Solidity Tutorial # 01

Blockchain
BS/MS Fall 2023

Topics To Be Covered

- Introduction to Ethereum
- Overview of Etherscan.io
- Smart Contracts
- Ethereum Virtual Machine
- Introduction to Solidity
- Layout & Structure of Solidity

Introduction To Ethereum

- Ethereum is a decentralized, open-source blockchain with smart contract functionality.
- Ether is the native cryptocurrency of the platform.
- The state is made up of objects called "accounts”. There are two types of accounts.
- Externally owned accounts: controlled by private keys and with only Ether balance;
- Smart contract account: that has executable code

Overview of Etherscan.io

- The blockchain explorer of Ethereum blockchain.

  Link: https://etherscan.io/

Smart Contracts

- A program that runs on the Ethereum blockchain. It's a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain.
- Not controlled by a user.
- Deployed to the network and run as programmed.
- User interacts with a smart contract by submitting transactions that execute a function defined on the smart contract.

Smart Contracts Contd.

- To do a transection you need to pay Gas in the same way you need to pay gas for a simple ETH transfer.
- Gas refers to the fee required to conduct a transaction on Ethereum successfully.
- Languages used: **Solidity and Viper**

The Ethereum Virtual Machine (EVM)

- Runtime environment for smart contracts.
- The EVM execution environment works in the following way:
  - Inheritance (you can extend other contracts)
  - Write code in Solidity (or another front-end language)
  - Compile to EVM bytecode (recent projects use WASM or BPF bytecode)
  - Miners use the EVM to execute contract bytecode in response to a Tx
- Every EVM instruction costs gas i-e upon creation, paid by the originator of the transaction.

Introduction To Solidity

- Object-oriented, high-level language for implementing smart contracts.
- Statically typed (the type of a variable is known at compile time).
- Supports:
  - Inheritance
  - Libraries
  - Complex user-defined types.
- *We will use Remix for deploying solidity small contracts.*

  Link: https://remix.ethereum.org/

Layout and Structure of a Solidity Smart Contract

- Contracts in Solidity are similar to classes in object-oriented languages.
- Each contract can contain declarations of:
  - Pragma directives
  - State Variables
  - Functions
  - Function Modifiers
  - Events
  - Errors
  - Struct
  - Enum

Contract

- Every contract defines its own type. You can implicitly convert contracts to contracts they inherit from.
- Contracts can be explicitly converted to and from the `address` type.
- If you declare a local variable of contract type (`MyContract c`), you can call functions on that contract. Take care to assign it from somewhere that is the same contract type.
- Contracts do not support any operators.
- The members of contract types are the external functions of the contract including any state variables marked as `public`.

Contract Example

```
pragma solidity >=0.4.22 <0.9.0;

contract Oracle {

    struct Request {

        bytes data;

        function(uint) external callback;

     }
```

Pragma

- Used to specify the compiler version for specific solidty file
- You need to add pragma in your all files as its specific to a file
- The pragma version is used as follows:

```
pragma solidity ^0.5.2;

pragma solidity >=0.4.22 <0.9.0;
```

State Variable Types

- State Variables:
  - Variables whose values are permanently stored in a contract storage.
- Local Variables:
  - Variables whose values are present till function is executing.
- Global Variables:
  - Used globally and give information about transection and Blockchain properties
  - e.g gasleft() returns the amount of gas left

More on State Variables

- Variables whose values are permanently stored in contract storage.

```
pragma solidity >=0.4.0 <0.9.0;

contract SimpleStorage {

    uint storedData; // State variable

}
```

State Variable Visibility

- **`Public:`** Can be accessed by the contract and also other contracts.
- **`Internal:`** Can only be accessed within the contract and in its subclasses.
- **`Private:`** Variable can only be accessed within the contract its defined.

Value Types

Booleans

`bool`: The possible values are constants `true` and `false`.

Operators:

- `!` (logical negation)
- `&&` (logical conjunction, “and”)
- `||` (logical disjunction, “or”)
- `==` (equality)
- `!=` (inequality)

Value Types

Integers

`int` / `uint`: Signed and unsigned integers of various sizes.

Operators:

- Comparisons: `<=`, `<`, `==`, `!=`, `>=`, `>` (evaluate to `bool`)
- Bit operators: `&`, `|`, `^` (bitwise exclusive or), `~` (bitwise negation)
- Shift operators: `<<` (left shift), `>>` (right shift)
- Arithmetic operators: `+`, `-`, unary `-` (only for signed integers), `*`, `/`, `%` (modulo), `**` (exponentiation)

Value Types

Address

The address type comes in two flavours, which are largely identical:

- `address`: Holds a 20 byte value (size of an Ethereum address).
- `address payable`: Same as `address`, but with the additional members `transfer` and `send`.

Members of Address Types Example

```
address payable x = payable(0x123);

address myAddress = address(this);

if (x.balance < 10 && myAddress.balance >= 10)
x.transfer(10);
```

Value Types

Fixed Size Byte Arrays

- The value types `bytes1`, `bytes2`, `bytes3`, …, `bytes32` hold a sequence of bytes from one to up to 32.
- Prior to version 0.8.0, `byte` used to be an alias for `bytes1`.
- contract ByteArrays { bytes32 y = 0xa5b9…; // y.length == 32 }

```
enum FreshJuiceSize{ SMALL, MEDIUM, LARGE }
```

Value Types

Enum

- Enums are one way to create a user-defined data type in Solidity.

- Enums retrict a variable to have one value of few predefined values e.g
  `enum` size {small, medium, large} size only can have value from these three

Enum this slide can be removed upto you

```
ppragma solidity ^0.8.8;
contract test {
    enum ActionChoices { GoLeft, GoRight, GoStraight, SitStill }
    ActionChoices choice;
    ActionChoices constant defaultChoice = ActionChoices.GoStraight;
    function setGoStraight() public {
        choice = ActionChoices.GoStraight;
    }
    function getLargestValue() public pure returns (ActionChoices) {
        return type(ActionChoices).max;
    }}
```

Functions

- Usually defined inside a contract, but they can also be defined outside of contracts.
- Function Calls can happen internally or externally

```
pragma solidity >=0.7.1 <0.9.0;

contract SimpleAuction {

    function bid() public payable { // Function }

}

// Helper function defined outside of a contract

function helper(uint x) pure returns (uint) {

    return x * 2; }
```

Function Visibility

- **`External:`** Can be accessed from other contracts using the transections.
- **`Public:`** Part of the contract interface and can be either called internally or via message calls.
- **`Internal:`** Internal functions can only be accessed from within the current contract or contracts deriving from it.
- **`Private:`** Private functions are like internal ones but they are not visible in derived contracts.

Function Modifiers

- Used to change the behaviour of functions.
- We can use a modifier to check a condition prior to executing the function.

```
pragma solidity >=0.4.22 <0.9.0;
contract Purchase {
    address public seller;
    modifier onlySeller() { // Modifier
        require(
            msg.sender == seller,
            "Only seller can call this."
        );
        _;
    }
    function abort() public view onlySeller { // Modifier usage
        // ...
    }
}
```