MockBlock
Due: February 14, 2019

Contents

1 Introduction 1
2 Set Up 1
3 Stencil 2
   3.1 blockchain.py 2
   3.2 network.py 2
4 Assignment 2
   4.1 Block 2
   4.2 Blockchain 3
   4.3 Transactions 4
   4.4 Questions 4
5 Demo 5
6 Grading 5
7 Handing In 5

1 Introduction
Welcome to MockBlock! In this project you will be creating a centralized blockchain that can store transaction data. You will be able to interact with your blockchain through an interface provided for you.

2 Set Up
You will be implementing a simple blockchain in Python. To get started, run

cs1951l_install MockBlock

to copy the stencil for this project into /course/cs1951l/MockBlock. After 1951, make sure you type a lowercase letter ‘L,’ not a number 1.
3 Stencil

3.1 blockchain.py

You have been provided with the following stencil file:

blockchain.py - contains the Blockchain and Block classes with function definitions
that you will need to implement

3.2 network.py

You are also provided with the file network.py which will run your implementation
of MockBlock and provide you with an interface to test your blockchain. To use the
provided interface, you will need to install flask by running the following command:

    pip install flask

To run the program, use:

    python network.py

and open it in your browser with the url http://127.0.0.1:8000/

4 Assignment

You will be implementing the functions inside of blockchain.py. These functions are
within two classes in the file. You will also have to answer questions provided to you as
well as provide a summary of how your code works. You are also expected to comment
each function and any helpers you may have implemented.

4.1 Block

    def _init_(self, index, data, previous_hash):

A block is an object containing transaction information on the blockchain. You must
initialize the following parameters

    • The index of each new block on the blockchain
    • A timestamp of when the block was created using the time() function
    • The hash of the previous block
    • A list of transaction data
    • The nonce for proof of work calculations
def compute_hash(self):
The hash of each block is based on the hash of the previous block as well as the data of
the current block. Our blockchain will use the SHA256 hash algorithm imported from
the hashlib library.

You will need to use the SHA256() method on the hasher object and then update your
result with a concatenation of the index, timestamp, previous hash, data, and nonce.
Each field should be converted to a string before concatenated. The method should
return a string of hexadecimal digits, which can be done by running the method hexdi-
gest() on your updated hash.

4.2 Blockchain
def _init_(self):
The blockchain is the network of blocks containing all the transaction history. Your
implementation of the blockchain must include the following parameters:

• A difficulty representing the number of leading zero bytes the hash will produce
given the nonce of the block. This should be initialized as 4

• A list of unconfirmed transactions that haven’t been added to a block yet

• A list of all the blocks

Upon initialization, the blockchain should also mine the genesis block, the first block
of the blockchain.

def create_genesis_block(self):
The genesis block is the first block of your blockchain. The block should be initialized
with an index of 0 and a previous hash of 0 and contains no data. Make sure to perform
proof of work and that the block is added to the blockchain.

def new_transaction(self, sender, recipient, value):
Create a new transaction and add it to the list of unconfirmed transactions as a dictio-
nary. The transaction should contain information on who the sender and recipient of
the transaction is and the value of the transaction. The function should return the new
transaction object

def mine(self):
Mining is the process of adding new blocks to the blockchain by confirming transac-
tions. Our blockchain will use proof of work to confirm each transaction.

Your function should create a new block, perform proof of work on the new block, and
then add the newly confirmed block to the blockchain. The function should return the
new block.
proof_of_work(self, block):
Proof of work checks that a node has performed the necessary calculations to confirm a
block by brute forcing the solution to a particular output of an encrypted hash function.

To perform proof of work, you have to first produce a hash for the given block. You
must then verify that the hash produced using the current nonce of the block has the
correct amount of leading zeroes, which for this assignment we have specified as 4.
For each degree of difficulty, the average time to perform a proof of work doubles.

The function should return the desired hash.

def add_block(self, block):
This function will add the given block to the blockchain. Before doing so, you should
verify that the hash of the block before the given block is equal to the hash of the last
block in the blockchain. If the hashes are not equal, the function should return false.
Otherwise, upon successful execution, the function should return true.

def check_integrity(self):
To help check your progress, we will require you to implement a function to check the
integrity of the blockchain. A valid blockchain must meet the following criteria:

• Each block is indexed one after the other
• Each block’s previous hash is the hash of the previous block
• The block’s hash is valid given the set difficulty

4.3 Transactions
You are also given an already implemented Transactions class. This class should be
used when new transactions are created.

4.4 Questions
You’ll find the written questions in the file named questions in your database folder.
Answer the provided questions in this file and make sure it is included when you hand
in. For reference the questions are:

1. Explain the definition, purpose and significance of both the nonce and the dif-
culty level in performing proof of work calculations. In particular, as more
miners join the network explain how the unit of difficulty is used to ensure a
consistent interval of valid blocks.

2. Consider and discuss the benefits and downsides to using proof of work in blockchain
networks. What are some alternatives to proof of work consensus? List at least
three, and discuss the benefits/downsides of each.
5 Demo

We have prepared a demo so you can see how your final implementation should behave. To run it, use the command:

```
cs1951l_MockBlockDemo
```

The demo allows you to send a transaction between two parties. You will see this transaction in the unconfirmed transactions table and clicking Mine will confirm it and add it to the blockchain. You can also test the validity of the blockchain and reset it by clicking the Check Validity and Reset Blockchain buttons.

6 Grading

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blockchain</td>
<td>67%</td>
</tr>
<tr>
<td>Block</td>
<td>9%</td>
</tr>
<tr>
<td>Style and Comments</td>
<td>9%</td>
</tr>
<tr>
<td>ReadMe and Questions</td>
<td>15%</td>
</tr>
<tr>
<td>Total</td>
<td>100%</td>
</tr>
</tbody>
</table>

Your project will be graded primarily on the functionality of the functions within your Blockchain and Block classes. Your answers to your questions in your README, your description of your implementation, and your comments of the functions will also factor into your grade.

7 Handing In

To hand in the project, run

```
cs1951l_handin MockBlock MockBlock
```