LearnWeb3 bootcamp – Freshman – Learn basics of web3

Table of Contents

Intro to programing

There are a lot of topics that are certainly covered within web2 that are highly useful in Web3. In fact, I would say that Web3 is an extension of Web2 when it comes to the technology being used.

For example, if you make a smart contract, you still need to provide your users a way to interact with that smart contract. You can do this by building a website or application that allows the user to interface with the smart contract in an easy manner.

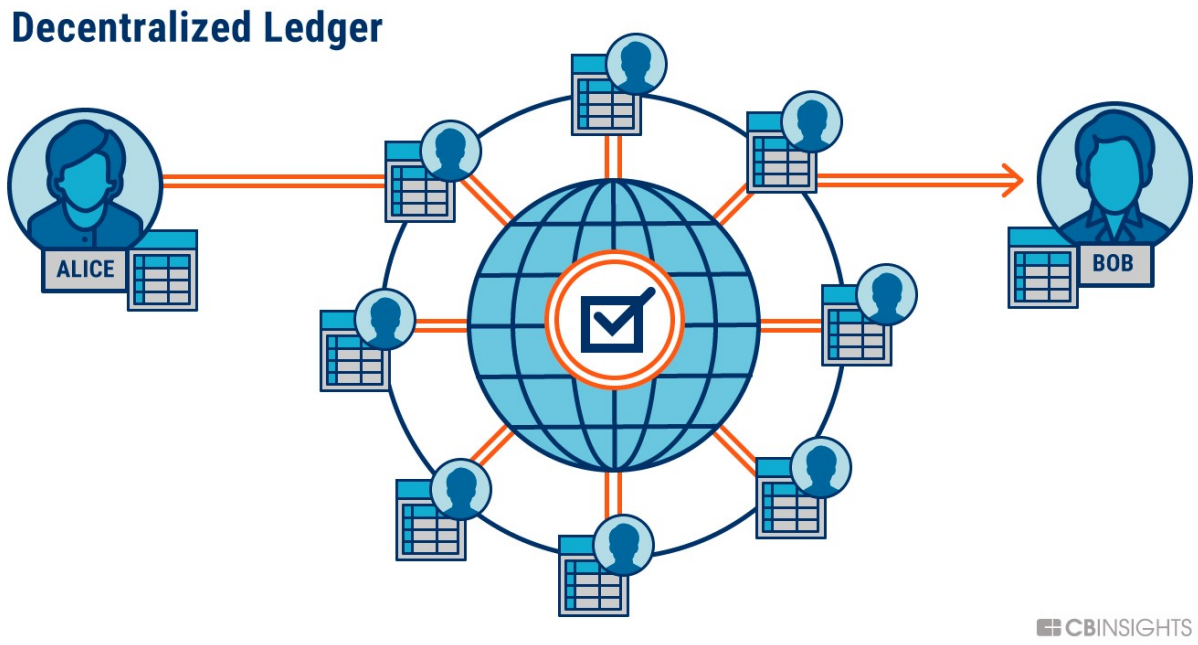
Frontend Technologies

Frontend is the interface with which an user interacts. On the web, the frontend refers to a website you can browse around. Mobile apps, and desktop apps, are also examples of valid frontend interfaces. For the purposes of this course, we will focus on the web, and develop frontend interfaces using web technologies such as HTML, CSS, and Javascript. React knowledge would also be useful as you progress further.

Backend Technologies

The backend refers to that part of a software that allows it to operate and cannot be (necessarily) accessed by the user directly. Most private data, user data, business logic, data processing, etc happens on the backend, whereas the frontend is just used to offer a visual representation of that data and allow the user to perform certain tasks with it. The backend receives requests from the clients, and contains the logic to send the appropriate data back to the client. Backend services can be written in a variety of programming languages - Python, Java, Javascript, Go, Rust, etc.

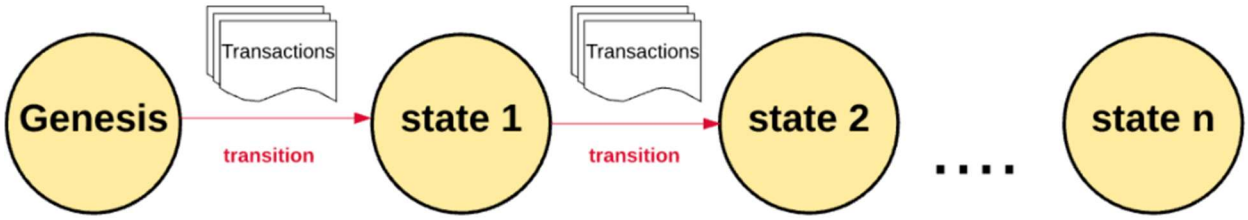
What is blockchain



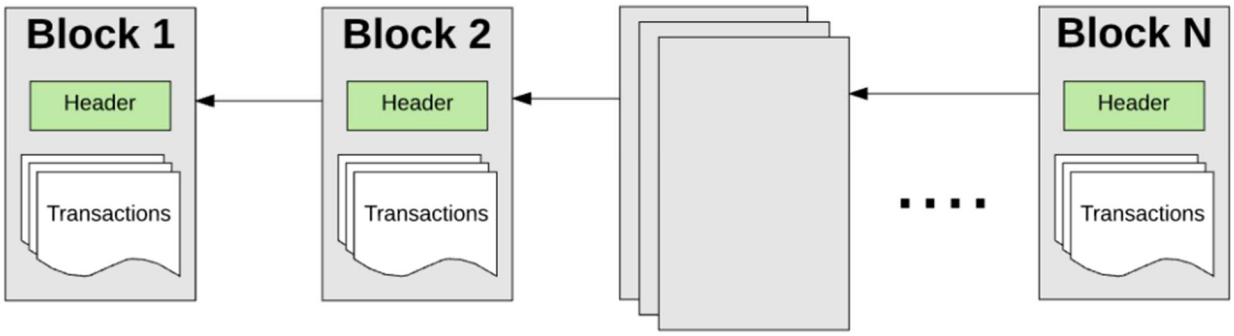
A blockchain is a shared, distributed, and permanent database shared among multiple nodes on a computer network. They record data in a way that makes it probabilistically impossible to modify or hack the system.

blockchains record data as a chain of blocks. Each block contains a group of transactions, which could be transferring assets around the network, or updating the information stored on the blockchain.

State Management



Blockchains start off with a Genesis State when they launch. Bitcoin's genesis state happened in 2009 when the public network launched. Ethereum's Genesis State happened in 2015, when it launched.



Since there are millions of transactions, transactions get grouped together in blocks. Hence the name. These blocks are chained together in a cryptographically verifiable way so they are historically traceable. The current state of a network can be recalculated at any time by starting from the genesis block and transitioning the state according to each block's information up until now.

Nodes

A blockchain network is managed autonomously through a peer-to-peer distributed network of computer nodes. Without going into too much detail, you can simply think of each node in the network as keeping a copy of the global transaction ledger. Therefore, each node can individually verify and audit transactions happening on the network and ensure there was no illicit behaviour.

Another type of node, called a mining node, is responsible for grouping together new transactions being made on a network into a block, verifying them, and proposing the block to be included onto the global ledger by everyone else. Mining is computationally hard, and very important to do securely, so miners whose blocks get accepted are given a token reward for their hard work.

Decentralization

By storing data in a peer to peer network of nodes, the blockchain is a decentralized network. This has significant benefits over the traditional approach of storing data in a centralized manner. There are significant examples of problems with centralization - a few of which we will list here:

- Data breaches in centralized systems expose a lot of data
- Centralized authorities can censor and shut down speech
- Reliance on a central authority means upstream problems affect downstream consumers (e.g. AWS going down means most of the internet goes down with it)

On the other hand, decentralization brings about the opposite benefits.

- No censorship as there is no single authority or middleman that can censor you
- No downtime as the overall network is running across 1000's of nodes across the globe
- Highly attack resistant making it infeasible to manipulate or destroy data

Use Cases

- Cryptocurrency
- Smart Contracts
- Decentralized Finance
- Gaming
- Supply Chain Tracking
- Counterfeiting Protection
- Data Privacy
- Decentralized Governance
- Verifiable ownership of assets

What is Web3

Web1 (1980s - early 2000's)

The first phase of the Internet, Web1, was mainly about providing the everyday consumer with online content and information.

As consumers could only read information or content online, and not yet interact with it, Web1 was incredibly static.

What is Web2?

Web2 is the version of the internet most of us know and use today. Where Web1 was static and "read-only," Web2 is "read-write," and interactive. Under Web2, the internet became more usable: web2 was dynamic and users could consume, interact with, and create content on the internet themselves.

In the centralized internet we know today, Apple can take a 30% cut on all paid-app downloads and in-app purchases, Twitter and Facebook can de-platform the President of the United States, and the everyday consumer has less privacy, security, and control over their online information than ever before.

We also see a lot of data breaches happening all across web2 leading to reduced security and privacy for one's personal data. When a user's data gets breached its easy for them to become a victim of identity theft, personal attacks etc.

What is Web3?

Web3, the future internet we're moving towards, is a decentralized internet. Under Web3, the internet is shared online and governed by the collective "we," rather than owned by centralized entities. The Web3 world is one that has open-source protocols at its foundation. Web3 is about rearchitecting internet services and products so that they benefit people rather than entities.

Web3 enhances the web we know today by making it decentralized, distributed, open, trustless and permissionless.'

- It is getting built such that everything would happen in a decentralized distributed way giving no central authority access to control the system.
- 'Open' as it would be open sourced software built by an open and accessible community of developers and executed in full view of the world.
- 'Trustless' in that the network itself allows participants to interact publicly or privately without a trusted third party.
- 'Permissionless' in that anyone, both users and suppliers,can participate without authorisation from a governing body.

What is Ethereum?

1. Theory

What is Ethereum

Ethereum is a decentralized blockchain that supports smart contracts. Unlike Bitcoin, which only supports the transfer of the Bitcoin token around the network, Ethereum is more general purpose.

Developers can build dApps, or decentralized applications, which can be executed on the Ethereum network on the Ethereum Virtual Machine (EVM). The global state of Ethereum therefore consists of more than just the balance of every account, but also the state of each dApp.

dApps are built on Ethereum using it's programming language, Solidity. You can write smart contracts using

Solidity and deploy the smart contracts to the Ethereum Network.

Ethereum currently uses which consensus algorithm → PROOF OF STACK

History

He went ahead and proposed the development of a new blockchain platform with a Turing-complete programming language (Solidity), that went on to be what we know as Ethereum.

What is Ether

Ethereum has a native currency called "Ether", or "ETH". This token is required to pay transaction fees for transactions done on the Ethereum network.

What are Smart Contracts

Smart contracts are small computer programs that are replicated and processed on all the computers on the Ethereum network without a central coordinator. Smart Contracts allow you to program contracts that can be automatically enforced by computer code.

The general-purpose nature of Ethereum allows for any number of possible applications to be built on top of it, which all inherit the security and decentralization benefits that come from running on the Ethereum blockchain.

ERC20 Tokens

In addition to Ether, people can create and use their own currencies on Ethereum. The most common form of currency is ERC20 tokens. ERC20 Tokens are smart contracts that fit a specific standard. Developers can extend beyond the standard, but should meet the minimum requirements when making their own token. The standardization allows for digital wallets to easily support all types of tokens, without needing specialized code for each token created.

Ethereum has smaller blocks and Ethereum's block time is shorter

Setting Up a crypto wallet

Introduction

To understand crypto wallets fully, we have to understand some concepts about the blockchain, which will help us in understanding how a wallet aids us. Let's start off.

What is an address? 😒

An address is a string of text generated using cryptography to represent your account on the blockchain. This address can be shared publicly with others, and is completely safe to do so. You can send and receive funds from and to your wallet address. Basically, the address is your unique identifier on the blockchain and represents your 'account'. An example of an Ethereum address is: `0x01573Df433484fCBe6325a0c6E051Dc62Ab107D1`.

What are private keys?

Each address has an associated private key. As the name suggests though, this is meant to be kept private and not shared with anyone.

Anyone who has the private key has access to make transactions from your address i.e. send money from your address to theirs.

A private key looks something like this: `E9873D79C6D87DC0FB6A5778633389F4453213303DA61F20BD67FC233AA33262`

If you think of your address as a username for your account, the private key is it's password.

Since blockchains are decentralized, there is no 'forgot password' option.
If you lose your private key, you lose access to your account.

For developers, we often use the private key as part of our codebase to perform certain transactions, such as deploying our own smart contracts to the Ethereum network. While you are still learning, we highly suggest you use a separate account entirely for development than you use for storing any sort of funds. Unfortunately, beginner developers often use the same account they have funds on, and accidentally share their codebase publicly - and hackers can see your private key in the codebase and end up stealing funds. Please take that as a tale of caution.

What is a seed phrase?

A seed phrase is like a master password - the password of passwords!

Think of a password manager, something like Lastpass or 1Password. These applications, within them, store your usernames and passwords for other apps securely, and themselves have a password. So, if someone hacks your password manager, they also get access to all accounts stored within it.

A crypto wallet is kind of like a password manager, where you can manage multiple blockchain accounts. If the private key is the password to a single account, the seed phrase is kind of like the master password for that wallet.

When you create a new crypto wallet, you will be presented with a seed phrase you should absolutely securely store and back up. Any new accounts you generate from inside that wallet will all be linked to the seed phrase. That

one seed phrase will always generate the same accounts, with the same private keys and addresses for each.

So for example if you created a wallet, and then created 5 accounts within it, your seed phrase manages all 5. If you wanted to switch to a new wallet, you could either import the 5 wallets individually - by using their individual private keys - or just import using the seed phrase, and it would regenerate the same 5 accounts.

An example of a seed phrase is: `dove lumber quote board young robust kit invite plastic regular skull history`

What is a crypto wallet then?

Crypto wallets are a manager for your accounts, and mainly their private keys. They also allow you to interact with decentralized applications, and allow connecting to a dApp through the wallet, acting as a single sign-on for all applications built on the blockchain.

Introduction to Remix

Remix is an open-source, web and desktop Integrated Development Environment (IDE) for Ethereum development. It is the easiest development tool to get started with building on Ethereum, and has a huge collection of plugins to extend its experience.

Remix helps you write Solidity code directly in the browser, and has tools for testing, debugging, and deploying your smart contract to the blockchain.

You can write code on Remix in languages other than Solidity?

Remix allows you to interact with contracts you did not deploy

More on this site : https://remix-ide.readthedocs.io/en/latest/

Intro to Solidity

What is Solidity ?

- Solidity is an object-oriented, high-level language for implementing smart contracts. It is designed to target Ethereum Virtual Machine(EVM)

- It is statically typed, supports inheritance, libraries and complex user-defined types among other features.

Building in Solidity :

1- Initialize smart contract:

```
// Define the compiler version you would be using
pragma solidity ^0.8.10;

// Start by creating a contract named HelloWorld
contract HelloWorld {

}
```

2- Variables and types

There are 3 types of variables in Solidity

- Local
  - Declared inside a function and are not stored on blockchain
- State
  - Declared outside a function to maintain the state of the smart contract
  - Stored on the blockchain
- Global
  - Provide information about the blockchain. They are injected by the Ethereum Virtual Machine during runtime.
  - Includes things like transaction sender, block timestamp, block hash, etc.
  - Examples of global variables

Example on solidity variables:

```
pragma solidity ^0.8.10;

contract Variables{

    /* `public` means that the variable can be accessed internally
     by the contract and can also be read by the external world
    */
    //uint stands for unsigned integer, meaning non negative integers
    //different sizes are available.
     /*
        ******** State variable **********
     */
    unit8 public u8=10;
    unit public u256 = 600; // uint is an alias for uint256

    int public i = -123; // int is same as int256 support negative numbers

    // address stands for an ethereum address
    address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa7;
```

```
    // bool stands for boolean
    bool public defaultBoo1 = false;

    // Default values // Unassigned variables have a default value in Solidity
    bool public defaultBoo2; // false
    uint public defaultUint; // 0
    int public defaultInt; // 0
    address public defaultAddr; // 0x0000000000000000000000000000000000000000
    function doSomething() public {
        /*
        ******** Local variable **********
        */
         uint ui = 456;
        /*
        ******** Global variable **********
        *//*
            block.timestamp tells us whats the timestamp for the current block
            msg.sender tells us which address called the doSomething function
        */
        uint timestamp = block.timestamp; // Current block timestamp
        address sender = msg.sender; // address of the caller
     }
}
```

Example on solidity functions – if/else - loop:

```
contract Conditions {
    // State variable to store a number
    uint public num;
    function set(uint _num) public {
        num = _num;
    }
    function get() public view returns (uint) {
        return num;
    }

    function foo(uint x) public returns (uint) {
        if (x < 10) {
            return 0;
        } else if (x < 20) {
            return 1;
        } else {
            return 2;
        }
    }
```

```
    function loop() public {
        // for loop
        for (uint i = 0; i < 10; i++) {
            if (i == 3) {
                // Skip to next iteration with continue
                continue;
            }
            if (i == 5) {
                // Exit loop with break
                break;
            }
        }
    }


}
```

Example on solidity arrays and strings

```
pragma solidity ^0.8.10;

contract Array {
    // Declare a string variable which is public
    string public greet = "Hello World!";
    //Arrays initialized here are considered state variables that get stored on
the blockchain
    // These are called storage variables
    uint[] public arr;
    uint[] public arr2 = [1, 2, 3];
    // Fixed sized array, all elements initialize to 0
    uint[10] public myFixedSizeArr;


    //function that gets the value of element stored in an array's indes
    function get(uint i) public view returns (uint){
        return arr[i];
    }
        /*
```

```
     Solidity can return the entire array.
     This function gets called with and returns a uint[] memory.
     memory - the value is stored only in memory, and not on the blockchain
              it only exists during the time the function is being executed

     Memory variables and Storage variables can be thought of as similar to RAM
vs Hard Disk.
     Memory variables exist temporarily, during function execution, whereas
Storage variables
     are persistent across function calls for the lifetime of the contract.
     Here the array is only needed for the duration while the function executes
and thus is declared as a memory variable
    */
    function getArr(uint[] memory _arr) public view returns (uint[] memory) {
        return _arr;
    }
     /*
        This function returns string memory.
        The reason memory keyword is added is because string internally works as
an array
        Here the string is only needed while the function executes.
    */
    function foo() public returns (string memory) {
        return "C";
    }
    function doStuff(uint i) public {
        // Append to array
        // This will increase the array length by 1.
        arr.push(i);
        // Remove last element from array
        // This will decrease the array length by 1
        arr.pop();
        // get the length of the array
        uint length = arr.length;
        // Delete does not change the array length.
        // It resets the value at index to it's default value,
        // in this case it resets the value at index 1 in arr2 to 0
        uint index = 1;
        delete arr2[index];
        // create array in memory, only fixed size can be created
        uint[] memory a = new uint[](5);
        // create string in memory
        string memory hi = "hi";
    }
```

```
}
```