Gerald Nash Follow
Howard University ’21. I write about math, computer science, technology, and society. http://www.auny…
Jul 16 · 4 min read

Let’s Build the Tiniest Blockchain

In Less Than 50 Lines of Python

Note: Part 2 of this piece can be found here.

Although some think blockchain is a solution waiting for problems, there’s no doubt that this novel technology is a marvel of computing. But, what exactly is a blockchain?

Blockchain

> *a digital ledger in which transactions made in bitcoin or another cryptocurrency are recorded chronologically and publicly.*

In more general terms, *it’s a public database where new data are stored in a container called a block and are added to an immutable chain (hence blockchain) with data added in the past*. In the case of Bitcoin and other cryptocurrencies, these data are groups of transactions. But, the data can be of any type, of course.

Blockchain technology has given rise to new, fully digital currencies like Bitcoin and Litecoin that aren’t issued or managed by a central authority. This brings new freedom to individuals who believe that today’s banking systems are a scam or subject to failure. Blockchain has also revolutionized distributed computing in the form of technologies like Ethereum, which has introduced interesting concepts like smart contracts.

In this article, I’ll make a simple blockchain in less than 50 lines of Python 2 code. It’ll be called SnakeCoin.

We’ll start by first defining what our blocks will look like. In blockchain, each block is stored with a timestamp and, optionally, an index. In SnakeCoin, we’re going to store both. And to help ensure integrity throughout the blockchain, each block will have a self-identifying hash. Like Bitcoin, each block’s hash will be a cryptographic hash of the block’s index, timestamp, data, and the hash of the previous block’s hash. Oh, and the data can be anything you want.

```
import hashlib as hasher

class Block:
  def __init__(self, index, timestamp, data, previous_
    self.index = index
    self.timestamp = timestamp
    self.data = data
    self.previous_hash = previous_hash
    self.hash = self.hash_block()

  def hash_block(self):
```

Awesome! We have our block structure, but we’re creating a block**chain**. We need to start adding blocks to the actual chain. As I mentioned earlier, each block requires information from the previous block. But with that being said, a question arises: **how does the first block in the blockchain get there?** Well, the first block, or *genesis block,* is a special block. In many cases, it's added manually or has unique logic allowing it to be added.

We'll create a function that simply returns a genesis block to make things easy. This block is of index 0, and it has an arbitrary data value and an arbitrary value in the "previous hash" parameter.

```python
import datetime as date

def create_genesis_block():
  # Manually construct a block with
  # index zero and arbitrary previous hash
```

Now that we're able to create a genesis block, we need a function that will generate succeeding blocks in the blockchain. This function will take the previous block in the chain as a parameter, create the data for the block to be generated, and return the new block with its appropriate data. When new blocks hash information from previous blocks, the integrity of the blockchain increases with each new block. If we didn't do this, it would be easier for an outside party to "change the past" and replace our chain with an entirely new one of their own. This chain of hashes acts as cryptographic proof and helps ensure that once a block is added to the blockchain it cannot be replaced or removed.

```python
def next_block(last_block):
  this_index = last_block.index + 1
  this_timestamp = date.datetime.now()
  this_data = "Hey! I'm block " + str(this_index)
  this_hash = last_block.hash
```

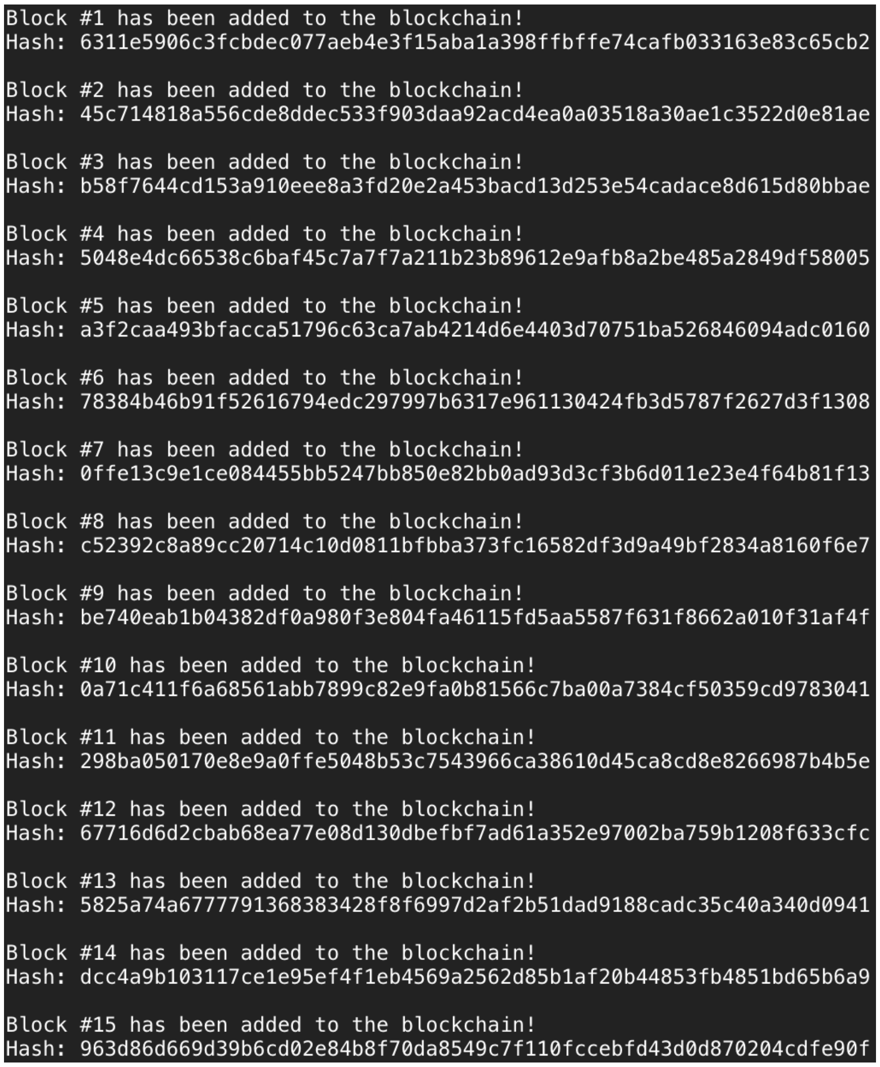
That's the majority of the hard work. Now, we can create our blockchain! In our case, the blockchain itself is a simple Python list. The first element of the list is the genesis block. And of course, we need to add the succeeding blocks. Because SnakeCoin is the tiniest blockchain, we'll only add 20 new blocks. We can do this with a for loop.

```python
# Create the blockchain and add the genesis block
blockchain = [create_genesis_block()]
previous_block = blockchain[0]

# How many blocks should we add to the chain
# after the genesis block
num_of_blocks_to_add = 20

# Add blocks to the chain
for i in range(0, num_of_blocks_to_add):
  block_to_add = next_block(previous_block)
```

Let’s test what we’ve made so far.



```
Block #1 has been added to the blockchain!
Hash: 6311e5906c3fcbdec077aeb4e3f15aba1a398ffbffe74cafb033163e83c65cb2

Block #2 has been added to the blockchain!
Hash: 45c714818a556cde8ddec533f903daa92acd4ea0a03518a30ae1c3522d0e81ae

Block #3 has been added to the blockchain!
Hash: b58f7644cd153a910eee8a3fd20e2a453bacd13d253e54cadace8d615d80bbae

Block #4 has been added to the blockchain!
Hash: 5048e4dc66538c6baf45c7a7f7a211b23b89612e9afb8a2be485a2849df58005

Block #5 has been added to the blockchain!
Hash: a3f2caa493bfacca51796c63ca7ab4214d6e4403d70751ba526846094adc0160

Block #6 has been added to the blockchain!
Hash: 78384b46b91f52616794edc297997b6317e961130424fb3d5787f2627d3f1308

Block #7 has been added to the blockchain!
Hash: 0ffe13c9e1ce084455bb5247bb850e82bb0ad93d3cf3b6d011e23e4f64b81f13

Block #8 has been added to the blockchain!
Hash: c52392c8a89cc20714c10d0811bfbba373fc16582df3d9a49bf2834a8160f6e7

Block #9 has been added to the blockchain!
Hash: be740eab1b04382df0a980f3e804fa46115fd5aa5587f631f8662a010f31af4f

Block #10 has been added to the blockchain!
Hash: 0a71c411f6a68561abb7899c82e9fa0b81566c7ba00a7384cf50359cd9783041

Block #11 has been added to the blockchain!
Hash: 298ba050170e8e9a0ffe5048b53c7543966ca38610d45ca8cd8e8266987b4b5e

Block #12 has been added to the blockchain!
Hash: 67716d6d2cbab68ea77e08d130dbefbf7ad61a352e97002ba759b1208f633cfc

Block #13 has been added to the blockchain!
Hash: 5825a74a6777791368383428f8f6997d2af2b51dad9188cadc35c40a340d0941

Block #14 has been added to the blockchain!
Hash: dcc4a9b103117ce1e95ef4f1eb4569a2562d85b1af20b44853fb4851bd65b6a9

Block #15 has been added to the blockchain!
Hash: 963d86d669d39b6cd02e84b8f70da8549c7f110fccebfd43d0d870204cdfe90f
```

Don’t worry. It goes up to 20.

There we go! Our blockchain works. If you want to see more information in the console, you could edit the complete source file and print each block’s timestamp or data.

That’s about all that SnakeCoin has to offer. To make SnakeCoin scale to the size of today’s production blockchains, we’d have to add more features like a server layer to track changes to the chain on multiple machines and a proof-of-work algorithm to limit the amount of blocks added in a given time period.

If you’d like to get more technical, you can view the original Bitcoin whitepaper here. Best of luck and happy hacking!

Note: Part 2 of this piece can be found here.

Thank you very much for reading!

Twitter, Github, Snapchat, Instagram