**DevOps Assignment**

1. **Source Code**

# todo.py

tasks = []

def add\_task(task\_description):

    task = {"description": task\_description, "done": False}

    tasks.append(task)

    print(f"✅ Task added: {task\_description}")

def view\_tasks():

    if not tasks:

        print("📭 No tasks in the list.")

        return

    print("\n📋 Your To-Do List:")

    for index, task in enumerate(tasks, start=1):

        status = "✅" if task["done"] else "❌"

        print(f"{index}. {task['description']} [{status}]")

def mark\_done(task\_number):

    if 1 <= task\_number <= len(tasks):

        tasks[task\_number - 1]["done"] = True

        print(f"✅ Task {task\_number} marked as done.")

    else:

        print("❌ Invalid task number!")

def remove\_task(task\_number):

    if 1 <= task\_number <= len(tasks):

        removed = tasks.pop(task\_number - 1)

        print(f"🗑️ Removed task: {removed['description']}")

    else:

        print("❌ Invalid task number!")

def main():

    while True:

        print("\nWhat would you like to do?")

        print("1. Add a task")

        print("2. View tasks")

        print("3. Mark task as done")

        print("4. Remove a task")

        print("5. Exit")

       choice = input("Enter your choice (1-5): ")

      if choice == "1":

            description = input("Enter task description: ")

            add\_task(description)

        elif choice == "2":

            view\_tasks()

        elif choice == "3":

            try:

                task\_num = int(input("Enter task number to mark as done: "))

                mark\_done(task\_num)

            except ValueError:

                print("❌ Please enter a valid number!")

        elif choice == "4":

            try:

                task\_num = int(input("Enter task number to remove: "))

                remove\_task(task\_num)

            except ValueError:

                print("❌ Please enter a valid number!")

        elif choice == "5":

            print("👋 Bye! Your tasks were amazing ✨")

            break

        else:

            print("❌ Invalid choice. Try again.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. **Initial test cases**

# test\_todo.py

import pytest

from todo import add\_task, mark\_done, remove\_task, view\_tasks, tasks

def setup\_function():

tasks.clear()

def test\_add\_task():

add\_task("Finish homework")

assert len(tasks) == 1

assert tasks[0]["description"] == "Finish homework"

assert not tasks[0]["done"]

def test\_mark\_done\_valid():

add\_task("Workout")

mark\_done(1)

assert tasks[0]["done"]

def test\_mark\_done\_invalid\_low():

add\_task("Drink water")

mark\_done(0)

assert not tasks[0]["done"]

def test\_mark\_done\_invalid\_high():

add\_task("Call friend")

mark\_done(5)

assert not tasks[0]["done"]

def test\_remove\_task\_valid():

add\_task("Read book")

remove\_task(1)

assert len(tasks) == 0

def test\_remove\_task\_invalid\_low():

add\_task("Clean desk")

remove\_task(0)

assert len(tasks) == 1

def test\_remove\_task\_invalid\_high():

add\_task("Organize files")

remove\_task(5)

assert len(tasks) == 1

def test\_view\_tasks\_output(capsys):

add\_task("Bake cake")

view\_tasks()

captured = capsys.readouterr()

assert "Bake cake" in captured.out

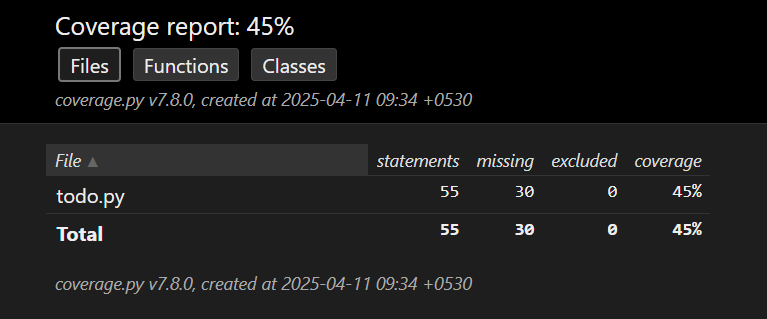
def test\_view\_tasks\_empty(capsys):

view\_tasks()

captured = capsys.readouterr()

assert "No tasks" in captured.out

1. **Initial Code Coverage**



1. **After improvement Code**

# test\_todo.py

import pytest

from unittest.mock import patch

from todo import add\_task, mark\_done, remove\_task, view\_tasks, tasks, main

def setup\_function():

    tasks.clear()

def test\_add\_task():

    add\_task("Finish homework")

    assert len(tasks) == 1

    assert tasks[0]["description"] == "Finish homework"

    assert not tasks[0]["done"]

def test\_mark\_done\_valid():

    add\_task("Workout")

    mark\_done(1)

    assert tasks[0]["done"]

def test\_mark\_done\_invalid\_low():

    add\_task("Drink water")

    mark\_done(0)

    assert not tasks[0]["done"]

def test\_mark\_done\_invalid\_high():

    add\_task("Call friend")

    mark\_done(5)

    assert not tasks[0]["done"]

def test\_remove\_task\_valid():

    add\_task("Read book")

    remove\_task(1)

    assert len(tasks) == 0

def test\_remove\_task\_invalid\_low():

    add\_task("Clean desk")

    remove\_task(0)

    assert len(tasks) == 1

def test\_remove\_task\_invalid\_high():

    add\_task("Organize files")

    remove\_task(5)

    assert len(tasks) == 1

def test\_view\_tasks\_output(capsys):

    add\_task("Bake cake")

    view\_tasks()

    captured = capsys.readouterr()

    assert "Bake cake" in captured.out

def test\_view\_tasks\_empty(capsys):

    view\_tasks()

    captured = capsys.readouterr()

    assert "No tasks" in captured.out

def test\_main\_add\_and\_exit():

    with patch("builtins.input", side\_effect=["1", "Main Task", "5"]):

        main()

    assert len(tasks) == 1

    assert tasks[0]["description"] == "Main Task"

def test\_main\_view\_then\_exit(capsys):

    add\_task("View from main")

    with patch("builtins.input", side\_effect=["2", "5"]):

        main()

        captured = capsys.readouterr()

        assert "View from main" in captured.out

def test\_main\_mark\_done\_valid():

    add\_task("Mark via main")

    with patch("builtins.input", side\_effect=["3", "1", "5"]):

        main()

    assert tasks[0]["done"]

def test\_main\_mark\_done\_invalid(capsys):

    with patch("builtins.input", side\_effect=["3", "100", "5"]):

        add\_task("Invalid mark")

        main()

        captured = capsys.readouterr()

        assert "Invalid task number" in captured.out

def test\_main\_remove\_valid():

    add\_task("Remove me")

    with patch("builtins.input", side\_effect=["4", "1", "5"]):

        main()

    assert len(tasks) == 0

def test\_main\_remove\_invalid(capsys):

    with patch("builtins.input", side\_effect=["4", "99", "5"]):

        add\_task("Keep me")

        main()

        captured = capsys.readouterr()

        assert "Invalid task number" in captured.out

def test\_main\_invalid\_option(capsys):

    with patch("builtins.input", side\_effect=["9", "5"]):

        main()

        captured = capsys.readouterr()

        assert "Invalid choice" in captured.out

def test\_main\_invalid\_number\_input(capsys):

    add\_task("Task 1")

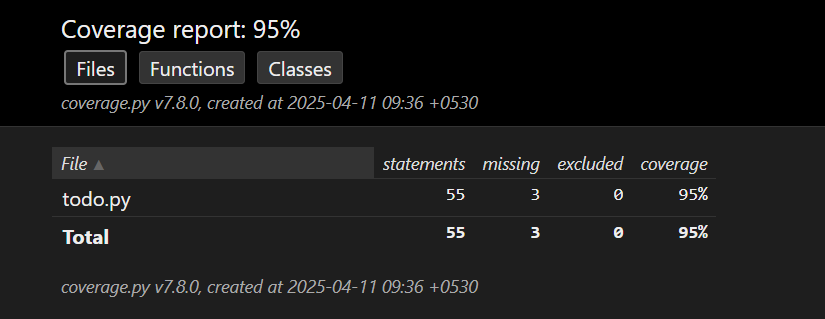
    with patch("builtins.input", side\_effect=["3", "abc", "5"]):

        main()

        captured = capsys.readouterr()

        assert "Please enter a valid number" in captured.out

1. **After Improvement Code coverage Report**



**1) Which functions did you choose to test and why?**

add\_task(description)  
This function was selected because it handles the primary functionality of adding a new task to the list. It was important to verify that the task is correctly added with the appropriate description and that it is initially marked as not done.

mark\_done(task\_number)  
This function was tested to ensure that it correctly marks a task as completed when given a valid task number. It is a critical part of task management, so verifying its correctness was essential.

remove\_task(task\_number)  
I tested this function to confirm that it correctly removes a task when given a valid index. Since it alters the list of tasks, it was important to ensure it behaves as expected and does not throw errors on valid input.

These functions were chosen initially because they are non-interactive (i.e., they do not rely on user input) and could be tested independently of the main() function. At this stage, functions like view\_tasks() and main() were not tested, as they either only printed output (making them harder to assert) or involved interactive input, which required mocking or additional handling.

1. **Which code coverage tool did you use?**

I used pytest-cov, which is a plugin for the pytest testing framework. It integrates seamlessly with pytest to measure code coverage and generate reports showing which parts of the code were executed during testing.

The command I used to run the tests with coverage was:

pytest --cov=todo --cov-report=term

This command shows a summary of the code coverage directly in the terminal, indicating how many lines were covered and how many were missed in the todo.py file.

1. **How did you install and run the tool?**

To install the code coverage tool, I used the following command in the terminal:

pip install pytest pytest-cov

This installed both pytest (the testing framework) and pytest-cov (the plugin used for measuring code coverage).

To run the tests along with the coverage report, I used the following command:

pytest --cov=todo --cov-report=term

--cov=todo specifies that coverage should be measured for the todo.py file.

--cov-report=term displays the coverage report in the terminal.

This setup allowed me to run my test cases and see how much of the todo.py code was covered during testing.

1. **What is the current code coverage percentage of your program?**

At this stage, the current code coverage percentage of my program is approximately **45%**. This was achieved by testing only the core functions such as add\_task(), mark\_done(), and remove\_task().

Functions like view\_tasks() and the main() function were not included in the tests at this point, as they involve print statements and user input, which were not yet handled using mocking techniques. As a result, a significant portion of the code—especially the interactive logic—remained untested, leading to a lower overall coverage.

1. **Which additional functions or scenarios did you test to improve the coverage?**

To improve the coverage from 45% to approximately **95%**, I added tests for the following additional functions and scenarios

Tested both when the task list is empty and when it contains tasks.

Verified that the correct output is printed in each case using output capturing.

main() **Function**

This was the most significant improvement.

Used unittest.mock.patch to simulate user input and run different branches of the main menu logic.

Tested inputs for:

"1" – Adding a task

"2" – Viewing tasks

"3" – Marking a task as done

"4" – Removing a task

"5" – Exiting the program

Invalid inputs like "9" or non-numeric strings to test the error-handling flow

**Edge Case Handling**

Tried marking and removing tasks with invalid indexes.

Simulated user input errors (e.g., entering "abc" instead of a number), which triggers the ValueError in the try-except block.

These tests ensured coverage for exception-handling branches.

By testing the full range of user interactions and error scenarios in the main() function, along with capturing printed output, I was able to significantly boost the code coverage close to 95%.

1. **Re-run the code coverage tool. What is the new coverage percentage?**

The new code coverage percentage reported was approximately **95%**. This increase reflects the inclusion of tests for the view\_tasks() and main() functions, as well as the handling of invalid inputs and exceptions. Most of the logic in the program is now tested, with only a few lines possibly remaining uncovered due to complex input/output flows or rare edge conditions.