Smart Contract

Progrmming

Smart Contracts

- Smart contracts are blocks of code that reside on the blockchain.
- It is like an Ethereum account but there is a critical difference between an external account and a smart contract.
- Unlike a smart contract, an external account can connect to multiple Ethereum networks (Goerli testnet, mainnet, etc.) whereas a smart contract is only specific to one individual network (the network it is deployed on).
- When a smart contract is deployed, it creates an instance (contract account) on the network.
- One can create multiple instances of a smart contract on the network or multiple networks.
- Deployment of a smart contract is done by sending a transaction to the network with bytecode.

Deploying To A Local Network

- An emulator can be used to deploy a smart contract on a local network eg. Ganache-cli.
- Ganche takes care of everything and the user doesn’t have to worry about the security and the gas amount required for transactions since everything is happening on a local test network.
- All one has to do is pass the ganache provider as an argument to the web3 instance(web3 facilitates the connection between the blockchain network and the js application).

Deploying To Actual Ethereum Network

- Before deploying a smart contract to an actual Ethereum network make sure the account has some ether in it.
- Deploying a contract is like sending a transaction and it needs some gas amount to process.
- Unlike deploying on a local network, transactions will take some time to complete (anywhere between 15 seconds to 5 minutes).
- Web3 is used to interact with the network the same way it is done in local deployment except customize the provider that will be passed into the web3 instance.
- Instead of creating our own node that connects to the Ethereum network, one can use a developer platform with RPC endpoints.
- With one of these accounts, you have an API key that gives access to their blockchain nodes that are already hosted on the Ethereum network.

Solidity – Constructors

- A constructor is a special method in any object-oriented programming language which gets called whenever an object of a class is initialized.
- The concept is totally different in case of Solidity, Solidity provides a constructor declaration inside the smart contract and it invokes only once when the contract is deployed and is used to initialize the contract state.
- A default constructor is created by the compiler if there is no explicitly defined constructor.

Constructor

```
// Solidity program to demonstrate
// creating a constructor
pragma solidity ^0.5.0;

// Creating a contract
contract constructorExample {

    // Declaring state variable
    string str;

    // Creating a constructor
    // to set value of 'str'
    constructor() public {
        str = “SVNIT";
    }

    // Defining function to
    // return the value of 'str'
    function getValue(
    ) public view returns (
      string memory) {
        return str;
    }
}
```

Constructor

- Constructors are very useful in a smart contract, a parameter value can be defined at the run time.
- Constructor overloading is not supported in Solidity, it only allows one constructor at a time.

While Loop

This is the most basic loop in solidity, Its purpose is to execute a statement or block of statements repeatedly as far as the condition is true and once the condition becomes false the loop terminates.

```
// Solidity program to
// demonstrate the use
// of 'While loop'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring a dynamic array
    uint[] data;

    // Declaring state variable
    uint8 j = 0;

    // Defining a function to
    // demonstrate While loop'
    function loop(
    ) public returns(uint[] memory){
    while(j < 5) {
        j++;
        data.push(j);
     }
      return data;
    }
}
```

Do-While Loop

This loop is very similar to while loop except that there is a condition check which happens at the end of loop i.e. the loop will always execute at least one time even if the condition is false.

```
// Solidity program to
// demonstrate the use of
// 'Do-While loop'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring a dynamic array
    uint[] data;

    // Declaring state variable
    uint8 j = 0;

    // Defining function to demonstrate
    // 'Do-While loop'
    function loop(
    ) public returns(uint[] memory){
    do{
        j++;
        data.push(j);
     }while(j < 5) ;
      return data;
    }
}
```

For Loop

This is the most compact way of looping. It takes three arguments separated by a semi-colon to run. The first one is ‘loop initialization’ where the iterator is initialized with starting value, this statement is executed before the loop starts. Second is ‘test statement’ which checks whether the condition is true or not, if the condition is true the loop executes else terminates. The third one is the ‘iteration statement’ where the iterator is increased or decreased.

```
// Solidity program to
// demonstrate the use
// of 'For loop'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring a dynamic array
    uint[] data;

    // Defining a function
    // to demonstrate 'For loop'
    function loop(
    ) public returns(uint[] memory){
    for(uint i=0; i<5; i++){
        data.push(i);
     }
      return data;
    }
}
```

Rules For Naming Variables

- A variable name should not match with reserved keywords.
- Variable names must start with a letter or an underscore (_), and may contain letters from “a to z” or “A to Z” or digits from “0 to 9”.
- The name of variables are case sensitive

```
<type> <access modifier> <variable name> ;
```

Global Variables

Variable	Return value
blockhash(uint blockNumber) returns (bytes32)	Hash of a given block, works for only 256 most recent transactions excluding current blocks
block.coinbase (address payable)	Address of current blocks miner
block.difficulty (uint)	The difficulty of the current block
block.gaslimit (uint)	Gaslimit of the current block
block.number (uint)	Block number of the current block
block.timestamp (uint)	The timestamp of the current block as seconds since Unix epoch
gasleft() returns (uint256)	Amount of gas left
msg.data (bytes calldata)	Complete call data of block
msg.sender (address payable)	The sender of message i.e. current caller
msg.sig (bytes4)	First four bytes of call data i.e. function identifier
msg.value (uint)	Amount of Wei sent with a message
now (uint)	The timestamp of the current block
gasleft() returns (uint256)	Amount of gas left
tx.gasprice (uint)	Price of gas for the transaction
tx.origin (address payable)	Transaction sender

Features of Smart Contracts

- The following are some essential characteristics of a smart contract:
  - **Distributed:** Everyone on the network is guaranteed to have a copy of all the conditions of the smart contract and they cannot be changed by one of the parties. A smart contract is replicated and distributed by all the nodes connected to the network.
  - **Deterministic:** Smart contracts can only perform functions for which they are designed only when the required conditions are met. The final outcome will not vary, no matter who executes the smart contract.
  - **Immutable:** Once deployed smart contract cannot be changed, it can only be removed as long as the functionality is implemented previously.
  - **Autonomy:** There is no third party involved. The contract is made by you and shared between the parties. No intermediaries are involved which minimizes bullying and grants full authority to the dealing parties. Also, the smart contract is maintained and executed by all the nodes on the network, thus removing all the controlling power from any one party's hand.
  - **Customizable:** Smart contracts have the ability for modification or we can say customization before being launched to do what the user wants it to do.
  - **Transparent:** Smart contracts are always stored on a public distributed ledger called blockchain due to which the code is visible to everyone, whether or not they are participants in the smart contract.
  - **Trustless:** These are not required by third parties to verify the integrity of the process or to check whether the required conditions are met.
  - **Self-verifying:** These are self-verifying due to automated possibilities.
  - **Self-enforcing:** These are self-enforcing when the conditions and rules are met at all stages.

Applications of Smart Contracts

- **Real Estate:** Reduce money paid to the middleman and distribute between the parties actually involved. For example, a smart contract to transfer ownership of an apartment once a certain amount of resources have been transferred to the seller's account(or wallet).
- **Vehicle ownership:** A smart contract can be deployed in a blockchain that keeps track of vehicle maintenance and ownership. The smart contract can, for example, enforce vehicle maintenance service every six months; failure of which will lead to suspension of driving license.
- **Music Industry:** The music industry could record the ownership of music in a blockchain. A smart contract can be embedded in the blockchain and royalties can be credited to the owner's account when the song is used for commercial purposes. It can also work in resolving ownership disputes.
- **Government elections:** Once the votes are logged in the blockchain, it would be very hard to decrypt the voter address and modify the vote leading to more confidence against the ill practices.
- **Management:** The blockchain application in management can streamline and automate many decisions that are taken late or deferred. Every decision is transparent and available to any party who has the authority(an application on the private blockchain). For example, a smart contract can be deployed to trigger the supply of raw materials when 10 tonnes of plastic bags are produced.
- **Healthcare:** Automating healthcare payment processes using smart contracts can prevent fraud. Every treatment is registered on the ledger and in the end, the smart contract can calculate the sum of all the transactions. The patient can't be discharged from the hospital until the bill has been paid and can be coded in the smart contract.

Advantages of Smart Contracts

- **Recordkeeping:** All contract transactions are stored in chronological order in the blockchain and can be accessed along with the complete audit trail. However, the parties involved can be secured cryptographically for full privacy.
- **Autonomy:** There are direct dealings between parties. Smart contracts remove the need for intermediaries and allow for transparent, direct relationships with customers.
- **Reduce fraud:** Fraudulent activity detection and reduction. Smart contracts are stored in the blockchain. Forcefully modifying the blockchain is very difficult as it's computation-intensive. Also, a violation of the smart contract can be detected by the nodes in the network and such a violation attempt is marked invalid and not stored in the blockchain.
- **Fault-tolerance:** Since no single person or entity is in control of the digital assets, one-party domination and situation of one part backing out do not happen as the platform is decentralized and so even if one node detaches itself from the network, the contract remains intact.
- **Enhanced trust:** Business agreements are automatically executed and enforced. Plus, these agreements are immutable and therefore unbreakable and undeniable.
- **Cost-efficiency:** The application of smart contracts eliminates the need for intermediaries(brokers, lawyers, notaries, witnesses, etc.) leading to reduced costs. Also eliminates paperwork leading to paper saving and money-saving.

Challenges of Smart Contracts

- **No regulations:** A lack of international regulations focusing on blockchain technology(and related technology like smart contracts, mining, and use cases like cryptocurrency) makes these technologies difficult to oversee.
- **Difficult to implement:** Smart contracts are also complicated to implement because it’s still a relatively new concept and research is still going on to understand the smart contract and its implications fully.
- **Immutable:** They are practically immutable. Whenever there is a change that has to be incorporated into the contract, a new contract has to be made and implemented in the blockchain.
- **Alignment:** Smart contracts can speed the execution of the process that span multiple parties irrespective of the fact whether the smart contracts are in alignment with all the parties’ intention and understanding.

Example #1 - Constructor

```
// Solidity program to demonstrate
// creating a constructor
pragma solidity ^0.5.0;

// Creating a contract
contract constructorExample {

    // Declaring state variable
    string str;

    // Creating a constructor
    // to set value of 'str'
    constructor() public {
        str = “SVNIT";
    }

    // Defining function to
    // return the value of 'str'
    function getValue(
    ) public view returns (
      string memory) {
        return str;
    }
}
```

Solidity – View and Pure Functions

- The **view functions** are read-only function, which ensures that state variables cannot be modified after calling them. If the statements which modify state variables, emitting events, creating other contracts, using **selfdestruct** method, transferring ethers via calls, Calling a function which is not ‘view or pure’, using low-level calls, etc are present in view functions then the compiler throw a warning in such cases. By default, a get method is view function.

Example #2-View function

```
// Solidity program to
// demonstrate view
// functions
pragma solidity ^0.5.0;

// Defining a contract
contract Test {

    // Declaring state
    // variables
    uint num1 = 2;
    uint num2 = 4;

   // Defining view function to
   // calculate product and sum
   // of 2 numbers
   function getResult(
   ) public view returns(
     uint product, uint sum){
       uint num1 = 10;
       uint num2 = 16;
      product = num1 * num2;
      sum = num1 + num2;
   }
}
```

Example #3 Smart Contract that Returns Address and Balance of Owner using Solidity

```
// Solidity program to
// retrieve address and
// balance of owner
pragma solidity ^0.6.8;

// Creating a contract
contract MyContract
{
    // Private state variable
    address private owner;

     // Defining a constructor
     constructor() public{
        owner=msg.sender;
    }

    // Function to get
    // address of owner
    function getOwner(
    ) public view returns (address) {
        return owner;
    }

    // Function to return
    // current balance of owner
    function getBalance(
    ) public view returns(uint256){
        return owner.balance;
    }
}
```

Escrow Smart Contract

- Escrow is the third party which holds the asset(asset can be money, bond, stocks) on the presence of two parties. Escrow will release the fund when certain conditions are met.
  For Example, “A” is a seller and wants to sell his car, “B” is a buyer who wants to buy “A”‘s car so they will contact Escrow “C”(an arbiter) which hold the asset until “B” receives the car. When this condition will be met, Escrow will release the fund to “A”. This solves the issue of trust and prevents any discrepancy.

Example #4 **Escrow Smart Contract**

- Look into the additional file

Solidity – Fall Back Function

- The solidity fallback function is executed if none of the other functions match the function identifier or no data was provided with the function call. Only one unnamed function can be assigned to a contract and it is executed whenever the contract receives plain Ether without any data. To receive Ether and add it to the total balance of the contract, the fallback function must be marked payable. **If no such function exists, the contract cannot receive Ether through regular transactions and will throw an exception.**

Properties of a fallback function

- Has no name or arguments.
- If it is not marked **payable**, the contract will throw an exception if it receives plain ether without data.
- Can not return anything.
- Can be defined once per contract.
- It is also executed if the caller meant to call a function that is not available
- It is mandatory to mark it external.
- It is limited to 2300 gas when called by another function. It is so for as to make this function call as cheap as possible.

EXAMPLE #5

- Look into the additional file