Blockchain Bootcamp Day 9(30 minutes)

Events and Return Variables

* Events and return values.
* EVM has logging functionality it is behind events. Stored on logs.
* Indexed keyword, if a event parameter as indexed. They are being searched, and can be searched later on the site chain.
* The events at most can hold only 3 arguments
* Can be used as a cheap data storage.
* Much cheaper than storing it string, held in site chain.
* Events actually inform external users that something happened in blockchain, which would be return values.
* It can also be a trigger
* Storing data is extremely expensive on Ethereum blockchain
* Store data in event logs if not necessary
* The strings can be stored in a form of a event.
* Events in dApps
* Applications can subscribe and listen to these events through rpc interface of Ethereum client.
* Events are inheritable members of contracts.
* Log and it’s event data are not accessible from within contracts
* ,,indexed”events parameters can be searched for later.”
* When communicating with smart contract outside of solidity, store don blockchain.
* Somehow start transactions using decentralized applications, (apps outside of blockchain.)
* Event values cannot be used in solidity, but can be used by external applications. External application can subscribe to gprc events to get those values.
* Events are used to return values, data store or trigger.
* Events cannot be retrieve from within smart contract.
* Event arguments can be indexed, can be searched for.
* Events are cheap.

ABI Array, Debugging and Gas.

* ABI(Application Binary Interface, generated when solidity file gets generated.)
* In ABI it will refer to view/pure functions as constants.
* For all functions including getters and settings they will reference a function hash.
* Use function hashes to debug.
* When using a setter function, in the input property of the transaction details, it will display multiple zeros followed by argument.

Function and Opcodes

* Deployment(Smart contract creation-> high level code, can write assembly in solidity as well)). In bytecode won’t see plain text any more.
* Interaction happens low-level when you do a interaction.
* Calculating hash, and sending right hash.
* ABI array will be max to the address.
* Function has will be first 4 bytes of the keccask256 hash of function signature.
* Interact with getter function with parameters such as -> **function someFunction(uint \_myUint1, address \_someAddr).** First 4 bytes of keccak256 hash of function signature. Bytes4(keccak256(,,someFunction(uint256,address”))
* Interact with setter.
* The function signature parameters are padded by 32 bytes.
* The debugger, run transaction and see transaction in debugger.
* Show gas costs for opcodes, stack, and memory.
* Debugging relatively good for prototyping.
* Gas -> detach execution cost from ether.
* Gas price isn’t always the same, it is based on how congested the network is.
* If there is gas left, it will refund rest of gas.
* Opcodes in address is taking care of the first 4 bytes of the data field and match function signature.
* Functions are encoded are hashed, they are no plain text functions in smart contact.
* More complex a function, it uses more gas.