Complete Guide to Ethereum Blockchain with Python

BEGINNER BLOCKCHAIN GUIDE PYTHON WEB 3.0

Introduction

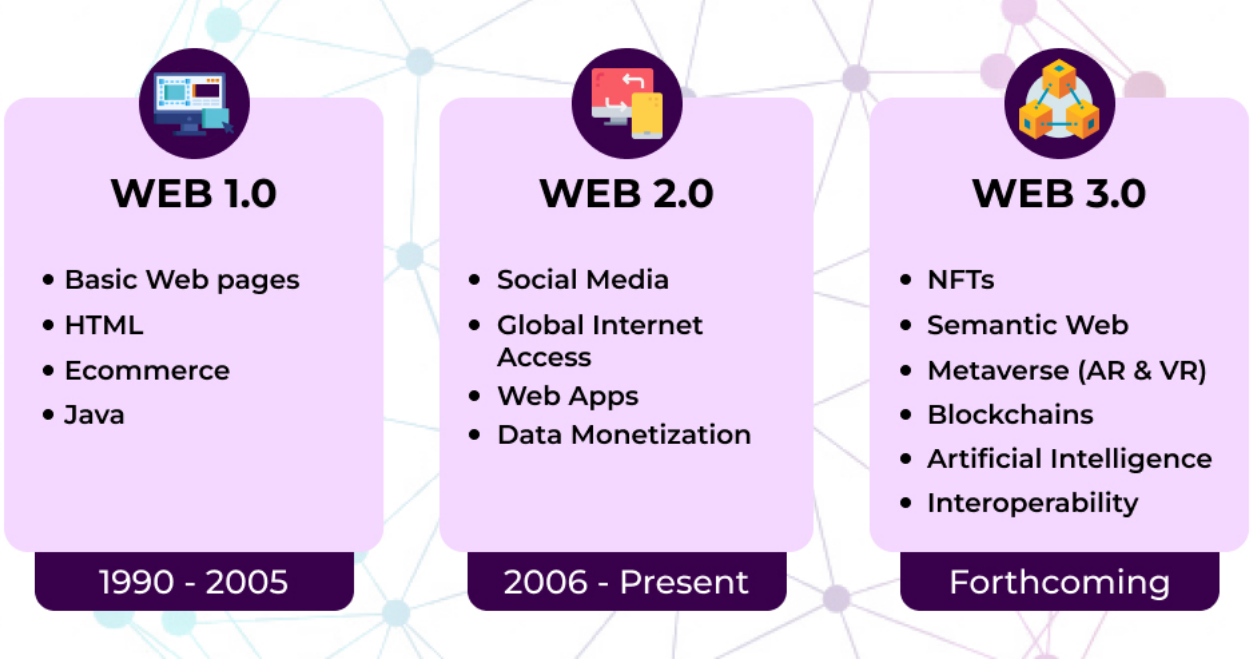
We have internet access in every corner of the country. Almost all businesses around us, like manufacturing, retail, consultancy, etc., are trying to go digital and display themselves on the web to scale and increase sales. If you want to know anything about your surroundings, then the internet is the first choice we prefer. People's experience over the web transforms daily, and demand constantly expands over time. With the expansion of the web, the new buzzword WEB 3.0 is revolving everywhere, which will significantly impact society and make the internet more feasible and easy to use. If you are new to this word, then no need to worry because, in this article, we will deep dive into the world of the decentralized web with the Ethereum blockchain.

This article was published as a part of the Data Science Blogathon.

Table of Contents

What is Web 3.0?

Web 3. O is the next step in the evolution of the internet, allowing it to process data with near-human intelligence through the use of artificial intelligence and blockchain technologies. With the growth in technology, businesses are emerging and allowing management from a centralized to a decentralized ecosystem. In simple words, web 2.0 is a centralized management system emerging towards decentralized to allow everyone to play an equal role with great privacy and security, referred to as web 3.0. Web 3.0 allows fair usage and open web for everyone.



Source – Intelegain

Web 3.0 enables the peer-peer architecture where each peer maintains a copy of the same data on the network, and the entire network is also kept in sync, known as nodes. So the transaction is highly secure over web 3.0, providing a more personalized and customized surfing experience over the internet.

Blockchain Terminologies

Web 3.0 defines some of the most used concepts and terminologies to know before diving into the development part of a smart contract with python.

1. **Blockchain –** Blockchain is a decentralized, peer-peer architecture that stores transactional records in the form of blocks which get replicated on several databases from a chain in a connected network of peer-to-peer nodes. Hence blockchain is also defined as an immutable ledger which is entirely transparent.
2. **Block –** Group of interconnected nodes defined as a block. Every chain consists of multiple blocks in the blockchain, and a block contains 3 elements named data, nonce, and hash.
3. **Node –** It stores the exact data so that the state of the blockchain can be easily queried.
4. **Smart Contract –** A digital agreement (program)runs on an Ethereum blockchain.

Getting Started with Web3.py and the Ethereum Blockchain

Ethereum blockchain is a decentralized application empowering multiple businesses. There are many ways we can connect with the Ethereum blockchain using different clients. In this post, we will be using the WEB3.py python library.

To begin with development, first, we need to set up our system. If you do not have Python version 3.6 or above, please check out this link to update or install python.

Connecting to Ethereum Nodes

First is to choose and connect with the Ethereum node using web3 and then run the basic commands. There are two types of nodes which is a local and hosted nodes. We will use Infura, a hosted version.

There are many other hosted versions available. The 3 basic ways to connect with Ethereum nodes are HTTP, WebSockets, and IPC.

Setting up Infura

Let's head over to Infura and create an account to get a link to the Ethereum node. And that's what we will use to create our connection to the blockchain. Infura is Ethereum as a node service that gives us network access. If you sign-up and verify with your email, you will get a dashboard where you have to create a new API key with WEB3, give it a name, and click create. You will receive an API key and keep it handy.

Source: Infura

Connect to the Ethereum Test Node

Now we can start writing python code and interact with the blockchain through web3.py. To get started with web3, you need to install the web3.py library in a working directory using the PIP package with the below command.

```
pip install web3
```

Web3 library depends on a connection to an Ethereum node, so we have to connect to an Ethereum node with web3 python. Node is just a computer part of the blockchain network. We can use a connection provider to connect with the blockchain. There are many providers like an Ethereum tester provider, HTTP provider, and a local provider so let's start with connecting to an Ethereum tester provider. For this, you

have to create a new python test file, or you can use Google Colab. You can get help from the official documentation if the code does not run.

```
from web3 import Web3, EthereumTesterProvider w3 = Web3(EthereumTesterProvider()) w3.isConnected()
```

How to connect to Ethereum with HTTP Provider?

We have set up an Infura and created a new project with a Web3 API endpoint where you need to copy the Ethereum endpoint with Mainnet, our network connection provider. And copy the below code, which indicates creating the HTTP provider object with the endpoint as an attribute, and test the connection. In the below code, you can copy the provider link and add your API key and run the code.

```
provider_url = 'https://mainnet.infura.io/v3/' w3 = Web3(Web3.HTTPProvider(provider_url)) w3.isConnected()
```

Interact with Ethereum Blockchain with Python

We can connect with Ethereum Mainnet through an HTTP provider, so now we can use our web3 instance to get information about the latest block, such as block number, nonce, gas limit, size, timestamp, hash, and transactions.

```
latest_block = w3.eth.get_block('latest') print(latest_block)
```

Next, learn **how to check an Ethereum address** and check whether the address is valid or not. In the below code, you can pass the user's address, wallet, or smart contract and display the results.

```
print(w3.isAddress('0xDAFEA492D9c6733ae3d56b7Ed1ADB60692c98Bc5'))
```

If you want to **get the balance of the wallet,** then use the checksum address function. First, you need to pick the wallet from the address and then check the balance.

```
#check the total number of ether wallet = w3.toChecksumAddress('0xDAFEA492D9c6733ae3d56b7Ed1ADB60692c98Bc5')
print(w3.eth.get_balance(wallet))
```

You can do much more functionality in web3.py, for example, convert WEI to ether. WEI is the smallest denomination of ETH, like a penny to a dollar.

How does Ethereum Blockchain Interact with Smart Contracts using web 3.0 Python?

We will use the smart contract that already exists on the Ethereum blockchain. To find the contract, visit ether scan. It is a famous Ethereum blockchain explorer where you can search contracts or transactions by address, token, or block. Under the blockchain, click on verified contracts and see the multiple smart contracts. Open any contract and scroll down to copy the contract ABI that contains all the information about what the contract contains, like inputs, functions, names, variables, etc.

Source: Learn By Bit

- In Colab, paste the copied ABI; we will also require the contract address.
- Visit the contract, and on top, copy the contract address that starts with 0x.
- Create an instance (object) of the contract with address and ABI as object parameters.
- Call any function from the contract.
- Try the code to run in the Colab and observe the output.

```
abi        =        '[{"inputs":[{"internalType":"contract        ArtData","name":"data_","type":"address"},
{"internalType":"contract              FoundNote","name":"note_","type":"address"},{"internalType":"contract
DailyMint","name":"mint_","type":"address"}],"stateMutability":"nonpayable","type":"constructor"},{"inputs":
[{"internalType":"uint256","name":"coinId","type":"uint256"},
{"internalType":"uint256","name":"start","type":"uint256"},
{"internalType":"uint256","name":"end","type":"uint256"}],"name":"artStats","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":
[{"internalType":"address","name":"addr","type":"address"}],"name":"coinBalances","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":
[],"name":"coinStats","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":
[{"internalType":"uint256","name":"num","type":"uint256"}],"name":"encodeDecimals","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"pure","type":"function"},{"inputs":
[{"internalType":"uint256","name":"tokenId","type":"uint256"}],"name":"ownerOf","outputs":
[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},
{"inputs":[],"name":"tokenCount","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},
{"inputs":[{"internalType":"uint256","name":"tokenId","type":"uint256"}],"name":"tokenData","outputs":
```

```
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":
[{"internalType":"uint256","name":"tokenId","type":"uint256"}],"name":"tokenDataURI","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":
[{"internalType":"uint256","name":"tokenId","type":"uint256"}],"name":"tokenImage","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":
[{"internalType":"uint256","name":"tokenId","type":"uint256"}],"name":"tokenImageURI","outputs":
[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":
[{"internalType":"uint256","name":"coinId","type":"uint256"}],"name":"tokenStats","outputs":
[{"internalType":"bytes","name":"","type":"bytes"}],"stateMutability":"view","type":"function"}]'
contract_address = '0x3008B7ddAc628Ad0Dcbaa0d7CFE622F0F9C3cD10' #create instance contract_instance =
w3.eth.contract(address=contract_address, abi=abi) #call the totalSupply function total_supply =
contract_instance.functions.totalSupply().call() print(total_supply)
```

Build a Hello World Solidity Smart Contract

We will build a smart contract using solidity programming language and run a Python SOLCX library. To create a smart contract, we use Remix IDE. You can create a smart contract directly in a Python file or Google Colab. If you do not know much about solidity and smart contract but want to explore them, please refer to this smart contract article. If creating a new contract in Remix, create a new file with a dot sol extension.

```
//SPDX-License Identifier pragma solidity 0.8.17; contract HelloWorld { string public message; constructor()
{ message = "Hello World"; } function setMessage(string memory _message) public { message = _message; }
function displayMessage() view public returns(string memory) { return message; } }
```

At the top of the file, you must put the SPDX license identifier, include the solidity version, and start creating the contract just like the class creation in c++. We define one string and a constructor to define the value. After that, we added a function to set the message and one function to display the message in the console, which is of type view that means to display the value. We only use remix because It provides an easy compiling and deploying feature where you can check that contract is working fine.

Compile the solidity Smart Contracts with Python

Now you can copy the contract and use Solcx to compile the contract. For this, we need to call the compiled source and the output values we need.

```
from solcx import compile_source import solcx solcx.install_solc()
```

```
compiled_solidity = compile_source( ''' //SPDX-License Identifier pragma solidity 0.8.17; contract HelloWorld
{ string public message; constructor() { message = "Hello World"; } function setMessage(string memory
_message) public { message = _message; } function displayMessage() view public returns(string memory) {
return message; } } ''', output_values = ['abi', 'bin'] )
```

After running the above cell, we can start interacting with the contract or getting information. We can grab the compiled solidity or call the pop item and observe the results.

```
contract_id, contract_interface = compiled_solidity.popitem() print(contract_id) print(contract_interface)
print(contract_interface['abi'])
```

Interact with the Smart Contract through Python

Now we have to build a simple, smart contract, but how can we interact with it to observe the results and track the transactions? So you only need to deploy it to any URLs like Metamask or ganache, where you get the intelligent contract HTTP URL to connect the contract with the python web3 library. And copy the address and web3 transaction ether from remix ide. And after a successful connection, we can call the function from python that you have created in the contract.

```
import json ganache_url = 'http://127.0.0.1:7545' w3 = Web3(Web3.HTTPProvider(ganache_url)) abi = json.loads('[{"inputs":[],"stateMutability":"nonpayable","type":"constructor"},{"inputs":[],"name":"displayMessage","outputs":[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"message","outputs":[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"string","name":"_message","type":"string"}],"name":"setMessage","outputs":[],"stateMutability":"nonpayable","type":"function"}]') address = w3.toChecksumAddress("0xd9145CCE52D386f254917e481eB44e9943F39138") contract = w3.eth.contract(address=address, abi=abi) print(contract.function.displayMessage().call())
```

How to Deploy Smart Contracts with Python?

It is time to learn the smart contract deployment direct from the code. So first, you need to instantiate the web3 HTTP connection as we have done above. After that, we need to define the output resources as ABI value and bytecode to copy the bytecode from the smart contract compilation details. After that, we try to get the contract and generate the transaction hash through which we get the transaction receipt and deploy the contract over Ganache. Now we can call the contract function the same way we compile it in python.

```
ganache_url = 'http://127.0.0.1:7545' w3 = Web3(Web3.HTTPProvider(ganache_url)) bytecode = "608060405234801561001057600080fd5b506004361061004157600035600e01c80632d59dc1214610046578063368b877214610064578063e21f
greeter = web3.eth.contract(abi=abi, bytecode = bytecode) tx_hash = greeter.constructor().transact() tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash) #get the contract. contract = web3.eth.contract(address = tx_receipt.contractAddress, abi=abi) print(contract.functions.displayMessage().call())
```

Build DAPP (Transaction) without Metamask

Metamask is a decentralized Ethereum-based wallet that stores the Ether used to buy, send, swap, or convert crypto tokens. It is available on Mobile and is an extension for almost all browser support, making it easy to use and track transactions. But what if we can do the same marketing and get the complete account information through scripting? We are studying web3 and blockchain, So we can connect our account with an HTTP provider and create a temporary account that will link to our Ethereum account, from where we can get several account details and make a transaction. Below is the code snippet that follows a similar approach to demonstrate creating transactions through a script.

```
provider_url = 'https://mainnet.infura.io/v3/820a7144d1f24094bd8aef3cb79ddda3' web3 = Web3(Web3.HTTPProvider(provider_url)) web3.eth.account.create() account = web3.eth.account.create() print(account) print(account.address) keystore = account.encrypt('foobar') print(keystore)
```

Conclusion

Web 3.0 is the new internet era that provides machines with human intelligence to think and take action. After reading and following the complete article, we learned the practical usage of blockchain technology or Ethereum blockchain and how it creates impactful products that disrupt the internet and allow all to follow. So, in this article, let's go through all the key points we have learned about web 3.0 and Ethereum blockchain.

1. **Decentralization –** Blockchain will help to decentralize the data store while establishing trust in the virtual world because web 3.0 allows information to be retrieved based on the content that can be kept at several locations simultaneously, making it decentralized.
2. **Trustful and Permissionless –** Participants can interact directly with the network without needing an intermediary or permission from a governing body. Due to this, we can access all relevant data of our choice with great trust.
3. **Artificial Intelligence and Machine Learning –** In web 3.0, computers can understand information in the same way people do through technology based on semantic web ideas and natural language processing.
4. **Connectivity and Ubiquity –** In web 3.0, the internet will be accessible to everyone anywhere and at all times because IOT will be launched on several new smart devices. These internet-connected devices will no longer be limited to pcs and smartphones because they support Web 2.0.

Source – 101 blockchain.com

Understanding the shortcomings of web 2.0, corporates like Apple, Amazon, Google, etc., are transforming their existing services into internet 3.0 apps that abide by the above 4 principles. Siri and wolfram alpha are two applications that use web 3.0 features. After reading the complete article, I hope you are convinced that web 3.0 will make a huge difference in society.

So, please answer one question in the comment section below: How is web 3.0 related to blockchain? The following choices can be the answer that you have to select.

- It helps create smart contracts for web pages on the internet.
- It helps web 3.0 store its data in the blockchain.
- It offers a decentralized experience to its users
- All of the above mentioned.

Please take a minute to answer the quiz in the comment section below. I hope it was easy to cope with each step discussed in the article. You can post queries in the comment section below or connect with me. Connect with me on Linkedin and check out my other articles on Analytics Vidhya

---

Article Url - https://www.analyticsvidhya.com/blog/2023/01/complete-guide-to-web3-0-ethereum-blockchain-using-python/

Raghav Agrawal

I am a final year undergraduate who loves to learn and write about technology. I am a passionate learner, and a data science enthusiast. I am learning and working in data science field from past 2 years, and aspire to grow as Big data architect.