91 Unit Testing
單元測試實戰操練營

寫測試的正確姿勢

Agenda

- 寫單元測試的基本原則
  - 重構，拆解包山包海的函式
  - 隔絕依賴，只測試單一功能
  - 重構測試
- 什麼東西要寫測試？什麼不用寫測試？
- 到底要寫哪種測試？

基本原則

- 重構，拆解包山包海的函式
- 隔絕依賴，只測試單一功能
- 重構測試

原本的樣子 (1/2) - 程式碼不適合寫測試該怎麼辦？

這裡有一個 Holiday 的 class，其中包含一個方法 checkValentinesDay，checkValentinesDay 會確定今天是否為情人節 (2/14)

因此

- 若是情人節，就回傳字串「情人節快樂」
- 若不是情人節，則回傳「今天不是情人節」

```
1.  export default class Holiday {
2.   checkValentinesDay() {
3.     const today = new Date();
4.     const month = today.getMonth() + 1;
5.     const day = today.getDate();
6.     return month === 2 && day === 14 ? '情人節快樂' : '今天不是情人節';
7.   }
8.  }
```

原本的樣子 (2/2) - 程式碼不適合寫測試該怎麼辦？

- 今天的確不是情人節 -> 測試失敗
  - 無法自行指定要測試的時間
- 預期要能測試成功和失敗的兩種狀況

```
1.  describe('今天是情人節嗎？', () => {
2.    it('今天是情人節', () => {
3.      const holiday = new Holiday();
4.      const res = holiday.checkValentinesDay();
5.      expect(res).toBe('情人節快樂');
6.    });
7.  });
```

Step 1: 重構 - 拆解包山包海的函式 (1/2)

隔絕依賴，只測試單一功能

```
1.  export default class Holiday {
2.
3.
4.
5.
6.    checkValentinesDay() {
7.      const today = new Date();
8.      const month = today.getMonth() + 1;
9.      const day = today.getDate();
10.     return month === 2 && day === 14 ? '情人節快樂' : '今天不是情人節';
11.   }
12. }
```

← 這是依賴，所以不是本次測試目標

Step 1: 重構 - 拆解包山包海的函式 (2/2)

隔絕依賴，只測試單一功能

```
1.  export default class Holiday {
2.    getToday() {
3.      return new Date();
4.    }
5.
6.    checkValentinesDay() {
7.      const today = this.getToday();
8.      const month = today.getMonth() + 1;
9.      const day = today.getDate();
10.     return month === 2 && day === 14 ? '情人節快樂' : '今天不是情人節';
11.   }
12. }
```

抽出來變成函式

Step 2：隔絕依賴，只測試單一功能 (1/2)

- Mock 依賴的函式，就能專心測特定的函式
  - 本次測試的重點是 checkValentinesDay，確認今天是否為情人節，其他 **不相關的**（例如：getToday）**都要被 mock 起來**
  - Mock 的函式能依照測試需求 **回傳指定的值**，來測試預期的成功和失敗的兩種狀況

Step 2：隔絕依賴，只測試單一功能 (2/2)

```
1.  describe('今天是情人節嗎？', function () {
2.    it('今天是情人節 2/14', () => {
3.      const holiday = new Holiday();
4.      holiday.getToday = jest.fn();
5.      holiday.getToday.mockReturnValue(new Date('2021-02-14'));
6.
7.      const res = holiday.checkValentinesDay();
8.      expect(res).toBe('情人節快樂');
9.    });
10.
11.   it('今天不是情人節 2/12', () => {
12.     const holiday = new Holiday();
13.     holiday.getToday = jest.fn();
14.     holiday.getToday.mockReturnValue(new Date('2021-02-12'));
15.
16.     const res = holiday.checkValentinesDay();
17.     expect(res).toBe('今天不是情人節');
18.   });
19. });
```

```
PASS test/holiday.test.js
  今天是情人節嗎？
    ✓ 今天是情人節 2/14 (4 ms)
    ✓ 今天不是情人節 2/12

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.025 s
```

Mock 的函式能依照測試需求回傳指定的值

Step 3: 重構測試

- 提出共用的判斷
  - 簡化邏輯、重用
- 隱藏無關的商業邏輯
  - 人都是健忘的，簡單易懂好維護

提出共用的判斷 (1/5)

經過一些需求變更，情人節必須包含白色情人節 (也就是 3/14)...

因此，情人節包含兩種狀況：2/14 與 3/14

```
export default class Holiday {
  ...

  checkValentinesDay() {
    const today = this.getToday();
    const month = today.getMonth() + 1;
    const day = today.getDate();
    return (month === 2 || month === 3) && day === 14 ? '情人節快樂' : '今天不是情人節';
  }
}
```

提出共用的判斷 (2/5)

在測試上就需要包含多種情況...以下會測試 4 種狀況

- 2/14：是情人節
- 3/14：是情人節
- 4/14：不是情人節：因為判斷了月份和日期，測測看同月但不同日的狀況
- 2/12：不是情人節

提出共用的判斷 (3/5)

```
1.  describe('今天是情人節嗎？', function () {
2.    it('今天是情人節 2/14', () => {
3.      ...
4.      const res = holiday.checkValentinesDay();
5.      expect(res).toBe('情人節快樂');
6.    });
7.
8.    it('今天是情人節 3/14', () => {
9.      ...
10.   });
11.
12.   it('今天不是情人節 4/14', () => {
13.     ..
14.   });
15.
16.   it('今天不是情人節 2/12', () => {
17.     ...
18.   });
19. });
```

提出共用的判斷 (4/5)

抽出共用的判斷 shouldBeValentinesDay 與 shouldNotBeValentinesDay,

目的?

- 清楚明瞭到底是判斷什麼
- 精簡程式碼
- 重用

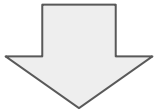
提出共用的判斷 (5/5)

抽出共用的判斷 shouldBeValentinesDay

- 清楚明瞭到底是判斷什麼
- 精簡程式碼
- 重用

```
it('今天是情人節 2/14', () => {
  const holiday = new Holiday();
  holiday.getToday = jest.fn();
  holiday.getToday.mockReturnValue(new Date('2021-02-14'));
  const res = holiday.checkValentinesDay();
  expect(res).toBe('情人節快樂');
});
```

```
it('今天是情人節 2/14', () => {
  const holiday = new Holiday();
  holiday.getToday = jest.fn();
  holiday.getToday.mockReturnValue(new Date('2021-02-14'));
  shouldBeValentinesDay(holiday.checkValentinesDay());
});
```



```
今天是情人節嗎？
  ✓ 今天是情人節 2/14 (3 ms)
  ✓ 今天是情人節 3/14 (1 ms)
  ✓ 今天不是情人節 4/14
  ✓ 今天不是情人節 2/12 (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.555 s
```

隱藏無關的商業邏輯 (1/2)

測試要用來描述情境，無關的商業邏輯都要隱藏

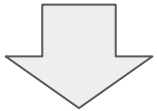
- 例如：情人節的年份
  - 年份不重要，月/日 才重要

隱藏無關的商業邏輯 (2/2)

隱藏無關的商業邏輯

- 測試要用來描述情境，無關的商業邏輯都要隱藏

```
it('今天是情人節 2/14', () => {
  const holiday = new Holiday();
  holiday.getToday = jest.fn();
  holiday.getToday.mockReturnValue(new Date('2021-02-14'));
  shouldBeValentinesDay(holiday.checkValentinesDay());
});
beforeEach(() => {
  holiday = new Holiday();
  mockGetToday = jest.fn();
  holiday.getToday = mockGetToday;
});
it('今天是情人節 2/14', () => {
  givenToday('2021-02-14');
  shouldBeValentinesDay(holiday.checkValentinesDay());
});
```

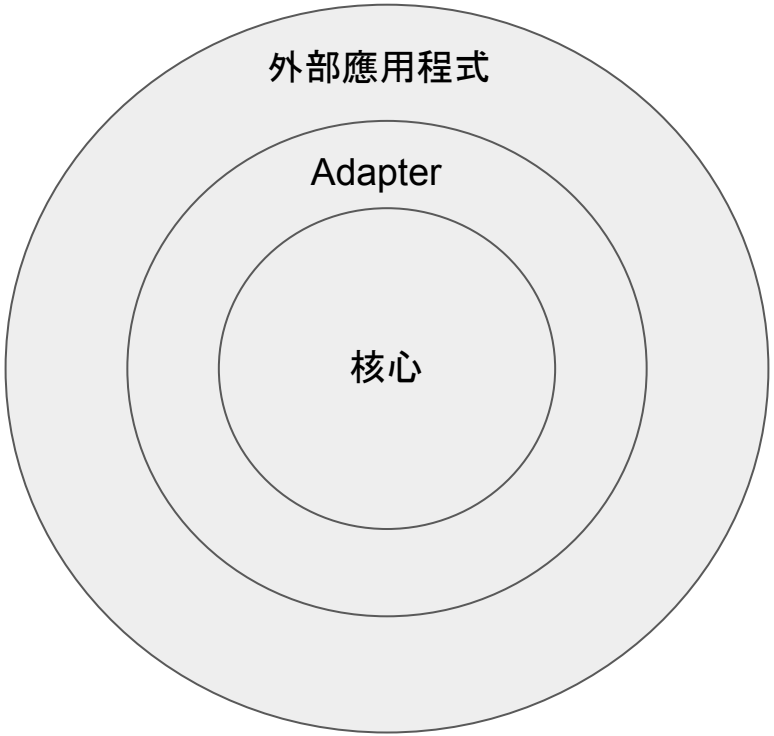


利用 givenToday 包裝要設定的日期，並放到環境假設的前置作業 beforeEach 中。

什麼東西要寫測試？什麼不用寫測試？

- 一定要寫測試的是哪些東西呢？選擇投資回報率 (Return On Investment, ROI) 較高的
  - 錢
  - 安全性
  - 最常被修改但很容易改錯
  - 過去有 bug 所補上的測試
  - 主流程（主流程先寫、最後才寫防呆）
  - 商譽
- code coverage 怎麼用才是對的？
  - code coverage 不是愈高愈好
  - 從沒測到的比例，反推沒有測試 (uncovered) 的情境 -> 評估這些情境是否 值得寫測試。
- 相對趨勢 > 絕對數字
  - 每次進 code 都要增加測試程式

洋蔥式架構



一個完整的系統分為以下部份

- 核心：自家產品的主要服務
- Adapter：串接自家產品與第三方服務的介面
- 外部應用程式：第三方服務

到底要寫哪種測試？

洋蔥式架構或稱六角架構：核心 (domain / core business) -> adapter -> 外部應用程式

- 核心：寫單元測試
- adapter -> 外部應用程式：寫整合測試

寫單元測試 + 整合測試，就有基礎建設、可以保持好的品質了

感想

上課前覺得自己應該還滿會寫測試的 (不知哪來的自信？), 結果第一個小範例就讓我見識到自己的無知和渺小 XD 像是

- 怎麼拆解原始碼並隔離依賴來做測試的觀念就花了一些時間練習
- 還有過去「重構」只會發生在實作功能的原始碼上, 從沒想過測試程式也要寫得模組化, 都只能算是「有寫」而已。

釐清觀念、吸收新知 (๑•̀ㅁ•́)و✧