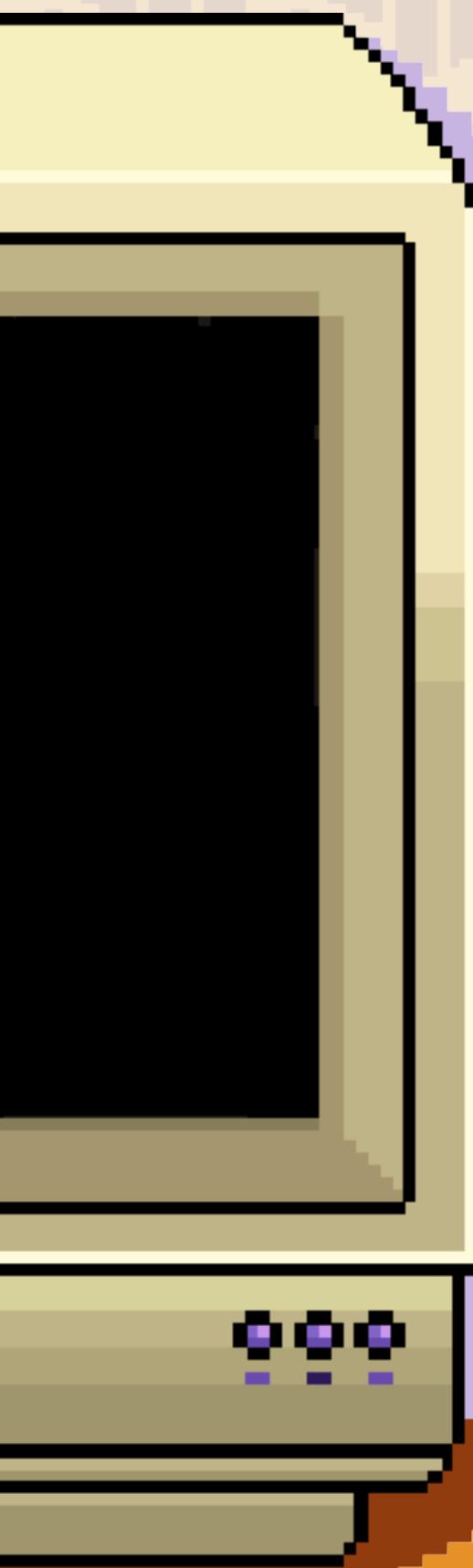


HEAVY DUTY CAMP V4

INVOCACIÓN
ENTRE
PROGRAMAS
(CPI)

CLASE
#5



PDAS

- Estas direcciones no están asociadas a una clave privada, lo que significa que solo el programa que las genera puede firmar transacciones que afecten esas direcciones.
- Se utilizan para crear cuentas que pueden almacenar datos asociados con un programa de manera segura. Esto es útil para almacenar estados o registros en programa en Solana

PROGRAMAS EN SOLANA

- Los programas (contratos inteligentes) son piezas de código que se ejecutan en la blockchain para realizar tareas específicas.
- Cada programa está compuesto por una o más instrucciones que pueden ser invocadas desde otro programa diferente.

INVOCACIÓN ENTRE PROGRAMAS

- Estas llamadas a instrucciones de otros programas se conocen como CPIs (Cross-Program Invocations)
- Las CPIs permiten que un programa llame una instrucción específica definida en otro programa dentro de la blockchain de Solana.

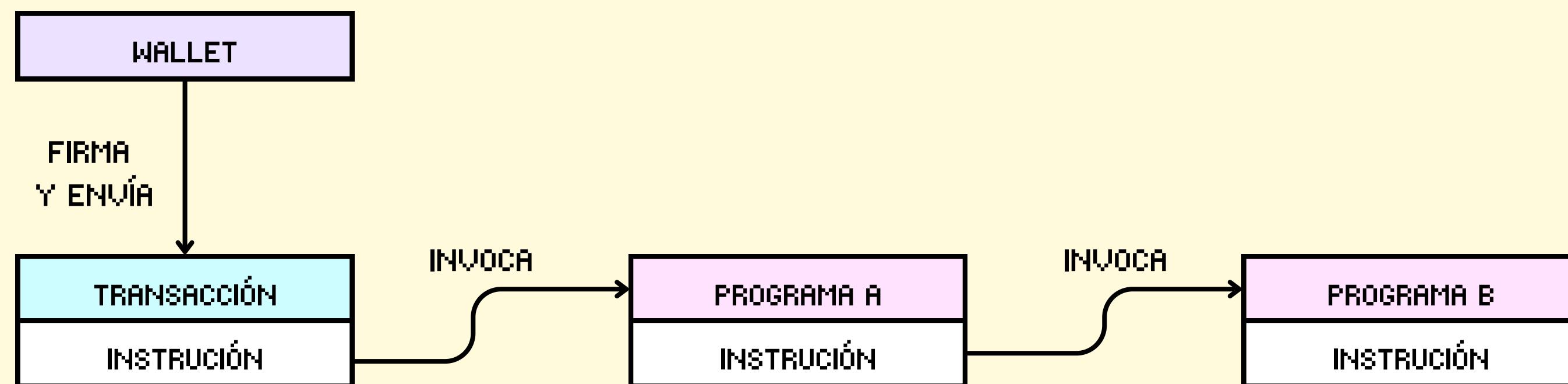
INVOCACIÓN ENTRE PROGRAMAS

- Se puede pensar en las instrucciones de un programa como endpoints API que el programa expone a la red.
- En este contexto, una CPI equivale al caso en que una API internamente invoca otra API.

INVOCACIÓN ENTRE PROGRAMAS

- Existen dos tipos de invocación entre programas (CPIs) dependiendo de si las cuentas involucradas en las firmas incluyen una PDA o no.
- En el caso de las PDAs el programa firma la transacción haciendo usos de las semillas de la PDA.

INVOCACIÓN ENTRE PROGRAMAS



INVOCANDO UNA INSTRUCCIÓN

- Dada una transacción al programa A que invoca al programa B, El programa invocado (B) puede hacer CPI a otros programas, hasta una profundidad máxima de 4 (ej: programa B -invoca-> programa C, programa C -invoca-> programa D, etc)

PILA DE INVOCACIÓN

- El tiempo de ejecución del programa Solana define una constante llamada max_invoke_stack_height, que se establece en el valor de 5. Esto representa la altura máxima de la pila de la invocación de instrucciones del programa.
- La altura de la pila comienza en 1 para las instrucciones de una transacción, incrementa en 1 cada vez que un programa invoca otra instrucción.

CPIs: EJEMPLOS

- Interacción con SPL Token: La mayoría de los programas necesitan invocar el programa SPL Token para:
 - Crear nuevas cuentas de token.
 - Realizar transferencias de tokens.
 - Configurar permisos o congelar cuentas.

PERMISOS PARA FIRMAR

- Dado que el programa que llama en una CPI no tiene control sobre las cuentas o los datos que se pasan al programa invocado, es fundamental que el programa invocado verifique todos los parámetros. Esto garantiza que los datos maliciosos o incorrectos no comprometan la seguridad del programa.

PERMISOS PARA FIRMAR

- Los privilegios de firmante hacen referencia a la capacidad otorgada a una cuenta para firmar transacciones y autorizar modificaciones a sus datos o estado. Estos garantizan que solo se puedan ejecutar acciones autorizadas en nombre de la cuenta.

FIRMANDO EN NOMBRE DE UNA PDA

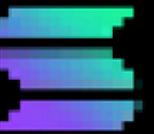
- Si bien los PDA no tienen claves privadas, pueden actuar como firmantes en una instrucción a través de una CPI.
- Para verificar que un PDA se deriva del programa que realiza la llamada, las semillas utilizadas para generar el PDA deben incluirse como signers_seeds.

FIRMANDO EN NOMBRE DE UNA PDA

- Para verificar que el programa tiene los permisos sobre la cuenta PDA parte de la firma la componen las semillas utilizadas para generar la PDA.
- Con las semillas, el entorno de ejecución verifica que el programa asociado con esta dirección sea el que llama y esté autorizado para ser el firmante.

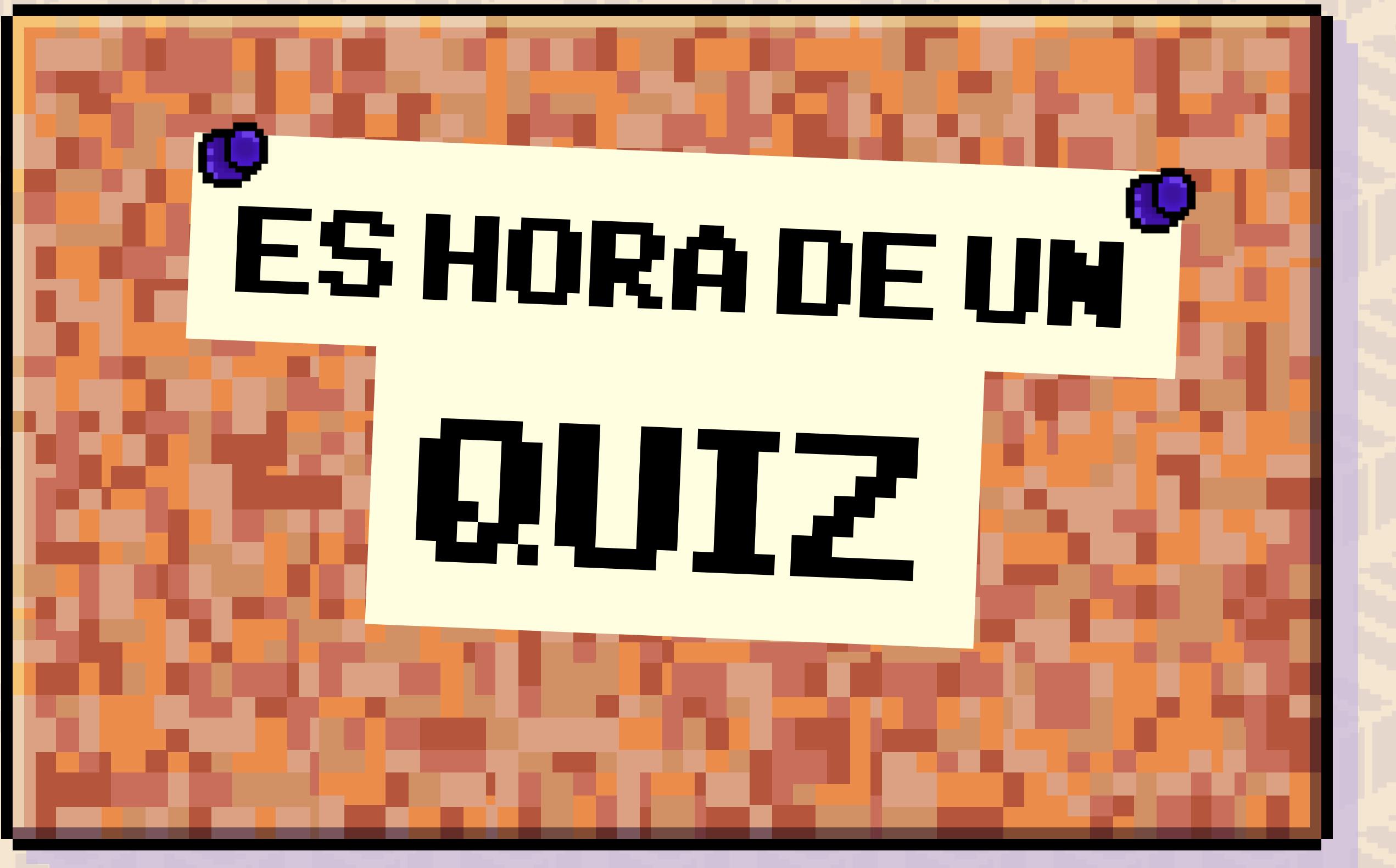
FIRMANDO EN NOMBRE DE UNA PDA

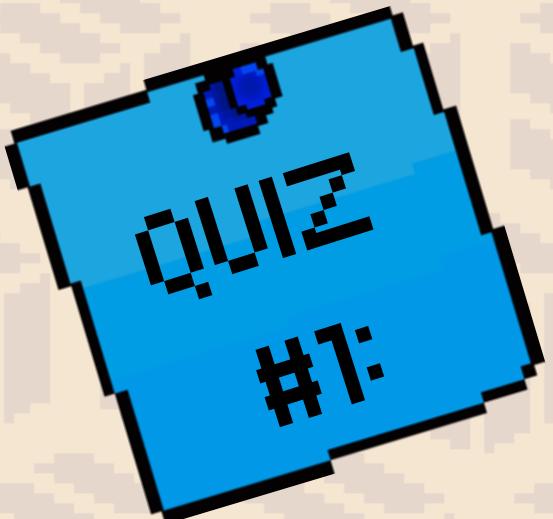
- Cuando se procesa la CPI, el entorno de ejecución de Solana llama internamente a create_program_address utilizando signers_seeds y el program_id del programa que realiza la llamada.
- Si se encuentra un PDA válido, la dirección se agrega como firmante válido.



19

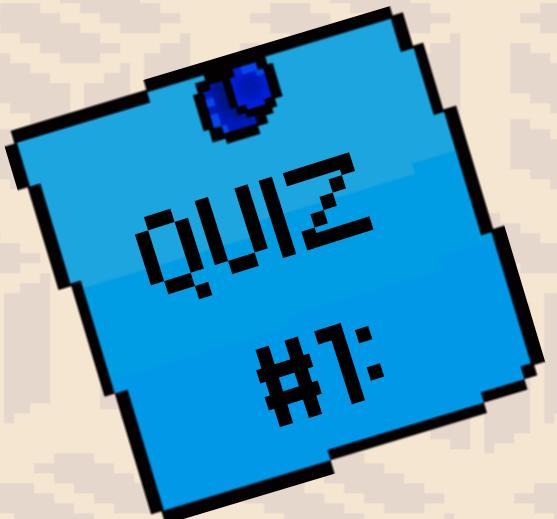
LUN	MAR	M
6	7	
13	14	
20	21	22
27	28	29





¿Cuál es el objetivo principal de una CPI en Solana?

- A. Evitar que los programas interactúen entre sí
- B. Optimizar la velocidad del programa
- C. Permitir que un programa invoque a otro programa
- D. Limitar las transacciones en un bloque



¿Cuál es el objetivo principal de una CPI en Solana?

- A. Evitar que los programas se ejecuten simultáneamente
- B. Optimizar la velocidad de ejecución de los programas
- C. Permitir que un programa invogue a otro programa
- D. Limitar las transacciones en un bloque

RESPUESTA CORRECTA:

- C. PERMITIR QUE UN PROGRAMA INVOCUE A OTRO PROGRAMA



ESCRIBIENDO UNA CPI

- Escribir una instrucción para un CPI sigue el mismo patrón que construir una instrucción para añadir a una transacción.
- Debe especificar el identificador del programa, las cuentas relacionadas con la instrucción y la información de la instrucción a ser ejecutada.

ESCRIBIENDO UNA CPI

- Las CPI se realizan utilizando los comandos `invoke` (CPI básica) o `invoke_signed`, y este último permite que los programas firmen en nombre de las direcciones derivadas de programa (PDA) que poseen.

CPI BÁSICA

- La función de invoke se utiliza al crear una CPI que no requiere firmantes de PDA. Al crear CPI, los firmantes proporcionados al programa que realiza la llamada se extienden automáticamente al programa que recibe la llamada.

CPI BÁSICA

```
pub fn invoke(  
    instruction: &Instruction,  
    account_infos: &[AccountInfo<'_>]  
) -> Result<(), ProgramError>
```

CPI CON PDA

- La función `invoke_signed` se utiliza al crear una CPI que requiere firmantes PDA. Las semillas utilizadas para derivar los firmantes PDA se pasan a la función `invoke_signed` como `signer_seeds`.

CPI CON PDA

```
pub fn invoke_signed(  
    instruction: &Instruction,  
    account_infos: &[AccountInfo<'_>],  
    signers_seeds: &[&[&[&[u8] ] ]]  
) -> Result<(), ProgramError>
```

CPI CON PDA

- La función `invoke_signed` se utiliza al crear una CPI que requiere firmantes PDA. Las semillas utilizadas para derivar los firmantes PDA se pasan a la función `invoke_signed` como `signer_seeds`.

CPI'S EN ANCHOR

- En anchor existen diferentes enfoques para definir una CPI, el primero implica la creación de un CpiContext, que incluye el program_id y las cuentas requeridas para la instrucción que se llama, seguido de una función auxiliar para invocar una instrucción específica.

CPI CONTEXT

```
let cpi_context = CpiContext::new(  
    program_id,  
    Transfer {  
        from: from_pubkey,  
        to: to_pubkey,  
    },  
);
```

CPI CONTEXT

- Internamente el CpiContext es un contenedor alrededor de la función de invoke del crate solana_program que utiliza un método específico definido en system_instruction para construir la instrucción.

CPIS EN ANCHOR

- También puede crear manualmente la instrucción para pasarla a la función `invoke()`.
- Esto resulta útil cuando no hay un `crate` disponible en `Anchor` para ayudar a crear la instrucción que desea invocar.

CPI'S EN ANCHOR: PDAS

- Estos enfoques aplican a los casos en los que las CPI involucran cuentas PDA.
- Al firmar con PDA, las semillas opcionales y la semilla de aumento se incluyen en CPI Context como `signer_seeds` usando `with_signer()`.

CPI'S EN ANCHOR: PDAS

```
let seed = to_pubkey.key();
let bump_seed = ctx.bumps.pda_account;
let signer_seeds: &[&[&[u8]]] = &[&[b"pda", seed.as_ref(), &bump_seed]];

let cpi_context = CpiContext::new(
    program_id,
    Transfer {
        from: from_pubkey,
        to: to_pubkey,
    },
)
.with_signer(signer_seeds);
```

CPIs EN ANCHOR: PDAS

- En esencia, el Cpi Context anterior es un contenedor de la función invoke_signed() que al igual que invoke() permite realizar CPIs con la diferencia de que requiere las semillas de la PDA como firmantes.

CPI'S EN ANCHOR: PDAS

```
pub fn sol_transfer(ctx: Context<SolTransfer>, amount: u64) -> Result<()> {
    let from_pubkey = ctx.accounts.pda_account.to_account_info();
    let to_pubkey = ctx.accounts.recipient.to_account_info();
    let program_id = ctx.accounts.system_program.to_account_info();

    let seed = to_pubkey.key();
    let bump_seed = ctx.bumps.pda_account;
    let signer_seeds: &[&[&[u8]]] = &[&[b"pda", seed.as_ref(), &[bump_seed]]];

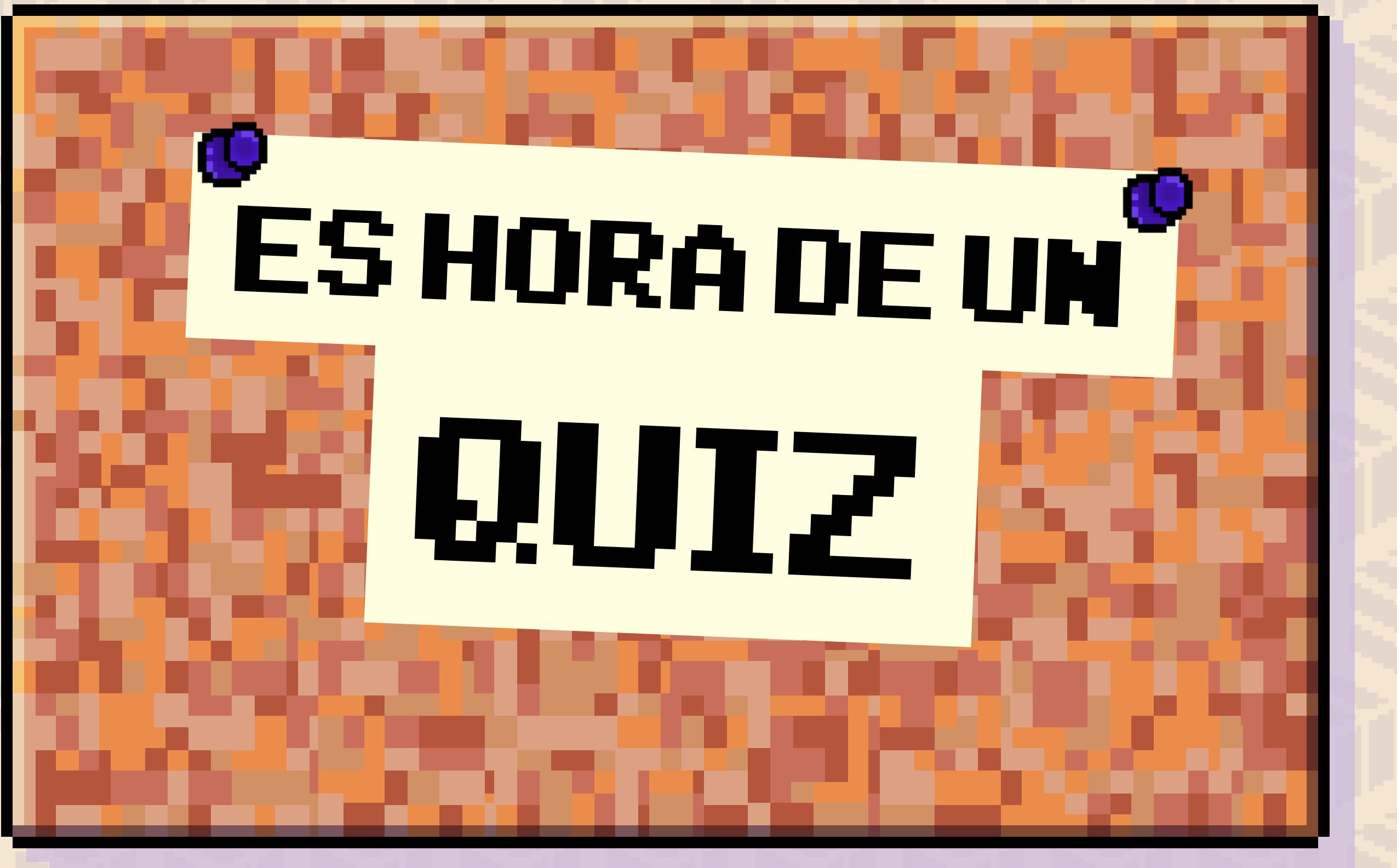
    let instruction =
        &system_instruction::transfer(&from_pubkey.key(), &to_pubkey.key(), amount);

    invoke_signed(instruction, &[from_pubkey, to_pubkey, program_id], signer_seeds)?;
    Ok(())
}
```



19

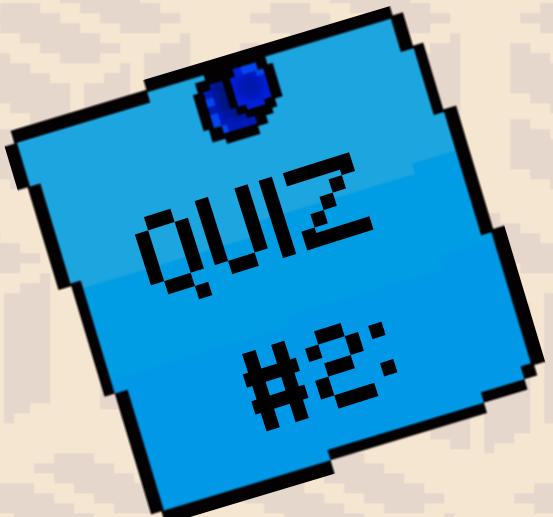
LUN	MAR	M
6	7	
13	14	
20	21	22
27	28	29





¿Qué función se utiliza en Anchor para realizar una CPI básica?

- A. anchor_call
- B. invoke_signed
- C. invoke_program
- D. invoke



¿Qué función se utiliza en Anchor para realizar una CPI básica?

- A. anchor_call
- B. invoke_signed
- C. invoke_program
- D. invoke

**RESPUESTA
CORRECTA:**

D. INVOCAR



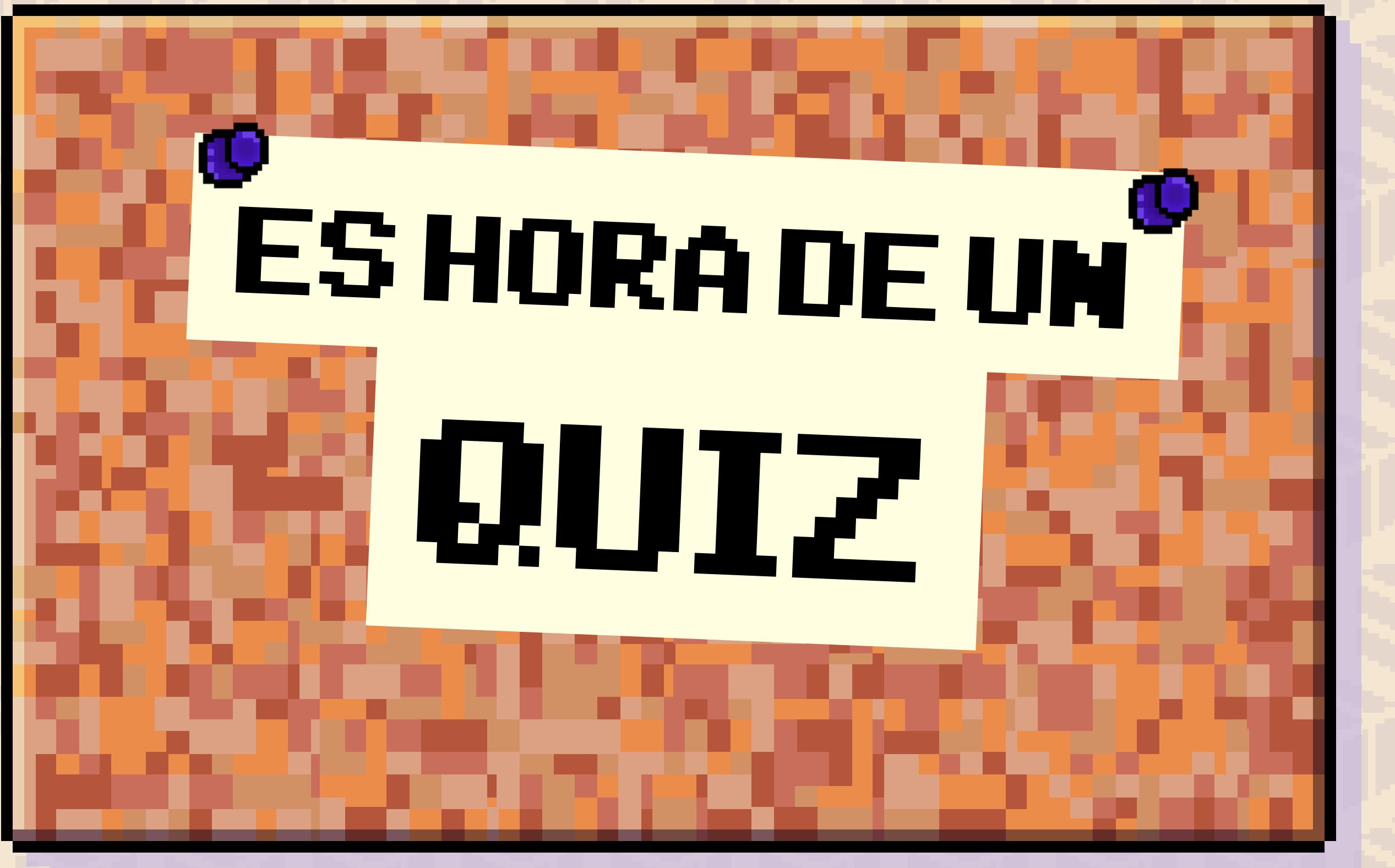
CPIS EN ANCHOR: EJEMPLO

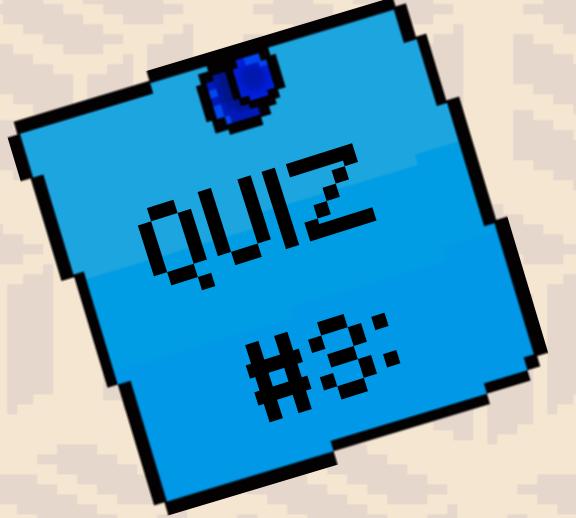
- Un caso de uso típico para las PDAs es un programa que administra cuentas de tokens en nombre de los usuarios.
- Esto requiere el uso de PDA como propietario de las cuentas de tokens para firmar programáticamente retiros de las cuentas.



19

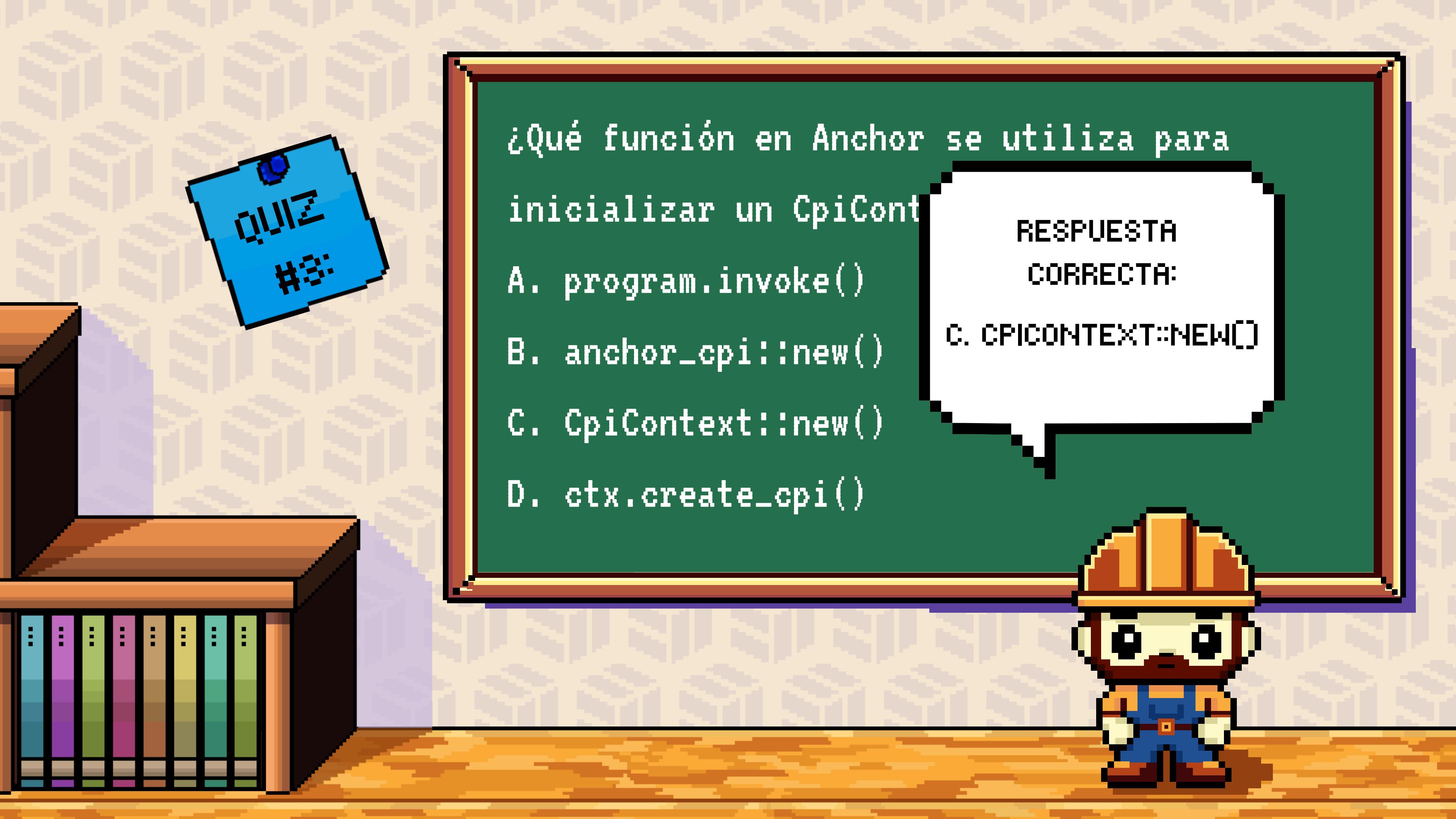
LUN	MAR	M
6	7	
13	14	
20	21	22
27	28	29





¿Qué función en Anchor se utiliza para inicializar un CpiContext?

- A. program.invoke()
- B. anchor_cpi::new()
- C. CpiContext::new()
- D. ctx.create_cpi()



¿Qué función en Anchor se utiliza para inicializar un CpiCont

- A. program.invoke()
- B. anchor_cpi::new()
- C. CpiContext::new()
- D. ctx.create_cpi()

RESPUESTA
CORRECTA:

C. CPICONTEXT::NEW()

HEAVY DUTY CAMP V4

NOS VEMOS
EN LA
PROXIMA
CLASE!

CLASE
#5