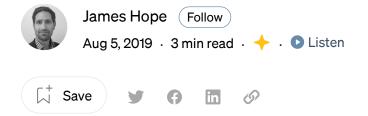
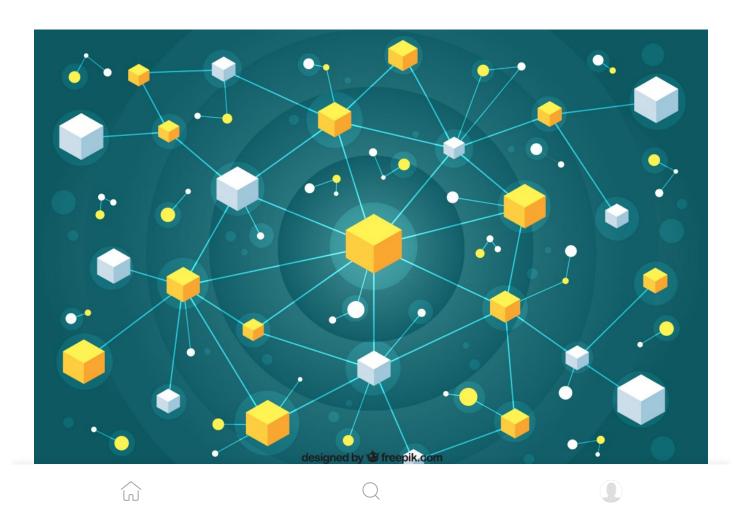
You have 2 free member-only stories left this month. Sign up for Medium and get an extra one



Build a blockchain protocol for a distributed ledger

Demystifying blockchain through a simple coding examples





Open in app Get started

mechanisms around blockchain and how they might be implemented in Python.

An quick overview of blockchain

The basic idea of a blockchain protocol is to preserve a database called a *ledger* across a distributed network of machines or *nodes*. Ledgers can be used for anything, to hold transactions or digital currency as is the case for BitCoin or Ethereum. To maintain the integrity of the ledger which is held in different places across the network, the blockchain protocol rewards participants on the network when they solve a computational intensive problem — a process referred to as *mining*. It is this process into which participants on the network enter into, that ensures the integrity of the distributed ledger on public networks. A blockchain is essentially a proof of the solutions to the problems that participants on the network solve, which are linked together in a chain, and with each new solution found, a new block is created, and into which the next set of records are copied.

Building a Blockchain Class

First, we'll build the Blockchain class constructor method that creates an initial empty list (to store our blockchain) and another to store the ledger —a record of transactions. The Blockchain class object will be responsible for managing the entire blockchain. We'll create the following methods to help us manage the blockchain:

- [1] register_node() this method will register a new node on the network
- [2] *new_block()* this method will create a new block in the blockchain, copy recent transactions into this new block and clear the set of transactions.
- [3] valid_proof() this method will check that a block being submitted to add to the blockchain solves the problem.
- [4] proof_of_work() this method will check that mining has yielded the correct proof to create the next block in the blockchain. We return the proof if the proof is correct, otherwise it will keep calling *valid_proof()* until we find a valid proof.

G







[7] $last_block()$ — this method will return the last block in the chain.

Here is how the Blockchain class looks:

'n

Q

21/09/2022, 20:45

```
Get started
```

```
SETI.CUITEIIC_CLAIISACCIOIIS - []
             self.chain = []
             self.nodes = set()
 6
 7
             # Create the genesis block
 8
             self.new_block(previous_hash='1', proof=100)
10
         def register_node(self, address):
11
12
             parsed_url = urlparse(address)
13
             self.nodes.add(parsed_url.netloc)
14
         def valid_chain(self, chain):
15
16
17
             last_block = chain[0]
18
             current_index = 1
19
             while current_index < len(chain):</pre>
20
                 block = chain[current_index]
21
22
                 print(f'{last_block}')
                 print(f'{block}')
23
                 print("\n----\n")
24
                 # Check that the hash of the block is correct
25
                 if block['previous_hash'] != self.hash(last_block):
26
27
                      return False
28
                 # Check that the Proof of Work is correct
29
                 if not self.valid_proof(last_block['proof'], block['proof']):
30
31
                      return False
32
                 last_block = block
33
34
                 current_index += 1
35
36
             return True
37
38
39
         def new_block(self, proof, previous_hash):
40
             block = {
41
                  'index': len(self.chain) + 1,
42
43
                  'timestamp': time(),
```

(i) Q

Get started

```
# Reset the current list of transactions
             self.current_transactions = []
50
51
             self.chain.append(block)
52
53
             return block
54
         def new_transaction(self, sender, recipient, amount):
55
56
57
             self.current_transactions.append({
                  'sender': sender,
58
59
                  'recipient': recipient,
60
                  'amount': amount,
61
             })
62
             return self.last_block['index'] + 1
63
64
65
         @property
         def last_block(self):
66
67
             return self.chain[-1]
68
         @staticmethod
69
70
         def hash(block):
71
72
             # We must make sure that the Dictionary is Ordered, or we'll have inconsistent hashes
73
             block_string = json.dumps(block, sort_keys=True).encode()
74
             return hashlib.sha256(block_string).hexdigest()
75
76
         def proof_of_work(self, last_proof):
             1.1.1
77
78
             Simple Proof of Work Algorithm:
79
              - Find a number p' such that hash(pp') contains leading 4 zeroes, where p is the previous
              - p is the previous proof, and p' is the new proof
80
81
82
             proof = 0
83
             while self.valid_proof(last_proof, proof) is False:
84
85
                 proof += 1
86
87
             return proof
88
```

Ĺ

V

21/09/2022, 20:45

Resolving conflicts on the network

Since the Blockchain protocol is run over a distributed network we'll need a method on the Blockchain class object to resolve conflicts on the network. We'll simply check if there are any other blockchains on the network that are longer than ours and if so, make that our own chain, otherwise we can continue to mine in the knowledge that we are mining for the next block and our efforts are worth it.

6 sur 11 21/09/2022, 20:45



```
Get started
```

```
HETRIHOULS - SETI HOUSE
             new chain = None
             # We're only looking for chains longer than ours
 6
7
             max length = len(self.chain)
 8
             # Grab and verify the chains from all the nodes in our network
             for node in neighbours:
10
                 response = requests.get(f'http://{node}/chain')
11
12
13
                 if response.status_code == 200:
14
                      length = response.json()['length']
                      chain = response.json()['chain']
15
16
                      # Check if the length is longer and the chain is valid
17
18
                      if length > max_length and self.valid_chain(chain):
19
                          max_length = length
20
                          new_chain = chain
21
22
             # Replace our chain if we discovered a new, valid chain longer than ours
23
             if new_chain:
24
                 self.chain = new_chain
25
                 return True
26
27
             return False
blockchain.py hosted with \visitetin by GitHub
                                                                                                view raw
```

Interacting with the Blockchain

Finally, to bring our blockchain to life and to interact with it, we can expose a number of methods via a web app interface using flask.

Registering a node on the network

We'll want to allow users to register a node on the network so they can participate in the *proof of work* process and be rewarded for their efforts. We call the register_node() method on the Blockchain class object to register a node.



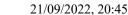


```
Open in app
          varues - request.get_Json()
         nodes = values.get('nodes')
         if nodes is None:
 6
 7
              return "Error: Please supply a valid list of nodes", 400
 8
         for node in nodes:
10
              blockchain.register_node(node)
11
         response = {
12
13
              'message': 'New nodes have been added',
              'total_nodes': list(blockchain.nodes),
14
15
         return jsonify(response), 201
16
blockchain.py hosted with \visitetin by GitHub
                                                                                                  view raw
```

We'll also want to allow these participants to check if their Blockchain is valid and fetch the Blockchain from the network if it isn't.

```
@app.route('/nodes/resolve', methods=['GET'])
 2
     def consensus():
         replaced = blockchain.resolve_conflicts()
 3
 4
 5
         if replaced:
 6
             response = {
 7
                  'message': 'Our chain was replaced',
                  'new_chain': blockchain.chain
 8
 9
10
         else:
             response = {
11
                  'message': 'Our chain is authoritative',
12
                  'chain': blockchain.chain
13
             }
14
15
         return jsonify(response), 200
16
blockchain.py hosted with \ by GitHub
                                                                                                 view raw
```







Open in app Get started

added.

```
@app.route('/transactions/new', methods=['POST'])
1
2
     def new_transaction():
         values = request.get_json()
3
4
5
         # Check that the required fields are in the POST'ed data
         required = ['sender', 'recipient', 'amount']
6
         #if not all(k in values for k in required):
7
              return 'Missing values', 400
8
9
         # Create a new Transaction
10
         index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])
11
12
         response = {'message': f'Transaction will be added to Block {index}'}
13
14
         return jsonify(response), 201
blockchain.py hosted with \visitetin by GitHub
                                                                                                view raw
```

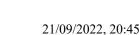
Mining

One last important thing! We need to create the interaction to allow participants on the network to mine their blockchain.









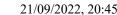


```
Get started
```

```
# we run the proof of work affortime to get the next proof...
         last_block = blockchain.last_block
         last_proof = last_block['proof']
         proof = blockchain.proof_of_work(last_proof)
 6
 7
 8
         # We must receive a reward for finding the proof.
         # The sender is "0" to signify that this node has mined a new coin.
10
         blockchain.new_transaction(
             sender="0",
11
             recipient=node_identifier,
12
13
             amount=1,
         )
14
15
         # Forge the new Block by adding it to the chain
16
         previous_hash = blockchain.hash(last_block)
17
18
         block = blockchain.new_block(proof, previous_hash)
19
20
         response = {
              'message': "New Block Forged",
21
22
              'index': block['index'],
              'transactions': block['transactions'],
23
              'proof': block['proof'],
24
              'previous_hash': block['previous_hash'],
25
26
27
         return jsonify(response), 200
blockchain.py hosted with 9 by GitHub
                                                                                                       view raw
```

Now we can instantiate the flask app, create the Blockchain class object and run the flask application and interact with our blockchain.









```
# Generate a globally unique address for this node
 5
     node_identifier = str(uuid4()).replace('-', '')
 6
 7
     # Instantiate the Blockchain
     blockchain = Blockchain()
 8
 9
10
     if __name__ == '__main__':
         from argparse import ArgumentParser
11
12
13
         parser = ArgumentParser()
         parser.add_argument('-p', '--port', default=5000, type=int, help='port to listen on')
14
15
         args = parser.parse args()
16
         port = args.port
17
18
          app.run(host='0.0.0.0', port=port)
blockchain.py hosted with \bigointmose by GitHub
                                                                                                         view raw
```

To download the full code file see: https://github.com/jamesdhope/blockchain

References

- [1] https://www.wikihow.com/Build-a-Blockchain-App
- [2] https://www.codecademy.com/learn/introduction-to-blockchain/modules/blockchain-in-python
- [3] Daniel van Flymen, https://hackernoon.com/learn-blockchains-by-building-one-117428612f46.
- [4] https://towardsdatascience.com/building-a-minimal-blockchain-in-python-4f2e9934101d
- [5] Omer Goldberg, https://hackernoon.com/building-a-blockchain-the-grey-paper-5be456018040



