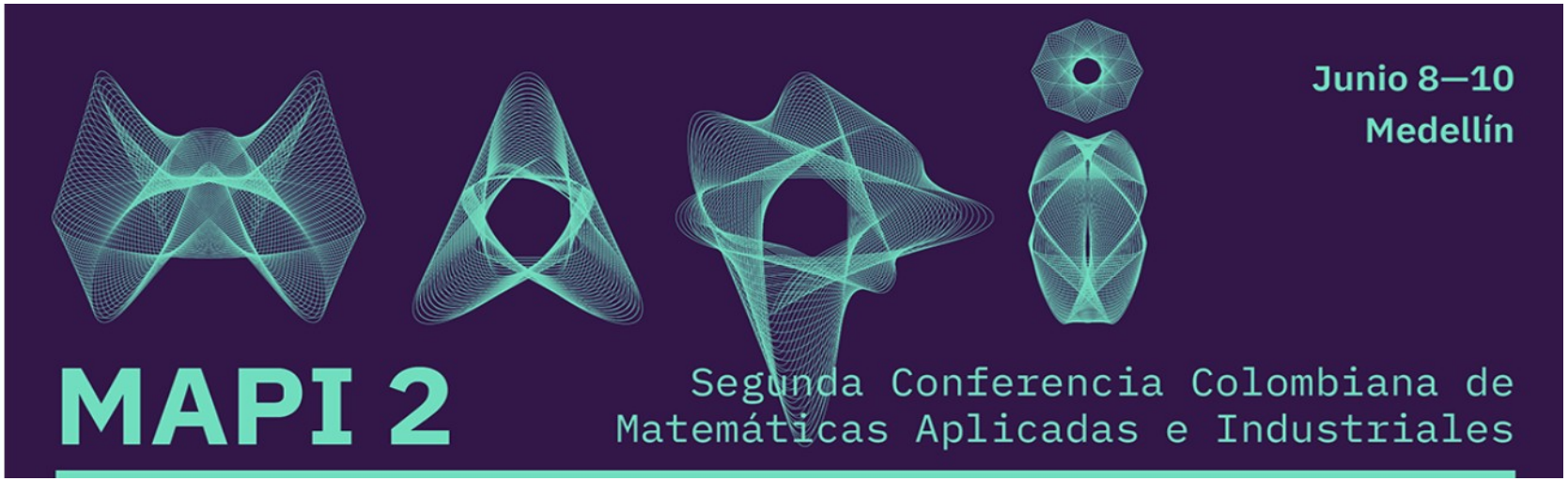
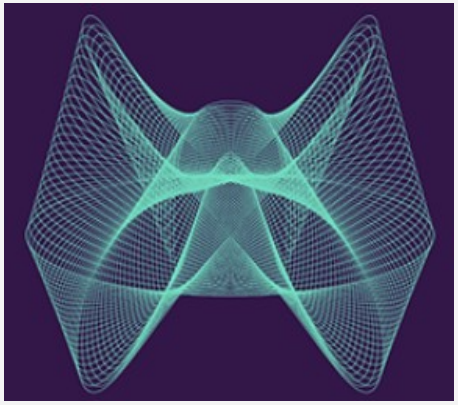


Curso Desarrollo de aplicaciones Blockchain

Parte II

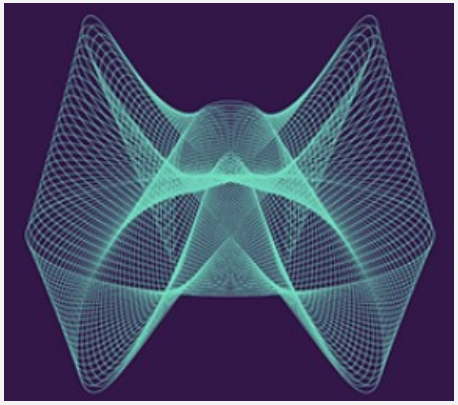
Darío Correal - Universidad de los Andes – Departamento de Sistemas y Computación



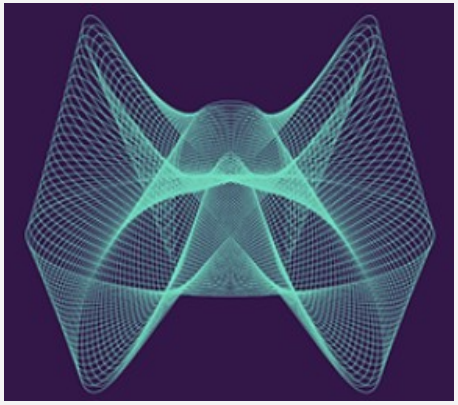
Agenda

Durante este curso veremos los siguientes temas

- Tipos de Blockchain
- Aplicaciones Web 3.0
- Tipos de Tokens - Solidity
- Smart Contracts
- Ambiente de ejecución
- Taller - Ejemplo



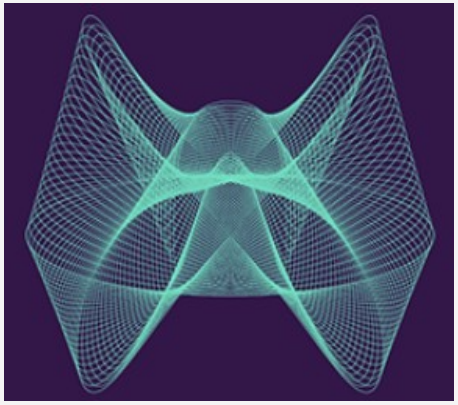
Objetivos



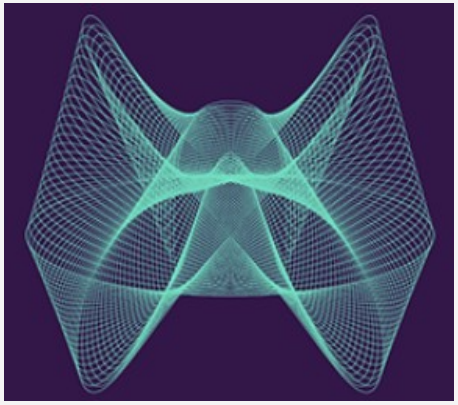
Objetivos

Al finalizar este módulo, los participantes estarán en capacidad de:

- Identificar las principales diferencias entre las implementaciones de blockchain mas utilizadas actualmente
- Entender la arquitectura general de una solución blockchain
- Entender y aplicar los conceptos principales en el desarrollo de una aplicación web 3.0 (Dapp)
- Construir un contrato inteligente para la creación de un Token

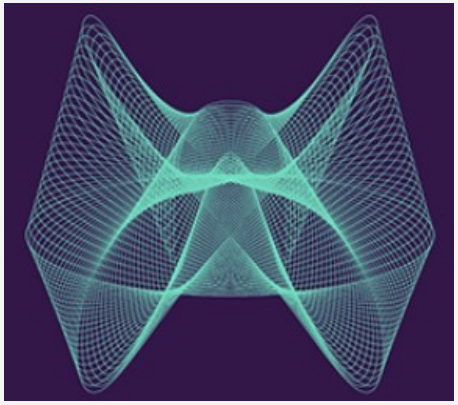


Tipos de sistemas Blockchain



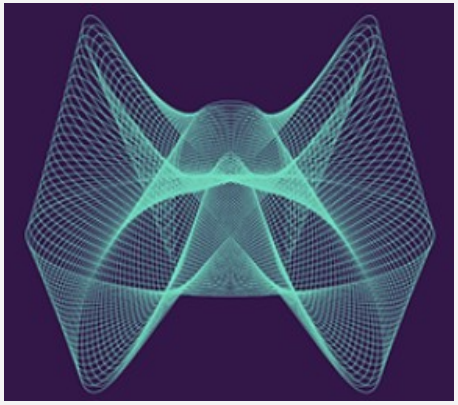
Sistema Blockchain (BlockChain System - BCS)

- Una red de máquinas llamadas **nodos**
- Una estructura de datos que implementa el libro contable, replicada a través de la red de bloques.
- Los nodos que tienen una réplica completa del libro contable se llamada ***full nodes***
- Un protocolo de red que define los derechos, responsabilidades y medios de comunicación, verificación y validación, así como los mecanismos de consenso de los nodos.
- Lo anterior incluye la autorización y autenticación de transacciones, mecanismos de adición de nuevos bloques y mecanismos de incentivos.



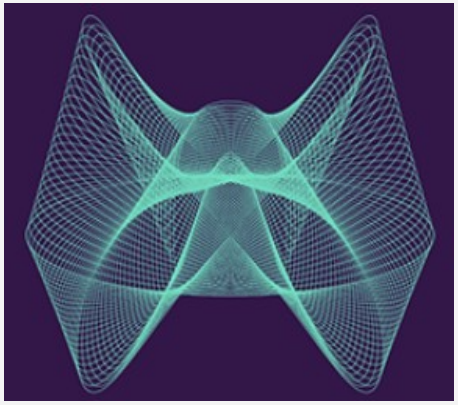
Blockchain público

- Está disponible para todos
- Está hecho por personas y para las personas
- Nadie está a cargo del BCS y cualquiera puede participar en los procesos de lectura, escritura y auditoría
- Se cuenta con reglas estrictas para garantizar el BCS de actores maliciosos
- Todas las decisiones se toman con base en algoritmos complejos de consenso
- Son BCS costosos computacionalmente hablando



Blockchain público

- Los nodos pueden unirse e irse cuando lo deseen
- Todos los ***full nodes*** en la red pueden verificar cada nueva pieza de datos adicionada a la lista de bloques
- Se incluye un mecanismo de incentivo como compensación del esfuerzo por mantener operativa la red
- Ejemplos: Bitcoin, Ethereum



Blockchain Privado

BCS operado por una organización de forma privada

Se cuenta con un administrador que vela por los permisos y las identidades

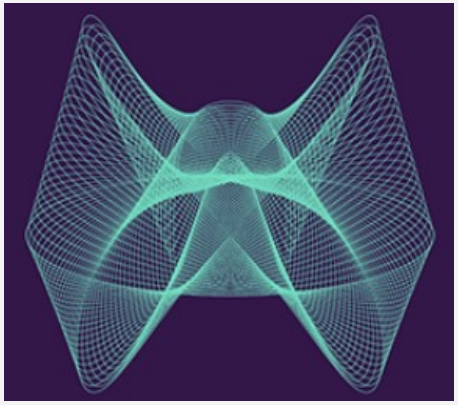
El mecanismo de consenso depende de la unidad central de administración que puede delegar o no la validación de la cadena entre los participantes

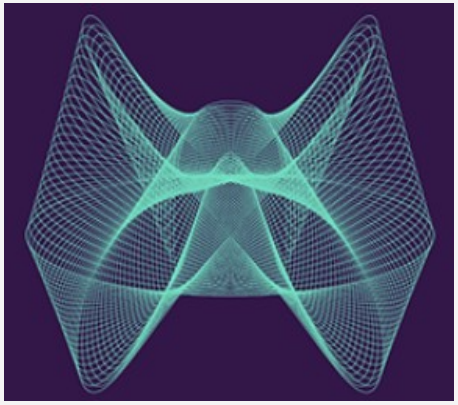
Comparada con un BCS público es mucho mas barato y rápido dado que no se requiere tanto consumo de energía buscando el consenso

Comparada con un BCS público puede ser menos seguro

Ejemplos: Bankchain, Medichain

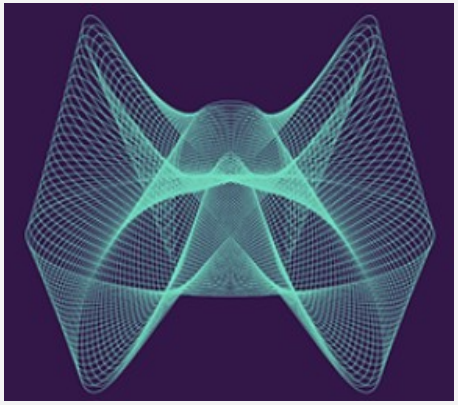
Desarollo de Soluciones Blockchain





Blockchain Consorcio

- BCS operado por un conjunto de organizaciones
- No se deja en manos de una sola entidad la administración de la cadena
- Se evita un punto único de falla
- Es una combinación ideal entre una privada y una pública
- Ejemplos: R3 (Corda), Energy Web Foundation (EWF)



Arquitectura Blockchain por Capas

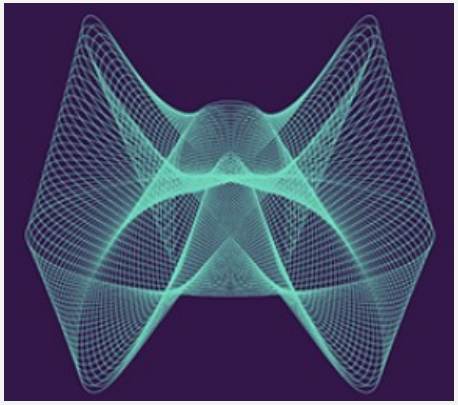
Capa 2

Capa 2

Capa 1

Capa 1

Capa 0

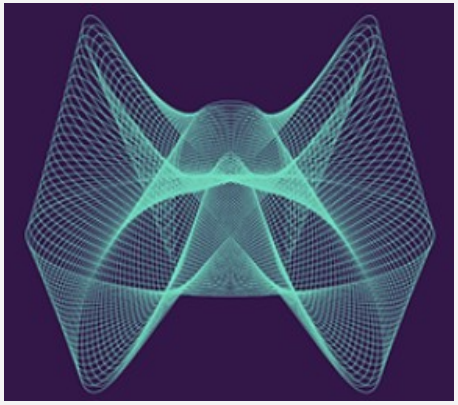


Arquitectura Blockchain por Capas

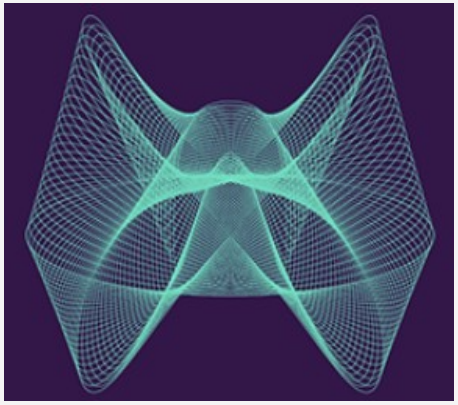
Capa 2: Lightning Network, Polygon

Capa 1: Bitcoin, Ethereum

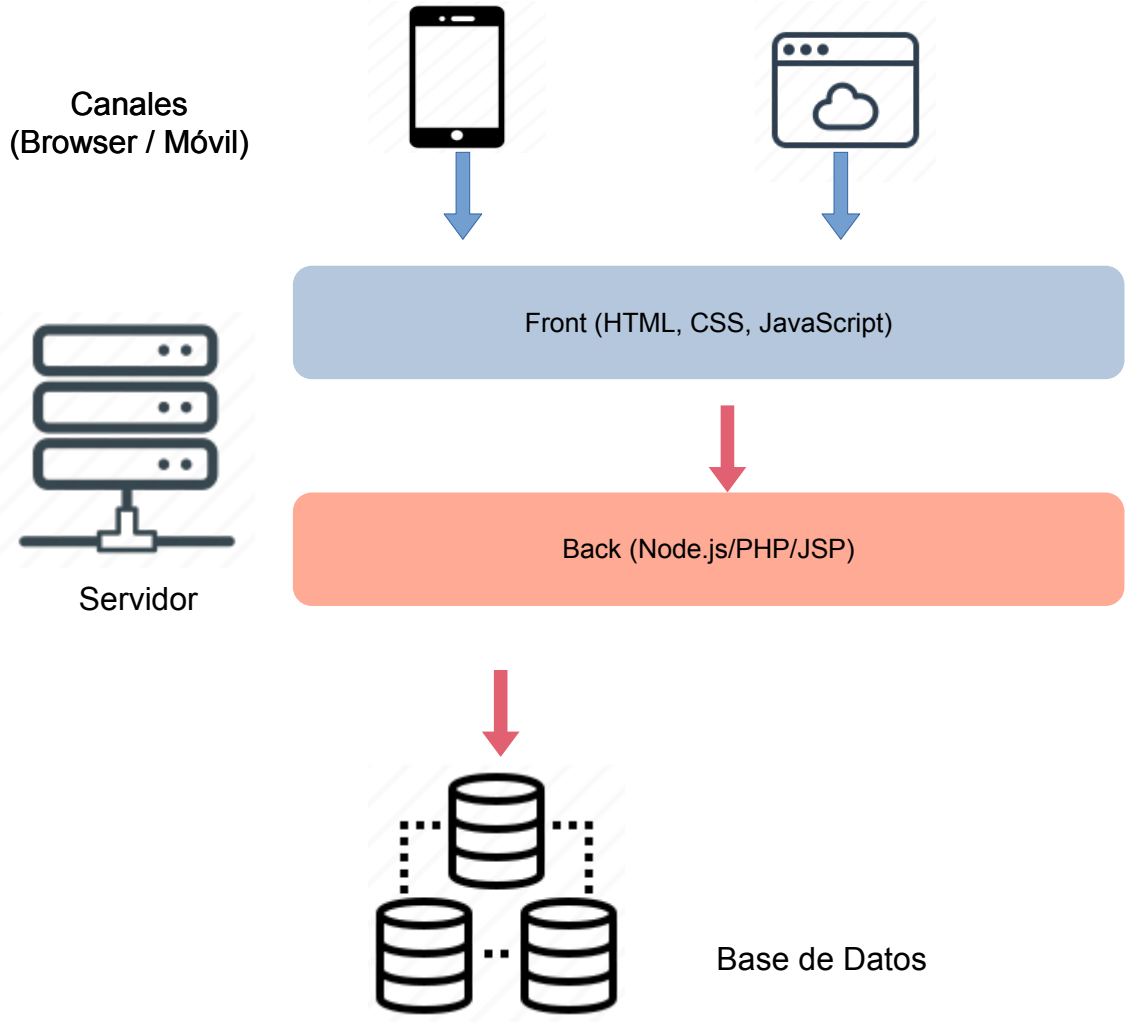
Capa 0: Polkadot

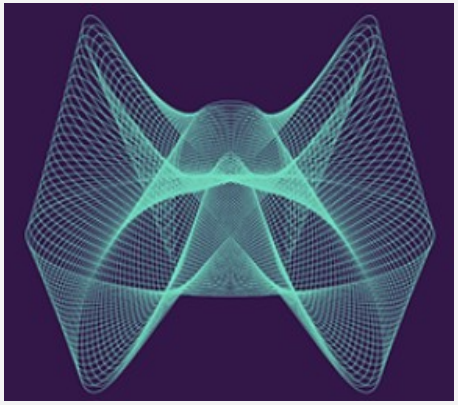


Aplicaciones Web 3.0

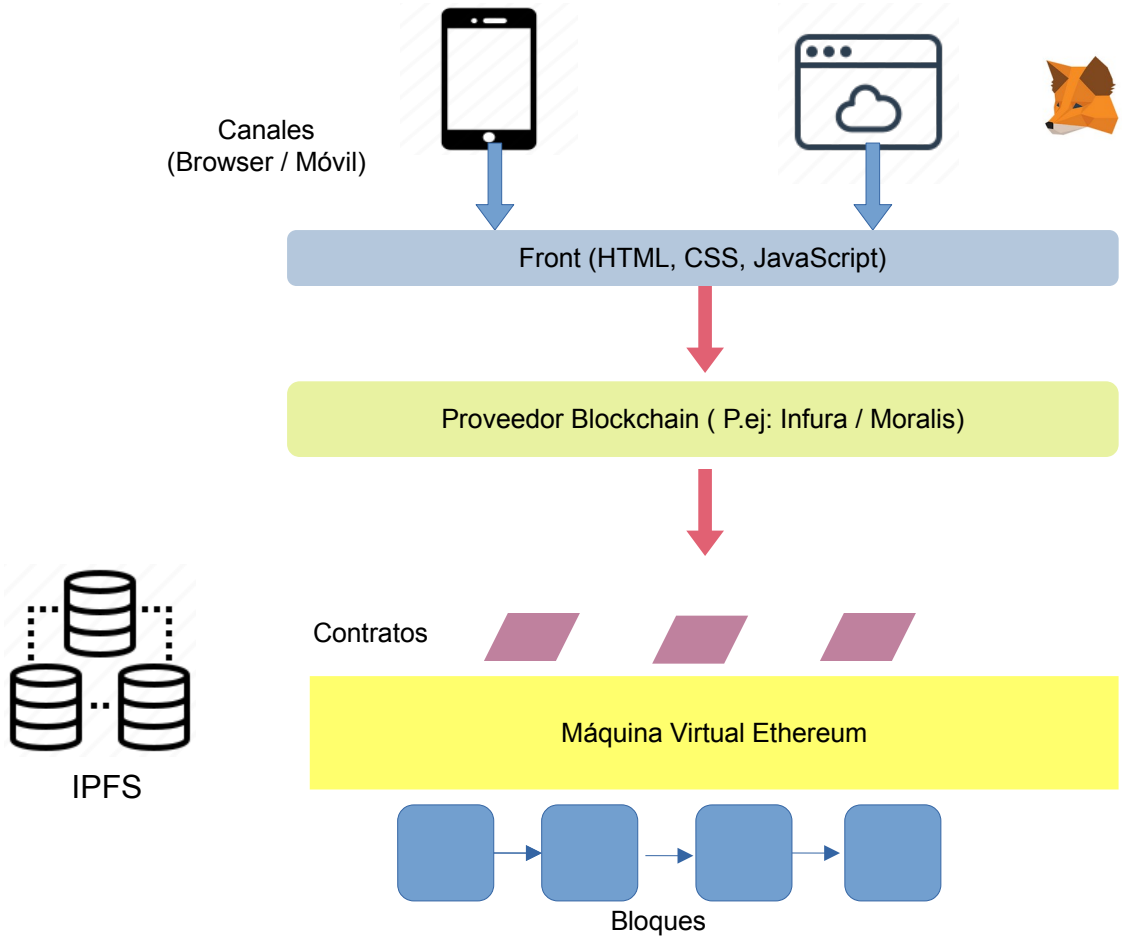


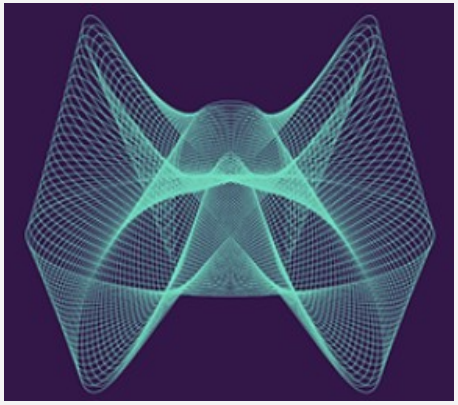
Arquitectura de una aplicación web tradicional



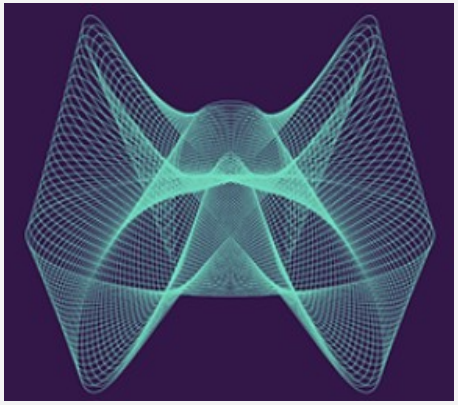


Arquitectura de una aplicación web 3.0





Tipos de Tokens - Solidity



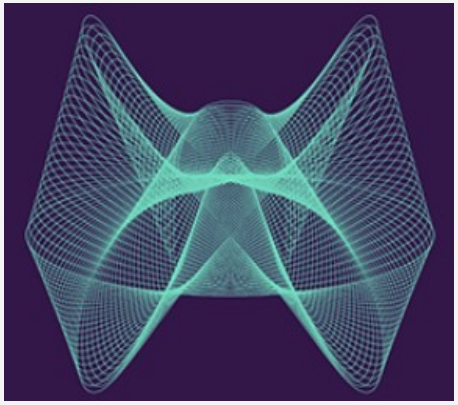
Ethereum Request for Comments - ERC

Estándares propuestos por la comunidad a nivel de aplicación.

Pueden definir tipos de tokens, librerías o formatos entre otras cosas.

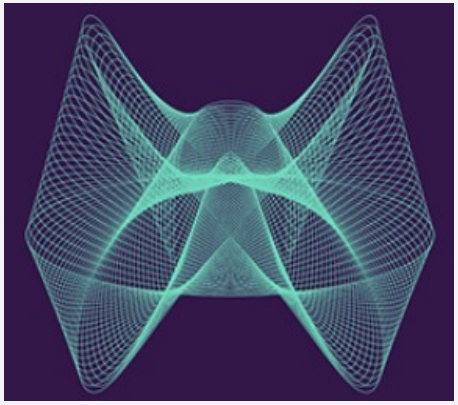
Los ERC mas conocidos son:

ERC-20
ERC-721
ERC-1155



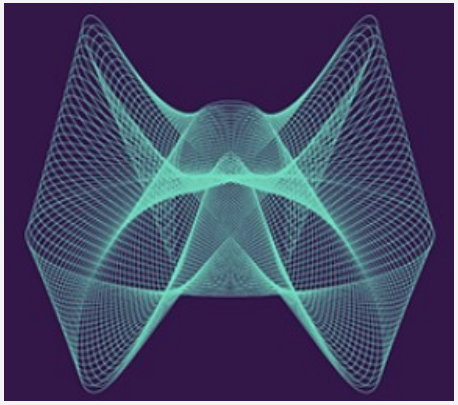
Ethereum - ERC 20

- Representa un Token intercambiable - (Fungible Token)
- Utilizado para manejo de cripto monedas estables y no estables
- Cada token es igual a otro token
- Se puede conocer el total de tokens en la red



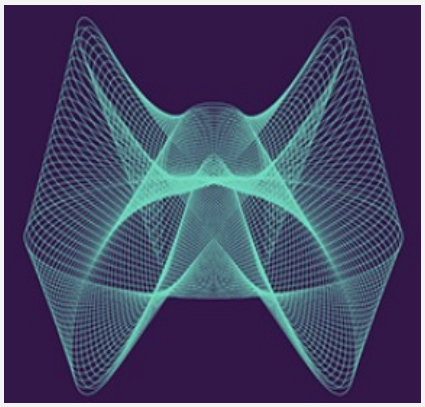
Ethereum - ERC 721

- Representa un Token no intercambiable - (Non Fungible Token)
- Utilizado para manejo de criptoactivos
- Cada token tiene un valor propio por lo que no tiene sentido intercambiarlo por otro token
- Cada token define su propio valor y puede ser comprado y vendido a diferentes precios
- Manejo de colecciones

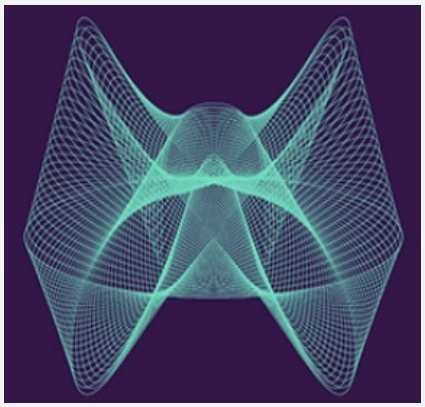


Ethereum - ERC 1155

- Representa un Token mixto
  - Se puede comportar o contener FTs y NFTs
- En una misma transacción se pueden crear múltiples instancias de un NFT
- Permite reducir costos asociados a las transacciones y a la computación asociada a cada transacción



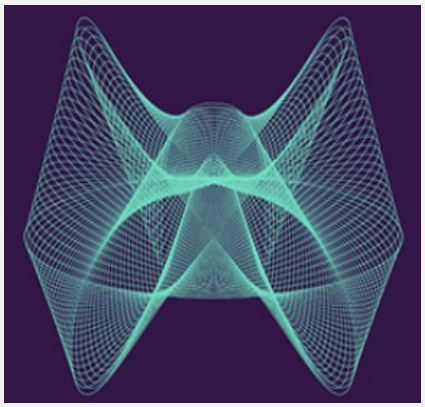
- Smart Contracts



Ethereum

Ethereum virtual machine

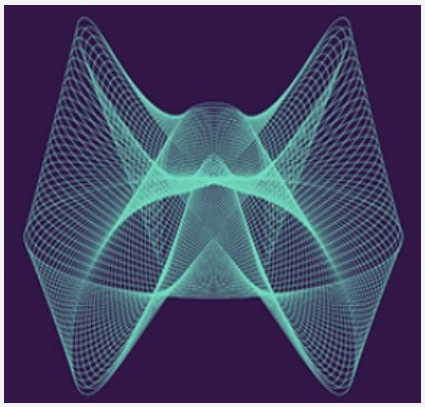
- Stack
- Memoria
- Alamacenamiento
- Variables de ambiente
- Logs
- Lenguajes específicos para máquina virtual
  - Serpent, Solidity, LLL



Ethereum

Smart Contracts

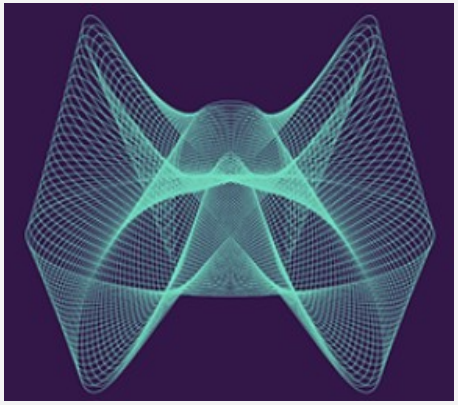
- Residen dentro de la cadena de Ethereum
- Tienen su propia cuenta (dirección y balance)
- Pueden enviar mensajes y recibir transacciones
- Pueden ser activados y desactivados a través de transacciones
- También deben pagar un fee por almacenamiento y procesamiento



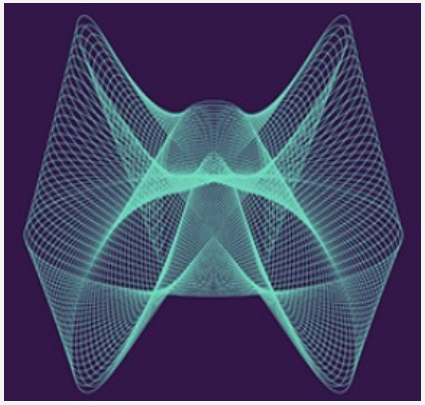
Ethereum

Smart Contracts

- Application Binary Interface - ABI
  - Cada contrato tiene asociado un ABI que intermedia entre el código compilado y la EVM
- Un ABI tiene
  - Nombres de las funciones
  - Entradas y salidas
  - Eventos y parámetros

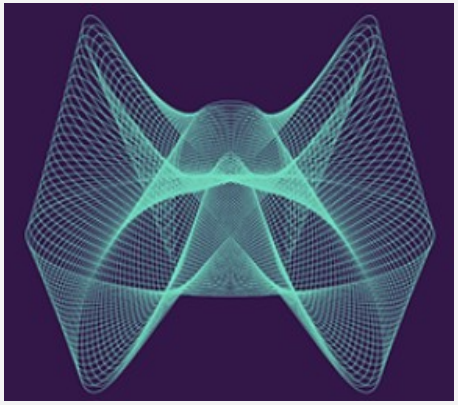


Ambiente de Ejecución

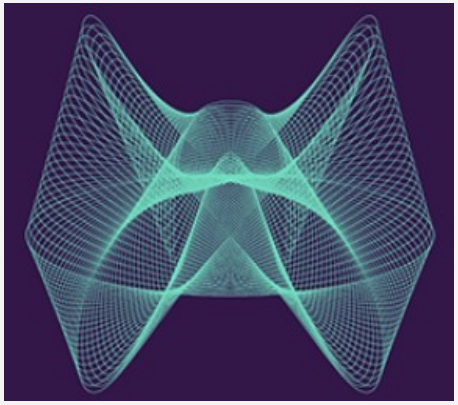


Desarrollo y ejecución de un Smart Contract

1. Programar smart contract en Solidity (Editor Remix / OpenZeppelin)
2. El contrato se compila (ABI / código EVM)
3. Se genera una transacción para crear el desplegar el contrato en la red
4. Se obtiene una dirección asignada al contrato desplegado

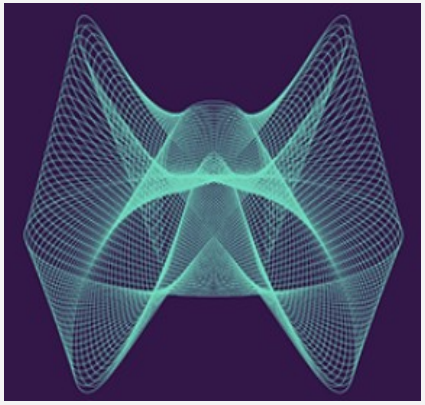


Taller - Ejemplo



Taller

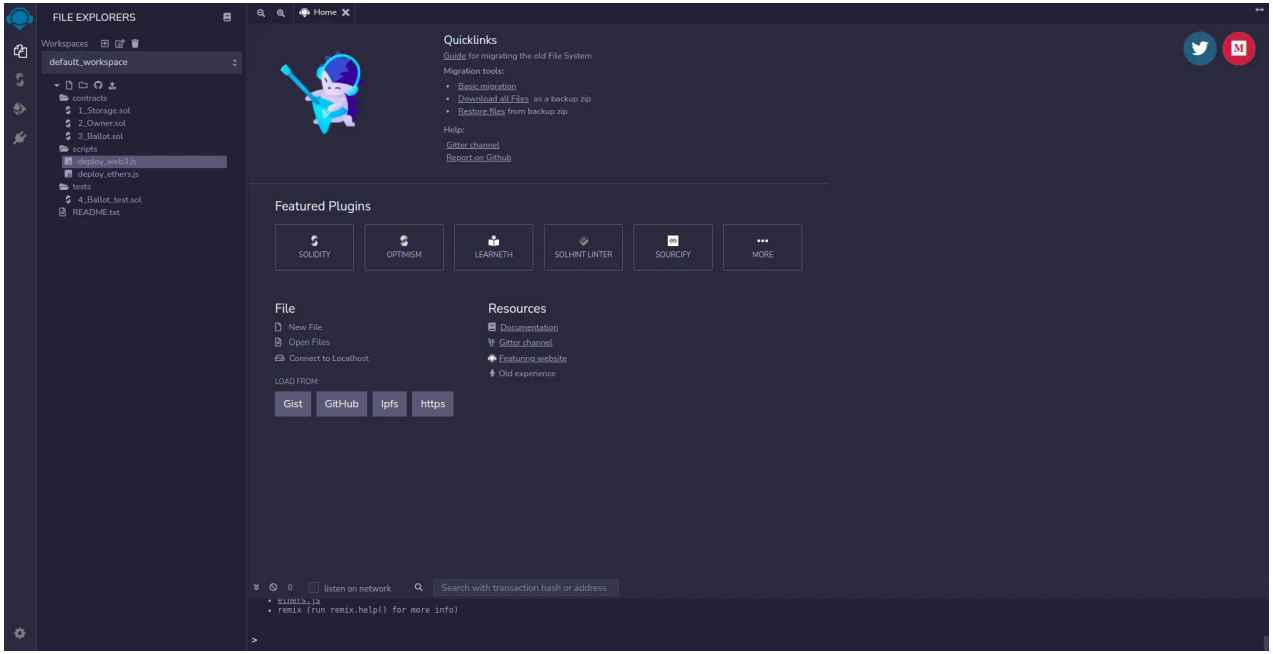
Desarrollar un Smart Contract que acumula dinero y permite que le transfieran desde una cuenta y transferir a una cuenta.

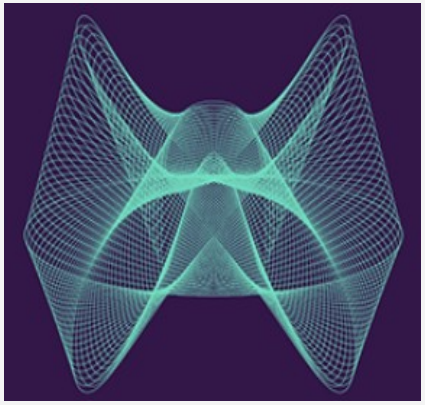


Taller - Ejemplo

Programar smart contract en Solidity
(Editor Remix)

http://remix.ethereum.org





Taller - Ejemplo

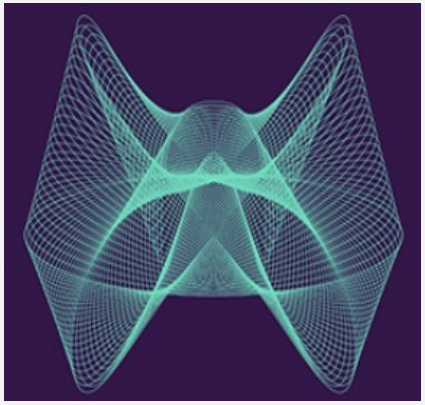
Programar smart contract en Solidity
(Editor Remix)

http://remix.ethereum.org

```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.4;
```

Pragma solidity define la versión de compilador a utilizar

SPDX expresa el tipo de licencia

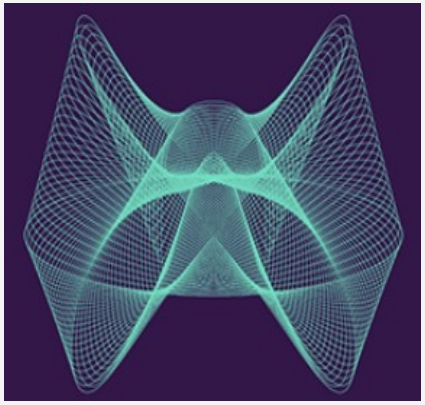


Programar smart contract en Solidity
(Editor Remix)

http://remix.ethereum.org

```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.4;

contract MAPI2 {

}
```

Contract es la unidad básica de desarrollo y ejecución



Taller - Ejemplo

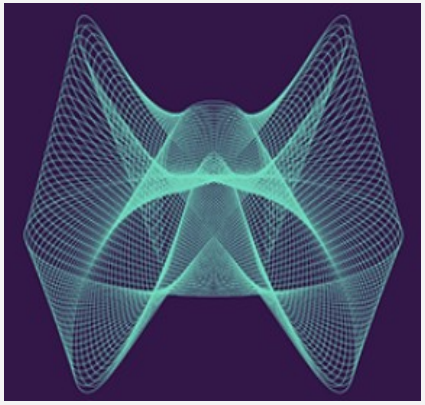
Programar smart contract en Solidity
(Editor Remix)

http://remix.ethereum.org

```
contract MAPI2 {

    address public propietario;
    uint256 public saldo;
```

Un contrato puede tener datos que se registran en Blockchain

El atributo público lo hace visible a quien desee consultar sus valores



Taller - Ejemplo

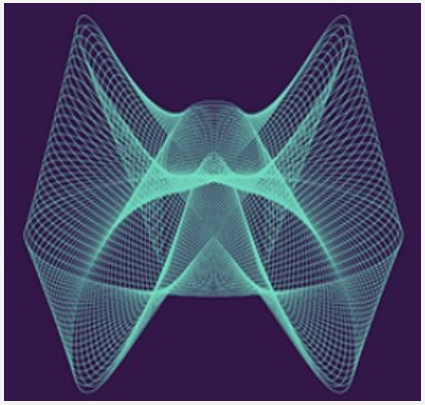
Programar smart contract en Solidity
(Editor Remix)

http://remix.ethereum.org

```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.7;

contract MAPI2 {

    address public propietario;
    uint256 public saldo;


    constructor(){
        propietario = msg.sender;
        saldo = 0;
    }
```

El constructor se ejecuta cuando se despliega el contrato en la red Blockchain

msg es una palabra reservada que se refiere a quien envía y recibe una transacción



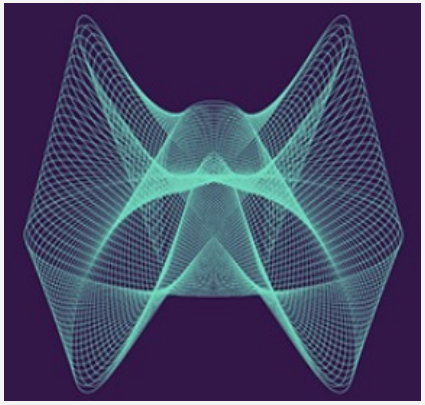
Programar smart contract en Solidity
(Editor Remix)

```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.7;

contract MAPI2 {

    address public propietario;
    uint256 public saldo;


    constructor(){
        propietario = msg.sender;
        saldo = 0;
    }

    function consignar (uint cantidad) public {
        saldo += cantidad;
    }
```

Un contrato puede tener funciones para modificar los atributos del contrato

uint : 256 bits o 32 bytes

public: la función puede ser consultada por todo el que pueda verla



Taller - Ejemplo

Programar smart contract en Solidity
(Editor Remix)

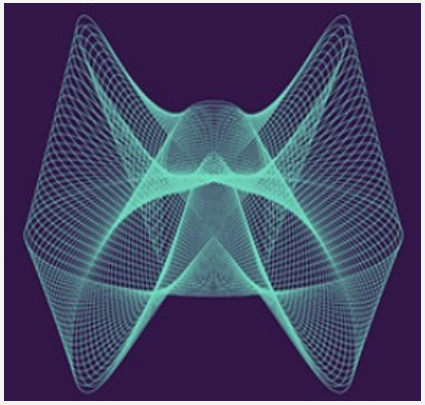
```
contract MAPI2 {

    address public propietario;
    uint256 public saldo;


    constructor(){
        propietario = msg.sender;
        saldo = 0;
    }

    function consignar (uint cantidad) public {
        saldo += cantidad;
    }

    receive() payable external {
        saldo += msg.value;
    }
```

Un contrato puede recibir ether proveniente de otras cuentas. Para ello debe implementar la función **receive**

payable : Autoriza al destinatario de la transacción a recibir ethers

external: La función puede ser llamada desde una cuenta externa

msg.value: El valor asociado a la transacción



Taller - Ejemplo

Programar smart contract en Solidity
(Editor Remix)

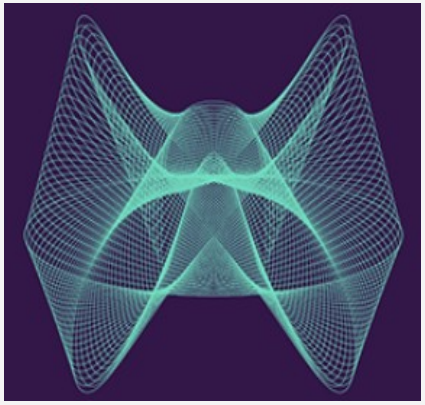
```
function transferir (uint cantidad, address payable destino) payable public {
    require(msg.sender == propietario, "solo el propietario puede transferir");
    require(cantidad <= saldo, "fondos insuficientes");
    destino.transfer(cantidad);
    saldo -= cantidad;
}
```

No es recomendable dejar dinero en un contrato sin poder retirarlo.
Lo mejor es transferirlo a una cuenta

Address : 20 bytes para registrar una dirección

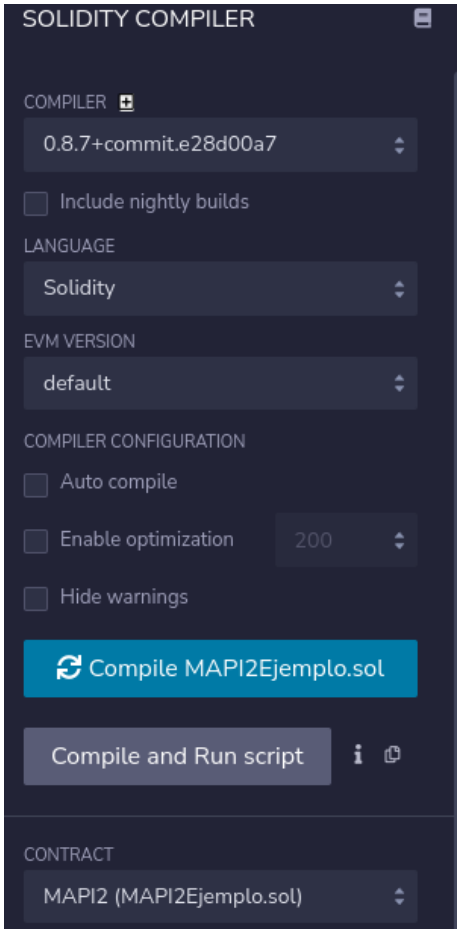
require: Precondición que debe cumplirse para continuar con la ejecución

transfer: Traspasa dinero a una cuenta, en este caso la cuenta destino

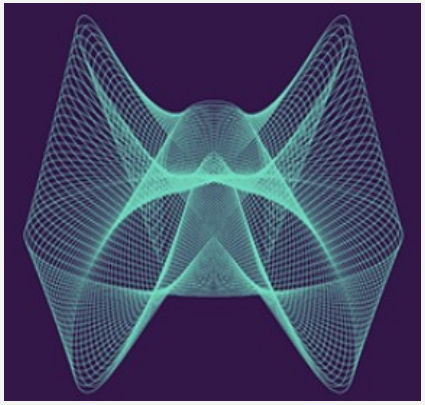


Taller - Ejemplo

Programar smart contract en Solidity
(Editor Remix)



El primer paso es compilar el contrato

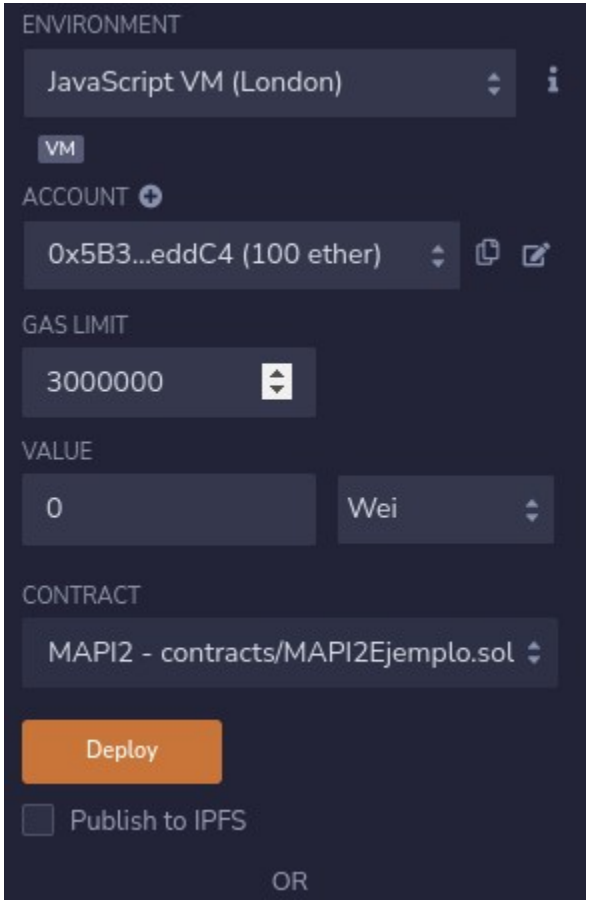


Taller - Ejemplo

Programar smart contract en Solidity
(Editor Remix)

El segundo paso es desplegar el contrato

Inicialmente se despliega localmente. Remix Ofrece un simulador local.

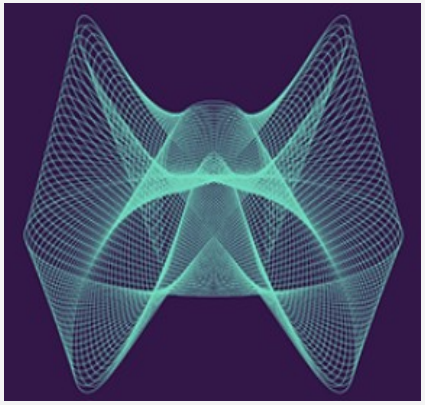


Red de pruebas

Cuentas de pruebas

Valor asociado a la transacción

Contrato a desplegar



Taller - Ejemplo

Programar smart contract en Solidity
(Editor Remix)

Ahora vamos a desplegar el contrato en una red de pruebas Blockchain

Rinkeby Test Network

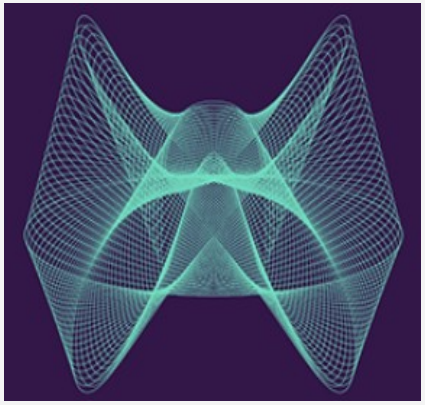
Red de pruebas de Ethereum

Rinkeby Faucet

Provee dinero (Ethers – ficticios) para realizar pruebas

Metamask

Billetera con las cuentas y dinero

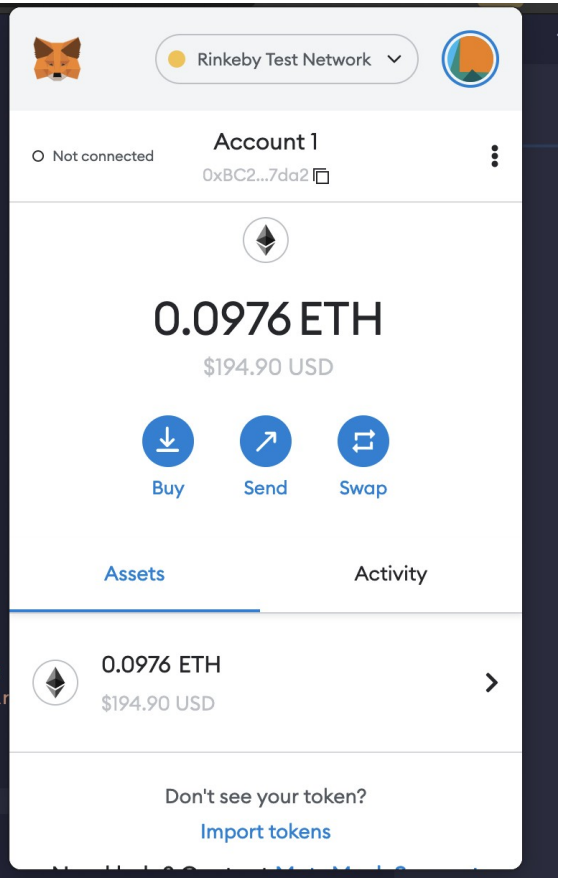


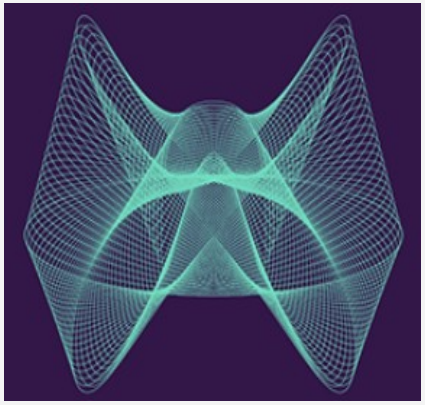
Taller - Ejemplo

Programar smart contract en Solidity
(Editor Remix)

MAPI2Ejemplo.sol

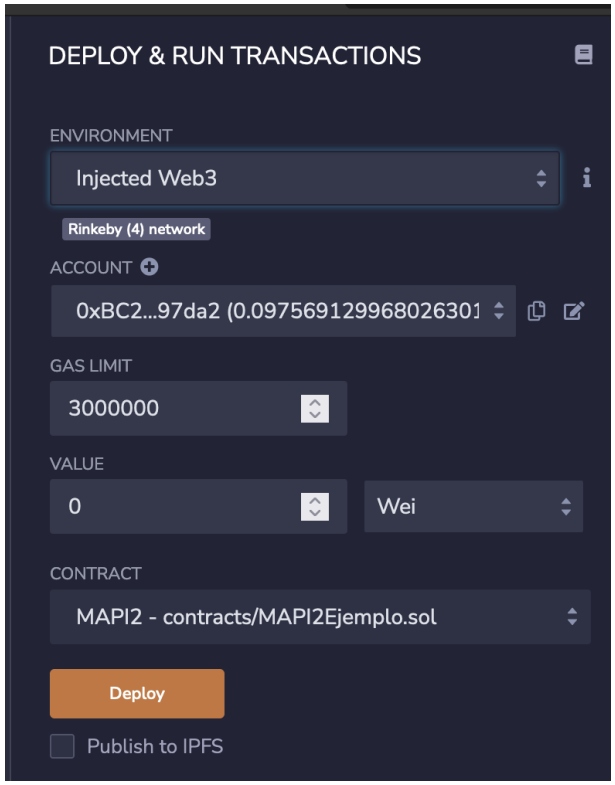
```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.7;

contract MAPI2 {

    address public propietario;
    uint256 public saldo;


    constructor(){
        propietario = msg.sender;
        saldo = 0;
    }

    function consignar (uint cantidad) public {
        saldo += cantidad;
    }

    receive() payable external {
        saldo += msg.value;
    }


    function transferir (uint cantidad, address payable destino) payable public
        require(msg.sender == propietario, "solo el propietario puede transferir
        require(cantidad <= saldo, "fondos insuficientes");
        destino.transfer(cantidad);
        saldo -= cantidad;
    }
```

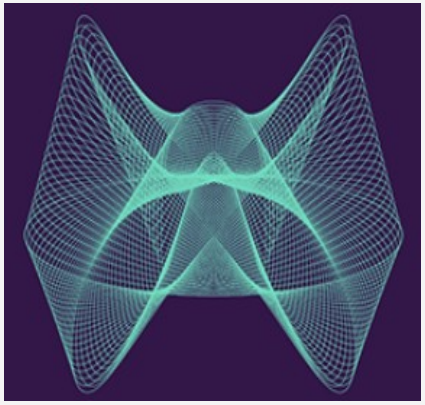




Taller - Ejemplo

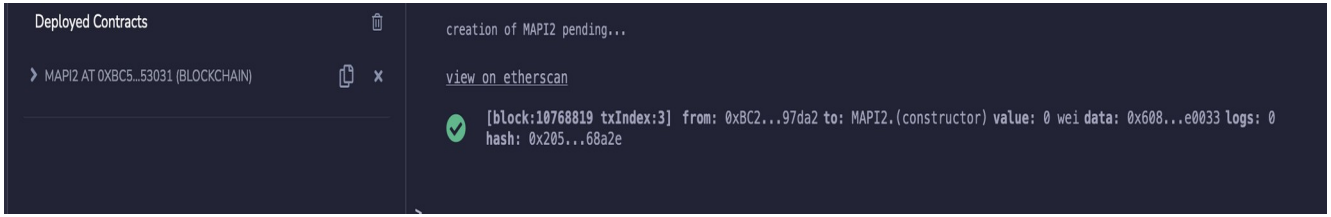
Programar smart contract en Solidity
(Editor Remix)

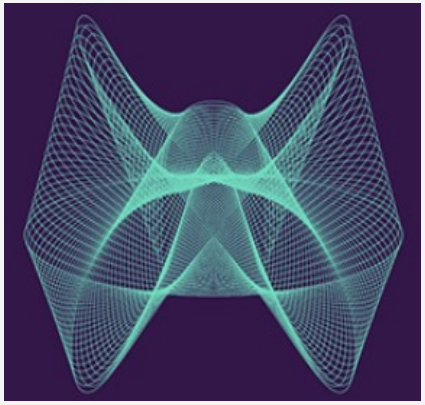




Taller - Ejemplo

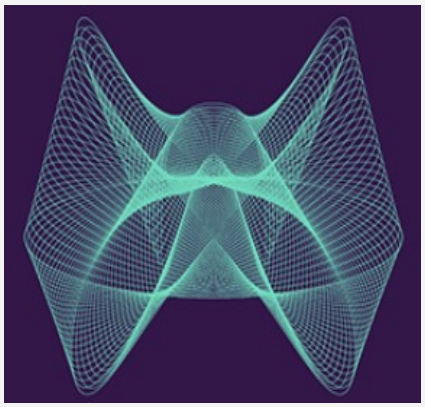
Programar smart contract en Solidity
(Editor Remix)





Taller - Ejemplo

Programar smart contract en Solidity
(Editor Remix)



Programar smart contract en Solidity (Editor Remix)

1- Cargar dinero en una cuenta Metamask

2- Enviar ether al contrato desde una cuenta en MetaMask

3- Enviar ether desde el contrato a una cuenta en Metamask