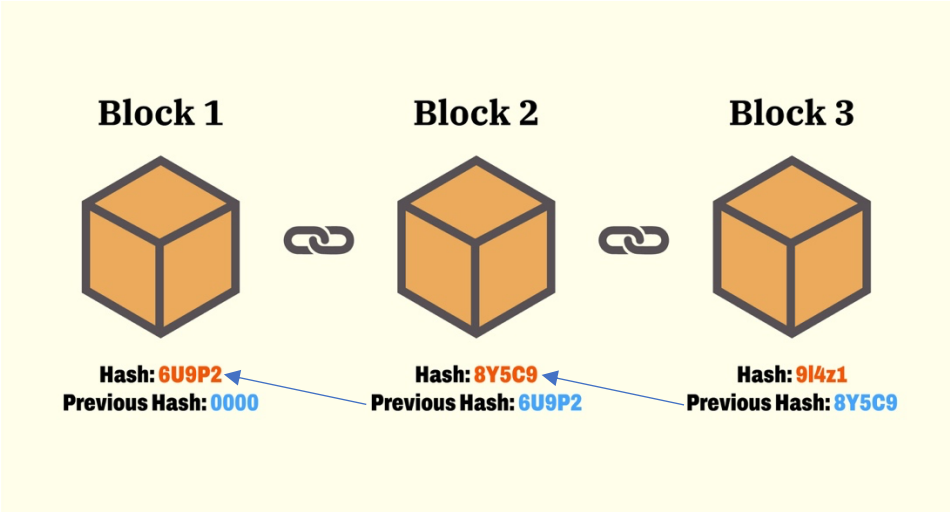
Homework #8 – Blockchain

Context

In the aftermath of world's economic collapse in 2008, a white-paper was pseudonymously published under the name of Satoshi Nakamoto that called for a new way of exchanging value. Bitcoin is peer-to-peer electronic cash system that uses hashing technology to create a decentralized bank leger. In this assignment, we will recreate aspects of a blockchain as a real-world example of hashing.

A blockchain is a chain of blocks. A block is simply a grouping of data. And if we place the hash of one block into the data of the next block, then we have created a linkage that chains our blocks together. Furthermore, since our hashing algorithm is deterministic (given the same input always produces the same output), there is a one-to-one relationship between the data values in a given block and the corresponding hash of that block. That also means any changes to the data inside the block will result in a new hash value for that block. In that way, the data in a blockchain cannot be changed without easy detection.

In the diagram below, notice how the hash of block 1 (6U9P2) is stored as a field called "Previous Hash" within Block 2. Likewise, notice that the hash of block 2 (8Y5C9) is stored as a field called "Previous Hash" within Block 3. (Does this remind you of a concrete data structure we have studied in the past?)



https://money.com/what-is-blockchain/

Traditional banks keep a ledger of all deposits and withdrawals. Regardless of how much money you believe you have in your account, the bank (and their centralized ledger) is the ultimate authority for the correct balance.

Bitcoin disintermediates the concept of a centralized bank. In other words, it removes all the "middlemen" from the transfer of currency or value, including national governments. Instead, the "ledger" is kept by all the peers or nodes in the network. But for that design to work, there must be a way to keep all the ledgers in sync and prevent them from being tampered with. (Otherwise, nefarious users would simply alter their copy of the ledger to give themselves more currency!)

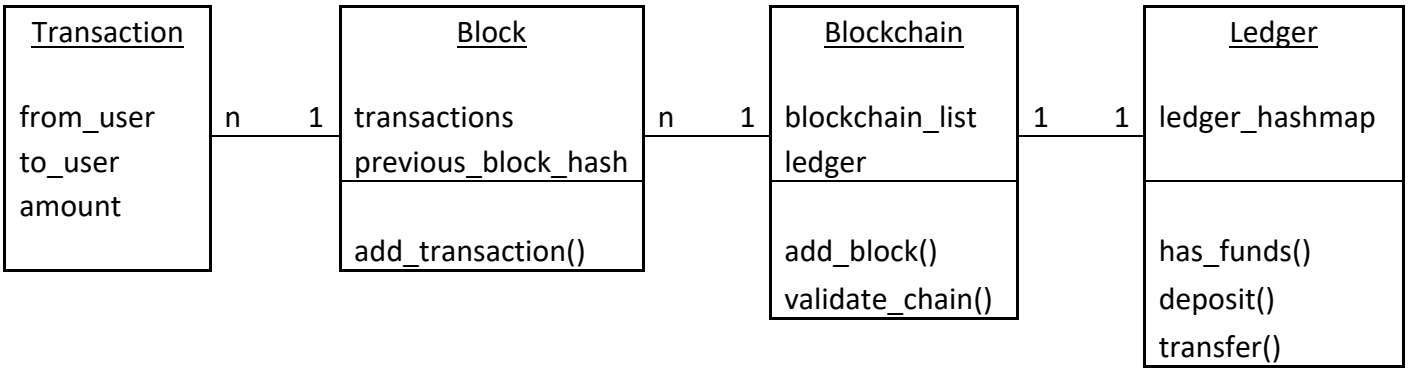
In this assignment, we will not attempt to simulate the entire blockchain network. But we will examine how hashing is used as a key aspect of a blockchain. And you will create code for some of the checks-and-balances that are common to a single blockchain node.

Your Assignment

You will create some simple blockchain code. But instead of bitcoin tokens, you will create HuskyCoin! This assignment is evaluating you on two specific topics. (1) Can you create and use a Hashmap implementation? (2) Do you understand what hashing is and how to use it?

Here are your specific tasks:

1. Create an implementation for the Hashmap ADT. Feel free to create a wrapper class that uses any concrete data structure. Place your class in "hashmap.py".
2. Create four classes according to the following class diagram and place them in the provided "blockchain.py". (You will notice some of the code has already been provided to get you going. Do not modify any of the provided code.)



(Note: the lines between the rectangles show the relationships between the classes. Notice the presence of "n" and "1" on each line. For instance, 1 Block can have n number of Transactions. Each Blockchain has n number of Blocks. Each Blockchain has only 1 Ledger.)

- Transaction
  - Contains a single transaction of HuskyCoin: the sending user, the receiving user, and the amount being transferred.
- Block
  - Contains a collection of Transaction objects. Each block can contain N number of transactions.
  - Contains the hash of the previous block in the chain.
  - add_transaction() - a method for adding a single transaction to the block.
- Ledger

        - Contains an instance of your Hashmap ADT that will be used to keep track of user balances
        - has_funds() – does the sending user have enough HuskyCoin to send the amount they are trying to transfer?
        - deposit() – adds an amount of HuskyCoin to the given user
        - transfer() – subtracts an amount of HuskyCoin from the given user
    - Blockchain
        - Contains a List of Block objects
        - Contains a Ledger instance
        - add_block()
            1. Store the hash of the previous block in the block being added
            2. Make sure users transferring HuskyCoin have the required balance. Do not allow the block to be added if any given transaction does not have required user funds. If any transaction is invalid, either throw an error or return False. But your other code must handle this possibility gracefully.
            3. Update the ledger of balances
            4. Return True/False value indicating whether the block was successfully added to the chain
        - validate_chain()
            1. Check to see if any of the blocks have been "tampered" with by comparing the hashes of each block in the chain are still correct.
            2. In your testing of this method, verify that if you try and modify any data value with a given transaction with a given block, that the validate_chain() method will detect the modification.
            3. Return a list of blocks that have been "tampered" with.

3. Implement the __hash__ and __eq__ methods as appropriate to whichever of the four classes above require it.
4. Add methods to your classes to print out the chain of blocks and the ledger of transactions to the console.

Additional Assumptions, Tips, and Useful Information

- In Bitcoin, the concept of a "userid" is simply a wallet address (a long series of random characters). This gives Bitcoin a level of anonymity. (Though the IRS has still managed to track down the real-life owners on Tax Day!) In this assignment, you may use simple strings such as "Ali", "Bob", or "Maya" for your user names.
- In Bitcoin, the timestamp of a transaction is important. But for simplicity in this assignment, we will not consider "timestamp." Therefore, order of transactions within a given block don't matter. When you check for whether a user has enough funds to perform a transfer, consider all transactions within a block as being processed at the same moment in time.
- The first block in a blockchain is called the "Genesis Block" and is handled slightly different than all the other blocks. For instance, the "Previous Hash" value will be None. You will notice that the code for

creating the genesis block has been provided so that we can focus on the salient parts of this assignment.

We hope you have fun with this assignment and learn a little bit about blockchain and foundations of cryptocurrency in the process. But please do not get overwhelmed by the use case. Blockchain is a complicated topic. But you really only need to understand hashing and hashmap to succeed on this assignment. You can do your own research on blockchain, but that additional knowledge is not required for this assignment.

However, blockchain reminds us that understanding a topic like hashing could one day make you one of the richest people in the world.

Submitting

At a minimum, submit the following files:

- blockchain.py (four classes to support your blockchain implementation)
- hashmap.py (your implementation of the Hashmap ADT)
- test_blockchain.py (all your test cases)

Additionally, you must include any other files necessary for your code to run, such as modules containing data structures you wrote yourself.

Do not import any modules you did not write yourself. The functionality added by external modules is one of the great things about python, but we restrict them for two reasons:

- Some of them circumvent the purpose of these assignments, which is to develop basic algorithm design habits
- Some of them are not supported by our Python install on Gradescope, making it impossible for us to run your code there (and thus slowing down our grading significantly)

Students must submit **individually** by the due date (typically, Tuesday at 11:59 pm EST) to receive credit.