blockchain.py

```
importhashlib
importjson
fromtimeimporttime
fromurlparseimporturlparse
importrequests

####### block generation & its principle

classBlockchain(object):
# initialize the blockchain info
def__init__(self):
self.chain= []
self.current_transaction= []
self.nodes=set()
# genesis block
self.new_block(previous_hash=1, proof=100)

defnew_block(self,proof,previous_hash=None):
block= {
'index': len(self.chain)+1,
'timestamp': time(), # timestamp from 1970
'transactions': self.current_transaction,
'proof': proof,
'previous_hash': previous_hashorself.hash(self.chain[-1])
 }
self.current_transaction= []
self.chain.append(block)
returnblock

defnew_transaction(self,sender,recipient,amount):
self.current_transaction.append(
 {
'sender' : sender,
'recipient' : recipient,
'amount' : amount

 }
 )
returnself.last_block['index'] +1

defregister_node(self, address):
parsed_url=urlparse(address)
self.nodes.add(parsed_url.netloc) # netloc attribute! network lockation

defvalid_chain(self,chain):
last_block=chain[0]
current_index=1
```

```
whilecurrent_index<len(chain):
block=chain[current_index]
print('%s'%last_block)
print('%s'%block)
print("₩n---------₩n")
# check that the hash of the block is correct
ifblock['previous_hash'] !=self.hash(last_block):
returnFalse
last_block=block
current_index+=1
returnTrue

defresolve_conflicts(self):
neighbours=self.nodes
new_chain=None

max_length=len(self.chain) # Our chain length
fornodeinneighbours:
tmp_url='http://'+str(node) +'/chain'
response=requests.get(tmp_url)
ifresponse.status_code==200:
length=response.json()['length']
chain=response.json()['chain']

iflength>max_lengthandself.valid_chain(chain):
max_length=length

ifnew_chain:
self.chain=new_chain
returnTrue

returnFalse

# directly access from class, share! not individual instance use it
@staticmethod
defhash(block):
block_string=json.dumps(block, sort_keys=True).encode()

returnhashlib.sha256(block_string).hexdigest()

@property
deflast_block(self):
returnself.chain[-1]

defpow(self, last_proof):
proof=0
whileself.valid_proof(last_proof, proof) isFalse:
proof+=1

returnproof
```

```

@staticmethod
defvalid_proof(last_proof, proof):
guess=str(last_proof+proof).encode()
guess_hash=hashlib.sha256(guess).hexdigest()
returnguess_hash[:4] =="0000"# nonce
```

- © 2021 GitHub, Inc.

server.py

```
fromflaskimportFlask, request, jsonify
importjson
fromtimeimporttime
fromtextwrapimportdedent
fromuuidimportuuid4

# Our blockchain.py API
fromblockchainimportBlockchain

# /transactions/new : to create a new transaction to a block
# /mine : to tell our server to mine a new block.
# /chain : to return the full Blockchain.
# /nodes/register : to accept a list of new nodes in the form of URLs
# /nodes/resolve : to implement our Consensus Algorithm

app=Flask(__name__)
# Universial Unique Identifier
node_identifier=str(uuid4()).replace('-','')

blockchain=Blockchain()

@app.route('/mine',methods=['GET'])
defmine():
last_block=blockchain.last_block
last_proof=last_block['proof']

proof=blockchain.pow(last_proof)
# print "DEBUGGING!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
blockchain.new_transaction(
sender='0',
recipient=node_identifier,
amount=1# coinbase transaction
 )
# Forge the new Block by adding it to the chain
previous_hash=blockchain.hash(last_block) # ??????????
block=blockchain.new_block(proof, previous_hash)

```

```
response= {
'message' : 'new block found',
'index' : block['index'],
'transactions' : block['transactions'],
'proof' : block['proof'],
'previous_hash' : block['previous_hash']
 }

returnjsonify(response), 200

@app.route('/transactions/new', methods=['POST'])
defnew_transaction():
values=request.get_json()

required= ['sender', 'recipient', 'amount']
ifnotall(kinvaluesforkinrequired):
return'missing values', 400

# Create a new Transaction
index=blockchain.new_transaction(values['sender'],values['recipient'],values['amount'])
response= {'message' : 'Transaction will be added to Block {%s}'%index}

returnjsonify(response), 201

@app.route('/chain', methods=['GET'])
deffull_chain():
response= {
'chain' : blockchain.chain,
'length': len(blockchain.chain),
 }

returnjsonify(response), 200

@app.route('/nodes/register', methods=['POST'])
defregister_nodes():
values=request.get_json()

nodes=values.get('nodes')
ifnodesisNone: # Bad Request 400
return"Error: Please supply a valid list of nodes", 400

fornodeinnodes:
blockchain.register_node(node)

response= {
'message' : 'New nodes have been added',
'total_nodes': list(blockchain.nodes),
 }
returnjsonify(response), 201
```

```
@app.route('/nodes/resolve', methods=['GET'])
defconsensus():
replaced=blockchain.resolve_conflicts() # True False return

ifreplaced:
response= {
'message' : 'Our chain was replaced',
'new_chain' : blockchain.chain
 }
else:
response= {
'message' : 'Our chain is authoritative',
'chain' : blockchain.chain
 }
returnjsonify(response), 200

if__name__=='__main__':
app.run(host='0.0.0.0', port=5000)
```

-

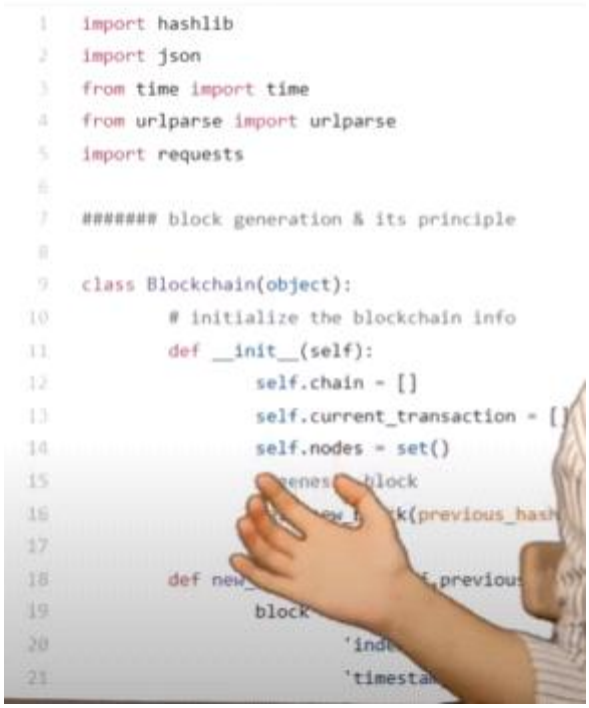
* 블록체인의 특징

1. 블록의 구조

Transaction
전 블록 hash
Time

2. 합의 과정

분산화된 환경에서
자료 동기화




'index' : len(self.chain)+1,
'timestamp': time(), # timestamp fr
'transactions': self.current_transa
'proof': proof,
'previous_hash': previous_hash or s
}
self.current_transaction = []
self.chain.append(block)
return block
def new_transaction(self,sender,recipient,amount):
self.current_transaction.append(
{
'sender' : sender
'recipient' :
'amount' :
}
)
return self.last_bloc
def register_node(self,
parsed_url = ur
self.nodes.add(
def valid_chain(self,chain):


blockchain.py
블록구조들
Transaction의 형태
chain과 연결법

```python
class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []

    def new_block(self):
        # Creates a new Block and adds it to the chain
        pass

    def new_transaction(self):
        # Adds a new transaction to the list of transactions
        pass

    @staticmethod
    def hash(block):
        # Hashes a Block
        pass

    @property
    def last_block(self):
        # Returns the last Block in the chain
        pass
```

index	몇 번째 블록인지
Timestamp	언제 블록이 생성되었는지
Transaction	거래 목록
Proof	마이닝의 결과
Prev_hash	블록의 무결성 위해

```
block = {
    'index': 1,
    'timestamp': 1506057125.900785,
    'transactions': [
        {
            'sender': "8527147fe1f5426f9dd545de4b27ee00",
            'recipient': "a77f5cdfa2934df3954a5c7c7da5df1f",
            'amount': 5,
        }
    ],
    'proof': 324984774000,
    'previous_hash': "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e730433629
}
```

Coi

__init__

```
def __init__(self):
        self.chain = []
        self.current_transaction = []

        # genesis block
        self.new_block(previous_hash=1, proof=100)
```

Genesis block 생성

Coin

new_block

```
def new_block(self,proof,previous_hash=None):
        block = {
                'index': len(self.chain)+1,
                'timestamp': time(), # timestamp from 1970
                'transactions': self.current_transaction,
                'proof': proof,
                'previous_hash': previous_hash or self.hash(self.chain[-1])
        }
        self.current_transaction = []
        self.chain.append(block)
        return block
```

Coin

new_transaction

```
def new_transaction(self,sender,recipient,amount):
        self.current_transaction.append(
                {
                        'sender' : sender,
                        'recipient' : recipient,
                        'amount' : amount
                }
        )
        return self.last_block['index'] + 1
```

송신자

수신자

금액

CoinS!ght

hash

```
def hash(block):
        block_string = json.dumps(block, sort_keys=True).encode()

        return hashlib.sha256(block_string).hexdigest()
```

```
import hashlib
import json
from time import time
from urlparse import urlparse
import requests
```

코드의 맨 앞 import 부분을 확인!

pow

```
def pow(self, last_proof):
    proof = 0
    while self.valid_proof(last_proof, proof) is False:
        proof += 1

    return proof
```

valid_proof라는 함수를 통해 맞을 때까지 반복적으로 검증

CoinS!ght

valid_proof

```
def valid_proof(last_proof, proof):
    guess = str(last_proof + proof).encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:4] == "0000" # nonce
```

전 proof와 구할 proof 문자열 연결

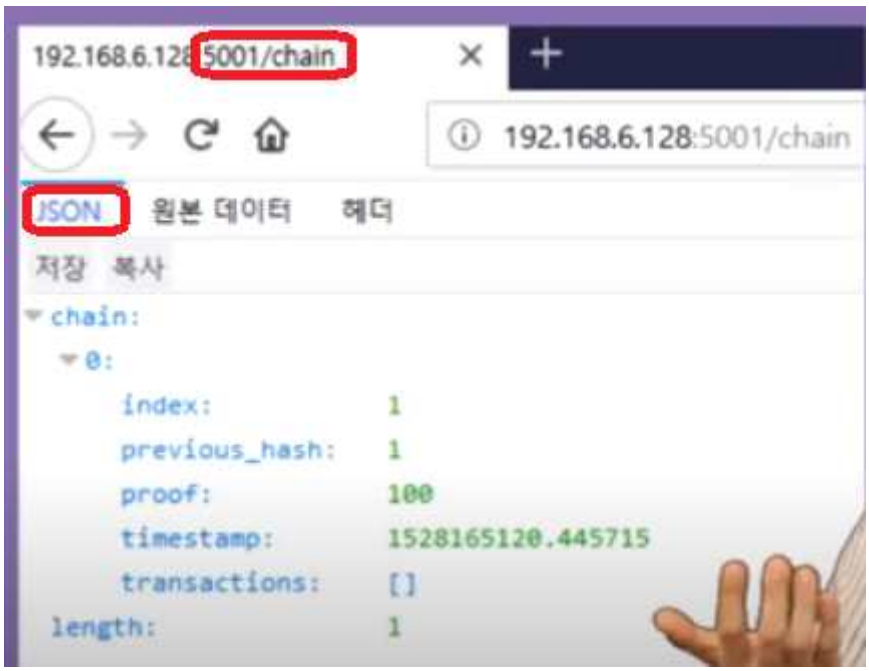
이 Hash값 저장

앞 4자리가 '0000'이면 True

/chain : 현재 블록체인 보여줌

/transaction/new : 새 트랜젝션 생성

/mine : server에게 새 블록 채굴 요청




```
if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000)
```

```
app = Flask(__name__)
# Universial Unique Identifier
node_identifier = str(uuid4()).replace('-','')
```

노드 식별을 하기 위해 uuid 함수를 사용!

```
blockchain = Blockchain()
```

```
blockchain = Blockchain()
```

짜놓은 블록체인 객체를 선언!

/chain

```
def full_chain():
        response = {
                'chain' : blockchain.chain,
                'length': len(blockchain.chain),
        }

        return jsonify(response), 200
```

json 형태로 리턴

```
return jsonify(response), 200
```

200은 웹 사이트 에러가 없을 때

* POST?

url에 데이터를 붙이는
GET방식과 다르게
숨겨서 보내는 방식

CoinS!ght

/transaction/new

```
def new_transaction():
    values = request.get_json()
```

json형태를 받아서 저장

```
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'missing values', 400
```

해당 데이터가 존재해야함

```
    # Create a new Transaction
    index = blockchain.new_transaction(values['sender'],values['recipient'],values['amount'])
    response = {'message' : 'Transaction will be added to Block {%s}' % index}

    return jsonify(response), 201
```

json형태로 반환

Coi

/mine

```
def mine():
    last_block = blockchain.last_block
    last_proof = last_block['proof']

    proof = blockchain.pow(last_proof)

    blockchain.new_transaction(
        sender='0',
        recipient=node_identifier,
        amount=1 # coinbase transaction
    )
    # Forge the new Block by adding it to the chain
    previous_hash = blockchain.hash(last_block)
    block = blockchain.new_block(proof, previous_hash)
```

채굴 시 생성되는 transaction

```
)
# Forge the new Block by adding it to the chain
previous_hash = blockchain.hash(last_block)
block = blockchain.new_block(proof, previous_hash)

response = {
```

전 블록에 대한 hash

블록을 새로 생성

```
response = {
        'message' : 'new block found',
        'index' : block['index'],
        'transactions' : block['transactions'],
        'proof' : block['proof'],
        'previous_hash' : block['previous_hash']
}

return jsonify(response), 200
```

블록이 생성되었다는 메세지를 json형태로 띄워줌

/mine

```
def mine():
        last_block = blockchain.last_block
        last_proof = last_block['proof']

        proof = blockchain.pow(last_proof)

        blockchain.new_transaction(
                        sender='0',
                        recipient=node_identifier,
                        amount=1 # coinbase transaction
        )
        # Forge the new Block by adding it to the chain
        previous_hash = blockchain.hash(last_block)
        block = blockchain.new_block(proof, previous_hash)

        response = {
                'message' : 'new block found',
                'index' : block['index'],
                'transactions' : block['transactions'],
                'proof' : block['proof'],
                'previous_hash' : block['previous_hash']
        }

        return jsonify(response), 200
```

https://github.com/tr0y-kim/ez_blockchain

4:00부터 영상에 등장하는 코드는
https://github.com/tr0y-kim/ez_blockc... 에서 확인하실 수 있습니다!.