Simple Blockchain with Python

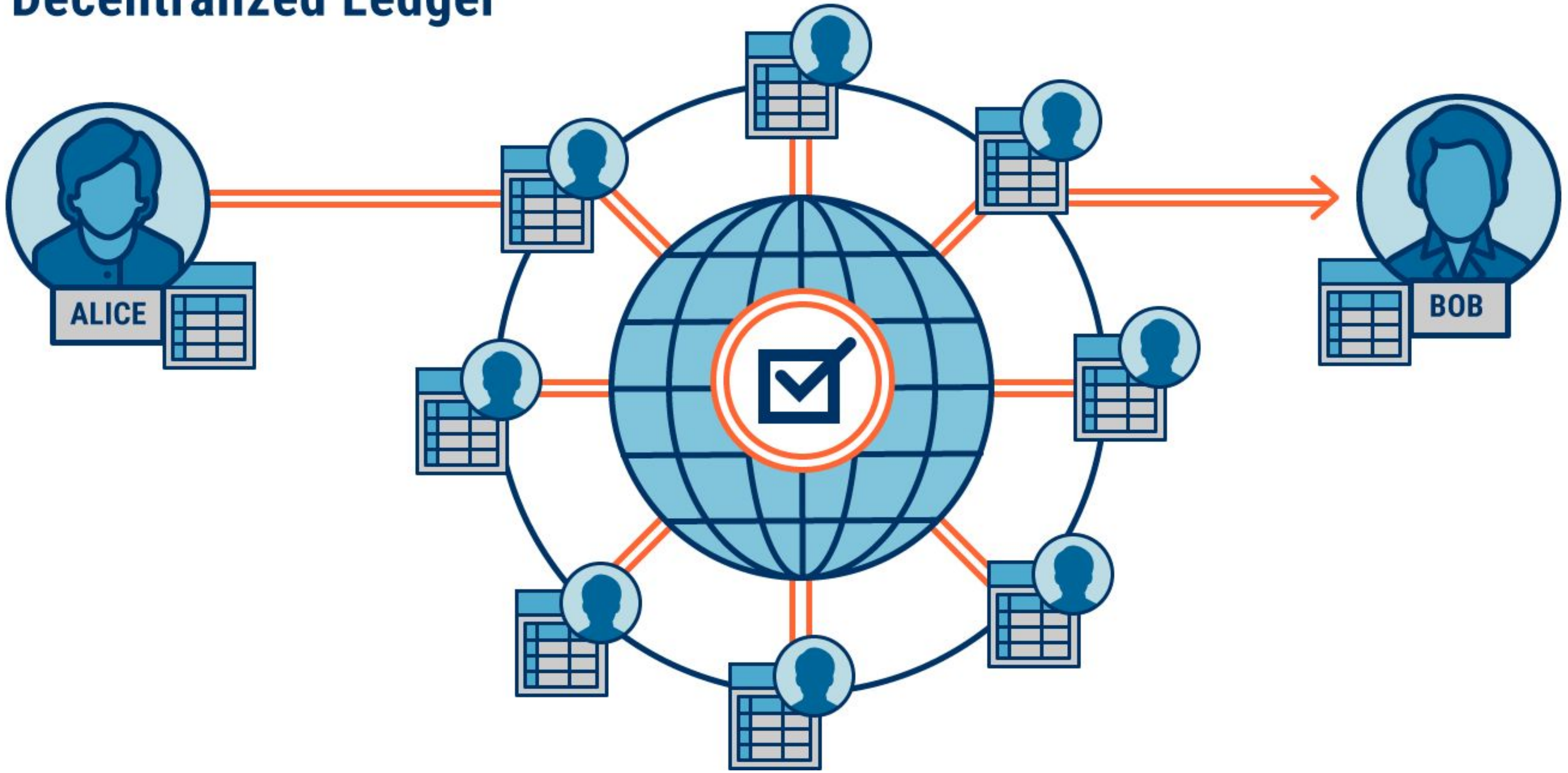
Lesson content

1. What is Blockchain?
   a. Decentralization
   b. Transparency
   c. Immutability
2. Prerequisites
3. Import dependencies
4. Create a block
5. SHA-256
6. Create a blockchain
7. Print out the blockchain
8. Demo
9. Links

What is Blockchain?

- Blockchain – a time-stamped series of immutable record of data that is managed by cluster of computers not owned by any single entity.
- The blockchain network has no central authority.
- It is shared and immutable ledger, the information in it is open for anyone and everyone to see.
- The three main properties which has helped it gain widespread acclaim are as follows:
    - Decentralization
    - Transparency
    - Immutability

Decentralized Ledger

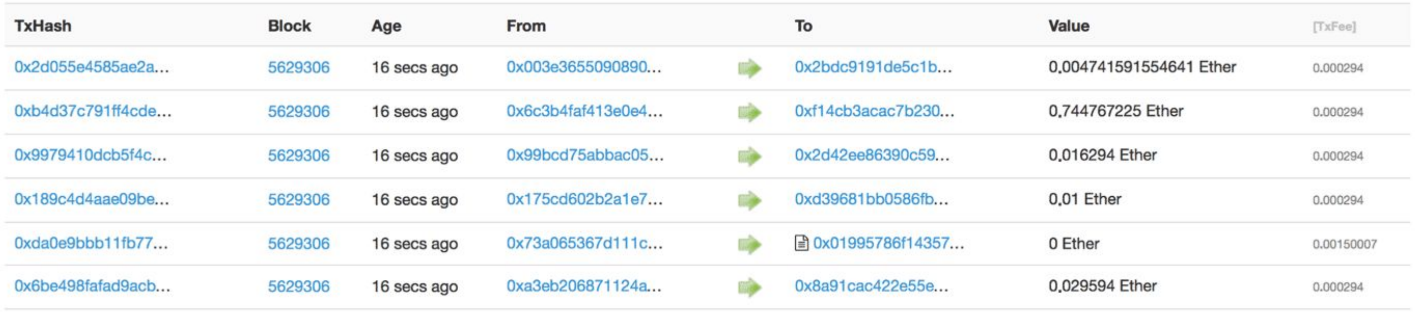


Decentralization

- Traditional centralized systems have several vulnerabilities:
  - All data is stored in one spot. This makes centralized systems easy target spots for potential hackers.
  - If the centralized system were to go through a software upgrade, it would halt the entire system.
  - If the centralized entity somehow shut down, nobody will be able to access the information.
  - Worst case scenario, what if the entity gets corrupted and malicious? If that happens then all the data will be compromised.
- In a decentralization system, the information is not stored by one single entity, everyone in the network owns the information.
- If you want to interact with your friend, you can do so directly, without going through a third party.

Transparency

- A person’s identity is hidden via complex cryptography and represented only by their public address. So, if you will look at a person’s transaction history, you will not see “Rob sent 1 BTC”, instead you will see “1MF1bhsFLkBzzz9vpFYEmvwT2TbyCt7NZJ sent 1 BTC”.
- So, while the person’s real identity is secure, you will still see all the transactions that were done by their public address.



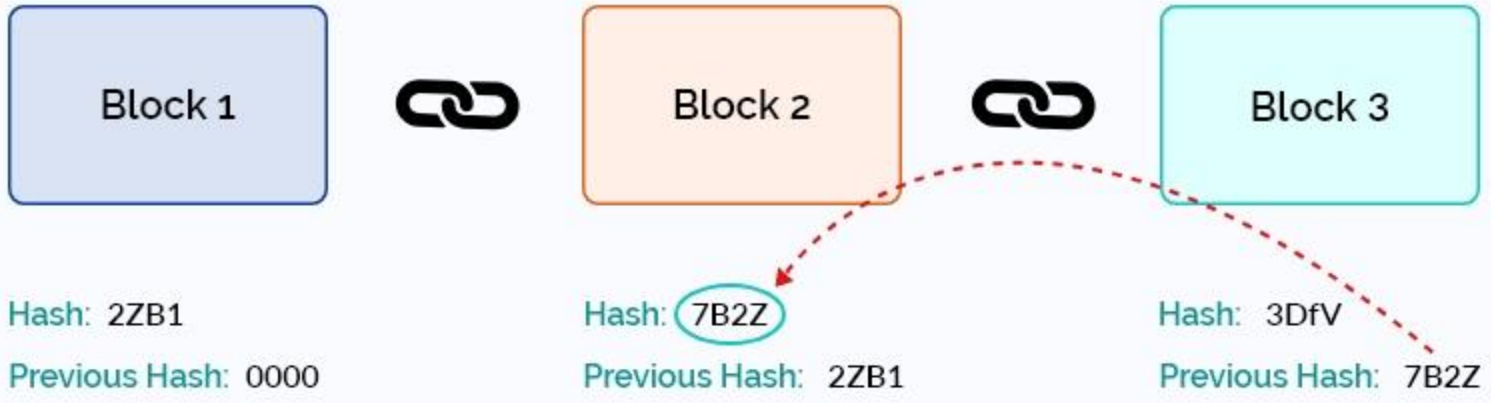
TxHash	Block	Age	From	To	Value	[TxFee]
0x2d055e4585ae2a...	5629306	16 secs ago	0x003e3655090890...	0x2bdc9191de5c1b...	0.004741591554641 Ether	0.000294
0xb4d37c791ff4cde...	5629306	16 secs ago	0x6c3b4faf413e0e4...	0xf14cb3acac7b230...	0.744767225 Ether	0.000294
0x9979410dcb5f4c...	5629306	16 secs ago	0x99bcd75abbac05...	0x2d42ee86390c59...	0.016294 Ether	0.000294
0x189c4d4aae09be...	5629306	16 secs ago	0x175cd602b2a1e7...	0xd39681bb0586fb...	0.01 Ether	0.000294
0xda0e9bbb11fb77...	5629306	16 secs ago	0x73a065367d111c...	0x01995786f14357...	0 Ether	0.00150007
0x6be498fafad9acb...	5629306	16 secs ago	0xa3eb206871124a...	0x8a91cac422e55e...	0.029594 Ether	0.000294

Immutability

- Immutability, in the context of the blockchain, means that once something has been entered into the blockchain, it cannot be tampered with.
- The reason why the blockchain gets this property is that of cryptographic hash function.
- Each block of information, such as facts or transaction details, proceed using a cryptographic principle or a hash value. That hash value consists of an alphanumeric string generated by each block separately.
- Every block not only contains a hash or digital signature for itself but also for the previous one. This ensures that blocks are retroactively coupled together and unrelenting. This functionality of blockchain technology ensures that no one can intrude in the system or alter the data saved to the block.

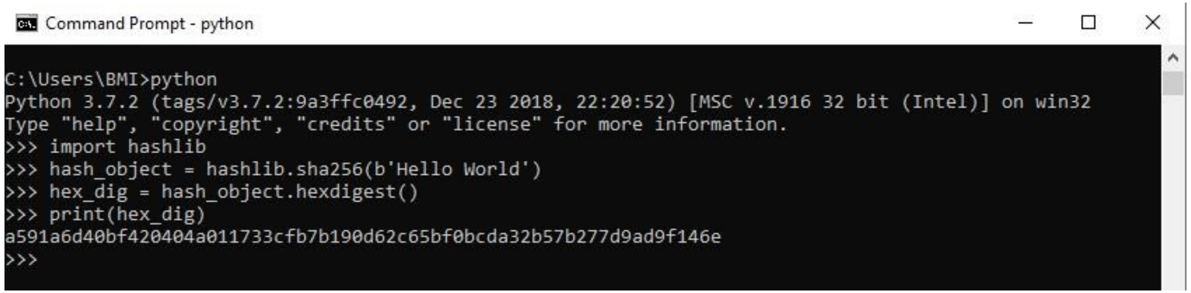
What is block?

- “Blocks” on the blockchain are made up of digital pieces of information. Specifically, they have three parts:
  - Blocks store information about transaction like the date, time, and the amount of money of your most recent purchase (for example).
  - Blocks store information about who is participating in transactions.
  - Blocks store information that distinguishes them from other blocks. Much like you and I have names to distinguish us from one another, each block stores a unique code called a “hash” that allows us to tell it apart from every other block.


Block 1
Block 2
Block 3
Hash: 2ZB1
Previous Hash: 0000
Hash: 7B2Z
Previous Hash: 2ZB1
Hash: 3DfV
Previous Hash: 7B2Z

SHA-256

- One of the most popular cryptographic hash algorithms in the blockchain space.
- A cryptographic hash is a kind of “signature” for a text or a data file. SHA-256 generates an almost-unique 256-bit (32-byte) signature for a text.
- Let’s generate a cryptographic hash in Python code:



```
C:\Users\BMI>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> hash_object = hashlib.sha256(b'Hello World')
>>> hex_dig = hash_object.hexdigest()
>>> print(hex_dig)
a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e
>>>
```

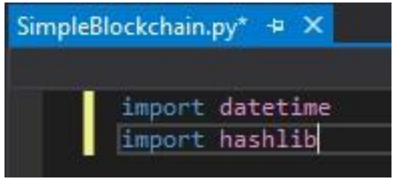
Prerequisites

- Python 3.3 or greater https://www.python.org/downloads/
- Visual Studio 2019 or another IDE https://visualstudio.microsoft.com/vs/

Step 1 - Import dependencies

We will use two standard Python libraries – **datetime** to generate timestamps and **hashlib** that contains hashing algorithm.

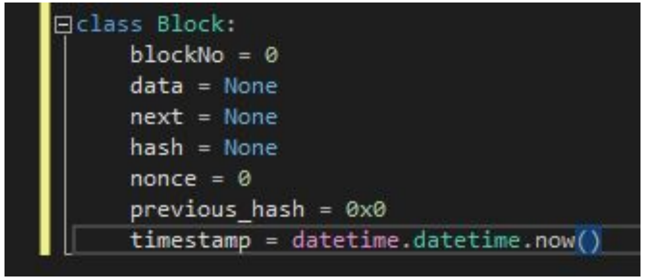
Let’s create a new .py file SimpleBlockchain and import them:



```
import datetime
import hashlib
```

Step 2 - Create a block

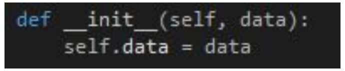
- Each block will have 7 attributes:
    - block number,
    - data,
    - pointer to the next block,
    - the hash of this block,
    - a nonce (a number only used once),
    - the previous hash and timestamp.
- Let’s create a class Block and define all these attributes in our file:



```
class Block:
        blockNo = 0
        data = None
        next = None
        hash = None
        nonce = 0
        previous_hash = 0x0
        timestamp = datetime.datetime.now()
```

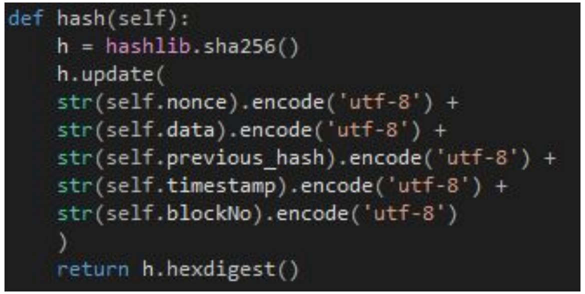
Step 2 - Create a block

- We need to initialize a block by storing some data in it:



```python
def __init__(self, data):
    self.data = data
```

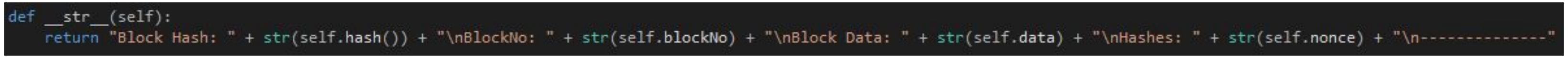
- Now we need to write a function that will compute the hash of the block. A hash acts as both a unique identifier and verifies its integrity. If someone changes the hash of a block, every block that comes after it is changed. This helps make a blockchain immutable.
- The input of the SHA-256 algorithm will be a concatenated string consisting of 5 block attributes: the nonce, data, previous hash, timestamp, and a number of a block:



```python
def hash(self):
    h = hashlib.sha256()
    h.update(
    str(self.nonce).encode('utf-8') +
    str(self.data).encode('utf-8') +
    str(self.previous_hash).encode('utf-8') +
    str(self.timestamp).encode('utf-8') +
    str(self.blockNo).encode('utf-8')
    )
    return h.hexdigest()
```

Step 2 - Create a block

- Now we need to print out the value of the block:

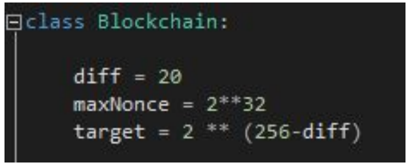


```python
def __str__(self):
    return "Block Hash: " + str(self.hash()) + "\nBlockNo: " + str(self.blockNo) + "\nBlock Data: " + str(self.data) + "\nHashes: " + str(self.nonce) + "\n--------------"
```

- Our Block class is done! We can launch the file (don't forget to save it first!) and see what we've done so far:

Step 3 - Create a Blockchain

- Now we need to define the blockchain data structure consists of blocks which should be linked together to form a “chain”.
- Let’s create a new class Blockchain and define mining difficulty, a maximum number that we can store in a 32-bit number and a a target hash for mining:



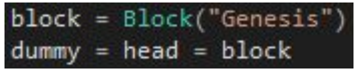
```
class Blockchain:

    diff = 20
    maxNonce = 2**32
    target = 2 ** (256-diff)
```

Mining

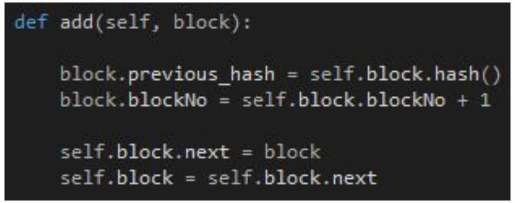
- It’s a process of adding transaction records to the public ledger of past transactions (blockchain).
- The blockchain serves to confirm transactions to the rest of the network as having taken place.
- Mining is intentionally designed to be resource-intensive and difficult so that the number of blocks found each day by miners remains steady.
- Individual blocks must contain a proof of work to be considered valid.
- The primary purpose of mining is to set the history of transactions in a way that is computationally impractical to modify by any one entity.
- **The difficulty** is the measure of how difficult it is to find a new block compared to the easiest it can ever be.

Step 3 - Create a Blockchain

- First, let’s generate the very first block in our blockchain - “genesis block” - and set it as a head of the blockchain: 

```python
block = Block("Genesis")
dummy = head = block
```

- Now let’s create a function **add** which will add the given block to the blockchain, set the hash of a given block as our new block’s previous hash, and set the next block equal to itself:



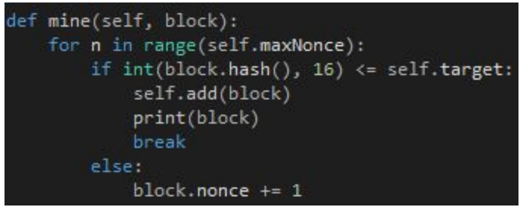
```python
def add(self, block):

    block.previous_hash = self.block.hash()
    block.blockNo = self.block.blockNo + 1

    self.block.next = block
    self.block = self.block.next
```

Step 3 - Create a Blockchain

- Now let's determine whether or not we can add a given block to the blockchain. For it we need to check if hash of a given block is less than a specific target number:

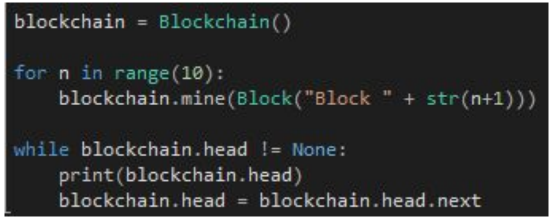


```
def mine(self, block):
    for n in range(self.maxNonce):
        if int(block.hash(), 16) <= self.target:
            self.add(block)
            print(block)
            break
        else:
            block.nonce += 1
```

- Let's launch or .py file and check, what we've done so far

Step 4 - Print the Blockchain

- In the final step we should initialize blockchain, mine 10 block (for example, you can change it to another number), and print out each block in blockchain:



```
blockchain = Blockchain()

for n in range(10):
    blockchain.mine(Block("Block " + str(n+1)))

while blockchain.head != None:
    print(blockchain.head)
    blockchain.head = blockchain.head.next
```

- Our simple Blockchain is ready to go! Let’s launch the file **SimpleBlockchain.py**.