WRITING RULES – SUMMARY MATERIALS

THE 903 VALIDATOR REPOSITORY

As a reminder, the link to the repository is https://github.com/SocialFinanceDigitalLabs/quality-lac-data-beta-validator. Basics of the layout are in the README file, and feel free to click around and have a look in more detail.

Some of the code covered references the API between Javascript and Python – for those who are unsure, the term API just refers to a "shared interface" between two different programs. So in this case, I try to make the guarantee that the frontend (Pyodide) will only call this library via the API functions – this makes development easier since I can focus on those and make sure they don't change as we make changes to the underlying code.

COLLABORATING ON GITHUB

This resource clearly explains the idea of branching: https://guides.github.com/introduction/flow/ and should cover all the concepts we use in more detail in this video – specifically **branching**, **committing** and **merging**.

It's helpful to also understand the different Git concepts around development. When working with GitHub, you have two version of the repository – your **local** version (on Repl.it for us, so not actually local) and the **remote** version on GitHub. So, for example, when you click "Run on repl.it", it's creating a new local version on Repl.it that is a clone of the remote version – this is called **cloning**.

To make sure your local changes are reflected on Git, you have to **push** these changes. On Repl.it this is done automatically when you commit.

Similarly, if you open an old local repository and the remote has changed (e.g. because someone else wrote a rule) you may need to **pull** and **merge** those changes into your local repository to make sure you are up to date.

The most confusing situation you can find yourself in tends to be where two people are working on the same file and both make changes on the same branch. You then can get what's called a "**merge conflict**" which you have to manually resolve by choosing how to merge both sets of changes. The way we are working should avoid this situation (since everyone works on their own branch), but if you see this happen it's possible – and these aren't very difficult to resolve on GitHub.

For most of this project, these details won't matter too much if you follow the contribution workflow.

If you are interested in reading more about Git and the concepts surrounding how work gets merged, GitHub also has a good guide here https://guides.github.com/introduction/git-handbook/ .

There is also a short tutorial recapping the Repl.it side of things here: https://replit.com/talk/learn/Replit-Git-Tutorial/23331

WRITING A RULE

Quite a big jump for this video, but we're putting all the skills we have together.

Testing and test-driven development

As in the video, we focus on writing tests first before we write the code. This means we can clearly document any edge cases and the outcome we want before we start to write the code itself.

This may initially feel very slow. In my experience, I spend about 75% of my time writing tests and 25% writing code – but the testing process means that the code itself has fewer bugs and works really well.

If you are still keen to play about in a repl.it with an idea for a rule before jumping in and writing it properly, that's of course possible too – you can just make a new Repl and try to write your rule function. This may feel more familiar initially as you transition towards using tests to run your code.

For testing, we use pytest – which is extremely well documented here https://docs.pytest.org/en/6.2.x/ . It has many more features than we'll need to use for this course.

Contribution Steps

There is an enclosed contribution guide which lays out the workflow for writing a rule.

EXERCISES

This week is straightforward – follow the attached contribution guide and write a rule!

Good luck and congratulations on getting to this point! Looking forward to working together.