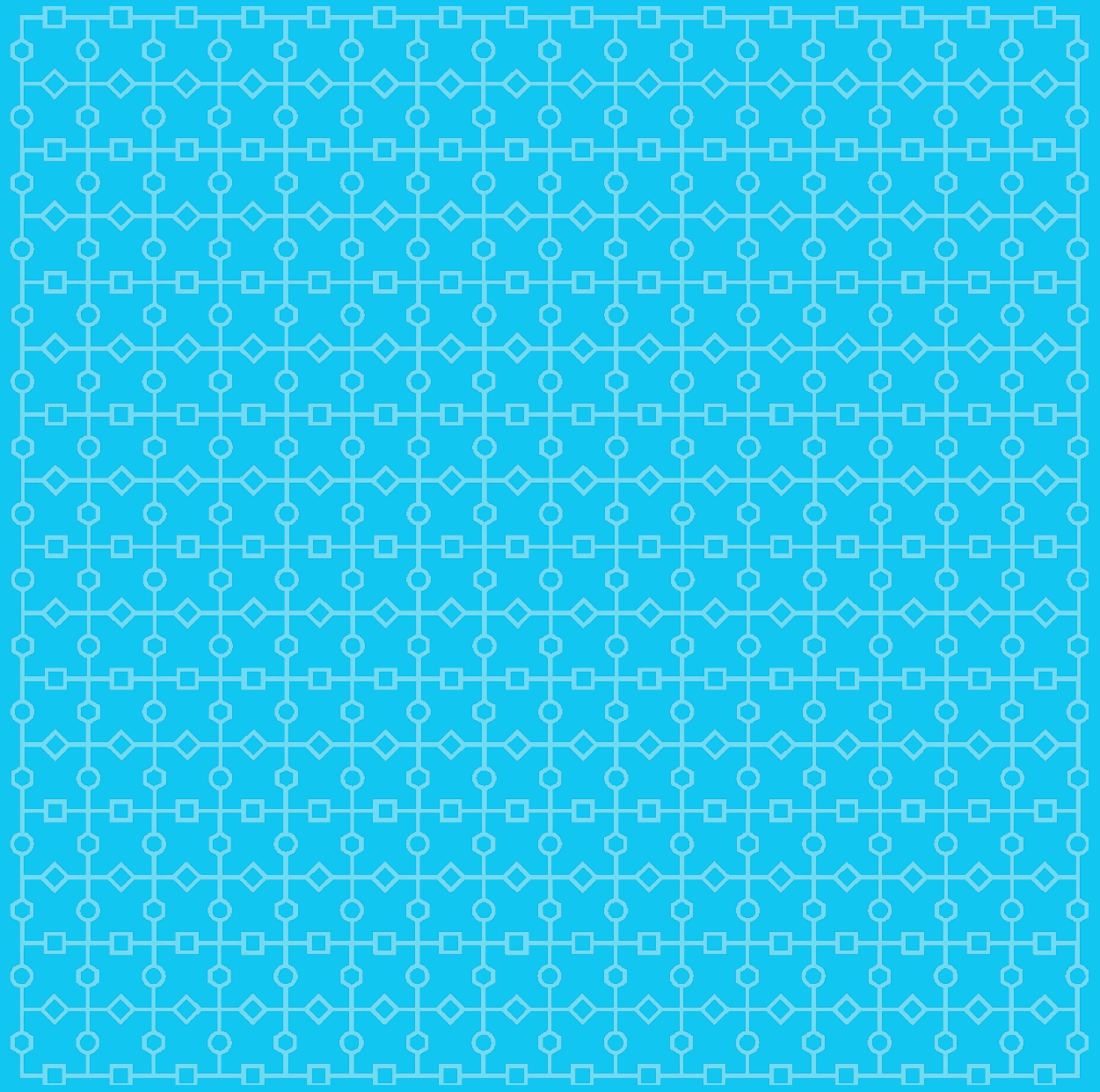
# Software Engineer
# Back End



TECH ACADEMY

# Test Driven Development (TDD) with Python

**Instructor:** Idris Shabanlı

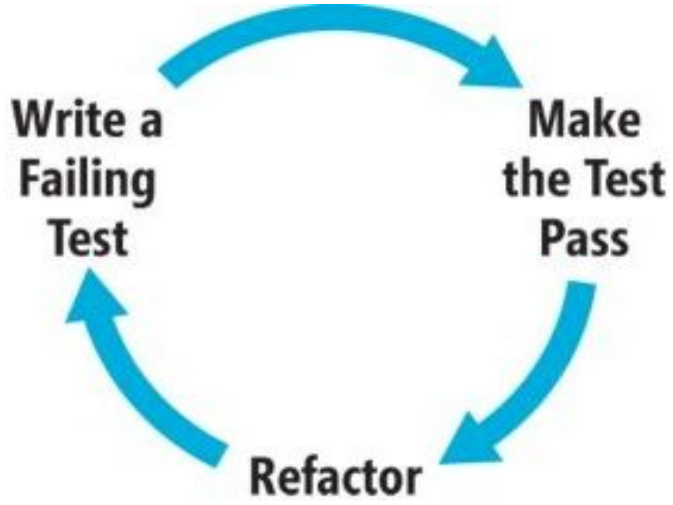
TECH ACADEMY

# Table of content

# What Is Test-Driven Development?

*This approach allows you to escape the trap that many developers fall into.*

TDD, in its most basic terms, is the process of implementing code by writing your tests first, seeing them fail, then writing the code to make the tests pass. You can then build upon this developed code by appropriately altering your test to expect the outcome of additional functionality, then writing the code to make it pass again.



# Automated vs. Manual Testing

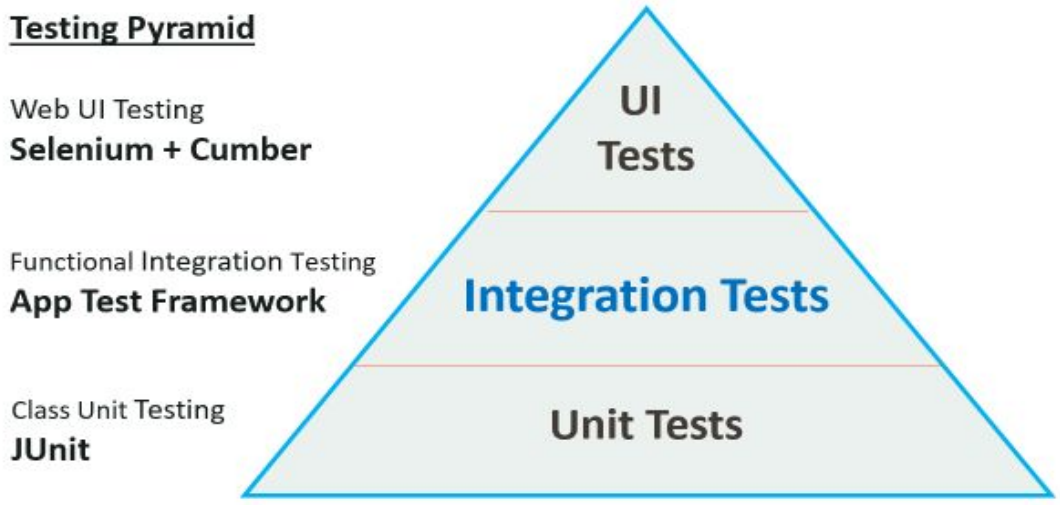
**Manual testing** is testing of the software where tests are executed manually by a QA Analyst. It is performed to discover bugs in software under development.

In **Automated Software Testing**, testers write code/test scripts to automate test execution. Testers use appropriate automation tools to develop the test scripts and validate the software. The goal is to complete test execution in a less amount of time.

# Unit Tests vs. Integration Tests

**Unit Tests** – It is a piece of a code that invokes another piece of code (unit) and checks if an output of that action is the same as the desired output.

**Integration Tests** – It is testing a unit without full control over all parties in the test. They are using one or more of its outside dependencies, such as a database, threads, network, time, etc.



# Choosing a Test Runner

There are many test runners available for Python. The one built into the Python standard library is called `unittest`. The principles of unittest are easily portable to other frameworks. The three most popular test runners are:

- unittest
- nose or nose2
- pytest

Choosing the best test runner for your requirements and level of experience is important.

# Unittest


unittest has been built into the Python standard library since version 2.1. You'll probably see it in commercial Python applications and open-source projects.

unittest contains both a testing framework and a test runner. unittest has some important requirements for writing and executing tests.

unittest requires that:

- You put your tests into classes as methods
- You use a series of special assertion methods in the unittest.TestCase class instead of the built-in assert statement

# How to Write Assertions

The last step of writing a test is to validate the output against a known response. This is known as an **assertion**. There are some general best practices around how to write assertions:

- Make sure tests are repeatable and run your test multiple times to make sure it gives the same result every time
- Try and assert results that relate to your input data, such as checking that the result is the actual sum of values in the sum() function

# How to Write Assertions

unittest comes with lots of methods to assert on the values, types, and existence of variables. Here are some of the most commonly used methods:

| Method | Equivalent to |
|---|---|
| `.assertEqual(a, b)` | `a == b` |
| `.assertTrue(x)` | `bool(x) is True` |
| `.assertFalse(x)` | `bool(x) is False` |
| `.assertIs(a, b)` | `a is b` |
| `.assertIsNone(x)` | `x is None` |
| `.assertIn(a, b)` | `a in b` |
| `.assertIsInstance(a, b)` | `isinstance(a, b)` |

# Basic example

- Import unittest from the standard library
- Create a class called TestSum that inherits from the TestCase class
- Convert the test functions into methods by adding self as the first argument
- Change the assertions to use the self.assertEqual() method on the TestCase class
- Change the command-line entry point to call unittest.main()

Python

```python
import unittest


class TestSum(unittest.TestCase):

    def test_sum(self):
        self.assertEqual(sum([1, 2, 3]), 6, "Should be 6")

    def test_sum_tuple(self):
        self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")

if __name__ == '__main__':
    unittest.main()
```

# Basic example



If you execute this at the command line, you'll see one `success` (indicated with .) and one `failure` (indicated with F):

Shell

```
$ python test_sum_unittest.py
.F
======================================================================
FAIL: test_sum_tuple (__main__.TestSum)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_sum_unittest.py", line 9, in test_sum_tuple
    self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")
AssertionError: Should be 6

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

# Resources

- A simple introduction to Test Driven Development with Python
- Test Driven Development (TDD) with Python
- Automation Testing Vs. Manual Testing: What's the Difference?
- Manual Testing vs. Automated Testing
- Getting Started With Testing in Python

# Please take notes for questions


# THANK YOU!