Anthony Castillo
Writing 105SW
June 17, 2018

How to Create a Blockchain in Python

Blockchain, possibly the most revolutionary computational technology since the Internet, is making waves in law, finance, and medicine. It is essentially a decentralized and distributed ledger system whose entries are immutable. We see blockchain appearing in technologies ranging from cryptocurrency to the Internet of Things. Below is how to create a blockchain in Python in less than 50 lines of code.

Step 1: Import Libraries

You will need the hashlib library to access the blockchain hash functions and to secure hashes and message digests. You will also need to import the datetime library so the genesis block can have a timestamp that helps create the header's unique digital signature.

```
import hashlib as hasher
import datetime as date
```

Step 2: Create the Blockchain Structure

We can classify this structure as a block since Python executes it as a unit. In other words, both functions work under one umbrella. The __init__ function is a recursive structure that uses "self" as the root of the structure. All other parameters (index, timestamp, data, and previous_hash) are indexed after "self" (as seen in the code). "hash" is indexed using hash_block() because this references the following command, which recursively structures an individual block using the hashlib library and the hash to the previous block. This structure essentially has a recursion within a recursion: as soon as the first function is executed, the second function executes (via self.hash_block()) and does so recursively because of the previous_hash parameter in the first function. This in turn makes the first function execute recursively as well, making for a well-oiled computational machine. Now, we have our blockchain structure created.

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

```
    def hash_block(self):
        sha = hasher.sha256()
        sha.update((str(self.index) + str(self.timestamp) +
            str(self.data) + str(self.previous_hash)).encode())
        return sha.hexdigest()
```

Step 3: Generate the Genesis Block

As soon as you create the structure for all non-initial blocks (as seen above), you can go ahead and manually construct the genesis block. The genesis block is the initial block of your blockchain with an index of zero and an arbitrary previous hash. The reason why we manually constructed the genesis block is because it does not have a previously existing hash in its metadata. After all, it's the first block in our chain. Therefore, we manually code in this metadata, which will work out well for this initial block and for the next blocks we want to add to the chain since they will have metadata to work from that already exists. Our index parameter is 0, our timestamp is now, our data is Genesis Block, and our previous hash is 0.

```
def create_genesis_block():
    return Block(0, date.datetime.now(), "Genesis Block", "0")
```

Step 4: Generate All Later Blocks in the Blockchain

With the genesis block and the structure for the blockchain now created, we can fill in for the gaps by creating data for our blocks. We create a function next_block(last_block) that works recursively. This function explicitly uses data from the previous block to create its own block. The new index (this_index) is created off the old index plus one. Our timestamp is once more in the immediacy. Our data for each block is a string (Hey! I'm block {this_index}) that declares the index of that block. Our hash for any new block will be recycled from an old block, and this all returns newly created blocks that would eventually be appended to the blockchain. In other words, we recursively created metadata that would fill in for a recursively created blockchain structure that also works in a recursive fashion.

```
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
    return Block(this_index,this_timestamp,this_data,this_hash)
```

Step 5: Create the Blockchain and Add the Genesis Block

With everything having been created in advance, we can now create the blockchain itself. We assign our genesis block to "blockchain," and then assign its initial value (that is, the value of the object indexed at 0) to previous_block. This enables us to always have recursively existing data on which we can fill in for our new blocks. We now have a chain that we can fill with blocks.

```
blockchain = [create_genesis_block()]
previous_block = blockchain[0]
```

Step 6: Decide How Many Blocks Should Go into the Blockchain

Pick whatever number of blocks you want to add to your blockchain and assign it to a variable to be used later. It doesn't matter. You can even use the random integer function for the sake of this step. We will use 20 for the sake of this assignment.

```
num_of_blocks_to_add = 20
```

Step 7: Add Blocks to the Chain

Set up a loop function that operates from 0 to whatever number of block you decided to add. This function will create and then append a block to the blockchain, and then the implicit memory pointer in this function moves from previous_block to block_to_add, prints a message saying the block was printed (and what its hash number was) and then repeats until the number of blocks desired have been added. By the end of this, you should have 20 blocks their unique hashes all printed out in the Python console.

```
for i in range(0, num_of_blocks_to_add):
    block_to_add = next_block(previous_block)
    blockchain.append(block_to_add)
    previous_block = block_to_add
    print("Block {} added to
          blockchain".format(block_to_add.index))
    print("Hash: {}\n".format(block_to_add.hash))
```

Thanks for reading through this tutorial. The blockchain code itself (which is in `Courier New` font) was inspired by a Howard University student named Gerald Nash. I credit him for the creation of the original blockchain itself by citing him below. The analysis and step-by-step tutorial (which is in Times New Roman Font) is my own work (all the writing that is not actual coding in Python but rather an explanation of the blockchain in action is my creation). Below is the test output for this particular blockchain. Have a nice day.

Sources: https://gist.github.com/aunyks/8f2c2fd51cc17f342737917e1c2582e2
https://medium.com/crypto-currently/lets-build-the-tiniest-blockchain-e70965a248b
Computer Science 8 w/ Prof Omer Egecioglu (Winter 2017 @ UCSB)

Appendix ⇒ Test Output:

```
Block 1 added to blockchain
Hash:
6796b45a65e5b5a1ae3775c90ee7e112a4416573b15d2c87dbe3f549f07dd165

Block 2 added to blockchain
Hash:
5c04b9eba56f5f5189e82e142f95651c8e3f73a7b652ed894e529a2af23d0ab2

Block 3 added to blockchain
Hash:
23a9b801ef5c2b8b535685615afd56491d08539951a5aba6c22ebd4183384241

Block 4 added to blockchain
Hash:
e57b19f1484b807731c9c69b53b1607e5f7e5a555a8c3986e09a8beae9bde557

Block 5 added to blockchain
Hash:
1868ba428181059036f3cbcbab19b7d67fabb9d0fa4777257b08b35143d1ea60

Block 6 added to blockchain
Hash:
43d9204f2d056ea55c40772f26d55894e2da55f10ebd8de92d97942da4673a3c

Block 7 added to blockchain
Hash:
66837dc971109c3c9248e310c25ddde83f79b0310c65a9dbaf2cf069198958a4

Block 8 added to blockchain
Hash:
ad142ca33740cd960eb6e92daeb693bb1d66159dda56726f3cbf5e7c6d12b21a

Block 9 added to blockchain
Hash:
86c94ed26a77f4f2c4eb59bf3dde67da4cbbf7dc8e0a36064a156119186f72f7

Block 10 added to blockchain
```

```
Hash:
71ccca72fd146e15b5004748b4224e39b5deee21dae3f75a5363e3274c7d2673

Block 11 added to blockchain
Hash:
23bbd7e19fe3e1d163179038ea7cd18a34273bfa785d4a9d3b0b4f7dc27427df

Block 12 added to blockchain
Hash:
0e70deb6ac3c61bedd9591cceedccd2c8efa93a09b3ea1990b4afcbdc725541b

Block 13 added to blockchain
Hash:
cc7ad270182aa4eb6080682cb6e891eec04f837b9e20bb15e81ea851abdaaefe

Block 14 added to blockchain
Hash:
955d4f6fb6796763751cd401e47fdaaae5835ac59f85ae545a1da784fd5b8cc8

Block 15 added to blockchain
Hash:
3ccb66323c90fa21bd572a1c5cf6062b204e451bb5811893e8d9e834ce31088d

Block 16 added to blockchain
Hash:
fc25bbf02a375fe6092e8a22bb339d87f31e6665c63b4764f19028930e533abb

Block 17 added to blockchain
Hash:
326c2db2738e29de5db6380ccff3206c50a8ca47ae76b1efa15c25702e668b10

Block 18 added to blockchain
Hash:
96b548a97e1d6e3b090e07a43b1146c0934c9293ffc97457c2bbf08450927b4e

Block 19 added to blockchain
Hash:
e90f697de550697fd369ff472529c618b0f3e1d9649e130a3b5cb3fca63ccad0

Block 20 added to blockchain
Hash:
f5faa48ca7d4f4d634e649b50040993a6ec320b65f15c77e38c3bb5957cd1c98
```