A2: TDD Hands On

✓ Published   ⋮

Motivation

This week we took a look at *Test Driven Development*. We walked through the steps on how to properly conduct TDD. The goal this week is to practice TDD while working to implement a specification.

Course Learning Outcome(s):

- **Apply** testing techniques, including black-box and white-box techniques, automatic testing activities, and regression testing (CLO 4)

Module Learning Outcome(s):

- **Apply** test driven development techniques

Description

For this assignment you will utilize TDD to implement the following specification:

`check_pwd` accepts a `string` and returns a boolean: `True` if it meets the criteria listed below, otherwise it returns `False`:

- Must be between 8 and 20 characters (inclusive)
- Must contain at least one lowercase letter
- Must contain at least one uppercase letter
- Must contain at least one digit
- Must contain at least one symbol from: `~`!@#$%^&*()_+-=` (copy and paste to avoid missing characters) These are the only permitted symbols

You may assume that only strings will be sent to the `check_pwd`.

This should look familiar, it is the specification we used in Week 5's Random Testing Exploration. The difference is that you now have to use TDD to create this function.

You may be thinking, "Couldn't we just copy what you wrote in the examples?" The answer is no. First, it doesn't work 100% correctly. Second, for this assignment you will have to *show your work*.

To accomplish this, you will be using a **private** GitHub repo to maintain a change log for each step of the TDD process. I have provided the TDD steps below in case you forgot.

TDD Steps

1. Write a test
2. Run all currently written tests
   - If the tests all pass, return to Step 1
   - If a test fails, proceed to Step 3
3. Write the bare minimum of code to make the test pass
4. Run all the currently written tests
   - If tests all pass, return to Step 1
   - If the failing test is still failing, return to Step 3
5. Occasionally evaluate if the code can be refactored to reduce duplication or eliminate no longer used parts of the code
6. Eventually stop development after adding "enough" tests without triggering a new failure

You are required to follow these steps when writing your code. You will need to maintain a file with the code you are writing and a file that houses your tests. If you write large chunks of the implementation at once or more than a single test at a time, you will lose points. You will create a new commit each time you do *either* of the following:

1. Write a *failing* test
2. Implement code to cause a failing test to pass

Each commit requires a message describing the changes. The message for each type of commit is as follows:

▶ Name of the test you wrote and a brief description of what part of the specification you are testing

2. Briefly describe the changes you made to the code and if it caused the test to pass

Sample Workflow

1. Read specification
2. Write first test and run it to ensure it fails
3. Commit changes, "Wrote test1 to see if an empty string is rejected correctly"
4. Write the bare minimum implementation to get the test to pass
5. Run test to see if it passes
6. Commit changes, "Made check_pwd return False no matter what, it passed"
7. Write second test...

Please note, you need to commit your changes after writing a test, but BEFORE you write the code to make it pass. You then need to commit your changes BEFORE writing another test. So, write test and commit, then write implementation and commit.

It is very important you follow these steps carefully as we will be checking your commit history to see that you adhered to the TDD process.

In order to submit this assignment, you will need to host your work on a private GitHub repo. You will also need to ensure that you have `coeCS362` added as a collaborator on the repo. Failure to do this will result in your assignment not receiving a grade.

Hints

- Do not immediately jump to writing all inclusive tests. Follow the examples in the exploration for how small your changes should be. Do not write the entire specification in one pass, do it in pieces
- DO NOT commit tests that initially passed (i.e. passed before writing implementation).
- Once written, a test should never be changed. So if you have a test that passes but then starts failing once you write more code, then something is wrong with your approach, reach out for help
- It is a good idea to start with simply `return True` in your `check_pwd` and then your tests should `assertFalse`
- Go slow and ensure you are committing BOTH after writing a test and the changes you made to pass said test
- Make your commit messages descriptive enough for the grader to understand or you risk losing points
- Don't fake this process or you won't gain the needed experience with TDD

▶ After you feel you have implemented the specifications, you can reassure yourself about your code by running some random testing before your final commit similar to in Random Testing's exploration (we will! HINT HINT). DO NOT include any random testing in your commits.

- Linking Gradescope to GitHub doesn't mean your submission to Gradescope is updated as you update the GitHub repo. You will still need to resubmit your repo if you make changes.

What to turn in

- On Canvas, you will supply the link to the **private** GitHub repo with `coeCS362` as a collaborator
- Your repo must have two files:
  - `check_pwd.py` that contains your password checking function (`check_pwd()`)
  - `tests.py` that contains the tests you wrote during the TDD process (with a main function)
- You will also need to turn in your files to Gradescope so we can run them against a set of Instructor tests (see below)

Submitting to Gradescope

In order for us to grade your assignment, you need to submit both your `check_pwd.py` and your `tests.py` to Gradescope. The easiest way to do this is to submit your private repo directly. This will only work if both `check_pwd.py` and `tests.py` are in the root directory.

The first step to submit your repo is to connect Gradescope with GitHub. To do this, open the Submit dialog box like you have before, but this time select *GitHub* instead of *Upload*. Once you have done this, you will be shown a button to "Connect to GitHub". Click it.

▶

Submit Programming Assignment

ℹ️ **Upload all files for your submission**

SUBMISSION METHOD

○ ⬆ Upload   ● ○ GitHub   ○ 🪣 Bitbucket

CONNECT YOUR ACCOUNT

[○ Connect to GitHub]

REPOSITORY

[Select a repository...   ▼]

BRANCH

[Select a branch...   ▼]

[Upload]   [Cancel]

You will then be taken to a GitHub page to authorize Gradescope to have access to your public and private repos. Don't worry, you may remove this permission at any time after you submit your work. Click authorize.

▶

Now you just have to select the repo and branch you want to submit for the assignment. Please note that you most likely will need to use `main` and not `master` here.

▶

Submit Programming Assignment

ℹ Upload all files for your submission

SUBMISSION METHOD

○ ↥ Upload   ◉ ⭘ GitHub   ○ ▣ Bitbucket

REPOSITORY

| ericianni/a2_repo | ▾ |

BRANCH

| master | ▾ |

SUBMITTING FOR

Eric Ianni

[Upload]   [Cancel]

The autograder will now start processing your submission. You will not see the results of our tests, but Gradescope will show you the results of *your* tests that are in `tests.py`. Please note, that since these aren't Gradescope tests, they will not indicate Pass/Fail with Green/Red. You will receive a Failed message for each failed test. You WILL NOT receive ANY message for passed tests. Please see the example below.

▶

test1 (tests.TestCase) (0.0/0.0)

Test Failed: False is not true

test2 (tests.TestCase) (0.0/0.0)

test3 (tests.TestCase) (0.0/0.0)

test4 (tests.TestCase) (0.0/0.0)

Test Failed: True is not false

FRIENDLY WARNING: Linking Gradescope to your GitHub repo DOES NOT mean any changes made on GitHub will be reflected on Gradescope. You need to resubmit your repo if you make any changes to your code. Again, Gradescope doesn't automatically track changes on GitHub.

Grading criteria

This is an activity; it is graded PASS/FAIL. You will receive full marks if you consistently followed the TDD process laid out above and is free from bugs. Your implementation will be run against a series of test cases to ensure it meets the specification.

Your assignment will receive 0 points if any of the following are true:

- Link not provided
- Repo is not **private**
- `coeCS362` is not a collaborator
- Failure to follow the TDD process at least a majority of the time (i.e. commit after EACH test and commit after EACH fix)
- Files are not named `check_pwd.py` and `tests.py`

  ▶ Files are not submitted to Gradescope
- Tests in `tests.py` generate a failure

Your assignment will be capped at 5 points if any of the following are true:

- Instructor tests generate a failure in your final implementation

Points   10

Submitting   a website url

Due	For	Available from	Until
Feb 14	Everyone	-	-

TDD Hands On

Criteria	Ratings			Pts
Description of criterion Requirements * Repo is private * Link provided * coeCS362 added as a collaborator * Followed the TDD process at least a majority of the time * Instructor tests do not generate a failure * Files are named check_pwd.py and tests.py	**10 pts** **Full Marks** All requirements met	**5 pts** **Half Marks** Meets all criteria, but the instructor tests generate a failure	**0 pts** **No Marks** At least one criteria missed	10 pts

Total Points: 10

▶