Meeting the Unittest

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

Recap of OOP

- **OOP** - programming paradigm based on objects and classes.
- **Class** - a template of an object that can contain methods and attributes.
- **Method** - a function or procedure that belongs to a class.
- **Attribute** - a variable that belongs to a class.
- **Object** - an instance of a class.

- **Example of a Python class:**

```
class Rectangle:
  # Constructor of Rectangle
  def __init__(self, a, b):
      self.a = a
      self.b = b
  # Area method
  def get_area(self):
      return self.a * self.b
# Usage example
r = Rectangle(4, 5)
print(r.get_area())
```

```
>> 20
```

OOP Inheritance

- Classes can inherit properties from other classes.
- Put the **parent** class in the brackets after the name of the new class.

```
class RedRectangle(Rectangle):
  self.color = 'red'
```

What is unittest

- `unittest` - built-in Python framework for test automation (it is installed with `Python`).
- `unittest` - **not only** for unit tests alone.
- **Based on OOP**: each test case is a class, and each test is a method.
- **Test case** - is an instance of testing.
- **Test suite** - is a collection of test cases.

unittest vs. pytest

unittest

- OOP-based - requires to create test classes)
- Built-in (is installed with the `Python` distribution)
- More assertion methods

pytest

- Functon-based - searches for scripts and functions starting with `test_`
- Third-party package (has to be installed separately from the `Python` distribution)
- Less assertion methods

How to create a test with unittest

Test of the exponentiation operator:

```
import unittest


# Declaring the TestCase class
class TestSquared(unittest.TestCase):
    # Defining the test
    def test_negative(self):
        self.assertEqual((-3) ** 2, 9)
```

Assertion methods

- `.assertEqual()`, `.assertNotEqual()`
- `.assertTrue()`, `.assertFalse()`
- `.assertIs()`, `.assertIsNone()`
- `.assertIsInstance()`, `.assertIn()`
- `.assertRaises()`
- Many others

Summary

- `unittest` - OOP-based built-in Python framework for test automation
- **Test case** - is a testing instance in `unittest`
- **To create a test:**
  1. Declare a class inheriting from `unittest.TestCase`
  2. Define test functions
- Assertion methods

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

CLI Interface

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

Example: code

Test of the exponentiation operator:

```
# test_sqneg.py
import unittest
# Declaring the TestCase class
class TestSquared(unittest.TestCase):
    # Defining the test
    def test_negative(self):
        self.assertEqual((-3) ** 2, 9)
```

CLI command:

```
python3 -m unittest test_sqneg.py
```

Run Python script `test_sqneg.py` using module `unittest`.

Example: output

The command: `python3 -m unittest test_sqneg.py`

The test output:

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Keyword argument -k

`unittest -k` - run test methods and classes that match the pattern or substring

Command: `python3 -m unittest -k "SomeStringOrPattern" test_script.py`

Example: `python3 -m unittest -k "Squared" test_sqneg.py`

Output:

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

[1] https://docs.python.org/3/library/unittest.html

Fail fast flag -f

`unittest -f` - stop the test run on the first error or failure.

Command: `python3 -m unittest -f test_script.py`

Use case example: when all of tests are crucial, like testing the airplane before a flight.

```
======================================================================
FAIL: test_negative (test_sqneg.TestSquared)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/alexander/OneDrive/Work/DataCamp/materials/ch4/42_CLI/slides/test_sqneg.py", line 5, in test_negative
    self.assertEqual((-3) ** 2, 10)
AssertionError: 9 != 10

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

[1] https://docs.python.org/3/library/unittest.html

Catch flag -c

Catch flag `unittest -c` - lets to interrupt the test by pushing "**Ctrl - C**".

- If "**Ctrl - C**"
  - is pushed once, `unittest` waits for the current test to end and reports all the results so far.
  - is pushed twice, `unittest` raises the `KeyboardInterrupt` exception.

Command: `python3 -m unittest -c test_script.py`

Use case example: when debugging a big test suite

[1] https://docs.python.org/3/library/unittest.html

Verbose flag -v

`unittest -v` - run tests with more detail

Command: `python3 -m unittest -v test_script.py` .

Use case example: debugging purposes

Output example:

```
> python3 -m unittest -v test_sqneg.py
test_negative (test_sqneg.TestSquared) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

[1] https://docs.python.org/3/library/unittest.html

Summary

- Basic command without arguments `python3 -m unittest test_script.py`
- Output in unittest
- Keyword argument: `python3 -m unittest -k "SomeStringOrPattern" test_script.py`
- Fail fast flag: `python3 -m unittest -f test_script.py`
- Catch flag: `python3 -m unittest -c test_script.py`
- Verbose flag: `python3 -m unittest -v test_script.py`

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

Fixtures in unittest

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

Fixtures recap

- **Fixture**
  - a prepared environment for a test
  - separate the preparation from the test code
- **Fixture setup** - setting up resources for tests
- **Fixture teardown** - cleaning up ("tearing down") resources that were allocated
- **Example:** preparing the food for a picnic and the cleaning at the end

Fixtures in the unittest library

- **Fixture in unittest** - the preparaton needed to perform one or more tests
- `.setUp()` - a method called to prepare the test fixture before the actual test
- `.tearDown()` - a method called after the test method to clean the environment

[1] https://docs.python.org/3/library/unittest.html

Example code

```
import unittest

class TestLi(unittest.TestCase):
    # Fixture setup method
    def setUp(self):
        self.li = [i for i in range(100)]

    # Fixture teardown method
    def tearDown(self):
        self.li.clear()

    # Test method
    def test_your_list(self):
        self.assertIn(99, self.li)
        self.assertNotIn(100, self.li)
```

Capital U and capital D

- The correct syntax: `setUp` with capital **U** and `tearDown` with capital **D**.

```
class TestLi(unittest.TestCase):
  # Fixture setup method
  def setUp(self):
      self.li = [i for i in range(100)]

  # Fixture teardown method
  def tearDown(self):
      self.li.clear()
```

Example output

The command: `python3 -m unittest test_in_list.py`

Output of a run with a `.setUp()` and `.tearDown()`:

```
> python3 -m unittest test_in_list.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Incorrectly named methods

Output of a run with a `.set_up()` :

```
======================================================================
ERROR: test_your_list (test_in_list.TestLi)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/alexander/OneDrive/Work/DataCamp/materials/ch4/43_fixtures/slides/test_in_list.py", line 14, in
 test_your_list
    self.assertIn(99, self.li)
AttributeError: 'TestLi' object has no attribute 'li'

======================================================================
ERROR: test_your_list (test_in_list.TestLi)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/alexander/OneDrive/Work/DataCamp/materials/ch4/43_fixtures/slides/test_in_list.py", line 10, in
 tearDown
    self.li.clear()
AttributeError: 'TestLi' object has no attribute 'li'

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (errors=2)
```

Summary

- **Fixture in unittest** - the preparation needed to perform one or more tests
- To create a fixture:
  - Implement the `.setUp()` method
  - Implement the `.tearDown()` method
- `.setUp()` - a method called to prepare the test fixture before the actual test.
- `.tearDown()` - a method called after the test method to clean the environment.

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

Practical examples

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

Data and pipeline

Data: salaries in data science.

Each row contains information about a data science worker with his salary, title and other attributes.

	A	B	C	D	E	F
1	work_year	experience_level	employment_type	job_title	salary	salary_currenc
2	2023	SE	FT	Principal Data Scientist	80000	EUR
3	2023	MI	CT	ML Engineer	30000	USD
4	2023	MI	CT	ML Engineer	25500	USD
5	2023	SE	FT	Data Scientist	175000	USD
6	2023	SE	FT	Data Scientist	120000	USD
7	2023	SE	FT	Applied Scientist	222200	USD
8	2023	SE	FT	Applied Scientist	136000	USD
9	2023	SE	FT	Data Scientist	219000	USD
10	2023	SE	FT	Data Scientist	141000	USD
11	2023	SE	FT	Data Scientist	147100	USD

Pipeline: to get the mean salary:

1. Read the data
2. Filter by employment type
3. Get the mean salary
4. Save the results

Code of the pipeline

```
import pandas as pd


# Fixture to get the data
@pytest.fixture
def read_df():
    return pd.read_csv('ds_salaries.csv')
# Function to filter the data
def filter_df(df):
    return df[df['employment_type'] == 'FT']
# Function to get the mean
def get_mean(df):
    return df['salary_in_usd'].mean()
```

Integration tests

Test cases:

- Reading the data
- Writing to the file

Code:

```
def test_read_df(read_df):
    # Check the type of the dataframe
    assert isinstance(read_df, pd.DataFrame)
    # Check that df contains rows
    assert read_df.shape[0] > 0
```

Integration tests

Example of checking that Python can create files.

```
def test_write():
    # Opening a file in writing mode
    with open('temp.txt', 'w') as wfile:
        # Writing the text to the file
        wfile.write('Testing stuff is awesome')
    # Checking the file exists
    assert os.path.exists('temp.txt')
    # Don't forget to clean after yourself
    os.remove('temp.txt')
```

Unit tests

Test cases:

- Filtered dataset contains only 'FT' employment type
- The `get_mean()` function returns a number

Code:

```
def test_units(read_df):
    filtered = filter_df(read_df)
    assert filtered['employment_type'].unique() == ['FT']
    assert isinstance(get_mean(filtered), float)
```

Feature tests

Test cases:

- The mean is greater than zero
- The mean is not bigger than the maximum salary in the dataset

Code:

```
def test_feature(read_df):
    # Filtering the data
    filtered = filter_df(read_df)
    # Test case: mean is greater than zero
    assert get_mean(filtered) > 0
    # Test case: mean is not bigger than the maximum
    assert get_mean(filtered) <= read_df['salary_in_usd'].max()
```

Performance tests

Test cases:

- Pipeline execution time from the start to the end

Code:

```
def test_performance(benchmark, read_df):
    # Benchmark decorator
    @benchmark
    # Function to measure
    def get_result():
        filtered = filter_df(read_df)
        return get_mean(filtered)
```

Final test suite

```
import pytest

## Integration Tests
def test_read_df(read_df):
      # Check the type of the dataframe
    assert isinstance(read_df, pd.DataFrame)
    # Check that df contains rows
    assert read_df.shape[0] > 0
def test_write():
    with open('temp.txt', 'w') as wfile:
        wfile.write('12345')
    assert os.path.exists('temp.txt')
    os.remove('temp.txt')

## Unit Tests
def test_units(read_df):
    filtered = filter_df(read_df)
    assert filtered['employment_type'].unique() == ['FT']
    assert isinstance(get_mean(filtered), float)
```

```
## Feature Tests
def test_feature(read_df):
    # Filtering the data
    filtered = filter_df(read_df)
    # Test case: mean is greater than zero
    assert get_mean(filtered) > 0
    # Test case: mean is not bigger than the maximum
    assert get_mean(filtered) <= read_df['salary_in_usd'].max()

## Performance Tests
def test_performance(benchmark, read_df):
    # Benchmark decorator
    @benchmark
    # Function to measure
    def pipeline():
        filtered = filter_df(read_df)
        return get_mean(filtered)
```

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

Congratulations!

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

Chapter 1 - Creating tests with pytest

- Testing and `pytest`
- CLI: `pytest test_script.py`
- Test markers

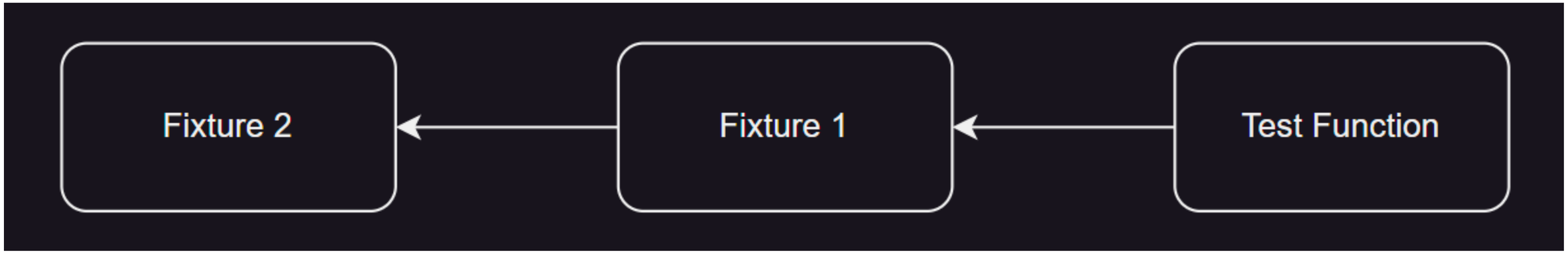
```
PS
============================================================ test session starts ============================================================
platform win32 -- Python 3.11.2, pytest-7.2.2, pluggy-1.0.0
benchmark: 4.0.0 (defaults: timer=time.perf_counter disable_gc=False min_rounds=5 min_time=0.000005 max_time=1.0 calibration_precision=10 w
armup=False warmup_iterations=100000)
rootdir:
plugins: benchmark-4.0.0
collected 3 items

slides.py ...                                                                                                                      [100%]

============================================================= 3 passed in 0.01s =============================================================
PS
```

Chapter 2 - Pytest fixtures

- Introduction to fixtures
- Chain fixtures requests



- Fixtures autouse
- Fixtures teardown

Chapter 3 - Basic Testing Types

- Unit testing
- Feature testing
- Integration testing
- Performance testing

Chapter 4 - Writing tests with unittest

- Meeting the unittest

```
import unittest
class TestSquared(unittest.TestCase):
  def test_negative(self):
      self.assertEqual((-3) ** 2, 9)
```

- Unittest CLI

```
> python3 -m unittest slides_sqneg.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

- Fixtures in unittest
- Practical examples

Congratulations!

INTRODUCTION TO TESTING IN PYTHON