

# Práctica 1: Introducción a la Programación de Smart Contracts en Solidity

# Índice

<b>Ejercicio 1</b>	<b>1</b>
<b>Ejercicio 2</b>	<b>2</b>
<b>Ejercicio 3</b>	<b>3</b>
<b>Ejercicio 4</b>	<b>4</b>
<b>Ejercicio 5</b>	<b>5</b>
<b>Ejercicio 6</b>	<b>6</b>
Análisis y definición del escenario	6
Diseño	6
Caso de uso	6
Contenido del contrato inteligente	6
Datos	6
Funciones	7
Usuarios del sistema	8
Implementación	8
Pruebas	9

# Ejercicio 1

Se ha creado una fábrica donde se fabrican o ensamblan diferentes productos inteligentes.

```
//SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.10;

contract FabricaContract{

    uint private idDigits = 16;

    struct Producto {
        string nombre;
        uint identificacion;
    }

    Producto[] public productos;

    function crearProducto(string memory _nombre, uint _id) private{
        Producto memory nuevoProducto = Producto(_nombre, _id);
        productos.push(nuevoProducto);
        emit NuevoProducto(productos.length-1, _nombre, _id);
    }

    function generarIdAleatorio(string memory _str) private view returns(uint){
        uint rand = uint(keccak256(abi.encodePacked(_str)));
        uint idModulus = 10^idDigits;
        return rand % idModulus;
    }

    function crearProductoAleatorio(string memory _nombre) public{
        uint randId = generarIdAleatorio(_nombre);
        crearProducto(_nombre, randId);
    }

    event NuevoProducto(uint ArrayProductId, string nombre, uint id);

    mapping (uint => address) public productoAPropietario;

    mapping (address => uint) propietarioProductos;

    function Propiedad(Producto memory _producto) private {
        address direccion = msg.sender;
        productoAPropietario[_producto.identificacion] = direccion;
        propietarioProductos[direccion]++;
    }

    function deProductosPorPropietario(address _propietario) external view returns(uint[] memory){
        uint contador = 0;
        uint contador2 = 0;
        uint[] memory resultado= new uint[](productos.length);
        for (contador; contador < productos.length; contador++){
            {
                if(productoAPropietario[productos[contador].identificacion] == _propietario){
                    resultado[contador2] = (productos[contador].identificacion);
                    contador2++;
                }
            }
        }
        return resultado;
    }
}
```

## Ejercicio 2

Implementación de un contrato para que cualquier token pueda ser comprado con Ether, siempre que el propietario todavía tenga suficientes tokens.

```
// SPDX-License-Identifier: Unlicensed
pragma solidity 0.8.18;
contract TokenContract {

    address public owner;

    struct Receivers {
        string name;
        uint256 tokens;
        uint256 _ether;
    }

    mapping(address => Receivers) public users;

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

    constructor(){
        owner = msg.sender;
        users[owner].tokens = 100;
    }

    function double(uint _value) public pure returns (uint){
        return _value*2;
    }

    function register(string memory _name) public{
        users[msg.sender].name = _name;
    }

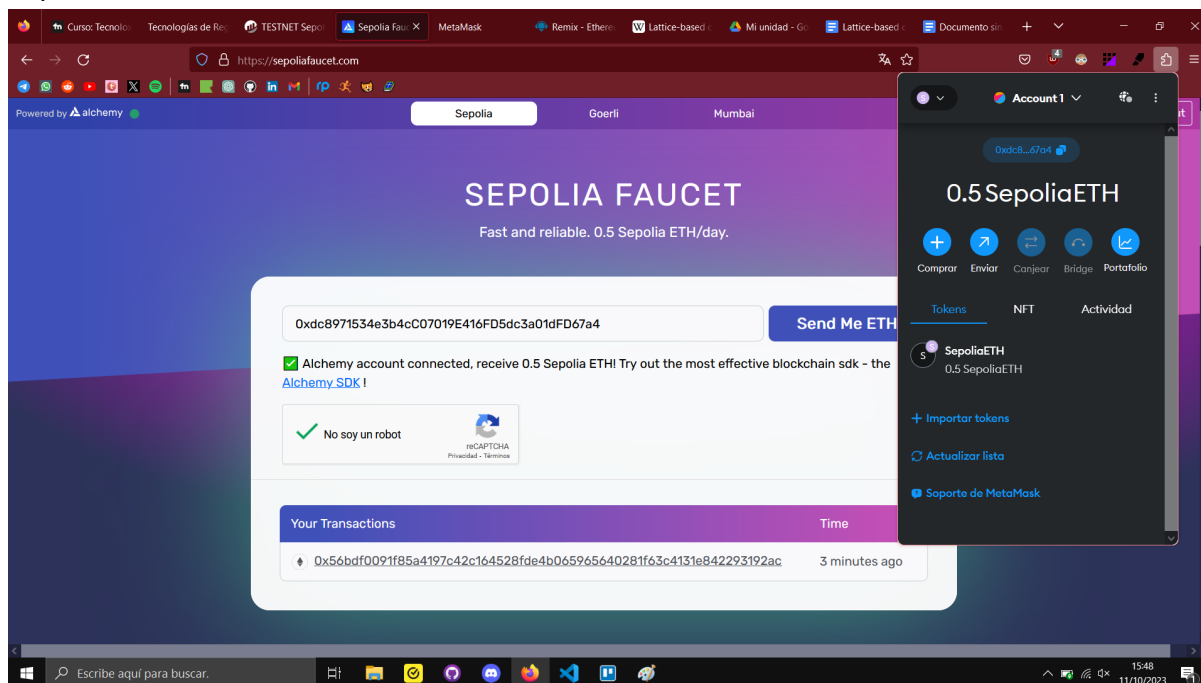
    function giveToken(address _receiver, uint256 _amount) onlyOwner public{
        require(users[owner].tokens >= _amount);
        users[owner].tokens -= _amount;
        users[_receiver].tokens += _amount;
    }

    function buyToken(address _receiver, uint256 _amount, uint256 _ether) public {
        require(users[owner].tokens >= _amount);
        require((_ether / 5) == _amount);
        require(users[_receiver]._ether >= _ether);
        users[owner].tokens -= _amount;
        users[owner]._ether += _ether;
        users[_receiver].tokens += _amount;
        users[_receiver]._ether -= _ether;
    }
}
```

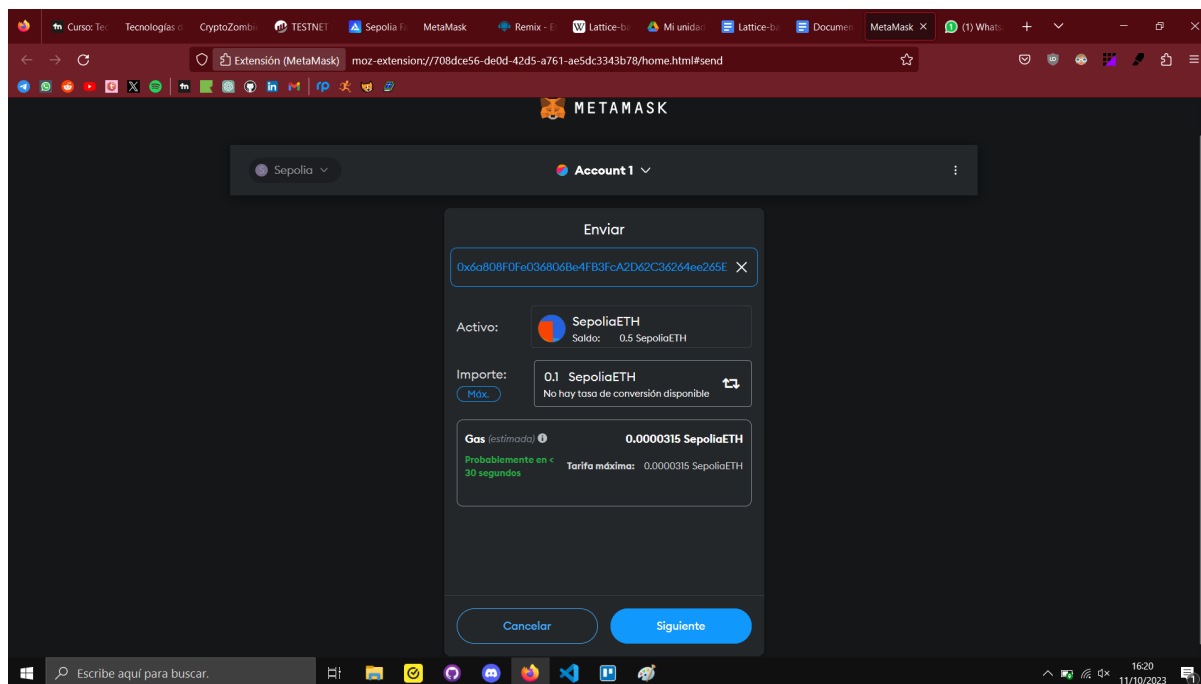
## Ejercicio 3

Obtener, enviar ETH de prueba de Sepolia a un compañero y realizar un seguimiento de las transacciones en Sepolia Blockchain Explorer.

Sepolia ETH obtenido:



Envío de ETH:



## Seguimiento de transacciones:

The screenshot shows the Etherscan Sepolia Testnet interface. The transaction details are as follows:

Field	Value
Transaction Hash	0x75aed6317a8c8b5e8bb5ffb0e34732ac18b84e2f15d808b61dc1fb6cd8a9938a
Status	Success
Block	4470351 (3 Block Confirmations)
Timestamp	31 secs ago (Oct-11-2023 02:20:36 PM +UTC)
From	0xdc8971534e3b4c07019E416FD5dc3a01dFD67a4
To	0x6a808F0Fe036806Be4FB3FcA2D62C36264ee265E
Value	0.1 ETH (\$0.00)
Transaction Fee	0.000031500000441 ETH (\$0.00)
Gas Price	1.500000021 Gwei (0.000000001500000021 ETH)

## Ejercicio 4

Completar Solidity: Beginner to Intermediate Smart Contracts.

The screenshot shows the course content for 'Solidity: Beginner to Intermediate Smart Contracts' on Cryptozombies.io. The course consists of 6 lessons, all of which are 100% completed:

- Creando la Fábrica de Zombies
- Los Zombies Atacan A Sus Víctimas
- Conceptos Avanzados de Solidity
- Sistema de Batalla Zombie
- ERC721 & Crypto-Coleccionables
- App Front-ends & Web3.js

## Ejercicio 5

Escoger y probar uno de los contratos de OpenZeppelin.

```
// contracts/access-control/Auth.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Auth {
    address private _administrator;

    constructor(address deployer) {
        // Make the deployer of the contract the administrator
        _administrator = deployer;
    }

    function isAdministrator(address user) public view returns (bool) {
        return user == _administrator;
    }
}
```

## Ejercicio 6

### Análisis y definición del escenario

Un contrato que permite a los inquilinos reservar y acceder a una propiedad de alquiler por un período determinado después de realizar un pago en criptomonedas.

### Diseño

#### Caso de uso

Identificador	Alquiler de una propiedad
Descripción	El sistema deberá permitir al inquilino solicitar el alquiler de una propiedad
Secuencia Normal	1.- El usuario realiza una petición al contrato
	2.- El sistema comprueba si la cantidad de criptomoneda es suficiente para la realización del contrato
	2a.- Si el usuario posee suficiente crédito, se le asigna la propiedad durante el tiempo especificado y se cobra el precio de su alquiler al usuario
	2.b- Si el usuario no posee suficiente crédito, se deniega la transacción
Importancia	De alta importancia para el funcionamiento de la aplicación

### Contenido del contrato inteligente

#### Datos

- Propiedad**
- Estructura de la propiedad a alquilar. Contiene:
  - nombre** - Nombre de la propiedad.
  - dirección** - Dirección física de la propiedad.
  - estado** - Estado físico de la propiedad.
  - precioDia** - Precio de la propiedad por día.
- Alquiler**
- Estructura para almacenar información sobre cada alquiler.
  - inquilino** - Dirección del inquilino.
  - inicio** - Fecha de inicio del alquiler.
  - fin** - Fecha de finalización del alquiler.
- propiedadADuenho** - Mapping que relaciona una propiedad con su dueño.
- duenhoAPropiedad** - Mapping que relaciona a un usuario con el número de propiedades que posee.



**propiedadAAquiler** - Mapping que relaciona una propiedad con un array de alquileres.

## Funciones

**reservar(uint \_idPropiedad, uint \_inicioAlquiler, uint \_finAlquiler) public payable {}**

Permite a un inquilino alquilar una propiedad.

**renovar(uint \_idPropiedad, uint \_inicioAlquiler, uint \_finAlquiler) public payable {}**

Permite a un inquilino volver a alquilar una propiedad que ya ha alquilado anteriormente.

**cancelarReserva(uint \_idPropiedad, uint \_inicioAlquiler, uint \_finAlquiler) public {}**

Permite a un inquilino cancelar un alquiler.

**anadirPropiedad(string memory \_nombre, string memory \_direccion, string memory \_estado, uint \_precioDia) public {}**

Permite a un propietario añadir una propiedad.

**eliminarPropiedad(uint \_idPropiedad) public onlyOwnerOf(\_idPropiedad) {}**

Permite a un propietario eliminar una de sus propiedades.

**modificarPropiedad(uint \_idPropiedad, string memory \_nombre, string memory \_direccion, string memory \_estado) public onlyOwnerOf(\_idPropiedad) {}**

Permite a un propietario modificar una de sus propiedades.

**cancelarReservaInquilino(uint \_idPropiedad, uint \_inicioAlquiler, uint \_finAlquiler, address \_inquilino) public onlyOwnerOf(\_idPropiedad) {}**

Permite a un propietario cancelar el alquiler de un inquilino.

**cancelarReservasInquilino(uint \_idPropiedad, address \_inquilino) public onlyOwnerOf(\_idPropiedad) {}**

Permite a un propietario cancelar todos los alquileres de un inquilino.

**cancelarReservasPropiedad(uint \_idPropiedad) public onlyOwnerOf(\_idPropiedad) {}**

Permite a un propietario cancelar todos los alquileres de una propiedad.

**\_terminoElAlquiler(uint \_idPropiedad, address \_inquilino) internal view returns (bool) {}**

Comprueba si el alquiler de un inquilino ya ha terminado.

**\_rangoAlquileres(uint \_idPropiedad, uint \_inicioAlquiler, uint \_finAlquiler) internal view returns (bool) {}**

Comprueba si una propiedad está alquilada durante un periodo de tiempo.

**\_esInquilino(uint \_idPropiedad, address \_inquilino) internal view returns (bool) {}**

Comprueba si un inquilino ha alquilado una propiedad.

**\_eliminarAlquiler(uint \_idPropiedad, uint \_id) internal {}**

Función interna para eliminar alquileres.

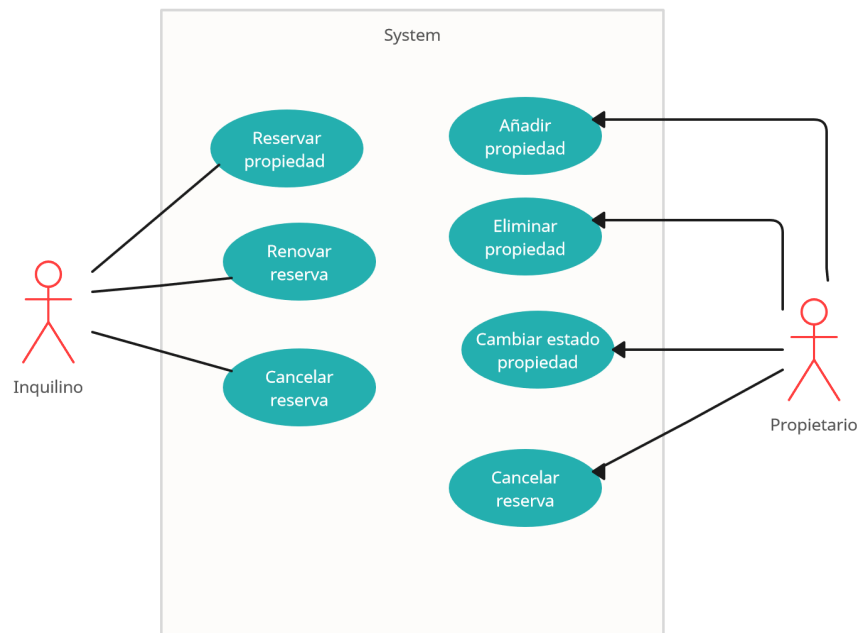
**transferirFondosAlDuenho() public {}**

Función para acceder a los fondos del contrato.

## Usuarios del sistema

Dueños de propiedades: Usuarios que ofrezcan propiedades a alquilar.

Inquilinos: Usuarios que deseen alquilar propiedades.



## Implementación

Disponible en [github](#).

# Pruebas

## Prueba reservar

Aquí se puede observar cómo la transacción ha sido cancelada porque no tengo suficiente ETH cómo para alquilar la propiedad.

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel displays a list of deployed contracts for 'ALQUILER AT 0xD8B...33Fa8 (MEMORY)'. The 'reservar' function is selected, with input fields for `_idPropiedad` (0), `_inicioAlquiler` (1638352800), and `_finAlquiler` (1638871200). The 'transact' button is highlighted in red. On the right, the Solidity code editor shows the `alquiler` contract. The `reservar` function is defined as follows:

```
function reservar(uint _idPropiedad, uint _inicioAlquiler, uint _finAlquiler) public {
    require(_idPropiedad < propiedades.length);
    require(_inicioAlquiler < _finAlquiler);
    require(msg.value >= propiedades[_idPropiedad].precioDia * (_finAlquiler - _inicioAlquiler));
    propiedades[_idPropiedad].estado = 'reservado';
}
```

The console on the right shows the following messages:

```
transact to alquiler.reservar pending ...
transact to alquiler.reservar errored: Error occurred: revert.
revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "La cantidad pagada es insuficiente para el alquiler.".
Debug the transaction to get more information.
```

A red error icon is visible in the console, indicating a revert due to insufficient funds.

## Prueba renovar

Lo mismo se puede observar con renovar.

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel displays the same list of deployed contracts. The 'renovar' function is selected, with input fields for `_idPropiedad` (0), `_inicioAlquiler` (1638352800), and `_finAlquiler` (1638352800). The 'transact' button is highlighted in red. On the right, the Solidity code editor shows the `alquiler` contract. The `renovar` function is defined as follows:

```
function renovar(uint _idPropiedad, uint _inicioAlquiler, uint _finAlquiler) public {
    require(_idPropiedad < propiedades.length);
    require(_inicioAlquiler < _finAlquiler);
    require(msg.value >= propiedades[_idPropiedad].precioDia * (_finAlquiler - _inicioAlquiler));
    propiedades[_idPropiedad].estado = 'disponible';
}
```

The console on the right shows the following messages:

```
transact to alquiler.renovar pending ...
transact to alquiler.renovar errored: Error occurred: revert.
revert
The transaction has been reverted to the initial state.
Note: The called function should be payable if you send value and the value you send should be less than your current balance.
Debug the transaction to get more information.
```

A red error icon is visible in the console, indicating a revert due to insufficient funds.

## Prueba anhadirPropiedad

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, displaying the 'anhadirPropiedad' function with its parameters: `_nombre` (Casa blanca), `_direccion` (RIO JAMAPA 36, CUALIHEMOC, 91069), `_estado` (habitable), and `_precioDia` (25). Below the function definition, a list of transactions is shown, including 'cancelarReserva', 'eliminarPropie...', 'modificarPropie...', 'renovar', 'reservar', 'transferirFond...', 'dueñoAPropie...', and 'propietadAAliq...'. The 'anhadirPropiedad' transaction is highlighted. On the right, the 'Solidity' editor shows the source code for the 'alquiler' contract, which includes a constructor and a 'onlyOwnerOf' modifier. The 'Run' button is visible, and the terminal at the bottom shows the execution of the transaction, with a message indicating that the transaction is pending.

## Prueba eliminarPropiedad

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, displaying the 'eliminarPropiedad' function with its parameter: `_idPropiedad` (1). Below the function definition, a list of transactions is shown, including 'modificarPropie...', 'removar', 'reservar', 'transferirFond...', 'dueñoAPropie...', and 'propietadAAliq...'. The 'eliminarPropiedad' transaction is highlighted. On the right, the 'Solidity' editor shows the source code for the 'alquiler' contract, which includes a 'eliminarPropiedad' function. The 'Run' button is visible, and the terminal at the bottom shows the execution of the transaction, with a message indicating that the transaction is pending.

## Prueba modificarPropiedad

**DEPLOY & RUN TRANSACTIONS**  
Transactions recorded 1

**Deployed Contracts**

ALQUILER AT 0XD8B...33FA8 (MEMORY1)

Balance: 0 ETH

anadirPropiedad... string \_nombre, string \_direccion, string \_estado, uint256 \_precioDia

cancelarReserva... uint256 \_idPropiedad, uint256 \_inicioAlquiler, uint256 \_finAlquiler

cancelarReserva... uint256 \_idPropiedad, uint256 \_inicioAlquiler, uint256 \_finAlquiler, address \_inquilino

cancelarReserva... uint256 \_idPropiedad, address \_inquilino

cancelarReserva... uint256 \_idPropiedad

cancelarReserva... uint256 \_idPropiedad

**modificarPropiedad**

\_idPropiedad: 1

\_nombre: Casa roja

\_direccion: RIO JAMAPA 36 CUALHTEMOC 91069

\_estado: habitable

\_precioDia: 25

Calldata Parameters transact

renovar uint256 \_idPropiedad, uint256 \_inicioAlquiler, uint256 \_finAlquiler

reservar uint256 \_idPropiedad, uint256 \_inicioAlquiler, uint256 \_finAlquiler

transferirFond... address

duenhoAPropi... address

```
85     propiedades[i] = propiedades[i+1];
86   }
87   propiedades.pop();
88 }
89
90 function modificarPropiedad(uint _idPropiedad, string memory _nombre, string memory _direccion,
91   Propiedad storage propiedadAModificar = propiedades[_idPropiedad];
92   propiedadAModificar.nombre = _nombre;
93   propiedadAModificar.direccion = _direccion;
94   propiedadAModificar.estado = _estado;
95   propiedadAModificar.precioDia = _precioDia;
96 }
97
98 function cancelarReservaInquilino(uint _idPropiedad, uint _inicioAlquiler, uint _finAlquiler,
99   require(!_esInquilino(_idPropiedad, _inquilino));
100   for(uint j = 0; j < propiedadAAquiler[_idPropiedad].length; j++){
```

transact to alquiler.modificarPropiedad pending ...

[vs] from: 0x583...edd4 to: alquiler.modificarPropiedad(uint256,string,string,string,uint256) 0xd8b...33fa8  
value: 0 wei data: 0x199...00000 logs: 0 hash: 0xae1...b021f