

Chapter 2：匿名函數詳述

一般來說

我們會給函數取名字

```
function name() {  
  // String complement number() {  
    print('Number is a very nice number');  
  }  
}
```

並不是所有函數都需要名字

```
String complement(int number) {  
  return 'Number is a very nice number';  
}
```

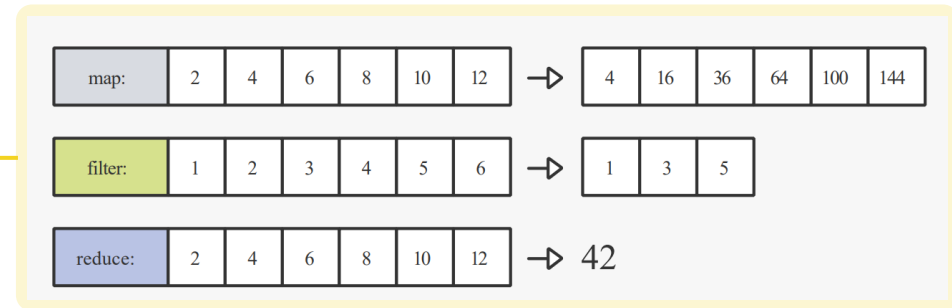
函數作為值：

- 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。

```
function multiply = (int a, int b) {  
  return a * b;  
};  
  
void namedFunction(function anonymousFunction) {  
  // function body  
}  
  
function namedFunction() {  
  return () => print('hello');  
}
```

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。
- 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。



```
final flowersColor = ['roses', 'lilies', 'tulips'];  
flowersColor.forEach((flower, color) {  
  print('$flower are $color');  
});  
  
print('I love $2764 Dart!');  
print('Want to do you?');
```

```
const numbers = [2, 4, 6, 8, 10, 12];  
final looped = <n>[];  
for (final x in numbers) {  
  looped.add(x * x);  
}
```

looped是list

final mapped = numbers.map((x) => x * x);

mapped是一個iterable 如果想轉為list可再添加mapped.toList()

```
void main() {  
  final mylist = [1, 2, 3, 4, 5, 6];  
  final odds = mylist.where((element) => element.isOdd);  
  print(odds);  
}
```

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

```
void main() {  
  const evens = [2, 4, 6, 8, 10, 12];  
  final total = evens.reduce((sum, element) => sum + element);  
  print(total);  
}
```

```
void main() {  
  const evens = [2, 4, 6, 8, 10, 12];  
  final total = evens.fold<int>(0, (sum, element) => sum + element);  
  print(total);  
}
```

```
void main() {  
  final desserts = ['cookies', 'pie', 'donuts', 'brownies'];  
  desserts.sort((d1, d2) => d1.length.compareTo(d2.length));  
  print(desserts); // [pie, cookies, donuts, brownies]  
}
```

```
void main() {  
  const desserts = ['cake', 'pie', 'donuts', 'brownies'];  
  final bigAllDesserts = desserts  
    .where((dessert) => dessert.length > 5)  
    .map((dessert) => dessert.toUpperCase())  
    .toList();  
  print(bigAllDesserts); // [CAKE, BROWNIES]  
}
```

```
const desserts = ['cake', 'pie', 'donuts', 'brownies'];  
final bigAllDesserts = <n>[];  
for (final item in desserts) {  
  if (item.length > 5) {  
    final upperCase = item.toUpperCase();  
    bigAllDesserts.add(upperCase);  
  }  
}
```

語法簡化：

- 1. 'myFunction()'：這個括號 '()' 表示立即執行 'myFunction' 函式的程式碼，它會運算函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函式的程式碼，而是取得對該函式的參考。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()' 會立即執行函式
 - 'myFunction' 只是獲取對該函式的參考，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 函式的參考

 printHelloRef(); // 呼叫 printHelloRef 會輸出 'Hello'
}
```
- 在這個例子中，'printHello()' 會立即執行函式並印出 'Hello'，而 'printHello' 則只是取得對該函式的參考，被賦值給 'printHelloRef' 變數。我們可以透過呼叫 'printHelloRef()' 來間接執行 'printHello' 函式。
- 取得函式參考的主要用途是將函式作為參數傳遞給其他函式，或將其賦值給變數以便之後使用。這在函數式程式設計和高階函數中很常見。
- Tear-off：Tear-off 的概念，即省略函數參數的語法，例如將 handleClick 方法作為回調函數傳遞。
- 類型別名：介紹 typedef 關鍵字的使用法，以及如何使用類型別名簡化複雜的函數類型簽名。

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。
- 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。

```
TextButton(
 child: Text('Click me!'),
 onPressed: () {
 print('Button clicked!');
 },
);
```

只有在其他程式碼執行完成後才會執行

```
void main() {
 button.onPressed();
}

class Button {
 final String title;
 final Function onPressed;

 Button(required this.title, required this.onPressed);
}

final button = Button(
 title: 'Click me!',
 onPressed: () {
 print('Button clicked!');
 },
);
```

```
class MyWidget {
 MyWidget({required this.onTouch});
 final void Function(double xPosition) onTouch;

 final myWidget = MyWidget(
 onTouch: (x) => print(x),
);

 void main() {
 myWidget.onTouch(3.14);
 }
}
```

```
class AnotherWidget {
 AnotherWidget({this.timeStamp});
 final String Function()? timeStamp;

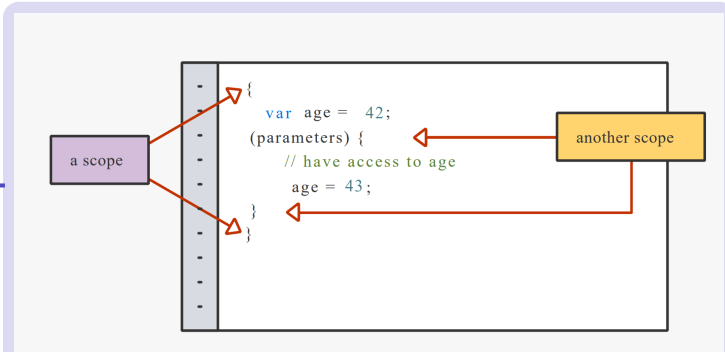
 final myWidget = AnotherWidget(timeStamp: () => DateTime.now().toIso8601String());

 final timeStamp = myWidget.timeStamp?.call();

 void main() {
 print(timeStamp);
 }
}
```

閉包和作用域：

- 作用域：Dart 中作用域的定義，以及閉包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。
- 示例：提供一個計數函數的例子，演示閉包如何捕獲作用域中的變量。



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，而非該變數的值本身。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函式更新了 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

```
void main() {
 final scores = [89, 77, 46, 93, 82, 67, 32, 88];
 // 2 Use sort to order the grades from highest to lowest.
 // 2 Use where to find all the B grades, that is, all the scores between 80 and 90
}
```

- Create a class named Surface.
- Give the class a property named onTouch, a callback function that provides x and y coordinates but returns nothing.
- Make a type alias named TouchHandler for the callback function.
- In Surface, create a method named touch, which takes x and y coordinates and then internally calls onTouch.
- In main, create an instance of Surface and pass in an anonymous function that prints the x and y coordinates.
- Still in main, call touch where x is 202.3 and y is 134.0.

(82, 88, 89)

答案出來