Ryan Parker
Professor Businge
CS 472
5 February 2024

CS472 - Dynamic Analysis Report

Forked Repository: https://github.com/rparker2003/PythonTestingLab

Task 1

```yaml
name: CI workflow

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest
    container: python:3.9-slim
```

Fig 1. GitHub Workflow code at the end of task 1.

Figure one shows the outcome of following the instructions for task one. I created a workflow with a name, a listening action, and a single job named ‘build’. At the end of this task, I pushed the code in figure one to my repository and the GitHub action came back red, giving this error message: “No steps defined in `steps` and no workflow called in `uses` for the following jobs: build.” You can see in figure two that the GitHub action did not accept the file and that it wants me to write steps and uses for the action build.

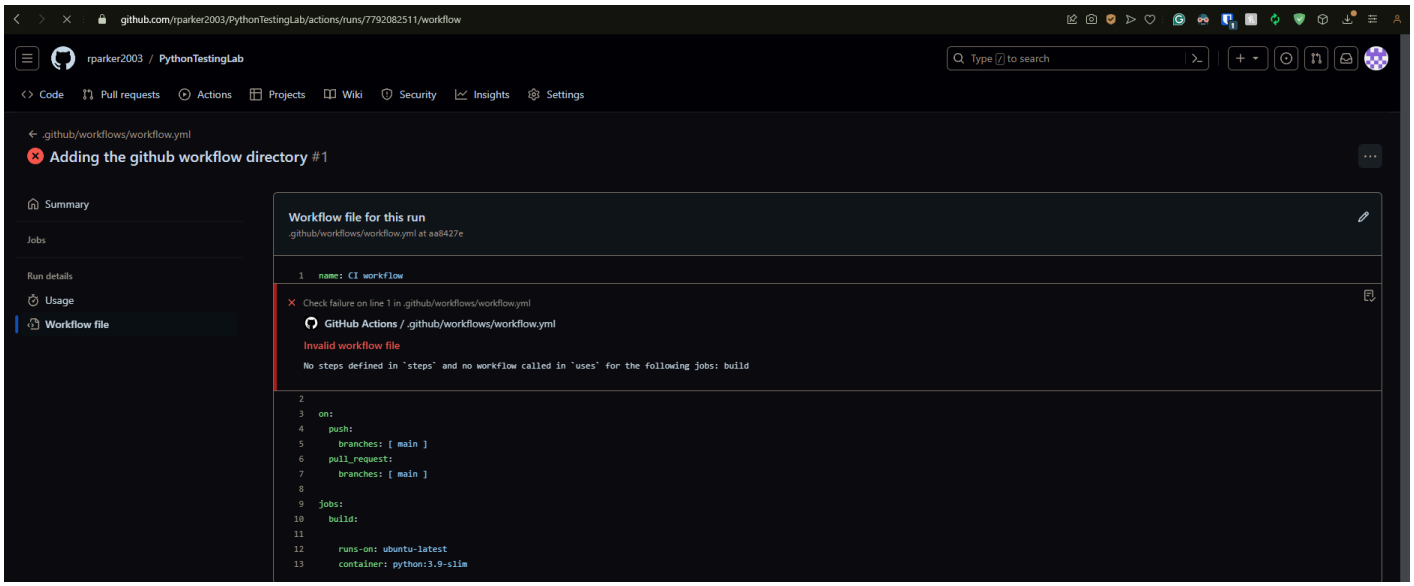

Fig 2. GitHub Action screenshot after the final commit of task 1.

Task 2

```yaml
name: CI workflow

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest

    container:
      image: python:3.9-slim

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Lint with flake8
        run: |
          flake8 src --count --select=E9,F63,F7,F82 --show-source --statistics
          flake8 src --count --max-complexity=10 --max-line-length=127 --statistics

      - name: Run unit tests with nose
        run: nosetests -v --with-spec --spec-color --with-coverage --cover-package=app
```

Fig 3. GitHub Workflow code at the end of task two.

In this task, I followed the steps to finish the GitHub workflow file in figure three. This workflow contains 4 actions, Checkout which allows the workflow to access the repositories files, install dependencies which does as the name implies using requirements.txt, Lint with flake8 which verifies all files follow the proper syntax and semantics, and finally run the unit tests with nose which will let us verify the coverage and validity of the tests.

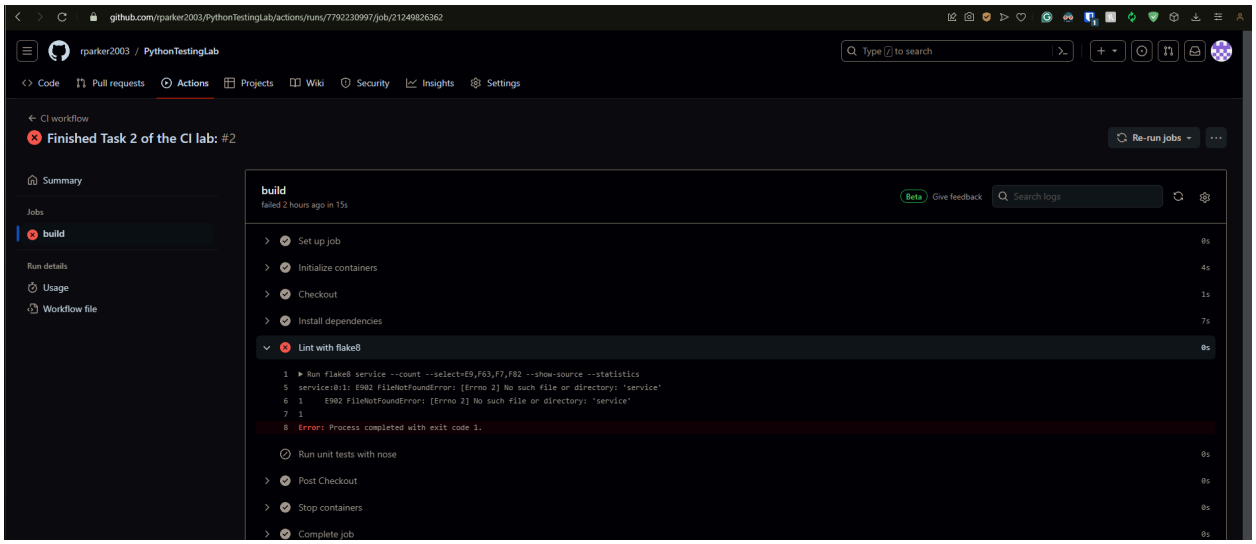

Fig 4. First GitHub Action report after pushing the finalized workflow file.

A challenge that I ran into was having to change the flake8 commands to reference ‘src’ instead of ‘service’. The error was “No such file or directory: 'service'” as you can see in figure four under the “Lint with flake8” step.

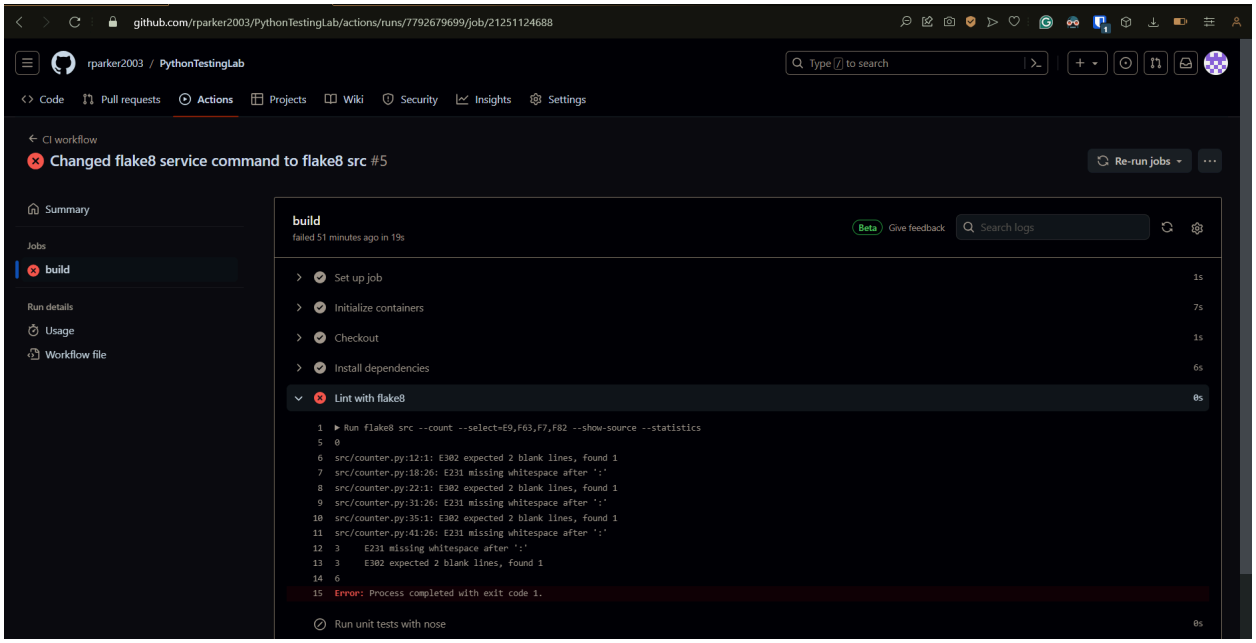

Fig 5. Second GitHub Action report after pushing the workflow file with updated flake8 command.

After changing the flake8 command, I noticed that flake8 was reporting that my files did not follow the guidelines that it was expecting. As you can see in figure five, I had six total corrections from flake8. All of the notifications are in my src/counter.py file and all of the errors referred to needing 1 more empty line between functions, and also adding whitespace after the colons in my return messages.

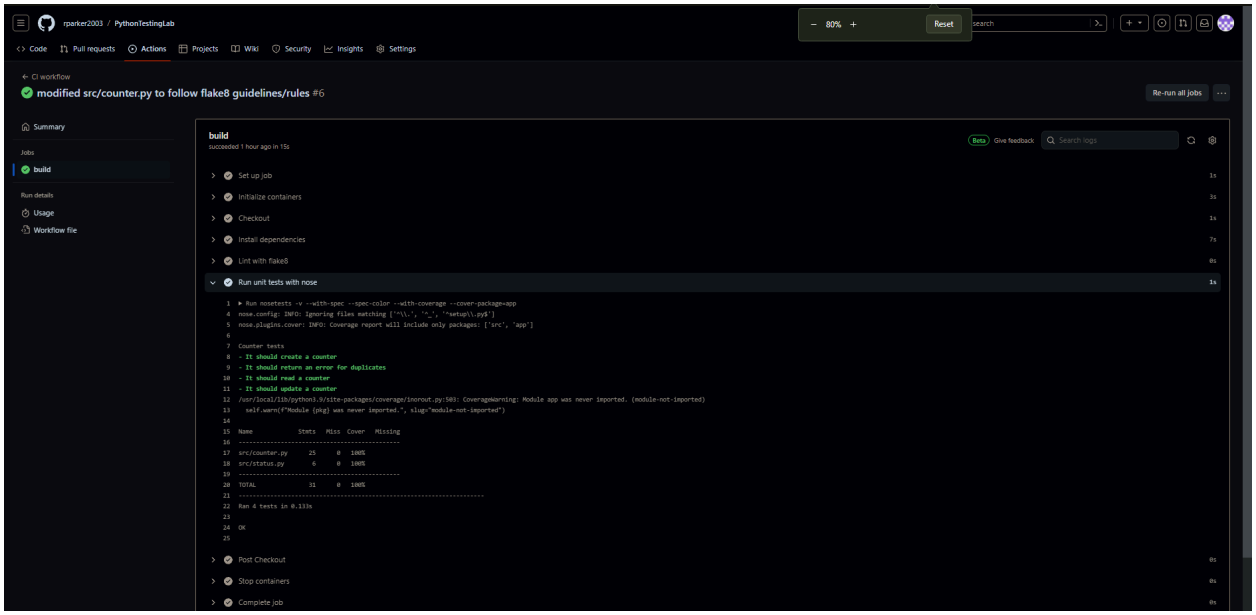

Fig 6. The first successful GitHub Action report after fixing the flake8 errors.

I returned to my src/counter.py file and modified it to follow the strict flake8 guidelines and verified this by running the flake8 commands locally before pushing the changes to the repository. Figure six shows the first successful GitHub action report and that all stages of the workflow have been passed. Figure six also shows that all of the current unit tests, create, read,

update, and check for duplicates, are passing. The next step for this assignment is to add a delete counter test following the red/green/refactor (TDD) order.

```python
def test_delete_a_counter(self):
    """It should delete a counter"""
    # Make a call to create a counter, and put the counter
    self.client.post('/counters/delete')
    self.client.put('/counters/delete')

    # Make a call to delete a counter, and ensure it returned a good code
    result = self.client.delete('/counters/delete')
    self.assertEqual(result.status_code, status.HTTP_204_NO_CONTENT)

    # Check that reading a fake client returns a proper return code.
    result2 = self.client.delete('/counters/delete_fail')
    self.assertEqual(result2.status_code, status.HTTP_404_NOT_FOUND)
```

Fig 7. Test case for deleting a counter.

```
Counter tests
- It should create a counter
- It should delete a counter (FAILED)
- It should return an error for duplicates
- It should read a counter
- It should update a counter

======================================================================
FAIL: It should delete a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/rparker/ws/2024/cs472/PythonTestingLab/tests/test_counter.py", line 88, in test_delete_a_counter
    self.assertEqual(result.status_code, status.HTTP_204_NO_CONTENT)
AssertionError: 405 != 204
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: read
src.counter: INFO: Request to update counter: read
--------------------- >> end captured logging << ---------------------

----------------------------------------------------------------------
```

Fig 8. Nosetests output after writing the delete counter test case. Red Phase!

As you can see in figures seven and eight, I wrote the test case for deleting a counter and finished the red phase of the test-driven development. Figure eight shows “AssertionError 405 != 204” as the source code for the test has not been written and the test case is asserting that the correct 204 code was returned, which it was not.

```python
@app.route('/counters/<name>', methods=['DELETE'])
def delete_counter(name):
    """Delete a counter"""
    app.logger.info(f"Request to delete counter: {name}")
    global COUNTERS
    if name not in COUNTERS:
        return {"Message": f"Counter {name} does not exist"}, status.HTTP_404_NOT_FOUND

    return {name: COUNTERS[name]}, status.HTTP_204_NO_CONTENT
```

Fig 9. Source code for deleting a counter.

```
Counter tests
- It should create a counter
- It should delete a counter
- It should return an error for duplicates
- It should read a counter
- It should update a counter

Name             Stmts   Miss  Cover   Missing
----------------------------------------------
src/counter.py      31      0   100%
src/status.py        6      0   100%
----------------------------------------------
TOTAL               37      0   100%
----------------------------------------------------------------------
Ran 5 tests in 0.132s
```

Fig 10. Nosetests after writing the delete counter source code. Green Phase!

Figures nine and ten show the final, green, and refactor, phases of test-driven development. In figure nine, I wrote the source code for the test that was previously written, and this source code purely checks if the name exists and returns a proper status code on that evaluation, only after deleting the counter if possible. Figure ten shows all five test cases being passed and also shows 100% coverage from nosetests.

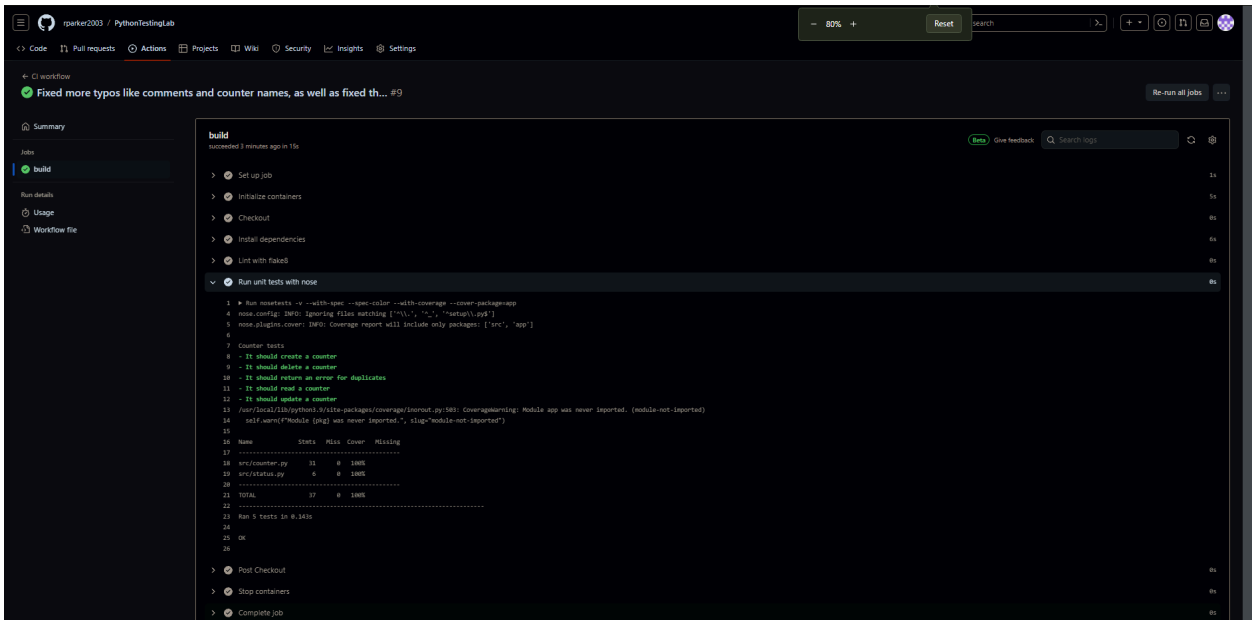

Fig 11. Final GitHub Action report showing the new passing test.

After verifying all code was functioning properly and that the test_counter.py and counter.py files I just edited still followed the flake8 guidelines, I committed my changes to the repository. Figure eleven shows the GitHub Action report from the workflow file created at the start of this assignment. It also shows the new test case and that it passed, as well as that I am achieving 100% coverage.