

Curso desarrollo Blockchain Ethereum con Solidity

Clase 3

Solidity

Solidity

El siguiente, es un ejemplo de un smartContract hecho con Solidity versión 0.4.17



The image shows a code editor window titled "browser/educacionItContract.sol". The editor contains the following Solidity code:

```
1 pragma solidity ^0.4.17;
2 import "./jds.sol";
3 //EducacionIT 2018 -
4 contract EjemploEducacionIT {
5     string public owner;
6
7     constructor (string name) public {
8         owner = name;
9     }
10
11     function getHelloWorldMessage() public pure returns (string) {
12         return "Hello World!";
13     }
14 }
```

The left sidebar of the editor shows a file tree with the following structure:

- browser
 - jds.sol
 - educacionItContract.sol
- config

Solidity

A screenshot of a Solidity code editor. On the left, a tree view shows a project structure with a 'browser' folder containing 'jds.sol' and 'educacionItContract.sol', and a 'config' folder. The main editor area displays the code for 'educacionItContract.sol'. The code is as follows:

```
1 pragma solidity ^0.4.17;
2 import "./jds.sol";
3 //EducacionIT 2018 -
4 contract EjemploEducacionIT {
5     string public owner;
6
7     constructor (string name) public {
8         owner = name;
9     }
10
11     function getHelloWorldMessage() public pure returns (string) {
12         return "Hello World!";
13     }
14 }
```

- **Pragma solidity** permite indicar la versión de Solidity a utilizar
- **Contract** es una palabra reservada que indica el inicio de la definición de un contrato
- **Public** es un modificador de acceso que le da visibilidad fuera del contrato
- **Function** es una palabra reservada para indicar el inicio de definición de una función
- **Returns** indica que la función retornará algún valor
- La asignación en Solidity es de derecha a izquierda

Versiones y retrocompatibilidad

Versiones y retrocompatibilidad

- La versión actual de Solidity es la **v0.5.0** *
- Es posible interactuar con contratos de versiones anteriores
- No es recomendable combinar mas de dos cambios en el número principal de versión**
- Cuando ocurre un cambio mayor de versión, es posible que muchos contratos dejen de compilar debido a cambios
- Todos estos cambios siempre son notificados públicamente y a todo se le puede hacer seguimiento en GitHub



Contrato

Contrato

- Un contrato no es más que un acuerdo entre dos o más partes, un entorno donde se define lo que se puede hacer, cómo se puede hacer, qué pasa si algo no se hace
- Son reglas de juego que permite, a todas las partes que lo aceptan, entender en qué va a consistir la interacción que van a realizar
- Se encuentran distribuidos en la Ethereum BlockchainA bajo nivel, se trata de una serie de algoritmos, hashes, lógica y varios miles de líneas de código.
- Son capaces de ejecutarse por sí mismos de manera autónoma **sin intermediarios**
- Los smartContracts son "scripts" desarrollados en diferentes lenguajes de programación



Contrato

Un Smart Contract podría tener infinitas razones de uso, como ser

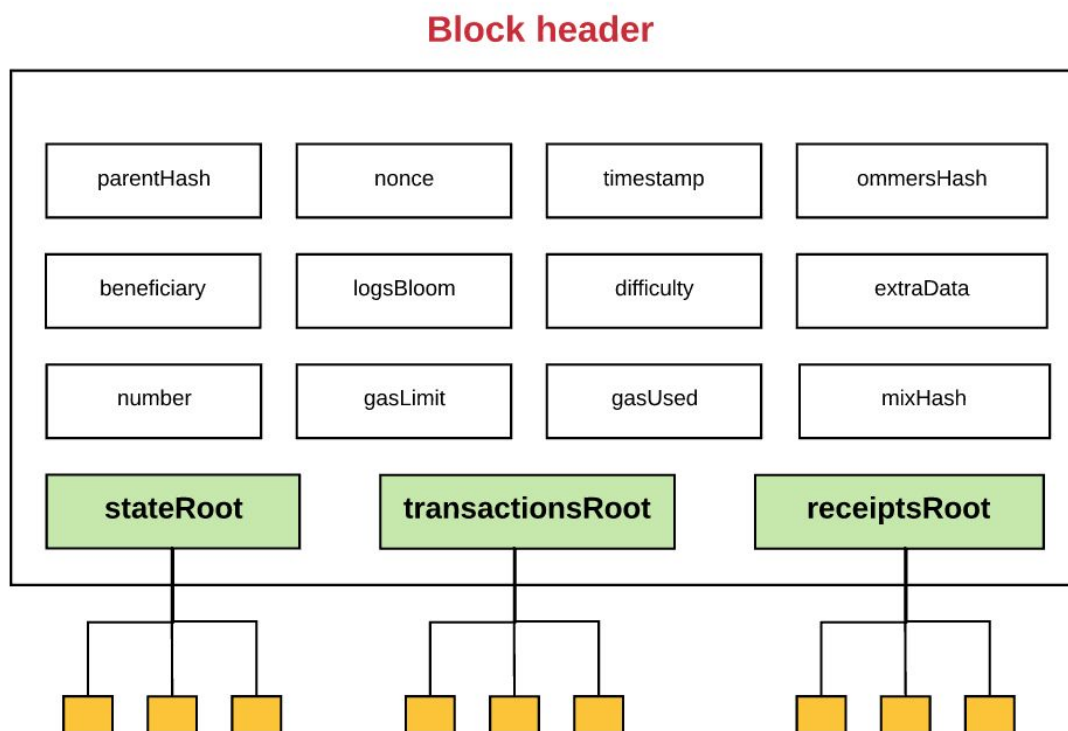
- **Mercados distribuidos** que permitieran implementar contratos P2P y trading en los mercados con CriptoMonedas (evitando el FIAT).
- **Propiedades** como automóviles, teléfonos, casas o elementos no físicos controlados a través de la blockchain (Smart Property)*
- **Automatización de herencias** estableciendo la asignación de los activos tras el fallecimiento. En cuanto llegase el fallecimiento, el contrato entraría en vigor y se ejecutaría repartiendo en este caso los fondos a la dirección establecida en el contrato.
- **Seguros** partes de accidente, pagos de la compañía para reparaciones, reducción del fraude en accidentes



Block Properties & Transaction Properties

Block Properties & Transaction Properties

Con toda transacción que es enviada a la Ethereum Blockchain, se envía además, información extra adicional que permite realizar ciertos manejos y validaciones sumamente útiles



Block Properties & Transaction Properties

Block

- `block.blockhash(uint blockNumber)` returns (bytes32): hash del bloque
- `block.coinbase(address)`: dirección del minero que cerró el bloque
- `block.difficulty(uint)`: dificultad del bloque
- `block.gaslimit(uint)`: limite de gas del bloque
- `block.number(uint)`: numero del bloque actual
- `block.timestamp(uint)`: Timestamp del bloque actual

Message

- `msg.data(bytes)`: llamada en bytecode
- `msg.gas(uint)`: gas sobrante
- `msg.sender(address)`: quien ha hecho esta llamada
- `msg.sig(bytes4)`: identificador de la función (son los 4 primeros bytes de `msg.data`)
- `msg.value(uint)`: cantdad en wei enviada
- `now(uint)`: timestamp del bloque actual (lo mismo que `block.timestamp`)

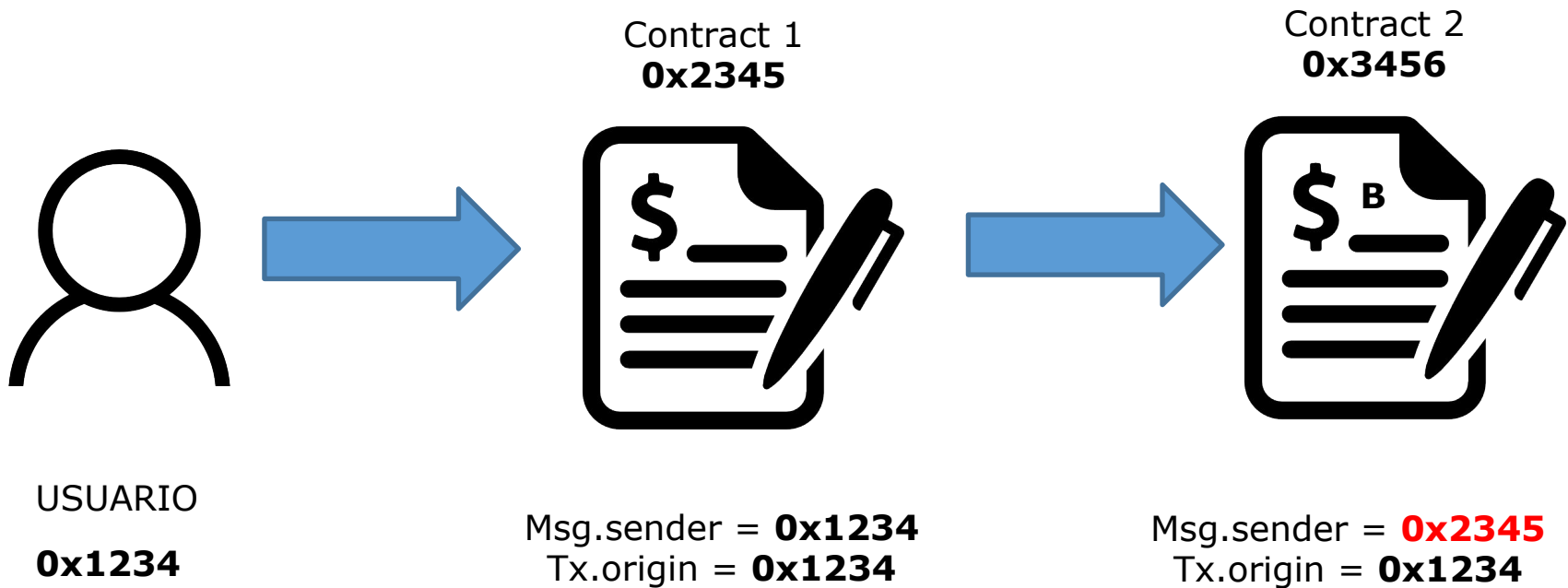
Transacción

- `tx.gasprice(uint)`: precio del gas para esta transacción
- `tx.origin(address)`: dirección del origen de la transacción.



Block Properties & Transaction Properties

Una de las utilidades que nos brinda tener Block & Transaction Properties es poder hacer cosas como

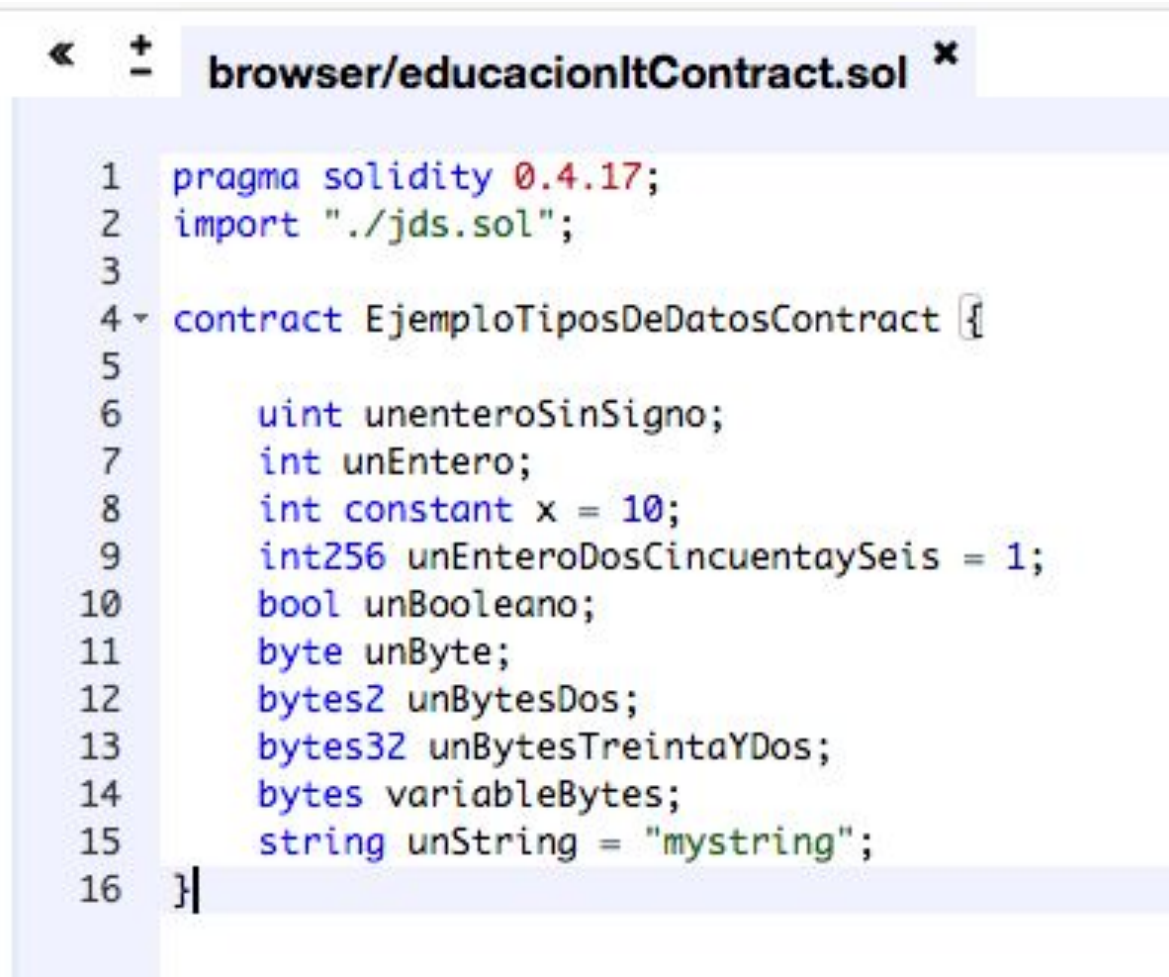


Si **tx.origin == msg.sender** quien llama al método no es un contrato.

Tipos, estructuras y modificadores

Tipos, estructuras y modificadores

Entre los tipos más comunes de Solidity encontramos los siguientes



```
browser/educacionItContract.sol x
1 pragma solidity 0.4.17;
2 import "../jds.sol";
3
4 contract EjemploTiposDeDatosContract {
5
6     uint unenteroSinSigno;
7     int unEntero;
8     int constant x = 10;
9     int256 unEnteroDosCincuentaySeis = 1;
10    bool unBooleano;
11    byte unByte;
12    bytes2 unBytesDos;
13    bytes32 unBytesTreintaYDos;
14    bytes variableBytes;
15    string unString = "mystring";
16 }
```

Tipos, estructuras y modificadores

Entre los tipos más comunes de Solidity encontramos los siguientes

uint unenteroSinSigno;	Entero sin signo
int unEntero;	Entero con signo (puede ser negativo)
int constant x = 10;	Entero con signo constante*
int256 unEnteroDosCincuentaySeis = 1;	Entero que ocupa 256 bits
bool unBooleano;	Booleano (true / false)
byte unByte;	Un unico byte
bytes2 unBytesDos;	Dos bytes juntos
bytes32 unBytesTreintaYDos;	Treintaydos bytes juntos
bytes variableBytes;	Es equivalente a un array de bytes
string unString = "mi string";	Texto, es un array en simismo

Tipos en Ethereum

Tipos en Ethereum

Entre los tipos más comunes de Ethereum encontramos los siguientes

```
« + browser/educacionItContract.sol x
1  pragma solidity 0.4.17;
2  import "./jds.sol";
3
4  contract EjemploTiposDeDatosContract {
5      address public owner;
6      mapping(address => uint) public balances;
7
8      bool etherToFinney = (2 ether == 2000 finney);
9
10     uint cuatromilWeis = 4000 wei;
11
12     bool unminuto = (1 minutes == 60 seconds);
13
14     uint timestamp = now;
15 }
16
```

Tipos en Ethereum



Entre los tipos más comunes de Ethereum encontramos los siguientes

Tipo de dato	Explicación
address public owner;	Es un tipo específico de Ethereum. Provee metodos y propiedades útiles para transaccionar
mapping(address => uint) public balances;	Es un tipo específico de Ethereum propio para mapear dos tipos de datos
bool etherToFinney = (2 ether == 2000 finney);	Finney al igual que zsabo, wei o ether, son tipos específicos disponibles en Ethereum
uint cuatromilWeis = 4000 wei ;	Finney al igual que zsabo, wei o ether, son tipos específicos disponibles en Ethereum
bool unminuto = (1 minutes == 60 seconds);	Minutes y seconds son dos tipos de equivalencias disponibles en Ethereum
uint timestamp = now ;	A traves de now se obtiene el momento actual

Tipos en Ethereum

Es posible definir enumeraciones como en el siguiente ejemplo

```

< + browser/educacionItContract.sol x
1  pragma solidity 0.4.17;
2  import "./jds.sol";
3
4  contract EjemploEstructurasContract {
5
6      enum Estados { EstadoUno, EstadoDos, EstadoTres, EstadoN }
7
8      Estados public state = Estados.EstadoN;
9
10     struct Usuario {
11         string nombre;
12         uint edad;
13         address wallet;
14     }
15
16     Usuario public miUsuario;
17 }
18
19
```

Tipos en Ethereum

Es posible definir **enumeraciones** como en el siguiente ejemplo

```
enum Estados { EstadoUno, EstadoDos, EstadoTres, EstadoN }
```

Luego, ya se encuentra disponible nuestro nuevo tipo de dato para utilizarlo y definir nuevas propiedades como en el ejemplo

```
Estados public state = Estados.EstadoN;
```

Al escribir Estados punto, el compilador nos sugerirá una de las opciones disponibles (las cuales fueron definidas con anterioridad)

Tipos en Ethereum

Es posible definir **estructuras propias** como en el siguiente ejemplo

```
struct Usuario {  
    string nombre;  
    uint edad;  
    address wallet;  
}
```

Dentro de cada estructura propia se pueden utilizar los mismos tipos que se encuentran disponible en Solidity. Es altamente recomendable la correcta utilización de los tipos dentro ya que es posible generar un consumo excesivo de no hacerlo

```
Usuario public miUsuario;
```

Luego, es posible acceder a las propiedades internas de la estructura mediante la invocación con el nombre de la variable y el punto

Logging & Eventos

Logging & Eventos

El siguiente es un ejemplo de uso de Logging y Eventos en un contrato dentro de Ethereum

```
browser/educacionItContract.sol x
1  pragma solidity 0.4.17;
2  import "./jds.sol";
3
4  contract EjemploLoggingYEventosContract {
5
6      function FuncionPrivada() private {
7
8      }
9
10     function hacerAlgo(string _mensajeRecibido) public {
11         PasoAlgoEvent(_mensajeRecibido);
12     }
13
14     event PasoAlgoEvent(string _unMensaje);
15 }
16
```


Logging & Eventos

Para definir un evento en Solidity para Ethereum, basta con utilizar la palabra clave “**event**” de la siguiente manera:

```
event PasoAlgoEvent(string _unMensaje);
```

Luego, para invocar al evento, en cualquier función, sea publica o privada, basta con llamar al evento con el nombre definido con anterioridad.

De manera que la llamada, dentro de una función se vería como la siguiente

```
function hacerAlgo(string _mensajeRecibido) public {  
    PasoAlgoEvent(_mensajeRecibido);  
}
```

Por supuesto, el valor enviado al evento ha de ser del mismo tipo que el definido y puede o no ser el recibido por la función

Simulación práctica de un contrato distribuido con Solidity

Simulación práctica de un contrato distribuido con Solidity

Escribiremos el siguiente contrato en Solidity usando Remix y lo haremos compilar

browser/educacionItContract.sol ✕

```
pragma solidity 0.4.17
import "./jds.sol";

contract EjemploEstructurasContract {

    enum Estados { EstadoUno, EstadoDos }
    address owner;

    Estados public state = Estados.EstadoN;

    struct Usuario {
        string nombre;
        uint edad;
        address wallet;
    }

    Usuario public miUsuario;

    function hacerAlgo(string _parametro) private {
        miUsuario = Usuario("Joaq", 26, 0x3a7D7125ba608175bBb48E014274127D81f4c25a);
        mi Usuario.nombre = "Joaquin";
    }
}
```