Chapter 2：匿名函數 詳述

一般來說

我們會給函數取名字

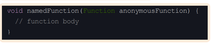
並不是所有函數都需要名字

函數作為值：

匿名函數：講解匿名函數的定義和特性，即沒有函數名且返回類型由函數體推斷。

將函數賦值給變量：演示如何將匿名函數賦值給變量，就像將值賦給變量一樣。

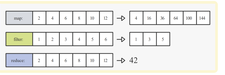
將函數傳遞給函數：說明如何將函數作為參數傳遞給其他函數，體現 Dart 函數作為一等公民的特性。



從函數返回函數：展示如何從函數中返回匿名函數。

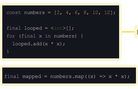
高階函數與集合操作：

高階函數：介紹高階函數的概念，即返回函數或接受函數作為參數的函數。



forEach 循環：如何使用 forEach 方法對集合元素進行操作，並與 for-in 循環進行比較。

集合映射：map 方法的用法，如何將一個集合轉換為另一個集合，例如對每個元素進行平方運算。



集合過濾：where 方法的用法，如何根據條件篩選集合元素，例如過濾出所有奇數。

where可以用在List或Set，但不能用在Map，除非你知道Map中的Key

集合合並：reduce 和 fold 方法的用法，如何將集合元素合並為一個值，例如計算所有元素的總和。

集合排序：回顧 sort 方法的用法，如何使用匿名函數自定義排序規則，例如按字符串長度排序。

高階函數組合：如何將多個高階函數鏈接在一起，實現聲明式編程，例如過濾出長度大於 5 的字符串並將其轉換為大寫。

語法簡化：

1. `myFunction()`：這個括號 `()` 表示立即執行 `myFunction` 這式的程式碼，它會運算這式內的語句並獲得返回值(如果有的話)。

2. `myFunction`：這種沒有括號的形式不會執行這式的程式碼，而是取得對該這式的參考。這個參考可以被賦值給一個變數，以便之後再呼叫該函式。

簡單來說：

- `myFunction()` 會立即執行這式
- `myFunction` 只是獲取對該這式的參考，而不執行其中的程式碼

舉個例子：

```dart
void printHello() {
  print('Hello');
}

void main() {
  printHello(); // 這會立即輸出 'Hello'

  var printHelloRef = printHello;
  // printHelloRef 現在持有對 printHello 這式的參考

  printHelloRef(); // 呼叫 printHelloRef 會輸出 'Hello'
}
```

在這個例子中，`printHello()` 會立即執行這式並印出 "Hello"，而 `printHello` 則只是取得對該這式的參考，被賦值給 `printHelloRef` 變數，我們可以透過呼叫 `printHelloRef()` 來間接執行 `printHello` 這式。

取得這式參考的主要用途是將這式作為參數傳遞給其他這式，或將其賦值給變數以便之後使用。這在這數式程式設計和高階這式中很常見。

Tear-off：Tear-off 的概念，即省略函數參數的語法，例如將 handleButtonClick 方法作為回調函數傳遞。

類型別名：介紹 typedef 關鍵字的用法，以及如何使用類型別名簡化屢複的函數類型簽名。

回調函數：

回調函數：回調函數的概念，即用於處理事件的匿名函數，例如按鈕點擊事件或音頻播放結束事件。

只有在其他程式碼執行完成後才會執行

空回調：不帶參數也不返回值的回調函數的用法，例如 TextButton 的 onPressed 回調。

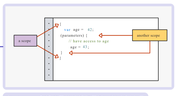
值設置回調：需要參數的回調函數的用法，例如音頻進度條的 onTouch 回調。

值獲取回調：返回值的回調函數的用法，例如 AnotherWidget 的 timeStamp 回調。

閉包和作用域：

作用域：Dart 中作用域的定義，以及限制如何訪問作用域中的變量。

閉包：閉包的概念，即匿名函數可以訪問其周圍作用域中的變量和函數。



在 closure 內部使用的外部作用域變數,實際上是對該變數的參考,而非該變數的值大身。這意味著,如果在外部作用域中更新了該變數的值,closure 內部存取該變數時會得到最新的值。

舉例來說:

```dart
void main() {
  String name = 'Alice';

  void printName() {
    print(name); // 在這裡存取 name 變數
  }

  printName(); // 輸出: Alice

  name = 'Bob'; // 更新外部作用域中的 name 變數
  printName(); // 輸出: Bob
}
```

在這個範例中，`printName` 是一個 closure,它存取了外部作用域中的 `name` 變數。當我們第一次呼叫 `printName()` 時,它會印出 `Alice`。但是在我們將 `name` 變數更新為 `Bob` 之後,再次呼叫 `printName()` 時,它會印出 `Bob`。這是因為 `printName` 實際上持有對 `name` 變數的參考,而非變數的值本身。

因此,如果在外部作用域中有其他這式更新了 closure 內部存取的變數,這並不會產生問題,closure 會自動獲得最新的變數值。

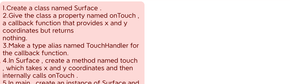
但是,需要注意的是,如果在閉包內部嘗試重新賦值給該變數,它將創建一個新的局部變數,而不會修改外部作用域中的變數。例如:

```dart
void main() {
  String name = 'Alice';

  void printName() {
    String name = 'Bob'; // 這是一個新的局部變數
    print(name); // 輸出: Bob
  }

  printName(); // 輸出: Bob
  print(name); // 輸出: Alice
}
```

在這種情況下，`printName` 內部的 `name` 變數是一個新的局部變數,與外部作用域中的 `name` 變數無關。

總的來說,closure 內部存取的外部作用域變數會自動反映外部作用域中變數的最新值,這是一個很有用的特性,可以讓我們編寫更加靈活和動態的程式碼。但是,如果在 closure 內部嘗試重新賦值給該變數,則會創建一個新的局部變數,而不會修改外部作用域中的變數。

練習



答案出來

1. Create a class named Surface .
2.Give the class a property named onTouch , a callback function that provides x and y coordinates but returns
nothing.
3.Make a type alias named TouchHandler for the callback function.
4.In Surface , create a method named touch , which takes x and y coordinates and then internally calls onTouch .
5.In main , create an instance of Surface and pass in an anonymous function that prints the x and y coordinates.
6.Still in main , call touch where x is 202.3 and y is 154.0 .