**Blockchain Report:**

**Libraries Imported:**

1. **Time:**
   * 1. Imported to calculate the time required to find nonce.
2. **Pickle:**
   * 1. Imported to save and load transactions as a **.dat** file.
3. **Random:**
   * 1. Used to randomly search for nonce.
4. **Hashes:**
   * 1. Used to create a SHA256 hash of a block based on the data and hash of the previous block.
5. **Serialization:**
   * 1. Pickle is not able to import Private and Public key classes as they are FFI (Foreign Function Interface). So serialization is used which helps express Public and Private keys as public and private bytes.
6. **Default Backend:**
   * 1. Used to pass default RSA encryption.

**Python Programs:**

1. **Signatures.py**

The various function involved here are:

* 1. **generate\_keys():**

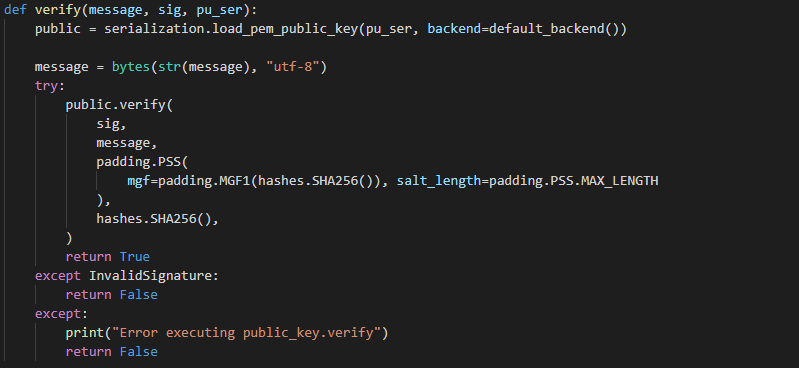
This function is used to generate the private and public key pairs. It uses the default cryptography rsa to generate the private and public key pairs.

1. **sign():**

This function is used to generate the signature on the basis of the transaction information and private key of the respective transaction.

1. **verify():**

This function is used to verify whether the encoded hash can be generated by the respective public key or not.



1. **Transactions.py**
   1. **Transactions class:**

The various function involved here are:

* + 1. **add\_input():**

This function is used to store all the input transaction details e.g. sender address and amount of transaction by the sender as a list array.

1. **add\_output():**

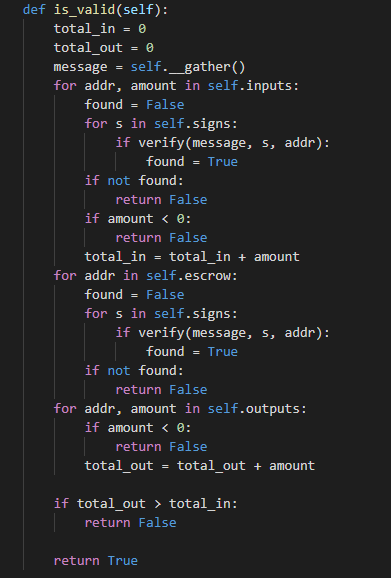
This function is used to store all the output transaction details e.g. receiver address and amount of transaction by the receiver as a list array.

1. **sign():**

This function generates the signature on the basis of input transactions and output transactions.

1. **is\_valid():**

This function validates whether the signature generated from the transaction details and private key of the sender could be generated with the pair public key and transaction details. Also, this function checks the feasibility of the transaction. Some of the criteria for measuring the feasibility are that the receiver cannot receive more than what's stored on the particular account and transactions below 0 cannot be carried out.



1. **\_\_gather():**

This inbuilt function is used to gather the information about the input and output transaction amount.

1. **\_\_repr\_\_:**

This inbuilt function returns the transaction on this particular class when called from outside the class/

1. **Blockchain.py**
   1. **BlockChain class:**

The various function involved here are:

1. **trans\_computeHash()**

Of the transactions involved in a block, the trans\_computeHash() function performs SHA256 hashing on the data involved in the transaction along with the hash of the previous block.

1. **is\_valid()**

is\_valid() function on the Blockchain class validates if the computed hash is equal to the hash of the previous block. This function returns true if the hashes are equal else returns False. Also, if the previous block has no hashes like in the case of the Genesis block, it returns true as well.

**4. Transaction-Blockchain.py**

* + - 1. **Btransac Class:**

The various function involved here are:

1. **add\_trans():**

There are several transactions on a block. This function is used to append different input transactions.

1. **\_\_inp\_out\_counter():**

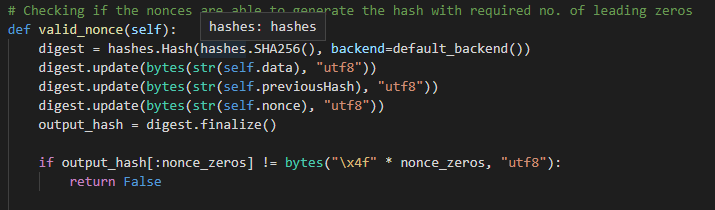
This is used to count the total number of inputs and output demands. There are multiple transactions on a block. So, this function adds up all the actual amount on the block of transactions and adds up all the demands on this block of transactions. And, returns the sum total of actual demand and input on the block.

1. **Is\_valid()**

The is\_valid() function on this file is used to verify if the overall transactions on a particular block is valid or not. A block has valid transactions if the sum total of output is less than sum of input and mining rewards.

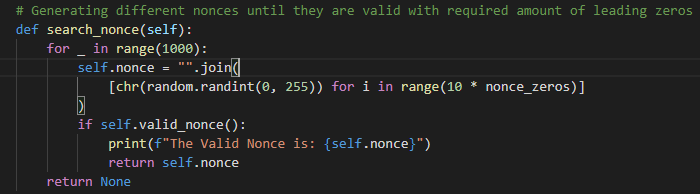
1. **Valid\_nonce()**

This function is used to validate if the nonce generated will give out hash with the required no. of leading zeros as predefined by the system such that the block could be mined by the miners.



1. **search\_nonce()**

This function is used to search various nonce that results in a required no. of leading zeros.

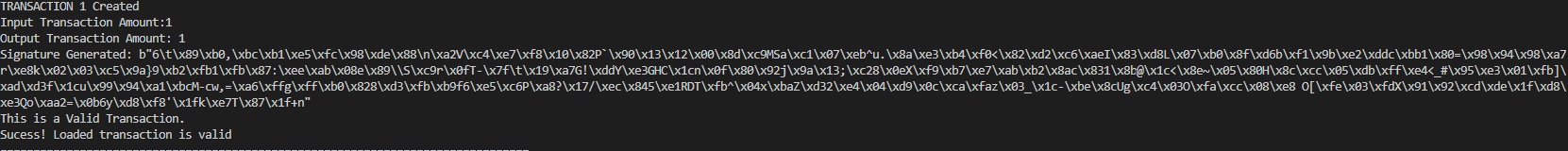
****

**Testing Outputs:**

**Transaction 1:**

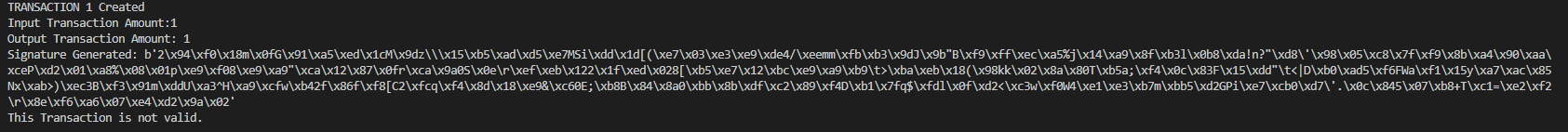
In this transaction the input amount is 1 and the output amount is also 1. The sender has used their private key to sign it. The code will first check if the transaction is valid using our is\_valid function and print if the transaction is valid or not.

After validation, our code will save the transaction into “transaction.dat” using pickle dump. To check if the saved file is correct, I have loaded the file and passed it through our is\_valid function again.



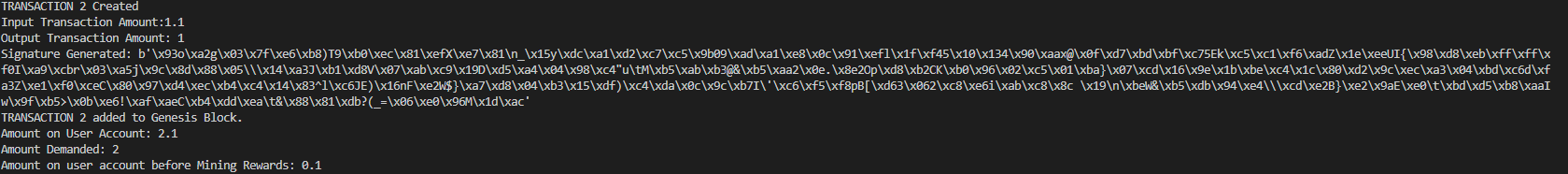
What if the private key of the receiver is used instead?

* Verification will fail and the transaction becomes invalid.



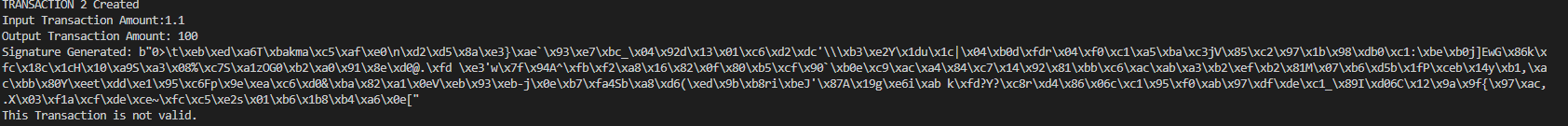
**Transaction 2:**

For our transaction 2, I have used 1.1 as the input amount and 1 as the output amount. Using correct signatures, the code will validate and will let us know if there is any amount on user account left after a valid transaction. The code has also added the amount from transaction 1.

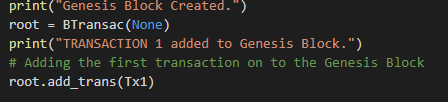


What if the output amount is more than the input amount?

* Transaction will not validate. Here, I tried 100 as an output and the validation failed as we only have 1 as an input.

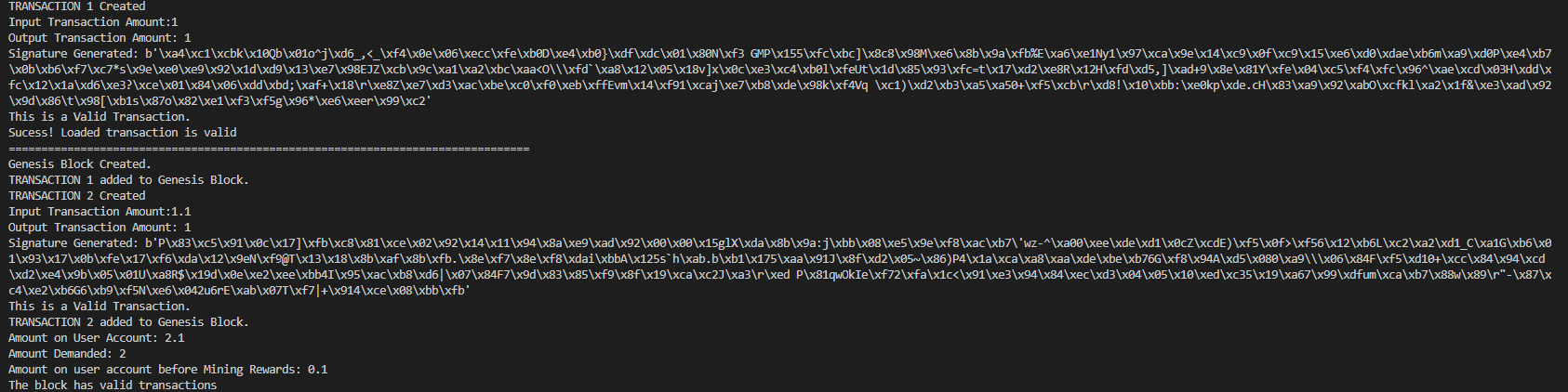


**Creating a Genesis block and adding transaction 1:**

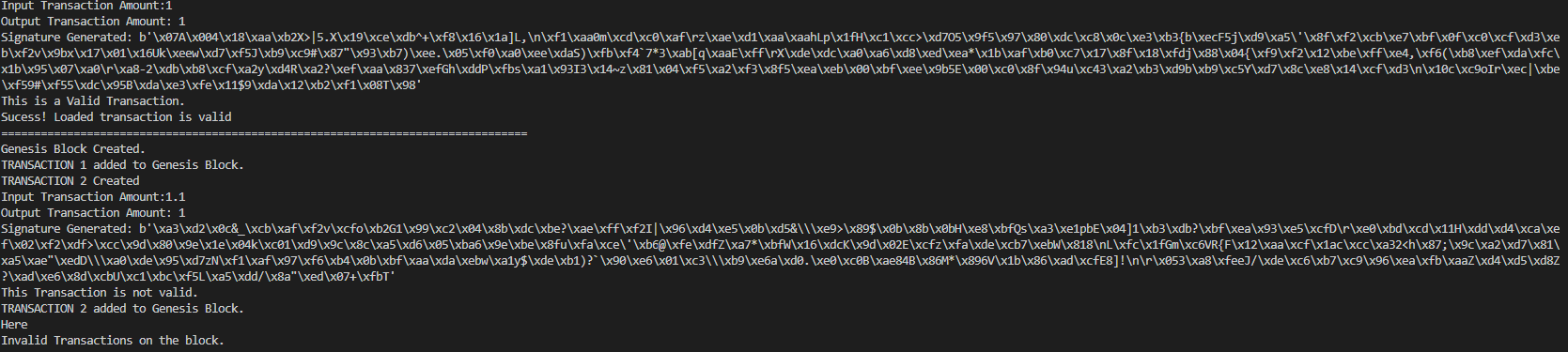
****

Similarly, I have added transaction 2 to the genesis block too. Now the question is, will our code let us know if we have a bad transaction in our Block?

* Yes, the code will validate the block and print if the transaction is valid or not.

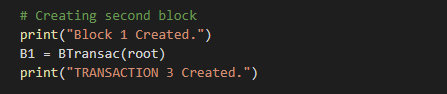


Here is the sample test run with transaction 1 being valid and transaction 2 being invalid:

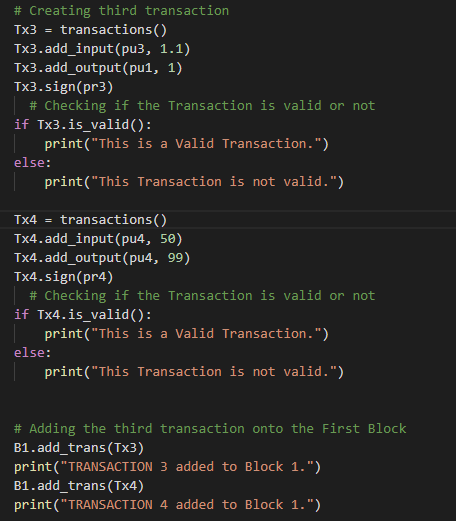


Is our code limited to only the genesis block?

* No, we can create blocks as we desire. Here in this screenshot, I have created a new block ‘B1’ with reference to our genesis (root) block.

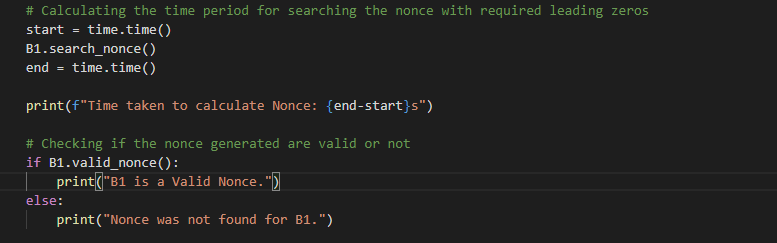


Now, let us create more transactions and add it to our Block 1.



How do we find and validate our ‘nonce’?

* We will call our search\_nonce and valid\_nonce to find and validate our nonce. In this code, number of leading zero is ‘1’. We can change the number of leading zero as per our need.

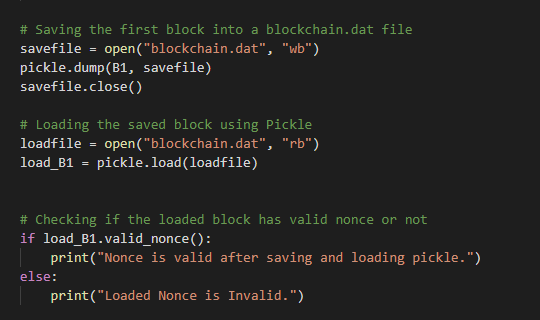


How does the nonce look like?



How do we save and load our Blockchain?

* We use pickle.dump and pickle.load to save and load our blockchain.dat file.
* Also, we will check if the nonce we found is valid after save and load.
* After validating nonce, the code will add a default mining reward of 12.5. We can change the value of our reward as per need.



Overall output of our code:

