Introduction to testing in Python

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

Why is testing so important?

Common problems:

- Bugs and errors
- Hardware failures
- Unexpected or unpredictable behavior

All of the problems might lead to significant **expenses** increase for the **fixing** purposes.

Testing helps to:

- Identify defects, bugs, and errors
- Reduce risks of software failures
- Enhance reliability, functionality, and performance of the software

What is testing?

- **Testing** - a process of evaluating a system or software
- We need testing to ensure it meets specified requirements
- **Test** - a procedure to verify the correctness of a software application or system

Course prerequisites

- Advanced Python programming
- `assert` statements
- Decorators
- OOP Concepts (classes, methods, inheritance)

Testing in real life

Think of airplanes:

- Visual inspection
- Electronics and mechanics check
- Fuel check
- Passengers check
- Weather check
- Permission to take off from air traffic controller

All of the above - are **tests**! And we need them for **safety.**

Assert in Python

- `assert condition` - lets to test if `condition` is `True`.
- If `condition` is `False`, Python will raise an `AssertionError`.

Testing with pytest - a simple example

`pytest` - a popular testing framework in Python, which provides a simple way to write tests.

Example of an "assert" test written with `pytest` in Python:

```
import pytest

# A function to test
def squared(number):
    return number * number

# A test function always starts with "test"
def test_squared():
    assert squared(-2) == squared(2)
```

Context managers recap

- **Context manager** - a Python object that is used by declaring a `with` statement
- We use **context managers** to set up and tear down temporary context

```
# Writing to file example
with open("hello_world.txt", 'w') as hello_file:
    hello_file.write("Hello world \n")
```

Meet the pytest: raises

`pytest.raises` - when you expect the test to raise an `Exception`

```
import pytest

# A function to test
def division(a, b):
    return a / b

# A test function
def test_raises():
    with pytest.raises(ZeroDivisionError):
        division(a=25, b=0)
```

Summary

Testing is:

- A process of evaluating, that software works as expected
- Present in everyday life
- Essential to tackle the challenges of the software development process
- Helps to ensure that the problems are addressed properly

Tests implementation:

- `pytest` - a powerful Python framework, that simplifies the testing process
- `assert` - a Python keyword, used in `pytest` for creating basic tests by validating a condition
- `pytest.raises` - a context manager, used to create a test that is expected to result in

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

Invoking pytest from CLI

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

Example CLI run: syntax

Command-Line Interface (CLI) - a user interface that allows to interact with a computer program by entering text commands into a terminal.

The command for running the `slides.py` from **CLI**:

```
pytest slides.py
```

Meaning: "Please, run the `pytest` framework using the tests from the `slides.py` module"

Example CLI run: output

Output of a test:

```
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
```

Example CLI run: output

Output of a test:

- **Modules versions**

```
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
```

Example CLI run: output

Output of a test:

- **Number of "collected" tests**

```
PS [redacted]
============================================================ 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: [redacted]
plugins: benchmark-4.0.0
collected 3 items

slides.py ...                                                                                                                      [100%]

============================================================ 3 passed in 0.01s =============================================================
PS [redacted]
```

Example CLI run: output

Output of a test:

- **Names of test scripts**

```
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
armup=False warmup_iterations=100000)
rootdir:
plugins: benchmark-4.0.0
collected 3 items

slides.py ...                                                                                                                      [100%]

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

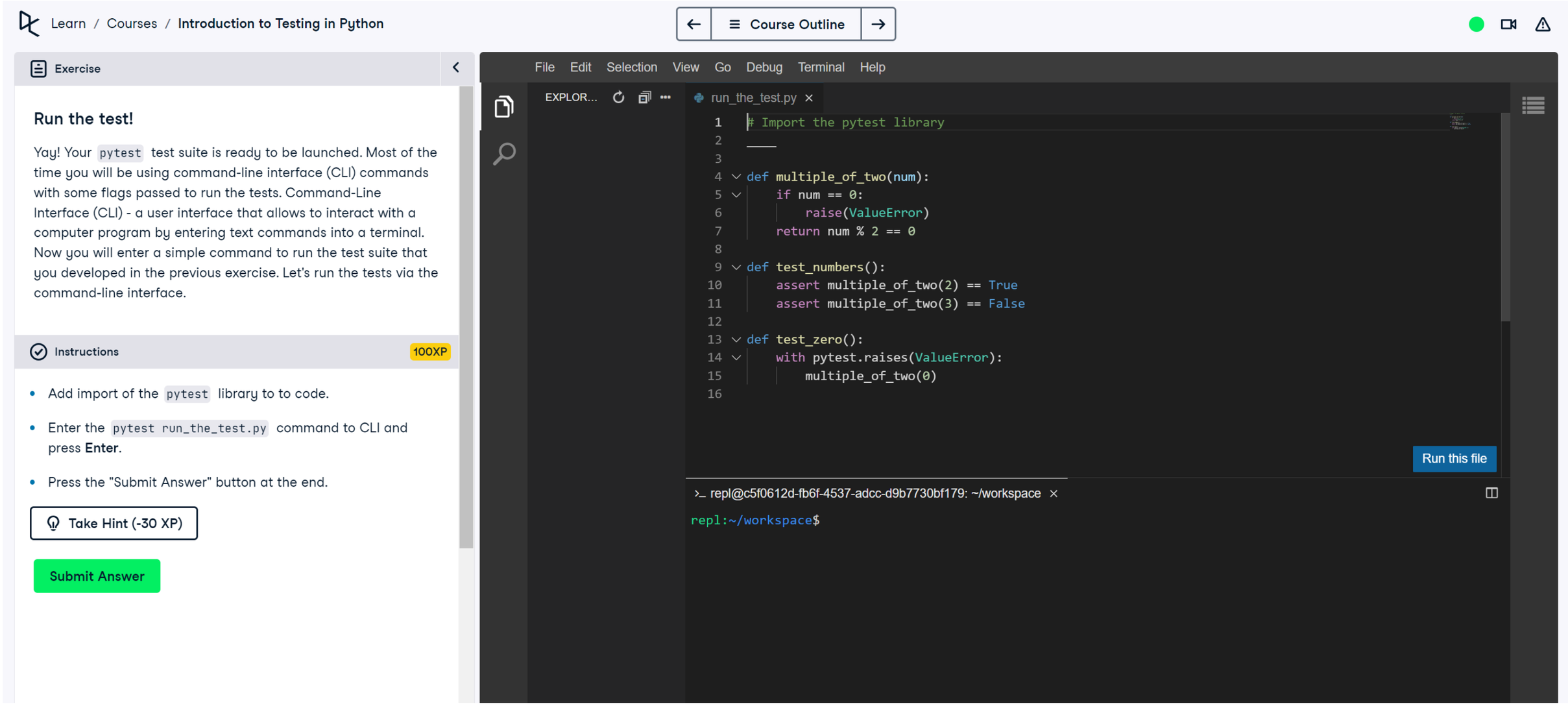
Example CLI run: output

Output of a test:

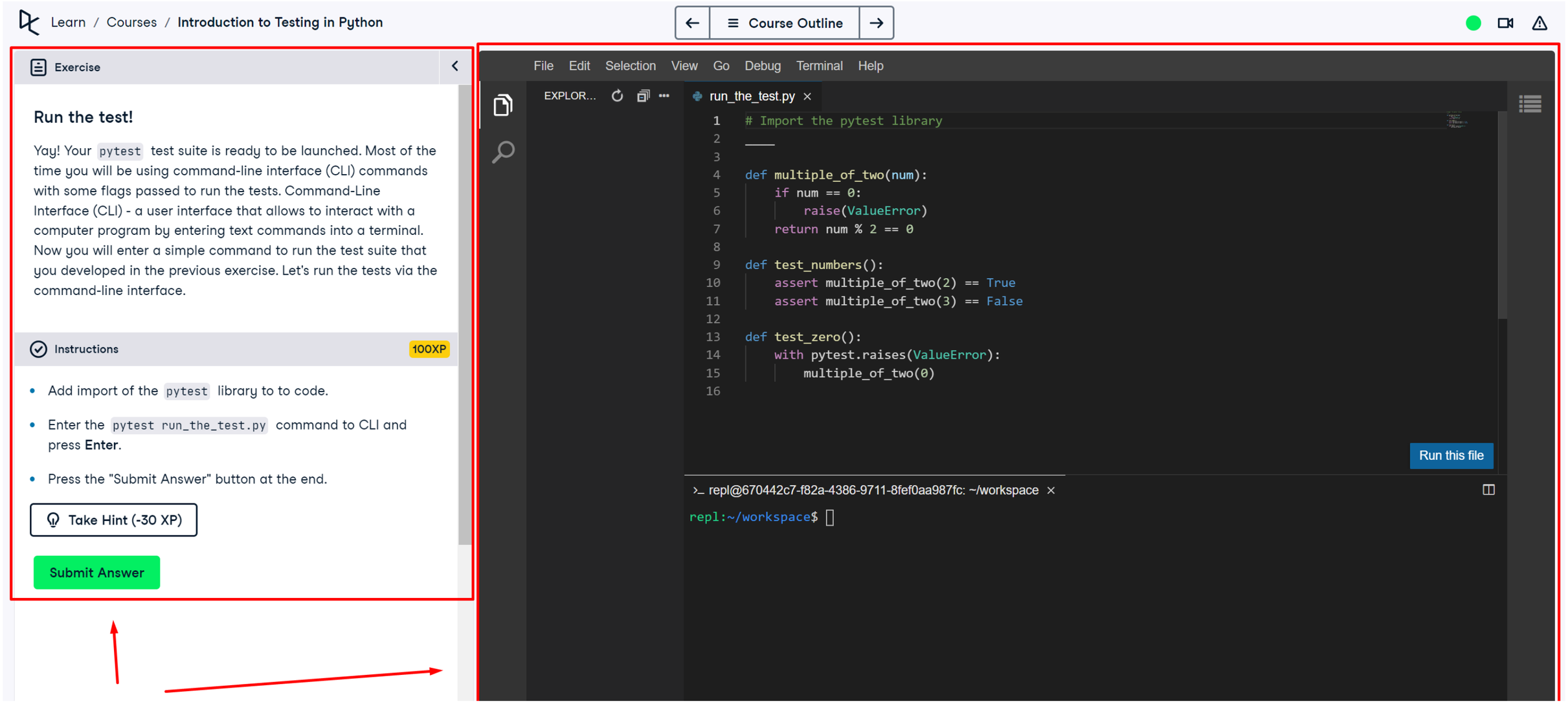
- **Test results**

```
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
```

IDE exercises



IDE exercises



Directory argument

The command for running all tests in `tests_dir/` :

```
pytest tests_dir/
```

Meaning: "Please, run the `pytest` framework using all found the tests from the `tests_dir` folder".

```
PS [redacted] pytest tests_directory
========================================================== 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 warm
p=False warmup_iterations=100000)
rootdir: [redacted]
plugins: benchmark-4.0.0
collected 6 items

tests_directory\test_slides_1.py ...                                                                                                  [ 50%]
tests_directory\test_slides_2.py ...                                                                                                  [100%]

========================================================== 6 passed in 0.01s ==========================================================
PS [redacted]
```

Keyword argument - filter tests by name

The command for running tests from `tests_ex.py` contains "squared":

```
pytest tests_ex.py -k "squared"
```

Meaning: "Please, run the `pytest` framework using all tests from the `tests_ex.py` script containing `squared` ".

Output:

```
PS [redacted] pytest .\keyword_tests.py -k "squared"
========================================================== 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 warm
p=False warmup_iterations=100000)
rootdir: [redacted]
plugins: benchmark-4.0.0
collected 3 items / 1 deselected / 2 selected

keyword_tests.py ..                                                                                                                [100%]

=================================================== 2 passed, 1 deselected in 0.01s ====================================================
```

Summary

- **IDE exercises** - let us to write code in an Integrated Development Environment and to use command-line interface (CLI)
- **CLI pytest command** starts with `pytest`
- **Sources of tests:**
  - One script, by passing `script_name.py`
  - A set of scripts from one folder, by passing `directory_name/`
- **Keyword argument:**
  - By passing `-k "keyword_expression"`
- **Output** of a test contains important information about the run

Let's practice!

INTRODUCTION TO TESTING IN PYTHON

Applying test markers

INTRODUCTION TO TESTING IN PYTHON

Alexander Levin
Data Scientist

datacamp

Overview of test markers

- **Use case 1:** skip the test if the condition met
- **Use case 2:** this test is expected to fail
- **Test marker** - a tag (a marker) of a test in the `pytest` library
- Allows to specify behavior for particular tests by tagging them (marking them)

Markers syntax

- **Decorator** - a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure
- **Test markers syntax** are started with `@pytest.mark` decorator:

```
import pytest

def get_length(string):
    return len(string)

# The test marker syntax
@pytest.mark.skip
def test_get_len():
    assert get_length('123') == 3
```

Skip and skipif markers

- Use `@pytest.mark.skip` - when you want a test to be skipped in any case
- Use `@pytest.mark.skipif` - if you want a test to be skipped if a given condition is `True`

Skip marker example

- Use `@pytest.mark.skip` - when you want a test to be skipped indefinitely.

```
import pytest
def get_length(string):
    return len(string)

# The skip marker example
@pytest.mark.skip
def test_get_len():
    assert get_length('123') == 3
```

Skip marker example: output

- Output with a skipped test:

```
PS [redacted] > pytest skip.py
========================================================== 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
armup=False warmup_iterations=100000)
rootdir: [redacted]
plugins: benchmark-4.0.0
collected 1 item

skip.py s                                                                                                                          [100%]

========================================================== 1 skipped in 0.01s ==========================================================
PS [redacted] >
```

Skipif marker example

- Use `@pytest.mark.skipif` - when you want a test to be skipped if the given condition is `True`.

```
import pytest
def get_length(string):
    return len(string)

# The skipif marker example
@pytest.mark.skipif('2 * 2 == 5')
def test_get_len():
    assert get_length('abc') == 3
```

Skipif marker example: output

- Output of a conditionally skipped test:

```
PS [redacted] > pytest .\skipif.py
========================================================== 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
armup=False warmup_iterations=100000)
rootdir: [redacted]
plugins: benchmark-4.0.0
collected 1 item

skipif.py .                                                                                                                    [100%]

========================================================== 1 passed in 0.02s ==========================================================
PS [redacted] >
```

Xfail marker

- Use `@pytest.mark.xfail` - when you expect a test to be failed

```
import pytest

def gen_sequence(n):
    return list(range(1, n+1))

# The xfail marker example
@pytest.mark.xfail
def test_gen_seq():
    assert gen_sequence(-1)
```

Xfail marker: output

- Output of a test, that is expected to fail:

```
========================================= 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 warmup=False warmup_iterations=100000)
rootdir: [redacted]
[redacted]
plugins: benchmark-4.0.0
collected 1 item

slides.py x                                                                          [100%]

========================================== 1 xfailed in 0.08s ==========================================
```

Summary

Test marker:

- Is an attribute of a test in the `pytest` library
- Is used to specify behavior for particular tests
- Has syntax started with `@pytest.mark.name_of_the_marker`
- Out-of-the-box implementations in pytest:
  - `@pytest.mark.xfail`
  - `@pytest.mark.skip`
  - `@pytest.mark.skipif`

Let's practice!

INTRODUCTION TO TESTING IN PYTHON