Python version: 3.6.5

**Objective:** These programs allow the user to add new transactions to a transaction file, after which a SHA256 hash of the transaction block is generated and the new block is chained to the previous block in a block chain. The Transaction Recording Program (TRP) allows the user to enter new transactions made up of sender, recipient, and amount, saving each transaction in the Transaction File until they wish to quit. The Block Mining Program (BMP) continuously monitors the Transaction File and adds new blocks for any new transactions founds in the file. Using an incrementing nonce value, it generates SHA-256 hashes until a hash with 14 zeros is created, which is then stored as the new hash for the block and is added to the block chain in the BlockChain file, using the previous block’s hash as a reference.

**Program Implementation:**

**Block Mining Program:** The program is set in a loop that gives the user the option to quit the program from the main menu. A variable *first\_trans* is set to True, the Transaction File.txt (TF) file is created, and the user is given the option to quit the program or continue. The program then enters a loop which gives the user the option to return to the main menu between creating new blocks. The BlockChain.txt (BC) file is opened in append mode and if no previous hashes of blocks exist, a new **Block** is created with an index of zero, current timestamp, data consisting of the words ‘first block’, and a previous block’s hash value of zero. This is new **Block** is set to the variable *last\_block* and written to the BC file, then the file is closed.

The program will use the ‘getsize’ function from the inbuilt Python OS module to determine the amount of data in the TF file (Quadri, 2015) and if the size equals zero, the user will be prompted to create a transaction using the TRP before proceeding, then the loop is broken, and the user is returned to the main menu.

The last transaction number is read from the TF file and saved to a variable *last\_trans*. The user is then given the option of returning to the main menu or continuing with the program; if they choose to quit the loop is broken, and the user is returned to the main menu. Else a *timer* variable is set to the current time and then the program enters a loop to allow for continuous monitoring of changes to the TF file. The line containing the last transaction number is once again read from the TF file and saved to a variable *current\_trans*. If *current\_trans* is equal to *last\_trans* AND the *first\_trans* variable is set to False; no changes have occurred, and it is not the first transaction in this instance of the program, so the timer advances 15 seconds before looping back to check the file again. If they are not equal or the *first\_trans* variable is True, a new transaction has occurred. The BC file and TF file are then opened in read mode and the last three lines containing the data (i.e. the sender, recipient, and amount) from the last transaction are read from the TF file and saved to a variable *transaction* (Ritzel, 2011). The index and hash of the last recorded block are also read into the variables *last\_index* and *last\_hash* from the BC file. The ‘*block\_mining’* function is then called, using the last\_indexand last\_hashof the previous block, along with the transactiondata from the TF file as the function parameters. After the function ends, the *first\_trans* variable is set to False and the program returns to the beginning of the loop, where it once more asks the user if they want to quit or continue checking the file for updates.

Block Mining function: This function uses three parameters of last\_index, last\_hash and transaction, to creates a new **Block** object; calculating the new block’s index by incrementing the index from the *last\_index* variable, creating a timestamp with current date and time, using the data drawn from the TF file in the *transaction* variable, and recording the last block’s hash value from the *last\_hash variable,* which is used to link the block into the blockchain. A *nonce* variable is set to zero and then the function enters a loop which creates a SHA256 hash of the new **Block**’s data, the previous block’s hash, and the nonce value. If the hash of this block has 14 zeros, the BC file is updated with the new hash and the function ends. Otherwise the *nonce* value is incremented, and the loop continues. If the *nonce* value reaches 50000, the BC file is updated with the last calculated hash and the function ends.

**Block** class: This allows the creation of **Block** objects (Nash, 2017), which always have four parameters: an index, the current date and time, the transaction data from the Transaction File.txt file, and the hash of the previous block. It also includes a *self.hash* variable which uses the class’s defined ‘*SHA\_hash’* function (Individual contributors, 2018) to return a SHA-256 hash value for the data of the **Block**.

**Transaction Recording Program:** The program is set in a loop that gives the user the option to quit the program after each new transaction is created. The Transaction File.txt (TF) is opened in append mode, as it will have been created prior to this by the BMP program, then the rest of the program is set in a loop to allow the user to create multiple transactions without returning to the menu. The user is asked whether they want to start a new transaction.

If they choose [y], the defined function *‘user\_string\_input’* is called, which prompts them for an input of alphabetic characters only, for the name of both the sender and the recipient. Following that, the defined function *‘user\_int\_input’* is called, which prompts the user for an input of numeric characters only, for the amount they wish to send. A *timestamp* variable is also created using the current date and time. The program will use the ‘getsize’ function from the inbuilt Python OS module to determine the amount of data in the TF file (Quadri, 2015) and if the size equals zero, the *index* variable is set to one. Else, the *last\_index* is read from the TF file and the *index* variable is set to the *last\_index* plus one. All the input variables are combined as a string into one *transaction* variable. The TF file is then opened in append mode, the transaction is written to the file, and then the file is closed and the program loops back to ask the user if they wish to start a new transaction.

Else if they choose [n], they are given the option to either return to the main menu by breaking the loop or continuing the loop to start a new transaction. If the user does not enter one of the two given options, they are prompted to enter a valid option and the loop continues, returning to the prompt.

**References**

Individual contributors., (2018). ‘SHA-256’. PyCryptodome. Retrieved from http://pycryptodome.readthedocs.io/en/ latest/src/hash/sha256.html

Nash, G., (2017). ‘Let’s Build the Tiniest Blockchain in Less Than 50 Lines of Python’. Medium Corporation. Retrieved from https://medium.com/crypto-currently/lets-build-the-tiniest-blockchain-e70965a248b

Quadri, M., (2015). 'How to check text file exists and is not empty in python'. StackOverflow. Retrieved from https://stackoverflow.com/questions/28737292/how-to-check-text-file-exists-and-is-not-empty-in-python/28737358

Ritzel, J., (2011). ‘Removing character in a list of strings’. StackOverflow. Retrieved from https://stackoverflow.com/ questions/8282553/removing-character-in-list-of-strings

Rove, D., (2014). ‘What is the best way to repeatedly execute a function every x seconds in Python?’. StackOverflow. Retrieved from https://stackoverflow.com/questions/474528/what-is-the-best-way-to-repeatedly-execute-a-function-every-x-seconds-in-python