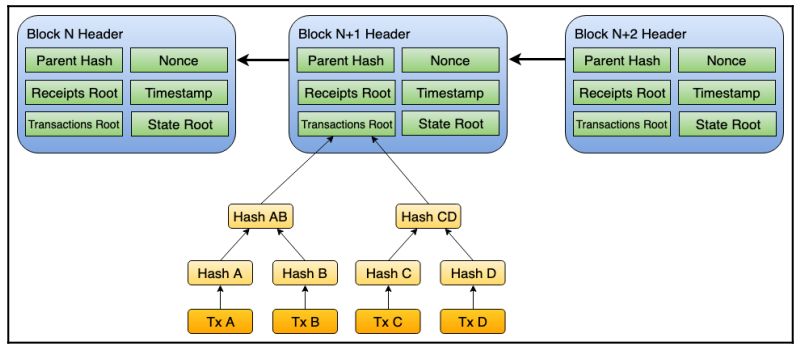
Intro to Blockchain:

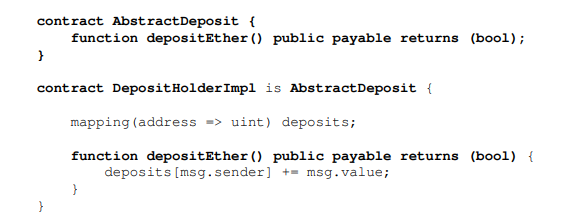
* **Node**: Nodes are devices in the blockchain network, which make the mesh of the blockchain network
  + **Full Node**: maintains the complete data structure of the blockchain ledger and participates in the blockchain network by verifying every block and transaction
  + **Lightweight Node**: It verifies the transaction using a method called Simple Payment Verification (SPV), SPV allows a node to verify whether a transaction has been included in the blockchain or not
* **Miners**: Owners of the nodes willingly contribute their computing resources or hard disk space to store and validate transactions.
* **Blocks**: The blockchain maintains a series of blocks. Each block is linked together with the last generated block. A block contains multiple transactions. Each block is generated in aprox 15 sec
  + **Genesis Block**: The very first block in the blockchain is called the genesis block
  + 
    - Parent Hash: points to the parent block
    - Nonce: number that is found by the miner to solve the cryptographic puzzle
    - Receipts Toot: Keccak 256-bit hash populated with the receipts of each transaction
    - Timestamp: block found and added t the blockchain
    - Transactions Root: Keccak 256-bit hash populated with each transaction
    - State root: Keccak 256-bit hash after all transactions are executed.
* **EVM:** runtime environment for smart contractswhich supports multiple scripting.
* **Ether currency:** Ethereum has its own cryptocurrency called ether, Ether is a fungible coin, which means that a coin can be subdivided into smaller units. Ether can be subdivided into max 18 decimal places(smallest value is called wei), it acts like crypto fuel for Ethereum network.
* **Gas**: Gas is the fuel of the Ethereum blockchain network. Gas is always paid in ether. Gas is the transaction fee paid to miners for executing and adding your transaction into the Ethereum blockchain
  + **Gas limit**: Each block has gas limit (8 million gas per block) e total gas consumed by the transactions included in the block cannot exceed the 8 million gas limit per block. Where each transaction need 21000 gas
  + **Gas Price**: The gas price is the price per gas unit you are willing to pay for executing your transaction.
  + **Formula**: Text

    Description automatically generated
* **Ethereum Accounts:**
  + Externally Owned Accounts (EOA): Ethereum wallet can hold one or more EOA. The combination of a private key and a public key is called an account or EOA. An EOA is controlled by its private key to initiate any transaction with it. An EOA owner can send ETH/ERC20 tokens to other EOA or contract addresses. They can also initiate a transaction on a deployed contract with EOA.
  + **Contract accounts:** Contracts only have a public address, and their code logic controls the flow of the funds and other states of the contract. You need an EOA account to create a new contract or to interact with an existing contract. Contract accounts have a Solidity code that defines their behavior. Public key is generated using public address of creating account + nonce
* **Transaction fields:**
  + **From**
  + **To**
  + **Value**
  + **Gas limit**
  + **Gas Price**
  + **Nonce**: starts with 0 (zero) for an EOA that initiates the **transaction**, it keeps increasing by one for every transaction an EOA initiates. It specifies the order execution of transactions.
  + **Data**: hexadecimal data you want to transfer along with a transaction
  + **Transaction** **hash**: A transaction hash is assigned to every transaction when it contains valid transaction fields and is broadcast to the Ethereum blockchain, it is not assigned by user
* **Testnets:** you can deploy and test your contract's functionality

Getting Started with solidity:

* **Balance**: uint256 balanceInWei = msg.sender.balance;
* **call and delegatecall functions:**
  + **call**: call a function of another contract or call a library function
  + **delegatecall**: only difference is that it takes the code of the target/called contract and executes it in current caller contract.
  + **Staticcall**: returns a Boolean condition along with the return data
* Understanding Variables:
  + **Memory**: All the function parameters and return types are by default set to the memory data location.
  + **Storage**: the local variables you use in functions are set to storage
* Global special variables and functions
  + **block**
    - block.coinbase: provides the miner's EOA address of the current block
    - block.difficulty: This provides the current difficulty level of the network as a uint256 value, which is used by the miners to solve the Proof of Work (PoW) puzzle.
    - block.gaslimit: This provides the total gas limit of the current block as a uint256 value
    - block.number: This provides the current block number
    - block.timestamp: This provides the timestamp when the block is generated
  + **msg**: provides the information related to the transaction sender and some transaction data.
    - msg.data: This provides the full call data, represented as bytes, which is sent by the transaction initiator
    - msg.sender:
    - msg.sig: This provides the first 4 bytes of the function signature.
    - msg.value: This provides the amount of wei sent along with the transaction.
  + **now:** provides the current block timestamp as a uint256 value. Don’t relay on now or block.timestamp because miners can manipulate that.
  + **tx:** provides transaction related information.
    - **tx.**gasprice: provide the gas price of the transaction set by transaction initiator.
    - **tx.**origin: This provides the original sender of the transaction from the full call chain as an address will always return an address of an EOA(don’t ues tx.origin)
  + **blockhas (uint blockNumber) returns (bytes32):** hash of the given block number as bytes32. This function will only work for the most recent 256 blocks, excluding the current block.
  + **gasleft () returns (uint 256):** provides the remaining gas left in the transaction
* **Cryptographic functions:**
  + keccak256(bytes memory) returns (bytes32): SHA3 computes the Ethereum-SHA-3 (Keccak-256) hash of the tightly packed arguments and returns it as bytes32.
  + sha256(bytes memory) returns (bytes32): computes the SHA-256 hash of the input bytes and returns it as bytes32
  + ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address): Recovers the address associated with the public key from an elliptic curve signature or returns 0 on error

Control Structures and Contracts

* **two ways to create a new contract on a blockchain:**
  + Deploy the contract from outside of the blockchain using Remix
  + Create new contracts from an already deployed contracts
* **Function visibility:**
  + public: accessible from both inside and outside
  + external: only accessible from outside of the contract (use **this** to use)
  + internal: only accessible internally within the contract
  + private: only accessible internally within the same contract
* **types of Solidity functions:**
  + **view:**
    - you can’t
      * writing to a contract state variable
      * calling any function within the current contract
      * Emitting any events
      * Creating new contracts
      * Using the selfdestruct call
      * Sending ether to another contract
  + **Pure**
    - You can’t
      * Reading any state variable defined in the contract
      * Using address(this).balance to read the ether balance of a contract
      * Reading any of the members of globally available variables such as msg, tx, and block
      * Calling any function that isn't marked as a pure function
    - You can
      * An arithmetic calculation without reading any contract state variables
      * Use msg.data and msg.sig.
      * Use the globally available function that does not make state changes, for example, the keccak256() and ripemd160() functions.
* **Abstract contracts:** there are only a few functions that don't have the function body defined, you can inherit the contract and provide the definition to each function that's declared in the abstract contract. 
* **Interface:** same as abstract contracts only difference is they cant have any constructor, all function should’ve external visibility and can not have state variables, interface cant inherit from ant other contractor interfaceGraphical user interface, text, application, email

  Description automatically generated
* **Creating custom reusable libraries:** deployed only once and their code is reused in the calling contractsText

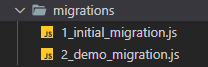
  Description automatically generatedto use the libraryText

  Description automatically generated

**Learning MetaMask and Remix:**

* **three different categories of cryptocurrency wallets:**
  + **Hot wallets:** three different categories of cryptocurrency wallets eg: MetaMask
  + **Cold wallets:** used to keep your cryptocurrencies offline and are not easily accessible on your electronic device for transactions eg: printed private key on paper
  + **Hardware wallets:** a form of cold wallet, a hardware wallets store the private keys of the wallet in a special device build to keep it secure
* MetaMask private key: shove decline gentle subject local donor dolphin voyage place claim crane crystal
* This **secret seed phrase** aka **mnemonic** is generated using a standard algorithm called BIP-44, also known as Hierarchical Deterministic (HD) wallets (BIP is short for Bitcoin Improvement Proposals).
* **Different Ethereum networks:**
  + Mainnet: each ETH has a real world economic value
  + Testnet: no value
    - Ropsten Test Network
    - Kovan Test Network
    - Rinkeby Test Network
  + Custom: no value
    - Localhost
    - Custom RPC connection URl
* **MetaMask plugin:**
  + Send ether
  + Create new EOA
  + Add ant ERC20 token and receive
  + Import any EOA account

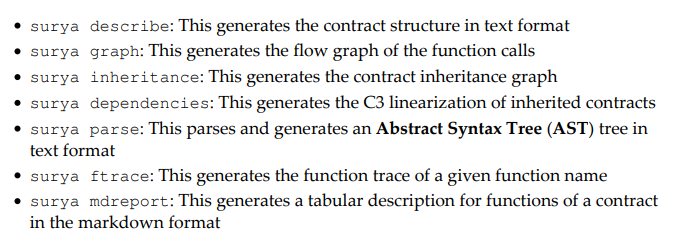
Using Ganache and truffle framework

* How to use truffle:
* First make a file where you want to make project
* truffle init
* truffle compile
* truffle migrate
* truffle migrate –rest(if you did changes after truffle migrate)
* Things to remember
  + In migration folder make in sequence files if ones a parent of other contract so parent sol file should complied first. 
  + To connect Ganache copy ip of ganacheGraphical user interface, website

    Description automatically generatedand paste at truffle-configText

    Description automatically generated

Taking Advantage of Code Quality Tools

* Surya: utility tool that is used to generate a number of graphical and other reports, can generate inheritance and function call graphs. 
* Linters: utility tools that analyze the given source code and report programming errors, bugs, stylistic errors and to improve the quality of their Solidity contracts.
  + solhint: provides security and style guideline-specific validations.
  + solium/ethlint: analyzes the Solidity code provided and reports the style guideline and security issues. The ethlint linter also supports an option to fix the reported issues.
* solidity-coverage tool: to determine which part of the code is covered and tested by the different test cases and which part is not tested. Once developers write the test cases for their Solidity project, they can use the coverage tools to find their code coverage.

ERC20 Token Standard

* Different ERC standards are present in the Ethereum blockchain.
* ERC20 is a token standard that's used to create the fungible tokens on Ethereum blockchain—just like bitcoin and ether are coins and have economic values.
* Using ERC20 you can mint as many tokens of a specific type.
* This standard allows anyone to easily create their own token on a blockchain
* **Coins**:  refer to any cryptocurrency that has a standalone, independent blockchain — like Bitcoin. Have a standalone blockchain.
* **Tokens**: Tokens are a unique outlay of broader smart contracts platforms like Ethereum that enable users to create, issue, and manage tokens that are derivatives of the primary blockchain. Live on another blockchain
* **ERC20**: is the token standard API that defines functions so that they can access the status of the token, the token's details, and account balances
* **require(ERC20.ANYFUNCTION());** use this argument if anything goes wrong it will revert the transaction.
* **Contract State Variables**:
  + **mapping(address => uint256) internal balances:** stores the number of tokens each address has. Since it is internal so all contracts implementing the ERC20 implementation can also read or change the balances mapping entry.
  + **mapping (address => mapping (address => uint256)) internal allowed:** My daddys account balance is 200$ and he gave me approval to use 50$ from his account**,** only I have access to get 50$ from his account because I am Approved person who can withdraw that amount aka allowanceDiagram

    Description automatically generated

**spenders** are allowed to spend **allowance** from **from** address

**Table

Description automatically generated**

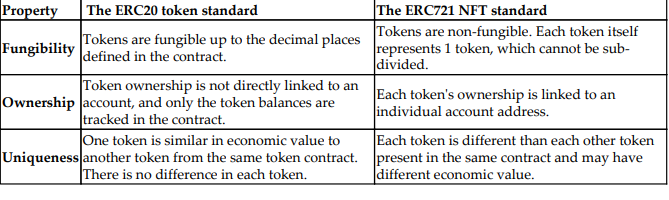
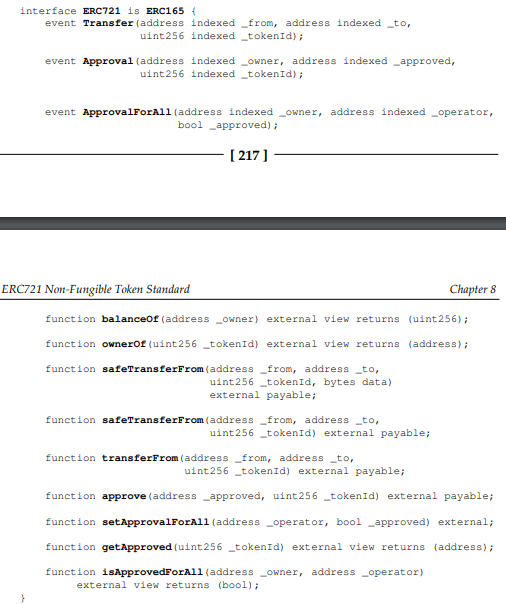
* + **uint256 internal totalSupply\_;** This stores the sum of all of the tokens that exist. This variable gives you an idea of how many of these types of tokens have been generated in total and are held by EOA addresses.
  + **function transfer(address to, uint256 value) external returns (bool success);** used to transfer the tokens from the owner of the token to some other address, the transfer() function must emit the Transfer event on a successful transferal of tokens
    - **address(0):** used when performing token minting or burning. If tokens are sent to address(0), they would be burned; therefore, this check is in place to prevent a user from unintentionally burning their token. It also verifies that the \_value is less than or equal to the balance of msg.sender.
  + **function approve(address \_spender, uint256 \_value) public returns (bool):** to approve the transfer of a certain amount to another account. Function transferForm is used to transfer allowance. Text, Word

    Description automatically generated
  + **function transferFrom(address from, address to, uint256 value) external returns (bool success);** 
    - **from:** to deduct the tokens from
    - **to:** to transfer the tokensText

      Description automatically generated
  + **function allowance(address owner, address spender) external view returns (uint256 remaining);** function returns the remaining balance of approved number of tokens.
    - owner: owner of the token who previously called the approve() function
    - spender: spender, who is allowed to withdraw tokens from the owner's addressText

      Description automatically generated with low confidence

**ERC721 Non-Fungible Token Standard**

* ERC721 standard is a NFT standard, in which an NFT is nondivisible and non-fungible. 
* All functions:
* 
* Diagram

  Description automatically generated