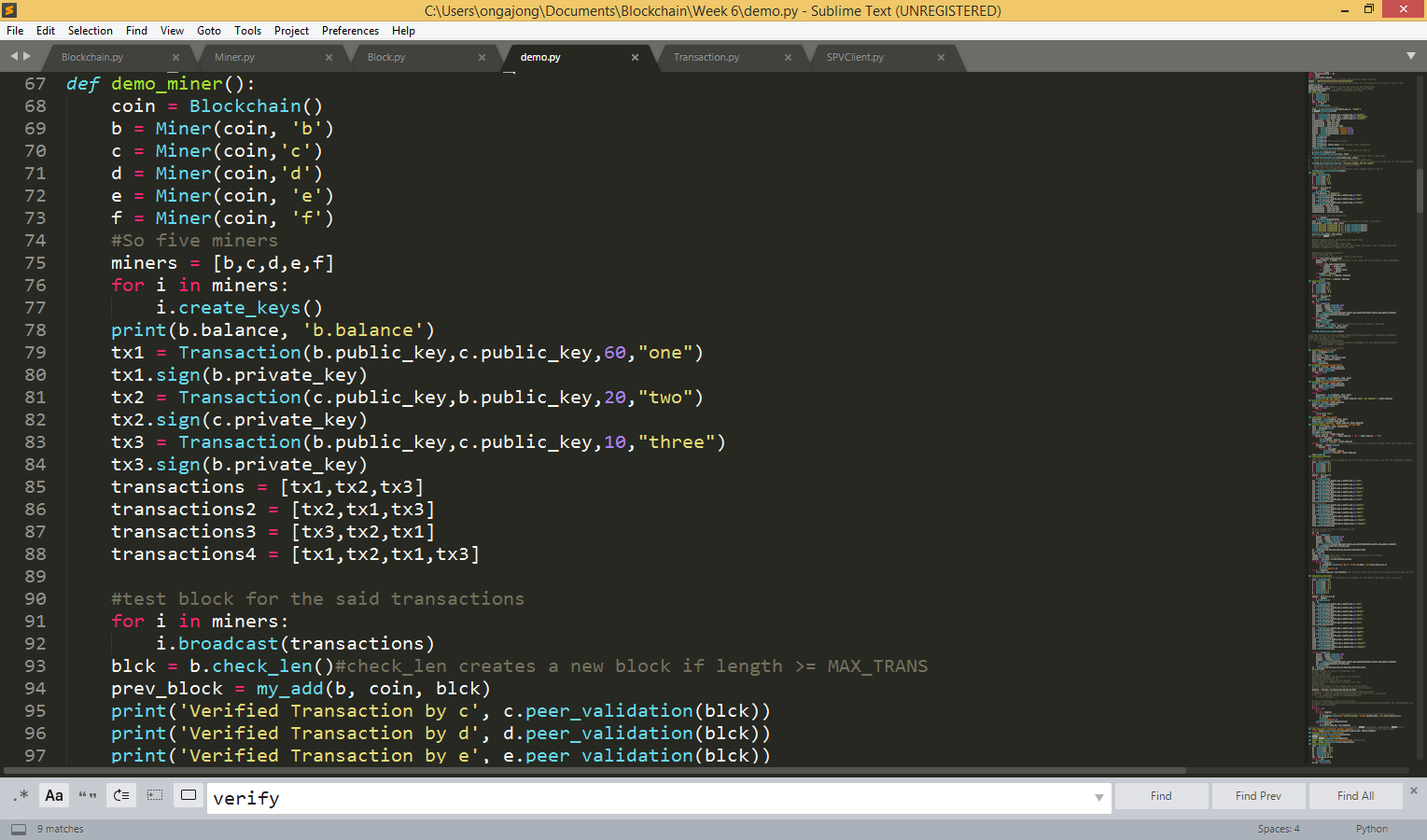
**Steps to Reproducing the Demo:**

1. Mining and coin creation:

In demo.py, run the test program (demo\_miner).



Blue box: creates instances of five miners as well as their keys

Line 78: prints the balance of miner ‘b’

Lines 79- 84: signs the three transactions.

Lines 85-88: Creates a list of the same three transactions

Lines 91-92: Broadcast ‘transactions’ to the five miners

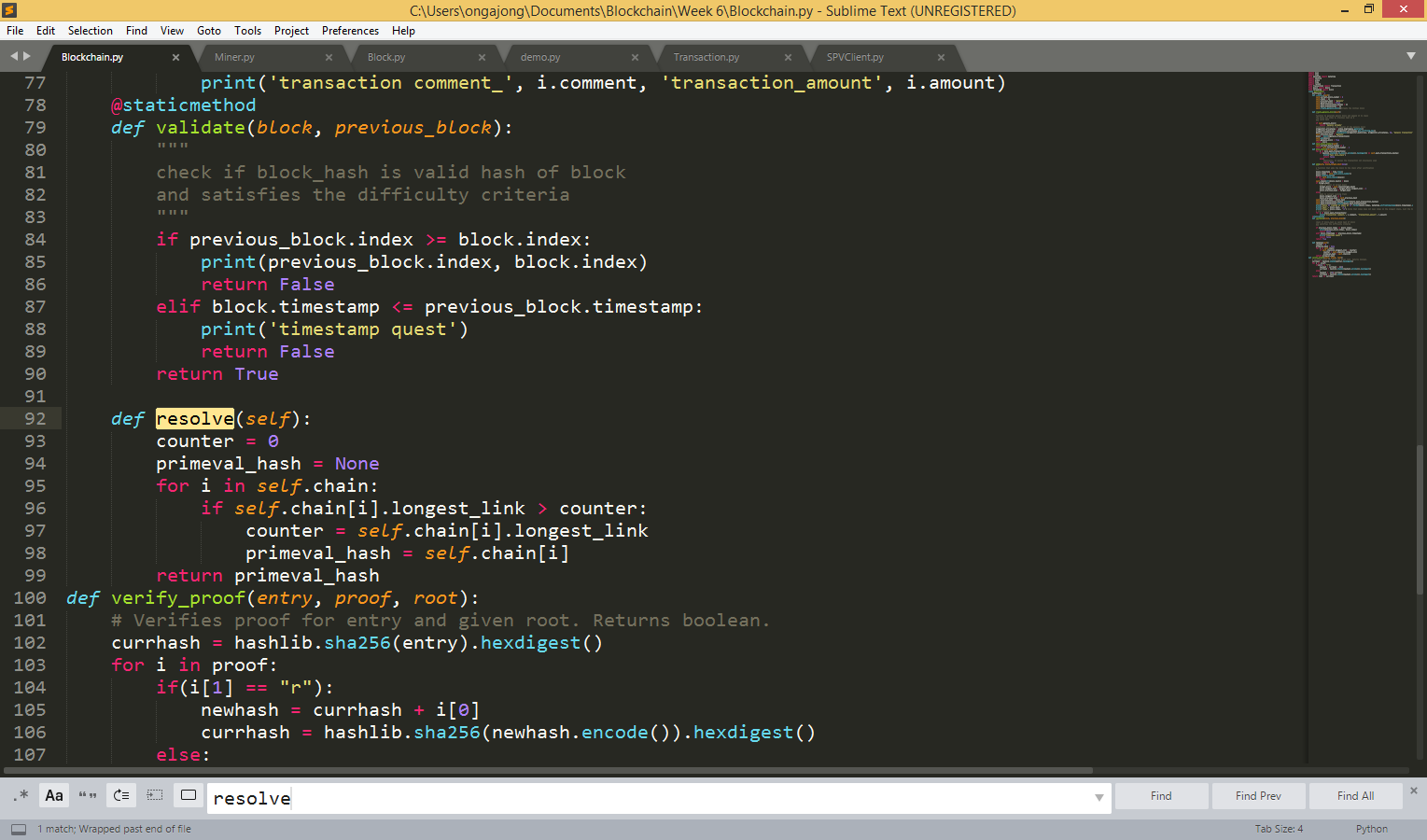
Line 93: b creates a new block (by mining it and doing the proof of work)

Line 94: Runs my\_add (within this block) Fork gets resolved.

Lines 95- 98: Run validation of peers to confirm that the checks were valid.

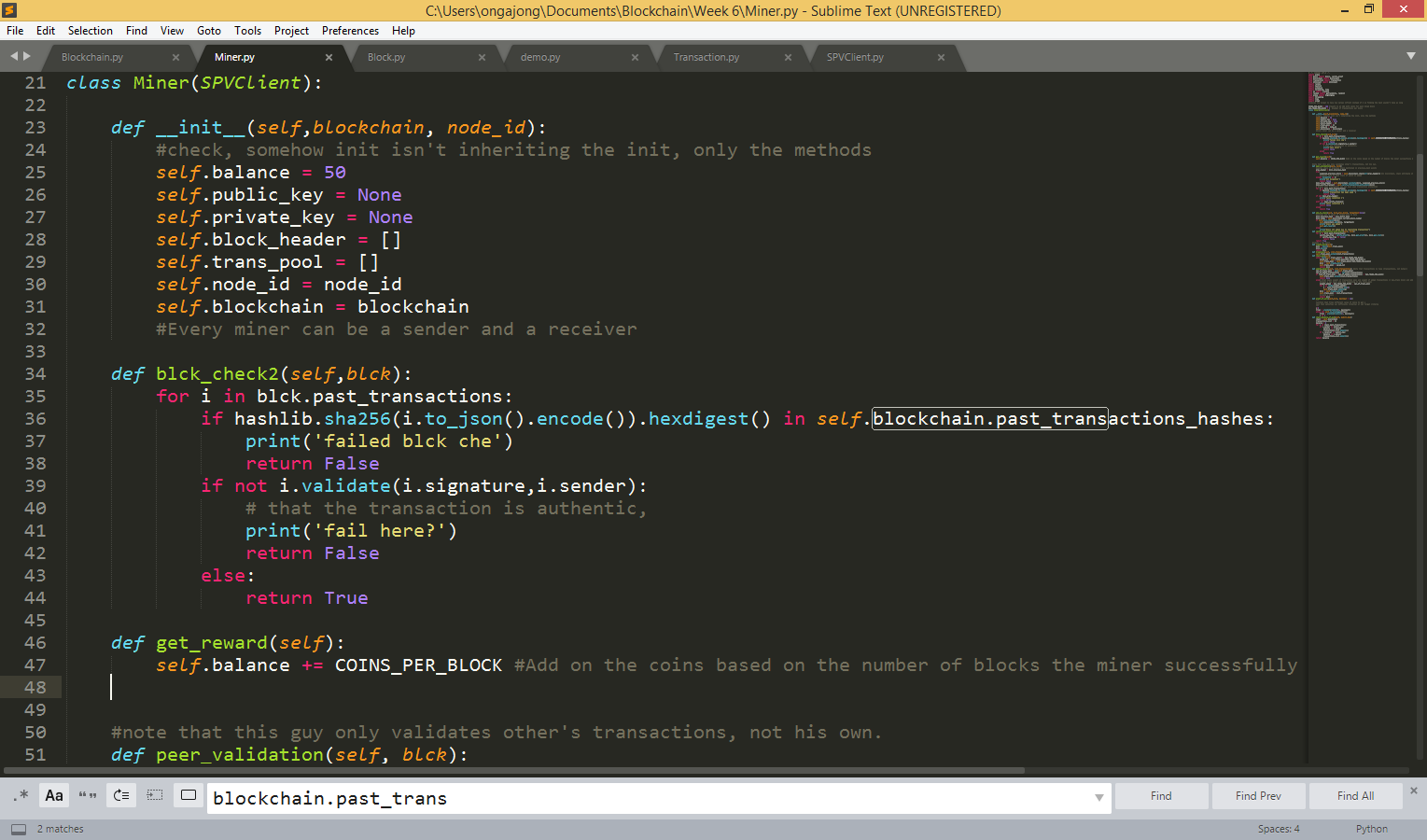
Line 100: Adds to chain.

1. Fork Resolution

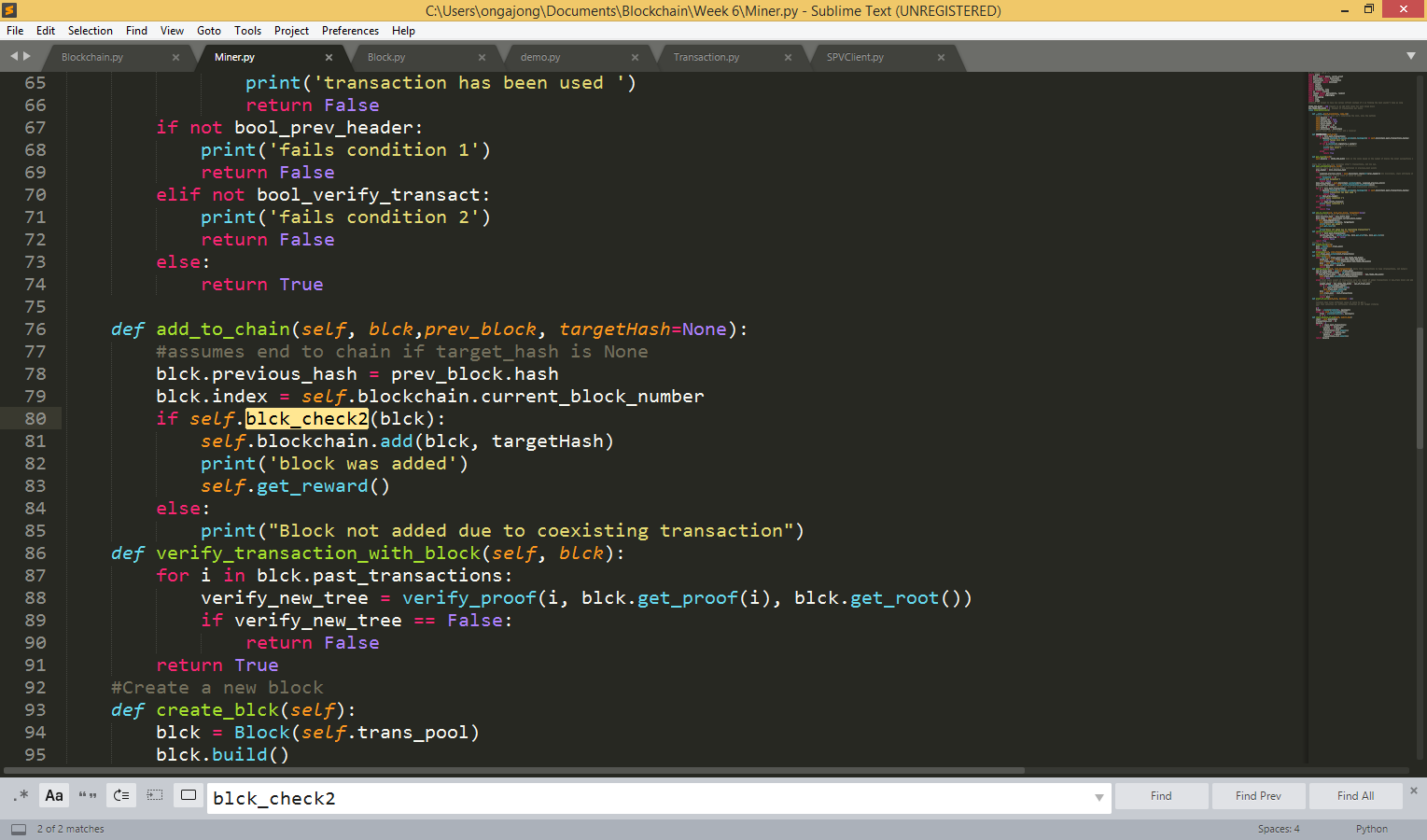


In Blockchain.py, resolve is called from the previous demo\_miner example. Each block stores the attribute known as longest\_link. This resolve function runs the longest loop, and gives back the hash value of the last block in the longest chain

1. Transaction resending protection (to verify that the transaction has not been reused)

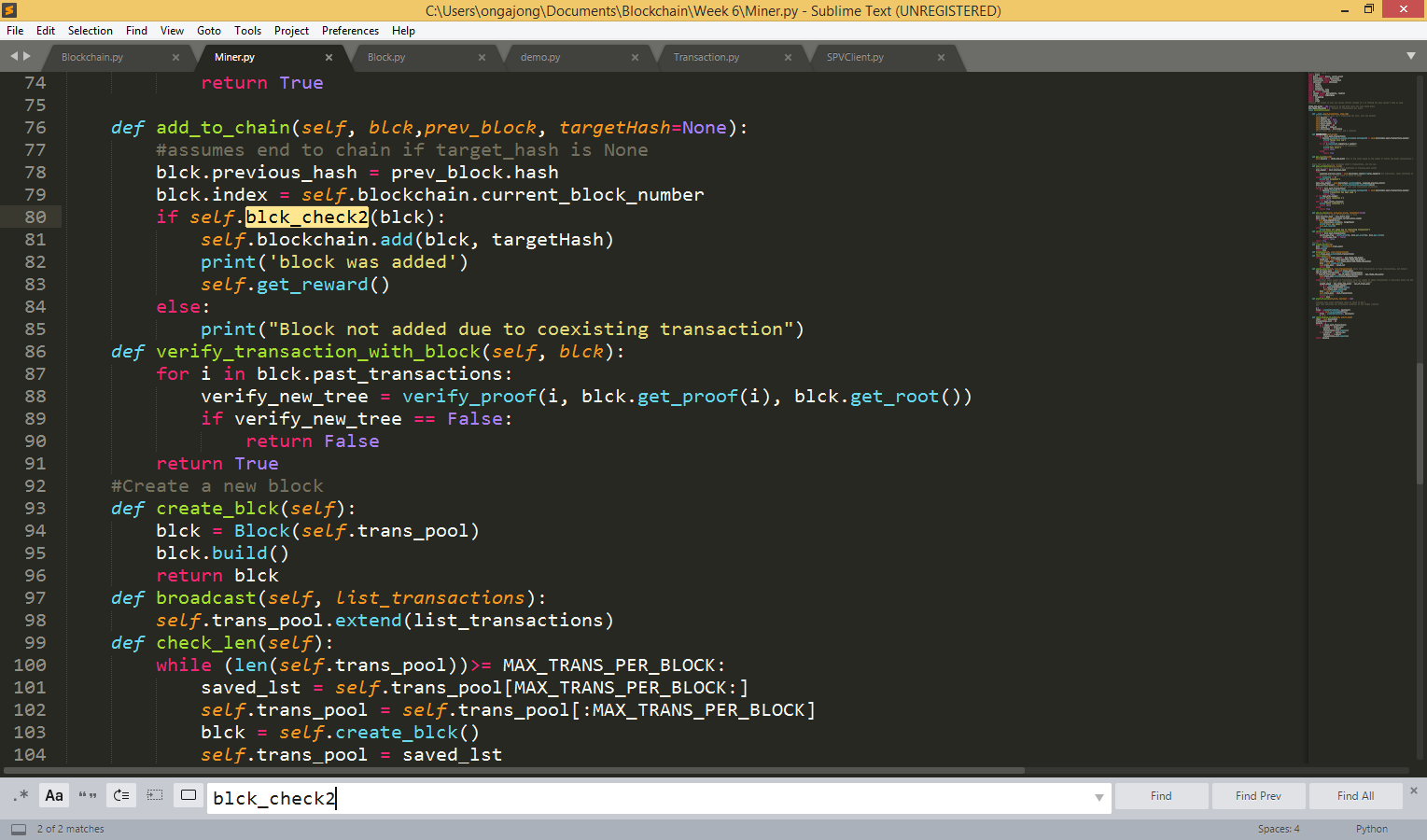


As seen in blck\_check2, each chain stores its past\_transactions hashes. If a transaction is in, then it would return False.



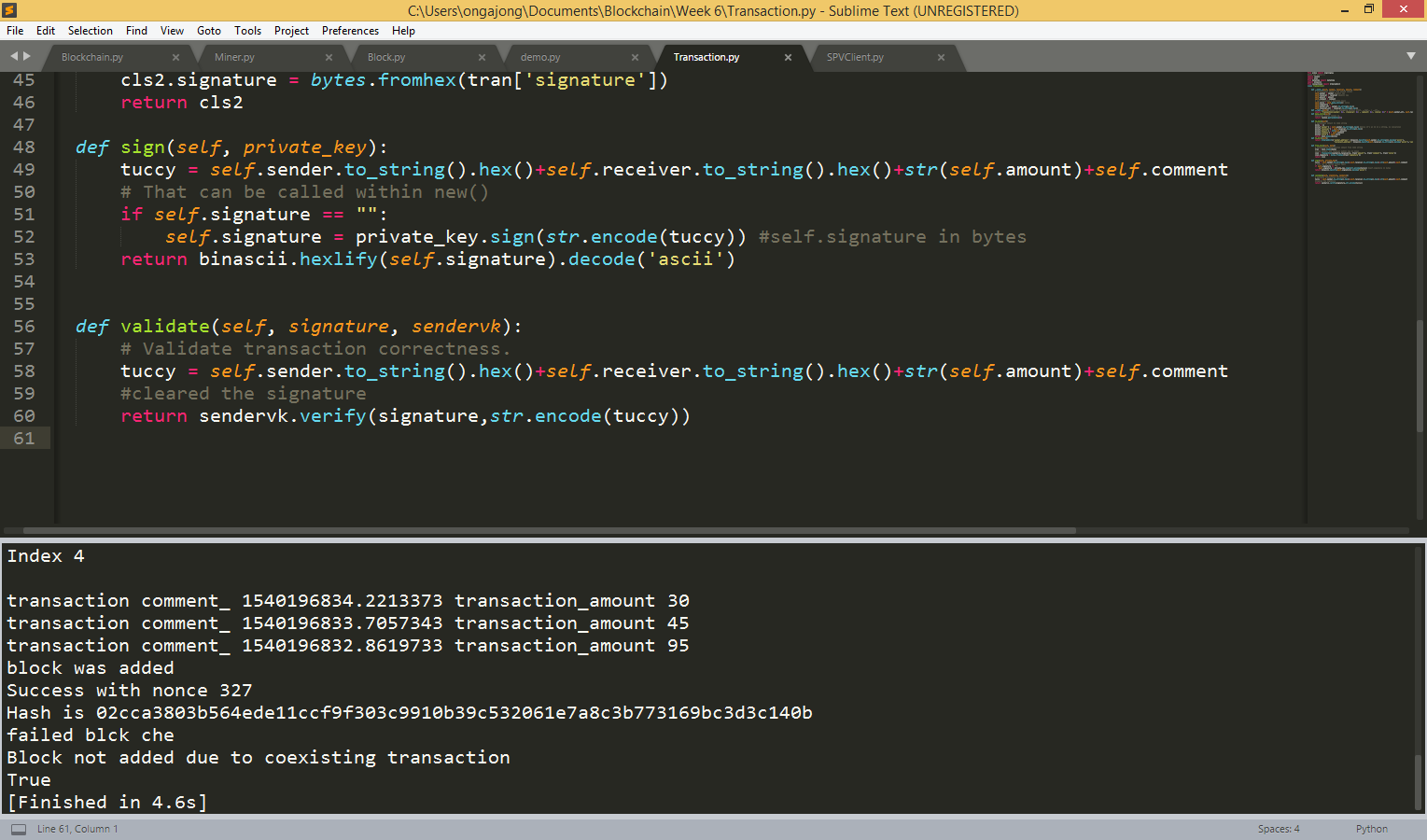
This check is run when add\_to\_chain is called by the Miner. The miner gets the reward once it has been successfully added.

1. Payments between miners and SPV Clients
   1. Transaction validation (for miners)



As can be seen in the blue box, verify\_transaction\_with\_block calls upon verify transaction to see if the transaction is stored within the block. If true (that is, can find the transaction within the Merkle Tree), the miner can validate that this transaction is within the block and pass it on the clients upon request.

* 1. Transaction validation (for SPV Clients

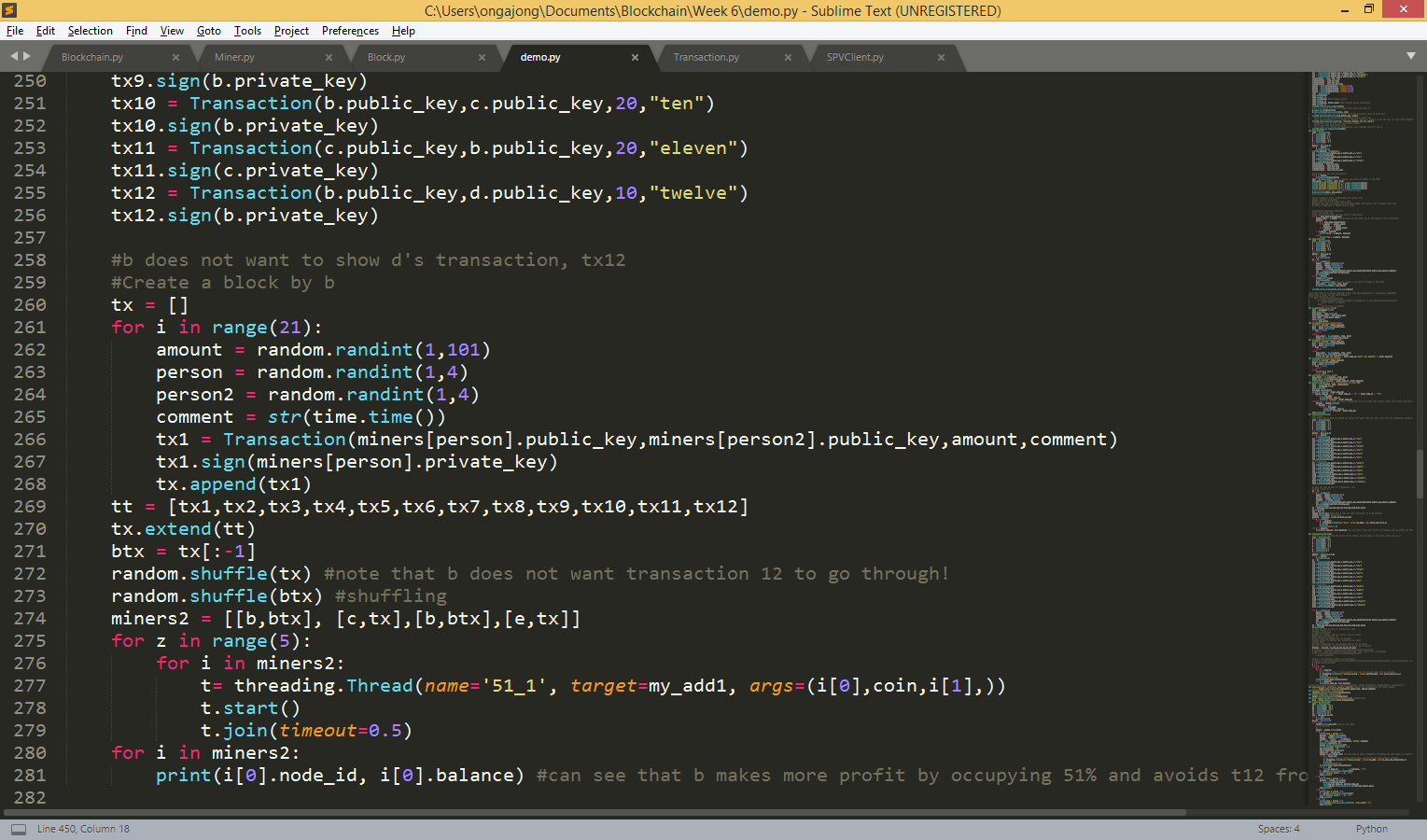


This validate function in the transaction class can be used to validate (or certify that certain transactions go through) with an SPV client

Assume that this validation by clients means: check that the transactions are validated by the SPV Client.

1. The attacks
   1. 51% mining attack

Run simulation\_attack()



Lines 222-270: Creates a list of transactions.

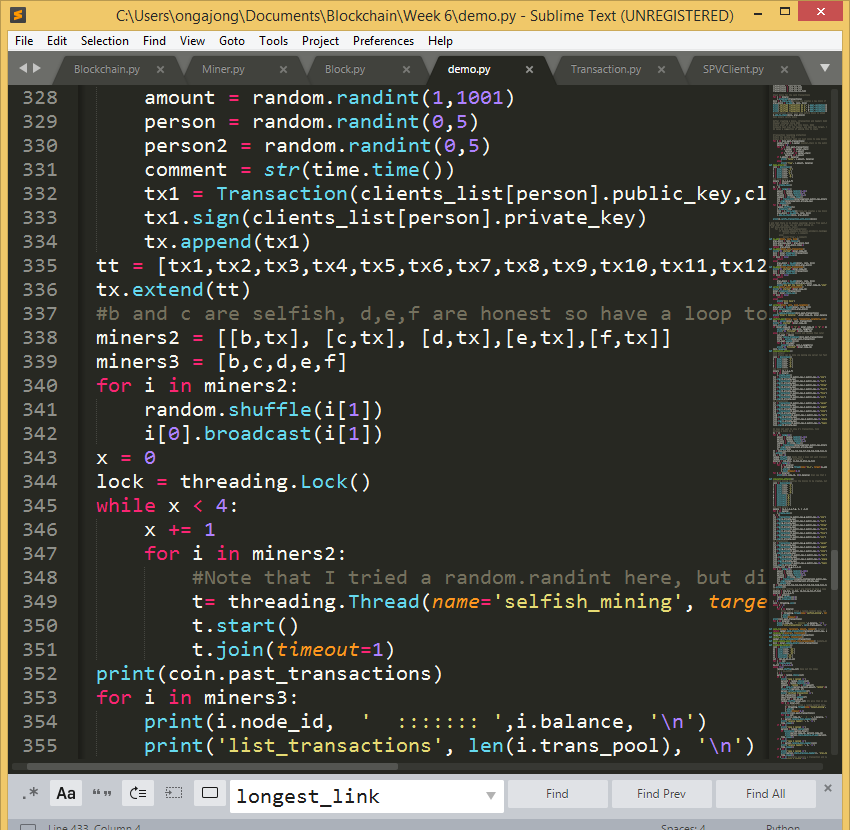
Line 271: Transaction 12 is a transaction that b does not want to put in.

Line 274: After shuffling about, because b has a list of transactions without tx12, we show that it has 50% of the computing strength by having 2 b and other players c and e.

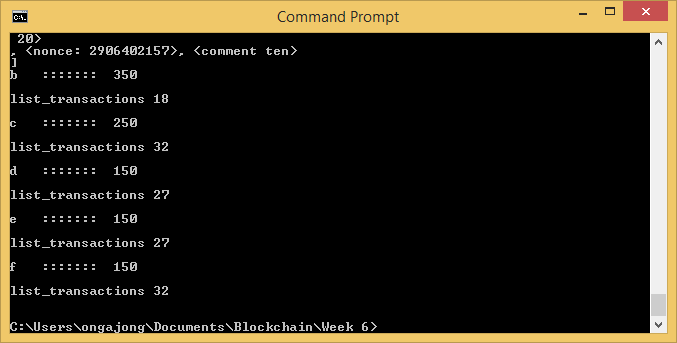
For five iterations, we create a thread, run my\_add1 for the miners to start mining. Because of the Python threading module, each miner is given the same amount of time, but with b appearing twice, it means b has twice the amount of processing power as compared to the other miners. We see that b makes more profit by occupying 51% and t12 is avoided from being added in the block

* 1. Selfish Mining Attack

Run simulation\_attack2()

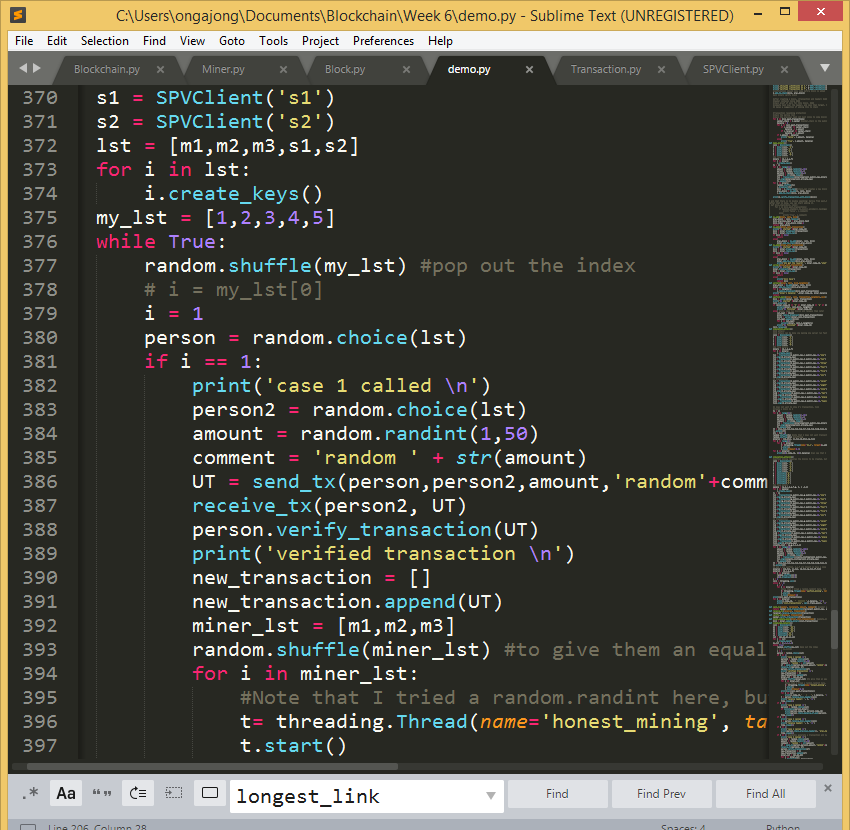


B and C are the selfish miners. With the same threading module we used earlier, we run the function selfish\_mining, which is included in the demo.py file.



We see the difference here: while d, e and f are mining honestly (that is to say, once it finishes creating a block, it adds instantly) while b and c are not adding honestly (here, we only clear the list of blocks once it starts adding everything in its list of kept blocks), b and c end up having a majority of the wealth.

1. Network model:
   1. Run scheduled\_network() where it cycles through a list of functions carried out by three miners and two SPVClients. As SPVClient’s methods is inherited by Miner, all five runners can run SPVClient functions.



**Differences between Bitcoin and SUTDCoin:**

SUTDCoin does not use a bloom filter to check if the transaction is present. Instead, it requests for access to a blockchain’s past transactions to verify the proof that the transaction exists.

SUTDCoin does not request a third party miner to download the headers, but accesses via a local copy of the Blockchain.

SUTDCoin operates on a balance and address model while Bitcoin operates on a UTXO model. While it is possible for SUTDCoin to operate on a UTXO model as the method for checking one’s balance based on the internal list of transactions conducted by the SPVClient, it is far more convenient for the SPVClient node to refer to one’s balance attribute.