**Table of Contents**

[**Introduction 1**](#_gjdgxs)

[Lab Overview 1](#_1fob9te)

[Lab Objectives 1](#_2et92p0)

[Lab Prerequisites 2](#_tyjcwt)

[**Instructions 2**](#_3dy6vkm)

[Lab Features / Functions 2](#_1t3h5sf)

[Lab Tutorial 2](#_4d34og8)

[Lab Challenge 4](#_2s8eyo1)

[**Expected Results 4**](#_17dp8vu)

[Lab Examples 4](#_3rdcrjn)

[Lab Takeaway 4](#_26in1rg)

[**Additional Resources 4**](#_lnxbz9)

# Introduction

Test-driven Development Testing using Pytest/Unittest

## Lab Overview

Test-driven development (TDD) is a software development practice focused on writing tests before code. It ensures that code is modular, purposeful, and maintainable. This lab will follow the process of writing a failing test, using the minimum code to make the test pass, and then refactoring the code. You will work with the Pytest framework and create Mock unit tests using Python’s built-in unit test framework. TDD ensures you understand how your program should work before coding it.

## Lab Objectives

* Gain proficiency in using Python’s testing framework, Pytest.
* Understand how to write testable, well-designed code before production code.
* Learn how to use Pytest for test-driven development.
* Develop Mock unit tests for testing the necessary functionalities of a program.
* Understand test-driven development’s purpose and best practices.
* Recognize how to use the Typing library for TDD purposes.

## Lab Prerequisites

* Basic knowledge of Python programming.
* Understanding of how to install Python libraries using pip.
* Installation of Pytest (pip install pytest).
* Familiarity with the basics of Pytest and unit testing.

# Instructions

## Lab Features / Functions

The lab contains six Python files. The zipped file contains an example program to guide you in creating a TDD program. The files will serve as templates for your program with TDD best practices to guide you through. You should have a simple, working program similar to the example provided by the end.

## Lab Tutorial

**Assignment Setup:**

1. Download/unzip the code base to your respective IDE.
2. Set up a virtual environment (recommended) for your testing environment.
3. Install the required dependencies using requirements.txt or install them manually using pip.

**Step 1: Import the Pytest Library**

First, make sure you have Pytest installed:

Using the command line, enter the following to install the necessary Python modules:

* bash
* pip install pytest

Or, install the Python modules directly from the requirements.txt file:

* pip install -r requirements.txt

Lastly, ensure Pytest is installed successfully by entering into the IDE terminal/CLI:

* Pytest --version

**Step 2: Review the TDD example**

Begin by unzipping the sample\_tdd file folder. Inside the folder is a sample program created using TDD, complete with individual files used for production code, test code, and mock tests.

Recall that the production code is responsible for a particular process, whereas the test code outlines the necessary functions that require implementation. In TDD, test code is written before production code, acting as a guiding hand in the design process. The production code serves to pass the tests by the simplest means.

Once you’ve finished reviewing the example program, open the tdd\_lab and test\_tdd\_lab files. The files directly reflect the process used for creating the sample\_tdd. Following through the files, you will practice the TDD cycle by creating failing tests and writing the minimum amount of code to make the tests pass. Lastly is the tdd\_template file, where you will begin designing your program using step-by-step TDD processes. You may consider using the tdd\_lab and test\_tdd\_lab files in developing/testing your program.

**Step 3: Begin designing a program under TDD**

You can reference the tdd\_template file, tdd\_lab, and test\_tdd\_lab files to use when developing.

The recommended method would be to use the test\_tdd\_lab for writing tests and the tdd\_template file for writing production code.

**Step 4: Reference the TDD example**

If you need to, you can continually reference the TDD sample. The program you create will, of course, conventionally differ. But, you will similarly need to implement mock tests, Pytest fixtures, and the typing library when designing your program.

Locate additional resources at the end of this document. There is official Python documentation for mock unit tests, Pytest, Pytest fixtures, and the typing library.

**Step 5: Ensure your TDD program is complete**

After you have completed your program, all the tests you’ve created should pass.

Remember to check whether you have successfully implemented the necessary tests, code, and libraries/frameworks into your program.

## Lab Challenge

To solidify your understanding of Test-driven development, check that your program contains at least one instance of the following:

* Mock values used for testing.
* Pytest fixtures.
* Typing library assigned variable types.

Lastly, consider adding additional tests for functions not yet implemented or altering the existing tests to achieve different results. Although the base program is simple, you can expand or change the design to fulfill another purpose.

# Expected Results

## Lab Examples

The program you create following this lab will differ from the example program provided in this lab. However, you will end with a program that follows a similar structure.

Although you may consider not using a TDD framework for testing/development (It isn’t always the best approach). TDD has numerous applications, and you should know when TDD is appropriate. Consider implementing TDD when you continually add new functionalities to a program or need easy-to-extend software. You can reference this lab when following TDD cycles in creating programs.

## Lab Takeaway

Upon completing this lab, you should be confident in implementing test-driven development strategies. You will understand how to use Python’s Pytest and Unittest libraries under the TDD cycle. Your understanding of TDD practices applies to other programming languages and paradigms.

# Additional Resources

Pytest:

<https://docs.pytest.org/en/7.4.x/>

<https://docs.pytest.org/en/7.1.x/how-to/fixtures.html>

Unittest:

<https://docs.python.org/3/library/unittest.html>

<https://docs.python.org/3/library/unittest.mock.html>

Test-driven Development:

<https://www.agilealliance.org/glossary/tdd/>

Typing Library:

<https://docs.python.org/3/library/typing.html>