流程語法與函式

邏輯分析與程式設計

國立雲林科技大學
王照明老師

助教：林育慶

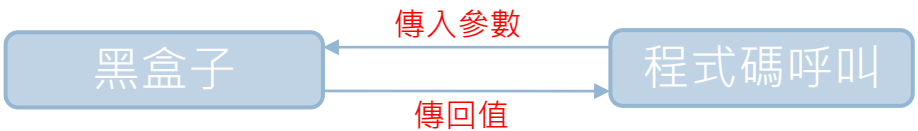
函數的基礎

基礎說明

- JavaScript函式是將程式中一些共用程式碼獨立成程式區塊，能夠重複呼叫來傳入參數和傳回執行結果，事實上，JavaScript資料型態都是一種物件，函式也是。
- 「函式」（Functions）是將程式中常用的共同程式碼獨立成程式區塊，以便能夠重複呼叫這些函式的程式碼。一般來說，函式都有傳回值，如果函式沒有傳回值，稱為「程序」（Procedures）。

函數是一個黑盒子

- 在JavaScript程式中執行函數稱為「函數呼叫」（Functions Call），我們並不需要了解函數內部實際的程式碼，事實上，也不想知道其細節，函數如同是一個「黑盒子」（Black Box），只要告訴我們如何使用這個黑盒子的「使用介面」（Interface）即可，如下圖所示：



基礎說明

- 「函式」指的是將一或多段程式指令包裝起來，可以重複使用，也方便維護。宣告函式的方法有好幾種，但不管是什麼方式，通常一個函式會包含三個部分：
  1. 函式的名稱。
  2. 在括號 ( ) 中的部分，稱為「參數 (arguments) 」，參數與參數之間會用逗號隔開。
  3. 在大括號 { } 內的部分，內含需要重複執行的內容，是函式功能的主要區塊。

```
function 名稱([參數]) {
  // 做某事
}
```

```
function abc(number) {
  return number * number;
}
```

建立涵式

JavaScript自訂函式

- 函式使用function關鍵字宣告，名稱為abc，在括號定義傳入函式的參數，此函式並沒有參數，在「{」和「}」程式區塊內是函式的程式碼。因為函式沒有傳回值，所以呼叫函式只需使用函式名稱。

```
function abc() {
 document.write("請輸入100以內的數字");
}
```

- JavaScript程式是如何執行函式，程式是呼叫abc()函式，此時程式碼執行順序就跳到abc()函式執行裡面內容。

```
<script>
abc();
</script>
```

```
function abc() {
document.write("請輸入100以內的數字");
}
```

變數種類

- JavaScript擁有兩種變數範圍
  - 區域變數（Local Variables）：在函數內宣告的變數，變數只能在函數程式區塊之中使用，函數之外的程式碼並無法存取此變數
  - 全域變數（Global Variables）：如果變數是在函數外宣告，整個JavaScript程式檔的函數和程式碼都可以存取此變數。

```
<script>

   var A=10;    //Global Variables

   function Test(){
      var B=200;    //Local Variables
   }

</script>
```

Lab01

店員：雞排1份60元，牛奶一瓶30元

客人：我要5 個雞排，和1 杯牛奶！

店員：計算答案，一共是*330*元。

Lab01

```
<body>
  <p>店員：雞排1份60元，牛奶一瓶30元</p>
  <p>客人：我要<input type="text" id='chiNumId'>個雞排，和<input type="text" id='
  milkNumId'>杯牛奶！</p>
  <p>店員：<button id='countId'>計算答案</button>，一共是<em id='totalId'></em>元。</p>
  <script>
    document.getElementById('countId').onclick = function(){
//用變數設定雞排跟牛奶的售價
    var chi = 60;
    var milk = 30;
//選取輸入欄位的 DOM 並宣告變數名，用 value 帶出輸入欄的值。
    var chiNum = parseInt(document.getElementById('chiNumId').value);
    var milkNum = parseInt(document.getElementById('milkNumId').value);
//再用一個變數儲存金額加總
    var total = chiNum*chi + milkNum*milk;
//把總金額顯示至網頁上
    document.getElementById('totalId').textContent = total;

  </script>
</body>
```

有參數的函式

- JavaScript函式可以傳入1至多個參數，函式如果擁有傳入參數，在呼叫函式時，只需傳入不同的參數值（稱為引數）就可以產生不同的執行結果。

```
function write(strMsg, intnum) {
   for(var i=1; i<=intnum; i++) {
      document.write(strMsg + "<br/>");
   }
}
```

- write ()函式擁有2個參數。因為函式擁有參數，在呼叫函式時需要傳入引數（Arguments）

```
write("國立雲林科技大學", 5);
```

國立雲林科技大學
國立雲林科技大學
國立雲林科技大學
國立雲林科技大學
國立雲林科技大學

函式的傳回值

- JavaScript函式可以傳回函式的執行結果，即函式的傳回值，此時的函式可以視為是一個黑盒子，只需傳入不同的參數值，就可以產生不同執行結果的傳回值，例如：建立計算數學公式的JavaScript函式。
- 對於華氏（Fahrenheit）和攝氏（Celsius）的溫度轉換來說，我們可以使用數學公式來進行計算。首先是攝氏轉華氏的公式：
  f = (9.0 * c) / 5.0 + 32.0;
- 華氏轉攝氏的公式：
  c = (5.0 / 9.0) * (f - 32);

函式的傳回值

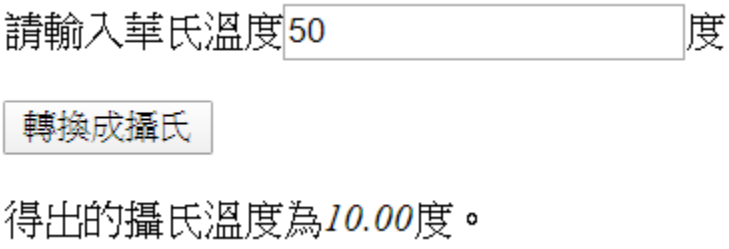
- 我們可以建立JavaScript函數替我們解數學問題，像是溫度轉換函數convert2F()計算攝氏轉成華氏的溫度，使用return關鍵字傳回函數的執行結果。若是輸入攝氏20度，則會得出華氏68度。

```
var f;
function convert2F(c) {

   f = (9.0 * c) / 5.0 + 32.0;
   return f;
}
f = convert2F(20);
document.write(f);
```

68

Lab02



Lab02

```
<body>
<p id="demo"></p>
<p>請輸入華氏溫度<input type="text" id='fahrenheitId'>度</p>
<p><button onclick="toCelsius(f);">轉換成攝氏</button></p>
<p>得出的攝氏溫度為<em id='totalId'></em>度。</p>
  <script>
var f;
function toCelsius(f) {
        var f = parseInt(document.getElementById('fahrenheitId').value);
        var c = (5/9) * (f-32);
        document.getElementById('totalId').textContent = c.toFixed(2);
        }
  </script>
</body>
```

Lab02

```
<body>
   <p id="demo"></p>
   <p>請輸入華氏溫度<input type="text" id='fahrenheitId'>度</p>
   <p><button onclick="toCelsius(f)">轉換成攝氏</button></p>
   <p>得出的攝氏溫度為<em id='totalId'></em>度。</p>
     <script>
   var f;
   function toCelsius(f) {
        var f = parseInt(document.getElementById('fahrenheitId').value);
        var c = (5/9) * (f-32);
        document.getElementById('totalId').textContent = c.toFixed(2);
        }
    </script>
   </body>
```

函數運算式

- 函式運算式是透過變數，將一個函式透過 = 指定給某個變數。函式實際上它仍屬於 Object 的類型，是一種可以被呼叫 (be invoked) 的特殊物件 (值)，所以可以透過變數存入。若是沒有為涵式取名的話，通常會稱它為「匿名函式」，像這類沒有名字的函式在 JavaScript 仍是合法的，。

```
var abc = function (number) {
  return number * number;
};
```

```
var x = 1;
    var f = function(y) {
        var x = 100;
        return x + y;
};
document.write(f(50)+"<br/>");
document.write(x);
```

150
1

巢狀函數

- 函數的傳回值可以是另一個巢狀函數的執行結果
- init()巢狀函數之中擁有displayName ()函數，而函數的傳回值就是此函數的執行結果。
- displayName() 自己並沒有局部變數，不過它可以訪問外面函式的變數、因而能取用在父函式宣告的變數 name。

```
function init() {
  var name = "Mozilla"; // name 是個由 init 建立的局部變數
  function displayName() { // displayName() 是內部函式，一個閉包
    alert(name); // 使用了父函式宣告的變數
  }
  return displayName();
}
init();
```

遞迴函數

- 遞迴是由上而下分析方法的一種特殊的情況，使用遞迴觀念建立的函數稱為遞迴函數，其基本定義如下所示：
  一個問題的內涵是由本身所定義的話，稱之為遞迴。
- 因為遞迴問題在分析時，其子問題本身和原來問題擁有相同的特性，只是範圍改變，範圍逐漸縮小到終止條件，所以可以歸納出遞迴函數的兩個特性：
  - 遞迴函數在每次呼叫時，都可以使問題範圍逐漸縮小。
  - 遞迴函數需要擁有終止條件，以便結束遞迴函數的執行。

遞迴條件

- 在程式語言的世界中，遞迴簡單來說，就是一 個函數在定義中呼叫自己。
- 一個遞迴如果沒有終止條件，那它就會先呼叫自己
  然後自己呼叫的自己再呼叫自己
  然後自己呼叫的自己呼叫的自己再呼叫自己
  然後自己呼叫的自己呼叫的自己呼叫的自己再呼叫自己
- 初始(終止)條件:如果沒有初始條件，遞迴函數永遠也跑不完
  遞迴條件:如果沒有遞迴條件，就只是普通的函數了

```
function a(n) {
    if (n == 0) {  //終止條件裡面沒有遞迴
        return 1;
    }
    return n * a(n - 1);  //遞迴條件
}
```

遞迴涵式-階層

- 遞迴最常見的應用是數學上定義的階層函數n! 舉例來說計算4!的值，從上述定義n>0，可以使用n!定義的第2條計算階層函數4!的值，如下所示：
  4!=4*3*2*1=24
- 又可以將4!的計算分解成子問題：

4!=4*(4-1)!=4*3!
3! = 3*(3-1)! = 3*2!
2! = 2*(2-1)! = 2*1!
1! = 1*(1-1)! = 1*0! = 1

```
function a(n) {
    if (n == 0) {  //終止條件裡面沒有遞迴
        return 1;
    }
    return n * a(n - 1);  //遞迴條件
}
```

- 當範圍改變逐漸縮小到一個終止條件，階層函數值也就計算出來了。