Blockchain Implementation in Python

Marisa Paryasto
25 Nov 2019

Steps

1. Make a Simple Chain
2. Adding Transactions to the Chain
3. Creating Genesis Block
4. Hash Function
5. Validating Proof
6. Mining Process
7. Verifying the Validity of a Blockchain

Make a simple chain

```
#let's create a list of blockchain
blockchain = []


#now creating a function to add elements to the list
def add_list(b):
   #add element to the list using append
   blockchain.append(b)
   print(blockchain) #printing the list



#now let's add something to the list
add_list('A - B, 10') #we pretend that we are making notes
#of a transaction between A and B for 10 bitcoins :)
add_list('B - C, 2')
add_list('C - A, 1')
```

Adding transactions to the chain

```
#import json


blockchain = []


def get_last_value():
    return(blockchain[-1])
```

```
def add_value(sender, recipient, amount=1.0):
    transaction = {'sender': sender,
    'recipient': recipient,
    'amount': amount}
    blockchain.append(transaction)


def get_transaction_value():
    tx_sender = raw_input('Enter the sender: ')
    tx_recipient = raw_input('Enter the recipient of the transaction: ')
    tx_amount = float(raw_input('Enter your transaction amount: '))
    return tx_sender, tx_recipient, tx_amount
```

```
def print_block():
    for block in blockchain:
        print("Here is your block")
        print(block)


again = True
while again == True:
    tx = get_transaction_value()
    s, r, a = tx
    add_value(s, r, a)
    print(blockchain)
    more = raw_input("add more block (Y/N)? ")
    if more.lower() == 'y':
        again = True
    else:
        again = False
```

.

Creating Genesis Block

```
genesis_block = {
   'previous_hash': '',
   'index': 0,
   'transaction': [],
   'nonce': 23
}


blockchain = [genesis_block]
```

```
def get_last_value():
   return(blockchain[-1])


def add_value(recipient, sender, amount=1.0):
   transaction = {'sender': sender,
   'recipient': recipient,
   'amount': amount}
   open_transactions.append(transaction)


def get_transaction_value():
   tx_sender = raw_input('Enter the sender: ')
   tx_recipient = raw_input('Enter the recipient of the transaction: ')
   tx_amount = float(input('Enter your transaction amount: '))
   return tx_sender, tx_recipient, tx_amount
.
```

```
def get_user_choice():
   user_input = input("Please give your choice here: ")
   return user_input


def print_block():
   for block in blockchain:
       print("Here is your block")
       print(block)


print_block()
'''
again = True
while again == True:
   tx = get_transaction_value()
   s, r, a = tx
   add_value(s, r, a)
   print(blockchain)
   more = raw_input("add more block (Y/N)? ")
   if more.lower() == 'y':
      again = True
   else:
      again = False
.
```

```
while True:
    print("Choose an option")
    print('Choose 1 for adding a new transaction')
    print('Choose 2 for mining a new block')
    print('Choose 3 for printing the blockchain')
    print('Choose anything else if you want to quit')

    user_choice = get_user_choice()

    if user_choice == 1:
        tx_data = get_transaction_value()
        recipient, amount = tx_data
        add_value(recipient, amount=amount)
        print(open_transactions)
    elif user_choice == 2:
        mine_block()
    elif user_choice == 3:
        print_block()
    else:
        break
'''
.
```

Hash Function

```
import hashlib
import json


genesis_block = {
    'previous_hash': '',
    'index': 0,
    'transaction': [],
    'nonce': 23
}
```

```
blockchain = [genesis_block]
open_transactions = []


def get_last_value():
    return(blockchain[-1])


def add_value(recipient, sender, amount=1.0):
    transaction = {'sender': sender,
    'recipient': recipient,
    'amount': amount}
    open_transactions.append(transaction)
.
```

```
def get_transaction_value():
    tx_sender = raw_input('Enter the sender: ')
    tx_recipient = raw_input('Enter the recipient of the transaction: ')
    tx_amount = float(input('Enter your transaction amount: '))
    return tx_recipient, tx_amount


def print_block():
    for block in blockchain:
        print("Here is your block")
        print(block)
```

```
def hash_block(block):
    return hashlib.sha256(json.dumps(block).encode()).hexdigest()


last_block = blockchain[-1]
print "last block: " , last_block
last_hash = hash_block(last_block)
print "last hash: ", last_hash
```

Validating Proof

```
import hashlib
import json


genesis_block = {
    'previous_hash': '',
    'index': 0,
    'transaction': [],
    'nonce': 23
}


blockchain = [genesis_block]
open_transactions = []


def get_last_value():
    return(blockchain[-1])


def add_value(recipient, sender, amount=1.0):
    transaction = {'sender': sender,
    'recipient': recipient,
    'amount': amount}
    open_transactions.append(transaction)
.
```

```
def get_transaction_value():
    tx_recipient = raw_input('Enter the recipient of the transaction: ')
    tx_amount = float(input('Enter your transaction amount '))
    return tx_recipient, tx_amount


def get_user_choice():
    user_input = input("Please give your choice here: ")
    return user_input


def print_block():
    for block in blockchain:
        print("Here is your block")
        print(block)


def hash_block(block):
    return hashlib.sha256(json.dumps(block).encode()).hexdigest()
```

```
def valid_proof(transactions, last_hash, nonce):
    guess = (str(transactions) + str(last_hash) + str(nonce)).encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    print(guess_hash)
    return guess_hash[0:2] == '00'


def pow():
    last_block = blockchain[-1]
    last_hash = hash_block(last_block)
    nonce = 0
    while not valid_proof(open_transactions, last_hash, nonce):
        nonce += 1
    return nonce


print blockchain

nonce = pow()

print nonce
```

Mining Process

```
import hashlib
import json


reward = 10.0


genesis_block = {
    'previous_hash': '',
    'index': 0,
    'transaction': [],
    'nonce': 23
}


blockchain = [genesis_block]
open_transactions = []
owner = 'procodecg'


def get_last_value():
    return(blockchain[-1])


def add_value(recipient, sender=owner, amount=1.0):
    transaction = {'sender': sender,
    'recipient': recipient,
    'amount': amount}
    open_transactions.append(transaction)
.
```

```
def get_transaction_value():
    tx_recipient = raw_input('Enter the recipient of the transaction: ')
    tx_amount = float(input('Enter your transaction amount: '))
    return tx_recipient, tx_amount


def get_user_choice():
    user_input = input("Please give your choice here: ")
    return user_input


def print_block():
    for block in blockchain:
        print("Here is your block")
        print(block)


def hash_block(block):
    return hashlib.sha256(json.dumps(block).encode()).hexdigest()
```

```
def valid_proof(transactions, last_hash, nonce):
    guess = (str(transactions) + str(last_hash) + str(nonce)).encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    print(guess_hash)
    return guess_hash[0:2] == '00'


def pow():
    last_block = blockchain[-1]
    last_hash = hash_block(last_block)
    nonce = 0
    while not valid_proof(open_transactions, last_hash, nonce):
        nonce += 1
    return nonce
```

```
def mine_block():
    last_block = blockchain[-1]
    hashed_block = hash_block(last_block)
    nonce = pow()
    reward_transaction = {
            'sender': 'MINING',
            'recipient': owner,
            'amount': reward
        }
.
```

```
open_transactions.append(reward_transaction)
    block = {
        'previous_hash': hashed_block,
        'index': len(blockchain),
        'transaction': open_transactions,
        'nonce': nonce
    }
    blockchain.append(block)


def verify_chain():
    index = 0
    valid = True
    for block in blockchain:
        if index == 0:
            index += 1
            continue
        elif block[0] == blockchain[index - 1]:
            valid = True
        else:
            valid = False
            break
        index += 1
    return valid
```

```
while True:
    print("Choose an option")
    print('Choose 1 for adding a new transaction')
    print('Choose 2 for mining a new block')
    print('Choose 3 for printing the blockchain')
    print('Choose anything else if you want to quit')

    user_choice = get_user_choice()

    if user_choice == 1:
        tx_data = get_transaction_value()
        recipient, amount = tx_data
        add_value(recipient, amount=amount)
        print(open_transactions)
    elif user_choice == 2:
        mine_block()
    elif user_choice == 3:
        print_block()
    else:
        break
```

Verify the Validity of a Blockchain

```
blockchain = []


def get_last_value():
    return(blockchain[-1])


def add_value(transaction_amount, last_transaction=[1]):
    blockchain.append([last_transaction, transaction_amount])
```

```
def get_transaction_value():
    tx_sender = raw_input('Enter the sender: ')
    tx_recipient = raw_input('Enter the recipient of the transaction: ')
    tx_amount = float(raw_input('Enter your transaction amount: '))
    return tx_sender, tx_recipient, tx_amount


def print_block():
    for block in blockchain:
        print("Here is your block")
        print(block)
```

```
def get_user_choice():
    user_input = input("Please give your choice here: ")
    return user_input


def verify_chain():
    index = 0
    valid = True
    for block in blockchain:
        if index == 0:
            index += 1
            continue
        elif block[0] == blockchain[index - 1]:
            valid = True
        else:
            valid = False
            break
        index += 1
    return valid


tx_amount = get_transaction_value()

add_value(tx_amount)
```

```
while True:
    print("Choose an option")
    print('Choose 1 for adding a new transaction')
    print('Choose 2 for printing the blockchain')
    print('Choose 3 if you want to manipulate the data')
    print('Choose anything else if you want to quit')
    user_choice = get_user_choice()


    if user_choice == 1:
        tx_amount = get_transaction_value()
        add_value(tx_amount, get_last_value())
    elif user_choice == 2:
        print_block()
    elif user_choice == 3:
        if len(blockchain) >= 1:
            blockchain[0] = 2
    else:
        break


    if not verify_chain():
        print('Blockchain manipulated')
        break
```