Appendix I: `main_wrapper.py`

```
05/12/19 05:08:21 C:\GitHub\basic_blockchain\main_wrapper.py
```

```python
1   from html5lib import *
2   from blockchain_utils import *
3   import os
4
5   TABLE_LABELS = ['Proof', 'Next Block', 'Update Body', 'Time']
6   SEED_LENGTH = 256
7
8   # adds a new block to the webpage
9   def append_update(blockchain, proof, update_file, private_mode):
10      with open(update_file, "r") as rd_file:
11          update_block = rd_file.read()
12          if private_mode:
13              salt = os.urandom(SEED_LENGTH)
14              update_block = salt.hex() + ' ' + HASH_FN(salt +
    str.encode(update_block)).hexdigest()
15          blockchain.append_block(proof, blockchain.tail, update_block)
16
17
18  # reads in information from an existing chain
19  def parse_chain(chain_folder, chain_head):
20      if chain_folder[-1] != '/':
21          chain_folder += '/'
22
23      curr_block = chain_folder + chain_head
24
25      with open(curr_block, "r") as ch_file:
26          curr_block = ch_file.read()
27          next_block = chain_folder + curr_block
28
29      blocks = []
30
31      while os.path.isfile(next_block):
32          with open(next_block) as block_file:
33              b1 = block_file.readline()
34              b2 = block_file.readline()
35              next_block = block_file.readline()[:-1]
36              b3 = block_file.readline()
37              b4 = block_file.readline()
38          curr_block = (b1, b2, next_block + "\n", b3, b4)
39          blocks += [curr_block]
40          next_block = chain_folder + next_block
41      return blocks
42
43  # uses existing blockchain files to generate the web page for that blockchain
44  def display(output_file, chain_folder, chain_head):
45      blocks = parse_chain(chain_folder, chain_head)
46
47      with open(output_file, 'w+') as out_file:
```

```python
48            out_file.write('<html><head><title>' + chain_folder + '</title>')
49            out_file.write('<link rel="stylesheet" href="format.css"></head>')
50            out_file.write('<body><h1 align="center">' + chain_folder + '</h1>')
51
52        for b in blocks:
53            out_file.write('<table>')
54            out_file.write('<tr><th colspan=2>' + b[0] + '</th></tr> <col
   width="20%"/><col width="80%"/>')
55            for i in range(0,4):
56                out_file.write('<tr><td><b>' + TABLE_LABELS[i] + '</b></td><td>' +
   b[i+1] + '</td>')
57            out_file.write('</table>')
58            out_file.write('<br>')
59
60        out_file.write('</body></html>')
61
62
63 # Example test code
64
65 # Generates the raw public version of the blockchain, with updates in plain text.
66 test_blockchain = Chain('test_blockchain', 512)
67 append_update(test_blockchain, 'proof1', 'test_blockchain_updates/update1', False)
68 append_update(test_blockchain, 'proof2', 'test_blockchain_updates/update2', False)
69 append_update(test_blockchain, 'proof3', 'test_blockchain_updates/update3', False)
70 append_update(test_blockchain, 'proof4', 'test_blockchain_updates/update4', False)
71 append_update(test_blockchain, 'proof5', 'test_blockchain_updates/update5', False)
72 display('test_blockchain.html', 'test_blockchain', 'test_blockchain')
73
74 # Generates the private version of the blockchain, with a salted and hashed
   version of the update, to protect proprietary code.
75 test_blockchain_private = Chain('test_blockchain_private', 512)
76 append_update(test_blockchain_private, 'proof1',
   'test_blockchain_updates/update1', True)
77 append_update(test_blockchain_private, 'proof2',
   'test_blockchain_updates/update2', True)
78 append_update(test_blockchain_private, 'proof3',
   'test_blockchain_updates/update3', True)
79 append_update(test_blockchain_private, 'proof4',
   'test_blockchain_updates/update4', True)
80 append_update(test_blockchain_private, 'proof5',
   'test_blockchain_updates/update5', True)
81 display('test_blockchain_private.html', 'test_blockchain_private',
   'test_blockchain_private')
```

Appendix II: `blockchain_utils.py`

05/12/19 05:04:07 C:\GitHub\basic_blockchain\blockchain_utils.py

```python
 1  '''
 2  A very, very naive blockchain.
 3
 4  Only used for demo purposes.
 5
 6  Not provably secure. Meant to show that it works in the context of this project.
 7  '''
 8
 9
10
11  import hashlib
12  import os
13  from stat import S_IREAD, S_IRGRP, S_IROTH
14  import binascii
15  import time
16
17  HASH_FN = hashlib.sha512
18  HASH_LENGTH = 512
19
20
21  # a placeholder function that users can customize, if they want a different
    layout
22  def format_update(update_file):
23      ''' Converts raw update file to a format for the blockchain. '''
24      with open(update_file, "rb") as update_file:
25          update_body = update_file.read()
26      # do whatever formatting here
27      return update_body # should be in byte format
28
29
30  # raised when a proof of work check fails
31  class ProofException(Exception):
32      ''' Raised when proof of work fails to verify. '''
33      pass
34
35
36  class Block():
37      '''
38      A single update unit to a blockchain. Block information is formatted in this
    order.
39
40      self.hash    =   hash of current block (includes everything else)
41      proof        =   proof of work
42      prev         =   hash of previous block
43      body         =   body of update for current block
44      append_time =    timestamp
45
46      Everything must be string format.
```

```python
47            '''
48        def verify_proof(self, proof, prev, body):
49            # return proof[:6] == '000000'
50            return True
51
52        def __init__(self, chain_name, proof, prev, body):
53            # verify proof
54            if self.verify_proof(proof, prev, body):
55                # generate hash of block
56                append_time = str(time.time())
57                try:
58                    self.hash = HASH_FN(str.encode(proof) + str.encode(prev) +
    str.encode(body) + str.encode(append_time)).hexdigest()
59                except:
60                    print("All arguments should be passed as string format.")
61
62                # write to record file
63                with open(chain_name + "/" + self.hash, "w+") as block_file:
64                    block_file.write(self.hash + '\n')
65                    block_file.write(proof + '\n')
66                    block_file.write(prev + '\n')
67                    block_file.write(body + '\n')
68                    block_file.write(append_time + '\n')
69
70                # make file read-only
71                os.chmod(chain_name + "/" + self.hash, S_IREAD)
72            else:
73                # do not create block if proof does not verify
74                raise ProofException("Proof of work failed.")
75
76
77  class Chain():
78        '''
79        Made of many Blocks strung together.
80        The first Block is always a bunch of random gibberish, i.e. a standard
    header.
81        This is to provide randomness and security to the rest of the blockchain,
82        and to prevent null pointers.
83
84        chain_header stores the tail, aka the hash of the most recently appended
    block.
85
86        self.name = name of the chain
87        self.tail = hash of most recently appended block
88            This is just meant to make appending easier for the sake of testing and
    demonstration.
89            This tail file is not meant to be a secure display of the latest appended
    block. Unlike the record files it is not write protected.
90        '''
91
92        # reads from existing chain if available, or creates a new one
93        def __init__(self, chain_name, seed_length):
94            self.name = chain_name
95
```

```python
 96             if os.path.isfile(chain_name): # existing chain already exists
 97                 print("Loading blockchain '" + chain_name + "'.\n")
 98                 with open(chain_name + '/' + chain_name, "r") as chain_header:
 99                     self.tail = chain_header.readline()
100
101             else: # initialize new blockchain
102                 print("Blockchain '" + chain_name + "'' does not exist. Initializing
    new chain.\n")
103                 os.mkdir(chain_name)
104
105                 # create new root block
106                 with open(chain_name + '/' + chain_name, 'w+') as chain_header:
107                     proof = os.urandom(seed_length).hex()
108                     prev = os.urandom(HASH_LENGTH).hex()
109                     body = os.urandom(seed_length).hex()
110
111                     root_block = Block(self.name, proof, prev, body)
112                     chain_header.write(root_block.hash)
113
114                 # initialize blockchain t ail
115                 with open(chain_name + '/' + chain_name, "r") as chain_header:
116                     self.tail = chain_header.readline()
117
118
119     # adds a new block onto the chain
120     def append_block(self, proof, prev, body):
121         try: # attempt to create new block
122             new_block = Block(self.name, proof, prev, body)
123
124             # update header file and tail
125             with open(self.name + '/' + self.name, 'w') as chain_header:
126                 chain_header.write(new_block.hash)
127                 self.tail = new_block.hash
128             print("Successfully appended update " + new_block.hash + "\n")
129
130         except ProofException:
131             # do not create new block record or update tail if proof fails
132             print("Proof of work failed. Block not appended.")
133
134
135
136
137 # test_chain = Chain('newchain_1', 720)
138 # test_chain.append_block('proof1', test_chain.tail, 'update1')
139 # test_chain.append_block('proof2', test_chain.tail, 'update2')
140 # test_chain.append_block('proof3', test_chain.tail, 'update3')
141 # test_chain.append_block('proof4', test_chain.tail, 'update4')
142 # test_chain.append_block('proof5', test_chain.tail, 'update5')
```

Appendix III: `format.css`

05/12/19 05:09:10 C:\GitHub\basic_blockchain\format.css

```css
 1  body {
 2    background-color: #c4e3fd;
 3    font-family: "Monospace", Courier;
 4  }
 5  table {
 6    table-layout: fixed;
 7    width: 100%;
 8    background-color: #FFFFFF;
 9    border: 3px solid black;
10    border-collapse: collapse;
11  }
12  th {
13    word-wrap: break-word;
14    background-color: #6ab9f9;
15    border: 2px solid black;
16    border-collapse: collapse;
17  }
18  tr {
19    word-wrap: break-word;
20  }
21  td {
22    border: 1px solid black;
23    border-collapse: collapse;
24    vertical-align: top;
25  }
26  td+td {
27    width: auto;
28  }
```