Smart Contracts - Key Takeaways

Below you will find a number of key points from this course. Defined terms are underlined.

Week One: Smart Contract Basics

Contract in the Ethereum Blockchain has:

1. Pragma directive
2. Name of the contract
3. Data or the state variables that define the state of the contract
4. Collection of functions to carry out the intent of a smart contract
5. Other items we will discuss in later lessons.

Remix is a web Integrated Development Environment (IDE) that creates, deploys, executes and explores the working of smart contracts on the Ethereum blockchain.

Two very simple smart contracts are:

1. Greeter
2. One integer storage: SimpleStorage

The 3 steps to developing a smart contract is:

1. Design
2. Code
3. Test

Remix Solidity compiler generates several artifacts such as:

1. Name of the contract
2. Bytecode executed for the contract “creation” on the EVM
3. ABI: Application Binary Interface, details functions, parameters and return value
4. Web3 deploy module that provides the script code for invoking the smart contract from a web application
5. Gas estimates for the execution of the function, and
6. Actual runtime bytecode of the smart contract

Smart contracts can be deployed from:

1. Remix IDE
2. Another smart contract
3. A command line interface
4. Another high level application
5. A web application.

Complete process of deployment using Remix IDE:

1. You enter the smart contract code in the Remix IDE and compile.
2. Remix generates several artifacts as discussed earlier and as shown in the picture.
3. For the ease of deployment, remix provides us with the web3 deployment script which contains byte code, Application Binary Interface (ABI), account details.
4. To deploy the smart contract we could just execute this script.
5. Once the deployment is done, the address is generated by hashing creator’s account number and nonce.
6. To interact with the smart contract we will use the smart contract address, the ABI definition, and the function hashes.

Week Two: Solidity

Detailed structure of a smart contract:

- Data or state variables
- Functions : There are several types of functions allowed,
  - Constructor (default or user specified; only one, meaning it cannot be overloaded)
  - Fallback function (This is a powerful feature of an anonymous function that we will discuss in the best practices module later in this course.)
  - View functions
  - Pure functions (no state change, it computes and returns a value; example: math functions)
  - Public functions (accessible from outside thru transactions, state changes recorded on the bc)
  - Private functions (accessible only with the current contract)
  - Internal functions (accessible inside current contract and inherited contract)
  - External functions (can be accessed only from outside the smart contract)
- User defined types in struct and enums
- Modifiers
- Events

"function" is a keyword at the beginning of all functions

Parameters are any number of pairs {type, identifier} . E.g. uint count

Return parameters: return values can be specified as pair {type, identifier} or just {type}. When only type is specified it has to be explicitly returned using a return statement.

If type and identifier are specified in the returns statement, any state change that happens to the identifier within the function is automatically returned.

Any number of values can be returned unlike common programming languages that allow only one return value. For example, multiple variables, age, gender can be assigned the return values from a function

1 Ether is 10^18 Wei.

Solidity is a high-level language that is a combination of Javascript, Java and C

The Bidder smart contract design contains three items:

1. Name of the contract
2. The data/states
3. The functions

“**Address**” is a special Solidity defined complex data type that can hold a 20 byte Ethereum address. Address is the base of a smart contract.

“**Mapping**” is a very versatile data structure that is similar to a <key, value> store. It can be thought of as a hash table.

“**Message**” is a complex data types specific to smart contract. It represents the call that can be used to invoke a function of a smart contract.

“Struct” is composite data type of a group of related data that can be referenced by a single meaningful collective name.

“enum” or enumerated data type allows for user-defined data types with limited set of meaningful values.

The main intent of a smart contract transaction is to execute a function.

You can think of a modifier as a gatekeeper protecting a function since it can change the behavior of a function

Week Three: Putting it all Together

Solidity features a function revert() that results in state-reverting exception.
This exception handling will undo all changes made to the state in the current call, and reverses the transaction and also flags an error to the caller.

An event feature is used to indicate to a client application (user interface or a transaction monitor) that a significant milestone has been reached.

A listener code can be used to:

1. Track transactions
2. Receive results through parameters of the events
3. Initiate a pull request to receive information from the smart contract.

The Chairperson is the creator of the smart contract. He/she will be the only person who can register the voters.
List of functions:

1. Constructor is a function that is called to create the smart contract. In Solidity, unlike a regular Object Oriented language, there can be only one constructor. Also, the constructor has the same name as the contract. The sender of the message invoking the constructor is the chairperson.
2. Register is a function to register the voter. Only the chairperson can register a voter. The Sender of the message for registration has to be the chairperson.
3. Vote is a function in which voters including the chairperson can vote for a proposal.
4. Winningproposal is the final function that determines the winning proposal and can be called by client applications.

Stage has four distinct stages: Init, reg, state, and done. The stage is initialized to init at the time of the deployment of the smart contract. Then, in the constructor, the init is changed to reg stage. After the registration period is over, the stage changes to vote. After the voting duration elapses, the stage is set to Done.

Week Four: Best Practices

Blockchain is not a solution for all applications. Make sure your application requirements need blockchain features

A blockchain solution is most suitable for applications with these characteristics:

1. Decentralized problems, meaning participants hold the assets and are not co-located,
2. involve peer-peer transactions without intermediaries,
3. operate beyond the boundaries of trust among unknown peers,
4. require validation, verification and recording on an universally timestamped immutable ledger
5. Autonomous operations guided by rules and policies

Make sure you need a smart contract on blockchain for your application.

Keep the smart contract code simple, coherent, and auditable.

Blockchain is not a data repository. Keep only the necessary data in the smart contract.

Make sure you use the integer arithmetic for most of your computational needs.
Maintain a standard order for the different function types within a smart contract, according to their visibility, as specified in Solidity docs.

The recommended order of functions within a smart contract are:
1. constructor
2. fallback function
3. external
4. public
5. internal
6. private

Within a grouping, place the constant functions last.

Use Solidity defined “payable” modifier when sending value

Pay attention to Order of Statements inside a function

Use Modifier Declarations for implementing rules

Using Events in Smart Contract

Use Secure hashing for protecting data

Pay Attention to Remix Console, Compiler, Transaction Log, and Static Analysis Details