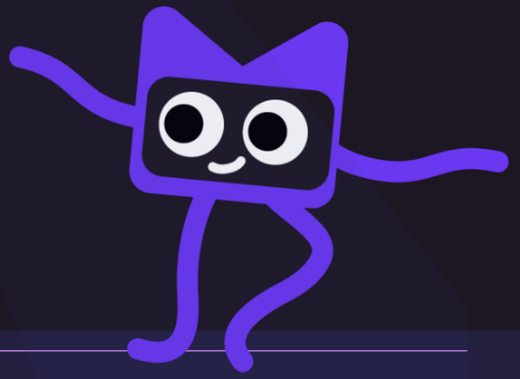
從紅燈到綠燈

范聖佑 / 高見龍



為你自己學 PYTHON

給新手的 Python 程式學習書

> **學習不需要為公司、為長官、同事、老師或是爸媽，只要為你自己。**

這是一本給新手的 Python 程式語言學習書！

本書以 Python 3 做為主要教學語言，前半部「**基礎應用篇**」，內容涵蓋環境安裝、Python 程式語言語法，包括各種常用資料型態、邏輯判斷、迴圈及流程控制、函數、物件導向程式設計等，並透過網站爬蟲程式抓取並分析資料。後半部「**網站開發篇**」則介紹如何使用 Flask、FastAPI 及 Django 等網站開發框架來開發自己的網站。

電子書

章節內容

線上課程

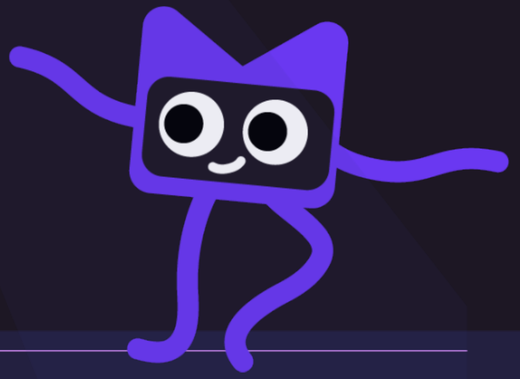
基礎應用篇

網站開發篇（錄製中）

實體書、電子書準備中

從紅燈到綠燈

范聖佑 / 高見龍



關於 TDD

TDD = Test-Driven Development

TDD = Test-Driven Development

TDD 是一種開發方法

TDD != Debugger


When i try to fix a bug

為什麼不寫測試？

為什麼不寫測試？

- 「光寫主程式都沒時間了，哪裡還有時間寫測試」
- 「跑測試太慢了」
- 「測試很脆弱耶，不小心改一下就爛掉了」
- 「不知道怎麼寫」

Kotest is a flexible and elegant multi-platform test framework for Kotlin with extensive assertions and integrated property testing

Get Started

Star 4,412

KOTLINLANG KOTEST | RELEASE V5.9.1 | LATEST SNAPSHOT V6.0.0-SNAPSHOT | LICENSE APACHE2.0 | STACKOVERFLOW KOTEST

Test Framework

The Kotest test framework enables test to be laid out in a fluid way and execute them on JVM, Javascript, or native platforms.

With built in coroutine support at every level, the ability to use functions as test lifecycle callbacks, extensive extension points, advanced conditional evaluation, powerful data driven testing, and more.

Read more

Assertions Library

The Kotest assertions library is a Kotlin-first multi-platform assertions library with over 300 rich assertions.

It comes equipped with collection inspectors, non-determistic test helpers, soft assertions, modules for arrow, json, kotlinx-datetime and much more.

Read more

Property Testing

The Kotest property testing module is an advanced multi-platform property test library with over 50 built in generators.

It supports failure shrinking, the ability to easily create and compose new generators; both exhaustive and arbitrary checks, repeatable random seeds, coverage metrics, and more.

Read more

安裝 & 執行

3A = Arrange, Act, Assert

情境：發送 Email 💌

Live Demo

玩真的還是玩假的？

相依性 Dependency

相依性

- 呼叫其它類別的函數
- 金流刷卡
- 網路服務
- 還沒寫好的 API

Mock vs Stub

Mock vs Stub

- Stub = 設定固定的回傳值，用來填充實際的依賴，使測試更可控。
- Mock = 模擬物件行為，用來驗證方法是否被呼叫或是以及呼叫的次數。
- 相同與不同：
    - 都是用來替代和模擬外部依賴的手法，讓測試在不依賴真實環境下進行。
    - Mock 會檢查有沒有做對的事（例如，是否正確呼叫方法），而 Stub 只是給你固定的答案，不管你怎麼用。

```
fn processOrder(order):
  if paymentService.charge(order):
    notifyService.sendMail(order.customerEmail, "訂單已完成")
```

Stub

```
class PaymentServiceStub:
    fn charge(order):
        return true

class NotificationServiceStub:
    fn sendMail(email, message):
        // 嘿嘿！不做事
```

Stub

```
class TestProcessOrderWithStub():
    paymentService = PaymentServiceStub()
    notificationService = new NotificationServiceStub()

    order = Order()
    order.customerEmail = "eddie@5xcampus.com"

    fn processOrder(order):
      if paymentService.charge(order):
        notifyService.sendMail(order.customerEmail, "訂單已完成")


    processOrder(order)
```

Mock

```
class PaymentServiceMock:
    let wasCalled = false

    fn charge(order):
        this.wasCalled = true
        return true
```

```
class NotificationServiceMock:
    let wasCalled = false
    let receivedEmail = null
    let receivedMessage = null

    fn sendMail(email, message):
        this.wasCalled = true
        this.receivedEmail = email
        this.receivedMessage = message
```

Mock

```
class TestProcessOrderWithMock():
    paymentService = PaymentServiceMock()
    notificationService = NotificationServiceMock()

    order = Order()
    order.customerEmail = "eddie@5xcampus.com"

    processOrder(order)

    assert(paymentService.wasCalled == true)
    assert(notificationService.wasCalled == true)
    assert(notificationService.receivedEmail == "eddie@5xcampus.com")
    assert(notificationService.receivedMessage == "訂單已完成")
```

Live Demo

常見問題

常見問題

- 「要怎麼測試還不存在的程式碼？」
- 「我怎麼知道哪些東西要測？」
- 「TDD 是寫程式來測試程式，那麼這些 TDD 的程式要誰來測？」

結論：為什麼寫測試？

寫測試是為了...

- 測試本身就是規格（Spec）
- 寫出更有信心的程式碼
- 可以做出比較好的設計
- 將來有重構（Refactor）的可能性

從紅燈到綠燈

范聖佑 / 高見龍