

## **Placement Empowerment Program**

### ***Cloud Computing and DevOps Centre***

Use Git Hooks for Automation: Create a Git hook that automatically runs tests before committing code to your repository.

Name: Hema S

Department: ECE

# Introduction:

This Proof of Concept (PoC) demonstrates the use of **Git hooks** to automate the process of running tests before committing code to a repository. Specifically, it uses the **pre-commit hook**, which triggers test execution via **pytest** for a Python project. The goal is to ensure that only code that passes the specified tests is committed to the version control system, thus maintaining code quality and preventing faulty code from being pushed to the repository.

# Overview:

This PoC explores the integration of **Git hooks** to automate the process of running tests before code is committed. Specifically, it uses the **pre-commit hook** to invoke **pytest** for running automated tests on Python code. If the tests fail, the commit is blocked, ensuring that only code with passing tests gets committed to the repository. This approach enhances the development workflow by automating quality checks and preventing problematic code from being pushed, ultimately improving the overall stability and reliability of the project.

## Objectives:

1. **Automate testing** by setting up a **pre-commit hook** to trigger tests before commits.
2. Integrate **pytest**, a powerful testing framework, to run automated tests on the Python code.
3. Ensure only error-free, validated code is committed, preventing bugs and improving code stability.
4. Learn how to utilize **Git hooks** for automating routine tasks in the development lifecycle.
5. Implement a seamless, automatic testing process that helps developers focus on writing code rather than performing manual checks.

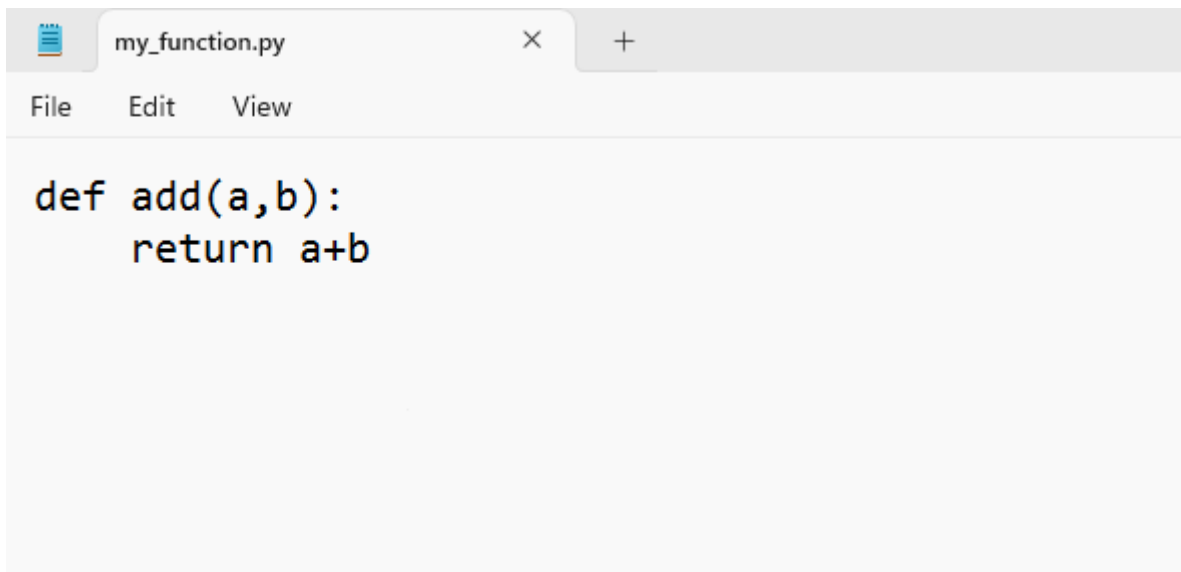
## Importance:

1. **Improves code quality** by ensuring all code commits are tested, reducing the likelihood of introducing errors into the codebase.
2. **Speeds up the development cycle** by automatically running tests before each commit, saving time and reducing manual intervention.
3. **Prevents code regressions** by ensuring that new changes don't break existing functionality.
4. **Increases consistency** in team environments, making sure that everyone's code passes the same automated tests before being committed.
5. **Enhances reliability** by catching errors early in the development process, leading to fewer bugs in production and smoother deployment cycles.

# Step-by-Step Overview

## Step 1:

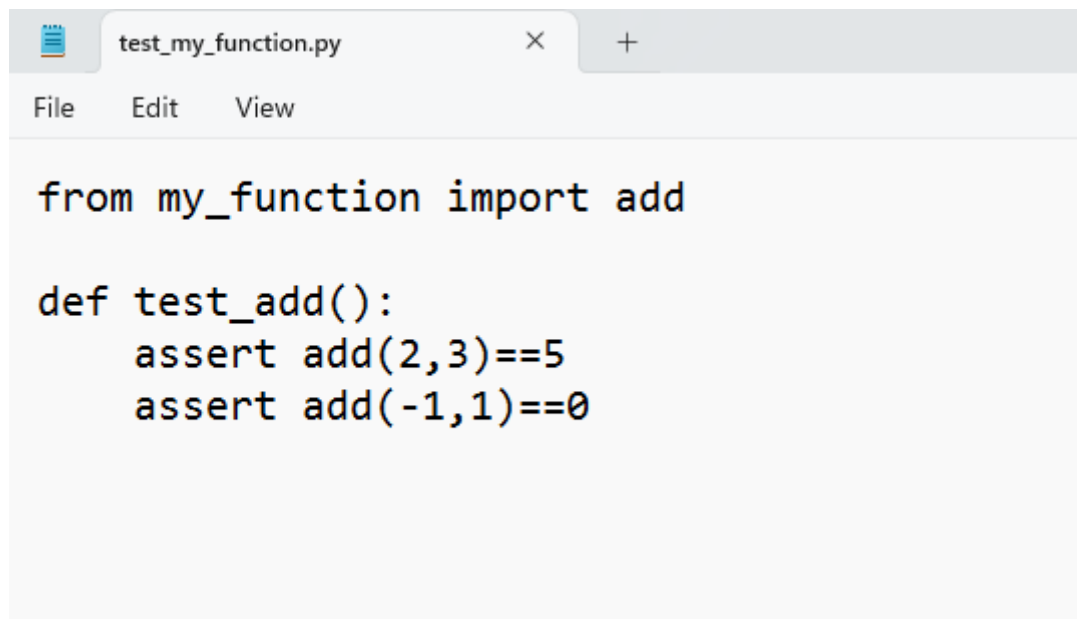
Create a folder in Desktop named my-python-project and Open Notepad Write the Python function to add two numbers and Save the file as **my\_function.py** in your folder.

A screenshot of a Notepad window titled 'my\_function.py'. The window has a menu bar with 'File', 'Edit', and 'View'. The main text area contains the following Python code:

```
def add(a,b):  
    return a+b
```

## Step 2:

Open Notepad again to create the test file and Write the test case code to test the add function and Save the test file as **test\_my\_function.py** in the same folder where my\_function.py is saved.

A screenshot of a Python IDE window. The title bar shows a single tab named 'test\_my\_function.py' with a close button (X) and a plus sign (+) for new tabs. Below the title bar is a menu bar with 'File', 'Edit', and 'View' options. The main editor area contains the following Python code:

```
from my_function import add

def test_add():
    assert add(2,3)==5
    assert add(-1,1)==0
```

## Step 3:

Open GitBash and Navigate to the folder where the files are saved.

```
MINGW32:/c/Users/sppra/desktop/my-python-project

sppra@DESKTOP-S8GOFLP MINGW32 ~ (master)
$ cd desktop

sppra@DESKTOP-S8GOFLP MINGW32 ~/desktop (master)
$ cd my-python-project
```

## Step 4:

Install pytest using pip by running the following command:

**pip install pytest**

```
sppra@DESKTOP-S8GOFLP MINGW32 ~/desktop/my-python-project (master)
$ pip install pytest
```

## Step 5:

Now that we have the function and test cases, let's run pytest to check if everything works.

```
sppra@DESKTOP-S8GOFLP MINGW32 ~/desktop/my-python-project (master)
$ python -m pytest test_my_function.py -v
===== test session starts =====
platform win32 -- Python 3.12.0, pytest-8.3.4, pluggy-1.5.0 -- C:\Program Files\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\sppra\desktop\my-python-project
collecting ... collected 1 item

test_my_function.py::test_add PASSED [100%]

===== 1 passed in 0.03s =====
```

## Step 6:

Run the following command to initialize Git in your project folder :

**git init**

```
sppra@DESKTOP-S8GOFLLP MINGW32 ~/desktop/my-python-project (master)
$ git init
Initialized empty Git repository in c:/Users/sppra/Desktop/my-python-project/.git/
```

## Step 7:

Copy the sample file to a new file called pre-commit (no .sample extension):

**cp .git/hooks/pre-commit.sample .git/hooks/pre-commit**

```
sppra@DESKTOP-S8GOFLLP MINGW32 ~/desktop/my-python-project (master)
$ cp .git/hooks/pre-commit.sample .git/hooks/pre-commit
```

## Step 8:

Open Notepad and type the following code and save as **pre-commit** in Inside the .git folder, **open the hooks folder:**

(Eg MyPath: C:\Users\Hi\desktop\my-python-project\**.git\hooks**)

```
pre-commit
File Edit View

#!/bin/bash
echo "Running pre-commit tests..."
pytest
if [ $? -ne 0 ]; then
    echo "Tests failed! Commit aborted."
    exit 1
fi
echo "All tests passed. Proceeding with commit."
exit 0
```

## Step 9:

Once the file is saved, you need to make it executable. You can do this by running the following command in Git Bash:

**chmod +x .git/hooks/pre-commit**

```
sppra@DESKTOP-S8GOFLL MINGW32 ~/desktop/my-python-project (master)
$ chmod +x .git/hooks/pre-commit
```

## Step 10:

You can test it by staging and committing some changes:

- Stage your changes:

**git add .**



- Commit the changes:

**git commit -m "Testing pre-commit hook"**

```
sppra@DESKTOP-S8GOFLLP MINGW32 ~/desktop/my-python-project (master)
$ git add .

sppra@DESKTOP-S8GOFLLP MINGW32 ~/desktop/my-python-project (master)
$ git commit -m "Testing pre-commit hook"
[master (root-commit) dc323c5] Testing pre-commit hook
5 files changed, 16 insertions(+)
create mode 100644 __pycache__/my_function.cpython-312.pyc
create mode 100644 __pycache__/test_my_function.cpython-312-pytest-8.3.4.pyc
create mode 100644 my_function.py
create mode 100644 pre-commit.txt
create mode 100644 test_my_function.py
```

## What happened:

1. The **pytest** ran successfully before the commit.
2. Since the test passed, the commit was allowed to go through.
3. You've now committed your changes, which includes the Python files and the test results.

You've successfully automated the process of running tests before committing with Git hooks!

# Outcomes:

By completing this PoC on using **Git Hooks** for automated testing, you will:

- 1. Implement a pre-commit hook** that automatically triggers test execution before every commit, ensuring that only valid code is committed.
- 2. Integrate pytest** with Git hooks to run automated unit tests, improving the overall code quality and preventing the introduction of bugs into the repository.
- 3. Learn to automate common tasks** using Git hooks, enhancing your understanding of version control automation in the development process.
- 4. Improve workflow efficiency** by eliminating manual test runs before commits, allowing developers to focus on coding without worrying about test execution.
- 5. Enhance collaboration** within teams by maintaining consistent code quality across different contributors, ensuring that all code changes meet the same testing standards before being committed.