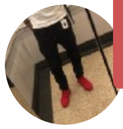


v.auny…

Jul 23 · 5 min read

Let’s Make the Tiniest Blockchain Bigger

Part 2: With More Lines of Python

Note: This article assumes you’ve read part one.

The tiniest blockchain was extremely simple, and it was relatively easy to make. But, with its simplicity came a few flaws. First, SnakeCoin only ran on one single machine, so it was far from distributed, let alone decentralized. Second, blocks could be added to the chain as fast as the host computer could create a Python object and add it to a list. In the case of a simple blockchain, that’s not a problem, but we’re now going to let SnakeCoin be an actual cryptocurrency so we’ll need control the amount of blocks (and coins) that can be created at a time.

From now on, SnakeCoin’s data will be transactions, so each block’s data field will be a list of some transactions. We’ll define a transaction as follows. Each transaction will be a JSON object detailing the sender of the coin, the receiver of the coin, and the amount of SnakeCoin that is being transferred. *Note: Transactions are in JSON format for a reason I’ll detail shortly.*

```
{
  "from": "71238uqirbfh894-random-public-key-a-alkjdfla
  "to": "93j4ivnqiopvh43-random-public-key-b-qjrgvnoeir
  "amount": 3
```

Now that we know what our transactions will look like, we need a way to add them to one of the computers in our blockchain network, called a node. To do that, we’ll create a simple HTTP server so that any user can let our nodes know that a new transaction has occurred. A node will be able to accept a POST request with a transaction (like above) as the request body. This is why transactions are JSON formatted; we need them to be transmitted to our server in a request body.

```
pip install flask # Install our web server framework first
```

```python
from flask import Flask
from flask import request
node = Flask(__name__)

# Store the transactions that
# this node has in a list
this_nodes_transactions = []

@node.route('/txion', methods=['POST'])
def transaction():
  if request.method == 'POST':
    # On each new POST request,
    # we extract the transaction data
    new_txion = request.get_json()
    # Then we add the transaction to our list
    this_nodes_transactions.append(new_txion)
    # Because the transaction was successfully
```

Awesome! Now we have a way to keep a record of users when they send SnakeCoins to each other. This is why people refer to blockchains as public, distributed ledgers: all transactions are stored for all to see and are stored on every node in the network.

But, a question arises: *where do people get SnakeCoins from?* Nowhere, yet. There’s no such thing as a SnakeCoin yet, because not one coin has been created and issued yet. To create new coins, people have to *mine* new blocks of SnakeCoin. When they successfully mine new blocks, a new SnakeCoin is created and rewarded to the person who mined the block. The coin then gets circulated once the miner sends the SnakeCoin to another person.

We don’t want it to be too easy to mine new SnakeCoin blocks, because that will create too many SnakeCoins and they will have little value. Conversely, we don’t want it to be too hard to mine new blocks, because there wouldn’t be enough coins for everyone to spend, and they would be too expensive for our liking. To control the difficulty of mining new SnakeCoins, we’ll implement a Proof-of-Work (PoW) algorithm. A Proof-of-Work algorithm is essentially an algorithm that generates an item that is difficult to create but easy to verify. The item is called the proof and, as it sounds, it is proof that a computer performed a certain amount of work.

In SnakeCoin, we'll create a somewhat simple Proof-of-Work algorithm. To create a new block, a miner's computer will have to increment a number. When that number is divisible by 9 (the number of letters in "SnakeCoin") and the proof number of the last block, a new SnakeCoin block will be mined and the miner will be given a brand new SnakeCoin.

Now, we can control the number of blocks mined in a certain time period, and we can issue new coins for people in the network to send to each other. But like we said, we're only doing this on one computer. *If blockchains are decentralized, how do we make sure that the same chain is on every node?* To do this, we make each node broadcast its version of the chain to the others and allow them to receive the chains of other nodes. After that, each node has to verify the other nodes' chains so that the every node in the network can come to a *consensus* of what the resulting blockchain will look like. This is called a consensus algorithm.

Our consensus algorithm will be rather simple: if a node's chain is different from another's (i.e. there is a conflict), then the longest chain in the network stays and all shorter chains will be deleted. If there is no conflict between the chains in our network, then we carry on.

We're just about done now. After running the full SnakeCoin server code, run the following commands in your terminal. *Assuming you have cURL* *installed.*

1. Create a transaction.

```
curl "localhost:5000/txion" \
     -H "Content-Type: application/json" \
     -d '{"from": "akjflw", "to":"fjlakdj", "amount": 3}'
```

2. Mine a new block.

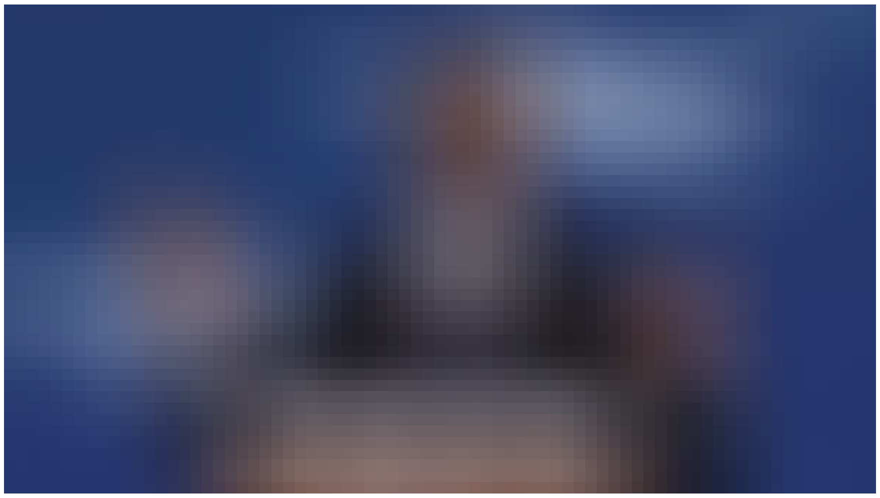
```
curl localhost:5000/mine
```

3. Check out the results. From the client window, we see this.

With a little bit of pretty printing we see that after mining we get some cool information on our new block.

```
{
  "index": 2,
  "data": {
    "transactions": [
      {
        "to": "fjlakdj",
        "amount": 3,
        "from": "akjflw"
      },
      {
        "to": "q3nf394hjg-random-miner-address-
34nf3i4nflkn3oi",
        "amount": 1,
        "from": "network"
      }
    ],
    "proof-of-work": 36
  },
  "hash":
"151edd3ef6af2e7eb8272245cb8ea91b4ecfc3e60af22d8518ef0bba8b
4a6b18",
  "timestamp": "2017-07-23 11:23:10.140996"
}
```

And that’s it! We’ve made a fairly sized blockchain at this point. Now, SnakeCoin can be launched on multiple machines to create a network, and real SnakeCoins can be mined. Please feel free to tinker with the SnakeCoin server code as much as you’d like, and ask as many questions as you need! **In the next part, we’ll discuss creating a SnakeCoin wallet, so users can send, receive, and store their SnakeCoins.**



Thank you very much for reading!

Twitter, Github, Snapchat, Instagram