Blockchain Tutorial

Introduction

Confused by terms such as distributed ledgers, hashing, and cryptography? You are not alone. Our Blockchain tutorial cuts through the technical jargon to the core of what makes this technology revolutionary and secure. We address the typical pain points of a beginner by breaking down complex concepts into simple, easy-to-understand modules.

Ready to unlock the future of decentralized systems? Click here to view the full **Blockchain Technology Course Syllabus**!

Why Students or Freshers Learn Blockchain

Learning Blockchain technology can provide a modern launch to high-growth careers, which is revolutionizing industries much further from finance itself.

- **Major Industry Disruption:** Blockchain powers DeFi, supply chain tracking, and digital identity, among other things, making experts incredibly in demand.

- **High Demand & Pay:** Skilled Blockchain Developers and architects are in very short supply, hence commanding premium salaries even for fresh talent.
- **Future-Proof your Skill Set:** Mastery of cryptography, distributed systems, and smart contracts make you valuable in any sector using Web3 technologies.
- **Entrepreneurial Potential:** The development of innovative, decentralized applications (dApps) and new business models are enabled.

Ready for a Blockchain job? Click here for the best **Blockchain Interview Questions and Answers**!

Step-by-Step Blockchain Tutorial for Beginners

This blockchain tutorial for beginners will take you through the basics of Blockchain and provide practical steps to set up a basic, localised development environment. We'll simulate the core mechanism of a simple blockchain.

Part 1. Understanding the Core Concept: What is a Blockchain?

A blockchain is a decentralized, distributed, and sometimes public digital ledger consisting of records, known as blocks, utilized for recording transactions over many computers in such a way that any block involved cannot be retroactively altered without the alteration of all subsequent blocks.

The three core components are:

- **Block:** A container for data – transactions, a timestamp, and a cryptographic hash of the previous block.

- **Chain:** The sequence of blocks linked together. The linking is done via the previous block's hash contained in the current block.
- **Decentralization:** It is the shared and synchronized ledger across the nodes in the network, without the presence of a central authority.

Part 2. Installation and Setup (Python Focus)

For this tutorial, we'll use Python, which is excellent for showing core logic in a straightforward and simple manner.

Step 1. Python Installation

Download Python: If you don't have it, install the latest available Python version (3.9+) from the official Python website.

Verify Installation: Open your terminal or command prompt and type:

python –version

You should see the installed version number.

Step 2. Installing Required Libraries

We need two simple libraries:

- **hashlib:** Standard Python library for cryptographic hashing. Comes included.
- **json:** Standard Python library to operate on JSON data – already included.

No installation of any external library for this simple implementation!

Part 3. Designing the Blockchain Architecture

We will define two main classes: Block and Blockchain.

Step 1. The Block class

The Block class represents the data container. Each block needs to store its index, a timestamp, a list of transactions, its hash, and the hash of the previous block.

```
import hashlib

import json

import time

class Block:

    def __init__(self, index, transactions, timestamp, previous_hash):

        self.index = index

        self.transactions = transactions

        self.timestamp = timestamp

        self.previous_hash = previous_hash

        self.hash = self.calculate_hash() # The block's own unique identifier

    def calculate_hash(self):

        # We need a string representation of the block's contents

        block_string = json.dumps(self.__dict__, sort_keys=True)

        # Hash the string using SHA-256

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

# Example of a Block creation:
```

 # first_block = Block(1, [“Alice sent 5 BTC to Bob”], time.time(), “0”)

 # print(first_block.hash)

- **self.index:** The position in the chain.
- **self.previous_hash:** The critical link to the chain.
- **self.calculate_hash():** Utilizes SHA-256 to create a unique fingerprint depending on ALL the block data. If even one transaction changes, then the hash changes completely.

Step 2. The Blockchain Class

The ***Blockchain class*** manages the chain, handles new transactions, and creates new blocks.

```
class Blockchain:

    def __init__(self):

        self.chain = []

        self.current_transactions = []

        # Create the genesis block (the first block in the chain)

        self.create_new_block(proof=100, previous_hash='1')

    def create_new_block(self, proof, previous_hash=None):

        """

        Creates a new Block and adds it to the chain

        """
```

```
        block = Block(
            index=len(self.chain) + 1,
            transactions=self.current_transactions,
            timestamp=time.time(),
            # Use the hash of the last block in the chain, or the provided hash for the genesis block
            previous_hash=previous_hash or self.chain[-1].hash,
        )
        # Reset the current list of transactions
        self.current_transactions = []
        self.chain.append(block)
        return block
    def new_transaction(self, sender, recipient, amount):
        """
        Adds a new transaction to the list of current transactions
        """
        self.current_transactions.append({
            'sender': sender,
```

```
            'recipient': recipient,
            'amount': amount,
        })
        # Returns the index of the Block that will hold this transaction
        return self.last_block.index + 1
    @property
    def last_block(self):
        # Returns the last Block in the chain
        return self.chain[-1]
    # — MINING SIMULATION (Proof of Work) —
    def proof_of_work(self, last_proof):
        """
        Simple Proof of Work Algorithm:
         – Find a number 'proof' such that hash(last_proof, proof) contains 4 leading zeroes.
        """
        proof = 0
        while self.valid_proof(last_proof, proof) is False:
```

```
        proof += 1
    return proof
@staticmethod
def valid_proof(last_proof, proof):
    """
    Validates the Proof: Does hash(last_proof, proof) contain 4 leading zeroes?
    """
    guess = f'{last_proof}{proof}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:4] == "0000"
```

Part 4. Testing the Blockchain: Mining and Transactions

Now let's use the classes to simulate the creation of a simple, working blockchain ledger.

Step 1. Initialization

We start by instantiating the Blockchain class; this automatically creates the Genesis Block.

```
# 1. Initialize the Blockchain
my_blockchain = Blockchain()
print("— Genesis Block Created —")
```

print(f”Index: {my_blockchain.chain[0].index}”)

print(f”Hash: {my_blockchain.chain[0].hash}”)

Step 2. Adding Transactions

Transactions are collected before a new block gets “mined.”

 # 2. Add some transactions

my_blockchain.new_transaction(sender=”Alice”, recipient=”Bob”, amount=50)

my_blockchain.new_transaction(sender=”Charlie”, recipient=”David”, amount=15)

print(“\n— Transactions Added to Pending List —“)

print(my_blockchain.current_transactions)

Step 3. Mining a New Block (Proof of Work)

Mining is a process to create a new block by solving a computational puzzle, namely Proof of Work, or simply PoW.

- **Puzzle Time:** Find the proof number.
- **Create the Block:** Add the transactions to the new block, link it to the previous block using its hash, then seal it with the found proof.

 # 3. Mine the Block (Solve the Proof of Work puzzle)

last_block = my_blockchain.last_block

last_proof = last_block.proof # Simplified, using a proof placeholder

proof = my_blockchain.proof_of_work(last_proof)

```
# 4. Forge the new Block, adding a transaction for the 'miner' (us)
my_blockchain.new_transaction(
    sender="0",  # Represents a network reward (mining reward)
    recipient="Miner-Node",
    amount=1, # The mining reward
)
# 5. Add the Block to the chain
previous_hash = my_blockchain.last_block.hash
new_block = my_blockchain.create_new_block(proof, previous_hash)
print("\n— New Block Mined and Added —")
print(f"New Block Index: {new_block.index}")
print(f"New Block Hash: {new_block.hash}")
print(f"Previous Hash: {new_block.previous_hash}")
```

Step 4. Mining a Second Block

Great, let's quickly add another block to see the chain grow.

```
# 6. Add more transactions for the next block
my_blockchain.new_transaction(sender="Bob", recipient="Eve", amount=5)
# 7. Mine again
```

last_block_2 = my_blockchain.last_block

last_proof_2 = last_block_2.proof # Simplified

proof_2 = my_blockchain.proof_of_work(last_proof_2)

 # 8. Forge and add the second block

my_blockchain.new_transaction(sender="0", recipient="Miner-Node-2", amount=1)

previous_hash_2 = my_blockchain.last_block.hash

new_block_2 = my_blockchain.create_new_block(proof_2, previous_hash_2)

print("\n— Second Block Mined and Added —")

print(f"Second Block Index: {new_block_2.index}")

print(f"Previous Hash for Block 2 (Must match Block 1's Hash): {new_block_2.previous_hash}")

Part 5. Security: Validating the Chain

An important feature about a blockchain is the immutability. Any attempt to alter an older block will cause the chain validation process to fail instantaneously.

Step 1. The Validation Function

We need a function that goes through the chain and checks for two things about each block – except the Genesis:

- Is the *previous_hash* in the current block correct, meaning is it the hash of the preceding block?

- Does the hash of the current block in fact match its contents (i.e., is the hash stored in the block correct)?

```
def is_chain_valid(self):
    current_block = self.chain[0]
    i = 1
    while i < len(self.chain):
        previous_block = current_block
        current_block = self.chain[i]
        # 1. Check that the block's hash is correct
        if current_block.hash != current_block.calculate_hash():
            print("Chain Invalid: Current block's hash is not correct.")
            return False
        # 2. Check that the previous_hash link is correct
        if current_block.previous_hash != previous_block.hash:
            print("Chain Invalid: Block linkage is broken.")
            return False
        # 3. Check Proof of Work (Simplified check for this tutorial)
        # if not self.valid_proof(previous_block.proof, current_block.proof):
```

```
        #     print("Chain Invalid: Proof of Work is incorrect.")
        #     return False
        i += 1
    return True
```

Note: You would typically add this method inside your Blockchain class.

Step 2. Testing Immutability

Check the Valid Chain:

```
print("\n— Final Chain Validation Status —")
if my_blockchain.is_chain_valid():
    print("Blockchain is VALID!")
else:
    print("Blockchain is INVALID!")
```

Attempt Tampering: Now, let's check manually the transaction in the first mined block.

```
# WARNING: This simulates an attack by manually modifying data!
print("\n— Simulating Tampering on Block 1 —")
my_blockchain.chain[1].transactions = [{"sender": "HACKED", "recipient": "Bob", "amount": 99999}]
print(f"Transaction in Block 1 was changed (not the stored hash, just the data).")
```

```
# 3. Check the chain again

if my_blockchain.is_chain_valid():

    print("Blockchain is VALID!")

else:

    print("Blockchain is INVALID! The change was detected.")
```

The validation will fail because the changed transaction data will cause my_blockchain.chain[1].calculate_hash() to produce a new hash, which no longer matches the original hash stored in my_blockchain.chain[1].hash. The chain is broken.

Part 6. What's Next? (Decentralization)

This basic tutorial only runs on your local machine. Real blockchain adds the element of Decentralization by doing the following:

- **P2P Network:** The code should be run as a server able to speak to other nodes.
- **Consensus:** Nodes must agree on the one true version of the chain. For example, Proof of Work, Proof of Stake. When two nodes mine a block at the same time, the longest chain is adopted by the network.
- **Client Application:** A simple UI or wallet that sends transactions over the network.

This structure gives the necessary logic to create the immutability and cryptographic linking that underlies every major blockchain, including but not limited to Bitcoin and Ethereum.

Are you ready to test your knowledge with real-world scenarios and overcome common development hurdles? Click here for advanced **Blockchain Challenges and Solutions**!

Real Time Examples for Blockchain Tutorial for Learners

Here are some real-time examples that demonstrate the practical usages of Blockchain:

Supply Chain Management & Provenance Tracking – e.g., Food & Retail

Problem: Consumers do not trust the origin, journey, and ethical sourcing of products. Examples include organic food and diamonds. Paper trails are slow andfraught with fraud.

- **Blockchain's role** is to record every step a product takes-from farm to processing plant to store shelf-as a transaction on a decentralized, immutable ledger. Thus, this creates a clear, permanent record.
- **Result:** It enables firms like Walmart to trace the source of contaminated food in seconds, instead of weeks, thereby improving public safety and efficiency.

Decentralized Finance (DeFi) and Lending

Problem: Traditional finance requires intermediaries, namely banks, which entails high fees and slow processing, excluding those without access to banking infrastructure.

- **The Role of Blockchain:** Blockchain-based systems, such as Ethereum, employ Smart Contracts that automatically allow lending, borrowing, and trading to take place without the need for a central authority.
- **Result:** Users can lock up crypto assets and automatically receive a loan or earn interest. Financial services will be faster, cheaper, and available anywhere globally, 24/7.

Non-fungible tokens and digital ownership

Problem: It was once very hard to establish verifiable, unique ownership of digital assets-art, music, in-game items-because digital files are easily copied.

- **Blockchain's Role:** The NFT is an exclusively created digital token recorded on a public blockchain, such as Ethereum or Solana. This token provides validation of the authenticity and ownership of a digital or physical item.
- **Result:** This enables the artist to sell provably unique digital works and ensure royalties on future sales, and even create new digital economies.

Ready to build your own decentralized application? Check out our list of **Blockchain Project Ideas** to begin coding and building your portfolio!

FAQs About Blockchain Tutorial for Beginners

1.What are the 4 types of Blockchain?

The four main types of blockchain networks are: Public (open to all, like Bitcoin), Private (controlled by a single entity, often for internal use), Consortium (governed by a group of organizations), and Hybrid (combining features of both public and private chains).

2.How much 1 dollar in blockchain?

$1 is a unit of fiat currency, while "blockchain" is a technology. There's no direct conversion. However, stablecoins like USDC or USDT are built on a blockchain and aim to maintain a value of $1 per token, representing a digital dollar.

3.Can I learn blockchain on my own?

Yes, you absolutely can learn blockchain on your own. There are vast open-source resources, documentation, **online blockchain courses**, and active developer communities. Practical learning involves studying Solidity/Rust, practicing with testnets, and building simple decentralized applications (dApps).

4.Is UPI a blockchain?

No, UPI (Unified Payments Interface) is not a blockchain. It is a centralized, real-time payment system developed by the National Payments Corporation of India (NPCI). It facilitates instant bank-to-bank transfers but does not use a distributed, decentralized ledger for settlement.

5.Who owns 70% Bitcoin?

No single entity owns 70% of Bitcoin. The biggest holder is thought to be the anonymous founder, Satoshi Nakamoto, who controls ~1.1 million BTC, or about 5-6% of the total supply, which has never been spent.

6.What is layer 0 in blockchain?

Layer 0 is the fundamental level of infrastructure on which the entire blockchain ecosystem has been developed.4 This layer provides the hardware, networking protocols, and connectivity that nodes need to communicate in order for Layer 1 blockchains to be built and function, such as Bitcoin or Ethereum.

7.What is Blockchain salary?

The demand for a Blockchain Developer is strong, so salaries tend to be pretty good. In India, **Blockchain developer salary for freshers** often start at ₹5-8 LPA, with mid-level professionals earning between ₹10-18 LPA.7 Senior architects can command significantly higher than that, often crossing ₹25 LPA.

8.Is Blockchain a good career?

Yes, Blockchain is considered to be an excellent career.8 It is one of those technologies that are still in the incipient stage of adoption and have disrupted finance, supply chain, and data management functions. This, in turn, creates high demand and premium salaries for professionals skilled in development, security, and consulting.

9.Is Blockchain real money?

Blockchain is not real money; it is the underlying technology or ledger. However, Cryptocurrencies-e.g., Bitcoin or Ethereum-that run on top of a blockchain can be considered forms of digital money, although they are usually not considered legal tender by governments.

10.Is Blockchain a coding language?

No, Blockchain isn't a coding language. It is a distributed ledger technology with a framework.10 It uses existing programming languages, for instance, Python, Java, Go, or Solidity-a language designed to write Smart Contracts running on Ethereum-to build on and interact with the chain.

Conclusion

You've made it to the end of your introduction to Blockchain, having learned its core principles of decentralization, immutability through cryptographic hashing, and consensus. You now have the foundational knowledge that underpins cryptocurrencies, NFTs, and the future of finance and data management. This skill set is your on-ramp into the high-demand Web3 world. Ready to build the future? Enroll in our comprehensive **Blockchain Development course in Chennai** today to master Smart Contracts and become a certified developer!