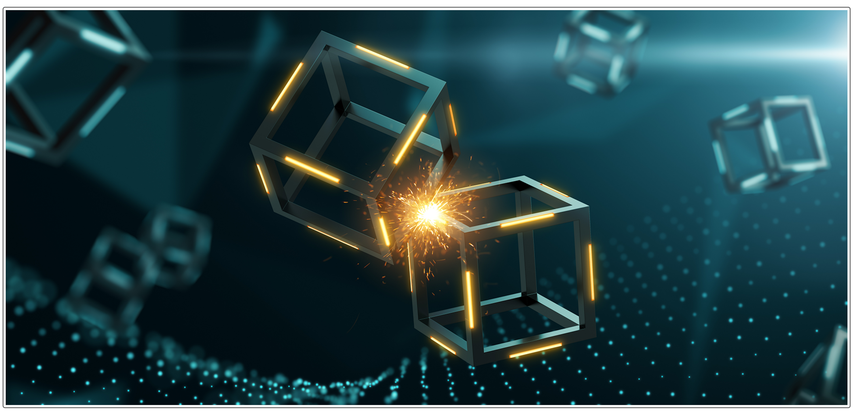
Module 18 Challenge

Start Assignment

Due Wednesday by 23:59 **Points** 100 **Submitting** a text entry box or a website url



Background

You're a fintech engineer who's working at one of the five largest banks in the world. You were recently promoted to act as the lead developer on their decentralized finance team. Your task is to build a blockchain-based ledger system, complete with a user-friendly web interface. This ledger will allow partner banks to conduct financial transactions (that is, to transfer money between senders and receivers) and to verify the integrity of the data in the ledger.

What You're Creating

You'll make the following updates to the provided Python file for this Challenge, which already contains the basic `PyChain` ledger structure that you created throughout the module:

- Create a new data class named `Record`. This class will serve as the template for the financial transaction records that the blocks of the ledger will store.
- Change the existing `Block` data class by replacing the generic `data` attribute with a `record` attribute that's of type `Record`.
- Create additional user input areas in the Streamlit application. These input areas will collect the relevant information for each financial record that you'll store in the `PyChain` ledger.
- Test your completed `PyChain` ledger.

You'll also update the `README.md` file in your GitHub repository to include an explanation of the Steamlit application, a screenshot or video of your deployed Streamlit application, and any other information that's needed to interact with your project.

Files

Download the following files to help you get started:

Module 18 Challenge files

Instructions

Open the provided `pychain.py` file, which you'll use to complete the steps for this Challenge. Notice that the `PyChain` ledger that you built throughout this module already includes the functionality to create blocks, perform the proof of work consensus protocol, and validate blocks in the chain.

The steps for this Challenge are divided into the following sections:

1. Create a Record Data Class
2. Modify the Existing Block Data Class to Store Record Data
3. Add Relevant User Inputs to the Streamlit Interface
4. Test the PyChain Ledger by Storing Records

Step 1: Create a Record Data Class

Define a new Python data class named `Record`. Give this new class a formalized data structure that consists of the `sender`, `receiver`, and `amount` attributes. To do so, complete the following steps:

1. Define a new class named `Record`.
2. Add the `@dataclass` decorator immediately before the `Record` class definition.
3. Add an attribute named `sender` of type `str`.
4. Add an attribute named `receiver` of type `str`.
5. Add an attribute named `amount` of type `float`.

NOTE

> You'll use this new `Record` class as the data type of your `record` attribute in the next section.

Step 2: Modify the Existing Block Data Class to Store Record Data

Rename the `data` attribute in your `Block` class to `record`, and then set it to use an instance of the new `Record` class that you created in the previous section. To do so, complete the following steps:

1. In the `Block` class, rename the `data` attribute to `record`.
2. Set the data type of the `record` attribute to `Record`.

Step 3: Add Relevant User Inputs to the Streamlit Interface

Code additional input areas for the user interface of your Streamlit application. Create these input areas to capture the sender, receiver, and amount for each transaction that you'll store in the `Block` record. To do so, complete the following steps:

1. Delete the `input_data` variable from the Streamlit interface.
2. Add an input area where you can get a value for `sender` from the user.
3. Add an input area where you can get a value for `receiver` from the user.
4. Add an input area where you can get a value for `amount` from the user.
5. As part of the Add Block button functionality, update `new_block` so that `Block` consists of an attribute named `record`, which is set equal to a `Record` that contains the `sender`, `receiver`, and `amount` values. The updated `Block`should also include the attributes for `creator_id` and `prev_hash`.

Step 4: Test the PyChain Ledger by Storing Records

Test your completed `PyChain` ledger and user interface by running your Streamlit application and storing some mined blocks in your `PyChain` ledger. Then test the blockchain validation process by using your `PyChain` ledger. To do so, complete the following steps:

1. In the terminal, navigate to the project folder where you've coded the Challenge.
2. In the terminal, run the Streamlit application by using `streamlit run pychain.py`.
3. Enter values for the sender, receiver, and amount, and then click the Add Block button. Do this several times to store several blocks in the ledger.
4. Verify the block contents and hashes in the Streamlit drop-down menu. Take a screenshot of the Streamlit application page, which should detail a blockchain that consists of multiple blocks. Include the screenshot in the `README.md` file for your Challenge repository.
5. Test the blockchain validation process by using the web interface. Take a screenshot of the Streamlit application page, which should indicate the validity of the blockchain. Include the screenshot in the `README.md` file for your Challenge repository.

Requirements

Step 1: Create a Record Data Class (20 points)

To receive all points, you must:

- Successfully define a new class named `Record`. (10 points)
- Implement the required class attributes for the `Record` class. (10 points)

Step 2: Modify the Existing Block Data Class to Store Record Data (20 points)

To receive all points, you must:

- Rename the `data` attribute in the `Block` class to `record`. (10 points)
- Set the data type of the `record` attribute to `Record`. (10 points)

Step 3: Add Relevant User Inputs to the Streamlit Interface (20 points)

To receive all points, you must:

- Add the correct user inputs for getting the sender, receiver, and amount. (10 points)
- Correctly update the Add Block button functionality. (10 points)

Step 4: Test the PyChain Ledger by Storing Records (10 points)

To receive all points, you must:

- Test the functionality of your chain. In the `README.md` file for your GitHub repository, include a screenshot that contains a blockchain consisting of several blocks. (5 points)
- Confirm that your blockchain is valid. In the `README.md` file of your GitHub repository, include a screenshot of the Streamlit application page displaying "Blockchain is Valid." (5 points)

Coding Conventions and Formatting (10 points)

To receive all points, your code must:

- Place imports at the top of the file, just after any module comments and docstrings, and before module globals and constants. (3 points)
- Name functions and variables with lowercase characters, with words separated by underscores. (2 points)
- Follow DRY (Don't Repeat Yourself) principles, creating maintainable and reusable code. (3 points)
- Use concise logic and creative engineering where possible. (2 points)

Deployment and Submission (10 points)

To receive all points, you must:

- Submit a link to a GitHub repository that's cloned to your local machine and that contains your files. (4 points)
- Use the command line to add your files to the repository. (3 points)
- Include appropriate commit messages for your files. (3 points)

Comments (10 points)

To receive all points, your code must:

- Be well commented with concise, relevant notes that other developers can understand. (10 points)

Submission

To submit your Challenge assignment, click Submit, and then provide the URL of your GitHub repository for grading.

NOTE

> You are allowed to miss up to two Challenge assignments and still earn your certificate. If you complete all Challenge assignments, your lowest two grades will be dropped. If you wish to skip this assignment, click Next, and move on to the next module.

Comments are disabled for graded submissions in Bootcamp Spot. If you have questions about your feedback, please notify your instructional staff or your Student Success Manager. If you would like to resubmit your work for an additional review, you can use the Resubmit Assignment button to upload new links. You may resubmit up to three times for a total of four submissions.