Lesson 4
Testing & Coin

Alvin from Typus Lab

什麼是單元測試 Unit test？

測試應用程式中的最小可測單元的運作

類別 Struct

```
public struct Pool has key, store {
    id: UID,
    pool_info: PoolInfo,
    config: PoolConfig,
    incentives: vector<Incentive>,
}
```

函式 fun

```
entry fun new_pool<TOKEN>(
    _manager: &PoolManager,
    unlock_countdown_ts_ms: u64,
    ctx: &mut TxContext
) {
```

智能合約需要做什麼樣的 Unit test？

測試單元的輸入輸出	驗證函式或方法在給定輸入下是否能夠產生正確的輸出
測試邊界條件	測試函式在接近邊界的極端輸入（例如最大值、最小值或空值）下的行為，以及是否有溢位可能等
測試例外狀況處理	確保函式或方法在異常情況下能夠正確處理，正常報錯
測試單元交互與狀態變化	除了輸出外，有些情境需檢查函式內部的狀態變化，以確保其邏輯的正確（例如在多用戶情境）

做 Unit test 是必要的嗎？看看有哪些優缺點

優點	缺點
早期發現問題： 合約部署後 upgrade 合約的彈性十分有限，後期修復漏洞成本高	時間成本高： 編寫和維護單元測試需要時間，延長開發週期
提供運行流程示範： 記錄代碼的預期行為，有助於代碼維護和新開發者理解代碼	維護成本高： 當代碼變更頻繁時，測試需要不斷更新，否則可能會過時或變得無效
提高程式碼穩健度： 強制開發者考慮邊界情況和異常處理	覆蓋範圍有限： 仍然有情境需要實際上線才能測試得到，或是有些情境測試時未考慮周全

Sui CLI Command

1. 將 terminal 位置移至與 package toml 檔案同一層位置
2. 輸入指令:

   sui move test <module>::<function>

   也可以只輸入 module 或 function, 或留白

   sui move test
   sui move test test_pool
   sui move test test_new_pool
   sui move test test_pool::test_new_pool

Sui Move Unit Test 的三個註釋

#[test_only]	放在 module、function、use、address、struct 前，表示這個 function 是作為某個測試情境的程式碼
	
	`#[test_only]`
	`module mover_token::test_pool {`
	`  …`
	`}`
	
	`#[test_only]`
	`public(package) fun test_init(ctx: &mut TxContext) {`
	`  init(ctx);`
	`}`
	
	`#[test_only]`
	`use std::string;`

Sui Move Unit Test 的三個註釋

#[test]	放在 function 前，表示這個 function 是作為某個測試情境的程式碼
	
	`#[test]`
	`public(package) fun test_new_pool() {}`
	
	`#[test(x: 0xabc)]`
	`public(package) fun test_new_pool(x: address) {}`

Sui Move Unit Test 的三個註釋

#[expect_failure]	• 當設計了一個情境是預期應該要出錯時，會放上這個註釋表示 • 一個 function 可以同時具備 test 和 expect_failure

```
#[test, expect_failure]
public(package) fun test_new_pool() {}

#[test]
#[expect_failure]
public(package) fun test_new_pool() {}

#[test]
#[expect_failure(abort_code = pool::E_ZERO_INCENTIVE)]
public(package) fun test_new_pool() {}
```

Module test_scenario

```
use sui::test_scenario::{Scenario, begin, end, ctx, next_tx, take_shared, return_shared,
take_from_sender, return_to_sender, sender};

#[test]
public(package) fun test_new_pool() {
    let mut scenario = scenario();
    ...
    ...
    ...
    end(scenario); // no drop => need to call sc
}
```

今日範例

	主要機制
發幣合約 smvr.move	● 用戶 可以用 100 SUI 鑄造 1 SMVR 並自動進入鎖倉合約鎖倉
鎖倉合約 pool.move	● 用戶 stake 進去即自動鎖倉，呼叫 unsubscribe 之後把指定的顆數進入倒數階段，倒數完成即可呼叫unstake 取出 token ● stake 用戶可以領取 incentive 獎勵，但是 unsubscribe 的部分不能參與獎勵計算

本日作業（作業 4）

- 透過 smvr.move 的 manager function 把 C_MAX_SUPPLY 數量的 SMVR 幣鑄造出來，並且傳給自己
- 交作業時請附上執行的 tx digest
- Hint：manager function public return Coin<SMVR>，需要用 ptb 才能完成任務