INPUT | OUTPUT

Pruebas

¿DE QUÉ SE TRATAN LAS PRUEBAS? _

¿De qué se tratan las pruebas?

- Tener confianza en la precisión de tu programa.
- Mostrar que casos comunes funcionan correctamente.
- Mostrar que casos atípicos son tratados correctamente.
- Las pruebas no garantizan la ausencia de errores.

+ PRECISIÓN_

Precisión

- ¿Cuándo un programa está correcto?
- ¿Qué es una especificación?
- ¿Cómo establecer una relación entre la especificación y la implementación?
- ¿Qué pasa con los errores en la especificación?

AIKEN

Pruebas

Pruebas

- Aiken ofrece soporte **first-class** para pruebas unitarias y property-based tests.

- El comando `aiken check` puede analizar pruebas, recompilarlas, ejecutarlas y mostrar un informe detallado.

- Puedes escribir pruebas **en cualquier parte** de un módulo Aiken, y pueden hacer **llamadas a funciones y usar constantes** de la misma manera.

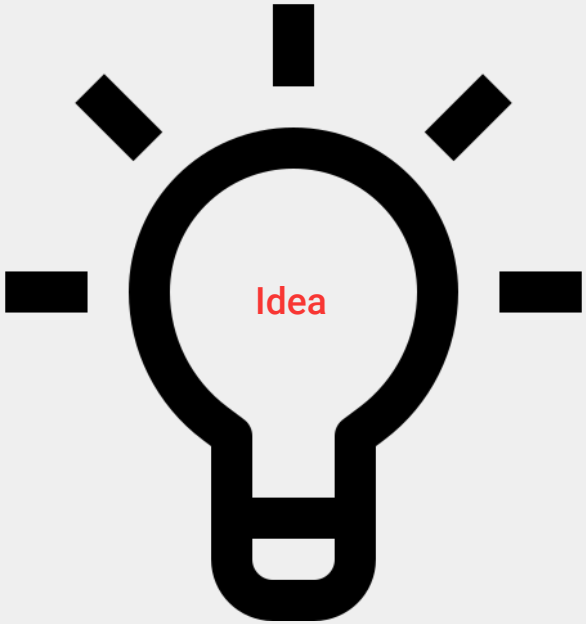
- Las pruebas usan la **misma máquina virtual** que la utilizada para ejecutar contratos on-chain (mismo contexto que tu **código de producción**) .

👣 Escribir pruebas para verificar:

Escribir pruebas para el smart contracts **“negativo después del plazo”**

1. `lower_bound` es 100 con `deadline == 50` y `redeemer = -1` pasa.
2. `lower_bound` es 100 con `deadline == 150` y `redeemer = -1` falla.
3. `lower_bound` es 100 con `deadline == 50` y `redeemer = 1` falla.
4. `lower_bound` es 100 con `deadline == 150` y `redeemer = 1` falla.

+ PROPERTY-BASED TESTING _



Property-based testing

Especificar formalmente las propiedades que debería tener el programa y probar nuestra implementación (código) contra esas propiedades.

Property-based testing: Puntos principales

- La Prueba Basada en Propiedades (PBT), al estilo de **QuickCheck**, es una metodología de pruebas popular.
- Es soportada en muchos lenguajes de programación modernos.
- Las pruebas se especifican utilizando **propiedades lógicas**.
- Estas pruebas se ejecutan automáticamente con entradas generadas **pseudo-aleatoriamente** en busca de **contraejemplos**.
- PBT no solo es útil para encontrar errores en programas.
- También se ha utilizado para reducir el esfuerzo en la **verificación formal.**

Property-based testing: Beneficios

Las propiedades se especifican formalmente:

- Fomenta pensar en el **código** de nuevas maneras.
- Aumenta la **comprensión** del sistema probado.
- Representación de **pruebas simple y compacta**.

La verificación no es costosa:

- Las **pruebas** proporcionan **retroalimentación** para depurar la especificación.
- Las verificaciones encuentran **errores** en el código.

Property-based testing

Propiedad de ejemplo: reverse conmuta con append

- **En lógica de predicados:** ∀ xs ys. reverse (xs ++ ys) ≡ reverse ys ++ reverse xs

- **Comparar resultados con resultados esperados**

```
reverse([1, 2, 3]) == [3, 2, 1]
```

- **Comparando con un modelo**

  Para todas las listas `xs` :

```
reverse(xs) == reverse_modelo(xs)
```

- **Comprobando propiedades**

  Para todas las listas `xs` e `ys`:

```
reverse(reverse(xs)) == xs

length(xs) == length(reverse(xs))

reverse(xs ++ ys) == reverse(ys) ++ reverse(xs)
```

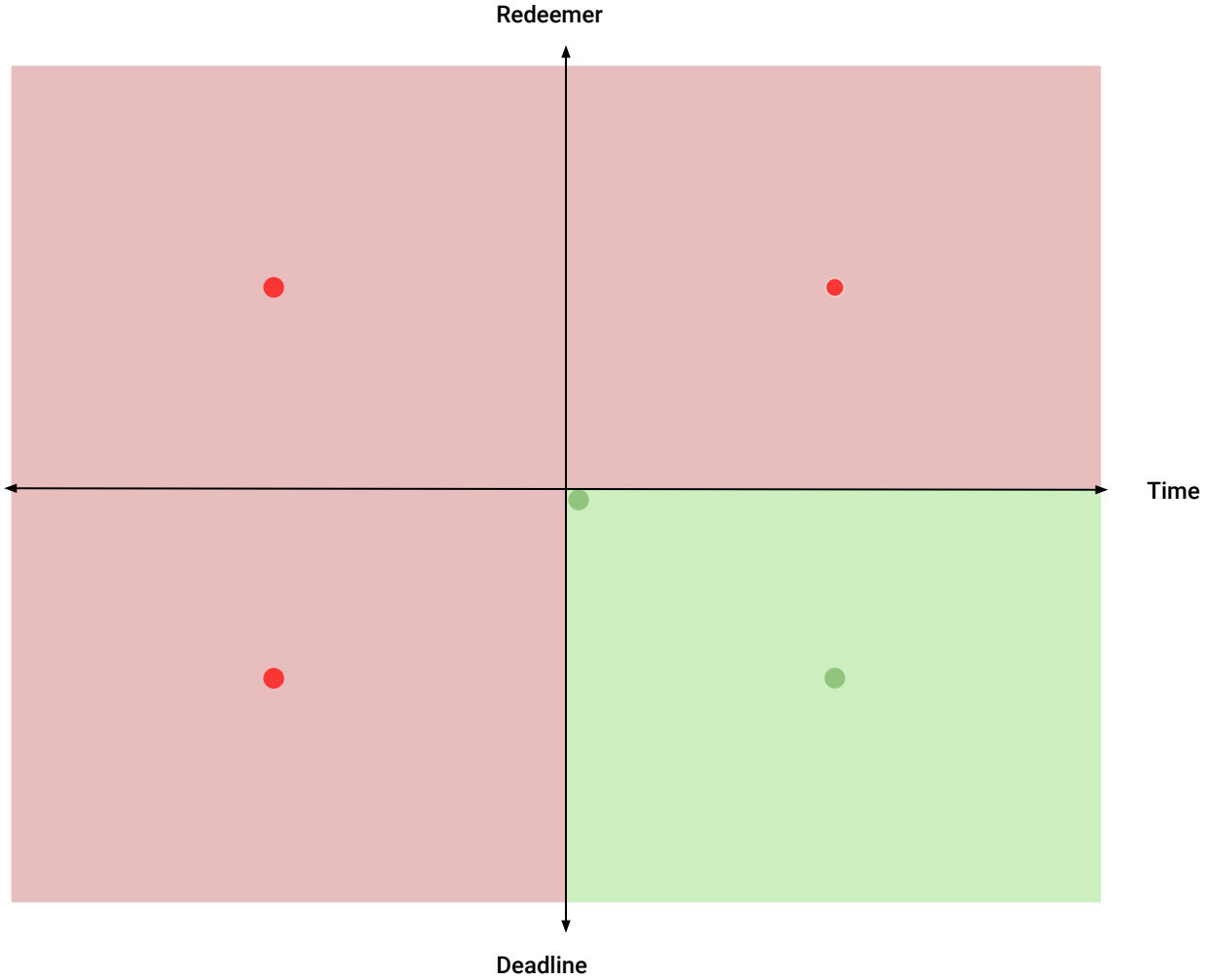
Property-based testing: Relación con verificación formal

Reducción de Costos de Verificación

- Es común probar la corrección de un programa contra una **especificación formal.**
- Someter el código a PBT extensivo antes de la verificación formal **reduce** el número de **defectos** e **inconsistencias** en la especificación.
- Esto, a su vez, reduce el **costo** de verificación.

Proceso de Verificación

- Los ingenieros de pruebas pueden primero **probar una propiedad** y solo intentar demostrarla después de haber ganado confianza razonable en su validez.



Transacción: True

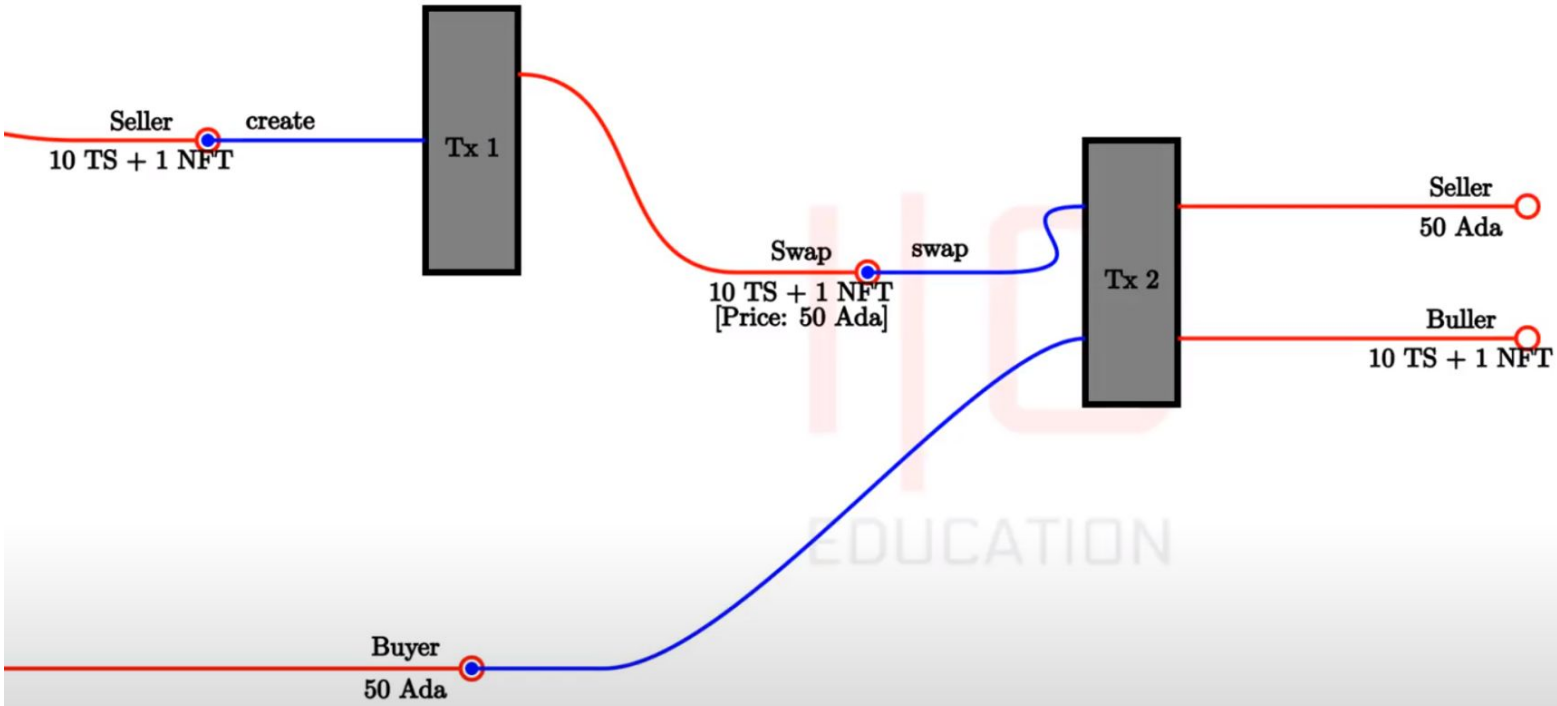
Transacción: False

👣Escribir pruebas para verificar:

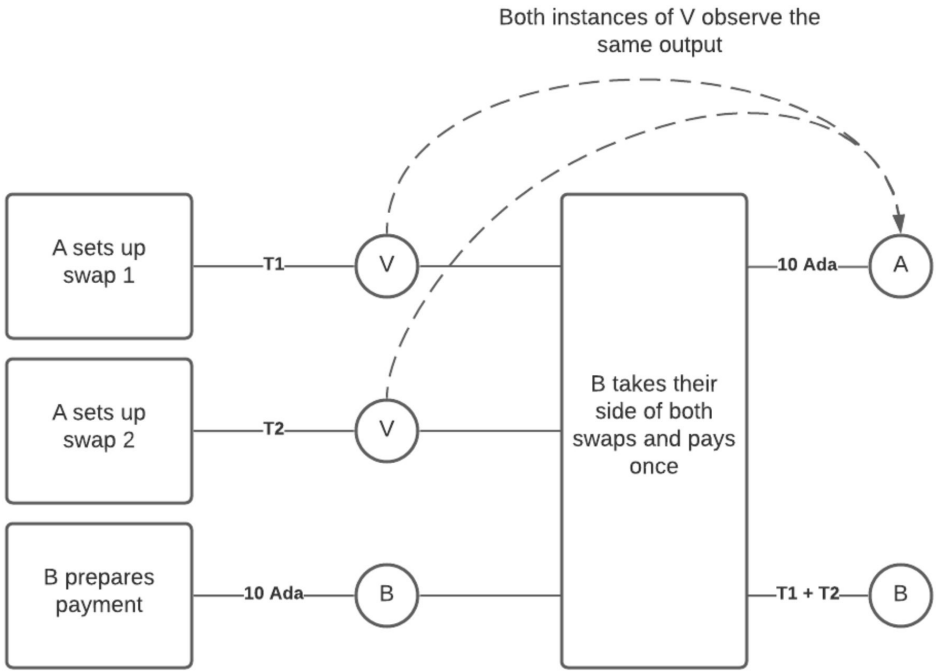
Escribir pruebas para el smart contracts **“negativo después del plazo”**

1. Cualquier cosa antes de la fecha límite falla siempre.
2. Redeemer positivo después de la fecha límite falla siempre.
3. Redeemer negativo después de la fecha límite pasa siempre.

Swap


Seller
10 TS + 1 NFT
create
Tx 1
Swap
10 TS + 1 NFT
[Price: 50 Ada]
swap
Tx 2
Seller
50 Ada
Buller
10 TS + 1 NFT
Buyer
50 Ada

Vulnerabilidad común - Satisfacción doble (Double satisfaction)



✍️ Escribir pruebas para “swap” y corregirlo (si es que hace falta 🤷‍♂️)

¿Preguntas?