



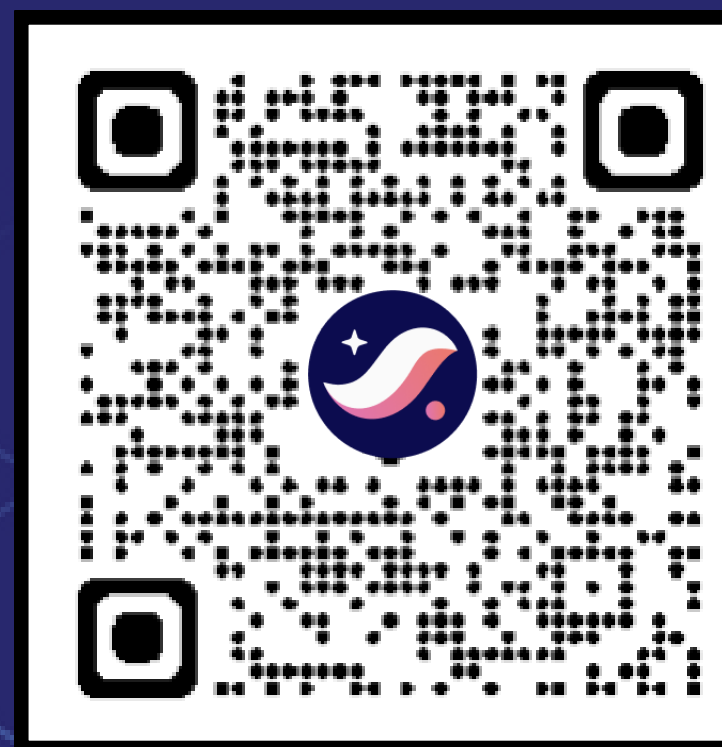
Nadai



Carlos



STARKWARE



Sesión 3: Las matemáticas detrás de las STARKs

● 4 de Mayo del 2023

Introducción



@Nadai02010



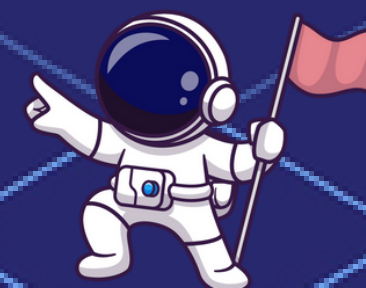
<https://github.com/Nadai2010>



@0xhasher_



<https://github.com/cliraa>



¿Qué es Starknet?

Starknet es una solución de escalabilidad de Capa 2 de Validity Rollup.
Lo que significa que **agrupa** muchas transacciones en una sola transacción y la **despliega** en la cadena principal de Ethereum.



Starknet utiliza matemáticas y
criptografía para escalar
Ethereum de forma segura.

STARKs



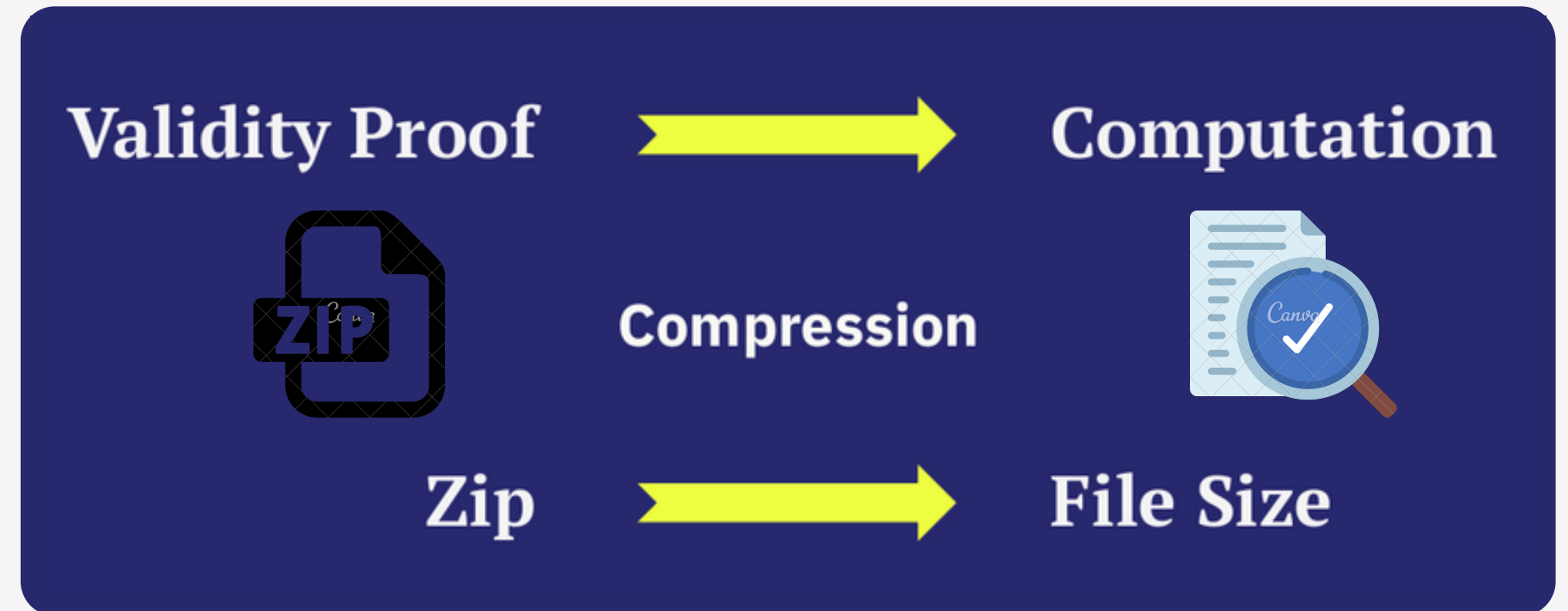
Presenta: Nadai y Carlos
Pioneros Starknet | CLASE 3

ZK Proof

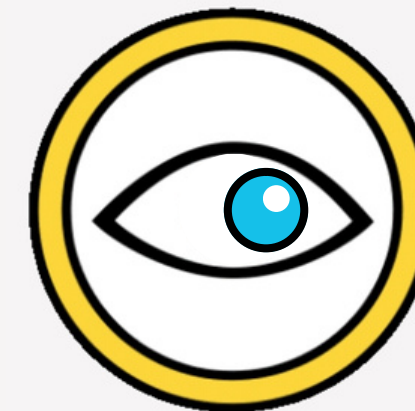
Una aplicación de **Zero Knowledge** (ZK Proof), utiliza pruebas **ZK** para garantizar la **integridad computacional** (CI).

No se trata de privacidad, se trata de escalar.

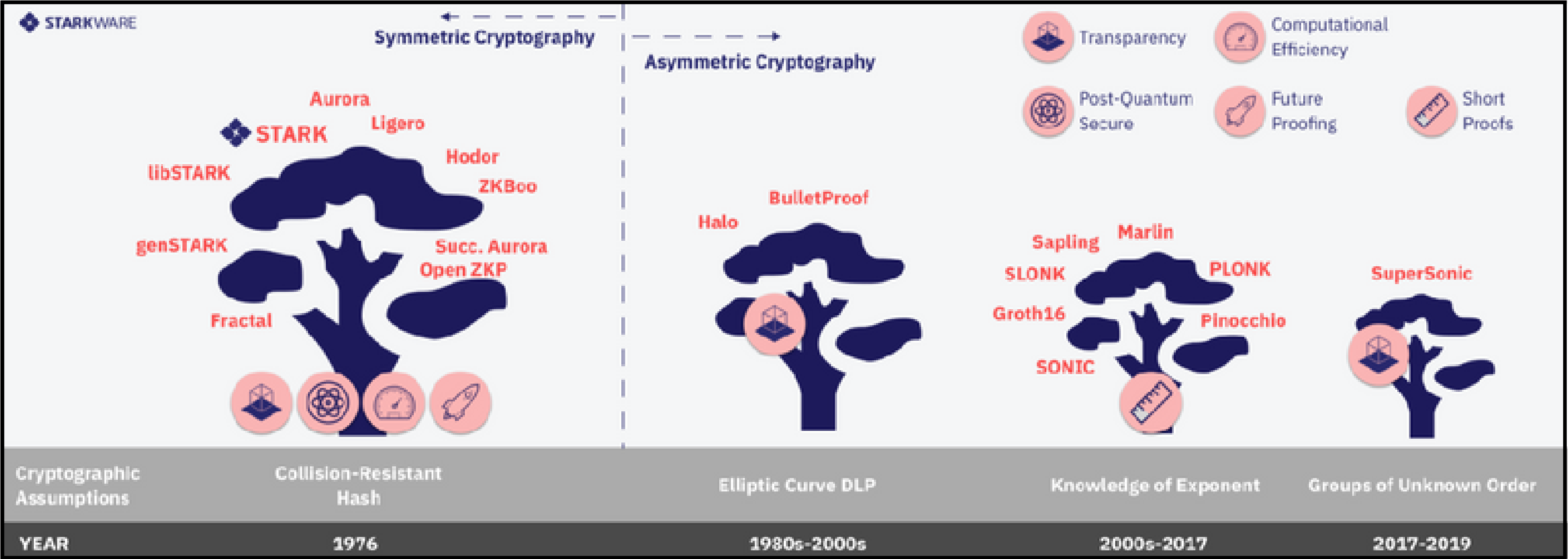
Validity Proof



"Hacer lo correcto incluso cuando nadie mira"



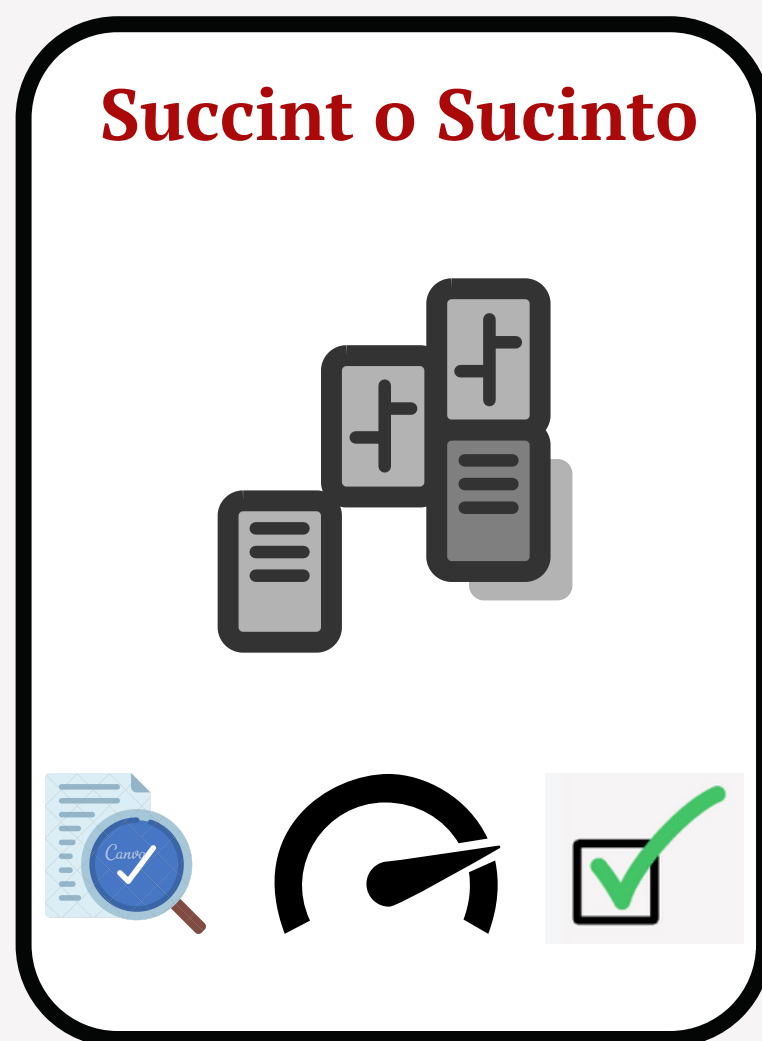
Tipos de pruebas según su criptografía



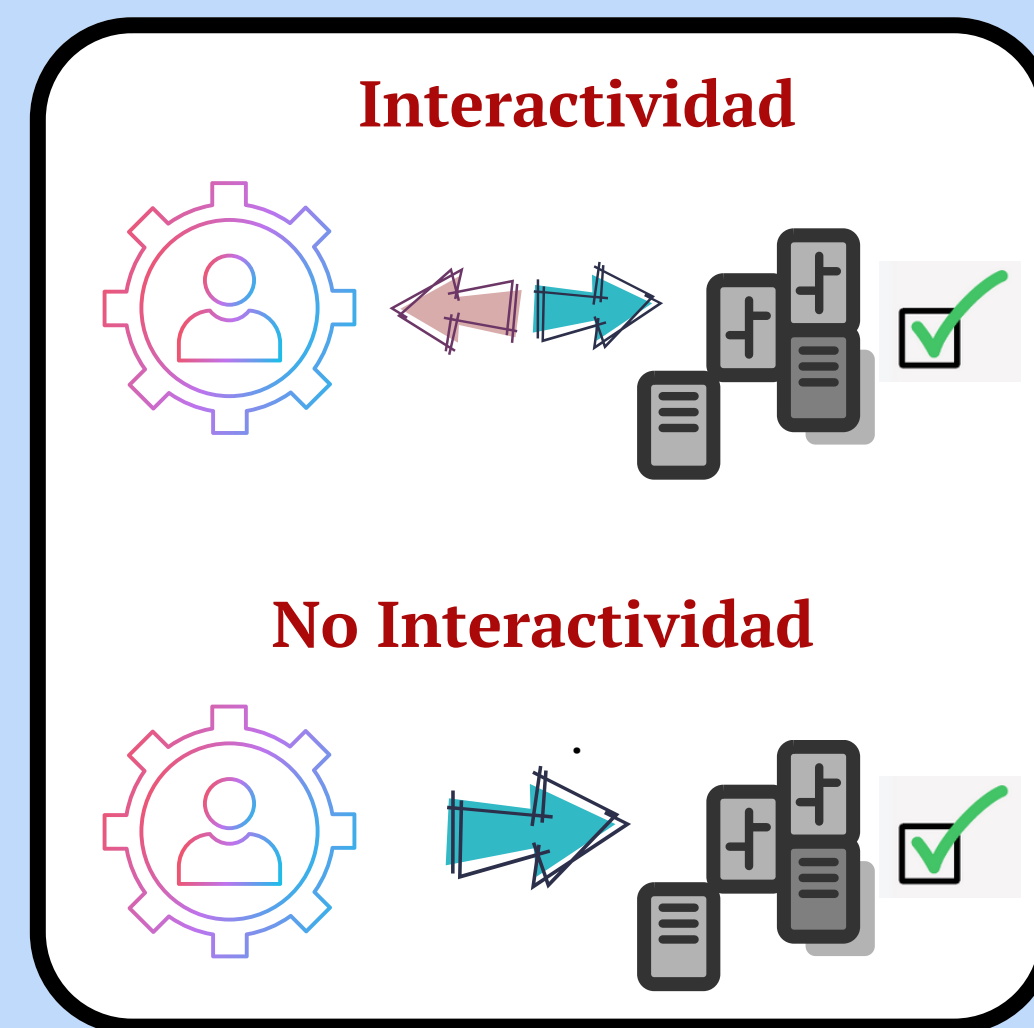
Presenta: Nadai y Carlos
Pioneros Starknet | CLASE 3

Propiedades de las Pruebas

Sucinta: Prueba que puede ser verificada de manera rápida y sencilla.



Interactividad: Forma en que **interactúan prover-verifier** con la Prueba.

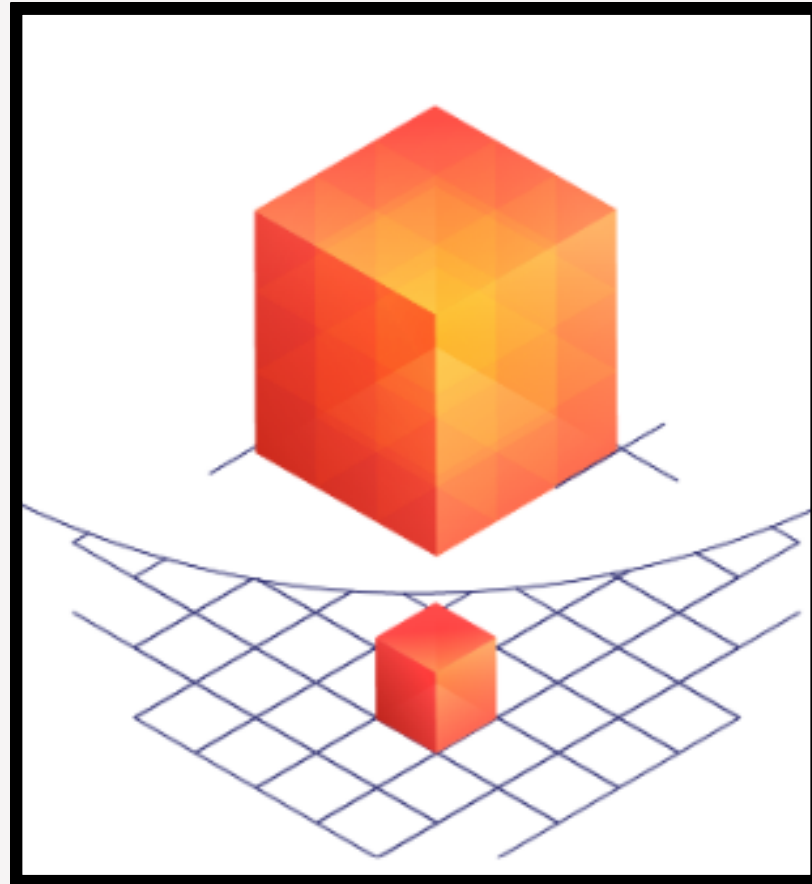


STARKs

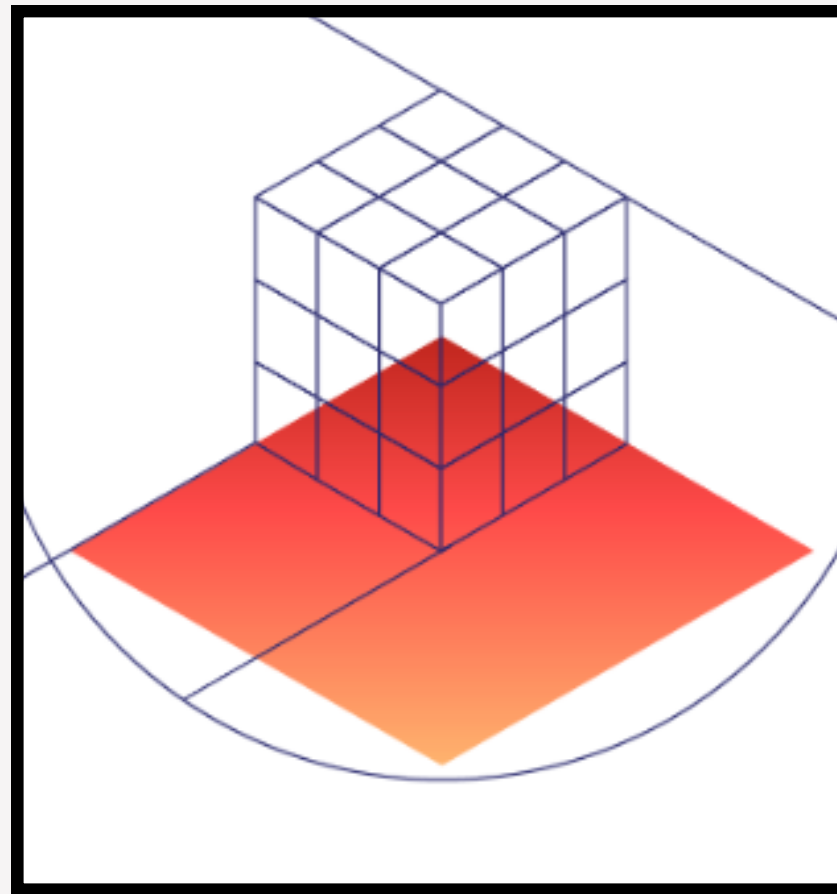
Presenta: Nadai y Carlos
Pioneros Starknet | CLASE 3

¿Qué es una STARKs?

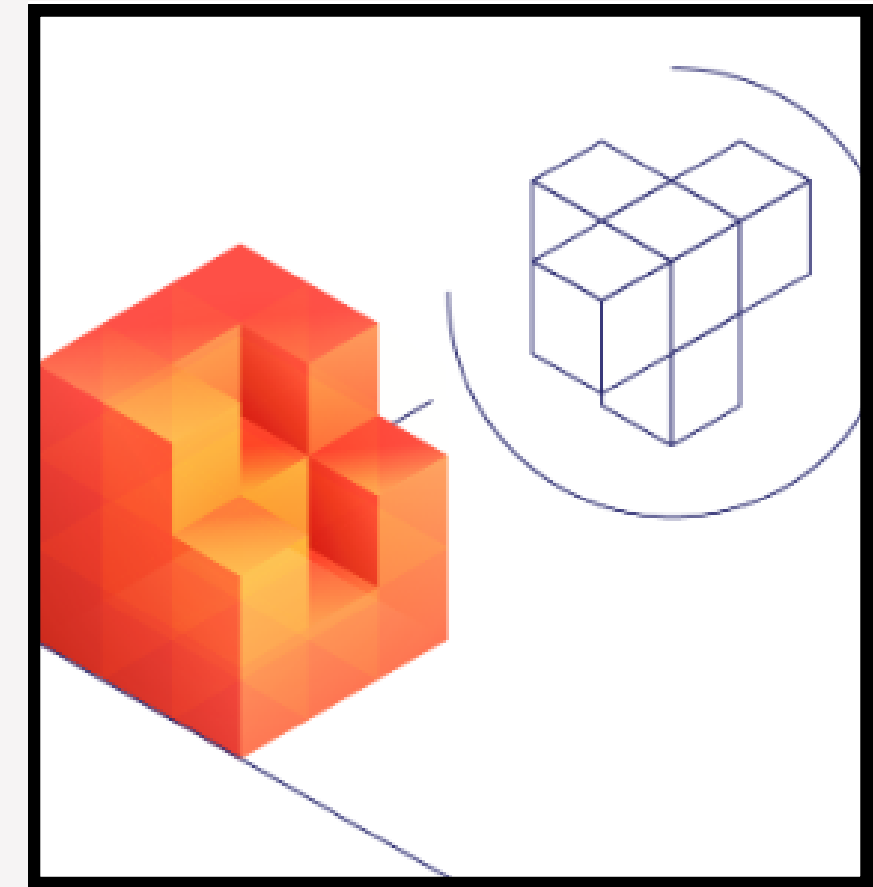
STARKs: Pruebas Escalables y Transparentes de Argumentos de Conocimiento



❖ Escalable



❖ Transparente

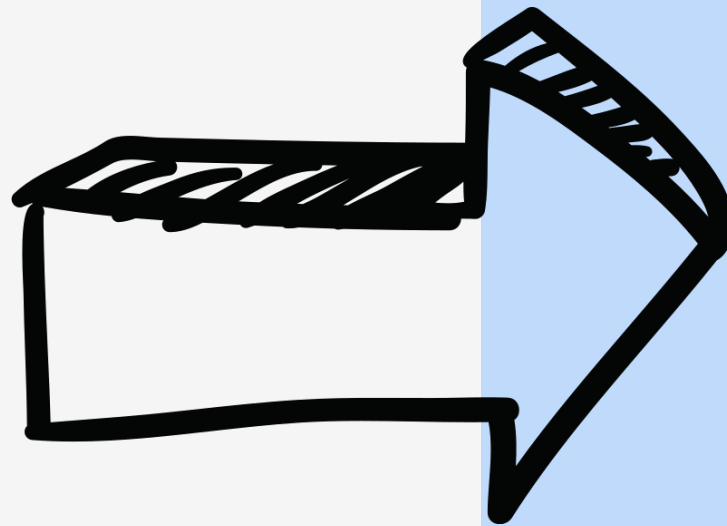
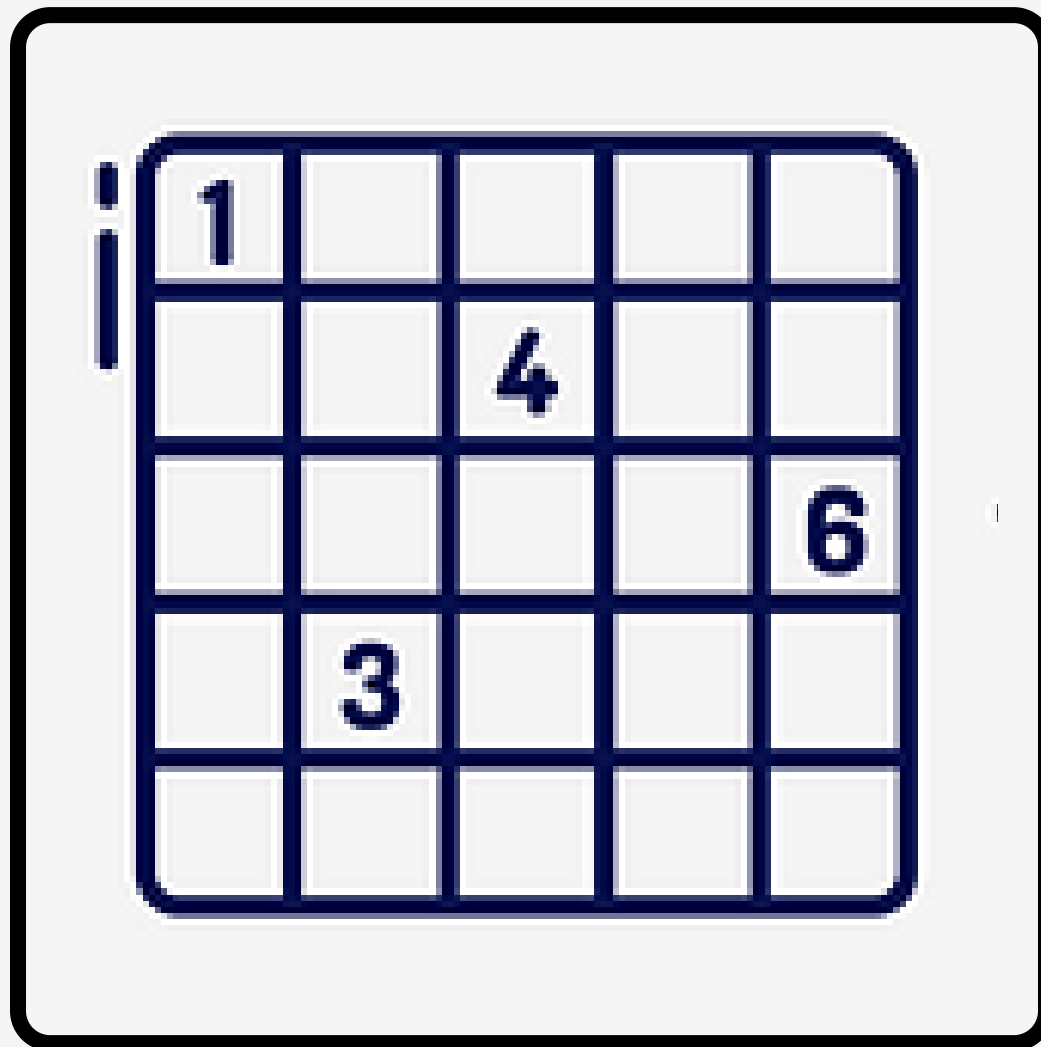


❖ Argumento de Conocimiento

Validity Proofs

Problema Sudoku 1

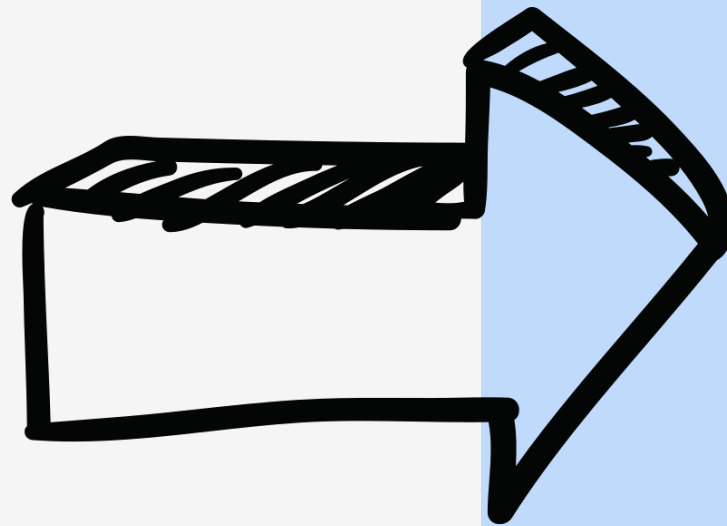
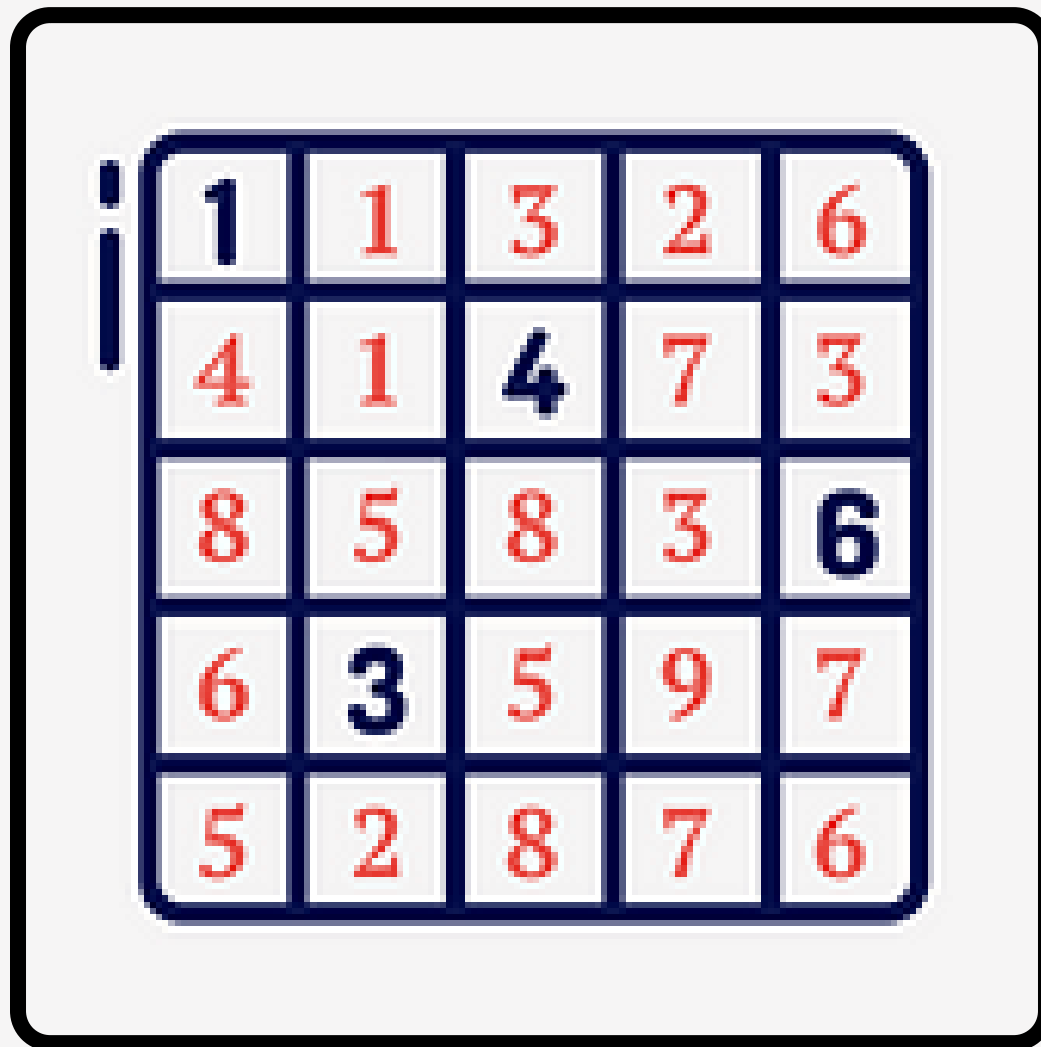
Resolviendo un SUDOKU entenderemos una Validity Proof (Prueba de Validez).



**Tenemos un
SUDOKU que
vamos a demostrar
que conocemos
alguna solución.**

Validity Proofs

Resolviendo un SUDOKU entenderemos una Validity Proof (Prueba de Validez).



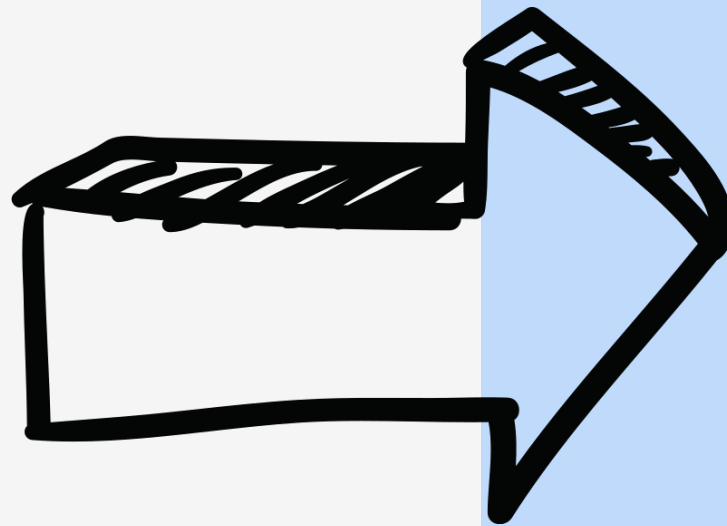
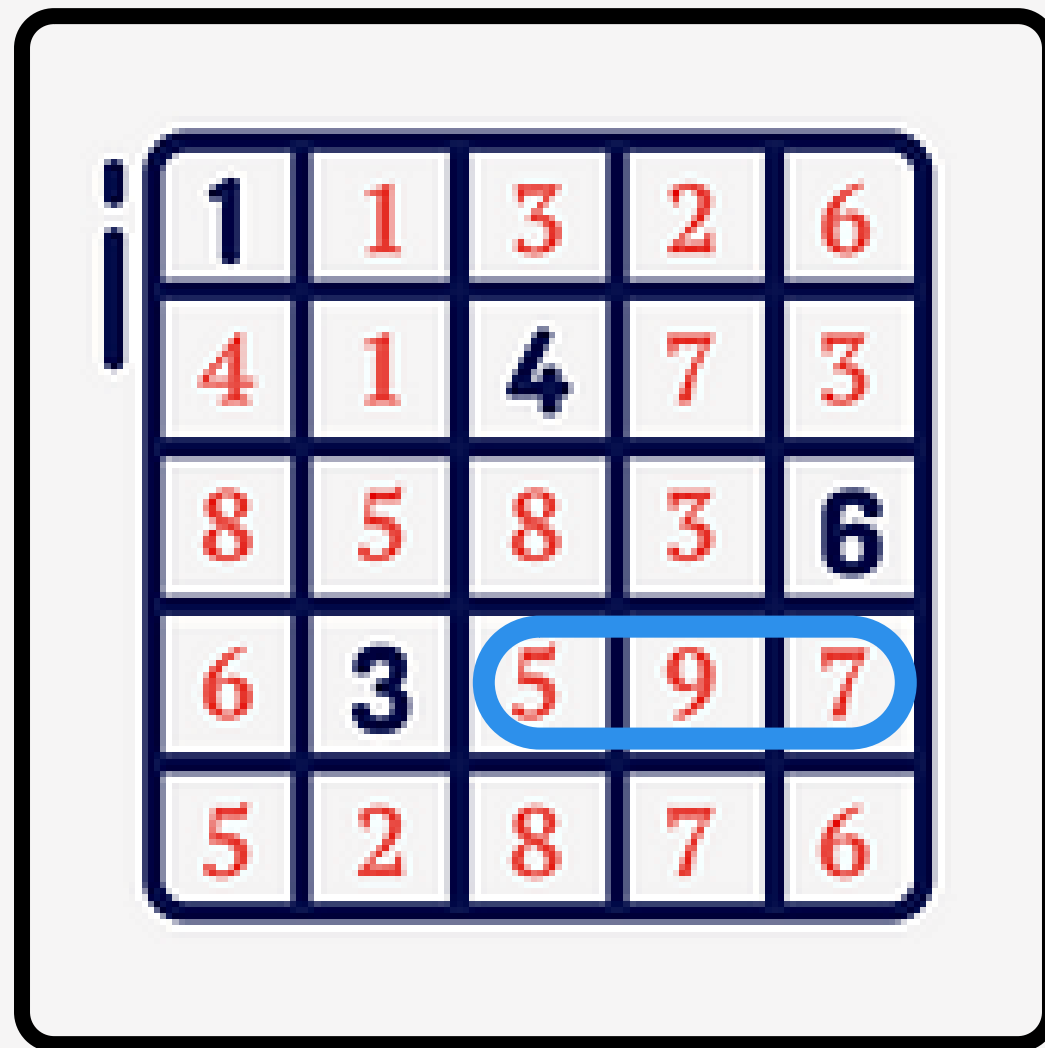
Problema Sudoku 2

Una vez resuelto
una parte, el
Prover lo entrega
al **Verifier** para
demostrar su
Validez.

Validity Proofs

Problema Sudoku 3

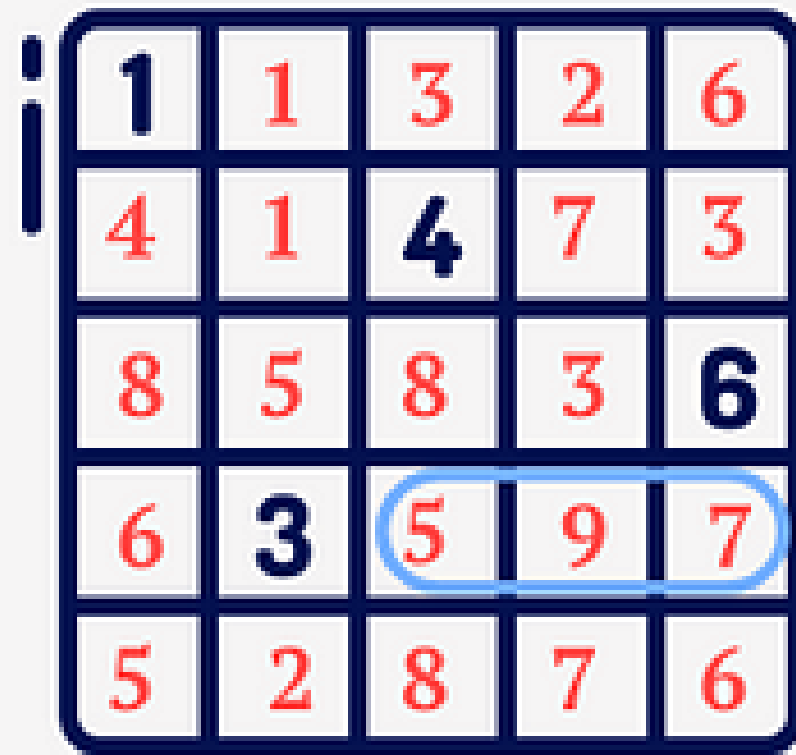
Resolviendo un SUDOKU entenderemos una Validity Proof (Prueba de Validez).



El **Verifier** con una
simple revisión
pueder comprobar
esa **Validez**

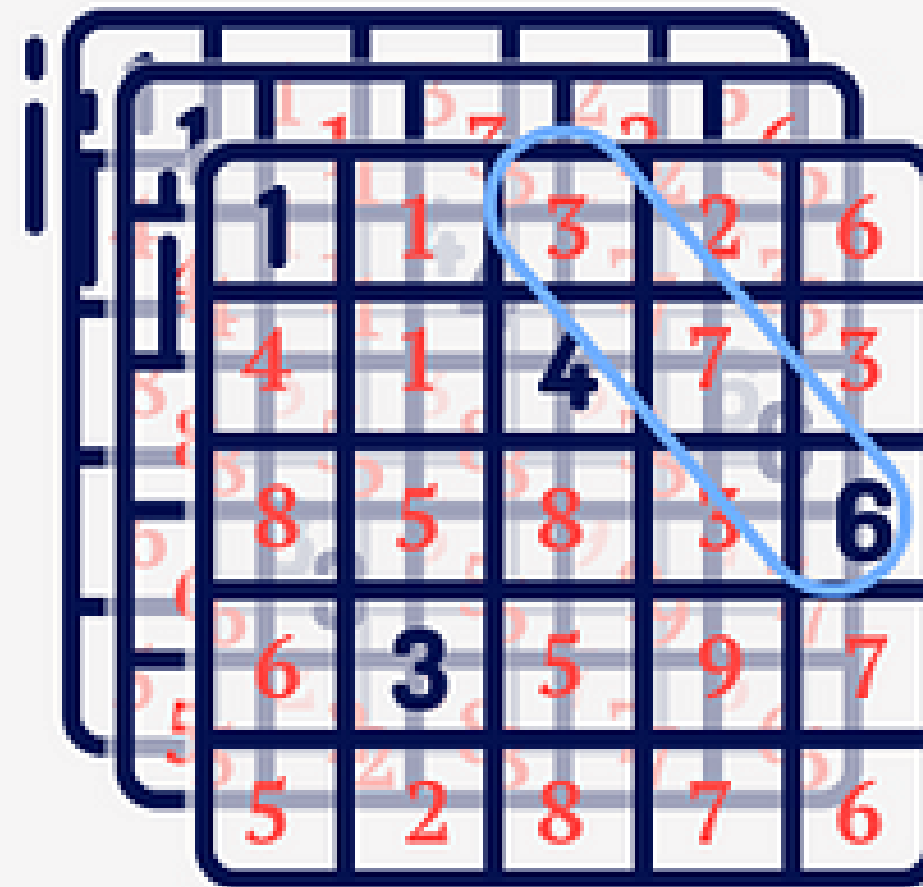
Validity Proofs - Múltiples verificaciones

Resolviendo un SUDOKU entenderemos una Validity Proof (Prueba de Validez).



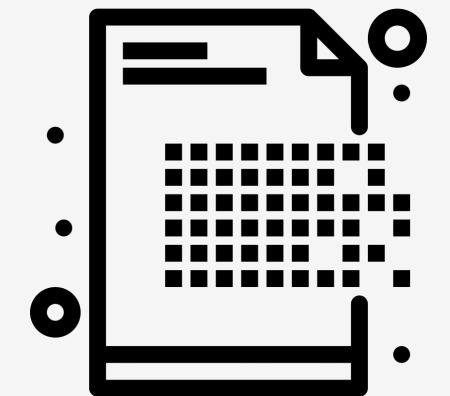
1	1	3	2	6
4	1	4	7	3
8	5	8	3	6
6	3	5	9	7
5	2	8	7	6

PCP



1	1	3	2	6
4	1	4	7	3
8	5	8	3	6
6	3	5	9	7
5	2	8	7	6

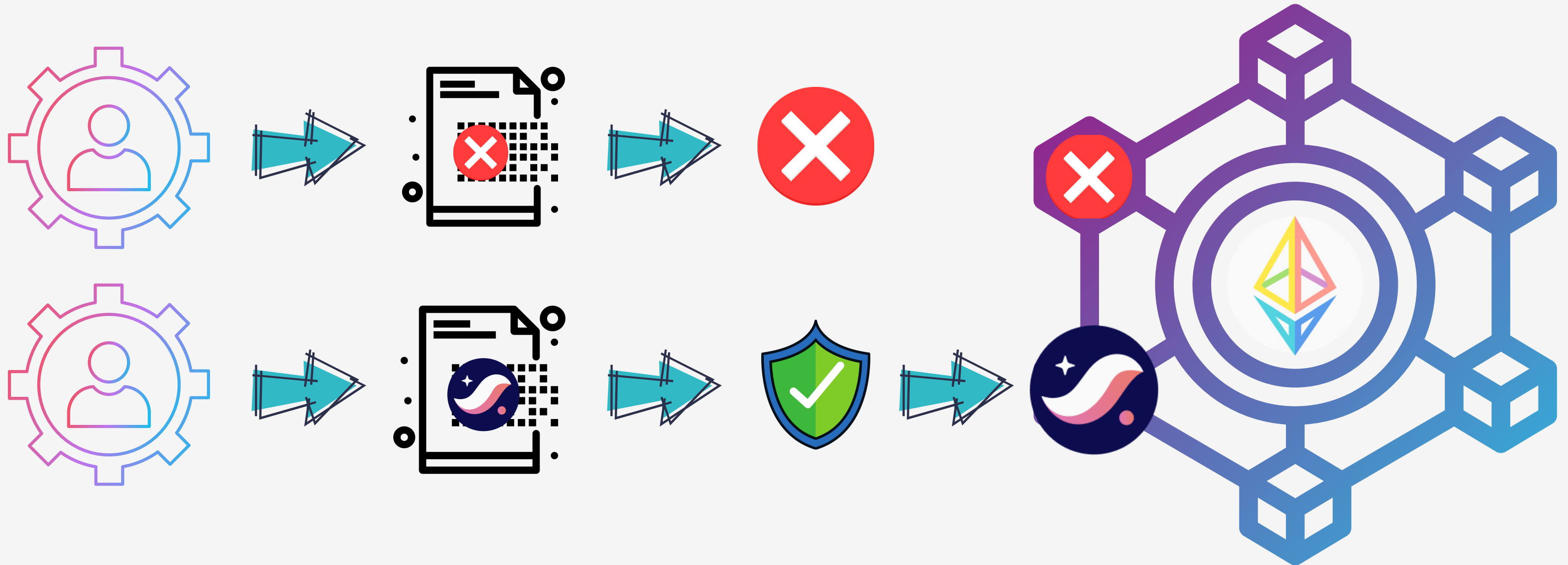
STARK



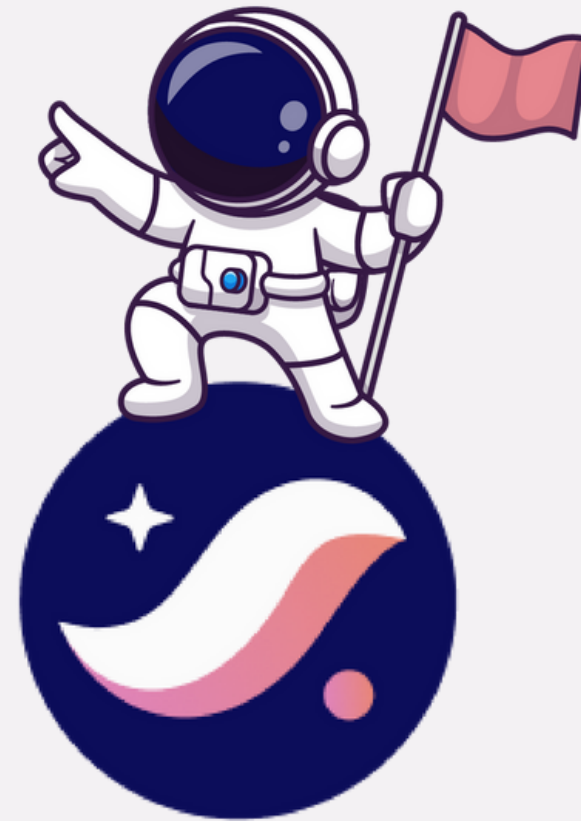
Presenta: Nadai y Carlos
Pioneros Starknet | CLASE 3

Introducción a STARKs y Validity Proofs

L1 no aceptará una prueba matemática errónea (recordamos que para L1, las L2 somos un Smart Contract)



Las matemáticas detrás de las STARKs

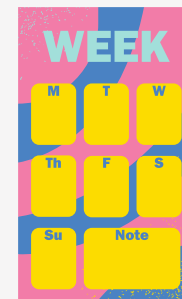


Presenta: Nadai y Carlos
Pioneros Starknet | CLASE 3

Conceptos Matemáticos - Campos

Los **grupos - campos o cuerpos**, conocidos como **Field**

Los **conjuntos** pueden tener elementos **finitos** o **infinitos**. El conjunto de los días de la semana (lunes a domingo), meses, edificios son **conjuntos finitos**.



El conjunto de los números enteros es un conjunto infinito y se designa \mathbb{Z} , "aquellos que no tienen decimales" por ejemplo, con:

$$\{\cdots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \cdots\}$$

Conceptos Matemáticos - Aritmética Mod



2 Horas desde
00:00

=

02:00 AM

$$2 \bmod 12 = 2$$



14 Horas desde
00:00

=

02:00 PM

$$14 \bmod 12 = 2$$



26 Horas desde
00:00

=

02:00 AM (Día Siguiente)

$$26 \bmod 12 = 2$$

Conceptos Matemáticos - Aritmética Mod



2 Horas desde
00:00
=
02:00 AM

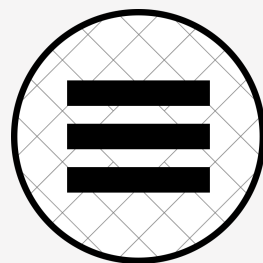


14 Horas desde
00:00
=
02:00 PM

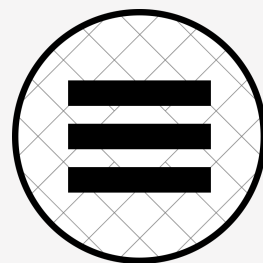


26 Horas desde
00:00
=
02:00 AM (Día Siguiente)

$2 \bmod 12 = 2$
Cociente = 0
 $\text{Resto} = 2$



$14 \bmod 12 = 2$
Cociente = 1
 $\text{Resto} = 2$



$26 \bmod 12 = 2$
Cociente = 2
 $\text{Resto} = 2$

Conceptos Matemáticos - Aritmética Mod



La aritmética modular es un conjunto de operaciones matemáticas que se realizan en un sistema numérico finito.

La suma, resta, multiplicación y división son algunas de las operaciones básicas en la aritmética modular.

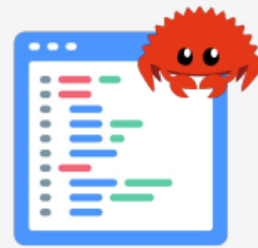
Ejemplos Prácticos

Presenta: Nadai y Carlos
Pioneros Starknet | CLASE 3

Conceptos Matemáticos - Pruebas

Analicemos las matemáticas detrás de estos códigos en Rust y Python ejecutando algunas operaciones en aritmética modular.

Rust



```
fn mod_add(a: i64, b: i64, m: i64) -> i64 {
    // Retorna la suma (a + b) % m
    (a + b) % m
}

fn mod_subtract(a: i64, b: i64, m: i64) -> i64 {
    // Retorna la resta (a - b) % m
    (a - b) % m
}

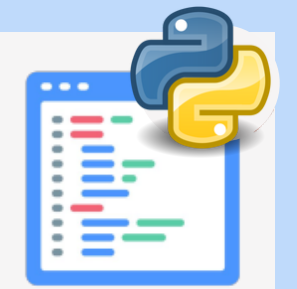
fn mod_multiply(a: i64, b: i64, m: i64) -> i64 {
    // Retorna el producto (a * b) % m
    (a * b) % m
}

fn mod_divide(a: i64, b: i64, m: i64) -> i64 {
    // Retorna la división (a / b) % m
    // Nota: Esto solo es válido si m es primo y b es coprimo con m
    // En otras palabras, b no tiene factores en común con m
    let b_inv = b.pow((m - 2) as u32) % m;
    return (a * b_inv) % m;
}
```

La suma, resta, multiplicación y división son algunas de las operaciones básicas en la aritmética modular

Las STARKs suelen usar también Fibonacci al Cuadrado.

Python



```
def mod_add(a, b, m):
    # Retorna la suma (a + b) % m
    return (a + b) % m

def mod_subtract(a, b, m):
    # Retorna la resta (a - b) % m
    return (a - b) % m

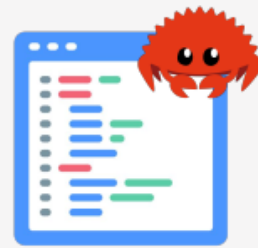
def mod_multiply(a, b, m):
    # Retorna el producto (a * b) % m
    return (a * b) % m

def mod_divide(a, b, m):
    # Retorna la división (a / b) % m
    # Nota: Esto solo es válido si m es primo y b es coprimo con m.
    # En otras palabras, b no tiene factores en común con m
    b_inv = pow(b, m-2, m)
    return (a * b_inv) % m
```

Conceptos Matemáticos - Pruebas

Analicemos las matemáticas detrás de estos códigos en Rust y Python ejecutando algunas operaciones en aritmética modular.

Rust



```
fn main() {  
    // Suma modular de 15 y 7 modulo 10  
    let result = mod_add(15, 7, 10);  
    println!("{}", result); // Output: 2  
  
    // Resta modular de 15 y 7 modulo 10  
    let result = mod_subtract(15, 7, 10);  
    println!("{}", result); // Output: 8  
  
    // Producto modular de 15 y 7 modulo 10  
    let result = mod_multiply(15, 7, 10);  
    println!("{}", result); // Output: 5  
  
    // División modular de 15 y 7 modulo 10  
    let result = mod_divide(15, 7, 10);  
    println!("{}", result); // Output: 5  
}
```

Suma modular

$$(15 + 7) \bmod 10 = 2$$

Resta Modular

$$(15 - 7) \bmod 10 = 8$$

Multiplación Modular

$$(15 \times 7) \bmod 10 = 5$$

División Modular

$$(15 / 7) \bmod 10 = 5$$

Python



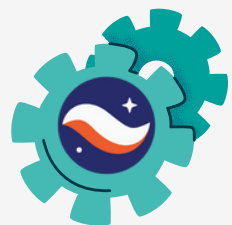
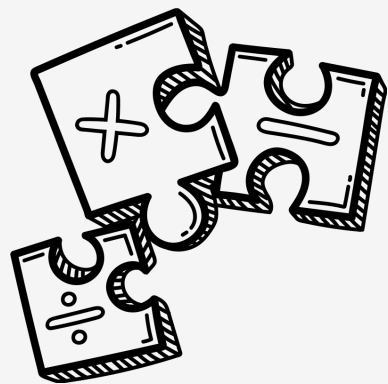
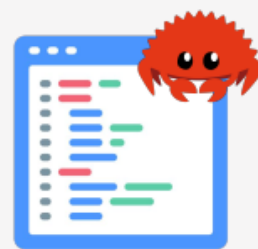
```
# Suma modular de 15 y 7 modulo 10  
result = mod_add(15, 7, 10)  
print(result) # Output: 2  
  
# Resta modular de 15 y 7 modulo 10  
result = mod_subtract(15, 7, 10)  
print(result) # Output: 8  
  
# Producto modular de 15 y 7 modulo 10  
result = mod_multiply(15, 7, 10)  
print(result) # Output: 5  
  
# División modular de 15 y 7 modulo 10  
result = mod_divide(15, 7, 10)  
print(result) # Output: 5
```


Conceptos Matemáticos - Pruebas

Analicemos las matemáticas detrás de estos códigos en Rust y Python ejecutando algunas operaciones en aritmética modular.

Rust

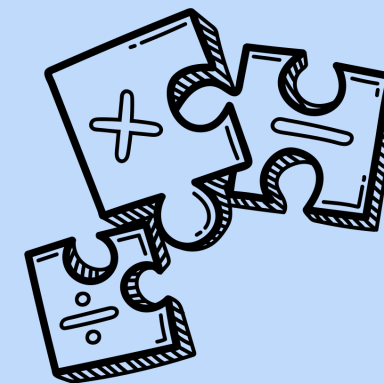
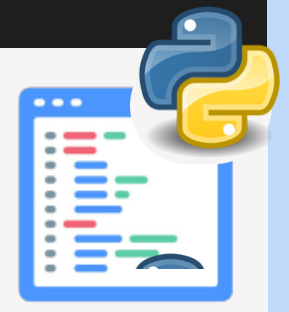
```
nadai@Nadai:~/Clases Maths/Contracts/Rust
2
8
5
5
```



Presenta: Nadai y Carlos
Pioneros Starknet | CLASE 3

Python

```
nadai@Nadai:~/Clases Maths$ /bin/python3
2
8
5
5
```



Suma modular

$$(15 + 7) \bmod 10 = 2$$

Resta Modular

$$(15 - 7) \bmod 10 = 8$$

Multiplación Modular

$$(15 \times 7) \bmod 10 = 5$$

División Modular

$$(15 / 7) \bmod 10 = 5$$

Comandos Rust y Python

Guía para Códigos de operaciones en Aritmética Modular en ambos lenguajes

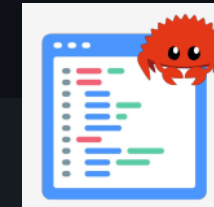
Aritmetica Modular Base

Para ejecutar los comandos dentro de la carpeta donde se encuentra nuestro archivo Rust y ejecute

```
rustc Aritmetica_Modular_Base.rs
```

Se generará un archivo el cual puede luego ejecutar corriendo en su terminal:

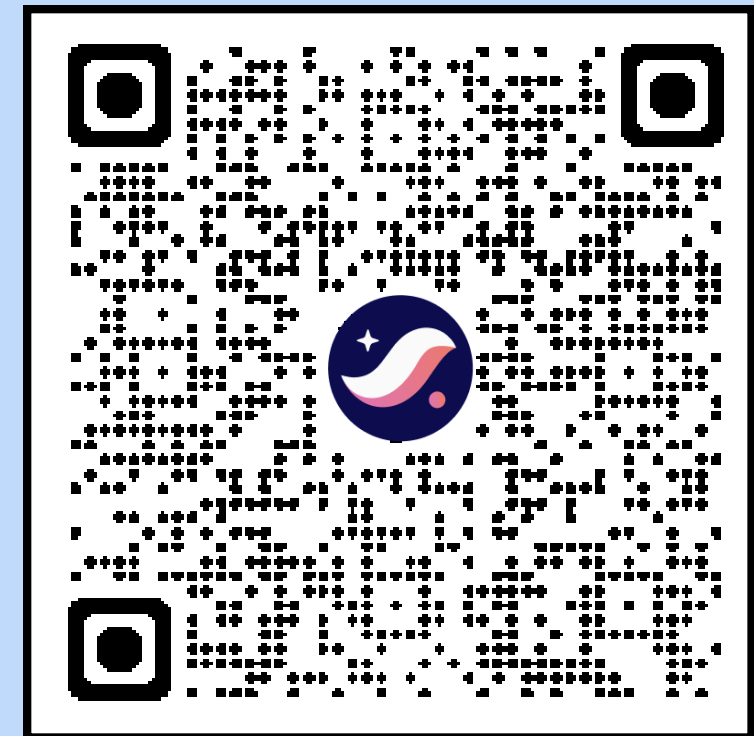
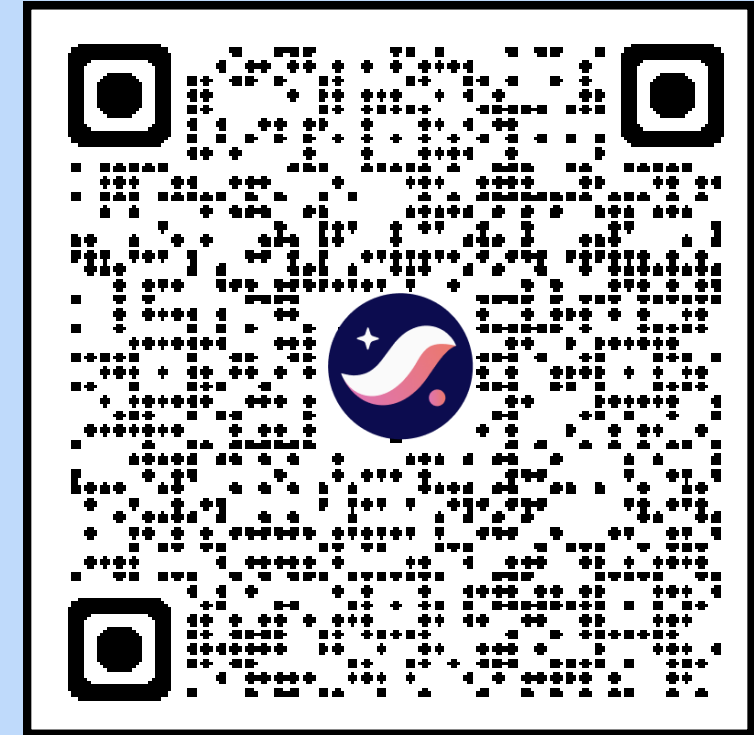
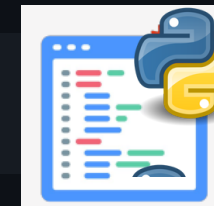
```
./Aritmetica_Modular_Base
```



Aritmetica Modular Base

Para ejecutar los comandos dentro de la carpeta donde se encuentra nuestro archivo Python ejecute

```
python3.9 Aritmetica_Modular_Base.py
```

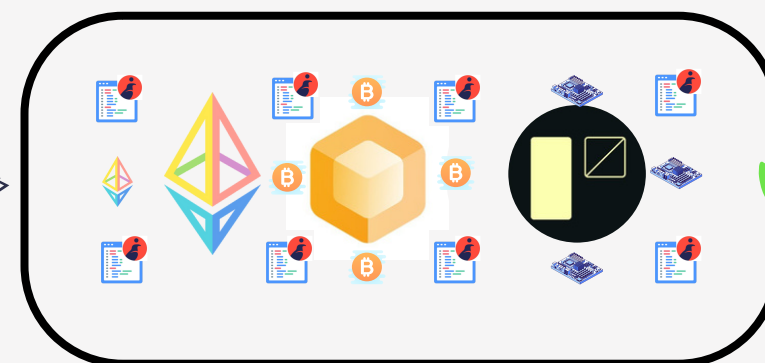
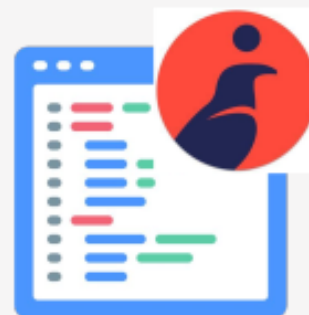


Presenta: Nadai y Carlos
Starknet Pioneros | CLASE 3

STARKs

Las STARKs están estableciendo unas bases matemáticas sólidas en la criptografía moderna

	STARK	SNARK
Verificación	$\log^2(n)$	constant
Tamaño de Prueba	~400 KB	288 bytes
Configuración Confiable	No	Si
Quantum Segura	Si	No



¡GRACIAS!