



Contratos inteligentes en Ethereum: Programa tu propia criptomoneda

Viktor Jacynycz García

Departamento de ingeniería del software e inteligencia artificial
Universidad Complutense de Madrid

vsjg@ucm.es

About me



UNIVERSIDAD
COMPLUTENSE
MADRID



viktor@jacynycz.es



@jacynycz



in/viktor-jacynycz

Conocimientos previos (Opcional)

- Programación
- Criptomonedas
- Blockchain



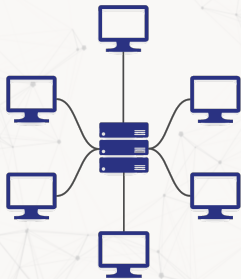
Contratos inteligentes en Ethereum

Introducción a blockchain

Blockchain nace con Bitcoin



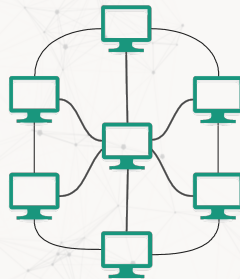
Blockchain es distribuido



Centralized
Architecture

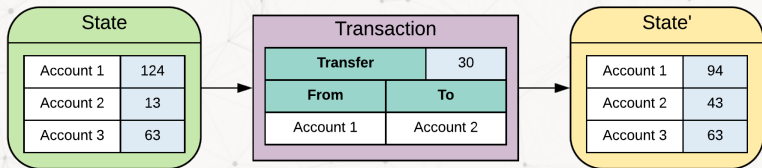


Decentralized
Architecture

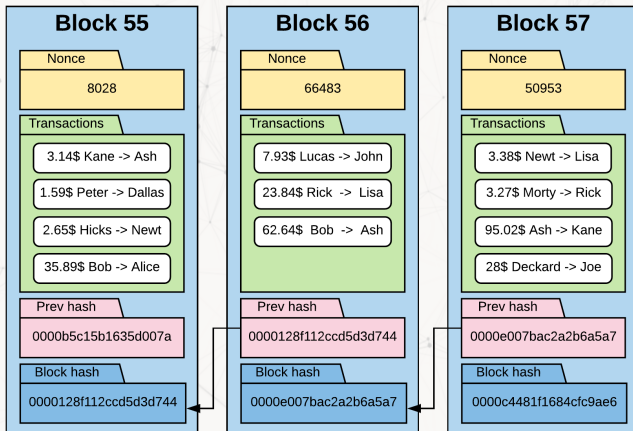


Distributed
Architecture

Blockchain se basa en transacciones



Blockchain se basa en transacciones

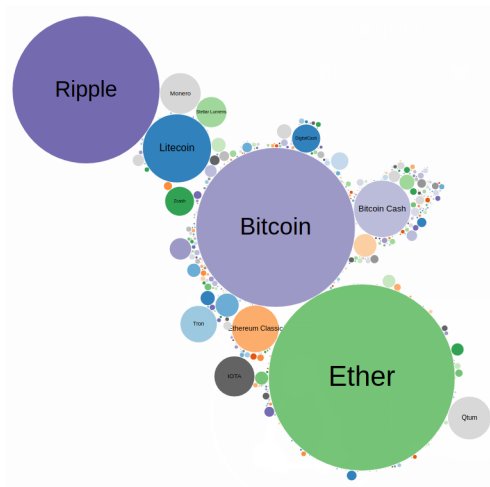


Ethereum utiliza la tecnología de Bitcoin



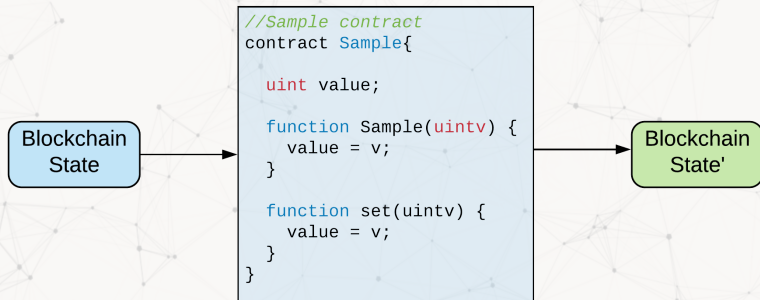
ethereum

Ether como criptomoneda



Mapa de el volumen de transacciones de las criptomonedas más usadas.

Ejecución de código distribuido





Contratos inteligentes en Ethereum

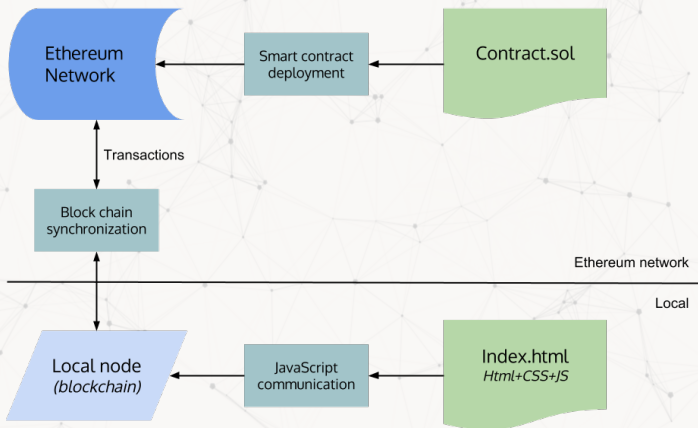
Cómo desarrollar smart contracts

Entorno de desarrollo

Existen dos formas de desarrollar contratos inteligentes:

- Local: Editor de texto/IDE y compilador.
- Navegador: Entorno de desarrollo en internet (<https://remix.ethereum.org/>).

Desarrollo en local



Es necesario un "frontend" para comunicarse con los contratos inteligentes.

Desarrollo en web

The screenshot displays a web browser window with a Solidity contract named 'Ballot' loaded. The contract code is visible in the left pane, and the web interface is shown in the right pane.

Contract Code (Solidity):

```
1 pragma solidity ^0.4.0;
2 contract Ballot {
3
4     struct Voter {
5         uint weight;
6         bool voted;
7         uint8 vote;
8         address delegate;
9     }
10
11     struct Proposal {
12         uint voteCount;
13     }
14
15     address chairperson;
16     mapping(address => Voter) voters;
17     Proposal[] proposals;
18
19     /// Create a new ballot with s(_numProposals) different proposals.
20     function Ballot(uint8 _numProposals) public {
21         chairperson = msg.sender;
22         voters[chairperson].weight = 1;
23         proposals.length = _numProposals;
24     }
25
26     /// Give s(toVoter) the right to vote on this ballot.
27     /// May only be called by s(chairperson).
28     function giveRightToVote(address toVoter) public {
29         if (msg.sender != chairperson || voters[toVoter].voted) return;
30         voters[toVoter].weight = 1;
31     }
32
33     /// Delegate your vote to the voter s(to).
34 }
```

Web Interface:

- Environment:** JavaScript VM
- Account:** 0xc3a3...a733c (99.9999999999999940296 wei)
- Gas limit:** 3000000
- Value:** 0 wei
- Ballot:** A dropdown menu showing the current ballot.
- Actions:** 'Create' button for 'uint8 _numProposals' and 'At Address' button for 'Load contract from Address'.
- Transactions:** 0 pending transactions.
- Ballot at 0xc3a3...a733c (memory):** A table showing the current state of the ballot.

winningProposal
delegate address to
giveRightToVote address toVoter
vote uint8 toProposal

Decoded Input:

```
{
  "uint8 _numProposals": "0"
}
```

Decoded Output:

```
-
```

Logs:

- 0 wei

Value:

- 0 wei



Contratos inteligentes en Ethereum

Elementos básicos de un smart contract

Ejemplo de contrato

```
pragma solidity ^0.5.1;

contract Basic{
    // 1 line comment
    uint stateVar;
    constructor() public {}
    function a(uint x, uint y) pure public{
        x+y;
    }
    function f(uint x) public returns(uint,uint){
        stateVar = x*x;
        return (x,x*x);
    }
}
```

Palabras reservadas

```
pragma solidity ^0.5.1;
```

Versión del compilador

```
contract Basic{
```

Inicio del contrato

```
// 1 line comment
```

```
uint stateVar;
```

Variables de estado

```
constructor() public {}
```

Constructor

```
function a(uint x, uint y) pure public{  
    x+y;  
}
```

Funciones

```
function f(uint x) public returns(uint, uint){  
    stateVar = x*x;  
    return (x, x*x);  
}
```

Contratos inteligentes en Ethereum

Tipos de datos

Tipos Básicos

```
bool booleanos;  
// operadores !, &&, ||, ==, !=  
  
uint8 unsignedInt;  
int128 integer;  
// operadores +, -, *, /, %, **, >>, <<  
// comparaciones <=, >=, ==, !=, <, >  
// operadores de bit &, |, ^, ~  
  
bytes32 arrayFijos;  
// arrays de tamaño fijo de bytes1 a bytes32  
// comparaciones <=, >=, ==, !=, <, >  
// operadores de bit &, |, ^, ~, >>, <<, .length  
  
bytes arrayDin;  
string cadenasDeTexto;
```

Arrays

```
uint[] array;

function f(uint len) public {
    // los arrays dinámicos se han de definir en memoria
    uint[] memory a = new uint[](7);

    // se han de construir con una longitud determinada
    bytes memory b = new bytes(len);

    // los strings de texto son arrays dinámicos
    string memory s = "los string en solidity son muy caros";

    // una vez creados, se pueden acceder y modificar
    a[6] = 8;

    /* se pueden pasar arrays de memory a storage
       pero es una operacion costosa */
    array = a;
}
```

Localización de los datos

Hay dos localizaciones importantes de los datos:

- **Memory:** Datos que están en la memoria y sólo existen durante la ejecución de una función.
- **Storage:** Datos que se “imprimen” dentro del contrato inteligente (por lo tanto se guardan en la cadena de bloques).

Localización de los datos

```
contract C {  
    uint[] x;  
  
    function mem(uint[] memory memoryArray) public {  
        x = memoryArray;  
        uint[] storage y = x;  
        y.length = 2;  
        delete x; // borra el array  
  
        uint z = f(memoryArray); //NO FUNCIONA  
        z = g(x);  
    }  
  
    function f(uint[] storage) internal pure returns (uint)  
    {  
        return 2;  
    }  
  
    function g(uint[] memory) internal pure returns (uint){  
        return 2;  
    }  
}
```



Contratos inteligentes en Ethereum

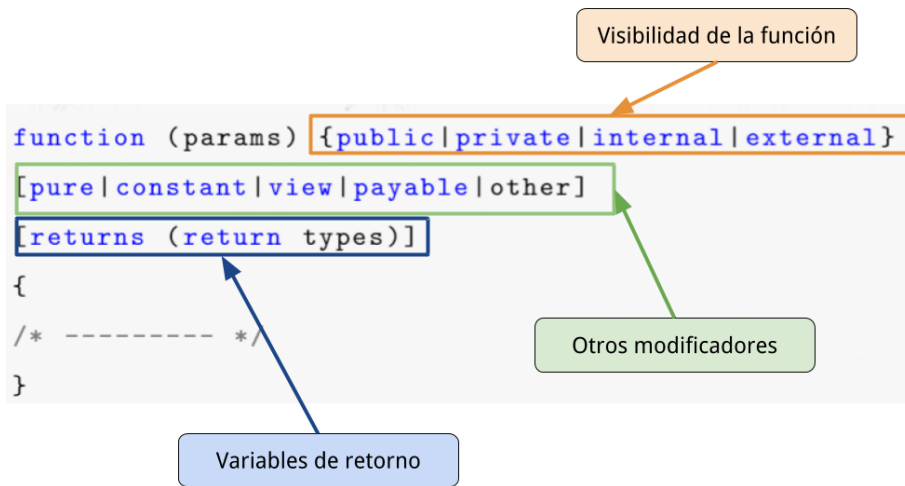
Modificadores en las funciones

Modificadores básicos

Los modificadores afectan en varios aspectos al comportamiento de las funciones:

```
function (params) {public|private|internal|external}  
[pure|constant|view|payable|other]  
[returns (return types)]  
{  
/* ----- */  
}
```

Modificadores Básicos



Visibilidad de las funciones

```
pragma solidity ^0.5.1;

contract Visibility{
    uint stateVar;

    function g(uint x, uint y) public{
        stateVar = x+y;
    }

    function f(uint x) private returns (uint){
        return x+1;
    }

    function h() internal{
        uint a = f(2);
    }
}
```

Visibilidad de las funciones

- **public**: Todos pueden acceder a la función.
- **external**: Sólo puede ser accedida externamente.
- **internal**: Sólo el contrato y los contratos derivados pueden acceder.
- **private**: Sólo se puede acceder dentro del contrato.

Gestión de memoria

```
pragma solidity ^0.5.1;

contract Memory{
    uint myVar;

    constructor() public {
        // mete el valor dentro del storage
        myVar = 0;
    }

    function viewF() public view returns (uint){
        // observa el valor de la variable en storage
        return myVar;
    }

    function pureF() public pure returns (uint){
        // no tiene que obtener ningun valor del storage
        return 2;
    }

    function memoryF(uint x) public{
        // tanto x como y -> memory
        uint y = x;
        // el valor pasa de memory -> storage
        myVar = y;
    }
}
```



Contratos inteligentes en Ethereum

Otros tipos de datos

Direcciones Ethereum

```
pragma solidity ^0.5.1;

contract Addresses{
    address direccionEthereum;
    address payable direccionEthereumPagable;
    // operadores <=, <, ==, !=, >=, >

    function addressOperations () public {
        if (address(this).balance < 10){
            direccionEthereumPagable.transfer(10);
            direccionEthereum.transfer(10); // no funciona
        }
    } // las direcciones tienen balance y pueden hacer
      transfer
}
```

Enums

```
pragma solidity ^0.5.1;

contract Tortilla{
    enum Choice{
        CONCEBOLLA, SINCEBOLLA
    }

    mapping (address => Choice) votos;

    constructor() public {}

    function votar(uint8 voto) public {
        votos[msg.sender] = Choice(voto);
    }
}
```


Structs

```
pragma solidity ^0.5.1;

contract Bank{

    struct Account{
        uint balance;
        uint256 lastMovement;
    }

    mapping (address => Account) public users;

    constructor() public{}

    function addUser(address _userAddr)
    public {
        users[_userAddr] = Account({balance: 0 ,
            lastMovement: now});
    }

    function addFunds() payable public{
        users[msg.sender].balance += msg.value;
        users[msg.sender].lastMovement = now;
    }
}
```

Structs y Enums

```
pragma solidity ^0.5.1;

contract Human{
    struct HumanInfo{
        uint age; Gender gender;
    }
    enum Gender{
        MALE,FEMALE,OTHER
    }
    HumanInfo public human;
    function createHuman(uint _age, uint _gender) public{
        human = HumanInfo({age:_age,gender:Gender(_gender)
        });
    }
}
```

Remitente de la transacción

La palabra **msg** contiene información sobre la dirección que realizó la transacción:

```
msg.data (bytes calldata): complete calldata
msg.sender (address payable): sender of the message (current
    call)
msg.sig (bytes4): first four bytes of the calldata (i.e.
    function identifier)
msg.value (uint): number of wei sent with the message
```

The background of the slide features a complex, light gray network pattern of interconnected nodes and lines, resembling a web or a molecular structure, set against a white background.

Contratos inteligentes en Ethereum

Eventos y modificadores personalizados

Eventos

```
pragma solidity ^0.5.1;

contract Events
{
    uint value;

    // definimos el evento
    event SetEvent(address who, uint val);

    function Sample(uint v) public {
        value = v;
    }

    function set(uint v) public{
        value = v;
        //Lanza el evento SetEvent
        emit SetEvent(msg.sender,v);
    }
}
```

Modificadores personalizados

```
contract Mods {  
    address owner;  
    constructor() public {  
        owner = msg.sender;  
    }  
    modifier costs(uint price) {  
        require (msg.value >= price,  
            "Esta función cuesta dinero");  
        -;  
    }  
    modifier onlyOwner{  
        require(msg.sender == owner);  
        -;  
    }  
    function payme() payable public costs(100){}  
    function ownerOnly() view public onlyOwner {}  
}
```

Herencia de contratos

```
pragma solidity ^0.5.1;

contract owned {
    constructor() public { owner = msg.sender; }
    address payable owner;

    modifier onlyOwner {
        require( msg.sender == owner );
        -;
    }
}

contract priced{
    modifier costs(uint price) {
        require (msg.value >= price);
        -;
    }
}

contract mortal is owned,priced {

    function close() public onlyOwner costs(1000) {
        selfdestruct(owner);
    }
}
```

¡A programar!



UNIVERSIDAD
COMPLUTENSE
MADRID



viktor@jacynycz.es



[@jacynycz](https://twitter.com/@jacynycz)



[in/viktor-jacynycz](https://in.linkedin.com/in/viktor-jacynycz)