

Project 6b - Open Source Contribution (Hedy)

Selected Project

Hedy is a gradual programming language aimed at teaching programming and teaching Python. It teaches using different levels with the first level just offering printing text and asking for input. This level is meant to introduce learners to the idea of a programming language, and the environment. From there, Hedy builds up to include more complex syntax and additional concepts. Generally speaking, it's core users are students or anyone who is interested in getting into coding. Hedy is linked at <https://hedycode.com/>.

Social Good Indication

Hedy contributes to Social Good. It falls under the fourth goal of the 17 Sustainable Development Goals: Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all. Hedy contributes to this goal by providing a free education service that is easily accessible (does not require additional software, just a browser) to many and promotes the opportunity to many to learn the skill of programming

Project context

Hedy was developed by Felienne Hermans as part of her research project at the Leiden Institute of Advanced Computer Science. Hedy was developed to be “a gradual language for programming education” with the aim of being “a new way of teaching the syntax of a programming language to novices, inspired by educational methods by which punctuation is taught to children” to combat the learning curve of remembering the right syntax. As described in her [paper](#):

“Prior research demonstrated that students submit source code with syntax errors in 73% of cases and even the best students do so in 50% of cases. An analysis of 37 million compilations by 250.000 students found that the most common error was a syntax error, which occurred in almost 800.000 compilations. It was also found that Java and Perl are not easier to understand than a programming language with randomly generated keywords, stressing the difficulties that novices face in understanding syntax.”

Hedy has been made open source with the permission of Leiden University and is mostly maintained by volunteers.

Project governance

Since the project is smaller, there is no streamlined or formalized process. Communication is mainly done on github through the issue thread (see figure 1) or in the discussion or project tab of the project. There is a slack channel, but communication there is extremely limited.

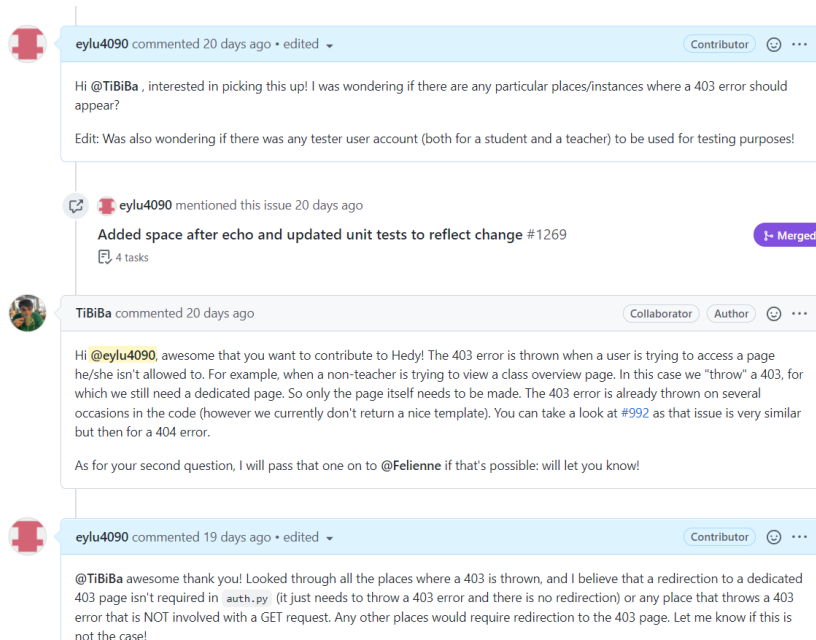


Figure 1: Communication on issue thread

The process to fix issues was also quite informal. Generally speaking, contributors could claim a task by dropping a comment in the issue thread, then submit a PR with changes. The PR request description requires three main components: a description of the fixes and changes, what issue the PR fixes, and how to test the changes. Upon submission of a PR, automated testing is run on the changes including running validations, end-to-end testing, and unit tests to ensure the changes will not break the current system (see figure 2)

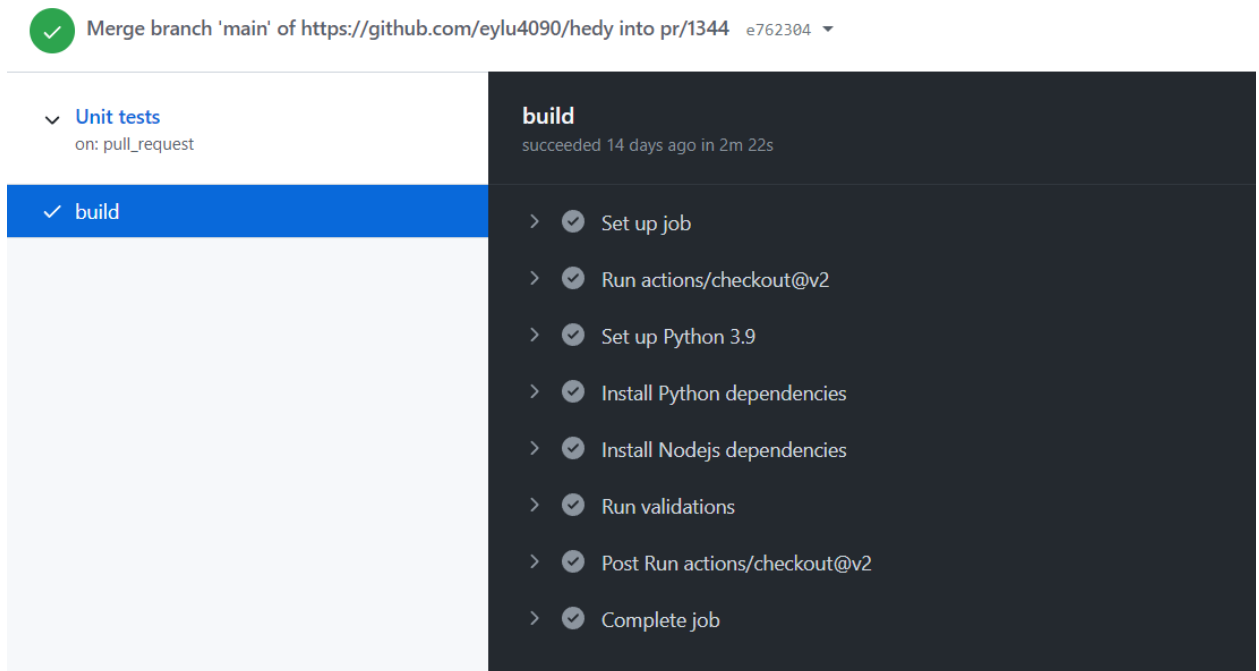


Figure 2: Automated testing on PR

Elizabeth Lu (eylu)

There are no formal requirements or documentation on design or quality assurance, though it is expected that every PR has been tested, added to unit testing, or provides some form of testing. Changes in designs usually are specified in the issue tracker or will be based on current design.

Task description

Task 1, priority 1: <https://github.com/Felienne/hedy/issues/1258>

Description

The bug involves fixing a functionality issue with the echo call. The current echo call is missing a space. The current echo has incorrect behavior from a previous patch that caused the output to be put directly next to the echo. The echo call would require an insertion of a space, and any test cases involving the echo call would have to be updated to account for this change.

Implementation

I updated the return call in [hedy/hedy.py](#) within the echo function to include a space before the answer. In addition to the space update, there were 2 unit tests where the answer had to be updated to reflect the change. This was done by running the unit tests on the code changes, identifying which tests failed, checking if failure was due to lack of space, then adjusting the tests.

Task 2, priority 2: <https://github.com/Felienne/hedy/issues/1161>

Description

This task is to create a 403 page when a user tries to visit a page he/she isn't allowed to visit; returning an unauthorized error. Currently, the website simply returns a 403 error and some text. This task involves creating a dedicated 403 page, then returning a 403 error and redirecting the user to the page when appropriate. This would usually happen when a student would attempt to access a teacher page or if a student attempts to visit a class he/she is not a part of. The solution could also require unit tests to ensure that the 403 error is returned if a user visits a page they are not authorized to visit.

Implementation

This task required more involvement than the first task. There were __ main components: adding warning messages, creating the html template, creating the endpoint, and identifying current 403 calls and updating them as appropriate. The warning messages had to be placed in a dedicated file that hosted all messages used. The created html template included a newly created 403 image (see figure 3), an informational message, as well as a redirection button back to the main page. This html page was rendered through a separate function that could be called and would also throw a 403 error. Finally, I had to parse through the current pages for 403 errors and render the 403 page in the cases where it would be appropriate to show the page

Elizabeth Lu (eylu)

(for example, if a student attempted to access a class page they did not have permission to or a page for teachers only. A dedicated page would **not** appear in any POST/PUT requests endpoints). Most of the 403 errors were in the app.py file where most of the main page content was hosted and the website/teacher.py file where most of the 403 errors thrown were if a student attempted to access a link that would enable them to edit or register for a class they did not have permission to.

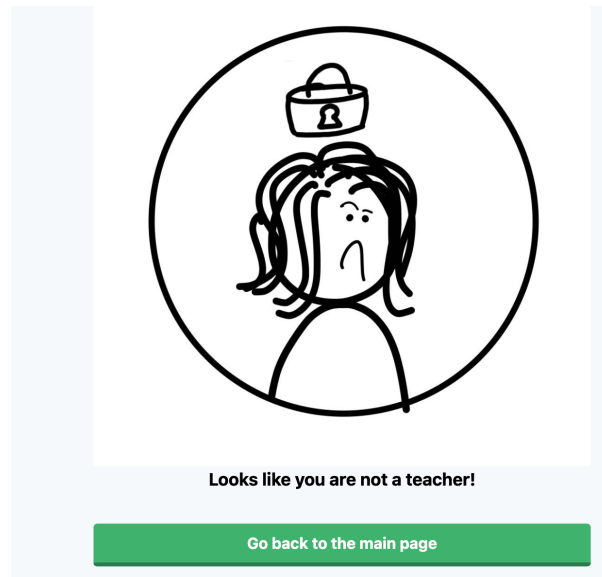


Figure 3: New 403 image

Submitted artifacts

Task 1 Accepted PR: <https://github.com/Felienne/hedy/pull/1269>

The linked PR includes all code changes and evidence of being merged into the code base. There are two changed files (bug fix and updated unit tests), automated build tests, and the PR description which includes my testing process and a more detailed description of changes.

Task 2 Accepted PR: <https://github.com/Felienne/hedy/pull/1344>

The linked PR includes all code changes (7 files -- 6 modified and 1 added) and evidence of being merged into the code base. The PR also includes a description of the changes and how to test the changes.

QA strategy

Any changes submitted in a PR are automatically built and run against end-to-end tests and unit tests (**integration testing**). Before submitting, I could also run the **end-to-end tests and unit tests locally** to ensure that any changes I made would work. I also included my testing process in all PRs to help the reviewers easily understand my steps or replicate the tests on their end.

Elizabeth Lu (eylu)

The project did not have any formal guidelines for **code review**, so participating in the code review process was usually waiting for feedback on a PR. Of the two PR I submitted, neither got any comments that required any resubmission of code. I purposely only submitted a PR when I thought I had thoroughly completed the task in terms of code and testing.

As the project I worked on was mainly based on the website, I could launch a locally hosted version of the website and visually test my changes. My workflow usually consisted of making the change, checking localhost if the changes were as intended, making modifications as needed, then running the unit tests and the end-to-end tests per minor change. This would ensure that each change did not break anything in the website and the small increments made pinpointing errors much easier. I also **added/modified test cases** as appropriate.

To test some changes, I also had to set up a testing environment to simulate login and a database. I was given the testing environment variables, and from there, I could login as a student or teacher. This was relevant for my second task as I needed to show the rendered 403 page given incorrect permissions (ie a student accessing a teacher page).

Generally speaking, this QA strategy was built upon my current testing strategy, what I used at my previous internships, and a general feel of what other contributors used for testing by looking at other accepted PR. This was to ensure I was comfortable with the workflow and it was the behavior that the Hedy team expected

QA evidence

Automated integration testing:

Task 1 PR: <https://github.com/Felienne/hedy/pull/1269/checks>

Task 2 PR: <https://github.com/Felienne/hedy/pull/1344/checks>

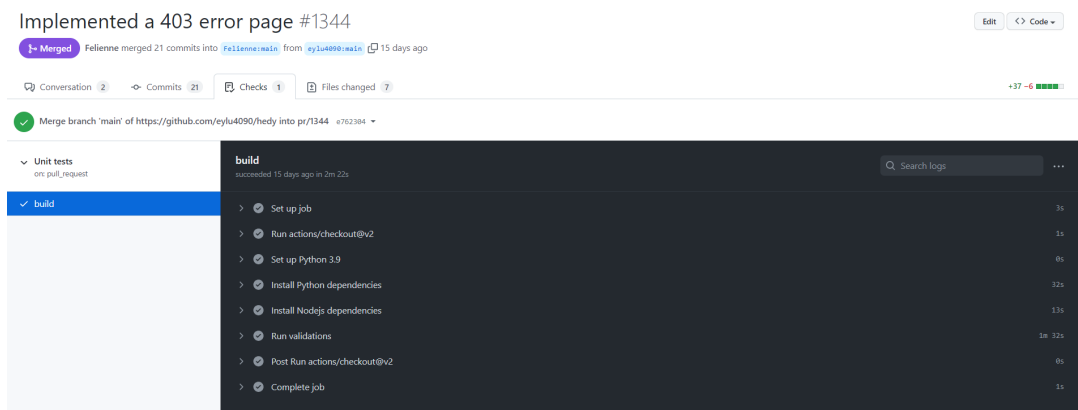


Figure 4: Automated integration testing on PR

Modified unit tests (from task 1 PR):

```
tests/test_level_01.py
@@ -136,7 +136,7 @@ def test_echo_without_argument(self):
136 136     def test_echo_with_quotes(self):
137 137         code = textwrap.dedent("""\
138 138             ask waar?
139 -             echo oma's aan de!""")
139 +             echo oma's aan de!""")
140 140
141 141         result = hedy.transpile(code, self.level)
142 142
@@ -209,7 +209,7 @@ def test_print_ask_echo(self):
209 209     expected = textwrap.dedent("""\
210 210         print('Hallo')
211 211         answer = input('Wat is je lievelingskleur')
212 -         print('je lievelingskleur is'+answer)""")
212 +         print('je lievelingskleur is'+answer)""")
213 213
214 214     result = hedy.transpile(input, self.level)
215 215     self.assertEqual(expected, result.code)
```

Figure 5: Modified Unit Tests

Locally run unit tests

```
tests/test_level_10.py ..... [ 48%]
tests/test_level_11.py ..... [ 12%]
tests/test_level_12.py ..... [ 12%]
tests/test_level_13.py .. [ 12%]
tests/test_level_14.py ..... [ 12%]
tests/test_querylog.py .. [ 12%]
tests/test_translating.py ..... [ 13%]
tests/test_utils.py ... [ 13%]
tests/test_z_adventures.py ..... [ 18%]
..... [ 25%]
..... [ 31%]
..... [ 38%]
..... [ 45%]
..... [ 51%]
..... [ 53%]
tests/test_z_defaults.py ..... [ 58%]
..... [ 65%]
..... [ 71%]
..... [ 78%]
..... [ 85%]
..... [ 91%]
..... [ 98%]
..... [ 99%]
tests/test_z_yamlfile.py . [100%]
===== 1886 passed in 61.59s (0:01:01) =====
```

Figure 6: Locally run unit testing before PR submission

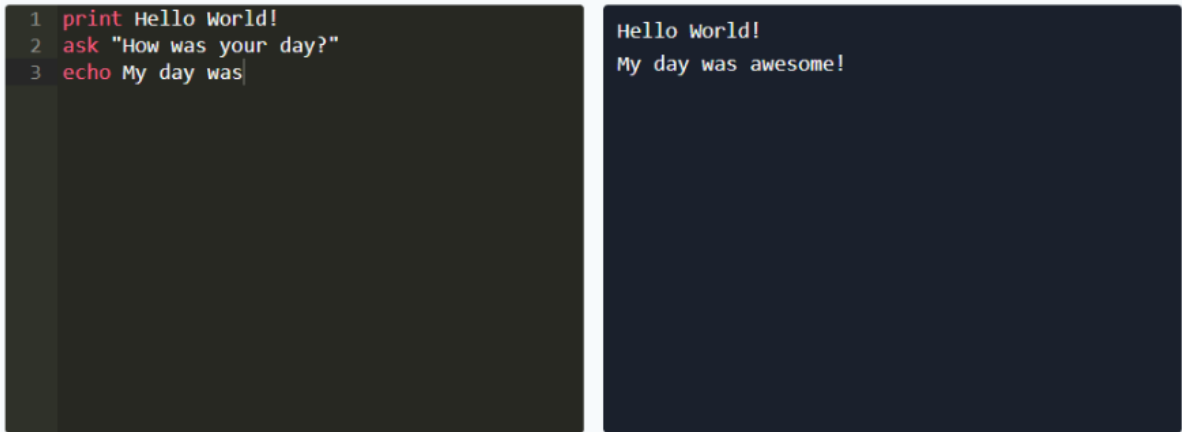
Elizabeth Lu (eylu)

Testing changes on website (in task 1 PR and task 2 PR)

Task 1: <https://github.com/Felienne/hedy/pull/1269>

Sample test behavior on local server:

1. Go to localhost:5000 and click on Hedy
2. Go to level 1, and input the follow text (or any text that involves echo with other statements)
3. Observe intended behavior



```
1 print Hello World!  
2 ask "How was your day?"  
3 echo My day was
```

```
Hello World!  
My day was awesome!
```

Figure 7: Directions on testing in PR for task 1

Task 2: <https://github.com/Felienne/hedy/pull/1344>

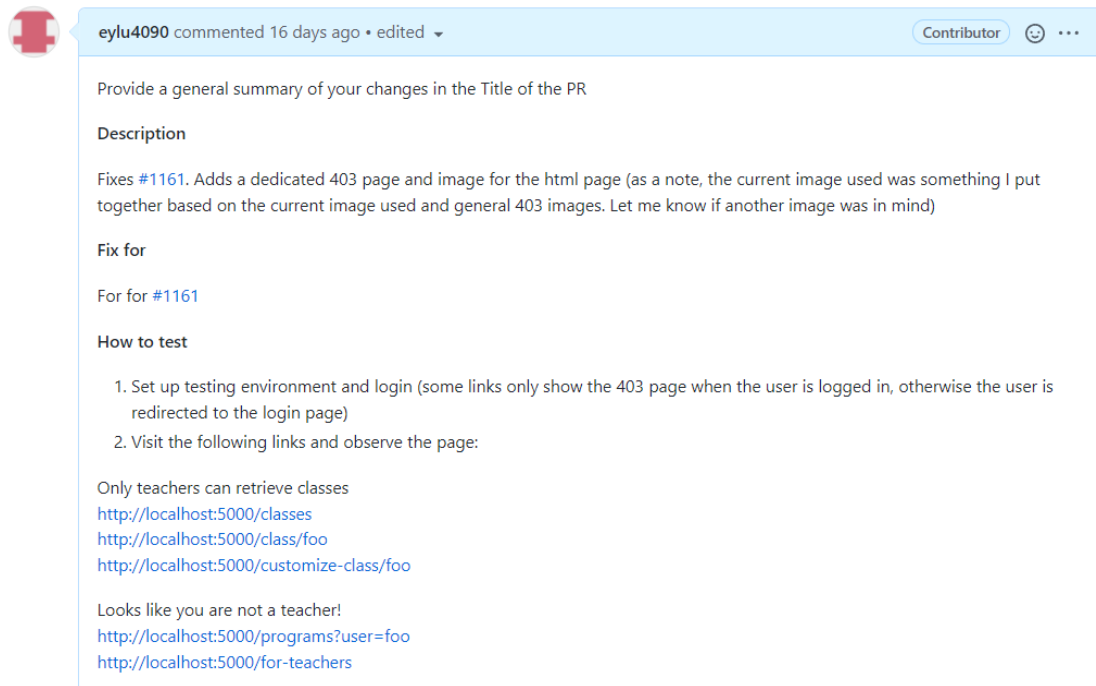


Figure 8: Directions on testing on PR for task 2

Testing Environment:

A redacted version of setting up the test environment is linked [here](#). Test environment variables and communication have been redacted with permission [via post 865](#) on Piazza. The testing environment setup was needed to connect to the test database to enable login on the local server specifically for task 2 (see figure 8) to enable the 403 page to show instead of being redirected to a login page.

Plan updates

Task 1 and task 2 were completed to full expectation in terms of deliverables. However, I was unable to complete task 3 and as noted in my previous report, task 3 would be completed if there was enough time. Both task 1 and task 2 took a bit longer than expected as I needed to provide evidence of QA. Setting up the testing environment for task 2 was substantial work in terms of waiting for communication, setting up the environment, identifying the links, and then debugging.

While I was able to complete some of the tasks ahead of schedule in terms of days, the hours spent on each task was much longer. There were also a few instances where the localhost would not load, so I ended up having to delete and repull the repo and rebuild everything. Doing so would waste about 0.5 - 1 hour of time as building all the dependencies took a long time. In the same vein, I would spend a decent amount of time trying to determine on my end if I broke something, if it was an issue with the build, an issue with VSCode, or an issue with permissions/my system. This was a risk I had not anticipated as I initially thought if I had built the project once, I would no longer have to build again.

Task 3 scope (splitting up current tests and rewriting the test runner) was also larger than I had anticipated in the code comprehension department. Most of my time was initially spent on understanding how the current test runner was framed, and I was unable to produce a new test runner before the deadline with enough time to write a report.

My schedule looked closer to this:

Task	Initial Estimation (in hours)	Actual Completion (in hours)
Set up (fork repo, set up environment, do initial run of test, launch localhost and run through product to understand the usage)	2	3 hours
Complete task 1 (issue #1258)	3	~5 hours (including communication, brainstorming, creating PR, testing, coding, and running into build issues)
Complete task 2 (issue #1161)	6+	~8 hours (including communication, brainstorming, creating PR, testing, coding, and running into build issues)
Complete task 3 (issue #1039)	6+	Started but did not complete
Complete the final report	5	~7 hours, took over the course of 5ish days

Experiences and recommendations

Finding the project

Getting into the open source world was very difficult. I spent a lot of time (probably even more time than part b of the project) finding a project I liked, and I felt comfortable contributing to. I had never contributed to open source projects, and despite combing through multiple repositories and starting weeks ahead to find a project, I could not find one where I felt I could apply my skills in a short time period, and I also felt comfortable hopping onto. It wasn't until a few days before the first part of the project that I found a project, Hedy, that I felt was within my skill set, and I felt welcome in.

The project code base was smaller than other open source projects I was looking at, which I believe was a better choice for me. This meant less onboarding time and a smaller team. The smaller codebase was something I purposefully was looking for. At a previous internship, the codebase I was working on was enormous, and onboarding time on my section of the codebase

Elizabeth Lu (eylu)

took 2 weeks to a month, time which I didn't think I would have with this project. I also was personally looking for a team of project maintainers that seemed involved with most aspects of the project and invested in bringing new people on.

Team Culture

The entire team was very welcoming. Feliene, the creator of the project, was always willing to lend a hand or hop onto a call to help out, and was open to any new ideas. For example, when asking for test accounts, she had mentioned she liked the suggestion and mentioned creating dedicated student and teacher accounts for testing in the future. All team members were positive and patient with the communication.

There were some delays with communication as the team was based in Europe and the slack was not active, but I never felt like I was waiting an extremely long time for an answer or for help. The project collaboration culture definitely helped my effort as I felt like the project team members cared about what I was doing. I was actually looking forward to contributing to the project and the tasks I had picked out, which I believe is an important component of open source contribution as well as potential work I may do in the future.

I found that project and team culture was much more important than I previously thought. I used to be more of the mindset that having a matching team culture was more of a "nice to have" rather than a vital component, and this project has proved otherwise. Having a very open and learning environment made it easy to ask questions and support, and it made the onboarding process much easier, and I believe it is critical to pulling in lasting contributors.

Submitting Changes and Previous Experiences

Submitting a PR was intimidating, and I probably took much longer than I needed to. I included a clear summary of changes, the purpose of the changes, proper links to issue threads, replicable testing process, test results (screenshot or links), and general formatting to ensure the reviewer would have to do minimal work.

One of the most important lessons I applied from this class and was alluded to in the code review [comments](#) in one of the code reviews was that a well-written PR and clean code review is extremely valuable. One reason is for documentation purposes: if someone new or unfamiliar with a portion of the code base were to look at the changes in the future, they would understand why the changes were made and how the changes were made. The other reason is time: having as few iterations of your changes as possible while still producing the correct work is valuable because engineering time is expensive.

This was actually a metric that was kept track of at one of my previous internships (ie, the number of iterations per "diff"/PR before the code changes were merged into the codebase and deployed). As mentioned many times in this class, this means communication is extremely important which includes clarifying what the specs/user wants, what changes actually need to

Elizabeth Lu (eylu)

be made, asking for help, asking for resources, etc. These can greatly reduce the number of versions of code submitted and can reduce frustration for all parties.

A second chance

I don't think I would make too many changes if I had a second chance at this project. I was pretty happy with the project I chose, and I plan to potentially to continuously contribute to Hedy in the future as I find the work to be interesting, and I enjoy working with the team. There are a few changes I wish I made to my schedule. As discussed in class, we often underestimate the time effort for tasks. Most of my tasks (unsurprisingly) took much longer than I anticipated even though I tacked on an additional hour or two to my initial estimation. The most significant change would have been to allocate more time and perhaps hash out more details within the tasks as there were multiple extra subtasks that I did not fully account for.

Part of me wishes that I did attempt to look at an open source project I actually use myself (ex: VSCode, Pandas, React, Jupyter), as I think it would be cool to work on a feature or bug fix that I may see and use in my own life. However, this project has really opened my eyes to the open source world, and that might be a project I tackle myself in the future :)

Advice

Find a project you are actually interested in and want to contribute to because there is no AG to motivate you or tell you how close you are to completing the task.

I'm willing to let future students read this report

Optional extra credit

Both linked PR have been merged

1. <https://github.com/Felienne/hedy/pull/1269>
2. <https://github.com/Felienne/hedy/pull/1344>