Mokastudy.tk

CodeCademy - Introduction to Blockchain

Table des matières

INTRODUCTORY BLOCKCHAIN CONCEPTS

Why Blockchain?

In 2010, a programmer paid 10,000 Bitcoins for 2 pizzas, roughly worth $30. In 2018, that same number of bitcoins is estimated at $83 million in value!

The exchange of Bitcoin is possible due to an underlying technology that secures and simplifies transactions removing the need for a bank or a central authority. Anyone with an internet connection has the freedom to own and exchange this digital currency. The powerful architecture that drove this revolution was blockchain. Businesses started to realize the potential of blockchain and are rapidly mobilizing to understand and implement it. But, what exactly is blockchain and what makes it so transformative?

After you complete this course, you will have a basic foundation of blockchain principles. You will also have the opportunity to create your own mini blockchain where you will transform these concepts into code.

What is Blockchain?

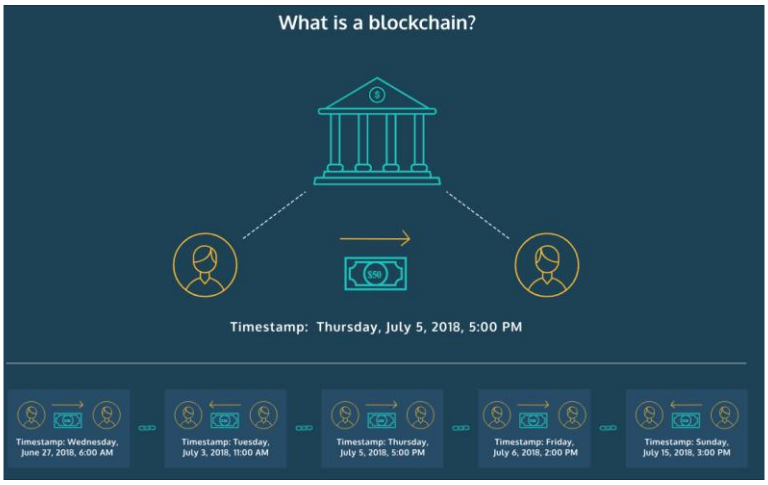
The blockchain is similar to a permanent book of records that keeps a log of all transactions that have taken place in chronological order.

Let's envision a bank transaction in which there are three parties: the sender, the bank, and the recipient. In order to ensure that there are no fraudulent transactions, the bank acts as the central authority between the parties.

The blockchain also logs transactions between senders and receivers, except there is no bank or central authority. Instead, the blockchain relies on a public network of computers to verify the transaction. The blockchain is just an accurate, and permanent record of all the transactions that have happened amongst the members in that blockchain's network. In this analogy, each block in the blockchain represents a transaction, and each transaction is connected to the prior transaction to form the entire connected blockchain.

Key Terms:

- Block: A block is an individual transaction or piece of data that is being stored within the blockchain.
- Blockchain: A blockchain is a continuously growing list ("chain") of records ("block"), called blocks, which are linked chronologically and secured using cryptography.


What is a blockchain?
$50
Timestamp: Thursday, July 5, 2018, 5:00 PM
Timestamp: Wednesday, June 27, 2018, 6:00 AM
Timestamp: Tuesday, July 3, 2018, 11:00 AM
Timestamp: Thursday, July 5, 2018, 5:00 PM
Timestamp: Friday, July 6, 2018, 2:00 PM
Timestamp: Sunday, July 15, 2018, 3:00 PM

The Blockchain Network

So how do Blockchain-based applications like Bitcoin and Ethereum validate transactions without a central authority?

In the blockchain, there are many participants in the network that are constantly checking to ensure that each transaction is valid. Each participant is a computer that owns a copy of the blockchain. These participants cross-reference their copy of the blockchain each time a new block is being introduced. Because this validation depends on multiple participants, the digital record is “decentralized”.

In order for a new block to be added, 51% of all of the participants in the blockchain network must verify that the new block is not fraudulent. Once a block has been verified as a valid transaction, it is added to each participant's copy of the blockchain.

By having the majority of participants validate a new transaction, the blockchain removes the need for a central authority and automates the completion of transactions, reducing transaction fees while ensuring a high level of security.

Key Terms:

- Blockchain Network: The blockchain network and blockchain are terms used interchangeably. They represent the entire blockchain from the structure itself to the network that it is a part of.
- Decentralization: The concept in which participants work together to validate transactions without relying on a central authority.
- Participant: A client that owns a copy of the blockchain and verifies transactions across the network.

What are Blocks in the Blockchain?

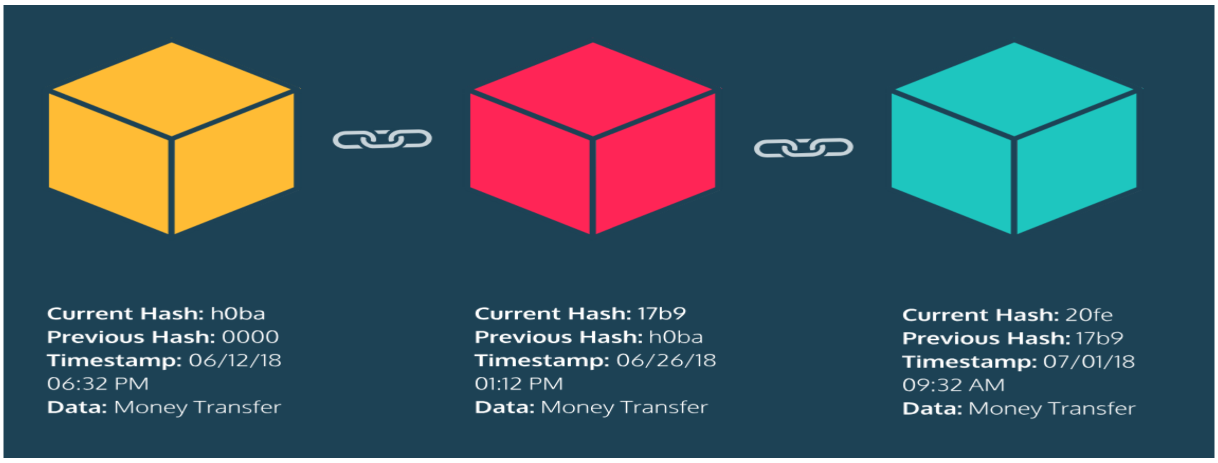
Just like bricks are the building blocks of a house, blocks themselves are the building blocks of a blockchain.

A block contains transaction data and other important details related to the creation of that block, such as the time when it was created and other unique information. In order to create a block, we must have a record that we wish to store.

In this lesson, we will be discussing transaction data. This is just one example, but blocks can hold different types of data depending on what the blockchain is used for. However, at its heart, a block will always contain a timestamp or the data regarding the time when the block was created. A block will also always contain a unique hash or a unique code produced by combining all the contents within the block itself — also known as a digital fingerprint. Lastly, a block will also always contain a previous hash or a reference to the prior block’s hash. This reference to the prior block is how blocks chain to one another. We will dive into more details about the hash later. These attributes about a block are what make it part of a blockchain structure.

Properties in a Block:

- Timestamp: The time the block is created determines the location of it on the blockchain.
- Transaction Data: The information to be securely stored in the block.
- Hash: A unique code produced by combining all the contents within the block itself — also known as a digital fingerprint.
- Previous Hash: Each block has a reference to the block prior to its hash. This is what makes the blockchain unique because this link will be broken if a block is tampered with.



Hashing

Hashing is an application of cryptography that is fundamental to the design of the blockchain. It is a way to generate a seemingly random, but calculated string of letters and numbers from any input. This is accomplished by the use of a hash function.

Think of a hash function as a machine that takes an object, such as an apple, and turns it into a varying combination of letters and numbers, such as “1a432bf”. The output (“1a432bf”) is known as the hash of the input, the apple. If you give the machine two apples instead of one, it will return a different hash (such as “26f5ce1”).

There are many types of machines out there, so the resulting hash varies from machine to machine. Similarly, there are many types of hash functions available. Blockchain uses a cryptographic hash function, meaning that the output is random but deterministic. This means the same input will always produce the same hash. That process is one-way, so the output (hash) cannot be used to produce the original input.

Key Terms:

- Deterministic: The same input will always produce the same output, but that output cannot produce the original input.
- Hash: A calculated string of letters and numbers produced from a specific input.
- Hash Function: A function that takes in an input of a random size, performs hashing on the input, and generates a seemingly random output of a fixed size, also known as the hash

The Genesis Block

To recap, a blockchain is similar to a permanent book of records — it keeps an accurate unchanging record of all data, or transactions, stored in chronological order. Each block has a reference to the block's previous hash. This is how blocks are "chained" together. If a block's contents are tampered with, the block's hash changes and the chain breaks, making it difficult to successfully tamper with any one piece of the chain.

Since all blocks in the blockchain have a reference to the previous block, the first block is a little different than the rest. It is known as the Genesis Block. The problem, however, is that the Genesis Block does not have a block before it. So it wouldn't make sense to have a previous hash stored inside it. To resolve this minor issue, the value of the previous hash is typically hard-coded into the Genesis Block with the default value of zero.
(https://s3.amazonaws.com/codecademy-content/courses/blockchain/Lesson+1/final_genesis.html)

Key Terms:

- Genesis Block:
  The genesis block is the first block on the blockchain and it is typically hard-coded into the blockchain structure. Being the first block on the blockchain, it does not have a link to a previous hash.

Visualizing Blockchain

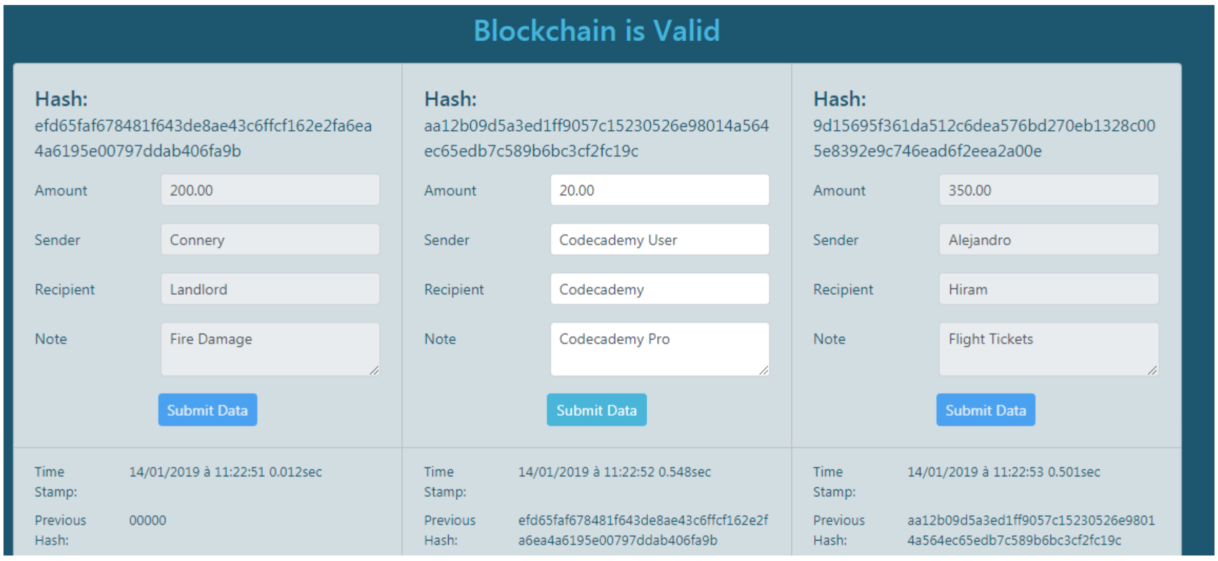
Below is a review of important terms that you may want to study to further solidify your knowledge on the blockchain.

Let's remember the vocabulary we learned

- Blockchain: A blockchain is an accurate and permanent record of transactions that have been verified and stored in a chronological sequence.
- Blocks: A block is an individual transaction or piece of data that is being stored within the blockchain.
- Blockchain Network: The blockchain network and blockchain are terms used interchangeably. They represent the entire blockchain from the structure itself to the network that it is a part of.
- Decentralization: The concept in which users work together to validate transactions without relying on a central authority.
- Participant: A client that owns a copy of the blockchain and verifies transactions across the network.
- Deterministic: The same input will always produce the same output, but that output will never produce the original input.
- Hash: A fixed-length string of a varying combination of letters and numbers produced from a specific input of arbitrary size.
- Hash Function: A function that takes in an input of a random size, performs hashing on this input, and generates a random output of a fixed size, also known as the hash.
- Genesis Block: The genesis block is the first block on the blockchain and it is typically hard-coded into the blockchain structure. Being the first block on the blockchain, it does not have a link to a previous hash.

Block Properties

- Timestamp: The time the block is created determines the location of it on the blockchain.
- Data: The information to be securely stored in the block.
- Hash: A unique code produced by combining all the contents within the block itself — also known as a digital fingerprint.
- Previous Hash: Each block has a reference to the block prior to its hash. This is what makes the blockchain unique because this link will be broken if a block is tampered with.



Blockchain is Valid

Hash: efd65faf678481f643de8ae43c6ffcf162e2fa6ea 4a6195e00797ddab406fa9b	**Hash:** aa12b09d5a3ed1ff9057c15230526e98014a564 ec65edb7c589b6bc3cf2fc19c	**Hash:** 9d15695f361da512c6dea576bd270eb1328c00 5e8392e9c746ead6f2eea2a00e
Amount: 200.00	Amount: 20.00	Amount: 350.00
Sender: Connery	Sender: Codecademy User	Sender: Alejandro
Recipient: Landlord	Recipient: Codecademy	Recipient: Hiram
Note: Fire Damage	Note: Codecademy Pro	Note: Flight Tickets
Submit Data	Submit Data	Submit Data
Time Stamp: 14/01/2019 à 11:22:51 0.012sec	Time Stamp: 14/01/2019 à 11:22:52 0.548sec	Time Stamp: 14/01/2019 à 11:22:53 0.501sec
Previous Hash: 00000	Previous Hash: efd65faf678481f643de8ae43c6ffcf162e2f a6ea4a6195e00797ddab406fa9b	Previous Hash: aa12b09d5a3ed1ff9057c15230526e9801 4a564ec65edb7c589b6bc3cf2fc19c

Diving Deeper into Blockchain

Gathering Blockchain Transactions

The magic of blockchain is that it's a secure digital ledger that records transactions in chronological order. In this exercise, we'll explore how blockchain transactions are handled.

As transactions are carried out, they get placed in a special location called the *mempool* that collects all these unvalidated transactions. The latest transactions in the mempool are broadcasted to all blockchain participants.

Each participant collects these transactions into a new block. However, each block can only hold a limited number of transactions. Therefore, not all transactions can be added to a block at once.

Once a block is full, the next set of transactions will have to wait in the memory pool. At this point, the block is said to be unconfirmed, and the transactions inside the block are said to be invalidated.

Next, we'll explore how blocks are added to the blockchain!

Key Terms:

- **Transactions:** An exchange of value among participants on the blockchain network.
- **Participants:** Individuals accessing the blockchain network through computers to exchange value.
- **Unconfirmed:** Blocks and transactions that are yet to be verified.

Adding More Blocks

The first step in adding blocks is verifying transactions. This means making sure that transactions haven't been swapped or duplicated. For simplicity, we will assume that all participants know how to verify transactions and that they will verify them honestly. As you progress through this lesson, you will explore how verification really works in later exercises.

The next step is establishing a consensus in the network. In other words, the entire network needs to agree to the transactions.

Assuming everyone honestly verified the transactions, a random participant broadcasts their block to the entire network. If more than 51% of the participants agree with the block, a consensus has been reached, and the block is now said to be confirmed!

However, the network might not agree on the first try. This would happen if someone tried to share an invalid block. The network would reject the attempt at introducing a fake transaction!

The key takeaway from this exercise is that — as long as the majority of participants verify transactions honestly, the blockchain remains secure.

Key Terms:

- Consensus: The process of agreeing to the transactions on the blockchain network.

How Hashing Maintains the Blockchain's Integrity

In the previous lesson, we briefly touched upon the idea of hashing — generating a random string of characters from a given input. Let's go a step further and explore why hashing is so fundamental to the design of the Blockchain.

In a blockchain, each block has a unique hash and a link to the previous block's hash. If a participant decides to tamper with a block by modifying the transactions, the block's unique hash gets changed. However, the following block does not update to reflect this change — it still points to the previous block's hash. Thus, there is a mismatch between hashes and the link between blocks is broken. This results in an invalid copy of the blockchain.

In this way, the records in the blockchain cannot be altered. In other words, the records are said to be immutable.

But what if someone tampers with a block and recalculates the hashes for every single block? Does hashing still guarantee that the blockchain is fully secure? Let's find out in the next exercise.

Key Terms:

- Hashing: Generating a random string of characters from a given input.
- Immutable: Something whose records can't be changed.

Is Hashing Enough to Secure the Blockchain?

We ended the last exercise on a cryptic note — what if an attacker tampers with a block and then somehow covers their tracks by recalculating the hash of each subsequent block to make the blockchain valid once again? Let's explore this concept through an example.

Let's say we have three blocks: A, B, and C with hashes X123, Y456, and Z789 that represent the state of each block. If an attacker tampers with Block A, its contents get changed, so its hash gets changed — let's say the hash is changed from X123 to 123X. Block B no longer points to Block A because the previous hash X123 no longer matches with the new hash 123X. The only way for the attacker to make the chain valid is by fixing this mismatch. For Block B to point to Block A, its previous hash needs to be changed from X123 to 123X.

However, this also counts as tampering with Block B's data. Thus, its hash also gets changed. If the attacker repeats this process for all subsequent blocks, they will have succeeded in creating a valid copy of the blockchain!

Key Terms:

- Recalculating Hashes: Replacing the incorrect hash with a "correct" one to validate the chain.

Securing the Blockchain Further

Believe it or not, the security measures introduced in the previous exercises are not enough to secure the entirety of the blockchain. There needs to be another layer of security to protect the blockchain from outside interference. Allowing anyone to tamper with their copy of the blockchain and trick everyone on the network to update their copies is a big problem.

An additional requirement needs to be introduced that makes it infeasible for someone to tamper with subsequent blocks and take over the blockchain.

Just like how the bank has an accountant to verify transactions manually, the blockchain has a clever technique called Proof-of-Work that accomplishes two important goals:

- It makes it difficult for participants to modify blocks by re-calculating hashes.
- It relies on bulletproof cryptography instead of anonymous participants to verify transactions.

This essentially creates a trustless system and is the main reason why the blockchain is so secure and powerful. Let’s see how Proof-of-Work actually works in the next exercise.

Key Terms:

- Proof-of-Work: A security feature in blockchain to prevent attackers from easily taking over the blockchain.
- Trustless: A feature of blockchain that states how the system doesn't rely on any participant to verify transactions.

Proof-of-Work

Since participants on the blockchain network are anonymous users on their computers, we can’t trust them to verify transactions honestly. Proof-of-Work does nothing more than introduce an additional security constraint to verify transactions. This constraint takes the form of a computationally difficult math problem, which means to say that it takes a lot of time even for the computer to solve the problem.

Instead of randomly being chosen to broadcast their unconfirmed block, a special group of participants, also known as miners, now need to solve a problem in order to be eligible to broadcast their block. The problem, also known as Proof-of-Work, takes the form of a guessing game that involves the use of hashing.

The hash function that’s most commonly used to create the hash for the block is the SHA-256. Miners first guess a nonce value, which is then combined with the contents of the block (i.e transactions, timestamp, hash, and previous hash). They repeat this process until the desired hash is generated.

The first miner to produce a proof broadcasts their unconfirmed block together with the correct nonce value. The rest of the network then verifies the calculation. If the majority of the participants agree, the Proof-of-Work for the block is now complete and the block has now been confirmed! The network then moves on to work on the next block.

Here’s an example of a simple problem — find a number which, when combined with the unconfirmed block’s contents, produces a hash whose first four digits equals 0000. Every participant uses their computer and a hash function (typically SHA-256) to find a number that generates a correct hash. Since this a random guessing game, everyone usually starts out with 0 and increases their guesses until they produce an acceptable hash.

Key Terms:

- Miners: Special participants who calculate the Proof-of-Work to mine new blocks.
- Nonce: A number to be guessed by miners which when combined with the block produces an acceptable hash.

Diving Deeper into Proof-of-Work

The blockchain participants always consider the longest chain to be the correct one. If someone is able to create the longest chain of blocks (even if the blocks are fake), the network is forced to accept the new chain.

The reason for this is simple — the blockchain network assumes that the longest chain has the most amount of computational work done in finding the Proof-of-Work for each block. Therefore, it is reasonable for the network to think that the longest chain contains the most proven record of transactions.

If a dishonest participant decides to tamper with a block, they would have to solve the Proof-of-Work for each subsequent block in order to introduce the tampered block into the network. This is computationally infeasible and almost impossible!

Furthermore, while the participant is busy finding the Proof-of-Work for each block, newer blocks are being added to the blockchain at a faster rate. The participant soon finds out that they are playing a losing battle against the entire network.

What is the key takeaway from all this? A block gets increasingly more tamper-proof as newer blocks are added next to it. Proof-of-Work makes it hard to get through the entire blockchain and allow someone to introduce a fake transaction.

Key Terms:

- Longest Chain: The most trusted chain with the largest amount of computational work done in calculating the Proof-of-Work.

Blockchain Transactions Review

You learned about how transactions work in the blockchain and some of the mechanisms that keep a blockchain valid and secure.

Let's review the key terms:

- Transaction: An exchange of value among participants on the blockchain network.
- Participants: Individuals accessing the blockchain network through computers to exchange value.
- Unconfirmed: Blocks and transactions that are yet to be verified.
- Consensus: The process of agreeing to the transactions on the blockchain network.
- Hashing: Generating a random string of characters from a given input.
- Immutable: Something whose records can't be changed.
- Recalculating Hashes: Replacing the incorrect hash with a "correct" one to validate the chain.
- Proof-of-Work: A security feature in blockchain to prevent attackers from easily taking over the blockchain.
- Trustless: A feature of blockchain that states how the system doesn't rely on any participant to verify transactions.
- Longest Chain: The most trusted chain with the largest amount of computational work done in calculating the Proof-of-Work.

BUILD YOUR OWN MINI-BLOCKCHAIN!

Representing Transactions

The blockchain is a new way of storing and moving data securely. The data mostly consists of transactions which include messages exchanged between two parties. Before we start creating our blockchain, let's think of a way to store a transaction like the one shown below:

```
amount: 30
sender: Alice
receiver: Bob
```

In this example, Alice is trying to transfer 30 units of some currency to Bob. This transaction is best represented in the form of a Python dictionary, with keys for the required fields and values specific to the transaction. These transactions are all stored inside the mempool, a pool of transactions that miners reference when selecting the set of transactions they want to verify.

Script.py

```
/#transaction 1, 2,......
my_transaction = {
  'amount': '5',
  'sender': 'a',
  'receiver': 'b',
}
mempool = [transaction1, transaction2, transaction3, transaction4, 
transaction5, transaction6, my_transaction]

block_transactions = [transaction1, transaction2, transaction3] #This will 
allow us to prepare the transactions to be added to our future Block 
structure.
```

Creating Blocks

We could create a bigger dictionary and store our data inside this dictionary. But since blocks can be represented as objects, let's create a Block Class which we can easily use to create new blocks.

Recall that a Block contains the following properties:

- Timestamp
- Transaction
- Hash
- Previous Hash
- Nonce

Script.py

```
# import datetime library
from datetime import datetime
# print current date and time
print(datetime.now())

class Block:
# default constructor for block class
  def __init__(self, transactions, previous_hash, nonce = 0):
    self.transactions = transactions
    self.previous_hash = previous_hash
    self.nonce = nonce
    self.timestamp = datetime.now()
```

Hashing and SHA-256

Script.py

```
# import sha256
from hashlib import sha256

# text to hash
text = "I am excited to learn about blockchain!"
hash_result = sha256(text.encode())

# print result
print(hash_result.hexdigest())
```

Script.py output

```
32ad45b332a7e5869d6d5aac178a1af413b04b206047709ea021df8d4d21ff56
```

Generating Block Hashes

Block hashes are used to uniquely identify and maintain the integrity of each block. The SHA-256 algorithm is used to generate the hash of the block using the timestamp, data, and previous hash — the three properties of our Block class!

Script.py

```python
from datetime import datetime
from hashlib import sha256

class Block:
  def __init__(self, transactions, previous_hash, nonce = 0):
    self.timestamp = datetime.now()
    self.transactions = transactions
    self.previous_hash = previous_hash
    self.nonce = nonce
    self.hash = self.generate_hash()

  def print_block(self):
    # prints block contents
    print("timestamp:", self.timestamp)
    print("transactions:", self.transactions)
    print("current hash:", self.generate_hash())

  def generate_hash(self):
    # hash the blocks contents
    block_contents = str(self.timestamp) + str(self.transactions) +
str(self.previous_hash) + str(self.nonce)
    block_hash = sha256(block_contents.encode())
    return block_hash.hexdigest()
```

Creating the Blockchain Class

Each computer participant has their own copy of the blockchain. Ideally, each copy of the blockchain should have the same properties and functionality to add and validate blocks.

We can represent the blockchain as an object. We are using objects for our implementation, because they offer the flexibility to create specific attributes and methods. Representing it as an object also allows us to create blockchain instances for each computer participant.

To review, our blockchain contains the following:

- Chain: A list that that holds all the blocks inside the blockchain.
- Unverified Transactions: A list that represents all the unverified transactions before being passed into a block.
- Genesis Block: A block automatically generated when the blockchain is initialized.

Script.py
```
#imports the Block class from block.py
from block import Block

class Blockchain:
    def __init__(self):
      self.chain = []
      self.all_transactions = []
      self.genesis_block()

    def genesis_block(self):
      transactions = []
      previous_hash = "0"
      self.chain.append(Block(transactions, previous_hash))
```

Adding Blocks to the Blockchain

Script.py
```
  # Use the script of "creating the block chain"
  # prints contents of blockchain
  def print_blocks(self):
    for i in range(len(self.chain)):
      current_block = self.chain[i]
      print("Block {} {}".format(i, current_block))
      current_block.print_contents()

  # add block to blockchain `chain`
  def add_block(self, transactions):
    previous_block_hash = self.chain[len(self.chain)-1].hash
    new_block = Block(transactions, previous_block_hash)
    self.chain.append(new_block)
```

Checking for a Broken Chain

Hashing helps to maintain the integrity of the blockchain. In this exercise, we will see this in action. Let's write a .validate_chain()method that checks to see if blocks are linked to each other properly.

In order to validate the entire blockchain, we must loop through each of the blocks stored inside the blockchain itself. Then, we will check through each of the blocks to ensure that the previous hash value matches with the hash value inside our previous block.

Script.py
```
  def validate_chain(self):
    for i in range(1, len(self.chain)):
      current = self.chain[i]
      previous = self.chain[i-1]
      if(current.hash != current.generate_hash()):
        print("The current hash of the block does not equal the generated
hash of the block.")
        return False
      if(current.previous_hash != previous.generate_hash()):
        print("The previous block's hash does not equal the previous hash
value stored in the current block.")
        return False
    return True
```

Hacking the Chain

Now we can use the code in our previous exercise to spot a broken link. Let's try tampering with the contents of the block and see how that creates a mismatch between hash values.

Script.py

```
from blockchain import Blockchain

new_transactions = [{'amount': '30', 'sender':'alice', 'receiver':'bob'},
                {'amount': '55', 'sender':'bob', 'receiver':'alice'}]


my_blockchain = Blockchain()
my_blockchain.add_block(new_transactions)
my_blockchain.print_blocks()
my_blockchain.chain[1].transactions = 'fake_transactions'
my_blockchain.validate_chain()
```

Script.py Output

```
Block 0 <block.Block object at 0x7fe87fd5b7f0>

timestamp: 2019-01-16 12:42:43.977028

transactions: {}

current hash: 4e568cf0ca95a414664db049a0f7fcf6bfe613eb104664a61cef7a4489d1c7da

previous hash: 0

Block 1 <block.Block object at 0x7fe87fd5b828>

timestamp: 2019-01-16 12:42:43.977057

transactions: [{'sender': 'alice', 'receiver': 'bob', 'amount': '30'}, {'sender': 'bob', 'receiver': 'alice',
'amount': '55'}]

current hash: dc372483510e95411e2a8cafee49e99f6eb3119d86302440dc0551c60690fbea

previous hash: 4e568cf0ca95a414664db049a0f7fcf6bfe613eb104664a61cef7a4489d1c7da

The current hash of the block does not equal the generated hash of the block.
```