1. **History:**

* Bitcoin whitepapaper released on Oct31,2008 by Sakushi Nakamoto
* Paper described the creation of a system that would allow people to pay each other directly without the need for a central authority like a bank.
* Out of this paper, bitcoin network was eventually brought online in January 2009, which is the original bitcoin network
* One of the key characteristics of the bitcoin network was the idea of a blockchain.

**Bitcoin Blockchain:**

* Bitcoin blockchain was a tool to store a list of monetary transactions between different people
* Only purpose of Bitcoin was to enable financial Transactions

**Innovations from Bitcoin:**

* People realized that they could use some of Bitcoin’s original innovations to not only handle transfer handle of currency, but other types of exchange as well.

**Vitalik Buterin:**

* Mainly ***Vitalik Buterin*** proposed it and published a white paper on the same too.
* Vitalik was a proponent of Bitcoin project, but he thought that the Bitcoin project was too simple
* Vitalik envisioned using Blockchain technology to create far more complex applications.
* He describes Ethereum to a) Create sub cryptocurrencies b) domain name registration systems and c) Gambling applications
* Thus Vitalik did not want to just Bitcoin to transfer money. Instead he wanted to figure out some way to use these innovations to create much more advanced applications
* In his whitepaper, Vitalik goes on to describe the most important part of the Ethereum network something called a smart contract.

**Smart Contract:**

* It is a piece of code that lives in the Ethereum blockchain.
* This contract can be instructed to do certain things, by having a person or another contract send a message to it.
* Smart contracts are the absolute core of what Ethereum is
* Vast majority of this course will be spent on learning about the behaviour of these contracts

**Ethereum:**

* Ethereum system went online on July30, 2015 and it continues to flourish until this day.

**Links to Original Papers:**

* Bitcoin: A Peer-to-Peer Electronic Cash System (<https://bitcoin.org/bitcoin.pdf>)
* Ethereum: The Ultimate Smart Contract and Decentralized Application Platform ([http://web.archive.org/web/20131228111141/http://vbuterin.com/ethereum.html](http://web.archive.org/web/20131228111141/http:/vbuterin.com/ethereum.html))

1. **What is Ethereum:**

* When we work with Ethereum, we are working with a network of computers
* These networks are used to transfer money between different parties like you and me.
* And they are also used to store data.
* The transfer of money and the storage of data are what allows us to create interesting applications.



* There are many different Ethereum networks.
* In reality there is one main Ethereum network that everyone uses for the deployment of production applications and it is where Ether coins that worth US dollars or real money .
* However there are many other Ethereum networks out there
* There are test networks – Networks that used solely for testing code and testing transactions.
* You can even create your own private Ethereum network on your own computer that is restricted to just you or you can open your own network and open it other people as well.
* The point here is : There is not just one Ethereum network out there and there is many.
* A node is a machine(desktop or laptop or any computer) that is running an Ethereum client.
* Networks are formed by one or more nodes
* So here, we take a node, install some software on it , run that software and then it connects to an Ethereum network and starts becoming part of the network overall.
* All these nodes connect together to form the actual network.



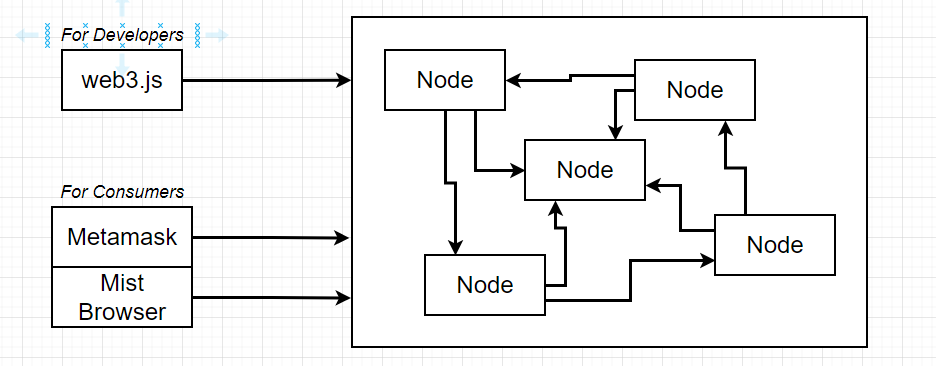
* Anyone can create and run a node. It is not just some specific set of people out there or important companies who run nodes on the real Ethereum network.
* All we have to do is to turn on our laptop, download this client software and run it. That’s it we a locally hosted node.
* Each node that we create or we can say as: each node on the network has a full and separate copy of the block chain.

1. **Blockchain:**

* Block chain is a database that stores a record of every movement of money between different parties and it is also a place where we store data.
* Blockchain can be considered to be a simple database and nothing else.

1. **Interfacing with Ethereum Networks:**

* Critical part left out in last section: did not discuss about how common people like you and me can connect to Ethereum network, make changes to it like sending money, store data or do anything else.
* We will see about different common ways that people like you and me can connect to this network and send money or store data.

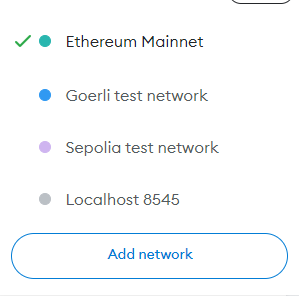
****

**Two Technologies for Connecting:**

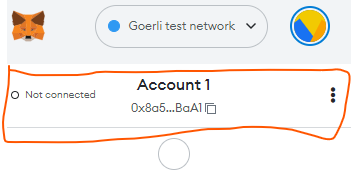
* Two different groups of technologies we need to be aware of, for connecting to the network.
* i) Technologies that are used by developers or essentially created for developers like you and me. For that we are going to make use of library called Web3.js.
* We can think of web3 as a portal or our window into the Ethereum Network
* It is going to allow us to send money, store data, deploy contracts or do essentially whatever we want to do on the network.
* Web3 is primarily used by developers and used to create applications.
* Ii) People who might want to work with Ethereum who are not programmers – i.e Consumers. For consumers there are two solutions in the market right now – a) MetaMask b) Mist Browser
* Metamask is a chrome extension or a browser extension that allows common people to interact with the Ethereum network.
* Mist Browser is more of a full featured web browser that is intended to be used to browse different Ethereum applications.
* We will be using Metamask more and wont be using Mistbrowser because it is still in beta phase

**Main Network:**

* Main network that we see in metamask extension is the network that is selected by default
* Main network is the public production network
* It is where coins are worth something and it is where we deploy real applications that we want to be used by our users.
* These three networks below main are the test networks:

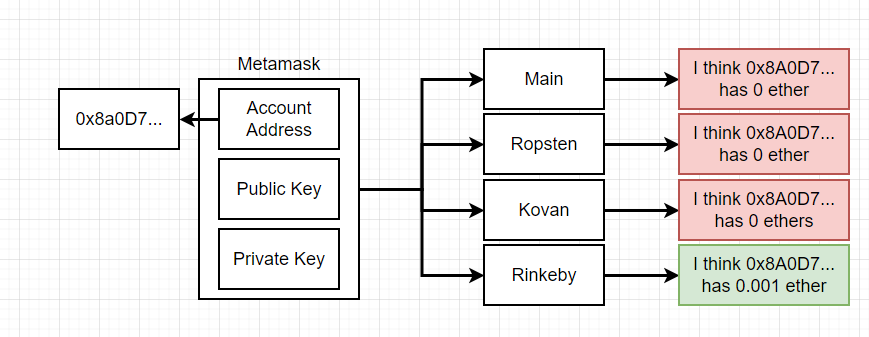
****

* These networks are used to test code and to get free ether to test our contracts with.
* Main network needs real money . Hence we wont use it.
* For development we wont use real money and we will use fake money and that’s why we will be using Goerli network.
* Below that we can also find option for Localhost 8545 , which is an option to connect to your local machine.
* We usually use localhost 8545 when we are hosting a local Ethereum node or a local Ethereum test network.
* Many times development is done on local networks, so we would be using localhost 8545 here.
* But you and me, will be using Goerli a lot rather than our own Local node.

****

1. **Ethereum Accounts:**

* Earlier when installed Metamask and entered initial password, metamask extension automatically generated a new account for us.
* We can see Account1 and weird characters below it
* When Metamask created an account for us, it created an account that has three distinct pieces of information a) an account address b) a public key, c) and a private key.
* These are the three pieces of information that constitute an account on the Ethereum network.

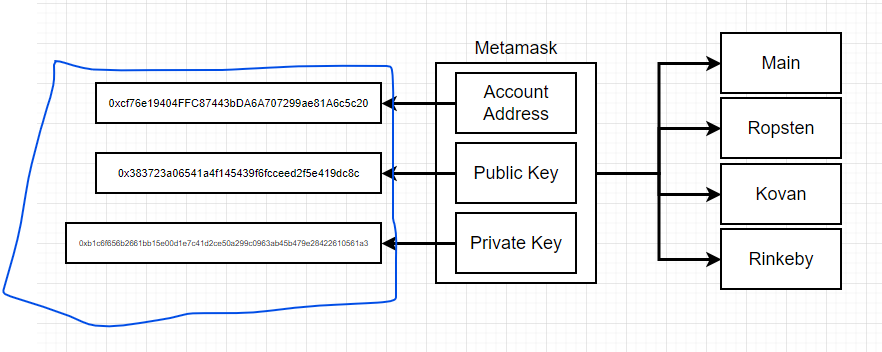


**a) an account address**

* The account address can be thought of as essentially being like an email address or like username
* It's a unique identifier that can be shared with anyone in the world, and it tells other people who you are.
* It identifies your account in some fashion.
* Again, you can think of it as being like an email or a username.

**b) a public key c) private keys**

* The other two pieces of information are the public key and the private key.
* These two pieces of information combined together to essentially form a password of sorts.
* The public key and the private key are used to authorise the sending of funds from your account to other accounts.
* So if someone does not have the private key to an account, they don't have true access to any of the funds that are assigned to that account.
* The account address, public key and private key are all stored as **hexadecimal numbers.**

****

**private keys:** 0xb1c6f656b2661bb15e00d1e7c41d2ce50a299c0963ab45b479e28422610561a3

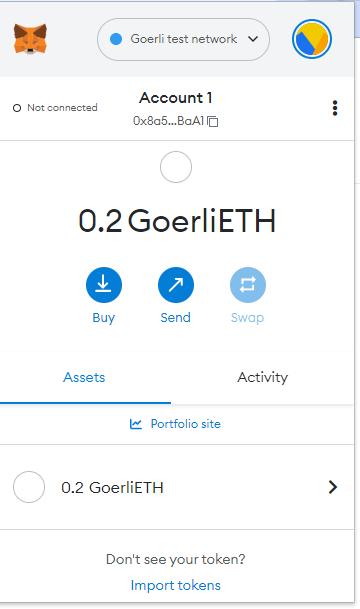
* Private key is long hexadecimal number . it is 10^76 large
* That's a gigantic number incomprehensibly large.
* This number is what uniquely constitutes our private key.
* The private keys should never be shared with other people, because if anyone else ever gets the private key, then they can easily take all the funds that are assigned to your account.

**One Account for all different Ethereum networks:**

* In the Ethereum world, one account is used across all different networks that we ever interact
* One account is used across all networks.
* We don't have to create separate accounts for each network unless we very specifically want to.

1. **Getting Test Ether to Use in the Course**

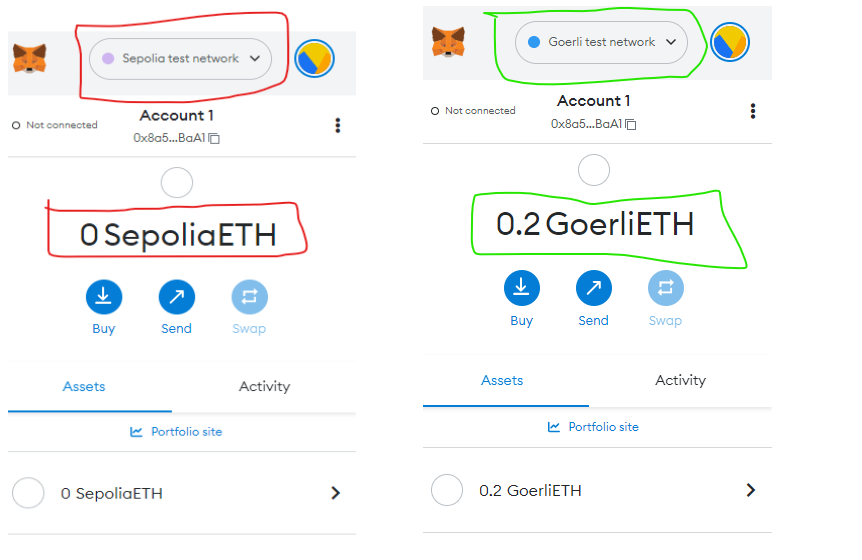
* Follow instructions mentioned in this page, to get .2 test ethers for free
* [**https://www.udemy.com/course/ethereum-and-solidity-the-complete-developers-guide/learn/lecture/25992414#learning-tools**](https://www.udemy.com/course/ethereum-and-solidity-the-complete-developers-guide/learn/lecture/25992414#learning-tools)
* Our try:

****

1. **What is a transaction?**

**Same Account different Values in different Networks:**

* Our account now has 0.2 Goerli Eth ether assigned to it on the Goerli Test network.



* If I select the server dropdown or the network dropdown up here and flip on over to say the Sepolia network, you'll notice that our balance goes down to zero.
* But if we go back on over to Goerli, we go back up to 0.2 Eth
* Hence every account that we make in the world of Ethereum lives in its own separate universe,like its own world, and it is completely decoupled from any of the different Ethereum networks.
* As far as Goerli is concerned, our account has 0.2 ether inside of it.

But as far as any other account is concerned, our account has zero ether inside of it.

* All these different networks can still have ties to this one singular account that we control.

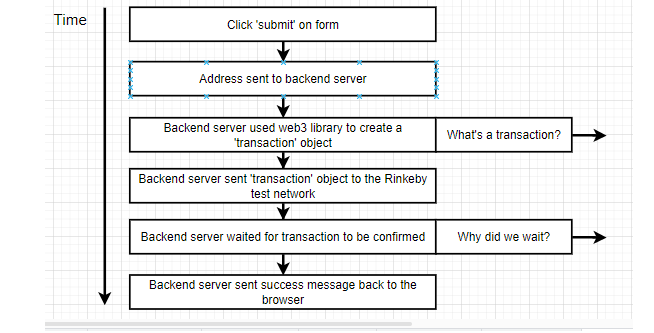
**Technical description: what happened when we submitted the address on that**

**Form:**

1. First we entered the address and clicked the submit button.
2. That web page then took the address that you entered and submitted it to a backend server.

**Node.JS server-------Connects to Network via--🡪Web3 library:**

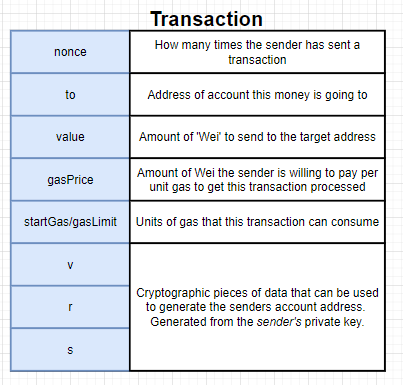
* + This is a Node.js server that I wrote specifically to take your address and send a tiny bit of ether to it. To make that server. I use the Web3 library.
* Remember, web3 library is a tool that we use for developers to interact with any given Ethereum network, not just the main network and not just Goerli. We can use it to interact with any Ethereum network.
* It's a programmatic portal that we can use to interact with all these different nodes that exists.



1. Back end server use Web3 library to create transaction object.

**Transaction:**.

* You can think of a transaction as being a record that describes one account attempting to send money to another account.
* A transaction is created any time two accounts exchange some amount of money.
* So when I just sent some money to you, I created a transaction object and then submitted it to the Ethereum network to be processed.



* This object has a variety of different properties assigned to it, which you'll see labelled over here to the left hand side

**Nonce:**

* The first property that exists on a transaction object is a number that tells us how many times the sender has sent a transaction. In this case, my server is the sender that is the person or the entity that just sent you some amount of money.

Eg:  *So if that account has been used, say, 1000 times to send 1000 different transactions than to send you money on that 1000 in first transaction, that transaction would have a nonce of 1001.*

**To:**

* This is the address of the account that some amount of ether is going to be sent to.
* So in the transaction that was just issued, the to field would have had an address equal to your account address.

**Value:**

* The value is the amount of ether that we want to send from the sending account to the target account.

**gas price:**

**start gas**:

* start gas is sometimes called alternatively gas limit.

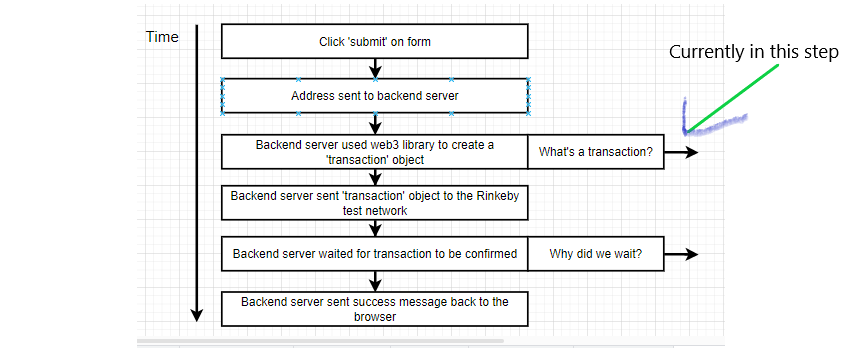
**V,r,s**

* V, r, and S are Cryptographic pieces of data that can be used to generate the senders account address.  Generated from the sender's private key.
* V r s properties are generated by the sender's private key, so the sender will take their private key. They generate these three values, and those three values can then be used to generate the account address of the person who is attempting to send the money.
* Generating the v, r, and s from the private keys or from the private key is a one way process. So if you have the private key, you can generate v r s, but if you have v r s, you cannot kind of back calculate the private key.

**Technical description: what happened when we submitted the address on that**

**Form(Continued):**

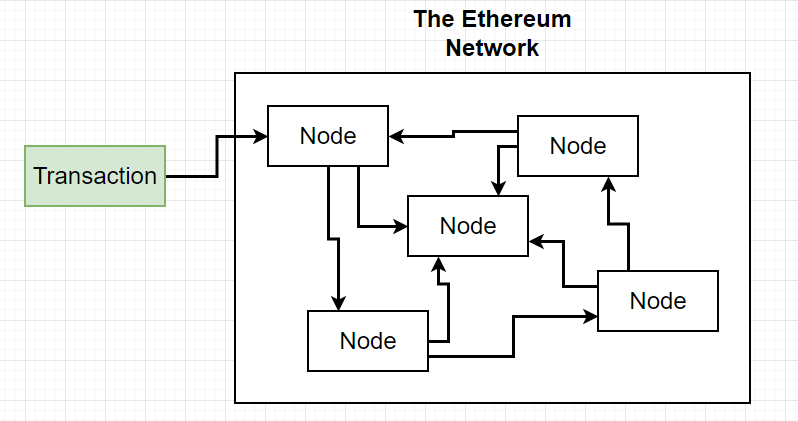
* Back at our flow diagram here, we're right on this step right here where we had said that we had used the Web3 library to create a transaction object.



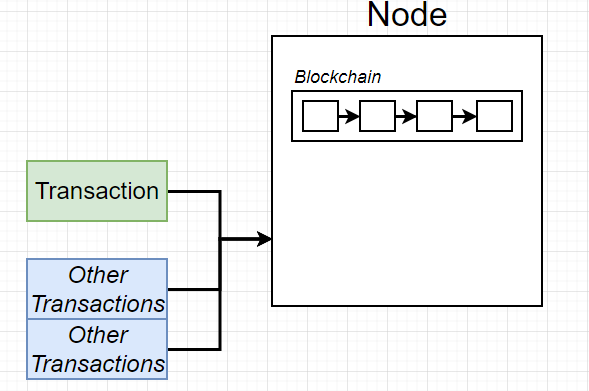
* After that object is created, the same Web3 library is then used to send that transaction object to the Goerli test network.
* The transaction goes out to the network and then we wait for the transaction to be confirmed(32 sec wait)

**Why Did we wait?**

* We sent our off our transaction to the Ethereum network.



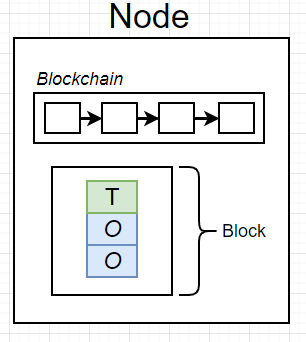
* when we sent the transaction out, The transaction goes to one specific node.
* So our applications are always going to be interfacing with one node, and that node will be communicating with the rest of the network later on.



* The node has an entire copy of the blockchain.
* And right now let's just understand the blockchain to be essentially a database.
* So our transaction arrived at this node, but you have to keep in mind that you and I are not the only people in the world.
* There are other people in the world who want to submit transactions as well.
* So maybe at the exact same time that you and I submitted a transaction, maybe two other people did as well.
* So maybe there are total three transactions that are coming into this node at the exact same time.

**Block**

* This node is going to take those transactions, which I've now represented here as T representing our transaction and O representing other people's transactions.



* And it assembles all three of those transactions into one list of transactions, which we refer to as a block.

**Validation Logic takes 30 seconds:**

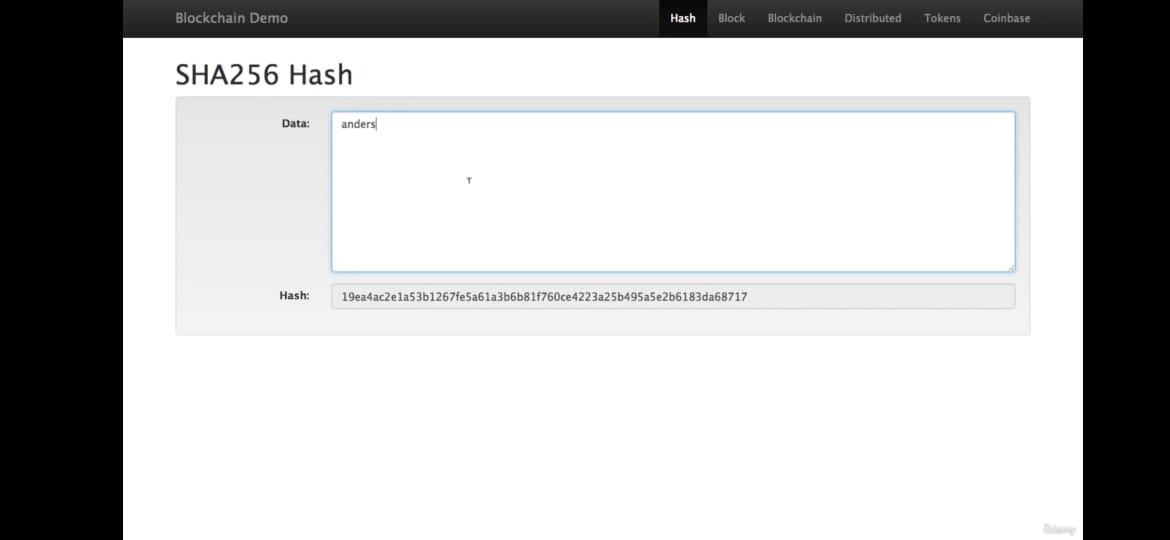
* The node then runs some validation logic on this block.
* That validation logic is what takes the 30 seconds.

**Mining:**

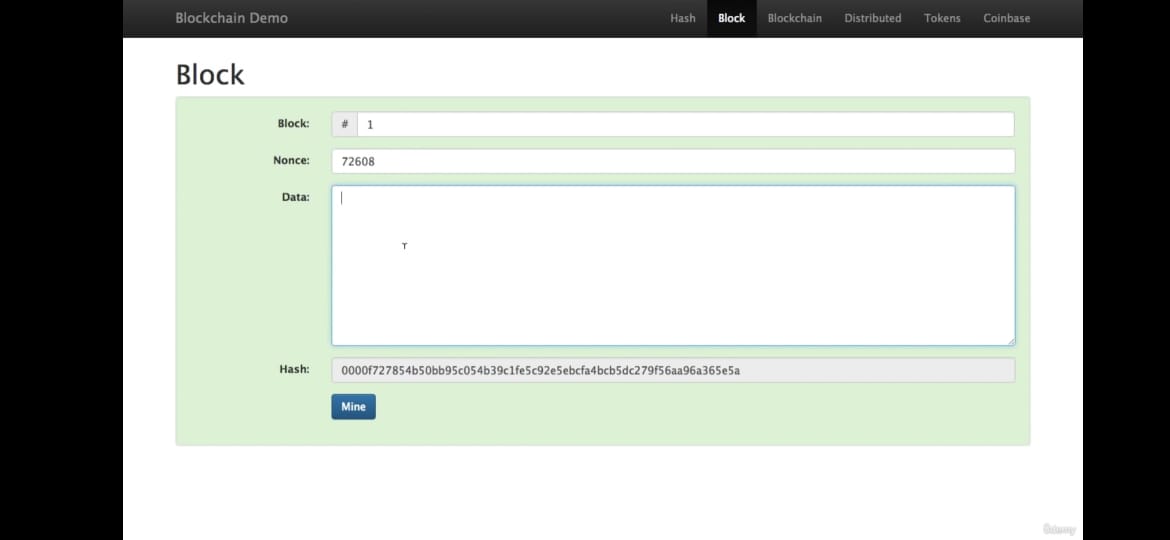
* When it comes to Bitcoin and Ethereum, you've probably heard the term mining
* So when these transactions right here get assembled into the block, the node starts running some calculations on the block, and that process is referred to as mining.
* We can also call process of running those validation logic as mining.
* The process of mining is where things get complicated.

1. **Basic Blockchain Demo:**

* Before we get started, we will take a look at: SHA256 hash and a hash.
* Hash looks like a bunch of random numbers and it is the fingerprint of digital data.



* Interesting thing about Hash is, regardless of if there's a tiny amount of information, no information or huge information, you're always going to get a hash that is this long as shown in figure.
* We are going to extend this idea of a hash to a block.



**Block Display:**

* So this is a block and it looks exactly like the hash.
* Data section is broken into three sections: a) Block b) Nonce c) Data

**BLOCK**.

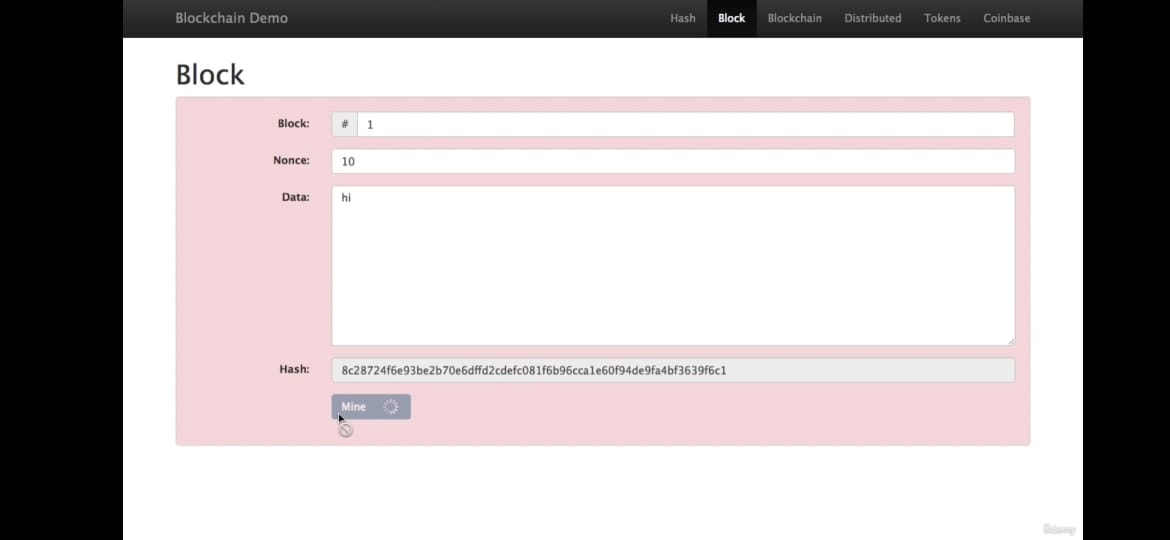
* This is just some kind of a number.
* This is block number one as shown in figure.

**Nonce:**

* Nonce, which is just yet another number.

**Hash:**

* However, the hash of this, which includes all of this information, begins with four zeros.
* That's a relatively unusual hash.
* If it starts with zeroes and if it looks green then it is signed



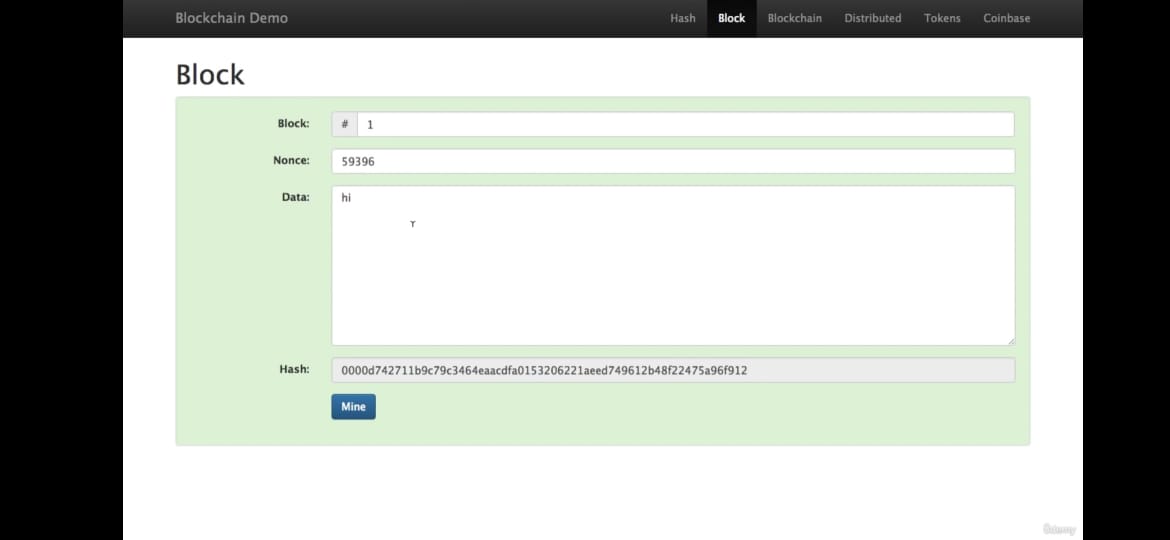
* If I change information, Hash changes. This hash does not start with four zeros.
* Now the big background here has turned red.
* Hence the block with this information not a valid or assigned block.

**Nonce in Block:**

* That’s where this nonce comes in.
* Nonce is a number which we set, so that this hash starts with four zeros again.

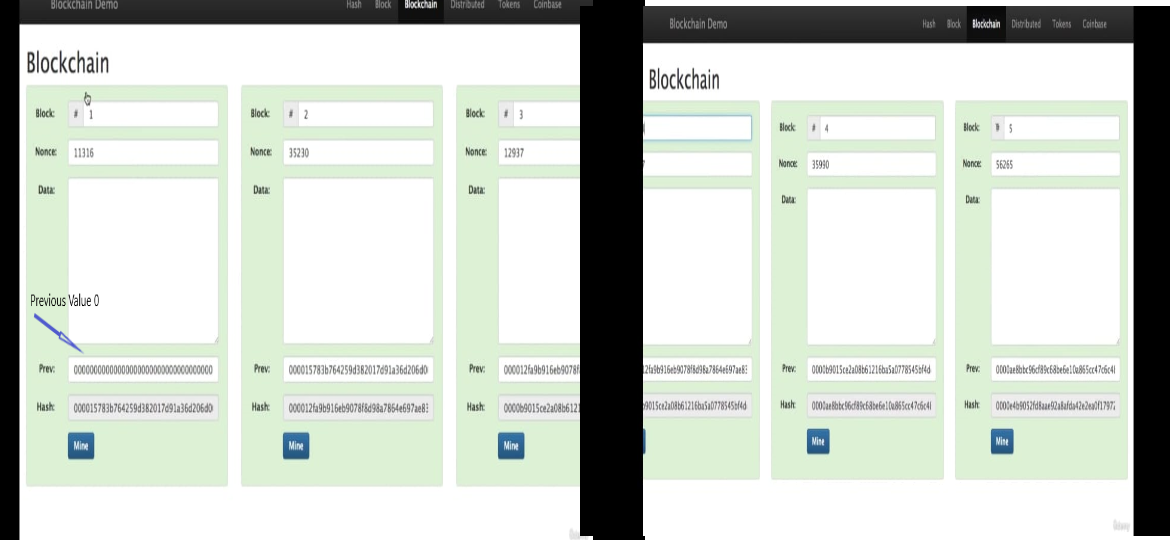
**Mine button:**

* when I press this mine button, it's going to run through all the numbers from one all the way up to finding exact number for Nonce, where the hash starts with four zeroes. This process is called mining.



* Once mine button is pressed, it's checking all of the numbers from one onwards
* Now it's stopped at 59,396. And that one just happens to hash out to something that starts with four zeros and it satisfies my little definition of what a assigned block is . So that's a block.

**What is blockchain:**

* Blockchain is a Chain of these blocks.
* All right, so here's my blockchain.
* Block chain in figure has 5 blocks. Block1 has previous value as 00000. It's actually just a fake number.
* The point to note here is : Previous contains the hash value of the previous block
* Each block through its previous value, points backward
* If I change data in block2, block2 hash will change and thus it will invalidate all blocks from 2 to 5.
* So if we want to solve this issue, for the changed data we should remine and thus new nonce will get selected and thus block will get validated
* Blockchains can have 400,000 or 500,000 blocks very easily.
* So rather than checking through all of them, all you really have to do is look at the hash of the most recent one.

**Blockchain with Token:**

* The above blockchain is not useful, because it does not contain any data.
* What we really need is token like below:



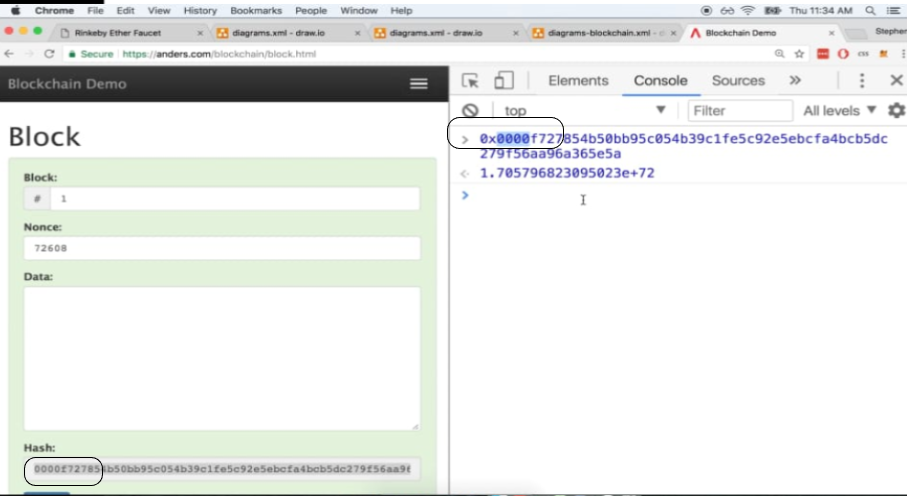
* Now, one thing I would mention here is that we're not listing, you know, Darcy has $100 and he's
* giving 25 of it to Bingley.
* The only thing we're saying is Darcy gives 25 to Bingley.
* We're not remembering a bank account balance, we're only remembering money movements.
* So this begs the question, does Darcy have $25?
* Well, we have a problem here in this version of the blockchain.
* We don't actually know if Darcy has $25.
* So let's look at a coin based transaction.
* So if we look back here, a coin base, we're going to add a coin based transaction to our blocks.
* And this is this is very similar to what we've seen before, but we've just adding a coin base at the top.
* And what's that what that saying is we're going to invent $100 out of thin air and give it to Anders.
* And there's no transactions in this block because nobody had any money previous to this.
* In the next block, another $100 comes out of nowhere and goes on or is I'm a fan.
* I love it.
* Right?
* I'll take 100 bucks and now we have some transactions.
* You can see that they're all from Anders.
* They're all from me because I'm the only one who has any money at this point.
* So I'm sending ten of my dollars to Sophie.
* Do I have $10?
* Yeah, I do.
* I look back and I see that this Coinbase transaction has given me 100, so I have at least ten and I can send it on.
* And you add all these up and they don't go over 100 and a follows, sort of a basic rule of, of a currency. Like you can't invent it out of thin air.
* You can't create money out of thin air.
* it's dispersion is controlled.
* So now if we look at this blockchain that we've created and we zip forward in time and we notice that
* we see that Jackson is giving Alexa $2, and so does Jackson actually have $2 to give Alexa?
* When we go back to the block before and we see that Emily, who had gotten $10 from Anders, gave ten
* to Jackson.
* And so Jackson does have the money, so we can just go backwards and and find that out.
* And that's actually one of the benefits of having a previous year. It's easy to go backwards.
* We just look for the block that looks like that, that has that hash.
* So you points to blocks back in time and allows us to trace the provenance of any coin that we want.
* So that's a basic blockchain and we're running a currency on top of it.
* And as you know, blockchains are their many copies.
* Everybody has a copy of it.
* So if we mutate this and make it $6, these go invalid.
* It does not agree with these blockchains down here, these copies of the same blockchain down here.
* So this resist tampering, which is what you want for a currency.
* It works very well for things that are small and transactional like like this.
* Go ahead and fix that.
* And they're they're of just a very efficient way to handle agreement on what has happened in the past.
* There's kind of this.
* So that's a basic blockchain and a token on it that we're glossing over some main points.
* But if you dig into the demo and click through these things and play around with it, you get a better
* and better idea of how this works.
* There will be a part two where we go into a little bit more detail about how the transactions are created.

1. **Block Time**

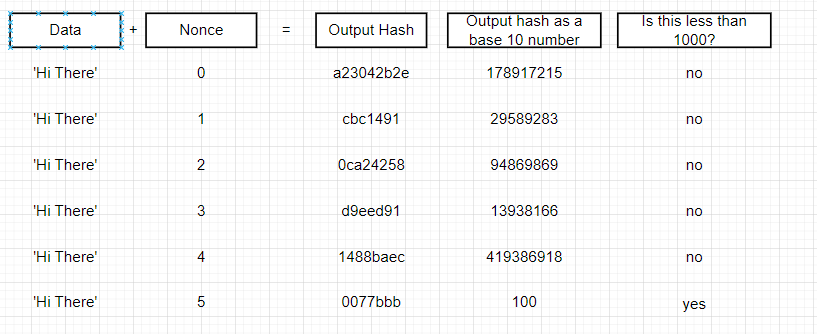
* Anders video was of a very generic general blockchain.

**How Ethereum Blockchain works?**

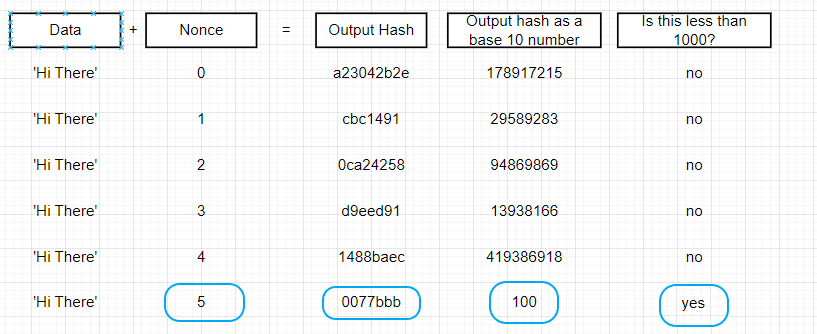
* Anders said during mining process, we are looking for a hash that starts off with some number of leading zeros
* That's not entirely correct
* What we are really looking for: is a hash that is less than some target value.



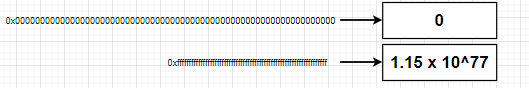
* So 0000f727 might be less than some target value that our algorithm is looking for.
* Let me show you an example of this in a flow diagram to just make sure it's really clear.



* So on the left hand side here, we've got some amount of data that will be input to our given block. And then next to it, we have that nonce value, which you can think of as being a counter variable.
* So on the first step through that proof of work algorithm, we take our data of Hi there. We join it together with the nonce.
* Like we literally stick those two values together and then we hash the output which might come out as : a23042b2e
* You can then imagine that behind the scenes we take that hash right there, convert it to a base ten number, and then we ask if this number right here is less than some target value(Assume to be 1000).
* If base10 number is not less than target value(100) we need to continue looking for another value.
* Lets increment nonce and do hashing again until we get decimal less than target value of 1000.
* We can see that for nonce 5 and hash value of 0077bbb, we get decimal less than 1000.



* That means we have found a valid solution.
* Thus we are actually looking at the value of the hash in decimal value rather than just a specific number of zeros.
* Another way of looking at this: is to imagine the entire range of possible hash values. The hashes that we are working are 64 characters long.





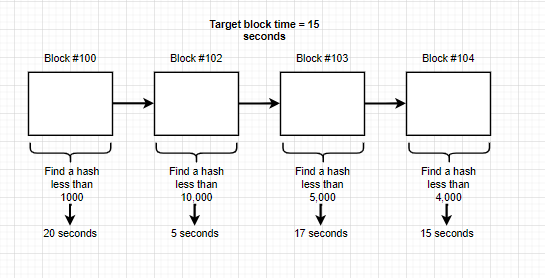
**Time for Rehashing data along with Incrementing Nonce:**

* Hence the entire process of rehashing the data along with the incrementing nonce takes some amount of time.
* Our computers can execute many, many, many hashes per second, like thousands, tens of thousands, hundreds of thousands of hashes per second.
* However, in the real blockchain, we are looking for small target values eg: less than 1000
* Practically , even though our computers can hash all this data very quickly, we have to execute so many hashes to eventually find that random number that is less than the target value, that it ends up taking many seconds of time in the real world.

**Block Time Definition:**

* Amount of time that takes to hash everything from zero to the target nonce value is called block time.
* That's essentially the amount of time it takes to run these hundreds of thousands of different possible hashes until we find that final value that actually equates what we're looking for.
* So when we think back to the transaction that we executed a couple of videos ago where we sent ourselves some money and that took like 30 seconds or so, that 30 seconds represented how long it took to run this proof of work algorithm to eventually find a solution.
* It was a block time of about 30 seconds.
* In reality, it was actually less than 30 seconds to run the proof of work algorithm.
* But once one individual node on the Ethereum network finds a solution, it then has to distribute that solution to other nodes.
* And so that additional amount of time is what eventually added up to the 30 seconds.

**Block time: What happens behind the scenes:**



* So what's really happening behind the scenes is that kind of difficulty target right here.
* The network looks at how long it took to calculate the previous block and it says, okay, if this was really high or weighed, if it took way too long to calculate the block, then let's raise the target number.
* Because when we raise the target number, that makes it more likely that someone is going to find a solution more quickly.

**Flow – Behind scenes:**

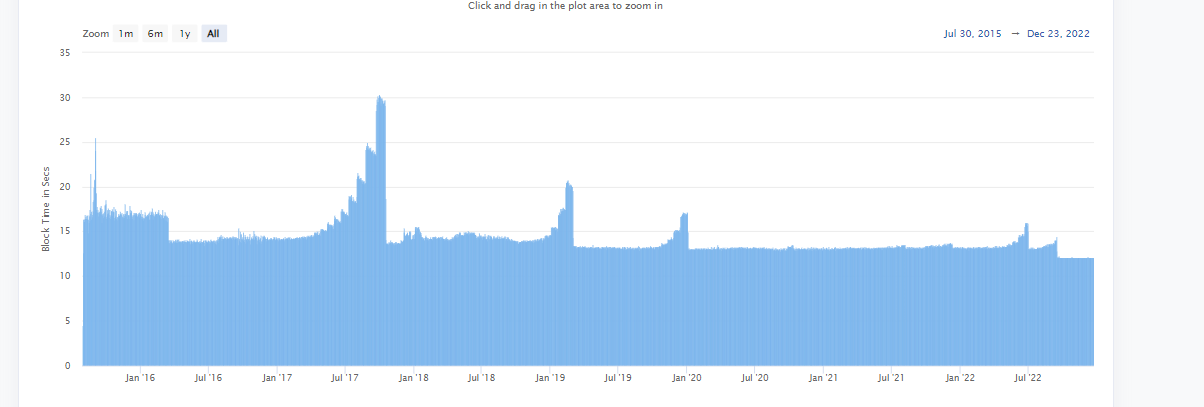
* So in practice, you might see a flow that looks like this right here.

A) Hash less than 1000 – 20 Seconds to execute

* B) Hash less than 10000 – 5 seconds
* C) Hash less than 5000 – 17 seconds
* D) Hash less than 4000 - 15 seconds
* **Why does it take different amounts of time?**
* The number of people who are running nodes on the Ethereum network at any given time is always in flux.
* People might be starting up nodes to calculate these hashes. Other people might be turning their computers off.
* So at any given time, the number of computers that are trying to calculate these valid hashes is always going to be changing.
* And so this difficulty target always has to be adjusted between every block to meet the actual computing availability of the Ethereum network.
* All right.

**Etherscan site:**

* <https://etherscan.io/chart/blocktime>
* Chart shows you the average amount of time that has taken to calculate hese blocks over the last two years or so.

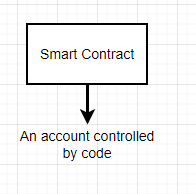


1. **Smart Contracts:**

* Smart contracts are what we use to build interesting applications on the Ethereum blockchain.

**Plain English definition of a smart contract:**

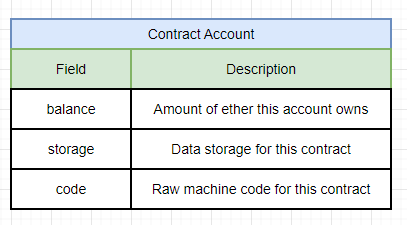
* Smart contract is an account, similar to what we created in metamask.



* But rather than being controlled by a human being like you and me, it is controlled by some amount of code which we authored by code developers
* This code instructs the smart contract how to behave.

**Properties in Smart Contract Account:**

* Properties in account we had: Public Key, Balance, Nonce etc.



**i) a balance:**

* Like our accounts, contract account too have balance.
* This records the amount of ether that is controlled by this account.

**ii) Storage:**

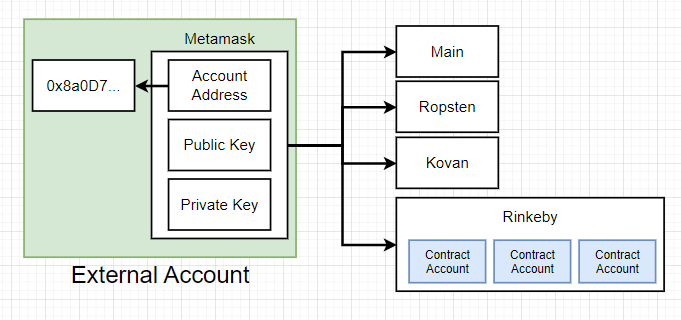
* Place, where we can store data related to our contract.
* Data is related to whatever application we build. It can be numbers, String or arrays.

**iii) Code:**

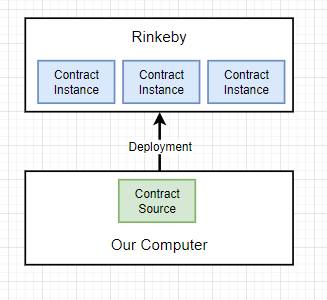
* Contains raw machine code for our contract. Ie Machine level code.
* So very machine level code.

**Critical difference between External Account(User’s) and Contract Account:**

* External accounts are accounts that humans own and we can use same account to connect to any type of network like: Main/Ropstei/Kovan/Rinkeyby/Goerli etc
* Thus external accounts are decoupled to any individual network.

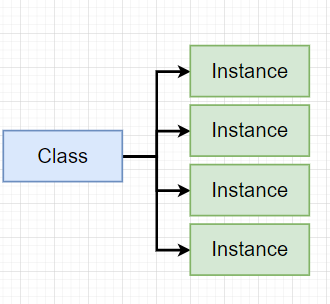


* On the other hand, contract accounts are specific to one individual network and it cannot be accessed across networks.
* Hence to use some code inside contract account present in a network, I need to take the account , redeploy as completely new account to other networks.
* So this is a little diagram of how we can start to picture these contract accounts.



**Contract Source:**

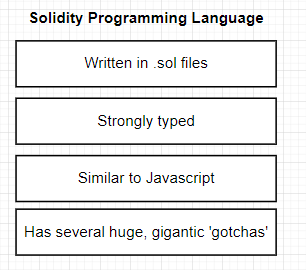
* When developer writes a code on his computer , that is called Contract Source.
* This code instructs the contract, how it should behave and how it should handle money.
* We will then take that source code, and deploy it to a network like Rinkeby.
* When we deploy this contract code, it creates an instance of the contract called as contract account.
* We can take a contract source code and deploy to multiple networks.



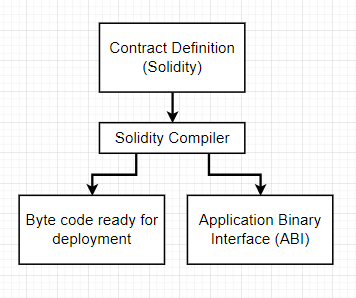
* *Hence we can assume Contract source code to be class and deployed instances as Objects in java*

1. **The Solidity Programming Language:**

* Solidity was invented specifically for authoring smart contracts, and so it has many features inside the language to make the execution of these contracts much easier.



* Solidity is written into .SOL files and they're compatible with any standard code editor.
* Solidity is strongly typed, while Javascript/Python are dynamically typed languages.
* Solidity is thought of as being similar to JavaScript.
* solidity is has several very big gotchas in general.



* Developer writes a contract definition using solidity.
* That contract definition is fed into a solidity compiler.
* Compiler gives two separate file outputs:

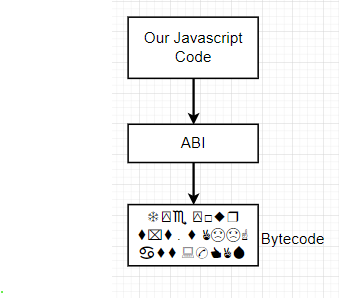
1. Byte Code file ready for deployment into Ethereum network

-> This byte code file is added as part of Contract Account.

B) Application Binary Interface(ABI)

* ABI is key for writing applications that can interact with these deployed smart contracts.

**Example:**

****

* After we deploy a contract and get an ABI out of it, we're going to eventually write some JavaScript code that will serve as kind of the front end or the actual user interface to our contract.
* But our JavaScript code has no ability whatsoever to interact with the byte code that has actually been deployed on the Ethereum blockchain.
* ABI acts as transaction layer between JavaScript code and Bytecode

1. **Our First Contract:**

* Remix is an online code editor made specifically for creating and testing solidity contracts.
* Remix has built in tools for auditing and analyzing your code.

*pragma solidity ^0.4.17;*

*contract Inbox {*

*string public message;*

*function Inbox(string initialMessage) public {*

*message = initialMessage;*

*}*

*function setMessage(string newMessage) public {*

*message = newMessage;*

*}*

*function getMessage() public view returns (string) {*

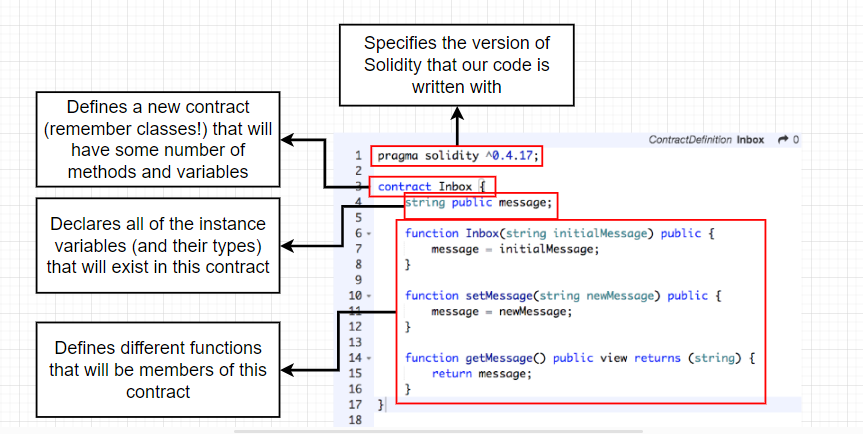
*return message;*

*}*

*}*

1. **Contract Structure:**





* Pragma definition, defines the solidity compiler version we are going to use.

**Storage Variable:**

*string public message;*

* You can think of this storage variable as being sort of like an instance variable.
* This is an instance variable that's going to exist for the life of the contract.
* storage variable is one that will be automatically stored with the contract on the blockchain.
* So that means whenever we change the value of message right here, the value that we assign to message will be automatically stored for all eternity on the Ethereum blockchain.
* And we can always pull our contract back up from the blockchain and look at the value of this variable right here.

**Local Variable:**

* This is in contrast to local variables.
* Local variables are kind of one time created when a contract is executed and then thrown away at the very end, and they are never actually persisted on the blockchain.
* After that variable declaration up here, we then list out three different functions.
* So these are three different functions that will be a part of this contract.

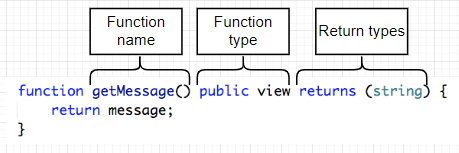
**Constructor function:** *function Inbox(string initialMessage) public {*

* This function is regarded as being a constructor function.
* Constructor functions are automatically called one time when a contract is first created.
* So you can think of this first function right here as being invoked automatically when we deploy our contract to the blockchain.

**Normal Functions:**

* The other two functions inside this file do not have the same name as the contract.
* And so instead these are functions that we can call on the contract after it has been deployed to the blockchain.

**getMessage() Function:**



**Function name:**

* Function starts with the function keyword and then the name of the function with this argument

**Function Type:**

* There's maybe two function types I did not list on here, but in general, you probably will not be
* using those other two very often.
* And I'll leave it up to you to do a little bit of research if you want to find out what they are.
* Honestly, just really stick to these right now.
* We're going to look at all the different function types that we can specify starting at the very top
* here.
* The first one is public.
* If we mark a function as being public, that means that anyone in the world can call this function.
* And what I mean when I say anyone, I mean anyone with an account.
* So anyone with an Ethereum account can access our contract and call a given function.
* Now, this is not really used for implementing any type of security per se.
* So just putting the public word in there doesn't mean that, oh, all of a sudden our contract is accessible
* by everyone and completely open.
* And then if we put a different word, all of a sudden it's 100% secure as opposed to public.
* We can mark a function as private, and that means that only our contracts code can call this function.
* Again, private doesn't really implement any level of security.
* Instead, you should think of public and private as communicating to other engineers as to whether or
* not they should be calling these functions at all.
* A good example of this would be maybe we put together a helper function.
* Maybe we make a helper function that just does some tiny little task that is very specific to our contract.
* And we only ever expect to call that function function from within another function in our contract.
* That would be in a good example of where we might want to use the private keyword.
* Notice that we can only use one of these per function.
* So a function is either public or private.
* In general, we're almost always going to be using the public keyword unless we are making very specifically
* a helper function, in which case we might use the private keyword.
* The next two key words right here really may mean the same exact thing.
* So this term used to be called constant and more recent versions of solidity.
* They have updated the name to be view in documentation.
* You're going to see sometimes constant and sometimes view.
* Again, they mean the same thing and you can use them interchangeably using the view or constant keywords
* inside of the function.
* Declaration specifies whether or not that function modifies any of the contracts data.
* So let's take a look at an example of this.
* Let's go back to our code editor over here.
* So in our get message function right here, you'll notice that we are not attempting to modify the message
* variable in any way, shape or form.
* We're not changing it.
* We're just accessing it because we are just accessing data inside the contract and we are not trying
* to modify or change it whatsoever.
* We are making use of the view keyword right here.
* This specifies to the solidity compiler that this does not modify any of our data and it actually changes
* the way in which we attempt to call this function.
* But we'll see an example of that very shortly.
* On the other hand, we can take a look at the set message function.
* In the case of set message, we clearly accept some argument and then update or modify the value of
* the data within our contract.
* So we are modifying the variable message right here.
* So this function right here would not be appropriate to mark as either view or constant because it is
* attempting to change our contract.
* The next function type is pure.
* Pure is very much like view and constant, but it goes one step further.
* A function marked as pure means that the function will not modify or even access the contracts data.
* In practice, I kind of expect you won't end up using pure that often, except out of some pretty interesting
* edge cases.
* Or I don't know.
* I'm sure you could contrive a good example or two, but in general, I personally have not found many
* instances where I might want to use pure.
* Now, the last keyword that we're going to cover, and this is one that we'll use quite a bit throughout
* this course, is the payable keyword.
* The payable keyword means that whenever some outside source or some outside entity like you or me attempts
* to call a function on a contract payable means that we might attempt to call it and send money to the
* contract.
* At the same time, we're going to end up using payable quite a bit because a lot of the contracts that
* you and I are going to write are going to attempt to accept money from people.
* So we're going to expect that people somehow access our contract and at the same time they do, we might
* require that they pay us some money for the privilege of running our code, so to speak.
* So this has been an overview of some common function types.
* Now you don't really have to memorize all these from the get go.
* I'm just trying to give you a sense of some of the different things that you can expect to see with
* the language.
* As time goes on and as we start to write more and more code, I promise you I will be telling you all
* the different function names that we'll be using.
* Now, last thing I want to point out inside of here is the returns keyword down on the git message function
* returns is used to specify the type of return value that we can expect to see back from a function.
* Returns is only ever going to be used on functions that are marked as view or constant.
* So if a function is meant to modify the state of the contract, this will excuse me.
* A function that modifies the state of the contract will not return any values.
* Now, to be clear, we can attempt to return a value.
* So if I say return message and then returns string.
* You'll notice that we don't really get an error message here, so that doesn't quite match up to what
* I just said about public or about functions that modify the contract data as not being able to return
* value.
* But in just a little bit, I'll show you an example of why we cannot actually return data from a function
* that modifies our contract.
* All right.
* I'm going to clean up that code really quickly and we'll remove the returns right here as well.
* Cool.
* All right.
* So I think that completes a first overview of our inbox contract, but we still need to use the remix
* tool quite a bit to figure out how we can run, deploy and test out this contract.
* So let's pause right here.
* We'll continue the next video and we'll get a better sense of how to use the remix tool.
* So I'll see you in just a minute.