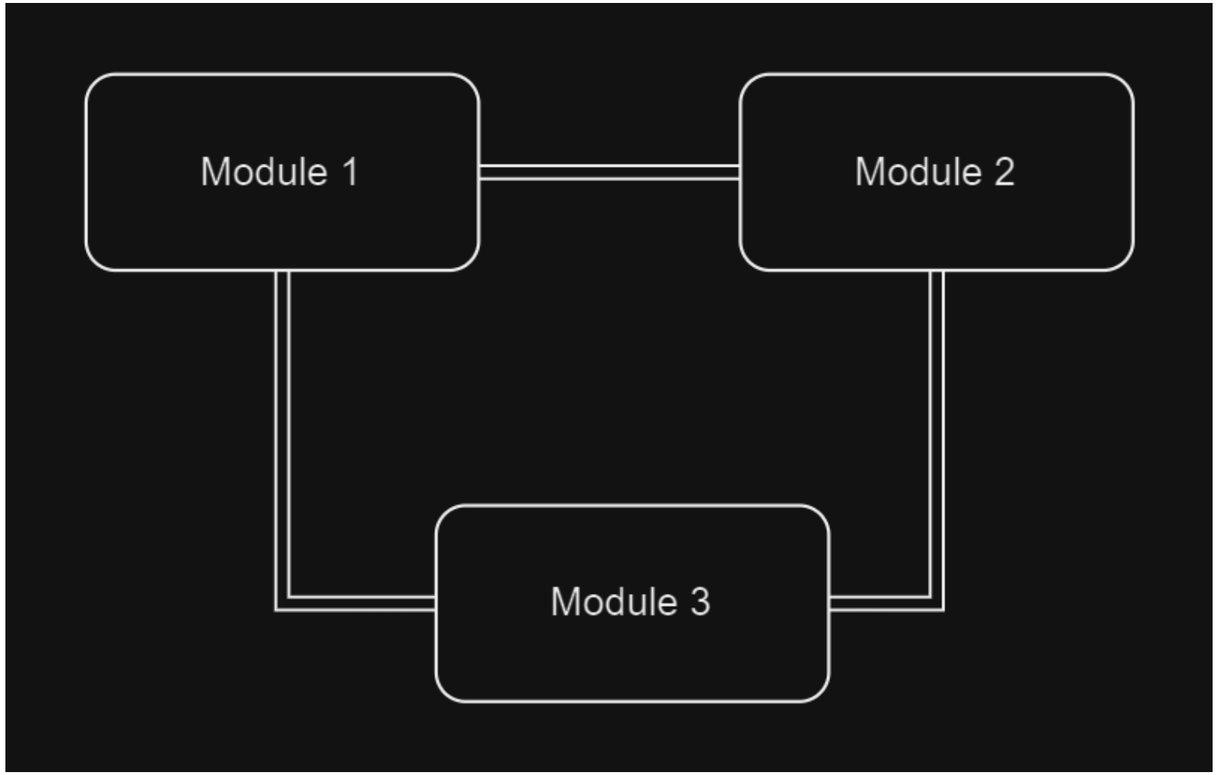
INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

What is integration testing?

- **Integration testing** - software testing method, that allows to verify that an interaction behaves normally.
- **Integration** - an interaction between 2 or more modules inside of a system.



Integrations in real-life projects

Examples:

- Power cable
- Internet connection
- File reading driver
- Database connection
- Application Programming Interface (API) integration

What can go wrong

Potential integration problems:

- Lost connection
- Loss of data
- Interaction delays
- Low bandwidth
- Version conflicts
- Interface mismatch
- Others

Example of integration testing

```
import pytest, os

@pytest.fixture
def setup_file():
    # Create temporary file
    file = "test_file.txt"
    with open(file, "w") as f1:
        f1.write("Test data 1")
    yield file
    os.remove(file)

def test_fs(setup_file):
    file = setup_file
    # Check that the file was created successfully
    assert os.path.exists(file)
```

Summary

- **Integration testing** - software testing method; allows to verify that an integration works as expected.
- **Real-life projects** include a lot of different integrations in them.
- Integration testing **helps to prevent** lots of potential problems.
- **Example**: checking integration between Python and a file system.

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

Performance testing with pytest

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

What is performance testing

Performance - how efficiently does software utilizes the resources of the system to accomplish a task.

Performance Testing - is a type of testing that measures software performance.

When performance testing is important

Resources:

- Execution Time
- CPU
- RAM
- Other resources

Cases:

- Website speed optimization
- App receiving millions of requests
- Path planning for a robot vacuum

Benchmark fixture

Installation:

```
pip install pytest-benchmark
```

```
# Example_1.py
import time
def test_func(benchmark):
    benchmark(time.sleep, 1)
```

CLI Command:

```
pytest Example_1.py
```

Benchmarking results

The results we get after we execute the CLI Command:

```
------------------------------------ benchmark: 1 tests -------------------------------------
Name (time in s)        Min     Max    Mean  StdDev  Median     IQR  Outliers     OPS  Rounds  Iterations
---------------------------------------------------------------------------------------------
test_func            1.0003  1.0151  1.0088  0.0067  1.0125  0.0115       1;0  0.9913       5           1
---------------------------------------------------------------------------------------------
```

For `time.sleep(3)` instead of `time.sleep(1)`:

```
------------------------------------ benchmark: 1 tests -------------------------------------
Name (time in s)        Min     Max    Mean  StdDev  Median     IQR  Outliers     OPS  Rounds  Iterations
---------------------------------------------------------------------------------------------
test_func            3.0005  3.0041  3.0026  0.0017  3.0036  0.0029       1;0  0.3330       5           1
---------------------------------------------------------------------------------------------
```

Benchmark decorator

```
# Example_2.py
import time
def test_func(benchmark):
    @benchmark
    def sleep_for_1_sec():
        time.sleep(1)
```

CLI Command:

```
pytest Example_2.py
```

Summary

- Performance testing - a type of testing that measures the software performance.
- Resources are usually finite.
- Helpful when resources are constrained.
- We can use `pytest-benchmark` fixture by:
  - calling `benchmark` directly
  - using `@benchmark` as a decorator
- The results describe the sample of measured runs in seconds.

Let's practice!

INTRODUCTION TO TESTING IN PYTHON