MemeEconomy (Codename Order66)

Release 0.5

Akshay Katyal, Anant Sujatanagarjuna, Chris Warin, Mehmed Mus

Jul 08, 2020

CONTENTS:

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

1.1 block.py

Defines the Block structure

class block.**Block**(*index*, *minerID*, *transactions*, *transaction_counter*, *timestamp*, *previous_hash*, *proof_of_work=0*)

Class that handles the functions and defines the structure of Blocks in the Blockchain

compute_hash()

Computes Hash of the Block

get_transactions()

Returns list of transactions within this Block

1.2 blockchain.py

Module that handles the blockchain

class blockchain.**Blockchain**

Class that maintains the functions and structure of the Blockchain

add_transaction(*transaction*)

Adds a transaction to the list of pending transactions (mempool)

append_block(*block*, *proof*)

Appends a block to the chain after verifying it's validity

classmethod check_validity(*chain*)

Checks whether the current chain is valid or not

create_naked_block(*_minerID*)

Create a block without valid nonce

create_origin_block()

The block has empty list of transactions The block has 0 as a value for index, previous_hash, proof_of_work

difficultyPattern = '000'
difficulty level of the Proof of Work shows the pattern with which each hash has to start with

find_image(*imageId*)
Checks whether the blockchain contains image with imageId if exists, returns image's decoded ascii value, otherwise -1

classmethod is_proof_valid(*block*, *block_hash*)
Checks whether the hash value of a block is valid and satisfies the difficulty pattern or not

pending_transactions()
Checks whether there are pending transactions or not

previous_block()
Returns the previous block of the chain

static proof_of_work(*block*)
Finds a value for proof_of_work which produces a hash that satisfies the difficulty pattern

blockchain.**consensus_mechanism**(*_chain*, *_connected_nodes*)
Consensus mechanism to make sure that the nodes in the network always have the longest (valid) chain

A basic algorithm which sends /get_chain requests to all other connected nodes in the network. If a longer chain is found, current node's chain is replaced in order to keep the blockchain up-to-date

blockchain.**construct_chain_again**(*json_chain*)
A function which builds chain and transactions structure from the json data

1.3 validation.py

Module that validates block transactions

exception validation.**BlockException**(*transactionExceptions=[]*)
Exception that is raised when TransactionExceptions occur when validating a Block

exception validation.**MemeFormatHasPendingSaleOfferException**(*memeFormatID*, *transactionID*)
Exception raised when a node tries to add an Ownership Sale offer when their previous offer for the same MemeFormat is still pending, and has no buyer.

exception validation.**MemeFormatNotFoundException**(*memeFormatID*, *transactionID*)
Exception raised when the specified MemeFormat is not found in node_state

exception validation.**MemeFormatNotOwnedByNodeException**(*nodeID*, *memeFormatID*, *transactionID*)
Exception raised when a node attempts to sell ownership to a MemeFormat that it does not own

exception validation.**MemeNotFoundException**(*memeID*, *transactionID*)
Exception raised when the specified Meme is not found in node_state

exception validation.**NodeNotFoundException**(*nodeID*, *transactionID*)
Exception raised when the specified Node is not found in node_state

exception validation.**OwnershipPurchaseFailedNoCreditsException**(*nodeID*, *memeFormatID*, *transactionID*, *message*)
Exception raised when OwnershipPurchase fails due to the buyer not having enough credits

exception validation.**OwnershipSaleAmountNotPositiveException**(*ownershipSaleOfferID*, *saleAmount*, *transactionID*)
Exception raised when OwnershipSaleOffer amount is non-positive

exception validation.**OwnershipSaleOfferAlreadyAcceptedException**(*ownershipSaleOfferID*, *nodeID*, *blockID*, *transactionID*)
Exception raised when node attempts to buy ownership based on a ownership sale offer that was already accepted

exception validation.**OwnershipSaleOfferNotFoundException**(*ownershipSaleOfferID*, *transactionID*)
Exception raised when Ownership Sale Offer is not found in node_state

exception validation.**TransactionException**(*transactionID*, *message*)
Exception raised when an exception occurs validating a transaction

exception validation.**UpvoteFailedNoCreditsException**(*nodeID*, *transactionID*, *message*)
Exception raised when Upvote transaction fails to proceed due to the upvoter not having enough credits.

validation.**apply_block**(*block*, *commit=False*)
Try to update node_state based on all the transactions in the block. If commit is False, then it will revert node_state, else it will commit node_state

validation.**apply_memeFormat_transaction**(*transaction_data*, *block_ID*, *miner_ID*, *just_validate=True*)
Update node_state based on a memeFormat transaction

validation.**apply_meme_transaction**(*transaction_data*, *block_ID*, *miner_ID*, *just_validate=False*)
Update node_state based on a meme transaction

validation.**apply_ownership_purchase_transaction**(*transaction_data*, *block_ID*, *miner_ID*, *just_validate=False*)
Update node_state based on a ownership purchase transaction

validation.**apply_ownership_sale_offer_transaction**(*transaction_data*, *block_ID*, *miner_ID*, *just_validate=False*)
Update node_state based on a ownership sale offer transaction

validation.**apply_transaction**(*transaction_data*, *block_ID*, *miner_ID*, *just_validate=False*)
Update node_state based on transaction_data

validation.**apply_upvote_transaction**(*transaction_data*, *block_ID*, *miner_ID*, *just_validate=False*)
Update node_state based on a memeFormat transaction

1.4 node_state.py

Module that keeps track of state such as wallet amounts of Nodes for making validation of transactions easy

node_state.**BUY_TRANSACTION_MINER_REWARD = Decimal('0.05')**
Percentage, (in the form of a fraction) of the successful sale of ownership credited to the miner of the Buy transaction

node_state.**MEME_FORMAT_MINER_REWARD = Decimal('0.10')**
Percentage, (in the form of a fraction) of upvote credits rewarded to the miner who mined to MemeFormat.

node_state.**MEME_FORMAT_OWNER_PORTION = Decimal('0.30')**
Percentage, (in the form of a fraction) of upvote credits claimed by meme owner.

node_state.**MEME_MINER_PORTION = Decimal('0.10')**
Percentage, (in the form of a fraction) of upvote credits claimed by node that mined the meme.

node_state.**MEME_POSTER_PORTION = Decimal('0.60')**
Percentage, (in the form of a fraction) ofupvote credits claimed by node that posted meme.

class node_state.**Meme**(*ID*, *title*, *meme_format*, *binary*, *poster_ID*, *block_ID*, *miner_ID*, *extension='jpg'*)
Class that handles all functions pertaining to maintaining state of a Meme.

__init__(*ID*, *title*, *meme_format*, *binary*, *poster_ID*, *block_ID*, *miner_ID*, *extension='jpg'*)
ID : Uniquely identifiable ID for the Meme

title : Some string Title for the Meme

meme_format : ID of the meme_format

binary : binary bits of the meme

poster_ID : ID of node that posted meme

block_ID : ID of block which contains the transaction posting the meme

miner_ID : ID of miner node who created the block with block_ID

__repr__()
Return repr(self).

add_upvote(*upvote_ID*)
Add upvote to the Meme

reward_upvoters(*block_ID*)
Reward upvoters who upvoted before block *block_ID*. All upvotes in the block should already be added using Meme.add_upvote

class node_state.**MemeFormat**(*ID*, *name*, *description*, *binary*, *owner*, *miner*)
Class that handles all functions pertaining to maintaining state of a MemeFormat.

__init__(*ID*, *name*, *description*, *binary*, *owner*, *miner*)
ID : Uniquely Identifiable ID for the MemeFormat

name : Any Display Name for the MemeFormat

description : Some textual description of the MemeFormat

binary : binary data of a meme example, possible related to the description

owner : ID(s) of the node(s) that own the MemeFormat

miner : ID(s) of the node that mined the MemeFormat

__repr__()
Return repr(self).

add_meme(*meme_ID*)
Add meme to MemeFormat

add_ownership_sale_offer(*ownershipSaleOfferID*)
Add Ownership Sale Offer to MemeFormat

class node_state.**Node**(*ID*, *credits*)
Class that handles all functions pertaining to maintaining state of a Node.

`__init__`(*ID*, *credits*)
Initialize self. See help(type(self)) for accurate signature.

`__repr__`()
Return repr(self).

`add_meme`(*meme_ID*)
Add a meme to the node

`add_meme_format`(*meme_format_ID*)
Add a Meme format to the Node ownership : percentage of ownership of meme_format

`add_upvote`(*upvote_ID*)
Add an upvote to the node

class `node_state.`**`OwnershipSaleOffer`**(*ownershipSaleOfferID*, *sellerID*, *memeFormatID*, *sellBlockID*, *sellBlockMinerID*, *amount=0*)
Class that handles mehthods pertaining to OwnershipSaleOffer

`__init__`(*ownershipSaleOfferID*, *sellerID*, *memeFormatID*, *sellBlockID*, *sellBlockMinerID*, *amount=0*)
Initialize self. See help(type(self)) for accurate signature.

`__repr__`()
Return repr(self).

`buy`(*buyerID*, *buyBlockID*, *buyBlockMinerID*, *discredit_only=False*)
Method that handles the buying of Ownership based on the ownership Sale offer

`node_state.`**`SELL_TRANSACTION_MINER_REWARD = Decimal('0.05')`**
Percentage, (in the form of a fraction) of the successful sale of ownership credited to the miner of the Sell transaction

`node_state.`**`UPVOTE_MINER_REWARD = Decimal('0.10')`**
Percentage, (in the form of a fraction) of upvote credits rewarded to miner who mined the upvote.

`node_state.`**`UPVOTE_REWARD = Decimal('0.10')`**
Percentage, (in the form of a fraction) of upvote credits rewarded to upvoters from previous block.

class `node_state.`**`Upvote`**(*ID*, *meme_ID*, *upvoter_ID*, *block_ID*, *miner_ID*, *credits=1*, *discredit_only=False*)
Class that handles all the functions pertaining to maintaining state of an Upvote

`__init__`(*ID*, *meme_ID*, *upvoter_ID*, *block_ID*, *miner_ID*, *credits=1*, *discredit_only=False*)
Initializes the upvote and transfers appropriate credits to meme poster, MemeFormat owner. Also rewards the UpvoteMiner, MemeMiner, MemeFormatMiner

`__repr__`()
Return repr(self).

`node_state.`**`backup_state`**()
Create backup of node_state

`node_state.`**`commit_state`**()
Commit node_state

`node_state.`**`fresh_state`**()
Create a fresh empty node_state

`node_state.`**`revert_state`**()
Revert node_state to backup

1.5 wallet.py

Module that handles operations relating to a node’s wallet

exception wallet.**NotEnoughCreditsException**(*wallet_ID*, *current_amount*, *discredit_amount*)

Exception raised when a wallet does not have enough credits for discredit operation.

class wallet.**Wallet**(*ID*, *credits=0*)

Class that handles all functions pertaining to a node’s crypto wallet. Objects of this class are never transmitted, but stored locally for ease of validating transactions and blocks.

credit_amount(*credits*)

Use this function to credit the wallet with a certain amount of credits

discredit_amount(*credits*)

Use this function to discredit the wallet with a certain amount of credits

1.6 atomic.py

Module that makes allows for changing the instance variables of multiple objects in one psuedo *atomic* operation.

class atomic.**Atomic**

Class that implements the psuedo ‘atomic’ operations of objects

commit()

Commit state of object’s instance variables into the __var_backup__

revert()

Revert state of object’s instance variables to values stored in the __var_backup__

atomic.**commit**()

Commit all objects currently tracked by the __initialized_objects__ list

atomic.**revert**()

Revert all objects currently tracked by the __initialized_objects__ list

PYTHON MODULE INDEX

INDEX