



Taller 3

Introducción a Solidity y Cairo,
Primeros pasos en el Desarrollo de
Ethereum y Starknet

Presentación:

Nadai - StarknetEs

Juan Carlos - EscuelaCryptoEs





Nadai
StarknetEs



Juan Carlos
EscuelaCryptoES



Indice

1. Historia de Solidity
2. Aspectos básicos de Solidity
3. Código en Solidity
 - 3.1 Tipos de Datos
 - 3.2 Variables, Estados, Funciones, Control de Flujo, Eventos, Herencia y Bibliotecas
4. Cairo como lenguaje comprobable
 - 4.1 Optimizaciones y crecimiento
 - 4.1 Diferencias con Solidity
 - 4.2 CVM y diferencias con EVM
5. Sierra y CASM
6. Código en Cairo
 - 7.1 Tipos de Funciones en Cairo en contratos de Starknet
7. Sintaxis básica
8. Casos de usos de Cairo
9. Despliegue en Remix Solidity - Cairo
10. Primeros pasos para desarrollar con Cairo en Starknet, recursos y ejercicios disponibles.



Historia de Solidity

Solidity fue propuesto en agosto de 2014 por Gavin Wood.

Posteriormente, el lenguaje fue desarrollado por el equipo de Solidity del proyecto Ethereum Core, dirigido por Christian Reitwiessner.

Solidity es el lenguaje principal en Ethereum, así como en otras blockchains privadas, como la blockchain orientada a empresas Hyperledger Fabric.

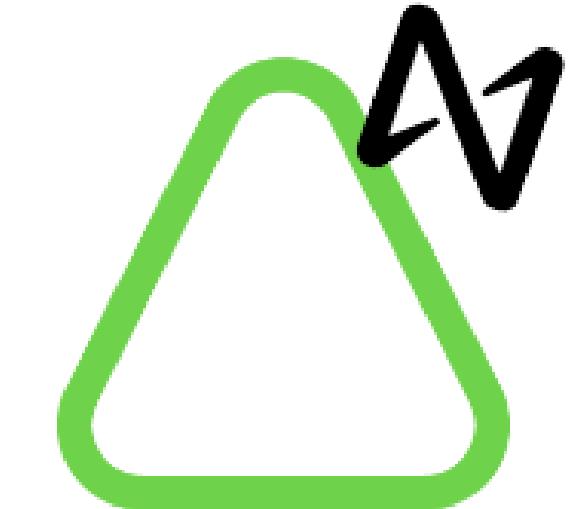
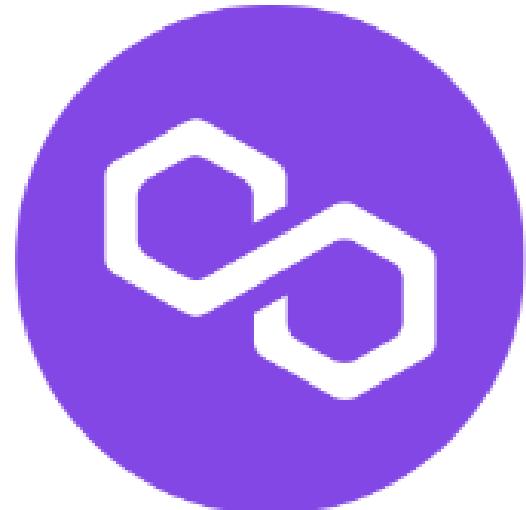
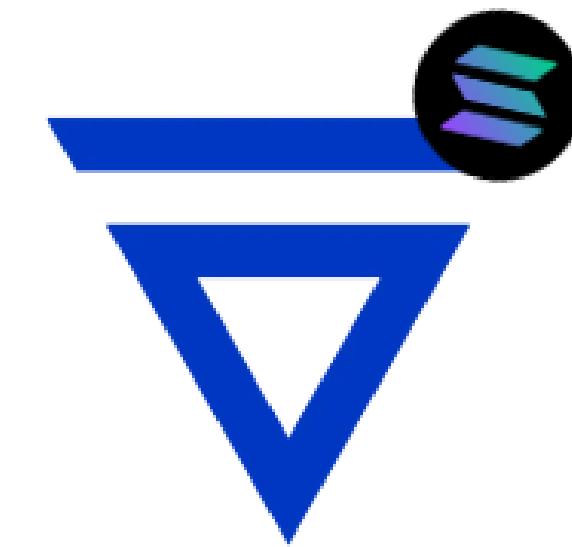
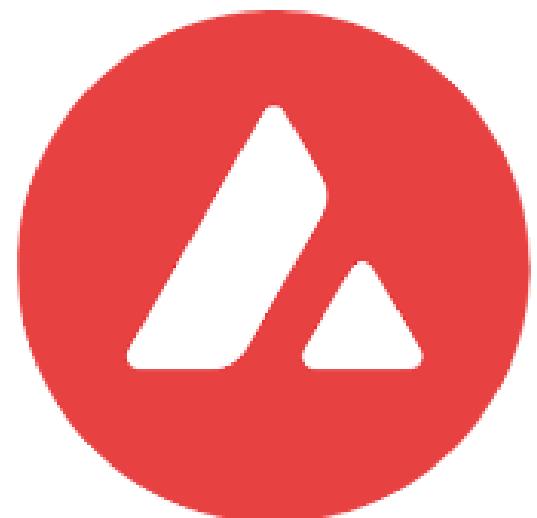
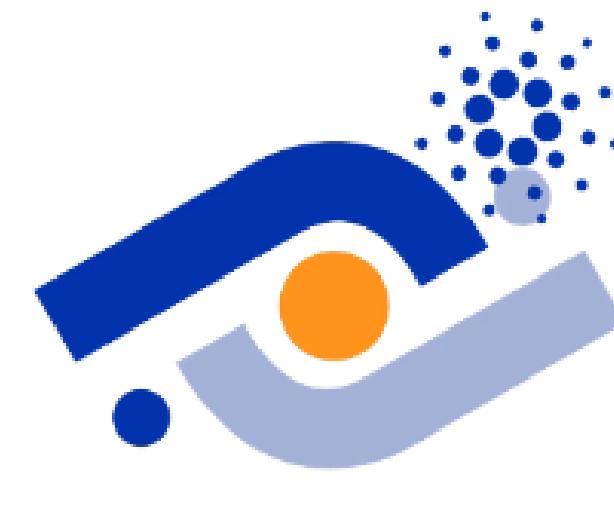
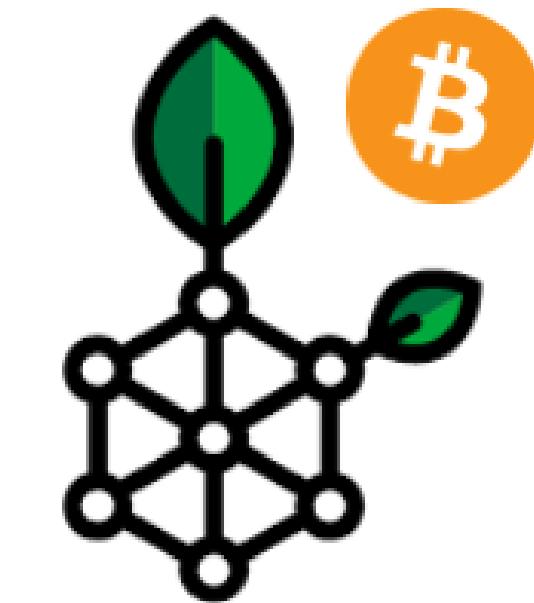
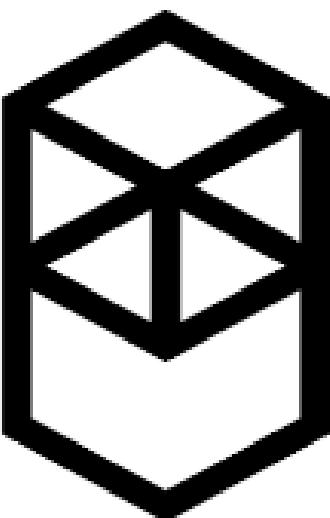
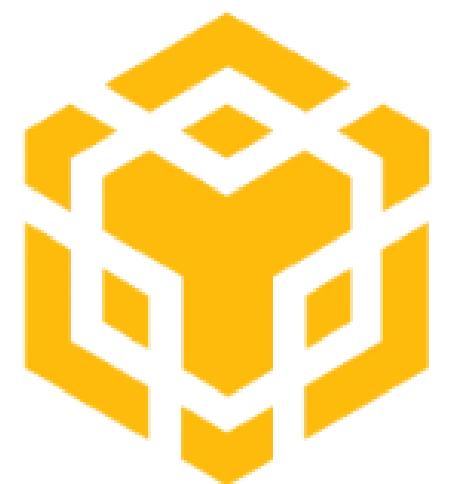
SWIFT implementó una prueba de concepto utilizando Solidity en Hyperledger Fabric.





- Lenguaje de alto nivel orientado a objetos
- *Curly-bracket language { }*
- Tipado estático
- Herencia
- Bibliotecas
- Librerías
- Etc...

Distintas chains
EVM Compatible



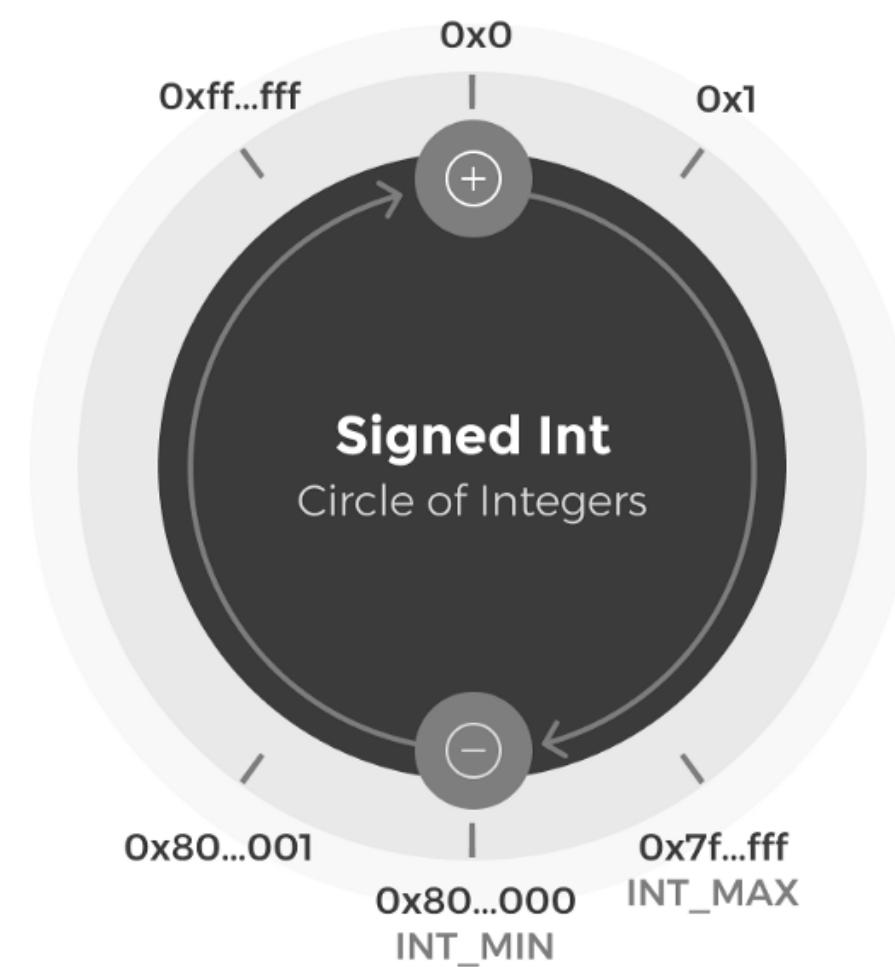
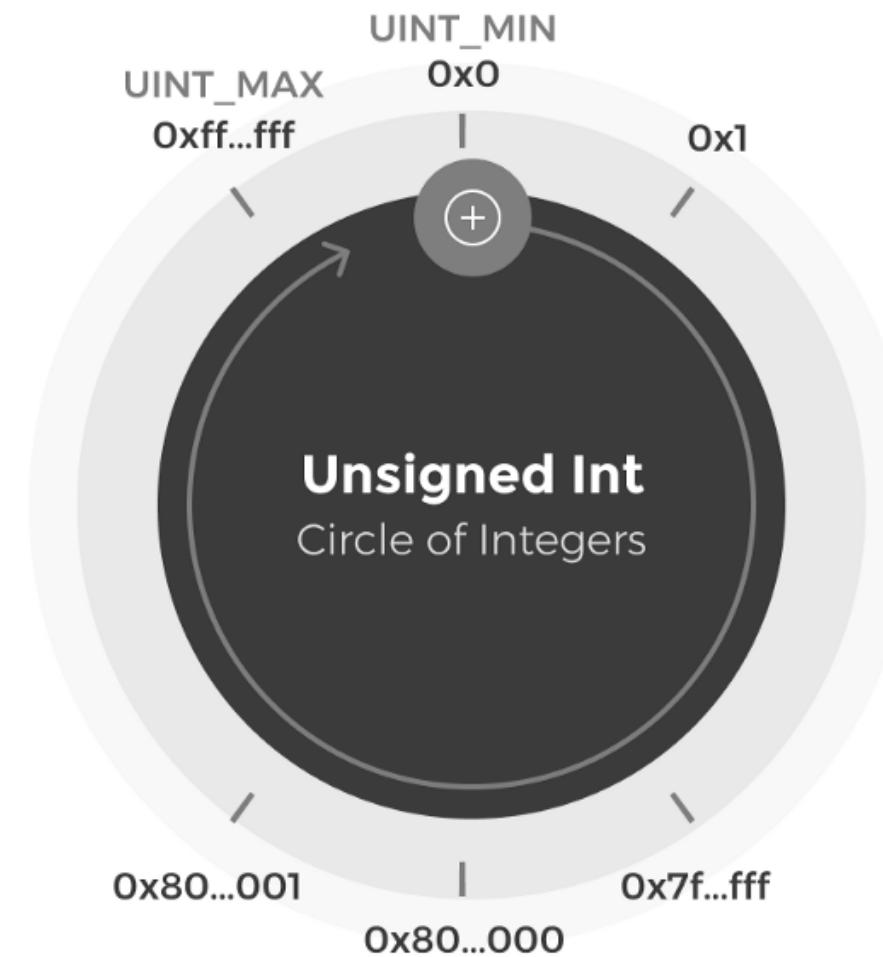
Tipos de datos en Solidity: Los ladrillos de la construcción

Solidity tiene varios tipos de datos que te permiten construir contratos complejos.



Enteros: uint y int

- **Un uint es un entero sin signo: solo valores positivos o cero.**
- **Un int puede ser negativo, cero o positivo.**



Enteros: uint y int

uint coffe = 1;



Enteros: uint y int

```
int coffe = -1;
```



Booleanos (bool)

Los interruptores lógicos:
Un booleano solo puede ser 'true' o 'false'.



Booleanos (bool)

bool redbull = true;



Booleanos (bool)

```
bool redbull = false;
```

Cadenas de caracteres (string)

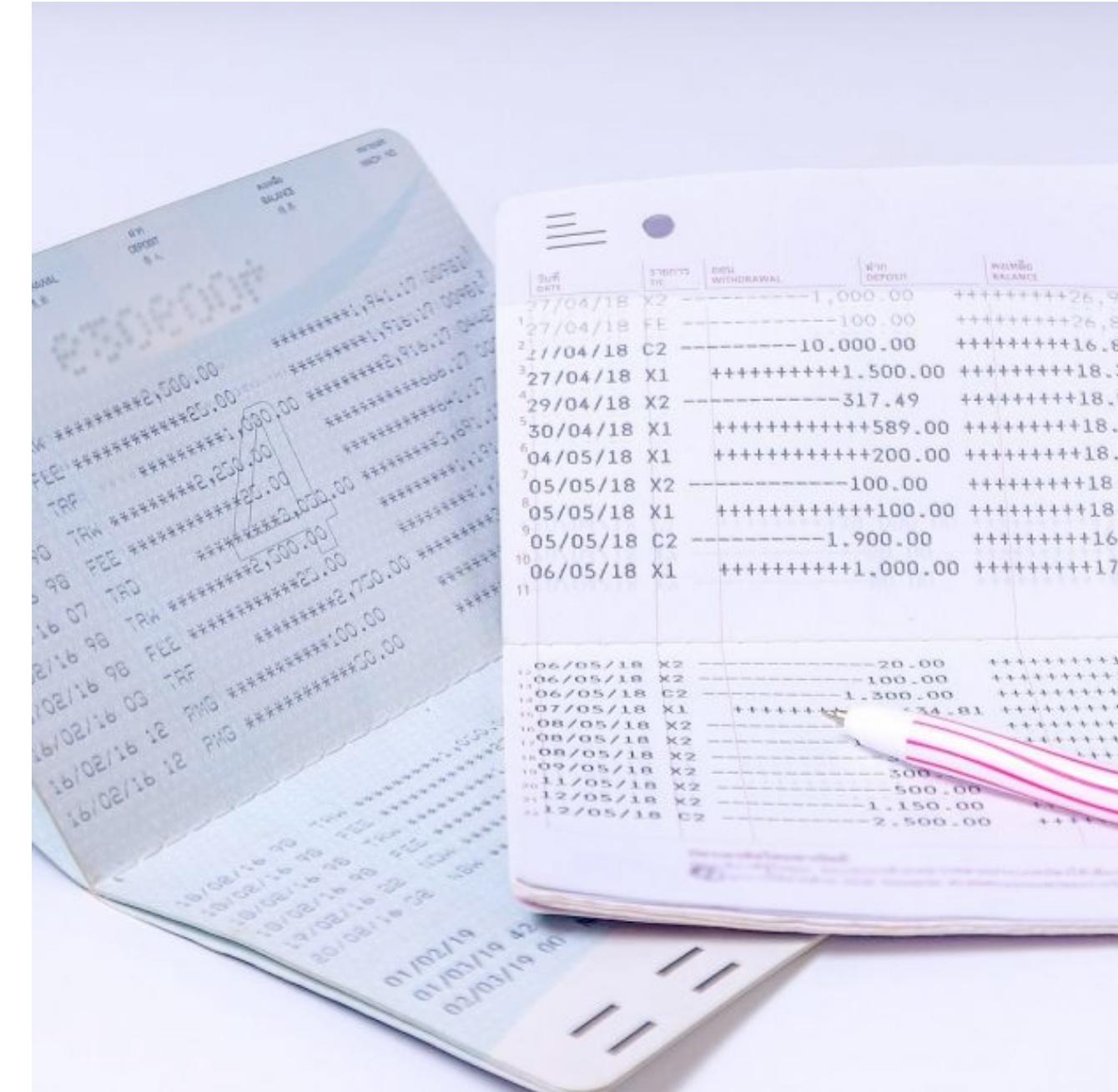
Las cadenas almacenan texto.

```
string necesito = "cafeina";
```



Direcciones (address)

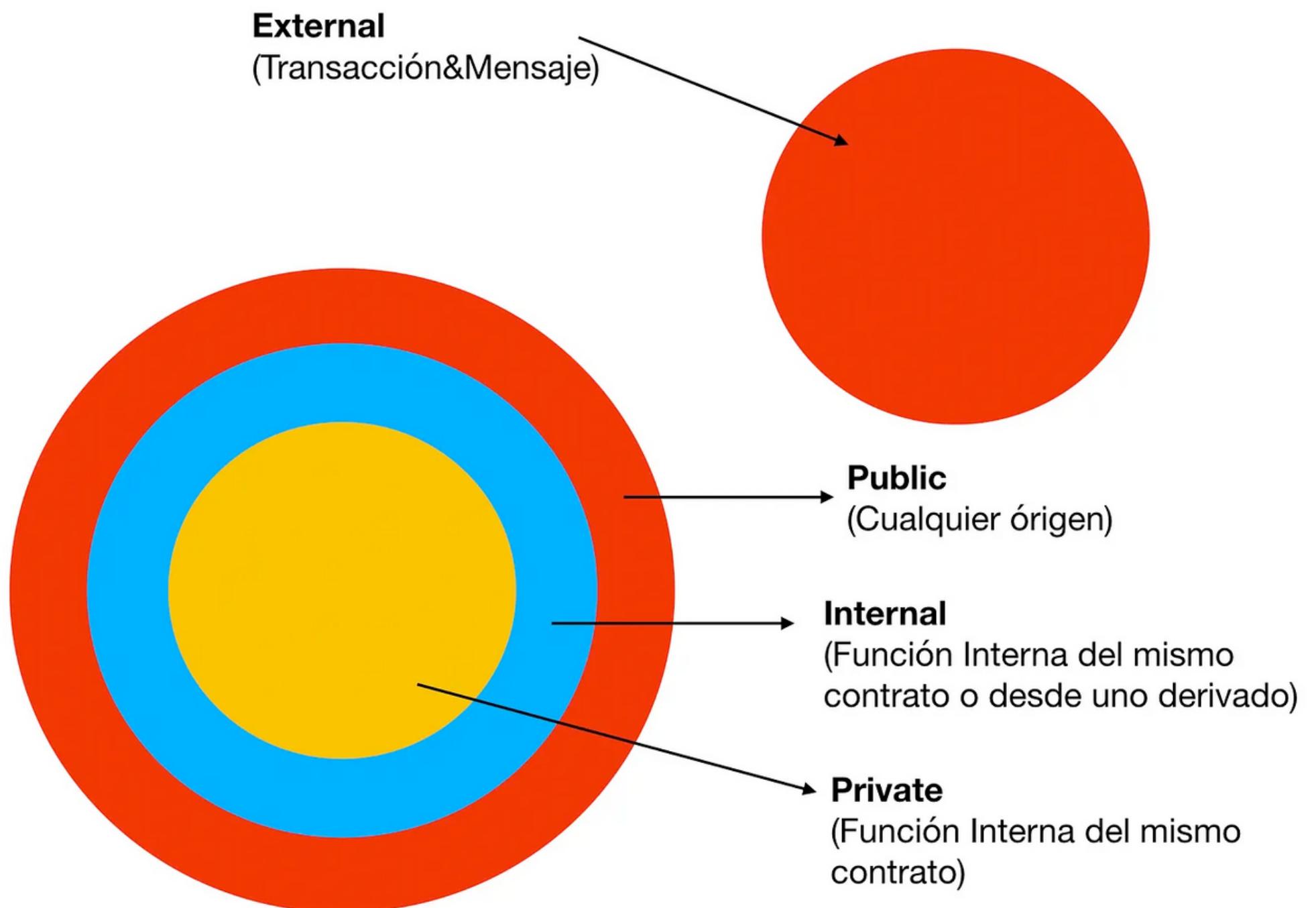
Una dirección Ethereum es como el IBAN de una cuenta de banco.



address donaciones = 0x6BB14696667429A82515863cd37aE31e2aDfe948

¿Quién puede verlo? Modificadores de Acceso

- **public:** Cualquiera puede verlo.
- **private:** Solo visible dentro del contrato.
- **internal:** Visible dentro del contrato y contratos derivados.
- **external:** Solo se puede acceder desde transacciones externas.

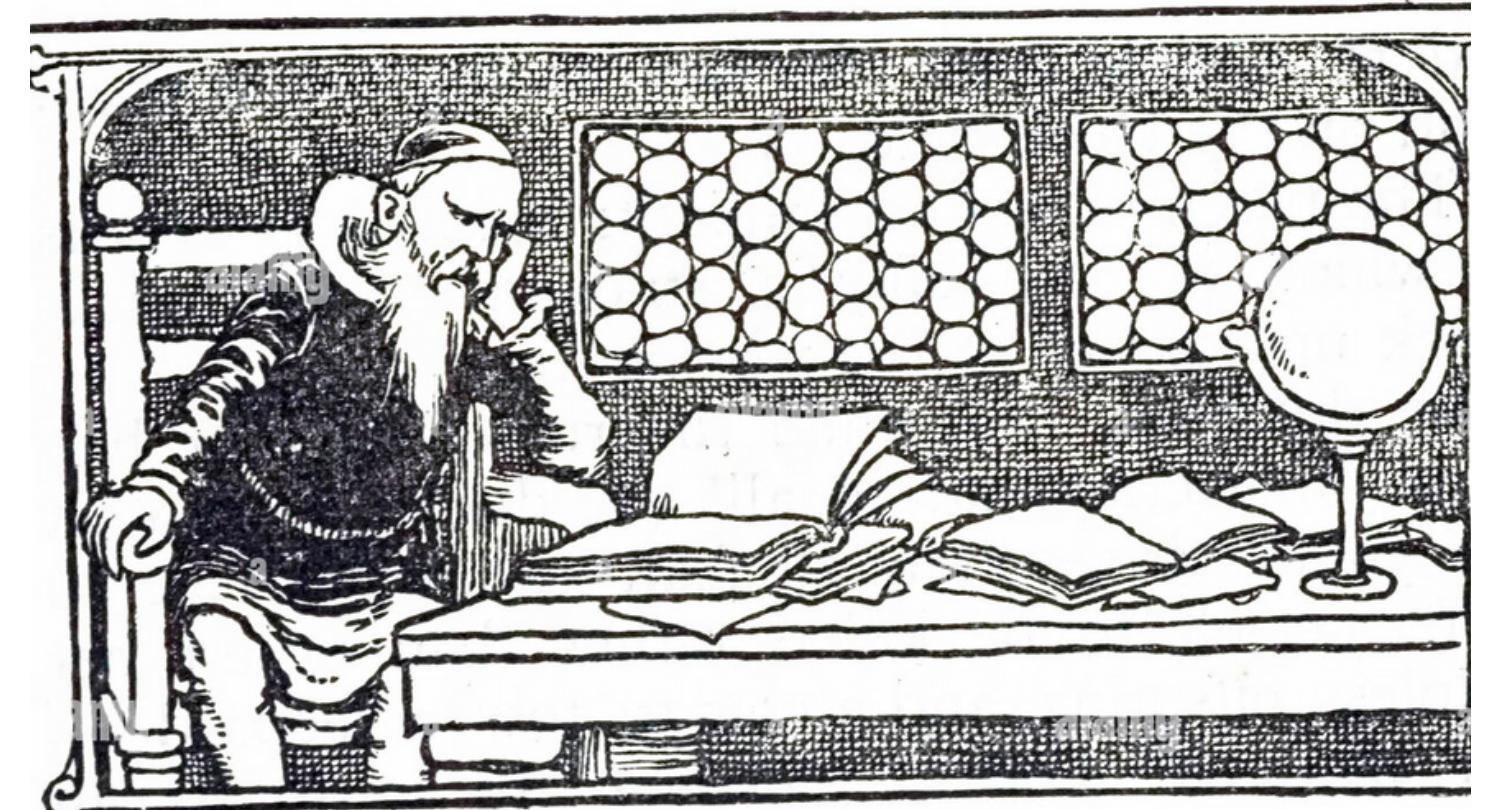


pure & view: Funciones que 'no tocan' los datos

- **pure**: No lee ni escribe datos. →



- **view**: Solo lee datos. →



PURE



```
function horasRestantesDePresentacionInterminable(uint256 diapositivasRestantes)
public pure returns(uint256 horas){
    horas = diapositivasRestante * 1; // Horas por diapositiva
}
```

VIEW



```
function horasRestantesDePresentacionInterminable(uint256 diapositivasRestantes) public view
returns(uint256 horas){
    uint256 horasPorDiapositiva = getHorasPorDiapositiva();
    horas = diapositivasRestantes * horasPorDiapositiva;
}
```

payable: ¡Funciones que reciben Ether!

payable:

Aceptan Ether junto con la llamada a la función.



PAYABLE



```
function sobornarAlPresentadorParaQueAcelere()
public payable{

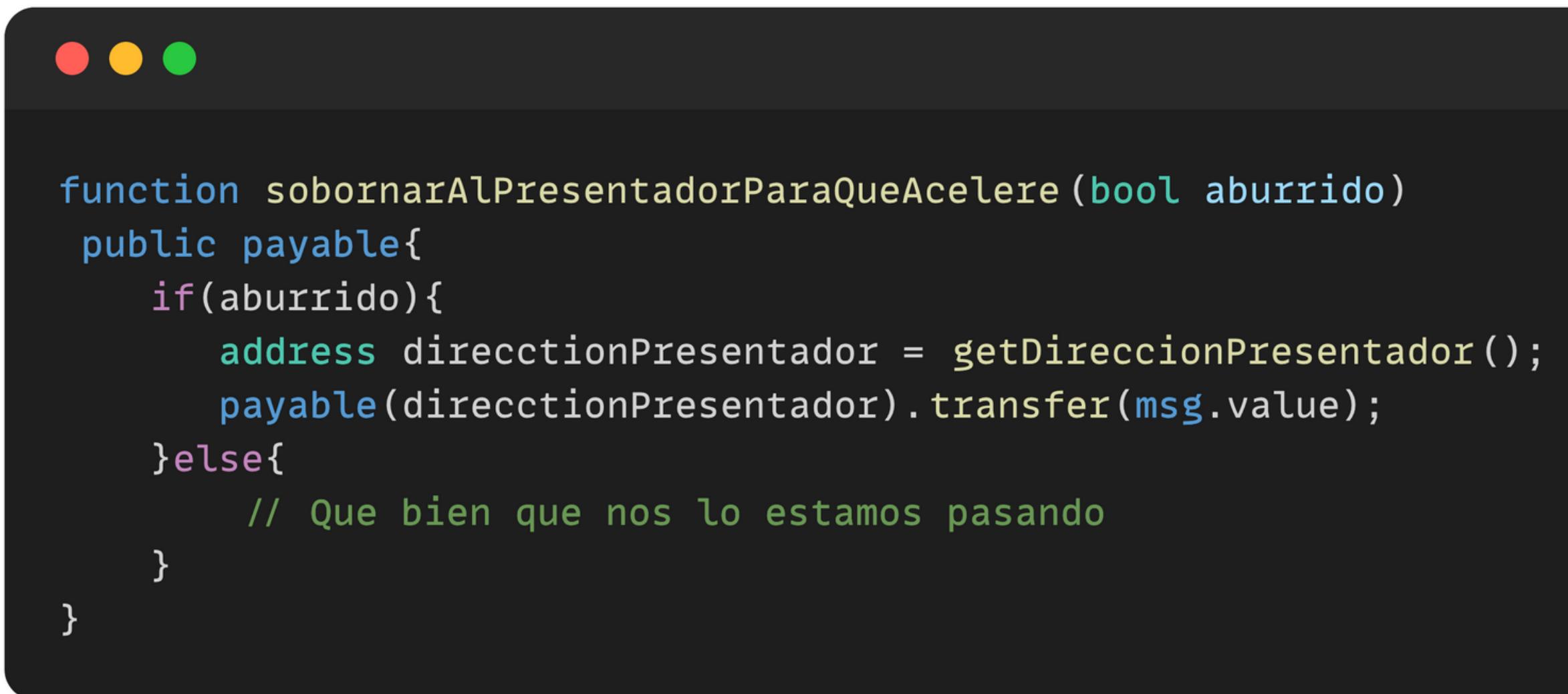
    address direcctionPresentador = getDireccionPresentador();

    payable(direcctionPresentador).transfer(msg.value);

}
```

if, else: Para tomar decisiones.

- **if:** Si la condición se cumple
- **else:** Si la condición no se cumple

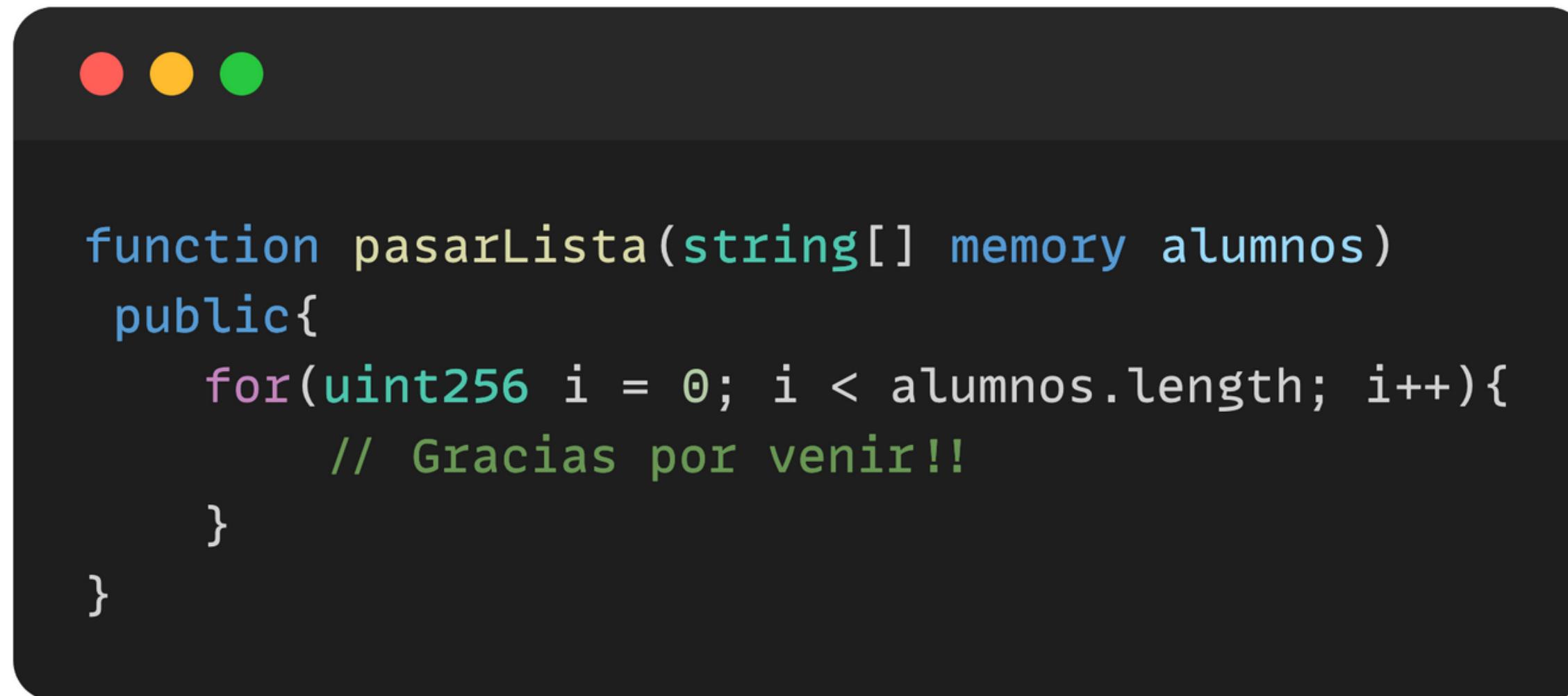


A screenshot of a Ethereum smart contract interface. The code is written in Solidity and contains an if-else conditional statement. The background is dark, and the code is displayed in a light-colored font. The interface includes standard window controls (red, yellow, green) at the top.

```
function sobornarAlPresentadorParaQueAcelere(bool aburrido)
public payable{
    if(aburrido){
        address direcctionPresentador = getDireccionPresentador();
        payable(direcctionPresentador).transfer(msg.value);
    }else{
        // Que bien que nos lo estamos pasando
    }
}
```

for: Para loops.

- **for:** Iteración por el indice indicado



A screenshot of a mobile application interface showing a snippet of Solidity code. The code defines a function named 'pasarLista' that takes an array of strings 'alumnos' as a parameter. The function contains a public modifier, a for loop that iterates from index 0 to the length of the 'alumnos' array, and a comment 'Gracias por venir!!'. The code is displayed on a dark background with three colored dots (red, yellow, green) at the top left.

```
function pasarLista(string[] memory alumnos)
public{
    for(uint256 i = 0; i < alumnos.length; i++){
        // Gracias por venir!!
    }
}
```

while: Para loops.

- **while:** Hasta que la condición deje de cumplirse



A Scratch script titled "recordarDonaciones ()". It starts with a variable "noHaDonado" set to true. A while loop runs as long as "noHaDonado" is true. Inside the loop, it checks if "msg.value > 0". If true, it sets "noHaDonado" to false. The loop then ends, and the script ends.

```
function recordarDonaciones ()
public{
    bool noHaDonado = true;
    while(noHaDonado){
        if(msg.value > 0){
            noHaDonado = false;
        }
    }
}
```

require: El control de seguridad

- Usamos **require** para asegurarnos que alguna de las variables cumple con las reglas de negocio que tenemos.



require: El control de seguridad



```
function noDejarEjecutarAPersonasQueNoHayanDonado ()  
public{  
    bool donacion = msgSenderHaDonado(msg.sender);  
    require(donacion, "ESTA PERSONA NO HA DONADO");  
}
```

Eventos: El diario del contrato

- Los **eventos** permiten registrar acciones en la cadena de bloques para luego ser recuperados en busca de información sobre la ejecución.



Eventos: El diario del contrato



```
event decisionTomada(bool decision, string agradecimiento);

function estaisSegurosQueNoQuereisDonar(bool decision) public{
    if(decision){
        emit decisionTomada(decision, "Te odio");
    }else{
        emit decisionTomada(decision, "Oleee");
    }
}
```

Herencia y polimorfismo: La familia en Solidity

- Herencia permite que un contrato herede propiedades y métodos de otro.



Herencia y polimorfismo: La familia en Solidity

```
contract ReceptorDeDonaciones{  
    constructor(){}
    function recibirDonaciones() public payable{
        //Esta funcion necesita logica compleja
    }
}
```

```
contract ProcesadorDePagos{
    address receptor;
    function procesarPago(address pagador) public payable{
        payable(receptor).transfer(msg.value);
        emit agradecerPago(pagador, "Gracias");
    }
}
```

```
import 'procesadorDePagos.sol';

contract ReceptorDeDonaciones is ProcesadorDePagos{
    constructor(){}
    function recibirDonaciones() public payable{
        procesarPago{value: msg.value}(msg.sender);
    }
}
```

Herencia y polimorfismo: La familia en Solidity

- **Polimorfismo:** Varios contratos pueden implementar la misma función de manera diferente.



Herencia y polimorfismo: La familia en Solidity

```
import 'procesadorDePagos.sol';

contract ReceptorDeDonaciones is ProcesadorDePagos{
    constructor(){}
    mapping(address => uint256) donacionPorUsuario;

    function recibirDonaciones() public payable{
        procesarPago{value: msg.value}(msg.sender);
    }

    function recibirDonaciones(address pagador) public{
        // Aqui no vamos a ejecutar nada por que no quiero yo
    }

    function recibirDonaciones(address pagador) public override{
        donacionPorUsuario(pagador) += msg.value;
    }
}
```

ABI: la interfaz de nuestro contrato

La ABI de Solidity es como un intérprete entre un contrato inteligente en la red Ethereum y el mundo exterior.

Es el "menú" que te dice qué funciones puedes llamar en un contrato y cómo hacerlo.

Piensa en ello como el manual de instrucciones para interactuar con un contrato inteligente. Esencialmente, es el lenguaje común que permite que tu aplicación hable con la blockchain.



ABI: la interfaz de nuestro contrato



```
[{"anonymous":false,"inputs":  
 [{"indexed":true,"internalType":"address","name":"donante","type":"address"},  
 {"indexed":true,"internalType":"address","name":"receptor","type":"address"}]
```

Gracias por la atención



Y buena suerte

Cairo

1. Cairo como Lenguaje Verificable

1.1 Cairo Optimiza Ethereum

1.1 Crecimiento de Desarrolladores

1.2 Diferencias con Solidity

1.3 Diferencias entre EVM y CVM

2. Sierra y CASM

3. Tipos de Funciones en Cairo en contratos de Starknet

4. Sintaxis básicas

4.1 Cairo Book

5. Casos de usos de Cairo

6. Despliegue en Remix Solidity - Cairo

7. Recursos de aprendizaje en Starknet-Cairo

7.1 Starkli y Scarb para [Workshop 4](#)

8. Ejercicios de aprendizaje en Starknet-Cairo

11. Primeros pasos para desarrollar con Cairo en Starknet, recursos y ejercicios disponibles.



Cairo como Lenguaje Verificable



STARKNET

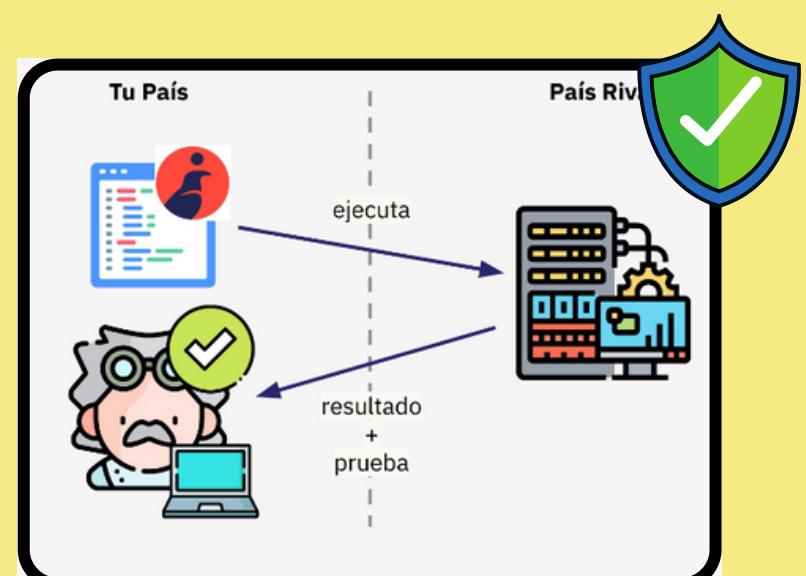
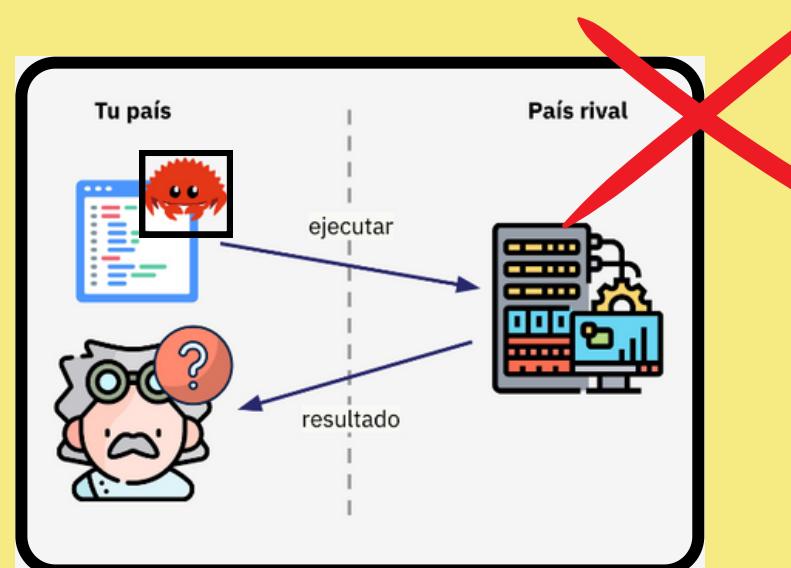
Cairo como Lenguaje Verificable

Cairo (**CPU Algebraic Intermediate Representation**) es un lenguaje **Turing-complete** con una propiedad impresionante: su capacidad para crear **Programas Verificables** mediante **STARKs**, destinados a la computación general.

Además, Cairo puede ser empleado **fuerza del ámbito de las blockchains** siempre que se necesite una prueba de cálculo, lo que amplía significativamente su versatilidad y utilidad.

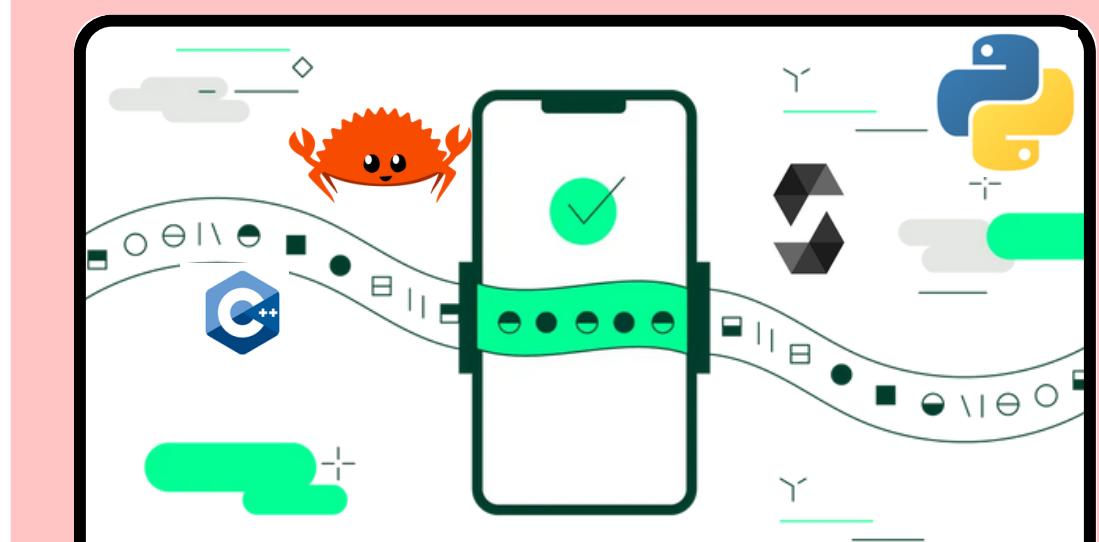
Programas verificables - STARK-provable

Los programas escritos en Cairo pueden ejecutarse de manera que se pueda crear eficientemente una prueba STARK de la ejecución, que permite a cualquier verificador independiente verificar de manera sencilla la ejecución honesta del programa, sin tener que confiar en la máquina en la que se realizaron los cálculos del programa.



Turing-complete

Computadora que puede realizar cualquier cálculo, ejecutar cualquier algoritmo o programa si se le proporciona suficiente tiempo y recursos.



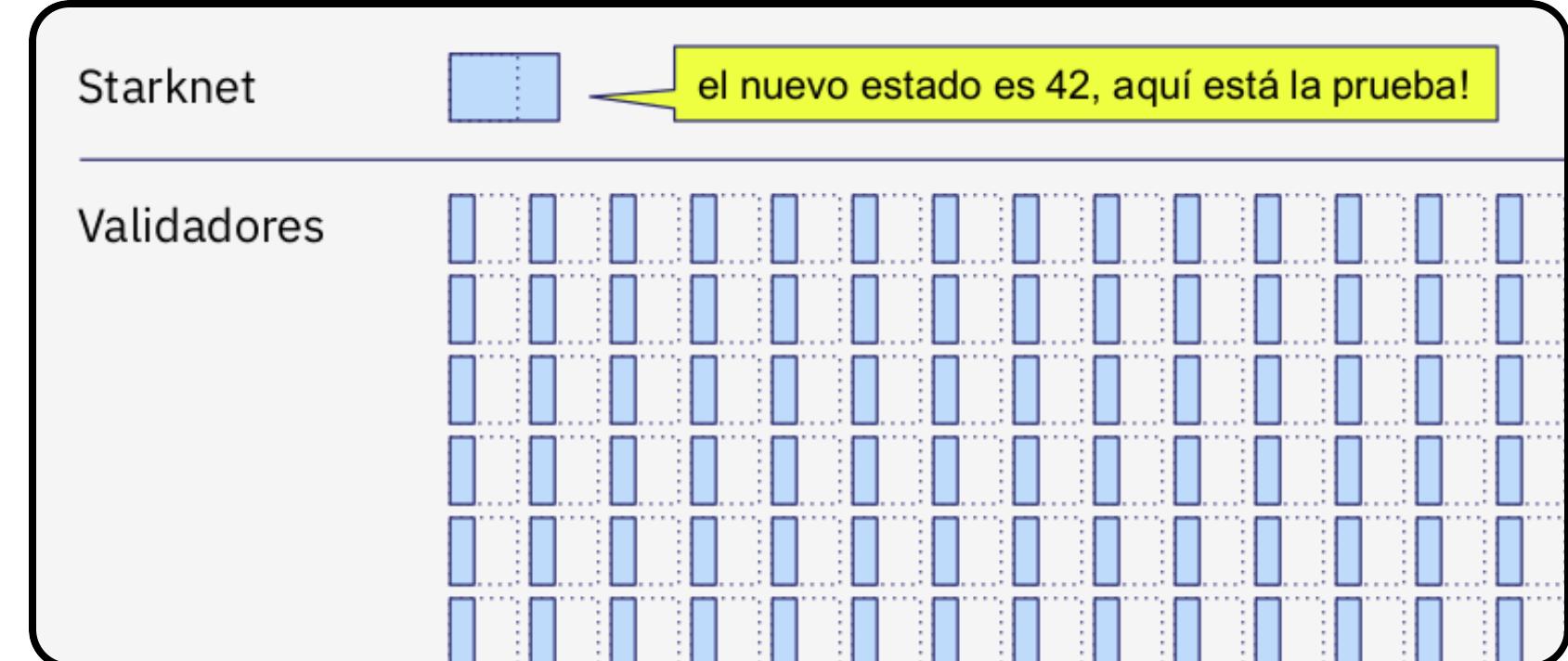
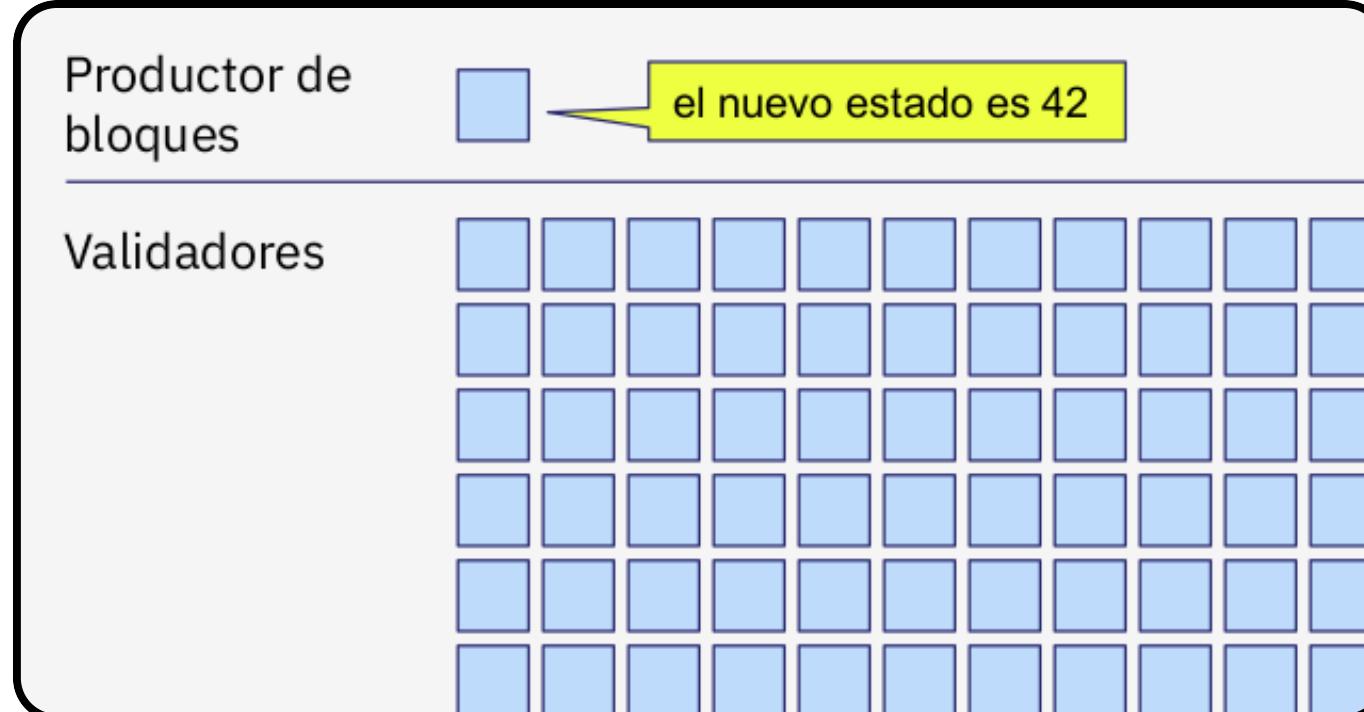
Cairo optimiza Ethereum

En este enfoque, el "**Prover**", una única máquina (no cientos de miles como todos los validadores de Ethereum), crea una prueba precisa de la ejecución de una transacción, permitiendo a otros participantes de la red verificarla.

Esta propiedad mejorará la eficiencia y confiabilidad haciendo de Cairo un sistema verificable mediante STARKs.

Ejecutar 1 Vez, Verificar en todas partes.

Los validadores de la red Ethereum repiten cálculos exhaustivamente, ahora solo necesitarán verificar las Proof.



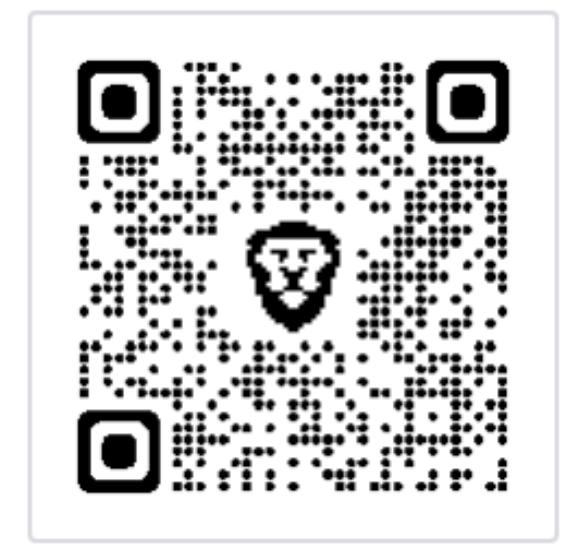
Crecimiento de Desarrolladores

La actividad de nuevos desarrolladores en el ecosistema blockchain sigue en aumento. CAIRO se emplea para escribir programas y contratos inteligentes en Starknet y StarkEx (dYdX, Rhino, ApexPro, ImmutableX, Sorare, entre otros), con un AUMENTO IMPRESIONANTE del 1550% en los últimos 2 años en desarrolladores a tiempo completo.

New Developer Activity

Top Ecosystems Monthly Active Developers ⓘ

Ecosystem	JUN-01 2023	Full-Time Developers ⓘ		Total Developers ⓘ	
		1y %	2y %	JUN-01 2023	1y %
Ethereum	1901	-11%	+35%	5946	-15%
Polkadot	645	-18%	+26%	1923	-22%
Cosmos	524	0%	+81%	1685	0%
Solana	363	-32%	+167%	1475	-45%
Bitcoin	322	-7%	+10%	963	-11%
Polygon	201	-35%	+105%	837	-40%
Kusama	186	-19%	+48%	535	-21%
Cardano	133	-18%	+21%	423	-17%
BNB	133	-39%	+13%	548	-40%
NEAR	132	-51%	+19%	531	-58%
Starknet	132	+38%	+1550%	407	+4%



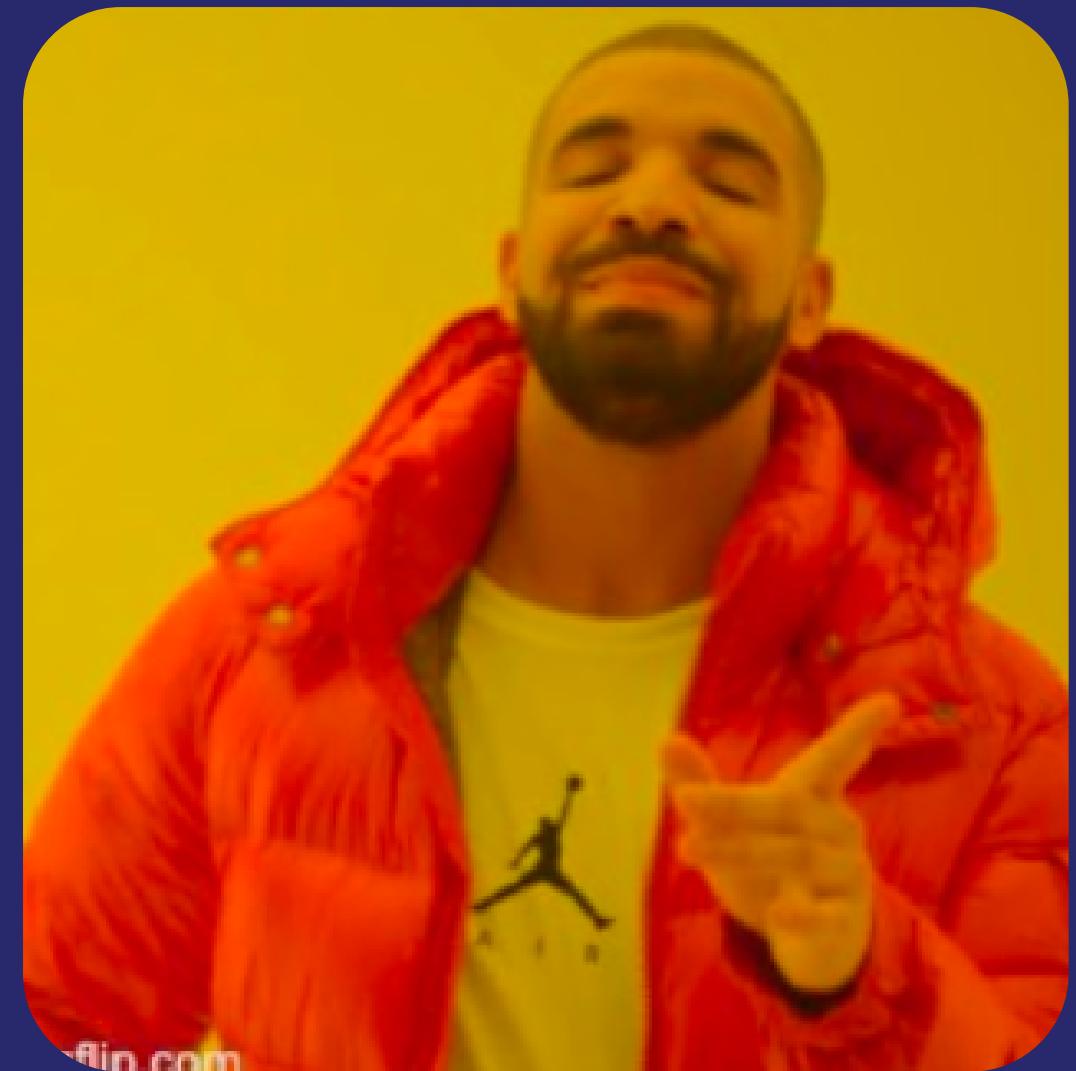
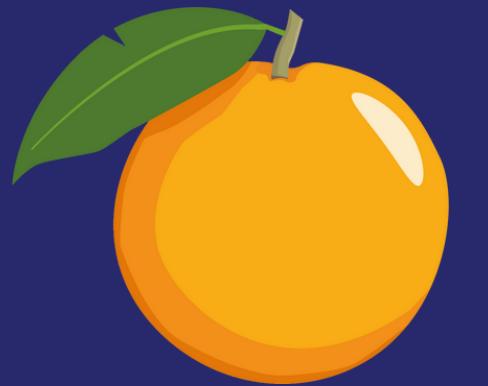
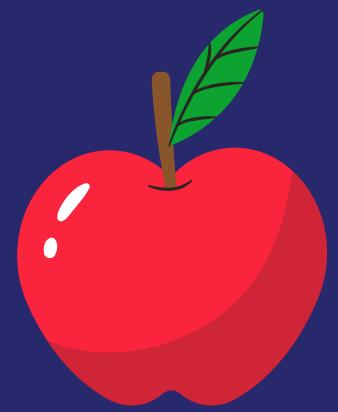
[Developer Report](#)

La Gran Pregunta

¿Solidity o Cairo?



Estás comparando



Diferencias con Solidity

Solidity es el idioma utilizado por primera vez para adoptar ampliamente cálculo composable, permitiendo a los Smart Contract grandes estructuras enlazadas como Legos. Cairo incluso puede vivir fuera de Blockchain, con la capacidad para crear Programas Verificables con STARKs.

Solidity nos enseño:

- Código de bytes que se hizo público y transparente para el mundo
- Las estructuras estándar de ABI, permitieron a otros interactuar con SC externos
- Uso de la red Ethereum para tener una vida útil o tiempo de actividad casi del 100%
- Uso de la red Ethereum como medio para ejecutar transacciones financieras
- Interoperabilidad global de SC debido a un único EVM común, con copia en manos de todos los validadores de la red

Cairo nos dará:

- Cairo ofrece funcionalidades similares a Solidity
- Permite la creación de Smart Contract y Programas Verificables con STARK
- Diferencias clave entre CVM y EVM para conseguir que la ejecución de los programas en Cairo se conviertan y sean posible verificar por un STARK
- Introduce Sierra como una capa de representación intermedia que agrega medidas de seguridad, como la prevención de ataques (DoS)
- Mejoras en un lenguaje más moderno con sintaxis basadas en Rust

Cairo Virtual Machine y diferencias con EVM



STARKNET

Fields o Campos

Cairo necesita una Virtual Machine preparada para admitir pruebas criptográficas con operaciones de Fields.

- Cairo utiliza campos finitos para pruebas criptográficas.
- Estas pruebas involucran operaciones matemáticas en elementos de campo específicos.
- Los elementos de campo usan aritmética modular para envolverse intencionalmente cuando su valor excede el rango especificado.
- Los elementos de campo son números enteros en Cairo.
- Operan dentro del rango $0 \leq x < P$, donde P es un número primo grande.
- El valor de P es $P = 2^{251} + 17 * 2^{192} + 1$.
- En Cairo, el tipo de dato básico es "**felt252**," que representa un elemento de campo.
- Esto permite a Cairo realizar operaciones en un rango de valores más amplio que las convenciones EVM.
- Cairo puede realizar pruebas en órdenes de magnitud diferentes durante la ejecución.

Diferencias entre EVM y CVM

Su arquitectura eficiente y sofisticada permite una gestión avanzada de la memoria, validación de instrucciones y deducción de valores.

Modelo de Memoria

La VM de Cairo utiliza un modelo de memoria de escritura única. Una ubicación de memoria no puede ser sobrescrita (a diferencia de la EVM).

- El Compilador resuelve la sobrecarga
- La reescritura en una forma abstracta se puede cubrir con variables mutables
- El modelo de escritura única hace Cairo más predecible y verificable, dejando un registro inmutable y claro en computación
- Facilita el trabajo de verificación de cálculos de la prueba STARK

Herencia y Polimorfismo

Cairo no tiene los conceptos de herencia y polimorfismo. La herencia es un concepto fundamental en la POO que permite a una clase (o tipo) heredar propiedades y comportamientos de otra clase.

- Los contratos Cairo pueden extenderse importando funciones y variables de almacenamiento específicas
- Cairo utiliza rasgos (traits), para lograr efectos similares a la herencia de la POO. Los Traits permiten definir conjuntos de métodos que pueden ser implementados por múltiples tipos, lo que facilita la composición de comportamientos compartidos de manera flexible y sin la necesidad de una jerarquía de clases.



CAIRO

Cairo Virtual Machine - Segmentos y Registros

La Máquina Virtual Cairo es un entorno de ejecución con memoria de solo lectura, direcciones continuas y segmentos de tamaño variable. Emplea registros para eficiencia y garantiza la integridad mediante deducción de valores y validación de instrucciones. Optimiza y paralleliza con pruebas recursivas, mejorando eficiencia y escalabilidad.

Segmentos

Los datos se almacenan en bloques o segmentos de memoria que son contiguos:

- **Segmento del Programa:** Contiene el bytecode de Cairo, con el Program Counter (pc) iniciando al principio de este segmento.
- **Segmento de Ejecución:** Aquí se generan datos durante la ejecución del programa Cairo y su longitud es variable según la entrada del programa. Los Allocation Pointer (ap) y Frame Pointer (fp) comienzan aquí.
- **Segmento Incorporado:** Cada función incorporada tiene su propio espacio continuo en memoria. La longitud es variable.

Registros

Los registros son ubicaciones de memoria especiales y de alto rendimiento que se utilizan para almacenar datos temporales y realizar operaciones rápidas.

- **Allocation Pointer (ap):** Apunta a una celda de memoria sin usar.
- **Frame Pointer (fp):** Apunta al marco de la función actual. Las direcciones de todas las variables locales y argumentos son relativas al valor de este registro. Al inicio de una function1, **fp** es igual a **ap** y permanece constante durante su ejecución. Cuando una function2 se llama dentro de una function1, el valor de **fp** cambia para function2 pero se restaura a su valor original al finalizar function2 (esto permite rastrear el valor original de function1).
- **Program Counter (pc):** Apunta a la instrucción actual. Cada instrucción toma 1 o 2 felts, y **pc** avanza 1 o 2 después de cada instrucción. Los saltos pueden cambiar el valor del **pc** a un valor absoluto o relativo, según las operaciones de salto.

Sierra y CASM



STARKNET

Sierra y CASM

SIERRA - Safe IntERmediate RepresentAtion, sirve como un intermediario crucial entre los lenguajes de programación de alto nivel de Cairo y los objetivos de compilación más primitivos, como CASM - Cairo ASseMBly, al garantizar que el CASM generado sea seguro de ejecutar en Starknet (un programa de Cairo y su entrada).

Cairo se convierte primero en código Sierra, que debe publicarse en la red.

Sierra están compuestos de declaraciones que invocan libfuncs. (conjunto de funciones de biblioteca integradas) las cuales garantiza que el código CASM generado sea seguro.

Sierra asegura que incluso las transacciones revertidas puedan incluirse en los bloques Starknet desde la versión 0.12.1

El código Sierra es una abstracción sobre el CASM interpretado por la CVM.

Incorpora un compilador robusto combinado con un sistema de tipo lineal para evitar errores de tiempo de ejecución.

Un sistema de gas incorporado para evitar bucles infinitos.



Cairo (Ergonómico)

Compile

Sierra (Seguro)

Compile

CASM (Ensamblaje)



STARKNET

Sierra simplifica la necesidad de agregar mecanismos criptoeconómicos complejos:

1. Los secuenciadores pueden cobrar tarifas en transacciones revertidas, lo que permite a Starknet evitar ataques DoS o pagar por ello.
2. La implementación de transacciones forzadas de L1 será posible, permitiendo a Starknet heredar la resistencia de censura completa de Ethereum.

Tipos de Funciones



STARKNET

Tipos de Funciones

Public Functions

- Accesibles desde fuera del contrato.
- Deben ser definidas dentro de un bloque de implementación anotado con el atributo `#[external(v0)]`
- Este atributo solo afecta a la visibilidad (pública vs. privada/interna).

Contratos en Starknet

```
#[external(v0)]
impl NameRegistry of super::INameRegistry<ContractState> {
    fn store_name(ref self: ContractState, name: felt252) {
        let caller = get_caller_address();
        self._store_name(caller, name);
    }

    fn get_name(self: @ContractState, address: ContractAddress) -> felt252 {
        let name = self.names.read(address);
        name
    }
}
```

View (Public Functions)

- Sólo de lectura que te permiten acceder a los datos del contrato
- El estado del contrato no se modifica.
- Pueden ser llamadas por otros contratos o de forma externa.
- El `self: @ContractState` se pasa como una instantánea (snapshot), lo que evita que puedas modificar el estado del contrato.

```
fn get_name(self: @ContractState, address: ContractAddress) -> felt252 {
    let name = self.names.read(address);
    name
}
```

External (Public Functions)

- Pueden modificar el estado de un contrato, incluidas los Contratos de Cuentas (AA).
- Pueden ser llamadas por otros contratos o de forma externa.
- El `self: ContractState` se pasa como referencia con la palabra clave `ref`, lo que permite modificar el estado del contrato.

```
fn store_name(ref self: ContractState, name: felt252) {
    let caller = get_caller_address();
    self._store_name(caller, name);
}
```

Tipos de Funciones

Constructor

- Es un atributo de las funciones en Cairo que se utiliza para inicializar y desplegar un contrato inteligente.
- En el constructor se pasan los argumentos necesarios para configurar el estado inicial del contrato al momento de su creación en la cadena de bloques.
- No puede tener más de un constructor.
- Debe estar definida con el atributo **#[constructor]**
- La función constructora debe tener el nombre **fn constructor**

Contratos en Starknet



```
#[constructor]
fn constructor(ref self: ContractState, owner: Person) {
    self.names.write(owner.address, owner.name);
    self.total_names.write(1);
    self.owner.write(owner);
}
```

Internal/Private

- Por defecto, todas las funciones de los contratos en Cairo se consideran internas.
- Estas funciones solo pueden ser llamadas desde dentro del contrato, algo similar a las funciones privadas en otros lenguajes.
- Deben ser definidas “fuera” de los bloques de implementación anotados con **#[external(v0)]**



```
#[generate_trait]
impl InternalFunctions of InternalFunctionsTrait {
    fn _store_name(ref self: ContractState, user: ContractAddress, name: felt252) {
        let mut total_names = self.total_names.read();
        self.names.write(user, name);
        self.total_names.write(total_names + 1);
        self.emit(Event::StoredName(StoredName { user: user, name: name }));
    }
}
```

Sintaxis básica de Cairo



STARKNET

Sintaxis básica de Cairo

Variables

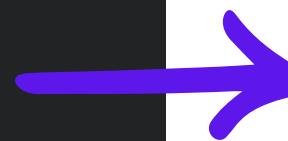
- Piense en una Variable como un contenedor para almacenar y gestionar datos en un programa.
- En Cairo son inmutables por defecto
- Si una variable es immutable, una vez que un valor está vinculado a un nombre, no puedes cambiar ese valor, ejemplo: `'let x = 5;'`
- También está la opción de hacer que tus variables sean mutables



Ejemplos - Variables

```
use debug::PrintTrait;
fn main() {
    let x = 5;
    x.print();
    x = 6;
    x.print();
}
```

```
error: Cannot assign to an immutable variable.
--> lib.cairo:5:5
    x = 6;
    ^***^
Error: failed to compile: src/lib.cairo
```



Variables Mutables

- En Cairo, `'let mut'` declara variables como mutables, ejemplo:
`'let mut x = 5;'`
- Evita errores al asegurar que valores inmutables no cambien.
- Con `let mut`, claramente indica cuándo se espera que una variable cambie.
- Ofrece flexibilidad controlada para modificar datos en el código.

```
use debug::PrintTrait;
fn main() {
    let mut x = 5;
    x.print();
    x = 6;
    x.print();
}
```

```
$ scarb cairo-run
[DEBUG]                                     (raw: 5)
[DEBUG]                                     (raw: 6)
Run completed successfully, returning []
```



STARKNET



Sintaxis básica de Cairo

Constantes

- Se declaran con la palabra clave **'const'**, al igual que las variables inmutables, no pueden cambiar su valor.
- Las constantes son útiles para valores que muchas partes del código necesitan conocer.
- Siguen una convención de nombres en mayúsculas con guiones bajos **'ONE_HOUR_IN_SECONDS'**
- Siempre requieren una anotación de tipo, ejemplo: **'u32 = 3600'**
- Las constantes no pueden usarse con **'mut'**.
- Permanecen válidas durante toda la ejecución del programa, un ejemplo:
'const ONE_HOUR_IN_SECONDS: u32 = 3600;'



STARKNET

Ejemplos - Constantes



```
const ONE_HOUR_IN_SECONDS: u32 = 3600;
```



```
const ONE_HOUR_IN_SECONDS: u32 = 3600;
```



```
const ONE_HOUR_IN_SECONDS: u32 = 3600;
```



Sintaxis básica de Cairo

Arrays

- Puedes utilizar métodos de arrays importando el trait 'array::ArrayTrait'.
- Un array es una colección de elementos del mismo tipo.
- Los arrays tienen opciones de modificación limitadas.
- Son colas cuyos valores no pueden modificarse.
- Para crear un array, se utiliza la función 'ArrayTrait::new()'.
- Para añadir elementos al final de un array, se utiliza el método 'append()', ejemplo: 'a.append(10);' ***
- Los elementos solo se pueden quitar del principio usando 'pop_front()'.
Esto devuelve un 'Option' con el elemento eliminado o 'Option::None' si el array está vacío, ejemplo: 'a.pop_front()' devolverá el primer valor 10 ***
- No se pueden modificar los elementos de un array una vez que se han añadido.
- Puedes especificar el tipo de elementos al instanciar el array o definir explícitamente el tipo de la variable.
- Estas operaciones actualizan los punteros en lugar de modificar directamente las celdas de memoria.



STARKNET

Ejemplos - Arrays

```
use array::ArrayTrait;
```

```
use debug::PrintTrait;
```

```
fn main() {
```

```
    let mut a = ArrayTrait::new();  
    a.append(10);  
    a.append(1);  
    a.append(2);
```

```
    let first_value = a.pop_front().unwrap();  
    first_value.print(); // print '10'  
}
```

```
let mut arr = ArrayTrait::<u128>::new();
```

```
let mut arr: Array<u128> = ArrayTrait::new();
```



Sintaxis Básica

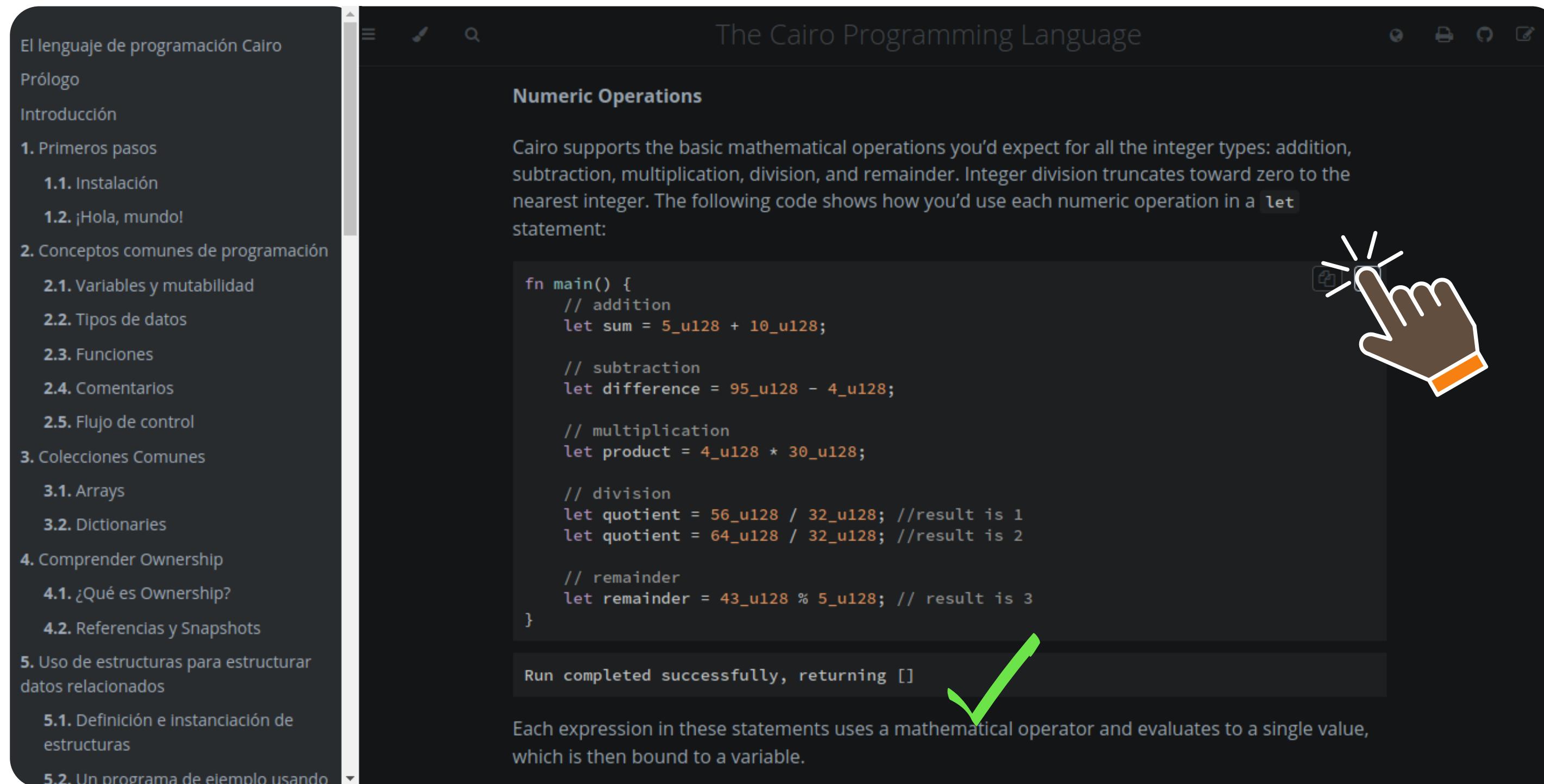
En Cairo se han agregado enteros sin signo, similares a los de Solidity, lo que promete simplificar la vida de los desarrolladores a pesar de posiblemente tener un costo mayor en términos de eficiencia para el secuenciador. El uso de Arrays en Cairo es similar en sintaxis a Rust y comparte lógica con Solidity, lo que facilita la programación y la comprensión de los desarrolladores.

Cairo	Solidity
<pre>#[view] fn is_cairo_fun() -> (u8, u8) { let yes = yes_var::read(); let no = no_var::read(); return (yes, no); }</pre>	<pre>function is_cairo_fun() public view returns (uint8, uint8) { uint8 yes = yes; uint8 no = no; return (yes, no); }</pre>
<ul style="list-style-type: none">• Declarar una función es 'fn'.• Los tipos de funciones se declaran antes que 'fn' (<code>#[view]</code>).• Declarar valores de retorno implica el uso del símbolo '<code>→</code>'.	<ul style="list-style-type: none">• Declarar una función es 'function'• El formato del tipo de función de 'view' es diferente.• La palabra clave "returns" se utiliza para indicar valores de retorno.

u8	unsigned 8-bit integer	uint8
u16, u32...u256	unsigned 16-bit/32-bit.../256-bit integer	uint16, uint32,... uint256
bool	boolean	bool
felt252	field element (the most basic type that represents a "number")	N/A
Option <T>	value that may or may not be present (hence "optional" type)	N/A
Result <T,E>	outcome of a computation that may result in an error	N/A
Array <T>	dynamic array structure	T[]

Cairo Book

Un libro creado por la Comunidad de Cairo y colaboradores sobre Cairo que explica su sintaxis y funcionamiento, con ejemplos que puedes compilar y ejecutar desde WASM para facilitar el aprendizaje sin necesidad de instalar Cairo en tu PC mientras estudias



The Cairo Programming Language

Numeric Operations

Cairo supports the basic mathematical operations you'd expect for all the integer types: addition, subtraction, multiplication, division, and remainder. Integer division truncates toward zero to the nearest integer. The following code shows how you'd use each numeric operation in a `let` statement:

```
fn main() {
    // addition
    let sum = 5_u128 + 10_u128;

    // subtraction
    let difference = 95_u128 - 4_u128;

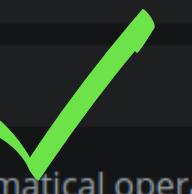
    // multiplication
    let product = 4_u128 * 30_u128;

    // division
    let quotient = 56_u128 / 32_u128; // result is 1
    let quotient = 64_u128 / 32_u128; // result is 2

    // remainder
    let remainder = 43_u128 % 5_u128; // result is 3
}
```

Run completed successfully, returning []

Each expression in these statements uses a mathematical operator and evaluates to a single value, which is then bound to a variable.



Casos de usos de Cairo



STARKNET

Workshops - Resumen General

- Poder de AA con la validación y ejecución separada.
- Cairo como Lenguaje de Programación Verificable.
- Sierra como Representación Intermedia Segura.
- STARKs como pruebas criptográficas para integridad de cálculos.
- Optimización con Cairo en interacciones con las dApps.
- Madara como Secuenciador modular basado en Substrate, base de dApp Chain.
- Kakarot como zkEVM que interpreta Solidity y ejecuta en Cairo.
- GIZA para zkML con Cairo.
- Mejor eficiencia desde Starknet 0.12 con implementaciones en RUST y mejoras ofrecidas por Lamda Class y Core.



STARKNET

Casos de usos de Cairo



Cairo como Lenguaje Verificable

Cairo es un lenguaje **Turing-complete** con una propiedad impresionante: su capacidad para crear **Programas Verificables** mediante STARKs, destinados a la computación general.

Además, Cairo puede ser empleado **frente del ámbito de las blockchains** siempre que se necesite una prueba de cálculo, lo que amplía significativamente su versatilidad y utilidad.

Programas verificables - STARK-provable

Los programas escritos en Cairo pueden ejecutarse de manera que se pueda crear eficientemente una prueba STARK de la ejecución, que permite a cualquier verificador independiente verificar de manera sencilla la ejecución honesta del programa, sin tener que confiar en la máquina en la que se realizaron los cálculos del programa.

Turing-complete

Computadora que puede realizar cualquier cálculo, ejecutar cualquier algoritmo o programa si se le proporciona suficiente tiempo y recursos.

DEX Ekubo - Add Liquidity con Rango en Goerli

The screenshots show the 'Crear Posición' (Create Position) screen and a mobile phone displaying the 'Sign Transaction' (Sign Transaction) screen.

¿Cómo mejora AA en Starknet vs ETH?

Algunos beneficios extras son: Convertir su teléfono inteligente en una Hardware Wallet, Verificación de firma personalizada, Paquete de transacciones (multillamada), Pago de tarifas con tokens no nativos, Delegación de pago (paymaster)

AA EN ETH VS STARKNET

Integración

- Ethereum: Capa de aplicación
- Starknet: Capa de protocolo

Apoyo al ecosistema

- Ethereum: ?
- Starknet: Full

Firmas personalizadas

- Ethereum: Soportado
- Starknet: Soportado

Delegación de tarifas

- Ethereum: Contrato inteligente de Paymaster
- Starknet: Meta transacciones

Válido también para Laptops y PCs

Procesador compartido / Memoria compartida → Sistema operativo / Aplicaciones (Secure Enclave, Titan M, Knox, arm TrustZone)

Procesador dedicado / Memoria dedicada → Generador de números aleatorios / Gestión de claves criptográficas / Firmante de hardware (secp256r1) / Llavero, TouchID y FaceID

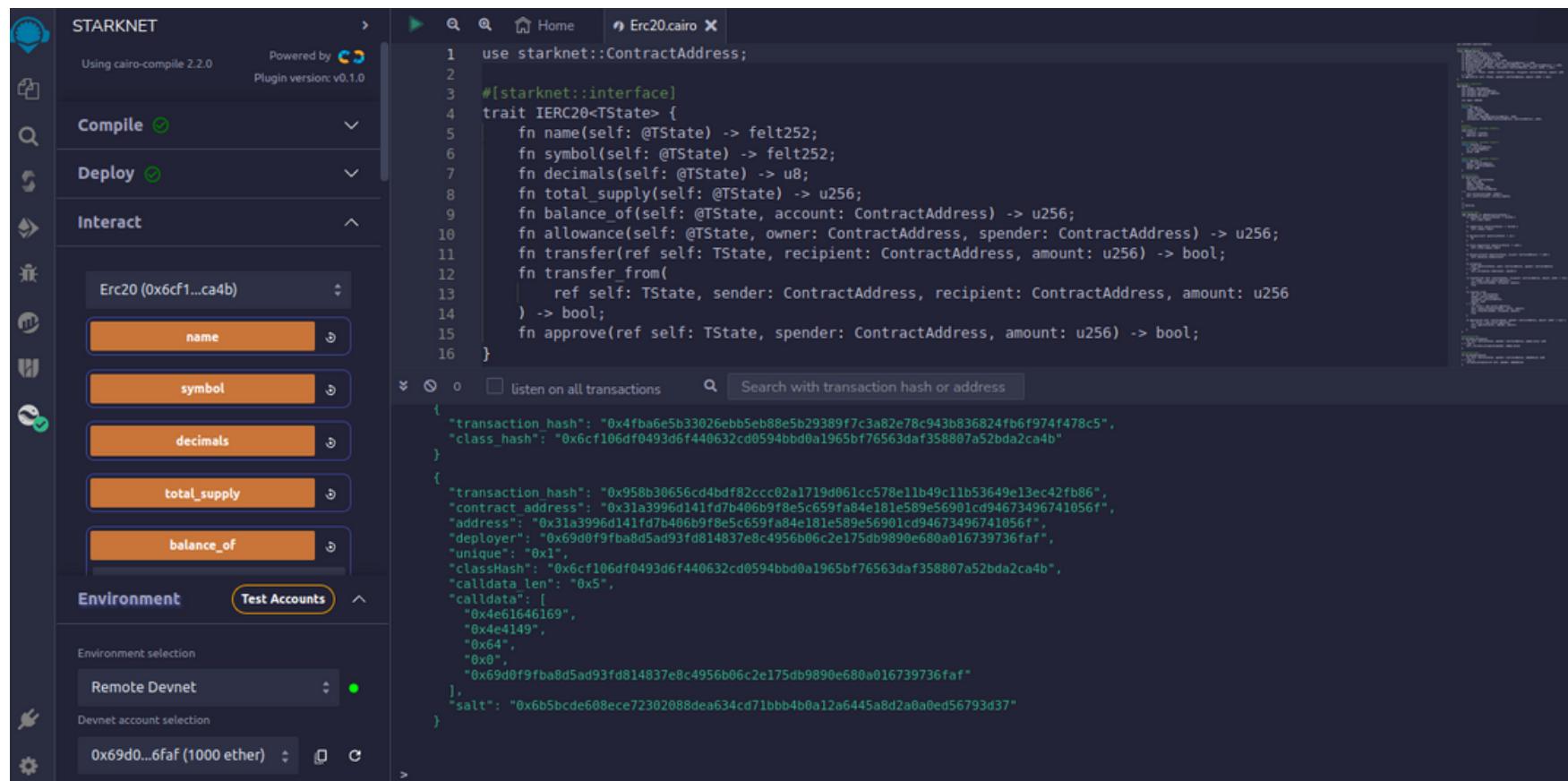
Despliegue en Remix Solidity - Cairo



STARKNET

Despliegue en Remix Solidity - Cairo

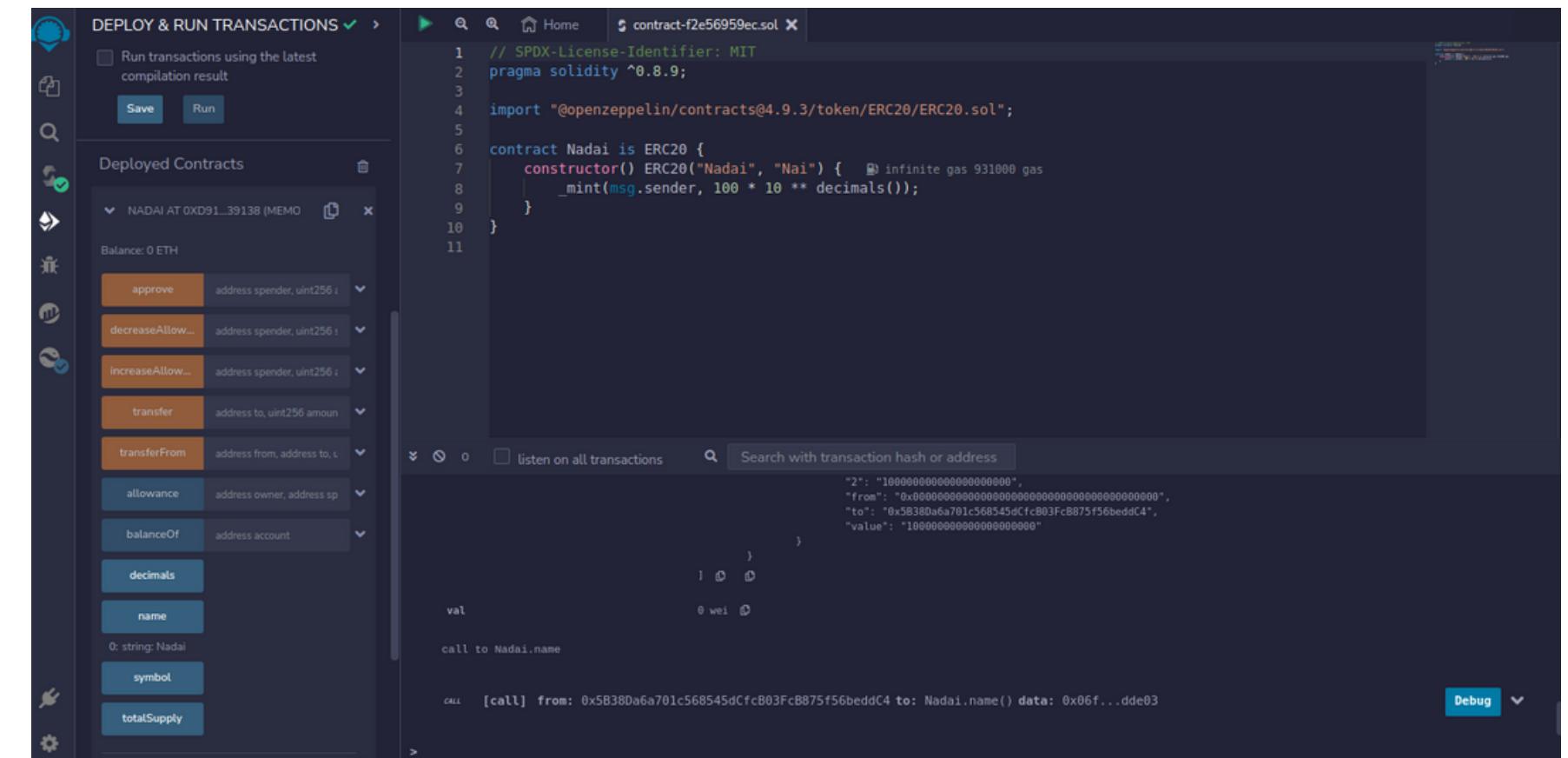
En esta práctica sencilla, exploraremos el proceso de compilación, declaración y despliegue rápido utilizando la plataforma de desarrollo de contratos inteligentes, Remix. Con esta experiencia, aprenderemos a copiar y pegar un contrato inteligente de un token ERC20 en Cairo y Solidity. Después de desplegarlo, interactuaremos con él para comprender su funcionamiento.



The screenshot shows the Starknet interface for developing Cairo contracts. On the left, there's a sidebar with options like 'STARKNET', 'Compile', 'Deploy', and 'Interact'. Under 'Interact', the 'Erc20 (0x6cf1...ca4b)' contract is selected. Below it, several buttons are shown for interacting with the contract: 'name', 'symbol', 'decimals', 'total_supply', and 'balance_of'. The main area displays the Cairo code for the Erc20 trait:

```
1 use starknet::ContractAddress;
2 
3 #[starknet::interface]
4 trait IERC20<@TState> {
5     fn name(self: @TState) -> felt252;
6     fn symbol(self: @TState) -> felt252;
7     fn decimals(self: @TState) -> u8;
8     fn total_supply(self: @TState) -> u256;
9     fn balance_of(self: @TState, account: ContractAddress) -> u256;
10    fn allowance(self: @TState, owner: ContractAddress, spender: ContractAddress) -> u256;
11    fn transfer(ref self: TState, recipient: ContractAddress, amount: u256) -> bool;
12    fn transfer_from(
13        ref self: TState, sender: ContractAddress, recipient: ContractAddress, amount: u256
14    ) -> bool;
15    fn approve(ref self: TState, spender: ContractAddress, amount: u256) -> bool;
16 }
```

At the bottom, there's an 'Environment' section with 'Test Accounts' and a 'Remote Devnet' selection. The 'Remote Devnet' dropdown shows '0x69d0..6faf (1000 ether)'.



The screenshot shows the Remix interface for deploying and running transactions. On the left, there's a sidebar with options like 'DEPLOY & RUN TRANSACTIONS', 'Deployed Contracts', and 'Interact'. Under 'Interact', the 'contract-f2e56959ec.sol' contract is selected. Below it, several buttons are shown for interacting with the contract: 'approve', 'decreaseAllow...', 'increaseAllow...', 'transfer', 'transferFrom', 'allowance', 'balanceOf', 'decimals', 'name', 'symbol', and 'totalSupply'. The main area displays the Solidity code for the Nadai contract:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.9;
3 
4 import "@openzeppelin/contracts@4.9.3/token/ERC20/ERC20.sol";
5 
6 contract Nadai is ERC20 {
7     constructor() ERC20("Nadai", "Nai") {
8         _mint(msg.sender, 100 * 10 ** decimals());
9     }
10}
```

On the right, there's a transaction history section with a table showing calls to the Nadai contract. One call is highlighted:

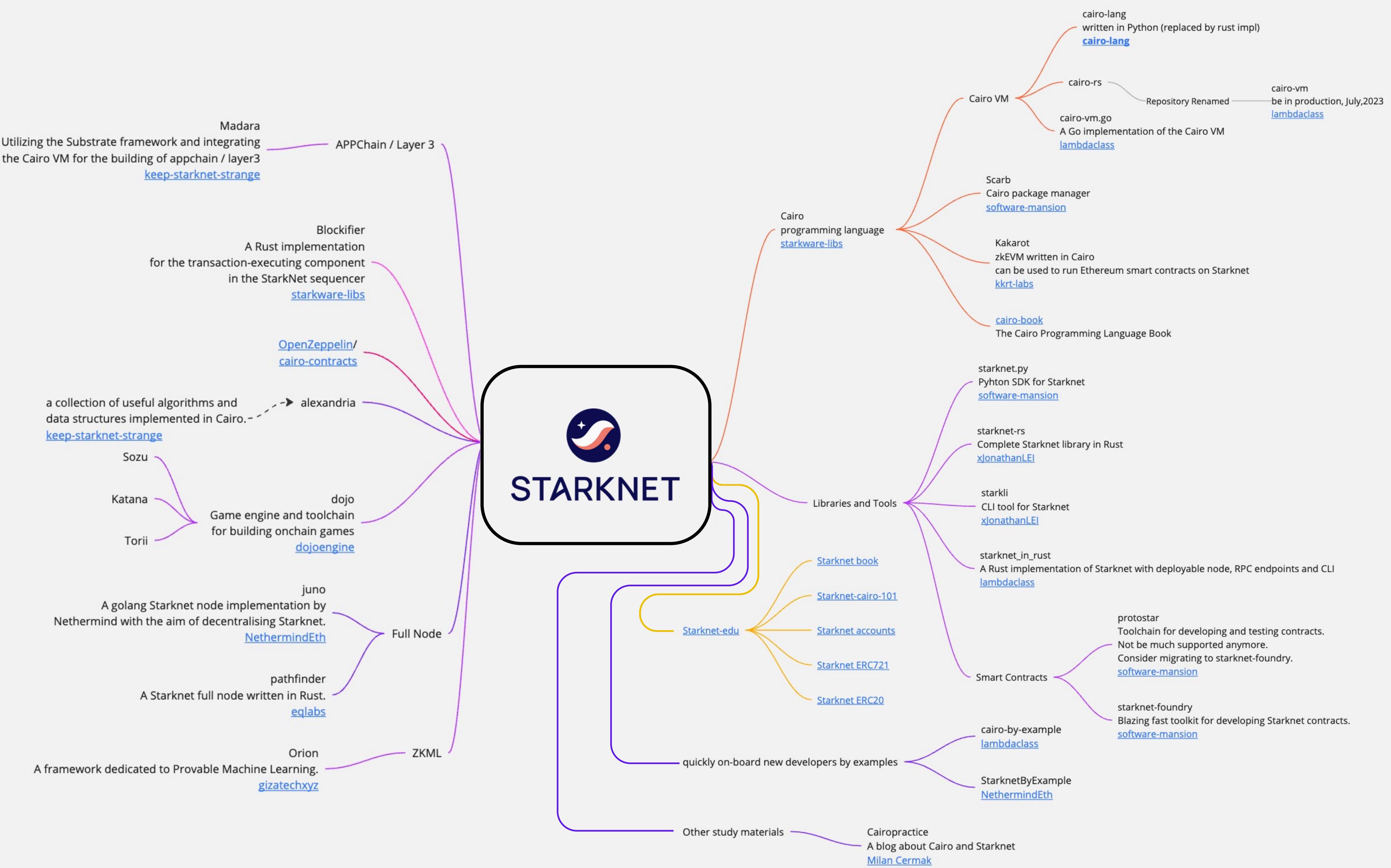
call	from	to	data	value
[call]	0x58380a6a701c568545dCfcB875f56beddC4	Nadai.name()	0x06f...dde03	1000000000000000000



Recursos y desarrollo en el ecosistema

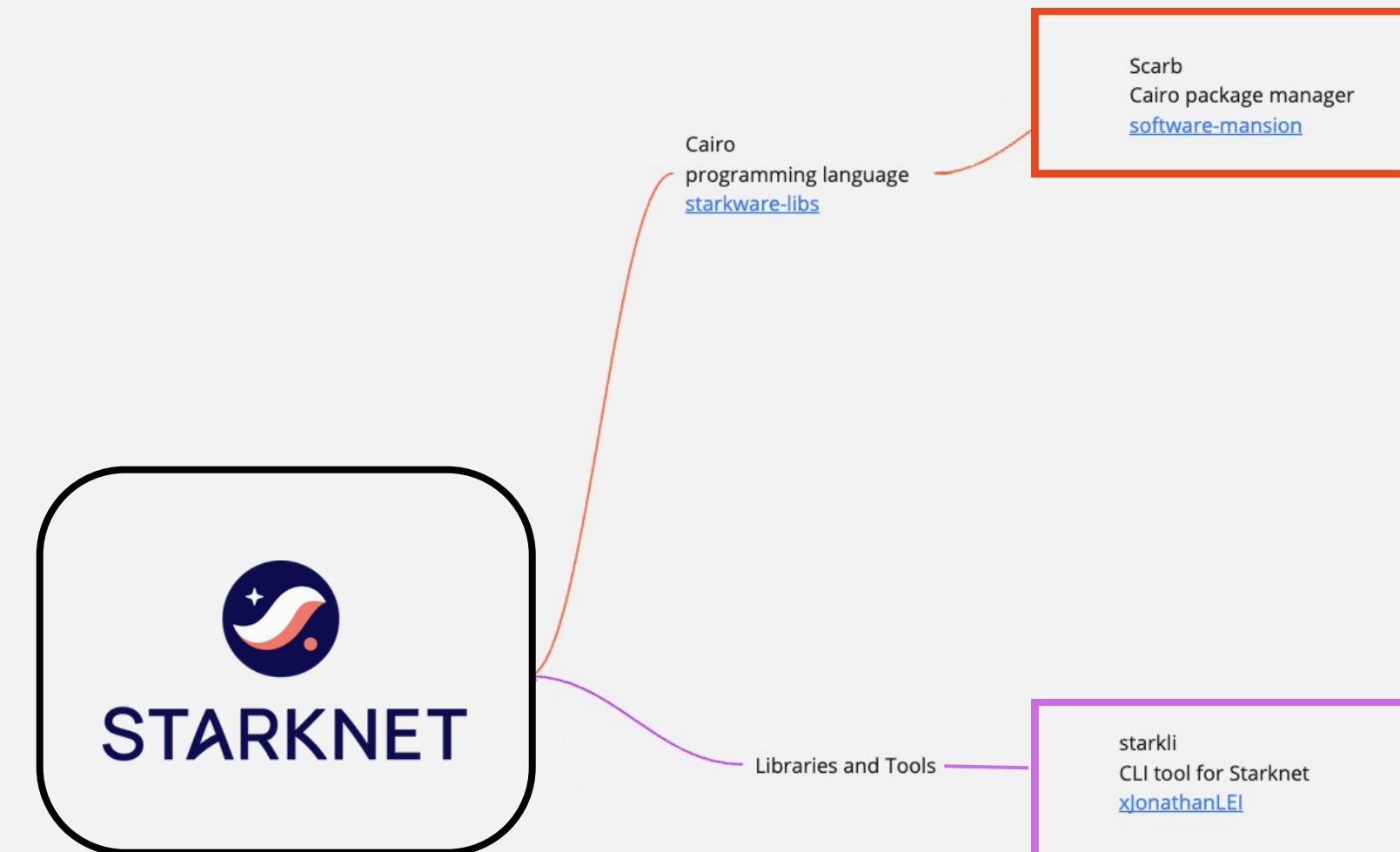


STARKNET





[Starknet-Tech-Stacks-Mindmap](#)



[Scarb](#)



xJonathanLEI/
starkli



[Starkli](#)

Workshop 4

Primeros pasos desarrollando con Cairo en Starknet

Workshop 4

En las prácticas de Starknet, aprenderemos a instalar Cairo de manera sencilla, crear una cuenta desde cero con Starkli y desarrollar nuestro primer contrato en Cairo utilizando Scarb para gestionar el proyecto.

Requisitos Previos

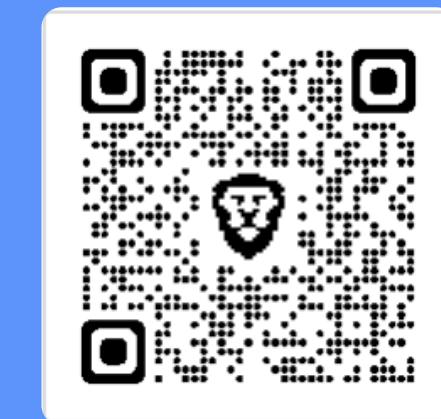
- Descargar Smart Wallet Braavos o Argent (Seguir guía Workshop 2)
- Conseguir Faucet (Seguir guía Workshop 2)
- Desplegar Contrato de Cuenta (Seguir guía Workshop 2)
- Instalar editor de código "VS Code"



STARKNET

PRESENCIAL

REGISTROS



Recursos de Aprendizaje en Starknet - Cairo



STARKNET

BOOK



[Starknet Book](#)

El libro del ecosistema de Starknet, una documentación exhaustiva de arquitectura y Starknet en general



CAIRO

BOOK



[Cairo Book](#)

El libro de aprendizaje sobre el Lenguaje de Programación de Cairo y su sintaxis



STARKNET

DOC



[Starknet Doc](#)

Documentos oficiales Generales de Starknet



CAIRO

DOC



[Cairo Doc](#)

Documentos oficiales Generales de Cairo

Ejemplos de Sintaxis en Starknet - Cairo

 **cairo** by example

assert

To make sure our tests work, we use assert.

```
fn main(x: felt252, y: felt252) {
    assert(x != y, 'error', x is equal to y);
}

#[test]
fn test_main() {
    main(1,2);
}
```

The first argument of assert is the condition we want to check, and the second is a message we will see on the console if the condition is false.

Run `cairo-test file_name`

Starknet by Example

Constructor

Constructors are a special type of function that runs only once when deploying a contract, and can be used to initialize the state of the contract. Your contract must not have more than one constructor, and that constructor function must be annotated with the `#[constructor]` attribute. Also, a good practice consists in naming that function `constructor`.

Here's a simple example that demonstrates how to initialize the state of a contract on deployment by defining logic inside a constructor.

```
#[starknet::contract]
mod ExampleConstructor {
    use starknet::ContractAddress;

    #[storage]
    struct Storage {
        names: LegacyMap::<ContractAddress, felt252>,
    }

    // The constructor is decorated with a `#[constructor]` attribute.
    // It is not inside an `impl` block.
    #[constructor]
    fn constructor(ref self: ContractState, name: felt252, address: ContractAddress) {
        self.names.write(address, name);
    }
}
```

Visit contract on [Voyager](#) or play with it in [Remix](#).



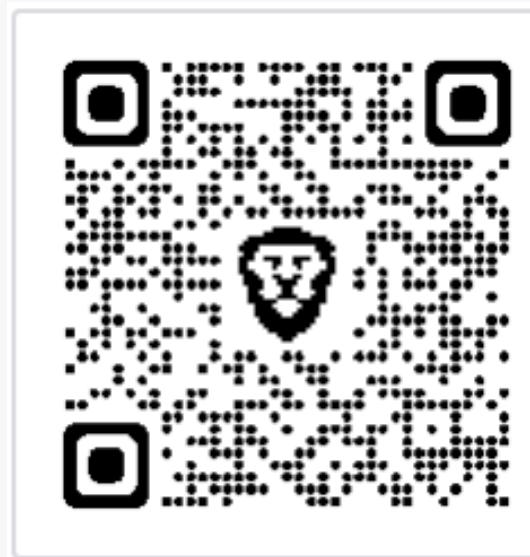
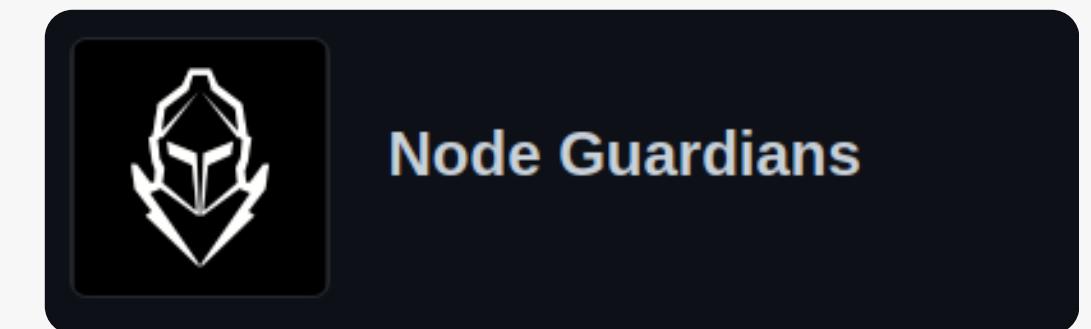
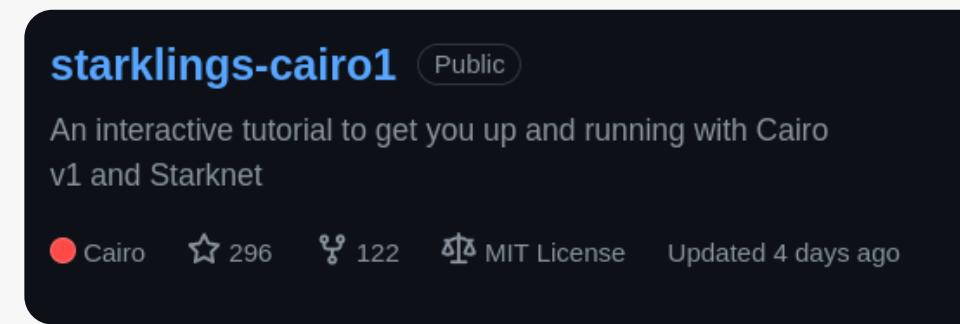
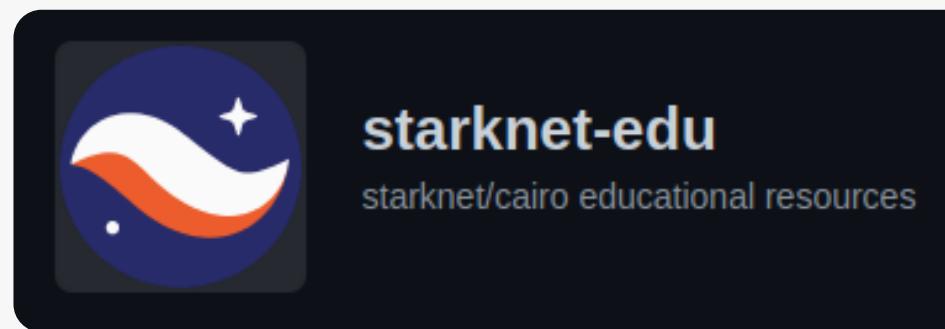
[Lambda Class - Cairo By Example](#)



[Nethermind - Starknet By Example](#)

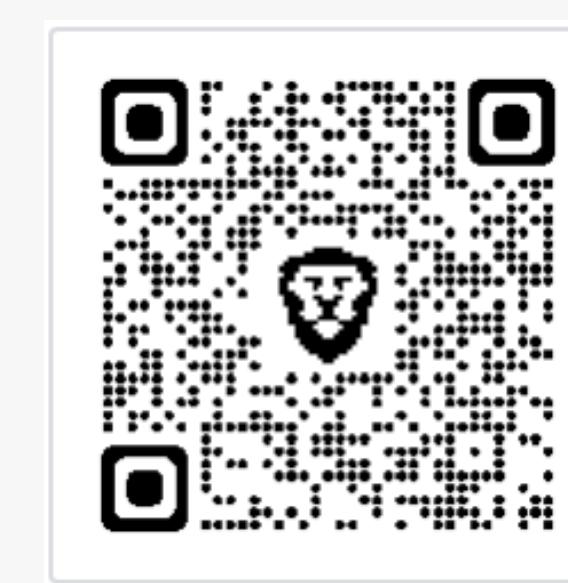


Ejercicios de Aprendizaje en Starknet - Cairo



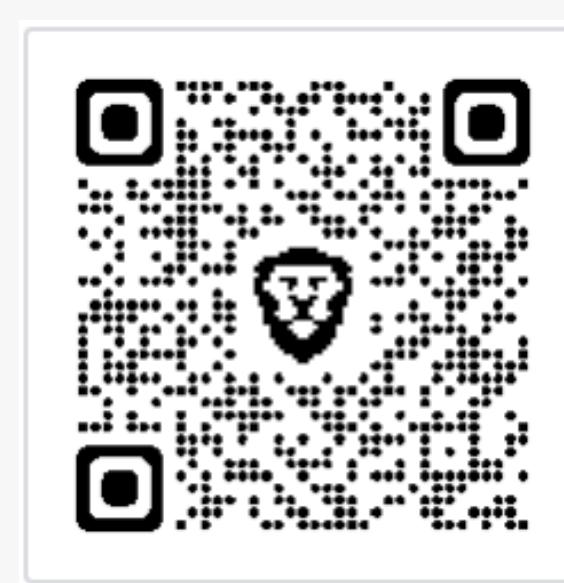
[Starknet Edu](#)

Ejercicios educacionales diseñado para desarrolladores interesados en aprender cómo leer código Cairo 1 y contratos inteligentes de Starknet, con un sistema de Evaluador y Puntos por ejercicios resueltos



[Starklings](#)

Un tutorial interactivo para que te familiarices con Cairo y Starknet. Deberás superar ejercicios de corrección de sintaxis para estar preparado para comenzar a ser un desarrollador de Cairo.



[Node Guardians](#)

Es un centro educativo para desarrolladores de Starknet que deseen aprender Cairo al superar las Quest que tienen preparadas, algunas de las cuales están relacionadas con Solidity.



Languages ▾

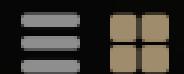
Skill Category ▾

Quest Type ▾

Campaign Status ▾

Difficulty ▾

No. Quests ▾



SETTING UP: CAIRO

Language Cairo

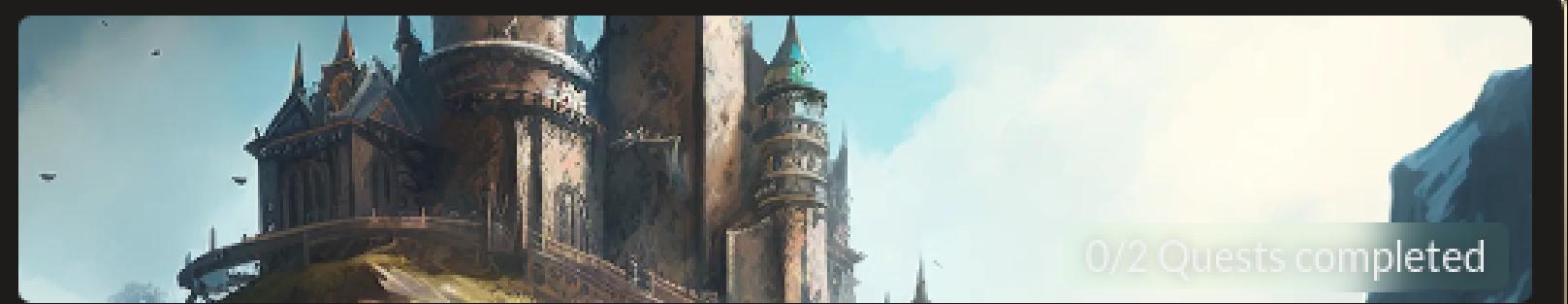
Difficulty

Skill Cat.

Total exp 1200

Types Build / CTF

No. Quests 2



SETTING UP: SOLIDITY

Language Solidity

Difficulty

Skill Cat.

Total exp 1200

Types Build / CTF

No. Quests 2



GET REKT

Language Solidity

Difficulty

Skill Cat.

Total exp 11000

Type CTF

No. Quests 3



BAD ACCOUNTS

Language Cairo

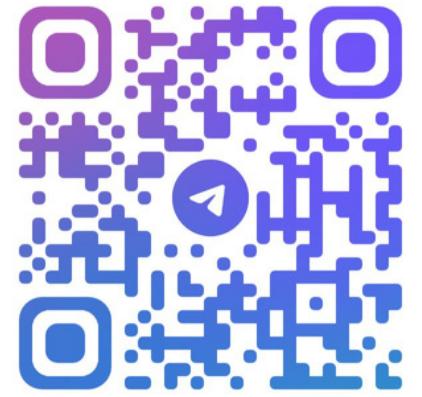
Difficulty

Skill Cat.

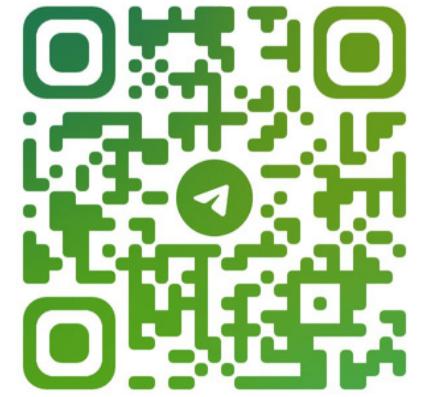
Total exp 8200

Type CTF

No. Quests 3



@STARKNET_ES



@DEFI_LAB



@ESCUELACRYPTOES



STARKWARE

Fin Taller 3

Gracias

