

行動網頁前端開發技術

JavaScript 程式設計 – 基本入門 (下)

講師：陳俊吉

cccluke@gmail.com

2016

課程大綱

- 程式基礎 - 陣列與迴圈
- 自訂功能函數 - 參數與傳回值
- 流程圖概要
- 基本網頁事件
- 變數的有效範圍與生命週期
- 物件導向與網頁物件架構DOM
- 動態效果核心技巧 - 定時與逾時
- 自訂物件
- 常用系統內建物件

資料變數的集合體 – 陣列

- 說明：

- 當同類資料變數的數量較多，需要統一管理時，就可以透過由連續的記憶體空間所組成的一種集合體來進行一致的控管處理。而這個集合體最常見且普及的一種形式，就是「陣列」（Array）。

- 使用方式：

- 其內容是以一個陣列變數名稱，搭配一個索引值來取得。
- 索引值由0開始計數。
- 範例：

```
myArr[0] = "1234";
myArr[1] = "abcd";
```

資料變數的集合體 – 陣列

■ 宣告方式

- 以內建Array函數來宣告
- 形式一：空陣列
 - 範例：`new Array()`
- 形式二：直接以內容產生
 - 範例：`new Array("a" , " b" , " c")`
- 形式三：方括弧直接宣告
 - 範例：`["a" , " b" , " c" , " d"]`

資料變數的集合體 – 陣列

- 常用屬性
 - `length` 陣列內容數量
- 常用方法
 - `push` 在最後附加一個新元素
 - `pop` 移除最後一個元素
- 取用方式
 - 陣列名稱[索引值]

- 範例

```
var myArr = new Array();
myArr[0] = "1234";
myArr[1] = "abcd";
console.log("陣列大小:" + myArr.length);
console.log("第一個元素:" + myArr[0]);
```

反覆執行處理作業的關鍵 – 迴圈

- 定義：
 - 程式裡頭常常有需要依序反覆進行處理的作業時，就需要透過「迴圈」（loop）機制來達成。
- 運作方式：
 - 徹底機制除了會反覆執行處理作業外，最重要的是會根據某些條件狀況來決定是要從頭繼續反覆進行，或者是該結束迴圈機制。
- 案例：
 - 從1數到100。
 - 遍增陣列索引值來存取陣列裡的每個元素。

迴圈 Part 1 – while迴圈

- 基本語法：

```
while( 迴圈條件式 ) {
    // 條件成立時的要重複進行的處理作業
}
```
- 運作方式：
 - 根據迴圈條件式（通常是一種比較運算）
若條件式成立，
就會繼續地重複執行底下程式區塊的處理作業，
當處理迴圈完成後，會再次從頭比對迴圈條件式。
若條件式不成立，
則會結束（離開）迴圈，並繼續緊接在後的陳述式。
- 範例：

```
while( count < 10 ) {
    console.log(count);
    count++;
}
```

迴圈 Part 2 – for迴圈

- 基本語法：

```
for( 初始陳述; 迴圈條件; 變化陳述 ) {  
    // 需要重複進行的處理程序  
}
```
- 運作方式：
 - 迴圈第一次運作時，會先執行一次初始陳述。
 - 根據迴圈條件式（通常是一種比較運算）
若條件式成立，
就會繼續地重複執行底下程式區塊的處理作業，
當處理迴圈完成後，會先執行一次變化陳述內容，
然後再次從頭比對迴圈條件式。
若條件式不成立，
則會結束（離開）迴圈，並繼續緊接在後的陳述式。

迴圈 Part 2 – for迴圈

- 範例：

```
for( i = 0; i < 10 ; i++ ) {  
    objArr[i] = i * 10;  
}
```
- 練習：
 - 以迴圈來計算出1加到100的結果。
 - 以迴圈來反覆產生陣列內容，並取得其內容。

自訂功能函數 - function

- 將經常會被重複使用的一連串演算陳述式，獨立成一段程式區塊，並加以命名，以便取用（呼叫）。

- 定義方式：

```
// 基本簡易函數  
function 函數名稱 () {  
    // 演算陳述式  
}
```

- 使用方式：

- 直接輸入函數名稱。

- 範例：

```
// 定義 myfunc1  
function myfunc1() {  
    alert("執行myfunc1");  
}  
  
myfunc1(); // 呼叫
```

自訂功能函數 – 有參數的函數

- 外部呼叫部分：

- 當呼叫某些函數時，可以把相關數據資料傳遞給函數，以便依據那些資料進行相關演算處理。
- 被傳遞給函數的數據資料，就被稱之為「參數」(parameter)。

- 使用方式：

- 直接輸入函數名稱，並在括弧內輸入必要參數。

- 範例：

```
myfunc2(100);  
sum(1, 100);
```

自訂功能函數 – 有參數的函數

- 內部接收部分：

- 函數內部會有相對應的變數來保存呼叫者所傳遞過來的數據資料，以便進行相關演算處理。
- 被傳遞進來的這些數據資料，以函數內部來說，稱之為「引數」(argument)。

- 定義方式：

```
// 具參數的函數  
function 函數名稱(引數1, 引數2, ...) {  
    // 演算陳述式  
}
```

- 範例：

```
// 定義 myfunc2  
function myfunc2( num ) {  
    alert("引數num內容為：" + num);  
}
```

```
myfunc2(100); // 呼叫
```

自訂功能函數 - return

- 在函數的程式區塊裡頭，若要把演算結果傳回給呼叫者，即可使用**return**來回傳演算結果。而呼叫者可使用一個變數來接收（保存）被傳回的內容。
- 語法：

```
function 函數名稱 () {  
    // 演算陳述式  
    return 演算結果; // 回傳結果  
}  
  
變數 = 函數名稱(); // 呼叫並取得傳回值
```

- 練習：
 - 試著編寫一些簡單函數、有參數的函數，以及有傳回值的函數。
 - 撰寫一個計算某數學式（如：平方）並傳回結果的函數。

程式設計藍圖 - 流程圖

- 目的

- 用來呈現某項任務所經歷的各種作業與運作程序的視覺化表達方式。

- 優點

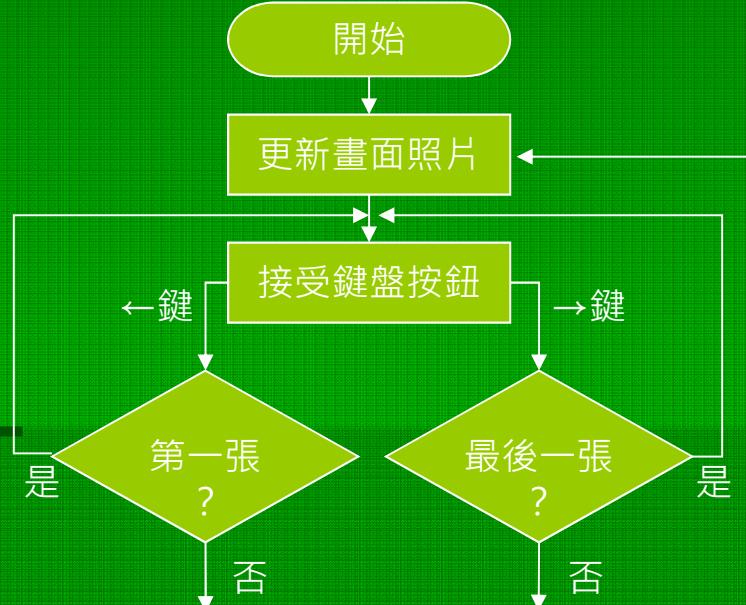
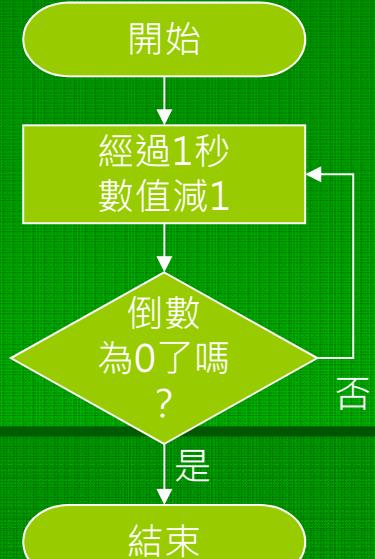
- 流程便具體
- 邏輯更清晰
- 實作高效率

程式設計藍圖 - 流程圖

- 使用時機
 - 程式撰寫前的分析與規劃階段
- 基本元素
 - 開始/結束端點 → 
 - 處理程序 → 
 - 條件判斷 (分歧點) → 
 - 程序動線 (連接線) → 

程式設計藍圖 - 流程圖

■ 實例



淺談網頁事件

■ 事件 (Event)

- 程式執行過程中，任何由程式內部或外部所發生的狀況變化。例如：滑鼠、鍵盤互動操作、檔案下載完成、音樂播放結束...甚至連瀏覽器的各種變動情況，也都會引發事件。
- 常見事件種類
 - 滑鼠：按下、移入、移出、移動...
 - 鍵盤：按下、放開
 - 瀏覽器：網頁載入完成、圖片下載完成、瀏覽器視窗變化。

HTML元素上的滑鼠互動事件

- 設定方式：

- 直接把事件設定在HTML元素的屬性上。
- 範例：

```
<input type="button" value="測試"
      onclick="alert('滑鼠點按');" />
```

- 常見事件：

- onclick – 滑鼠點按
- onmouseover – 滑鼠游標移入
- onmouseout – 滑鼠游標移出

- 練習：

- 以按鈕元素，或其他網頁元素來測試滑鼠事件。

函數的第二種定義方式

- 把函數給儲存到某個變數裡頭，同樣也可以達到定義函數的目的，達到相同效果。此時，變數名稱也就等同於函數的名稱。
- 定義方式：

```
var 變數名稱;  
變數名稱 = function (引數1, 引數2, ...) {  
    // 演算陳述式  
};
```

- 等同於：
- ```
function 函數名稱 (引數1, 引數2, ...) {
 // 演算陳述式
}
```

# 函數的第二種定義方式

- 呼叫方式：
  - 與原本函數使用方式完全一樣。  
也就是說，呼叫該函數的方式，就是直接把變數名稱當作函數名稱來使用。
- 範例：

```
// 定義函數
var testFunc;
testFunc = function () {
 // 演算陳述式
};

testFunc(); // 呼叫函數
```
- 練習：
  - 改用本次的方式來定義函數，並試著呼叫看看。

# 網頁載入完成事件 - **onload**

- 當瀏覽器視窗裡的網頁被整體載入完成時，就會引發**onload**事件，因此若有處理程序希望在網頁整體內容都確實載入時，才去啟動與運作的話，就會把作業程序指定給視窗的**onload**事件。
- 指定方式：

```
window.onload = function() {
 // 網頁載入後要執行的處理程序
};
```
- 練習：
  - 加入簡單程序來測試**onload**事件處理作業。

# 變數的有效範圍與生命週期

- 定義：

- 變數會根據其宣告的時機與位置，而擁有其特定的有效範圍（scope, 視野）和生命週期（life cycle）。

- 全域變數（global）

- 在程式最外層所宣告出來的變數。
  - 有效範圍：程式裡的任何地方。
  - 生命週期：整個程式運作期間。

- 區域變數（local）

- 在函數裡頭所宣告出來的變數。
  - 有效範圍：函數範圍內。
  - 生命週期：在函數執行結束後，就跟著結束。

# 區域與全域變數的使用問題

## ■ 可存取性：

- 在區域範圍裡，可直接存取到全域變數；然而，在區域範圍外的地方，將無法存取到該區域範圍裡的區域變數。

## ■ 名稱衝突問題：

- 在區域範圍內，若宣告了與全域變數同名的區域變數，則在該區域範圍內，將以該區域變數為準。

## ■ 範例：

```
var count = 100; // 全域變數

function func1 () {
 var count; // 區域變數
 count = 50;
 alert("區域變數count = " + count);
};

func1();
alert("全域變數count = " + count);
```

# 物件導向(Object Oriented)概要

## ■ 何謂「物件」？

- 將相關的資料變數，以及功能函數，所集合而成的一個整體，就是一個「物件」( Object )。

## ■ 物件內涵：

- 在物件裡頭，用來保存相關資料的變數，稱為「屬性」( property )；用來進行相關演算的函數，就叫「方法」( method )。

## ■ 以「人」為例：

- 「屬性」就相當於人的身高、年齡、體重之類的純「資料」數值。
- 「方法」就像是人擁有跑步、打掃、開車這類的行為（處理作業）或能力（功能）。

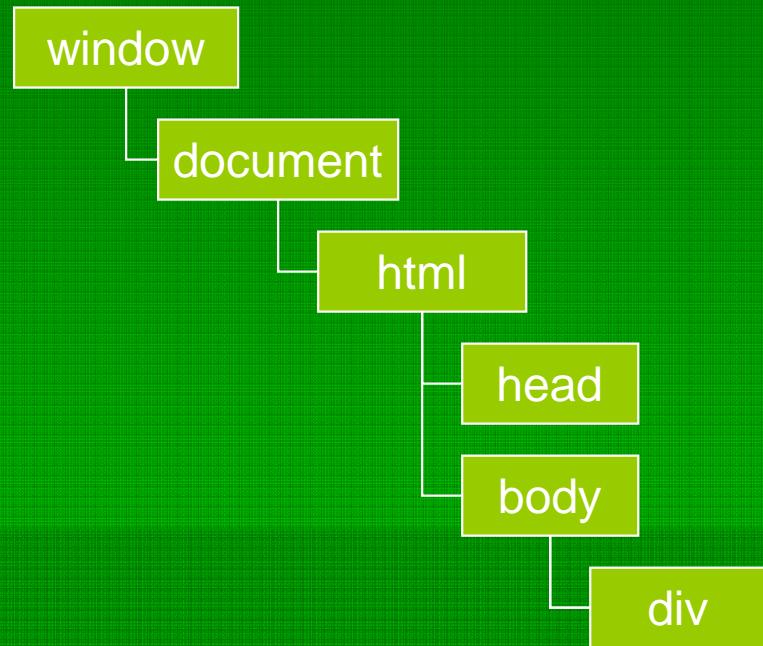
# 網頁物件架構 - DOM

## ■ 定義：

- 整個頁面裡的所有元素物件，在記憶體裡頭所組成的一個整體物件結構，就統稱為「DOM」，全名為「Document Object Model」（文件物件模型）。

## ■ 用途：

- 程式可以透過對DOM進行存取或處理，來操作或改變網頁各元素的外觀樣貌或行為。



# 物件內容存取方式

- 屬性：

物件名稱.屬性名稱

- 方法：

物件名稱.方法名稱( 參數1, 參數2, ... )

- 範例：

```
var menuObj;
menuObj = document.getElementById("menu");
menuObj.innerHTML = "<p>內部HTML選單測試</p>" ;
```

# 基本網頁元素 - document

- 常見屬性：
  - `referrer`：連結到目前頁面的來源（參考）位址。
  - `title`：網頁文件標題（可存取）。
  - `URL`：網頁所在位置（完整位址）。
- 常見方法：
  - `getElementById( ID名稱 )`：以ID取得網頁元素。
  - `write( 輸出內容 )`：輸出網頁內容到文件上。
- 完整參考列表：
  - [http://www.w3schools.com/jsref/dom\\_obj\\_document.asp](http://www.w3schools.com/jsref/dom_obj_document.asp)

# 網頁元素物件基本操作

- 常用基本屬性：

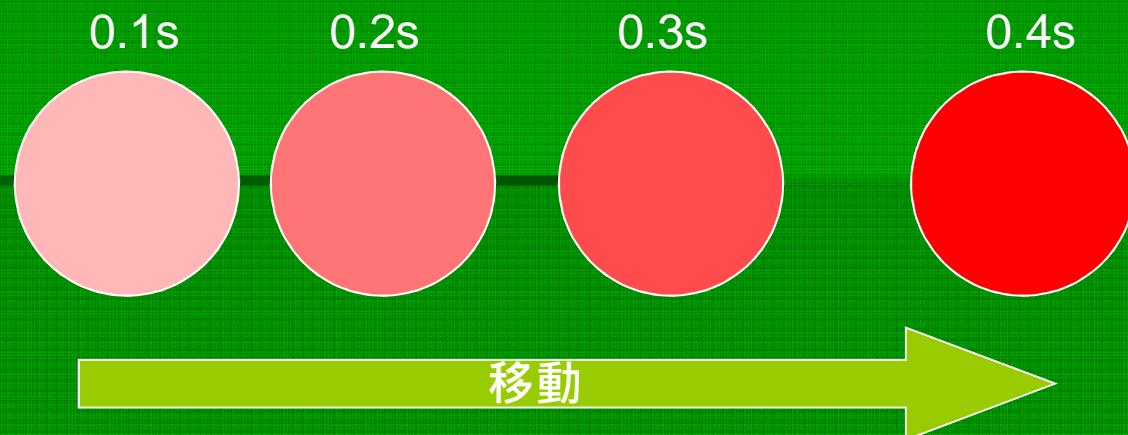
- innerHTML：元素內的HTML格式內容
- textContent：元素內的純文字內容
- style：元素的樣式設定集合
  - style.樣式屬性：樣式的細節屬性設定。
- className：樣式的類別

- 完整參考列表：

- [http://www.w3schools.com/jsref/dom\\_obj\\_all.asp](http://www.w3schools.com/jsref/dom_obj_all.asp)

# 動態效果核心技巧 - 定時與逾時

- 畫面上的物件動態移動，或是隨時間變化的各種動態效果，通常都是以定時方式去進行某些變更（如位置、顏色），或處理作業的方式來達成的。



# 動態效果核心技巧 - 定時與逾時

## ■ 定時相關函數：

- 啟動：每隔一段時間（毫秒），定期地去執行指定函數。
  - 語法：`定時作業代號 = setInterval( 執行函數名稱, 間隔時間 );`
- 停止：結束定時去執行指定函數的處理作業。
  - 語法 `clearInterval( 定時作業代號 );`

## ■ 逾時相關函數：

- 啟動：在經過逾時時間（毫秒）後，執行一次指定函數。
  - 語法：`逾時作業代號 = setTimeout( 執行函數名稱, 逾時時間 );`
- 停止：結束逾時後去執行指定函數的處理作業。
  - 語法：`clearTimeout( 逾時作業代號 );`

# 動態效果核心技巧 - 定時與逾時

- 案例：

- 選單從畫面邊緣滑進與滑出畫面（slide）。
- 畫面物件變大變小（resize）。
- 物件在畫面上淡入與淡出（fade）。
- 時鐘、倒數計時、鬧鐘等類似功能。

- 練習：

- 以顯示簡單訊息的方式來練習定時和逾時的差別。
- 透過改變物件屬性的方式，來讓物件進行顏色變化、位置移動，或大小改變之類的動態效果。
- 搭配時間物件來撰寫簡易小時鐘功能。

# 常見系統內建物件 – 日期物件

- 物件名稱 – Date

- 建立方式 ( 取得當下時間 )
    - new Date()
  - 常用方法
    - getFullYear() : 取得年 ( 四位數 )
    - getMonth() : 取得月 ( 0~11 )
    - getDate() : 取得日 ( 1~31 )
    - getHours() : 取得時 ( 0~23 )
    - getMinutes() : 取得分 ( 0~59 )
    - getSeconds() : 取得秒 ( 0~59 )

- 完整參考 :

- [http://www.w3schools.com/jsref/jsref\\_obj\\_date.asp](http://www.w3schools.com/jsref/jsref_obj_date.asp)

# 自訂物件

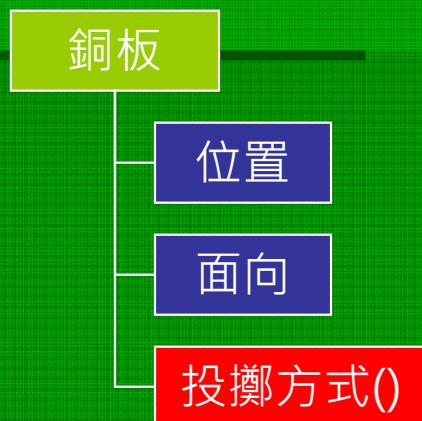
- 概要：

- 在實際運用上，常常需要把有相關性的變數和函數，打包成一個整體的獨立物件，以方便去進行系統的處理和運算。

- 案例：

- 在擲銅板遊戲中，銅板物件會有它存在畫面上位置，目前的面向（正面或反面），以及其投擲或運算方式。

- 物件架構：



# 自訂物件方式1 – 簡易宣告

- 簡易物件宣告方式：
  - 使用大括弧字元括住，並採用類似CSS樣式的定義格式。

- 語法：

```
物件變數 = {
 屬性名稱 : 屬性值,
 屬性名稱 : 屬性值,
 ...
 方法名稱 : function() {
 // 方法處理作業
 },
 方法名稱 : function() {
 // 方法處理作業
 },
 ...
};
```

- 範例：

```
var dice0 = {
 x : 100,
 y : 100,
 num : 3,
 showNum : function() {
 alert(this.num);
 }
};

dice0.x = 200; // 改變屬性值
dice0.num = 1;
dice0.showNum(); // 呼叫物件方法
```

# 特殊保留字 – this

- 說明：
  - 在物件定義的範圍當中，若需要參考到物件本身，就可以透過this這個特殊保留字來達成。
- 使用方式：

|                                         |                      |
|-----------------------------------------|----------------------|
| <code>this.屬性名稱</code>                  | <code>// 物件屬性</code> |
| <code>this.方法名稱( 參數1, 參數2, ... )</code> | <code>// 物件方法</code> |

# 自訂物件方式2 – 建構子

- 說明：
  - 簡易物件宣告方式一次只能定義出一個獨立物件，若需要可以多次產生具有同樣結構的多個物件，則要採用建構子（constructor）宣告方式。

- 建構子宣告：

```
function 建構子名稱(參數1, 參數2, ...) {
 this.屬性名稱 = 屬性值; // 定義屬性
 this.屬性名稱 = 屬性值; // 定義屬性
 this.方法名稱 = function() { // 定義方法
 // 方法處理作業
 };
 ...
}
```

- 物件產生方式：

```
new 建構子名稱(參數1, 參數2, ...)
```

# 自訂物件方式2 – 建構子

- 範例：
  - 產生兩個結構一樣，內容不同的骰子。

```
function Dice(x, y, num) { // 定義建構子內容
 this.x = x;
 this.y = y;
 this.num = num;
 this.showNum = function() {
 alert(this.num);
 };
}
```

```
var dice1 = new Dice(100, 100, 3); // 產生第一顆骰子
dice1.num = 1; // 改變屬性值
dice1.showNum(); // 呼叫物件方法
var dice2 = new Dice(200, 100, 2); // 產生第二顆骰子
dice2.showNum(); // 呼叫物件方法
```

# 物件屬性的有效範圍與生命週期

- 有效範圍 ( scope )：
  - 物件外部：只要能存取到該物件，就可以直接透過「物件名稱.屬性名稱」的方式存取到該物件屬性。
  - 物件內部：在物件內的任何地方，都可以透過「`this.屬性名稱`」的方式來存取本身的物件屬性。
- 生命週期 ( life cycle )：
  - 由於是依附著物件，因此與生命週期與物件完全相同。
- 特別注意：
  - 物件屬性是依附在物件上的特殊變數，它既不是位於物件外頭的全域變數，卻也不是在某個物件方法裡的區域變數。

# 物件架構與屬性關係

- 物件宣告程式碼：

```
function Dice(x, y, num) {
 this.x = x;
 this.y = y;
 this.num = num;
 this.setNum = function(newNum) {
 this.num = newNum;
 alert(this.num);
 };
}
```

- 物件結構示意圖：



# 外部JS檔案

- 說明：
  - 與CSS樣式表一樣，JavaScript也可以編寫成一個獨立程式庫檔案（副檔名使用.js），並在需要用到那些程式碼的網頁當中，於script標籤裡頭，以src屬性來指定要引用的檔案。
- 引用語法：

```
<script src="程式庫檔案位址"></script>
```
- 範例：

```
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script src="myJs.js "></script>
```
- 練習：
  - 試著把自己的程式碼獨立到一個外部.js檔案裡頭。

# 實戰案例練習 – 擲骰子遊戲

- 建立畫面：
  - 在頁面上繪製出一顆骰子和一個「Play」按鈕。
- 建立物件：
  - 建立骰子物件，並讓它與畫面上的圖像建立起關連性。
- 實際運作：
  - 按下「Play」，即可啟動擲骰子動作，並將最後結果顯示在骰子圖像上。
- 延伸練習：
  - 三顆骰子，並計算總點數。
  - 分成電腦和玩家兩方各三顆骰子，按下「Play」後，比對兩邊大小，並在畫面上顯示輸贏結果。

# 常見系統內建物件 – 數學物件

## ■ 物件名稱 – Math

- 建立方式：由於這是原本就存在的內建物件，因此可直接取用，而不需要另外建立新物件。

## ■ 常用屬性：

- PI : 圓週率 ( 大約為 3.14 )

## ■ 常用方法：

- abs(數值) : 取絕對值
- floor(數值) : 無條件省略小數 ( 取整數 )
- round(數值) : 四捨五入到整數 ( 以小數點以下一位數決定 )
- random() : 隨機數值 ( 0.0~1.0 之間的小數數值 )

## ■ 完整參考：

- [http://www.w3schools.com/jsref/jsref\\_obj\\_math.asp](http://www.w3schools.com/jsref/jsref_obj_math.asp)

42

# 常用系統內建物件 – 網址與視窗

## ■ 網址物件 - **location**

---

- href : 目前網頁網址 ( 可存取 ) 。
- reload() : 重新載入網頁。
- 相關參考列表：
  - [http://www.w3schools.com/jsref/obj\\_location.asp](http://www.w3schools.com/jsref/obj_location.asp)

## ■ 視窗物件 – **window**

- status : 狀態列內容 ( 可存取 ) 。
- open(網址,名稱,其他設定) : 開啟新視窗。
- 相關參考列表：
  - [http://www.w3schools.com/jsref/obj\\_window.asp](http://www.w3schools.com/jsref/obj_window.asp)