

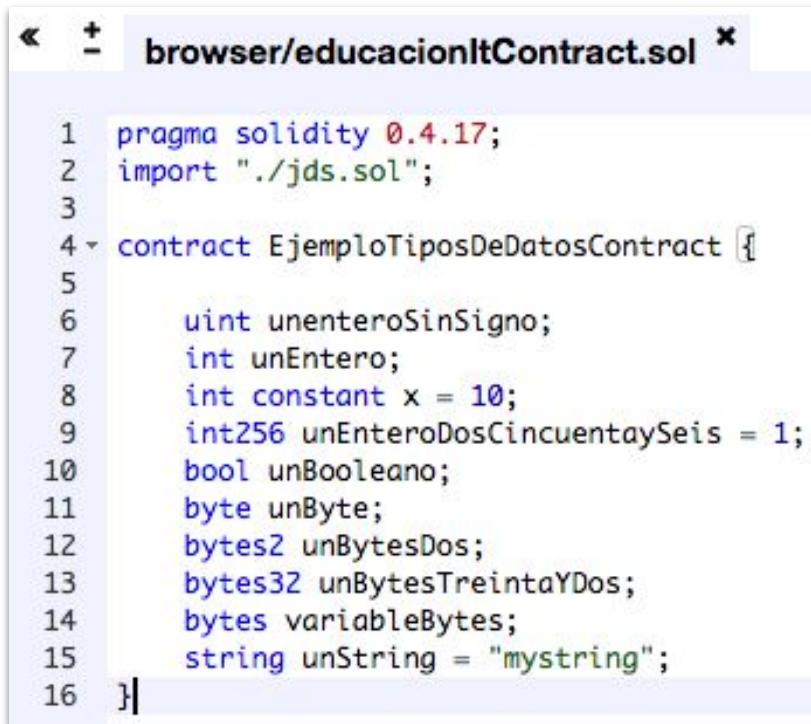
# Desarrollo Blockchain Ethereum con Solidity

Módulo 2 - Tipos de dato

# Tipos de dato

# Tipos, estructuras y modificadores

Entre los tipos más comunes de Solidity encontramos los siguientes, que detallaremos a continuación.



```
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

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

# Números decimales

Los números decimales no están soportados actualmente en Solidity.

*¿Cómo piensan que se soluciona este problema?*



# 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 }
```

## Tipos en Ethereum

Tipo de dato	Explicación
<b>address</b> public owner;	Es un tipo específico de Ethereum. Provee métodos y propiedades útiles para transaccionar.
<b>mapping(address =&gt; uint)</b> public balances;	Es un tipo específico de Ethereum propio para mapear dos tipos de datos.
uint cuatromilWeis = 4000 <b>wei</b> ;	<i>Finney</i> al igual que <i>gwei</i> o <i>ether</i> , son tipos específicos disponibles en Ethereum.
bool unminuto = (1 <b>minutes</b> == 60 <b>seconds</b> );	<i>Minutes</i> y <i>seconds</i> son dos tipos de equivalencias disponibles en Ethereum.
uint timestamp = <b>now</b> ;	A través de <i>now</i> 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 }
```



## 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.

# Tuplas

- Las tuplas con secuencias de valores agrupados.
- Sirven para agrupar, como si fuesen un único valor, varios valores que por su naturaleza deben ir juntos.
- Las tuplas son inmutables.
- Una vez que son creadas no pueden ser modificadas.
- Pueden ser retornadas en funciones dentro de un Smart Contract.
- Se definen entre paréntesis.



# Tuplas

## Analizando la siguiente función del contrato.

```
function funcionQueRetornaTuplaDosValores() public pure returns(string,uint) {  
    return ("un valor de texto", 1012);  
}
```

Se puede apreciar que al invocarla, efectivamente se retornan dos valores y, en este caso, de distintos tipos de datos.

QueRetornaTuplaDos

0: string: un valor de texto

1: uint256: 1012

Por supuesto, pueden haber tuplas con conjuntos de datos del mismo tipo, como el caso siguiente:

```
function funcionQueRetornaDosStrings() public pure returns(string,string) {  
    return ("un valor de texto", "otro texto mas");  
}
```

# ¡Muchas gracias!

¡Sigamos trabajando!