Blockchain Bootcamp Day 8(30 Minutes)

* Sometime you want a smart contract to act like eoa. Send money to contract.
* Without interacting with a special function such as sending money via meta mask(which will send funds without interacting with special function in smart contract.)
* Fallback function is a catch all, it will occur in cases when a function is called that doesn’t exist.
* It would be just function() external { fallback block code }
* Willl be used when using wallets such as meta mask.
* It’s triggered automatically, argument data is in the msg.data.
* View and pure functions
* 🡪 Two types functions that is responsible for viewing state, or interacting with state
* -> Use pure keyword to create a pure function.
* -> Pure function that does not interactive with storage variable or class variables.
* A pure function can call another pure function.
* Transactions needs to be mined(writing functions)
* Don’t need to mined a transaction to read anything.
* Calls are against local blockchain node.(reading functions)
* You are basically paying gas to yourself so it’s free.
* View function are reading from state, also can read from other view functions, and not modifying state. Or referred as constant functions.
* Pure functions don’t read or write to state. Can only call on other pure functions.
* In older version using constant instead of view/pure
* Function Visibility
* Public functions can be called inside, outside or a smart contracts that inherits or extends that smart contract.
* Private function can only be called within smart contract. Also cannot use it in a inherited smart contract.
* External functions, can be called from other contracts, and can be called externally. BUT cannot be called within the smart contract.
* Internal functions, can be called from the contract itself or from derived contracts. Cant’ be invoked by transaction.
* A constructor function can only be called during deployment
* Cannot be called afterwards.
* Is either public or internal.
* Fallback function is a function without a name.
* Fallback functions is called in two circumstance, send ether without a function call, or cannot find function.
* Fallback functions can only be external
* If sending ether without calling function and without fallback function, it will throw an exception.
* You cannot completely avoid receiving ether
* Three ways to receive ether in smart contract.
* If you selfdestruct another smart contract, and put your smart contract as beneficiary it will receive it no matter.
* If you set your address to a miner
* If ether is send to smart contract before it’s deployed.
* NEVER MATCH BALANCE OF VARIABLE TO TOTAL BALANCE OF SMART CONTRACT
* Forcefully prevent contract execution if called with contract data.(require(ms.data.length === 0)
* View/pure is only against blockchain node.

Modifier, inheritance, and Importing of files

* You would add a modifier after adding your access level for your function. For example -> function testFunc() <access level> <modifier> {}
* Is keyword will extend functionality of a smart contract or inherit a smart contract or multiple smart contracts
* Modifiers are in derived smart contracts
* Can only import smart contracts to inherit via import “./’DerivedSmartContract”
* The last smart contract set is the most derived smart contact.
* Only one smart contract gets deployed when it’s inheriting multiple smart contracts, not all smasrt contracts in use.
* Modifiers are a great way to check pre-conditions
* Don’t use modifiers too much, only enough to make sure source code is easy to read.
* Can import everything, all members or specific member in a file.
* Can abstract functionality way using inheritance

Events and Return Variables

* Cannot have return variables in solidity.
* Can have it inside smart contract, but can’t return it to address that initiated transaction.
* It will work in a form of decoded output in the javascript vm on remix, BUT WILL NOT WORK ON THE ETHEREUM MAINNET or test mainnet.
* A screenshot of a computer

  Description automatically generated with medium confidence
* Events will return values on the mainnet. Events are defined by event <event name>(<event arguments>);
* Events are emitted using the emit keyword <event name>(<event arguments>)