Agile Software Engineering

for

**DevOps**

**Draft**

**Prepared by Aditya Kushal**

**Group Name: BTech Section A**

|  |  |  |
| --- | --- | --- |
| **Aditya Kushal** | **1RVU22CSE008** | **adityak.btech22@rvu.edu.in** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

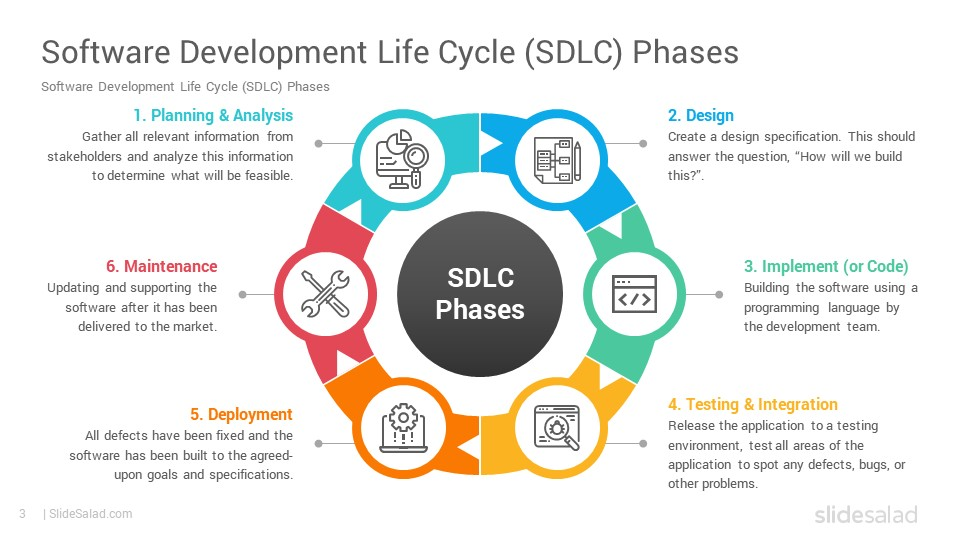
|  |  |
| --- | --- |
| **Instructor:** | **Prof. CVSN Reddy** |
| **Course:** | **Agile Software and Engineering** |
| **Lab Section:** | ***Project 2 – DevOps*** |
| **Teaching Assistant:** | ***Divya M*** |
| **Date:** | **06 October 2023** |

## 

# SDLC Life Cycle

## Phases –

* **SoW**: Single line statement of what the customer wants.
* **Requirements**: Analyze the different functional requirements of the product.
* **Requirements Analysis**: Break down high level requirements into its lower counterpart.
* **Design**: Identify the functions/methods of the product.
* **Coding**: Write code as per coding guidelines.
* **UT**: Unit Testing which is line-by-line testing of the code.
* **IT**: Integration Testing which is integrating code of different developers and testing how they mesh with each other.
* **ST**: System Testing, done by system testers where we test the functionality of the product.
* **Deployment**: Delivering the product built as per the agreement.
* **Maintenance**: Providing software updates to fix bugs or add more features to the existing product.



# Unit Testing (UT) – Line by Line Testing

**Library For Testing -** unittest, pytest (optional)

**Syntax for Testing -** self.assertEqual(a, self.calc.add(b, c), "Error Message")

* ‘a’ is the expected output
* ‘b’ and ‘c’ are parameters to add function
* ‘Error message’ is printed if the calculated value is not equal to the expected value.

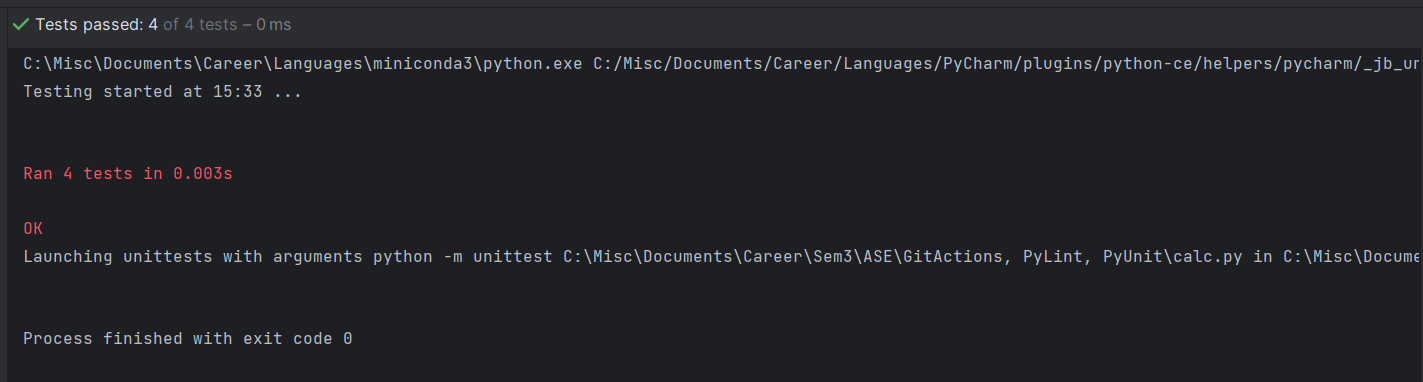
**Invoking Unittest -** unittest.main()

Full Code is in GitHub, python calc.py or pytest calc.py works for testing.

## 2.1 PyUnit - Passed

Ran 4 tests in 0.003s

OK

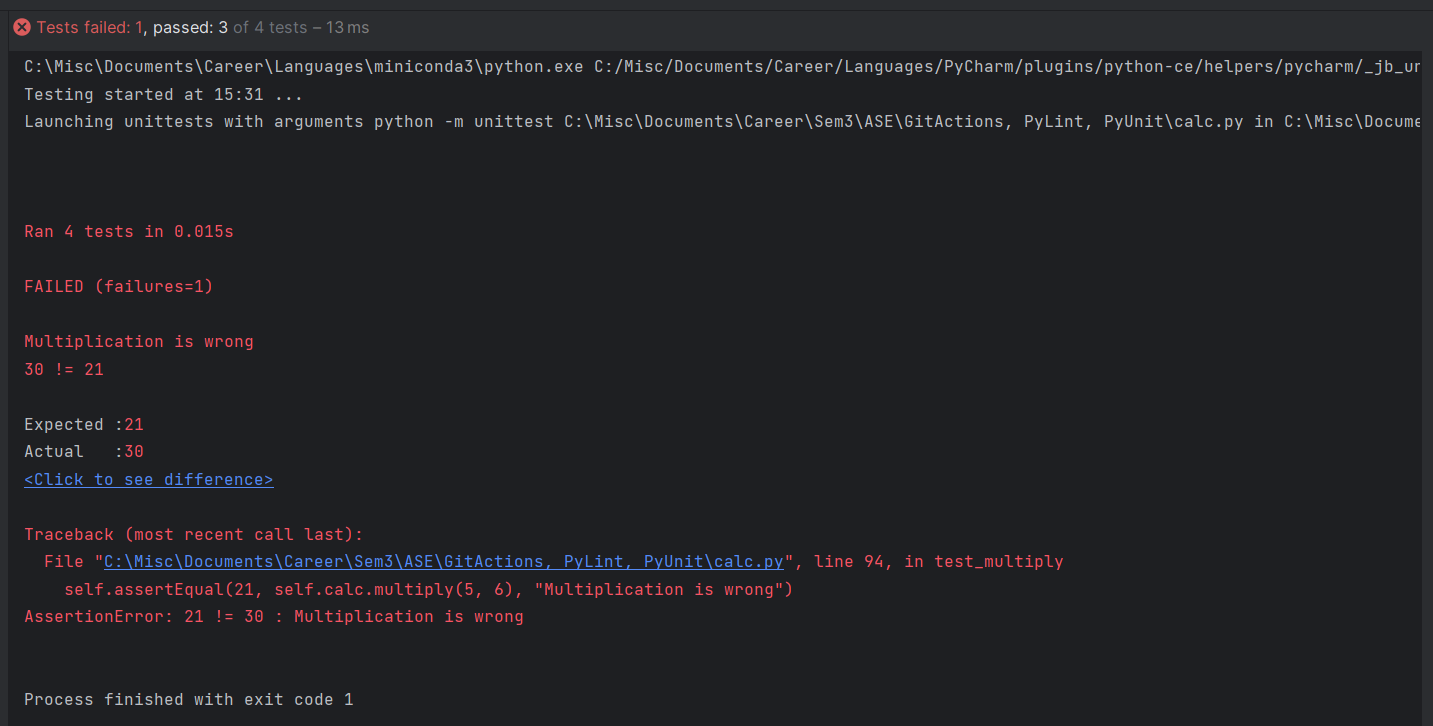


## 2.2 PyUnit - Failed

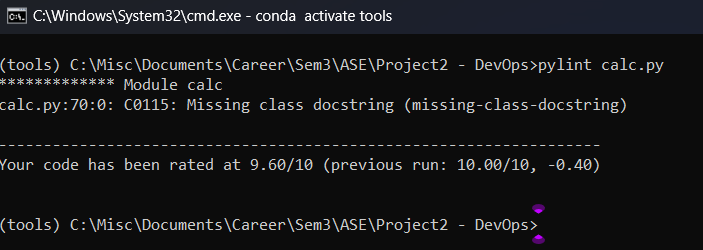
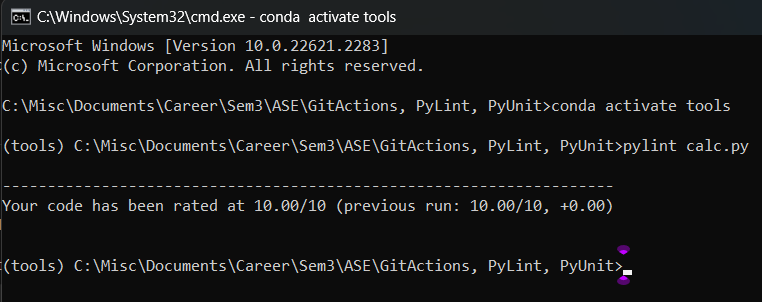
Ran 4 tests in 0.015s

FAILED (failures = 1)

30 != 21 (expected = 21, calculated = 30)



## 2.3 PyLint - Static Code Analysis Tool

* Can detect syntax errors, code style violations and potential bugs.
* Run **pylint calc.py** in cmd to get code analysis.
* Gives suggestions on how to improve the code and gives a score out of 10 based on how neat the code is written.
* We can use suggestions by PyLint with ChatGPT to improve our code further.

*PyLint - 10/10 Score*

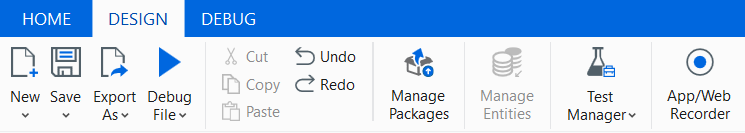
*PyLint - Below 10 score and suggestions on how to improve*

# System Testing (ST) - Functionality Testing

ST is also known as blackbox testing and is done by system testers to test the functionality of the product.

## 3.1 UiPath RPA - Test Script Automation

* Open UiPath Studio.
* Under New Project, Select Process.
* Launch Calculator App.
* Click on App/Web recorder.
* Hover over the calculator app till a green boundary comes over the entire app.
* Perform a calculation like 7 + 8 = 15.
* Save the recording.
* Under Debug File, Select Run File.
* UiPath will automatically perform the operation that was recorded earlier without human input.
* Save the project.



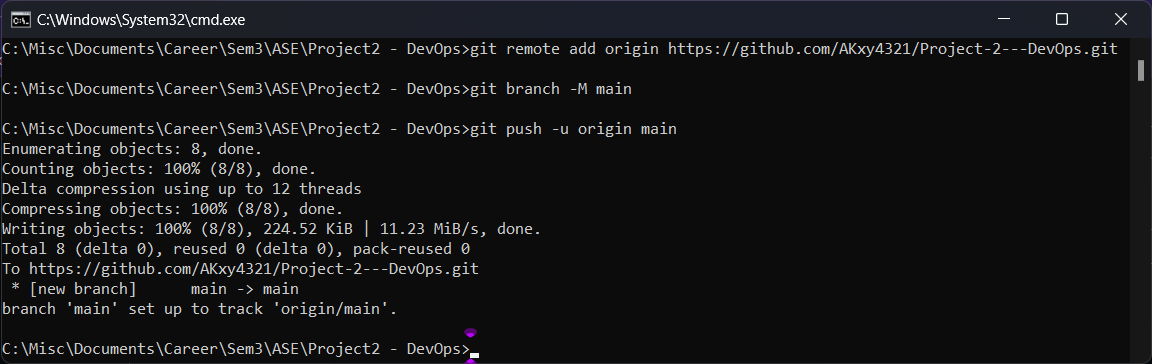
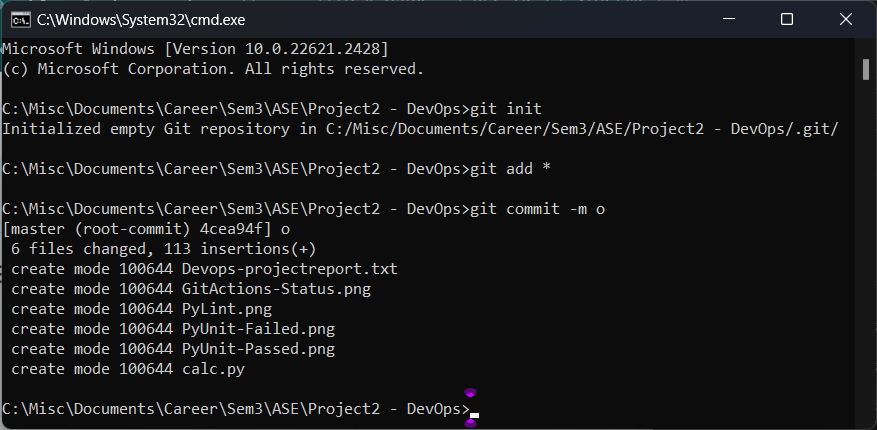
*Below is a snippet on how to create a new process.*

*Our focus is on App/Web Recorder and Debug File as shown below.*

# Version Control, Git and GitHub

* Keeping track of all changes to a given file so that we can go back to previous versions if the current file is not desirable.
* Git is one of the most popular version control systems, GitHub is git in the cloud.

## Git Commands

* **git init -** initialise repository (a directory where all files and their changes are saved).
* **git add -** add files to the staging area (in between local and git repository).
* **git commit -** save the files and changes to the git repository (from the staging area).
* **git status -** display current status of repository and staging area.
* **git clone -** clone a repository locally.
* **git push -** push files from git to github.
* **git pull -** pull files from github to git (local system).

**Git to GitHub**

*Now, in GitHub the pushed files will be reflected.*

# CI/CB/CT/CD and GitActions

**CI -** Continuous Integration

Regularly integrating code changes into a shared repository.

**CB -** Continuous Build

Automated compilation of a software project's source code.

**CT -** Continuous Testing

Automated testing of code changes as they are integrated and deployed.

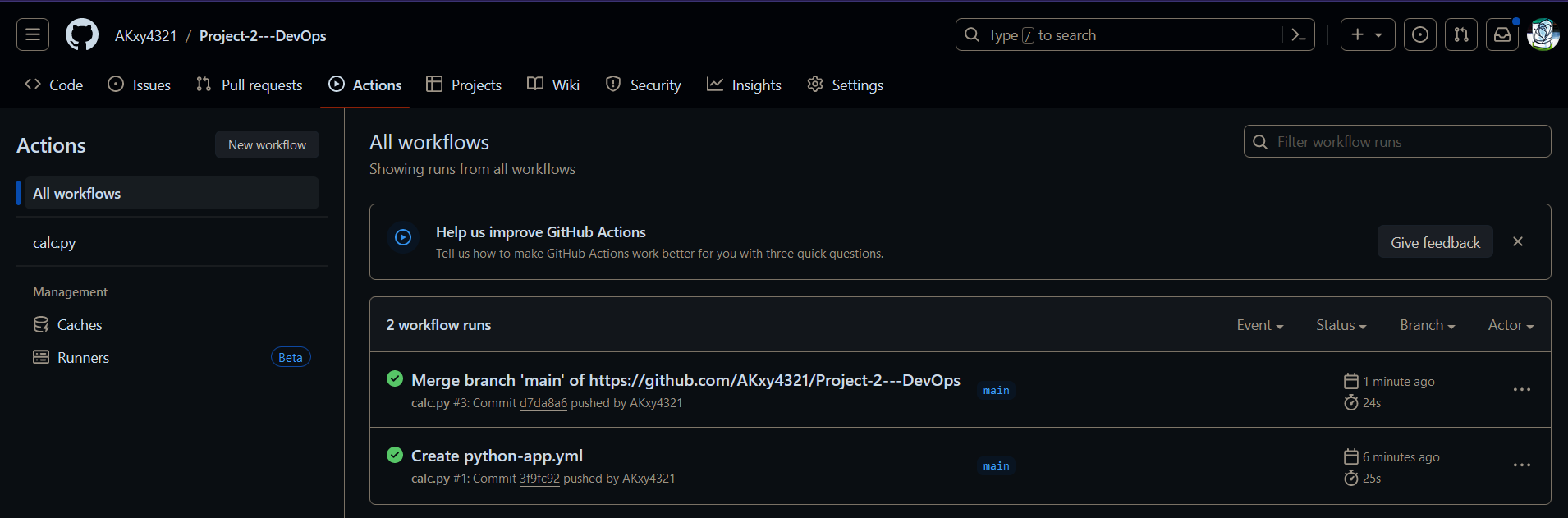
**CD -** Continuous Deployment

Automatic deployment of every code change that passes automated tests and quality checks directly to production.

Process of going through CI, CB, CT, CD is called the DevOps pipeline.

## GitActions

* Go to your repository in GitHub.
* Go to Actions Tab.
* Select Configure Python application option.
* Edit the yaml file (python calc.py, pytest calc.py and pylint calc.py - optional) and commit the changes.
* Pull the repository to your local machine with git pull.
* Add a dummy text file or make some changes to the code to make it different from the GitHub Repository.
* Push the repository from git to github.
* In the yaml file, the commands configured are run automatically and we can check their status in the Actions Tab.



**GitActions Summary**



*Actions taken by GitActions shown below.*

# Docker and DockerHub

* **Docker** is a tool that allows applications to be run anywhere using containers.
* **Docker Image** contains VM, OS, Software and any other additional libraries required to make the application run.
* **Docker container** is a running instance of a docker image.
* **Dockerfile** is a plain text configuration file used to build a Docker image.

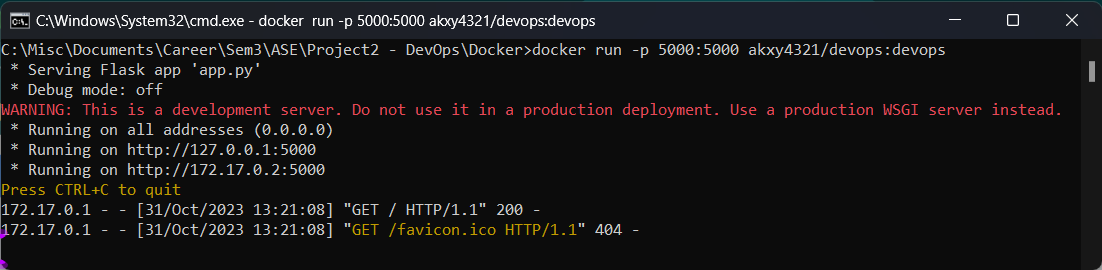
It contains instructions on how to create the docker image.

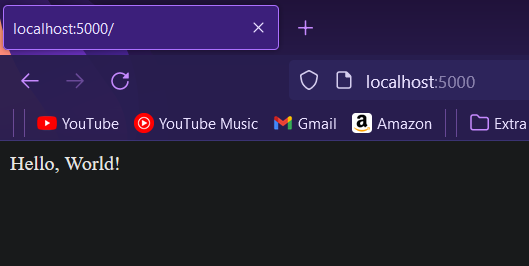
* **DockerHub** is docker in the cloud.

**Docker Commands**

* *docker build -t username/imagename:tag .* - Build docker image
* *docker run username/imagename:tag* - Run container
* *docker push username/imagename:tag* - Push to DockerHub

**Steps**

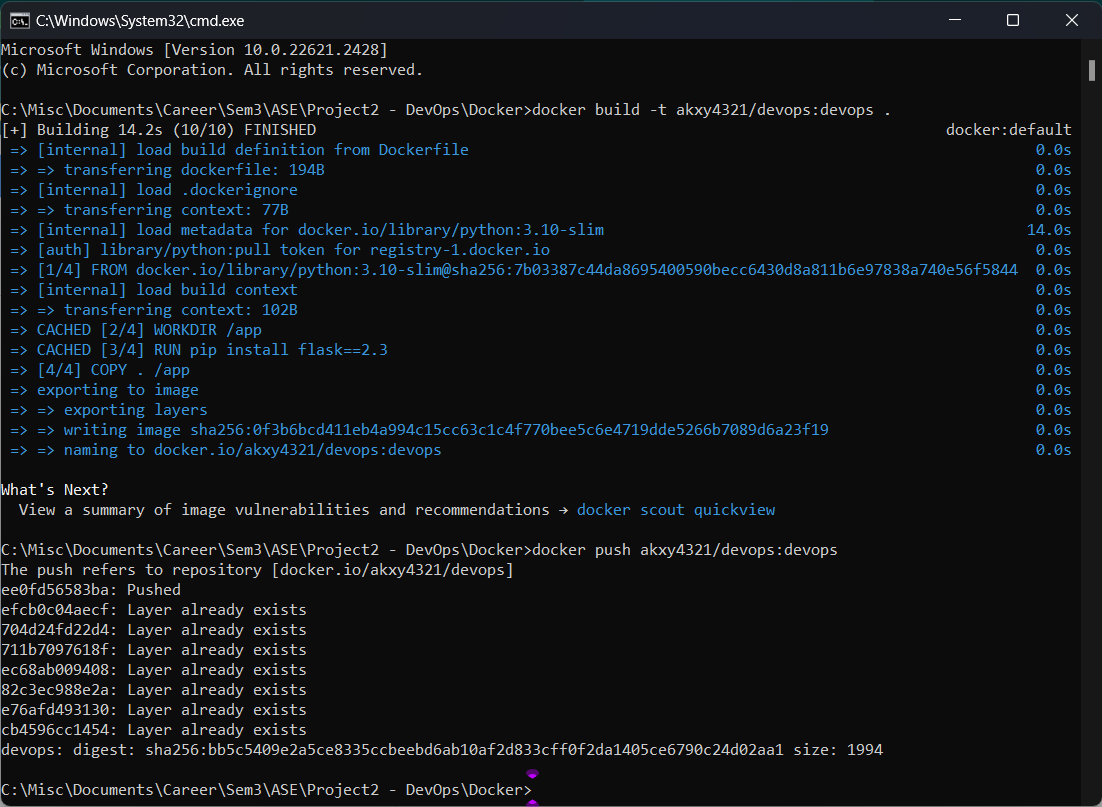
* Go to the desired directory, Alt+D and type cmd.
* In CMD, run **echo > Dockerfile**.
* Go to the Dockerfile and write the configurations.
* Go back to CMD, and run **docker build -t akxy4321/devops:devops .** .
* Push to DockerHub with **docker push akxy4321/devops:devops**.
* Run the container with **docker run -p 5000:5000 akxy4321/devops:devops**.
* Go to any browser and type **localhost:5000**.
* Hello World should be printed.

*Docker build and docker push demonstrated above.*

*Docker run demonstrated above.*

*Expected output when localhost:5000 is typed in any browser after docker run command.*

# Conclusions

Through this project I have learnt about the following -

* **SDLC** and its different phases.
* **PyLint -** Static Code Analysis.
* **PyUnit -** Unit Testing.
* **UiPath RPA -** Test Case Automation.
* **Git/GitHub -** Version Control System.
* **GitActions -** CI/CB/CT/CD.
* **Docker/DockerHub -** Deployment of containers.