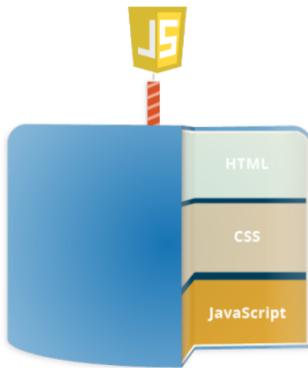


簡介

什麼是 JavaScript ?



- 通常我們說簡稱為 "JS"
- 由 Brendan Eich 發明
- 1996 年交給歐洲電腦製造商協會 ECMA(European Computer Manufacturer's Association) 進行標準化
- 與 HTML & CSS 共同打造網頁
 1. HTML 標籤來描述網頁裡的內容與架構
 2. CSS 制定樣式，美化網頁
 3. JS 具 **立即性** 地動態改變網頁，與使用者互動

關於 JavaScript

- JS 屬於 **腳本語言** (scripting language)、直譯語言(Interpreted language) 的程式語言類型，跟另一類的編譯語言不同喔!
- 直譯程式語言，指程式碼 **由上到下執行**，而且執行的結果是**立即**得到回應的。因此在瀏覽器執行前，我們不需要將程式轉換為其它的形式喔！
- 大部分瀏覽器都會有 JS 直譯器去解讀 JS 後，轉譯成電腦懂的指令去執行

JavaScript 能做什麼 ??

JS為 Web application 客戶端 (client) 的網頁技術之一，也就是說，JS為 **客戶端動態網頁設計** 的技術之一

- JS可以在網頁中添加新的**HTML**
- 修改網頁**已有的**內容和網頁的樣式
- **回應用戶**的行為
- 向遠程服務器發送網路請求 (所謂的 **AJAX** 技術)
- 簡單的表單驗證
- 獲取或設置 cookie

JavaScript 不能做什麼？

- 為了用戶的（訊息）安全，在瀏覽器中的 JavaScript 的能力是受限的。目的是防止惡意網頁獲取用戶私人訊息或損害用戶數據
- 不能讀、寫、複製和執行硬碟上的任意文件，它沒有直接訪問操作系統的功能
- 相機/麥克風要獲得用戶的明確許可
- 不同的標籤頁/窗口之間不能相互通信（同源策略）

補充：

瀏覽器環境外（例如在服務器上）使用JS，則不存在此類限制，因此，不要忘記JS是有些限制的！

開始使用

直接在HTML使用

```
1 | <script>(...想寫的寫在這裡...)</script>
2 |
3 |
```

- 可寫在<head>標籤中
- 可寫在<body>標籤中
- 可寫在<body>標籤後(提升網頁效能)
- 通常函數或事件處理程式寫在<head>標籤中
- 可以內嵌多份的<script> ... </script>區塊

注意：

學習中的我們，可以用寫在<body>標籤中來看效果
長大的我們，建議可寫在<body>標籤後，更推的方法是下一個方法，就是用.js檔來寫

在.js檔案中使用

如果要寫在js檔案中的話，我們要透過html中的script標籤，並且透過屬性src連結到這隻.js檔案

```
1 | <script src="JavaScript的外部檔案"></script>
2 |
3 |
```

注意：

外部檔案的內容不需再加上<script>標籤喔，並且我們不會再這個script標籤中再寫JS了～

觀察

👉 參閱 : 1_Introduction.html

觀察 <head> 標籤中的 <script>

```
1 | <head>
2 |
3 |     <!-- 這裡可以放外部連結的js檔案，中間不要再放入其他程式喔 -->
4 |     <script src="js/new.js"></script>
5 |
6 |     <!-- 通常函數或事件處理程式寫在<head>標籤中。 -->
7 |     <script>
8 |         document.write("head標籤中，通常函數或事件處理程式寫在<head>標籤中<br>");
9 |     </script>
10|
11| </head>
12|
```

觀察 <head> 標籤中的 <script src> 中的程式

```
1 | 
2 |     document.write('我在.js中喔！<br>');
3 |
```

觀察 <body> 標籤中的 <script>

```
1 |
2 |     <script>
3 |         document.write("Hello World<br>");
4 |     </script>
5 |     <script>
6 |         document.write("body標籤中一<br>");
7 |     </script>
8 |
9 |     <script>
10|         document.write("body標籤中二<br>");
11|     </script>
12|
13|     <script>
14|         document.write("body標籤中三<br>");
15|     </script>
16|
```

觀察 <body> 標籤之後的 <script>

```
1 |
2 |     <script>
3 |         document.write("body標籤後，提升網頁效能<br>");
4 |     </script>
5 |
```

等等!剛剛的document.write是 ?



瀏覽器視為一個物件

其中我們會說 window 物件以及document物件
而document.write就是**document**物件的方法之一

物件會有：
物件的方法
物件的屬性

作用

可以輕鬆的在網頁上顯示文字將字串內容自動印到頁面上可以印變數，字串，還可以包含 HTML 的標籤印到頁面上等等

用法

物件.方法 (參數值)

- `document.write(這裡放內容)`，用單引號 (`'`) 或雙引號 (`"`) 將字串包起來，這樣瀏覽器會自動把這些文字當成字串來呈現
- 如果你要呈現的文字內容是變數，則不需要放在引號中，必須用逗號 (`,`) 或加號 (`+`) 來連接變數字串。
- 當變數較多時，建議一律使用加號(`+`)來串接變數字串，避免瀏覽器判斷失誤而造成的顯示出錯。
- 新方法：模板字符串 (template literal) : `document.write(` ${ 變數 } 文字數字標籤 `)`

觀察

👉 參閱 : 2_documnet_write.html

```
1  document.write("直接寫文字就OK<br>");
2  var testString = "Have a good time." // 變數
3  document.write(testString, "<br>"); 
4  document.write("Welcome1" + testString + "<br>"); 
5  document.write("Welcome2", testString, "<br>"); 
6  document.write(`Welcome3${testString}<br>`); 
7  //模板字串符: ``反引號就可以直接寫j文字，遇到變數的時候加上${變數}
8
9
```

👉 補充 :

document.write 初學者適合使用
但較少被程式設計師使用，因為不太滿足使用經驗...

JavaScript 小提醒

- 大小寫視為不一樣，並以半形字來敘述喔！
- 結尾養成好習慣加上「；」(分號)，表示結束

註解

- 註解程式前面加上「//」(可以直接按下 **ctrl+ /**)

```
1  // document.write("直接寫文字就OK<br>");
2
3
```

- 註解 (comments) 可以用來說明這段 code 的用途，會使他人 (以後的自己)，更容易閱讀及理解這段 code，之後可以更好上手或維護程式
- 可以先寫下註解幫助思考再進行 coding

```
1
2  function init(){
3      //設定spanLogin.click 事件處理程序是 showLightBox
4      //彈窗
5
6      //設定btnLogin.click 事件處理程序是 sendForm
7      //檢查有沒有這個會員
8
9      //設定btnLoginCancel.click 事件處理程序是 cancelLogin
10     //登出
11
12 };
13
14 //window.load
15 window.addEventListener('load',init);
16
```

變數 (variable)

宣告變數

變數是數據/資料的“命名存儲”我們可以使用變數來保存商品、訪客和其他訊息，我們可以宣告一個變數，就是申請個儲存空間，用來存放可以變動的資料結構

先來看一個宣告變數的過程

```
1 // 先來命一個為 product 的變數
2 var product;
3
4 // 現在，我們可以通過指定運算符 = 為變數添加值
5 product = 200;
6
7 // 現在這個值已經保存到與該變數相關聯的內存區域了
8
9
```



補充：

var 是宣告變數的關鍵字，也就是程式讀到var就知道要來宣告變數了（後面會有更多解說）

我們可以通過使用該變數名稱訪問/使用它

```
1
2 var product;
3 product = 200;
4 document.write("$" + product); // $200
5
```

簡潔一點，我們可以將變數定義和給值合併成一行來寫

```
1
2 var product = 200;
3 document.write("$" + product); // $200
4
```



參閱：3_var_let_const.html 的宣告變數

變數的命名

- 變數必須使用字母 (letter) 、下底線 (_) 、錢號 (\$) 作為開頭
- 後面的字員組成可以包含數字 (0-9)
- 區分大小寫
- 建議取有意義的名稱，看得懂的
- 常見變數 / 方法命名規則(**snake_case || camelCase || PascalCase ...**)

👉 參閱 : 3_var_let_const.html 的取變數名字要注意的地方

```
1 //var 1pp =20;
2 //不行喔!!!會報錯誤
3 //錯誤訊息 : Uncaught SyntaxError: Invalid or unexpected token
4
5 var pp1 = 20;//ok
6 var apple = "小寫蘋果";
7 var APPLE = "大寫蘋果";
8 document.write("pp1:", pp1, "<br>"); //20
9 document.write("apple:", apple, "<br>"); //小寫蘋果
10 document.write("APPLE:", APPLE, "<br>"); //大寫蘋果
11
12
```

補充:開發者工具

- 在具備除錯功能的瀏覽器上，window物件中會註冊一個名為console的，指代除錯工具中的控制檯
- 通過呼叫該console物件的log()函式**console.log("文字"或變數)**，可以在控制檯中印出資訊

開啟開發工具

1. 按F12 < Console
2. 滑鼠右鍵 < 檢視 < Console
3. Ctrl + shift + i < Console

👉 參閱 : 3_var_let_const.html 的取變數名字要注意的地方以及**console.log**

觀察

我們先將剛剛在變數的命名的錯誤範例打開來~

```
1 | var 1pp =20;
2 |
3 |
```

因為這樣的命名不符合規定

因此會產生報錯

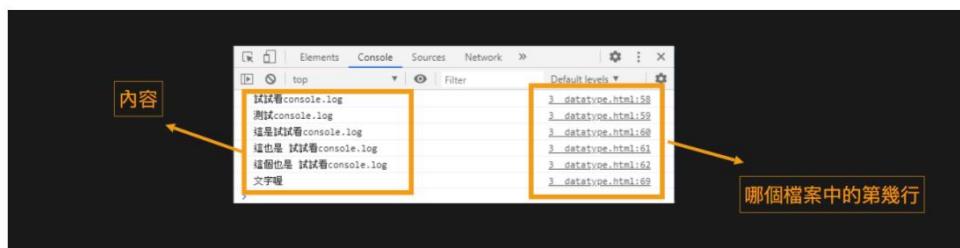
打開開發者工具會看到以下內容

```
✖ Uncaught SyntaxError: Invalid or unexpected token 3 var_let_const.html:33
```

在開發者工具中可以看到回報錯誤的內容以及這段是在哪支檔案中的第幾行

另一種幫助我們查看訊息的方法是**console.log()**

```
1 | var testString = "試試看console.log";
2 | console.log(testString);
3 | console.log("測試console.log");
4 | console.log("這是" + testString);
5 | console.log("這也是", testString);
6 | console.log(`這個也是 ${testString}`);
7 |
8 |
```



在開發者工具中可以看到我們輸入的內容以及這段是在哪支檔案中的第幾行

宣告變數的關鍵字

在 JS 中有三種宣告變數的方法

- **var** 可重複宣告、有 hoist 現象、沒有區塊作用域{ }
- **let** 不可重複宣告、沒有 hoist 現象、有區塊作用域{ }
- **const** 壓告一個只可讀取的不可變常數

```
var
```

var 有 hoist 現象

👉 參閱 : 3_var_let_const.html 的**var 有hoist**

```
1  document.write(a, "<br>"); //undefined
2  var a = 10;
3  document.write(a, "<br>"); //10
4
5
```

!!! 前面不是說JS一行一行讀下來嗎?變數 a 沒宣告也沒有給值???這樣可以嗎?
那我們繼續看下去

```
1  //已經把宣告的動作提昇到這裡
2  document.write(a, "<br>"); //undefined
3  //注意一下，當我們變數尚未給值的時候啊，這的變數會是“未定義”的值
4
5  var a = 10; //給值的動作在此才做;
6  document.write(a, "<br>"); //10
7
8
```

在 JavaScript 裡面，如果你試圖去對一個還沒宣告的變數取值，會發生以下錯誤

```
document.write(a)
// ReferenceError: a is not defined
```

會回傳一個 **a is not defined** 的錯誤，因為你還沒宣告這個變數，所以 JavaScript 也找不到這變數在哪，自然就會報錯

可是如果你這樣子寫：

```
document.write(a); // undefined
var a;
```

從以前學程式的時候我們就學到了一個觀念，「程式是一行一行跑的」，那既然是一行一行跑的，執行到第一行的時候不是還沒有宣告變數 a 嗎？那為什麼不是拋出 **a is not defined** 的錯誤，而是輸出了 **undefined** ?

這種現象就叫做 **hoisting**，提升，在第二行的 **var a** 被「提升」到了最上面，但就是 **只有變數的宣告會提升，值不會**，所以 **undefined**

var 可以重覆宣告

👉 參閱 : 3_var_let_const.html 的**var** 可以重覆宣告

就是變數名一樣是沒有問題的
困擾：可能會將不同作用的變數相互污染

```
1  var a = 100;
2  var a = 500;
3  document.write(a,"<br>");
4  //500
5
6
```

var 沒有區塊作用域

👉 參閱 : 3_var_let_const.html 的**var** 沒有區塊作用域

```
1  var avg = 66;
2
3  if(avg >= 60){
4      var word = "好棒~";
5  }else{
6      var word = "加油~";
7  }
8
9
10 document.write(word,"<br>"); //好棒
11 // 不會被if~else的 "{}" 困住
12
```

相信大家還是有點不太懂
接著來跟 **let** 比較會比較有感覺些

let

let 沒有hoist

👉 參閱 : 3_var_let_const.html 的let 沒有hoist

```
1 //document.write(aa, "<br>"); //此行會error  
2 //Uncaught ReferenceError: Cannot access 'aa' before initialization  
3 let aa = 10;  
4  
5
```

跟 var 很不一樣喔！let並不會幫忙提升

let 不可以重覆宣告

👉 參閱 : 3_var_let_const.html 的let 不可以重覆宣告

```
1 let sgo = 11;  
2 //let sgo = 12; //Uncaught SyntaxError: Identifier 'sgo' has already been declared  
3 document.write(sgo, "<br>");  
4  
5
```

let 有區塊作用域

👉 參閱 : 3_var_let_const.html 的let 有區塊作用域

```
1 var avg = 66;  
2 if(avg >= 60){  
3     let words = "好棒~"; //區塊作用域，只能在此區塊中使用  
4     document.write(words, "<br>");  
5 }else{  
6     let words = "加油~";  
7     document.write(words, "<br>");  
8 }  
9 document.write(words, "<br>");  
10 //Uncaught ReferenceError: words is not defined  
11  
12
```

const

👉 參閱 : 3_var_let_const.html 的 **const**

const 比 let 更為不同，因其意義為不變的 **常數**，因它代表特定意義，因此在宣告時就要給它值，否則會報錯。

```
1 | const a;
2 | //Uncaught SyntaxError: Missing initializer in const declaration
3 |
4 |
```

在一般情況中，一旦它被賦值後，就不能被更動，也不能重複宣告

```
1 | const PI = 3.14;
2 | const PI = 3; // 重複宣告
3 | //Uncaught SyntaxError: Identifier 'PI' has already been declared
4 |
5 |
```

```
1 | const PI = 3.14; //不可變之常數
2 | PI = 3; //改值
3 | //Uncaught TypeError: Assignment to constant variable.
4 | //錯誤，不能對常數重新賦值
5 |
6 |
```

而在複合類型中，例如物件，如果你的值是物件(物件以後再說)更改不是指值的更改，而是記憶體地址的更改

```
1 | const obj = {
2 |   x: 20,
3 |   y: 50,
4 | };
5 | obj.x = 50;
6 | document.write(obj.x, "<br>"); //50
7 |
8 |
```

const 可以常規命名也可以大寫字母和下劃線來命名

作為一個常數，意味著值永遠不變

有些常數在執行之前就已知了（比如紅色的十六進制值），這樣我們就可以用都英文大寫跟底線來命名

```
1 | const COLOR_RED = "#F00";
2 | const COLOR_GREEN = "#0F0";
3 | const COLOR_BLUE = "#00F";
4 |
5 |
```

另一種是頁面加載之前是未知的，但在執行期間被計算出來的，這樣會採用常規命名

資料型別

JS 屬於 **弱型別**，不需事先宣告，指定值給變數時即決定變數的資料型別

常見型別

- 基本型別：
布林 (Boolean) : true / false
數值 (Number)
字串 (String)

- 簡單型別：
空值 (null)
未定義 (undefined)

- 複合型別：
陣列 (Array)
物件 (Object)

- 特殊型別：
函式 (Function)

Typeof 判斷資料型別

常見有六種種回傳值：

1. **boolean**
2. **number**
3. **string**
4. **object**
5. **function**
6. **undefined**

以上這些皆以 **字串(string)** 回傳資料型態
可用於判斷變數、物件等是否存在，防止錯誤訊息發生

用法：

作為運算符：**typeof x**
函數形式：**typeof(x)**

接著我們要透過 Typeof 來觀察 JS 的資料型別囉！！

資料型別-基本

👉 參閱 : 4_typeof_typedata.html 的資料型別-基本

```
1  var boolean_test = true;
2  var num_test = 123;
3  var string_test = "文字";
4
5  document.write(typeof boolean_test + "<br>");
6  // boolean
7  document.write(typeof num_test + "<br>");
8  // number
9  document.write(typeof string_test + "<br>");
10 // string
11
12
```

資料型別-簡單

👉 參閱 : 4_typeof_typedata.html 的資料型別-簡單

```
1  var null_test = null;
2  document.write(typeof null_test + "<br>");
3  // object 【 特別注意 】
4
5
6  var undefined_test;
7  // 沒定義沒給值
8  document.write(typeof undefined_test + "<br>");
9  // undefined
10
```

資料型別-複合

👉 參閱 : 4_typeof_typedata.html 的資料型別-複合

```
1  var arr = [1, 2, 3, 4, 5, 6];
2  document.write(typeof arr + "<br>");
3  // object 【 特別注意 】
4
5
6  var obj = {
7      name: "jack",
8      age: 20,
9  };
10 document.write(typeof obj + "<br>");
11 // object
12
```

資料型別-特殊

👉 參閱 : 4_typeof_typedata.html 的資料型別-特殊

```
1  function function_test() {
2      console.log("yoyo function");
3  }
4
5  function_test();
6
7  document.write(typeof function_test + "<br>");
8  // function
9
10
```

Typeof 判斷資料型別

皆以字串回傳資料型態，所以要使用到這個資料類型的時候要記得他是字串喔!不要忘記加上字串的引號喔(" " 不要忘)

👉 參閱 : 4_typeof_typedata.html 的Typeof判斷資料型別

```
1  if (typeof string_test == "string") {
2      console.log("文字喔");
3  }
4
5
```

資料型別轉換

還記的我們對於變數的或是字串加上變數我們會用加號 + 來串接嘛!
但用加號有個小細節要注意!就是加號本身是具備"運算式"的意思
所以JS這個弱型別會因為不同型別用加號串接，而改變原本的型別

隱含轉換

字串 > 數值 > 布林

運算式(Expression)	結果
數值(弱)和字串(強)相加	字串
數值(強)和布林(弱)相加	數值
字串(強)和布林(弱)相加	字串

👉 參閱 : 4_typeof_typedata.html 的型別轉換

```
1  var typen = 12;
2  document.write(typen + 2, "<br>"); // 14
3  // number + number = number
4
5
6  var typen = 12;
7  var types = "2.5";
8  document.write(typen + types, "<br>"); // 122.5
9  // number + string = string
10
11 var typen = 12;
12 var typeb = true; //1
13 document.write(typen + typeb, "<br>"); // 13
14 // number + boolean = number
15
16 var typen = 12;
17 var typebf = false; //0
18 document.write(typen + typebf, "<br>"); // 12
19 // number + boolean = number
20
21 var types = "2.5";
22 var typeb = true;
23 document.write(types + typeb, "<br>"); // 2.5true
24 // string + boolean = string
25
26
```

強制型別轉換

- parseInt(XXXX)
函數可解析一個字串，並返回一個整數
- parseFloat(XXXX)
函數可解析一個字串，並返回一個浮點數

👉 參閱 : 4_typeof_typedata.html 的強制型別轉換

```
1  var types = "2.8";
2
3
4  document.write(parseInt(types) + "<br>"); //2
5  document.write(typeof parseInt(types) + "<br>"); //number
6
7  document.write(parseFloat(types) + "<br>"); //2.8
8  document.write(typeof parseFloat(types) + "<br>"); //number
9
```

運算式

運算式(expression)是由“運算子(operators)”和“運算元(operands)”組成的

例如： $5 * 20 + 50$ 或者 $60 > 30$

其中 5、20、30、50、60 屬於運算元

$+$ 、 $*$ 、 $>$ 屬於運算子

運算式種類

1. 算術運算
2. 字串運算
3. 比較運算
4. 邏輯運算
5. 位元運算
6. 指定運算
7. 三元運算
8. `typeof`運算
9. 型別運

算術運算

運算的結果傳回一個數值性的資料

架設： y 值為 5

運算子	說明	範例	結果	備註
$+$	加	$x = y + 2$	$x = 7$	
$-$	減	$x = y - 2$	$x = 3$	
$*$	乘	$x = y * 2$	$x = 10$	
$/$	除	$x = y / 2$	$x = 2.5$	
$\%$	取餘數	$x = y \% 2$	$x = 1$	
$++$	遞增	$x = y ++$	$x = 6$	$++$ 擺在後面，回傳+之前數字5。 $++$ 擺在前面， $X=6$
$--$	遞減	$x = y --$	$x = 5$	$--$ 擺在後面，回傳+之前數字5。 $X=5$



補充：

$++$ 在後我們後加， $++$ 在前先加（ $--$ 也是一樣的道理）

👉 參閱 : 5_expression.html 的算術運算式

```
1  var x;
2  var y = 100;
3
4  x = y + 2;
5  document.write(x + "<br>"); //102
6
7  x = y - 2;
8  document.write(x + "<br>"); //98
9
10 x = y * 2;
11 document.write(x + "<br>"); //200
12
13 x = y / 2;
14 document.write(x + "<br>"); //50
15
16 x = y % 2;
17 document.write(x + "<br>"); //0
18
19 x = y % 3;
20 document.write(x + "<br>"); //1
21
22
23
```

```
1  var x;
2  var y = 100;
3  x = y++;
4  document.write("x:" + x + "<br>"); //100
5  document.write("y:" + y + "<br>"); //101
6
```

```
1  var x;
2  var y = 100;
3  x = y--;
4  document.write("x:" + x + "<br>"); //100
5  document.write("y:" + y + "<br>"); //99
6
```

```
1  var x;
2  var y = 100;
3  x = ++y;
4  document.write("x:" + x + "<br>"); //101 (先加)
5  document.write("y:" + y + "<br>"); //101
6
```

```
1  var x;
2  var y = 100;
3  x = --y;
4  document.write("x:" + x + "<br>"); //99 (先減)
5  document.write("y:" + y + "<br>"); //99
6
```

其他關於 `++--` 的範例

👉 參閱 : 5_expression.html 的算術運算式 `++在前`/算術運算式 `++在後`/算術運算式 `++在前`/算術運算式 `++在後`

```
1  var x = 10; //x : 10
2  var y = ++x + 5; //x : 11 , 11+5
3  document.write("x : ", x, "<br>");//11
4  document.write("y : ", y, "<br>"); //16
5
6
```

```
1  var x = 10; //x : 10
2  var y = x++ + 5; //x : 11 , 10+5
3  document.write("x : ", x, "<br>");//11
4  document.write("y : ", y, "<br>"); //15
5
6
```

```
1  var a = 10;
2  var b = ++a + ++a;
3  document.write("a : ", a, "<br>"); //12
4  document.write("b : ", b, "<br>"); //23
5
6
```

```
1  var a = 10;
2  var b = a++ + a++;
3  document.write("a : ", a, "<br>"); //12
4  document.write("b : ", b, "<br>"); //21
5
6
```

比較運算式

運算的結果得到一個 **布林值**，可作為決策的依據

運算子	說明	範例	結果
<code>></code>	大於	<code>5 > 10</code>	<code>false</code>
<code>>=</code>	大於等於	<code>5 >= 10</code>	<code>false</code>
<code><</code>	小於	<code>5 < 10</code>	<code>true</code>
<code><=</code>	小於等於	<code>5 <= 10</code>	<code>true</code>
<code>==</code>	等於	<code>0 == false</code>	<code>true</code>
<code>====</code>	型值相等	<code>0 === false</code>	<code>false</code>
<code>!=</code>	不等於	<code>5 != 5</code>	<code>false</code>

🔥 注意：

`==` 等於等於：值一樣就可以為 `true`

`====` 等於等於等於：值跟型別都要一樣才為 `true`

👉 參閱 : 5_expression.html 的比較運算式

```
1  document.write('5 > 10 ', 5 > 10, "<br>");           //false
2  document.write('5 >= 10 ', 5 >= 10, "<br>");         //false
3  document.write('5 < 10 ', 5 < 10, "<br>");           //true
4  document.write('5 <= 10 ', 5 <= 10, "<br>");          //true
5  document.write('5 == 10 ', 5 == 10, "<br>");          //false
6  document.write('5 == "5" ', 5 == "5", "<br>");        //true
7  document.write('5 === "5" ', 5 === "5", "<br>");      //false
8  document.write('0 == false ', 0 == false, "<br>");     //true
9  document.write('0 === false ', 0 === false, "<br>");    //false
10 document.write('5 != 10 ', 5 != 10, "<br>");          //true
11 document.write('5 != "5" ', 5 != "5", "<br>");         //false
12 document.write('5 !== "5" ', 5 !== "5", "<br>");       //true
13
14
```

邏輯運算式

透過布林值來判斷邏輯

運算子	說明	範例	結果
&&	and	(x<30 && y>1)	true
	or	(x>5 y>305)	true
!	not	!(x == y)	true

👉 參閱 : 5_expression.html 的邏輯運算式

```
1  var x = 20;
2  var y = 30;
3  document.write(x < 30 && y > 1, "<br>");    //true
4  document.write(x < 30 && y < 1, "<br>");      //false
5  document.write(x > 5 || y > 100, "<br>");     //true
6  document.write(!(x == y), "<br>");            //true
7
8
```

💡 補充 :

常常會搭配之後會講的"流程控制的選擇結構"(EX : if~else)一起使用

指定運算式

要將某個數值或變數 **指定** 給另一個變數

其實就是等號運算子 (=) 來進行「指定」的工作

當計算完畢等號右邊的值後，接著將結果放進等號左邊的變數，這個放進去的動作就是指定

運算子	名稱	範例	說明
=	指定	x = y	$x = y$
+=	加法指定	x += y	$x = x + y$
-=	減法指定	x -= y	$x = x - y$
*=	乘法指定	x *= y	$x = x * y$
/=	除法指定	x /= y	$x = x / y$
%=	餘數指定	x %= y	$x = x \% y$
&=	AND指定	x &= y	$x = x \& y$
=	OR指定	x = y	$x = x y$



參閱 : 5_expression.html 的**指定運算式**

```
1  var x = 20;
2  var y = x;
3  document.write(y, "<br>"); //20
4
5
6  var x = 20;
7  var y = 30;
8  y += x;
9  // y=y+x;
10 document.write(y, "<br>"); //50
11
12 var x = 20;
13 var y = 30;
14 y -= x;
15 document.write(y, "<br>"); //10
16
17 var x = 20;
18 var y = 30;
19 y *= x;
20 document.write(y, "<br>"); //600
21
22 var x = 600;
23 var y = 30;
24 x /= y;
25 document.write(x, "<br>"); //20
26
27
```

三元運算式

根據條件運算式的結果為 true 或 false 來決定回傳 值A 或 值B

語法：

條件運算式 ? 值A(true) : 值B(false)

👉 參閱：5_expression.html 的三元運算式

```
1 | var avg = 66;
2 | document.write(avg >= 60 ? "通過" : "再努力", '<br>');
3 | //通過
4 |
5 |
```

```
1 | var avg = 50;
2 | document.write(avg >= 60 ? "通過" : avg >= 50 ? "差十分" : "差得有點多");
3 | //差十分
4 |
5 |
```

Math 物件

預先定義的 Math 物件具有針對數學常數和函數的屬性和方法。

例如，Math 物件的 PI 屬性有圓周率的值 (3.141...)，你可以在應用程式中如下使用~

方法	說明
abs	絕對值。
sin, cos, tan	標準三角函數；參數以弧度為單位。
acos, asin, atan, atan2	反三角函數；返回值以弧度為單位。
exp, log	指數函數和以 e 為底的自然對數。
ceil	返回大於等於參數的最小整數。
floor	返回小於等於參數的最大整數。
min, max	返回兩個參數中最大的或最小的。
pow	指數函數；第一個參數為底數，第二個為指數。
random	返回介於 0 和 1 之間的隨機數。
round	把參數捨入至最接近的整數。
sqrt	平方根。



參閱 : 5_expression.html 的**Math** 物件

```
document.write("Math.PI : " + Math.PI + "<br>");  
// 3.141592653589793  
  
document.write( "Math.min(11,44,33) : " , Math.min(11,44,33), "<br>");  
// 11  
document.write( "Math.max(11,44,33) : " , Math.max(11,44,33), "<br>");  
// 44  
  
var num = 16.25;  
document.write("Math.ceil(num) : " + Math.ceil(num) + "<br>");  
// 天花板  
// 17  
document.write(" Math.floor(num) : " + Math.floor(num) + "<br>");  
// 地板  
// 16  
  
var num = 16.4;  
document.write(" Math.round(num) : " + Math.round(num) + "<br>");  
// 16  
  
document.write("Math.random() : " + Math.random() + "<br>");  
// 隨機給你一個0~1之間的數  
  
document.write("Math.floor(Math.random()*10) : " + Math.floor(Math.random() * 10) + "<br>")  
// 0~9.9 隨機給 0 ~ 9 (0.1是0) (9.9是9)  
  
document.write("Math.ceil(Math.random()*10) : " + Math.ceil(Math.random() * 10) + "<br>");  
// 0~9.9 隨機給 1 ~ 10 (0.1是1) (9.9是10)
```

流程控制

流程控制結構

結構化程式設計的三個面相:

- 循序結構：依序指令來執行每個階段的步驟
- 選擇結構：根據條件判斷來選擇做不一樣的事情
- 重複結構：又稱迴圈結構，根據需求重覆執行該程式碼

結構化程式設計的三個面相(生活化例子):

- 循序結構：
去看電影：查電影時間>換衣服>出門>搭大眾交通工具>買電影票>等待>上廁所>進場看電影
- 選擇結構：
要不要帶雨傘：如果下雨，我就帶雨傘，沒下雨，我就不帶雨傘
- 重複結構：
時間：星期一星期二星期三星期四星期五星期六星期日，然後又是星期一星期二...

選擇結構：if else

👉 參閱：[7_control.html 的 if~else 句型1](#) / [if~else 句型2](#) / [if~else 其他](#)

語法1

```
1 | if(條件)
2 |   條件成立之敘述;
3 | else
4 |   條件不成立之敘述;
```

```
1 |
2 | var avg = 60;
3 | if (avg >= 60) document.write("通過", "<br>");
4 | else document.write("QQ", "<br>");
5 | //通過
6 |
```

👉 補充：

這種方法不好閱讀，現在比較少人這樣用

語法2

```
1 | if(條件){
2 |   條件成立之敘述;
3 | }else{
4 |   條件不成立之敘述;
5 | }
```

語法3

```
1 | if(條件){  
2 |   條件成立之敘述;  
3 | }
```

語法4

```
1 | if(條件){  
2 |   條件成立之敘述;  
3 | }else if(條件二){  
4 |   條件二成立之敘述;  
5 | }else if(條件三){  
6 |   條件三成立之敘述;  
7 | }
```

語法5

```
1 | if(條件){  
2 |   條件成立之敘述;  
3 | }else if(條件二){  
4 |   條件二成立之敘述;  
5 | }else if(條件三){  
6 |   條件三成立之敘述;  
7 | }else{  
8 |   條件皆不成立之敘述;  
9 | }
```

```
1  
2 var avg = 36;  
3 if (avg >= 60) {  
4     document.write("通過", "<br>");  
5     document.write("合格證書列印中....", "<br>");  
6 } else {  
7     document.write("再讀一學期", "<br>");  
8 }  
// 再讀一學期  
9  
10
```

```
1  
2 var name = "cat";  
3 if (name == "cat") {  
4     document.write("貓<br>");  
5 } else if (name == "dog") {  
6     document.write("狗<br>");  
7 } else {  
8     document.write("不明生物<br>");  
9 }  
// 貓  
10  
11
```

注意!!!請觀察 A 跟 B 的結果狀況

```
1 //==== A:  
2  
3 var year = 2020;  
4 if (year > 1919) {  
5   document.write(">1919<br>");  
6 } else if (year > 2009) {  
7   document.write(">2009<br>");  
8 } else if (year >= 2020) {  
9   document.write(">=2020<br>");  
10 } else {  
11   document.write("when?<br>");  
12 }  
13  
14 // >1919  
15  
16  
17 //==== B:  
18  
19 var year2 = 2020;  
20 if (year2 >= 2020) {  
21   document.write(">=2020<br>");  
22 } else if (year2 > 2009) {  
23   document.write(">2009<br>");  
24 } else if (year2 > 1919) {  
25   document.write(">1919<br>");  
26 } else {  
27   document.write("when?<br>");  
28 }  
29  
30 // >=2020  
31
```



注意：

if~else 會按照先後順序讀程式，所以如果第一個符合就會執行，不會管之後有更適合的答案！所以在程式設計上要記得這樣的問題~~

練習

題目：

我是政府，要來辦一個補助案

這次的補助案有分三類

第一類是補助五千元

第二類補助三千元

第三類補助一千元

另外第一類如果有碩士學歷證明的話就多給一千元

最後幫我顯示出補助款為多少錢

提醒：

請用變數自行假設情況(像是預設是第幾類補助以及有沒有碩士學歷證明等等)

再來看看其他if~else的用法

以下呢，我們可以透過 if(變數) 的方式做控制流程
變數如果不為 0 · null · undefined · false，都會被處理為 true
所以當我們寫成 if(!變數) 就是相反囉!

```
1 var text = "文字";
2 if(text){
3     document.write(text+"<br>");
4 } else{
5     document.write("沒東西"+<br>);
6 }
7 // 文字
8
9
```

```
1 var text1;
2 if (text1) {
3     document.write(text1+"<br>");
4 } else {
5     document.write(text1+"沒東西"+<br>);
6 }
7 // undefined沒東西
8
9
```

```
1 var text2;
2 if (!text2) {
3     document.write(text2+"真的沒東西"+<br>);
4 }
5 // undefined真的沒東西
6
7
```

選擇結構 : switch

語法

```
1 switch (運算式) {
2     case 值 1 :
3         敘述區塊1 ;
4         break;
5     case 値 2 :
6         敘述區塊1 ;
7         break;
8     default :
9         敘述區塊 ;
10    }
11
12
```

👉 參閱 : 7_control.html 的switch

```
1  
2 var game = 1;  
3 switch (game) {  
4     case 1:  
5         document.write(game, "集合啦！動物森友會");  
6         break;  
7     case 2:  
8         document.write(game, "薩爾達傳說");  
9         break;  
10    default:  
11        document.write(game, "沒有遊戲可以玩");  
12    }  
13 // 1集合啦！動物森友會  
14
```

再幫我觀察一下這個狀況

```
1  
2 var game = 1;  
3 switch (game) {  
4     case 1:  
5         document.write(game, "集合啦！動物森友會");  
6         // break; 我把這個關起來了！  
7     case 2:  
8         document.write(game, "薩爾達傳說");  
9         break;  
10    default:  
11        document.write(game, "沒有遊戲可以玩");  
12    }  
13  
14 // 1集合啦！動物森友會1薩爾達傳說  
15
```

迴圈

迴圈會講到:

- for
- while
- do~while

當我們要做迴圈相關的動作，這三個都可以使用

但他們去有語意上的差異

其中 do~while 比較特別

do~while 是不管怎麼樣都先做第一次

試著比較一下 **while** 與 **do~while**的語意

while 是當條件成立我就做什麼

do~while 是做什麼直到條件不成立

for()

語法1

```
1 | for ( var i = 0; i < 6; i++ ) {  
2 |     敘述區塊~~~  
3 | }  
4 |  
5 |  
6 |  
7 |
```

印三次 "I love JavaScript"

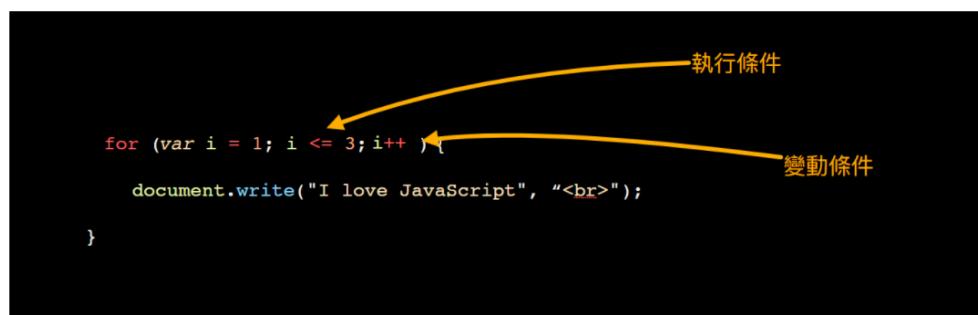
👉 參閱 : 8_control2.html 的 for

```
1 | for ( var i = 1; i <= 3; i++ ) {  
2 |     document.write("I love JavaScript", "<br>");  
3 | }  
4 |  
5 |
```

I love JavaScript
I love JavaScript
I love JavaScript

我們來讀一下這裡的執行步驟:

1. 初始值 : var i = 1;
2. 執行條件 : i 有沒有小於 3
3. 執行敘述區塊 : 小於 3 我們來印出 I love JavaScript
4. 變動條件 : i++ (執行完敘述區塊才變動喔!)



執行完敘述區塊才變動?

👉 參閱 : 8_control2.html 的for 印出數字1~10

```
1 | 
2 | for (var i = 1; i <= 10; i++) {  
3 |   document.write(i, "<br>");  
4 | }  
5 | 
```

```
1 | 
2 | for (var i = 1; i <= 10; i++) {  
3 |   document.write(i, "<br>");  
4 | }  
5 |  
6 | document.write(i, "<br>"); // 這裡會變成 11 呀!  
7 | 
```

印出1,2、3,...,9,10

👉 參閱 : 8_control2.html 的印出1,2、3,...,9,10/印出1,2、3,...,9,10(方法二)

```
1 | 
2 | for (var i = 1; i <= 10; i++) {  
3 |   if (i < 10) {  
4 |     document.write(i, ",");  
5 |     // 1,2,3,4,5,6,7,8,9,  
6 |     //因為10後面沒有逗點嘛!  
7 |   } else {  
8 |     document.write(i);  
9 |     //補上10  
10 |   }  
11 | }  
12 | 
```

```
1 | 
2 | for (var i = 1; i <= 9; i++) {  
3 |   document.write(i, ",");  
4 |   // 1,2,3,4,5,6,7,8,9,  
5 | }  
6 | document.write(i, "<br>");  
7 | //補上10  
8 | 
```

計算1加2加3加到...50的總和

👉 參閱 : 8_control2.html 的計算1加2加3加到...50的總和

看到總和!我們需要一個變數來幫我 $+1+2+3\sim$

```
1 var total = 0;
2 for (var i = 1; i <= 50; i++) {
3     total = total + i;
4     // total += i;
5 }
6 document.write("總和 : ", total, "<br>");
7 // 總和 : 1275
8
9
```

印出1到50間3的倍數及計算其總和

👉 參閱 : 8_control2.html 的印出1到50間3的倍數及計算其總和/印出1到50間3的倍數及計算其總和(方法二)

```
1
2 var total = 0;
3 for (var i = 1; i <= 50; i++) {
4     if (i % 3 == 0) {
5         document.write(i, "<br>");
6         total += i;
7     }
8 }
9 document.write("總和 : ", total, "<br>");
10
```

```
1
2 var total = 0;
3 for (var i = 3; i <= 50; i += 3) {
4     document.write(i, "<br>");
5     total += i;
6 }
7 document.write("總和 : ", total, "<br>");
8
```

設一個數字，求1到此數之總和，及其奇數和與偶數和

思考一下我們需要幾個變數呢？

👉 參閱 : 8_control2.html 的設一個數字，求1到此數之總和，及其奇數和與偶數和

```
1  var num = 10; // 假設的數字
2  var odd = 0;
3  var even = 0;
4  var total = 0;
5  for (var i = 0; i <= num; i++) {
6    i % 2 === 0 ? (even += i) : (odd += i);
7    total += i;
8  }
9  document.write(`數字: ${num} | 總和: ${total} | 奇數和: ${odd} | 偶數和: ${even}<br>`);
10
11
```

我們要來抽獎十次，獎金落在**1000**元以內，再把獎金加總

👉 參閱 : 8_control2.html 的我們要來抽獎十次，獎金落在**1000**元以內，再把獎金加總

```
1  var total = 0;
2  for (var i = 0; i < 10; i++) {
3    var money = Math.floor(Math.random() * 1000 + 1);
4    total += money;
5    document.write(`第${i + 1}次: ${money}<br>`);
6  }
7  document.write(`總獎金: ${total} 元<br>`);
```

for() 巢狀迴圈

迴圈中又有迴圈的意思

巢狀迴圈

👉 參閱 : 8_control2.html 的巢狀迴圈

```
1  for (var i = 1; i <= 3; i++) {
2      document.write("第一層", i, "<br>");
3      for (var k = 1; k <= 3; k++) {
4          document.write("&nbsp;&nbsp;&nbsp;第二層", k, "<br>");
5      }
6  }
7
8 }
```

```
第一層1
第二層1
第二層2
第二層3
第一層2
第二層1
第二層2
第二層3
第一層3
第二層1
第二層2
第二層3
```

表格搭配表情符號

```
1  document.write("<table>");
2
3
4  for (var i = 1; i <= 3; i++) {
5
6      document.write("<tr>");
7
8      for (var k = 1; k <= 3; k++) {
9          document.write("<td>", "ヽ(・ω・o)", "</td>");
10     }
11
12     document.write("</tr>");
13
14 }
15
16 document.write("</table>");
17 }
```

練習

題目：九九乘法表

1*1=1	2*1=2	3*1=3	4*1=4	5*1=5	6*1=6	7*1=7	8*1=8	9*1=9
1*2=2	2*2=4	3*2=6	4*2=8	5*2=10	6*2=12	7*2=14	8*2=16	9*2=18
1*3=3	2*3=6	3*3=9	4*3=12	5*3=15	6*3=18	7*3=21	8*3=24	9*3=27
1*4=4	2*4=8	3*4=12	4*4=16	5*4=20	6*4=24	7*4=28	8*4=32	9*4=36
1*5=5	2*5=10	3*5=15	4*5=20	5*5=25	6*5=30	7*5=35	8*5=40	9*5=45
1*6=6	2*6=12	3*6=18	4*6=24	5*6=30	6*6=36	7*6=42	8*6=48	9*6=54
1*7=7	2*7=14	3*7=21	4*7=28	5*7=35	6*7=42	7*7=49	8*7=56	9*7=63
1*8=8	2*8=16	3*8=24	4*8=32	5*8=40	6*8=48	7*8=56	8*8=64	9*8=72
1*9=9	2*9=18	3*9=27	4*9=36	5*9=45	6*9=54	7*9=63	8*9=72	9*9=81

題目：印星星

小到大

*
**

大到小

**
*

while()

語法

```
1  while (條件) {  
2      敘述區塊  
3  }  
4  
5  
6  
7
```

👉 參閱 : 8_control2.html 的**while()**

```
1  var i = 1;//初始值  
2  while (i <= 3) {//條件  
3      document.write("I love JavaScript", "<br>");  
4      i++;//結束時變動  
5  }  
6  
7
```

印出1、2、3、...、9、10

👉 參閱 : 8_control2.html 的**while()**

```
1  var i = 1;  
2  while (i <= 9) {  
3      document.write(i, "、");  
4      i++;  
5  }  
6  document.write(i, "<br>");  
7  
8
```

計算1加2加3加到...50的總和

👉 參閱 : 8_control2.html 的計算1加2加3加到...50的總和

```
1  var total = 0;  
2  var i = 1;  
3  while (i <= 50) {  
4      total += i;  
5      i++;  
6  }  
7  document.write("總和 : ", total, "<br>");  
8  
9
```

印出1到50間3的倍數及計算其總和

👉 參閱 : 8_control2.html 的印出1到50間3的倍數及計算其總和

```
1  var total = 0;
2  var i = 3;
3  while (i <= 50) {
4      document.write(i, "<br>");
5      total = total + i;
6      i += 3;
7  }
8  document.write("總和 : ", total, "<br>");
9
10
```

do~while()

do~while不管條件成不成立都會先做一次

語法

```
1
2  do {
3      敘述區塊
4  }while ( 條件 ) {
5      敘述區塊
6  }
7
```

👉 參閱 : 8_control2.html 的do ...while()

```
1  var i = 1;// 初始值
2  do {
3      document.write("I love JavaScript", "<br>");
4      i++;//結束時變動
5  } while (i <= 3); // 條件
6
7
```

印出1、2、3、...、9、10

👉 參閱 : 8_control2.html 的1、2、3、...、9、10

```
1
2  var i = 1;
3  do {
4      document.write(i, " ");
5      i++;
6  } while (i <= 9);
7  document.write(i, "<br>");
8
```

計算1加2加3加到...50的總和

👉 參閱 : 8_control2.html 的計算1加2加3加到...50的總和

```
1 | var total = 0;
2 | var i = 1;
3 | do {
4 |   total = total + i;
5 |   i++;
6 | } while (i <= 50);
7 | document.write("總和 : ", total, "<br>");
8 |
9 |
```

印出1到50間3的倍數及計算其總和

👉 參閱 : 8_control2.html 的印出1到50間3的倍數及計算其總和

```
1 | var total = 0;
2 | var i = 3;
3 | do {
4 |   document.write(i, "<br>");
5 |   total = total + i;
6 |   i = i + 3;
7 | } while (i <= 50);
8 | document.write("總和 : ", total, "<br>");
9 |
```

```
1 | var total = 0;
2 | var i = 1; //3
3 | do {
4 |   if (i % 3 == 0) {
5 |     document.write(i, "<br>");
6 |     total = total + i;
7 |   }
8 |
9 |   i++;
10 | } while (i <= 50);
11 | document.write("總和 : ", total, "<br>");
12 |
```

練習

題目 : do...while摸彩金

有11顆彩球, 彩球面額為0-10之間, 若摸到的彩球不為0, 則可繼續摸彩, 若摸到的彩球為0, 則停止摸彩, 並計算其摸彩次數及彩金總金額(單位:佰元)(假設抽到1就是100元)

break & continue

- break : 會立刻跳出迴圈以及退出 switch
可用在重複結構的所有迴圈句型中
- continue : 不是退出一個迴圈，而是在迴圈的下一次新迭代開始執行
只能用在 while 語句、do/while 語句、for語句、或者for/in 語句的迴圈體內

👉 參閱 : 8_control2.html 的**break & continue** 使用

break

```
1  for (var i = 0; i < 10; i++) {  
2    if (i == 4) {  
3      break;  
4    }  
5    document.write(i);  
6  }  
7  
8
```

0123

可以看出來4與4以後的就都不會繼續了

continue

```
1  for (var i = 0; i < 10; i++) {  
2    if (i == 4) {  
3      continue;  
4    }  
5    document.write(i);  
6  }  
7  
8
```

012356789

可以看出來除了4不見，其他的有繼續執行

練習

題目 :

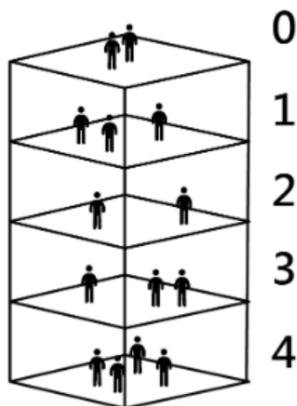
```
var time = "8000";  
document.write(time + "秒=" + ?? + "小時" + ?? + "分" + ?? + "秒");
```

陣列

- 陣列是個 **物件**
- 使用時機：
當有一堆資料，他們所代表的意義或是被拿出來處理的過程相似
- 例如：班上有30位同學，我們要來加總他們的這次的國英數考試成績
我們就可以用**陣列**來存放每個人各科加總後的成績

一維陣列的建立

array



語法

- var 陣列名稱 = new Array;
- var 陣列名稱 = new Array(元素個數);
- var 陣列名稱 = new Array(陣列元素一, 陣列元素二, 陣列元素三.....);
- var 陣列名稱 = [陣列元素一, 陣列元素二, 陣列元素三.....];

👉 參閱 : 9_array.html 的**Array**

先宣告空陣列，再把值放入

```
1 | var arr = new Array(); // 宣告變數arr 他指定為一個空陣列
2 | arr[0] = "A"; //arr[0] 指的是 arr陣列中的"索引值""第0個"的值為A
3 | arr[1] = "B";
4 | arr[2] = "C";
5 | document.write(arr, "<br>");
```

A,B,C

在陣列中每一值都會對應一個索引值

索引值是由 0 開始為第一個

以上面的作為例子來看

陣列 arr 現在是有 A,B,C 三個值

對應的索引值 0,1,2

當我們用這個陣列的變數名，也就是arr

再搭配 中括號[]，中括號放入索引值

因此

arr[0] 就是 A

arr[1] 就是 B

arr[2] 就是 C

我先宣告陣列總共會有多少值 new Array(陣列有幾個值)

```
1 var arr2 = new Array(6); // 這個列我要放六個值
2 arr2[0] = "D";
3 arr2[1] = "E";
4 arr2[2] = "F";
5 document.write(arr2, "<br>");
6
7
```

D,E,F,,,

有看到逗號嗎?!!!

因為你說好你的陣列有六個

結果沒給他!!!他只好空的給你囉~~

常用的方法new Array(值1,值2...)

```
1
2 var arr3 = new Array("G", "H", "I");
3 document.write(arr3, "<br>");
4
```

常用的方法[值1,值2...]

```
1
2 var arr4 = ["J", "K", "L"];
3 document.write(arr4, "<br>");
4
```

小心啊!!!

```
1  var arr5 = new Array(6);
2  document.write(arr5, "<br>");
3  // 如果你的值只有一個，而且型別還是數值，這時候要小心了！
4  // 這樣寫會被以為是跟陣列說你要請她幫你預留六個值
5
6
```

~~~~~

## 取陣列長度與全部值的方法

先來看看陣列的索引~

```
1  var arr=[11,22,33];
2
3
4  arr[0] 是 11
5  arr[1] 是 22
6  arr[2] 是 33
7
```

有沒有一種迴圈的感覺~~~~~

因此!

我們可以透過以下方法來幫我們取值

- `for(var i = 0; i < arr.length ; i++){};`
- `for(var i in arr){}`
- `for(var data of arr){}`

### arr.length for取值

👉 參閱 : 9\_array.html 的 **arr.length for取值**

```
1  var arr = [11, 66, 33, 55, 44, 22];
2  document.write("arr有幾個值 : ", arr.length, "<br>");
3
4
5  for (var i = 0; i < arr.length; i++) {
6      document.write(arr[i], "<br>");
7  }
8
```

### **for(i in arr)**

大部分我們知道陣列有多少值~或是值有哪些的時候可以用for搭被arr.length去做迴圈的動作，但當我們遇到陣列的索引值有跳號的時候該怎麼辦呢???

一起來看一下以下的情況~

👉 參閱 : 9\_array.html 的**for(i in arr)**

```
1  var arr = new Array(11, 22, 33);
2  arr[7] = 100;
3
4  for (var i in arr) {
5      document.write(` ${i} : ${arr[i]}<br>`);
6  }
7
8
```

👉 補充 :

for(i in arr) 他會自己挑有值的!!!沒值就是會跳過去

### **for(data of arr)**

👉 參閱 : 9\_array.html 的**for(data of arr)**

```
1  var arr = new Array(11, 22, 33);
2  arr[7] = 100;
3
4  document.write("arr有幾個值 : ", arr.length, "<br>");
5
6  for (var data of arr) {
7      document.write(`${data}<br>`);
8  }
9
10
```

👉 補充 :

for(data of arr) 有說少陣列長就用跑幾次

## 陣列物件

陣列是物件嘛~所以陣列會有陣列 物件.方法()

| 方法                           | 說明                                            |
|------------------------------|-----------------------------------------------|
| sort()                       | 排序                                            |
| toString()                   | 將陣列中的元素以逗點結合起來，傳回一個字串                         |
| reverse()                    | 將陣列中的資料順序反轉（會改變陣列內容）                          |
| join(字元)                     | 將陣列中的元素以特定的符號結合起來，傳回一個字串                      |
| concat(參數...)                | 將陣列結合成一個新的陣列，並傳回陣列長度                          |
| push(資料)                     | 將新的元素加到陣列的後便，並回傳陣列長度                          |
| pop()                        | 傳回陣列中的最後一個元素，並移除該元素                           |
| slice(start, end)            | 傳回索引值start處到end-1處的子陣列<br>( 不會從原陣列中移除 )       |
| splice(start,n,ele1,ele2...) | 傳回索引值start處後的n個元素（會被移除）並插入ele1, ele2...       |
| shift()                      | 傳回陣列中第一個元素，並移除該元素                             |
| unshift(資料)                  | 將新的元素加入陣列的最前面                                 |
| indexOf(data, start)         | 自索引start處找出data在array中的位置索引值<br>( 如果找不到會傳-1 ) |

## 手動陣列轉字串

👉 參閱 : 9\_array.html 的手動陣列轉字串

```
1  var arr = new Array(11, 22, 33, 44, 55, 66);
2  var str = "";
3  var length = arr.length;
4
5  for (var i = 0; i < length - 1; i++) {
6      str += arr[i] + ",";
7  }
8  str += arr[i];
9
10 document.write("原本陣列: ", arr, "<br>");
11 document.write("轉成字串: ", str, "<br>");
12 document.write("型別檢查: ", typeof str, "<br>");
13
14
```

## 陣列轉字串: 使用**toString()**

剛剛應該會發現!我直接印出陣列跟變成字串的陣列長得一樣啊!!!  
是因為陣列是一個物件，要輸出時，會自動呼叫**toString()**

👉 參閱 : 9\_array.html 的陣列轉字串: 使用**toString()**

```
1 | var arr = new Array(11, 22, 33, 44, 55, 66);
2 |
3 | document.write("轉成字串: ", arr.toString(), "<br>"); 
4 | document.write("typeof(arr.toString()) : ", typeof arr.toString(),"<br>"); 
5 |
6 |
```

## 陣列轉字串: **join()**

👉 參閱 : 9\_array.html 的陣列轉字串: **join()**

```
1 | var arr = new Array(11, 22, 33, 44, 55, 66);
2 |
3 | document.write("轉成字串: ", arr.join("@@"), "<br>"); 
4 | document.write("typeof(arr.join('@@')) : ", typeof arr.join("@@"),"<br>"); 
5 |
6 |
```

將資料放到陣列最後面: 自己做~ + **push()**

👉 參閱 : 9\_array.html 的將資料放到陣列最後面: 自己做~ + **push()**

```
1 | var arr = new Array(11, 22, 33);
2 | // 索引值 0,1,2
3 | // 陣列長 arr.length = 3
4 |
5 | arr[arr.length] = 100; //arr[3] = 100的意思
6 |
7 | // 這時的陣列長 arr.length = 4 囉!
8 | arr[arr.length] = 1000;//arr[4] = 1000的意思
9 | document.write(arr, "<br>"); 
10 |
11 | arr.push(555); // 直接放到陣列的最後一個
12 | document.write(arr, "<br>"); 
13 |
14 |
```

將資料放到陣列中的特定位置: 自己做~

👉 參閱 : 9\_array.html 的將將資料放到陣列中的特定位置: 自己做

```
1 // 將data放到陣列中的索引值為index的位置
2
3 var arr = new Array(11, 22, 33, 44, 55, 66, 77);
4 var index = 5;
5 var data = "我在這";
6 // 所以我要將"我在這"塞到陣列索引值5的位置
7
8 // 我要將arr陣列索引值為5之後的往後移
9 // 也就是原本最後的值arr[6]往後移成arr[7]
10 // 再將arr[5]往後移成arr[6]
11 // 到此 arr[5]跟arr[6]值一樣的喔
12 // 11, 22, 33, 44, 55, 66, 66, 77
13 // 所以再將arr[5]=data就可以將"我在這"放在索引值5了!
14 // 程式開始處理囉~
15 for (var i = arr.length; i > index; i--) {
16     arr[i] = arr[i-1];
17 }
18 arr[index] = data;
19
20 document.write(arr, "<br>");
21
22
```

找到資料(33)在陣列中第一次出現的位置: 自己做~

👉 參閱 : 9\_array.html 的找到資料(33)在陣列中第一次出現的位置: 自己做

```
1 //若找到， 請顯示其索引值，若找不到請顯示找不到的訊息
2
3 var arr = new Array(11, 22, 33, 44, 55, 66, 77);
4 var data = 33;
5 var length = arr.length;
6
7 for (var i = 0; i < length; i++) {
8     if (arr[i] == data) {
9         document.write(`${data}在陣列中的索引值為${i}<br>`);
10        break;
11    }
12 }
13
14 if (i == length) {
15     document.write("找不到", "<br>");
16 }
17
18
```

找到資料(33)在陣列中第一次出現的位置: 自己做~ (方法二)

👉 參閱 : 9\_array.html 的找到資料(33)在陣列中第一次出現的位置: 自己做 (方法二)

```
1 //若找到，請顯示其索引值，若找不到請顯示找不到的訊息
2
3
4 var arr = new Array(11, 22, 33, 44, 55, 66, 77);
5 var data = 33;
6 var length = arr.length;
7 var found = false;
8
9 for (var i = 0; i < length; i++) {
10   if (arr[i] == data) {
11     found = true;
12     document.write(`${data}在陣列中的索引值為${i}<br>`);
13     break;
14   }
15 }
16
17 if (found == false) {
18   document.write("找不到", "<br>");
19 }
```

### indexOf()

👉 參閱 : 9\_array.html 的找到資料(33)在陣列中第一次出現的位置: indexOf()

```
1
2 var arr = new Array(11, 22, 33, 44, 55, 66, 77, 22, 33);
3 var data = 33;
4 var index = arr.indexOf(data); // 找到第一個33
5 document.write(`${data}在陣列中的索引值為${index}<br>`);
6
7 index = arr.indexOf(data, 5); // 索引值5之後有沒有33
8 document.write(`${data}在陣列中的索引值為${index}<br>`);
9
10 data = 333;
11 index = arr.indexOf(data); // 找不到會傳回 -1;
12 document.write(`${data}在陣列中的索引值為${index}<br>`);
13
```

👉 補充 :

arr.indexOf(X) 找不到會傳回 -1 找得到會回傳索引值

## **push(), pop()**

👉 參閱 : 9\_array.html 的**push(), pop()**

```
1  var arr = new Array(11, 22, 33);
2  arr.push(1000);
3  arr.push(2000);
4  document.write(arr, "<br>");
5
6
7  var data = arr.pop();// 取走最後的資料
8  document.write(arr, "<br>");
9  document.write(`pop出資料${data}` , "<br>");
```

## **unshift(), shift()**

👉 參閱 : 9\_array.html 的**unshift(), shift()**

```
1  var arr = new Array(11, 22, 33);
2  arr.unshift(1000);// 塞進陣列最前面
3  arr.unshift(2000);
4  document.write(arr, "<br>");
5
6
7  var data = arr.shift();// 取走最前面的資料
8  document.write(arr, "<br>");
9  document.write(`shift出資料${data}` , "<br>");
```

## **slice(start, end)**

👉 參閱 : 9\_array.html 的**slice(start, end)**

```
1  var arr = new Array(11, 22, 33, 44, 55, 66, 77);
2  var arr2 = arr.slice(2, 5); // 從2開始 取到5之前
3
4
5  document.write("arr : " , arr, "<br>"); 
6  document.write("arr2 : " , arr2, "<br>");
```

### **splice(start, length, ele1)**

👉 參閱 : 9\_array.html 的**splice(start, length, ele1)**

```

1  var arr = new Array(11, 22, 33, 44, 55, 66, 77, 88, 99);
2  var arr2 = arr.splice(2, 5, 1000, 2000);
3  // 從 2 開始 共 5 個都要
4  // 從原 2 這裡補上 1000 與 2000
5
6
7  document.write("arr : ", arr, "<br>");
8  document.write("arr2 : ", arr2, "<br>");
9

```

👉 補充 :

slice 不會真的動到原本的陣列  
splice 會動到原本的陣列

### **reverse()**

將陣列順序反過來

👉 參閱 : 9\_array.html 的**reverse()**

```

1  var arr = new Array(11, 22, 88, 44, 55, 66, 77, 33, 99);
2  document.write("arr : ", arr, "<br>");
3
4
5  arr.reverse();
6  document.write("arr : ", arr, "<br>");
7

```

### **concat()**

將陣列結合成一個新的陣列

👉 參閱 : 9\_array.html 的**concat()**

```

1  var arr = new Array(11, 22, 33);
2  var arr2 = new Array(1, 2, 3);
3  var arr3 = arr2.concat(arr);
4
5  document.write("arr : ", arr, "<br>");
6  document.write("arr2 : ", arr2, "<br>");
7  document.write("arr3 : ", arr3, "<br>");
8

```

## 使用sort比大小

👉 參閱 : 9\_array.html 的使用sort

```
1 var arr = [40, 20, 50, 70, 30, 80, 7];
2 arr.sort();
3 document.write(arr, "<br>");
4
5
```

## sort(), compareFunction 由小到大

👉 參閱 : 9\_array.html 的sort(), compareFunction 由小到大

```
1
2 function myCompareAsc(a, b) {
3     //a:20, b:40
4     return a - b; //40-20-->20 --> 20,40 (由小到大)
5 }
6
7 var arr = [40, 20, 50, 70, 30, 80, 7];
8 arr.sort(myCompareAsc);
9 document.write(arr, "<br>");
10
11
12 // arr.sort(function myCompareAsc(a, b) {
13 //     if (a > b) {
14 //         return 1; // 正數時，後面的數放在前面
15 //     } else {
16 //         return -1 // 負數時，前面的數放在前面
17 //     }
18 // });
19
```

## sort(), compareFunction 由大到小

👉 參閱 : 9\_array.html 的 sort(), compareFunction 由大到小

```
1  function myCompareDesc(a, b) {
2      return b - a;
3  }
4
5
6  var arr = [40, 20, 50, 70, 30, 80, 7];
7  arr.sort(myCompareDesc);
8  document.write(arr, "<br>");
9
10
11 // arr.sort(function myCompareAsc(a, b) {
12 //     if (a < b) {
13 //         return 1; // 正數時，後面的數放在前面
14 //     } else {
15 //         return -1 // 負數時，前面的數放在前面
16 //     }
17 // });

18
```

👉 延伸閱讀 :

- <https://realdennis.medium.com/javascript-從array的sort方法-聊到各家瀏覽器的實作算法-c23a335b1b80>
- <https://www.youtube.com/watch?v=lyZQPjUT5B4&list=PLa3md2sAwEVo5RfnfDetgWAVOa1HiQ8-Y>

## 練習

題目 : 請問陣列長度 ?

```
var arr = [];
arr.push(1);
arr.push(2);
document.write(arr.length);
```

題目 : 請問陣列長度 ?

```
var arr = new Array(6);
arr.push(1);
arr.push(2);
document.write(arr.length);
```

## 二維陣列

就是陣列的值是陣列

```
1 | var ary = new Array(3);
2 | ary[0] = new Array( 1, 2, 3, 4);
3 | ary[1] = new Array( 11, 12, 13, 14);
4 | ary[2] = new Array( 21, 22, 23, 24);
5 |
6 |
```

```
1 | var arr = [[1,2,3],[11,12,13],[21,22,23]];
2 |
3 |
```

👉 參閱 : 10\_array2dim.html 的2維陣列初步認識

```
1 |
2 | var arr = [[1, 2, 3], [11, 12, 13], [21, 22, 23]];
3 | document.write(arr[0], '<br>');
4 | document.write(arr[1], '<br>');
5 | document.write(arr[2], '<br>');
6 | document.write(arr[0][0], '<br>');
7 | document.write(arr[1][0], '<br>');
8 | document.write(arr[2][0], '<br>');
9 |
```

我們可以用巢狀迴圈的方式處理二維陣列

```
1 |
2 | //取得二維陣列中每一個元素
3 |
4 | for(var i=0; i<=2; i++){
5 |   for(var j=0; j<=3; j++){
6 |     //陣列中第i列第j行的元素
7 |     document.write(arr[ i ][ j ]);
8 |   }
9 |   document.write("<br>");
10 |
11 | }
```

👉 參閱 : 10\_array2dim.html 的2維陣列初步與for搭配

```
1 var arr = [[1, 2, 3], [11, 12, 13], [21, 22, 23]];
2
3 for (var i = 0; i < arr.length; i++) {
4     for (var j = 0; j < arr[i].length; j++) {
5         if (j < arr[i].length - 1) {
6             document.write(arr[i][j], ",");
7         } else {
8             document.write(arr[i][j]);
9         }
10    }
11 }
12 }
13 }
```

還記得我們寫九九乘法表嗎？

我們還用表格將陣列的值一個一個放進去看看

```
1 var arr = [[1, 2, 3, 4], [11, 12, 13, 14], [21, 22, 23, 24]];
2
3 document.write("<table>");
4
5 for (var i = 0; i < 3; i++) {
6     document.write("<tr>");
7
8     for (j = 0; j < 4; j++) {
9         document.write("<td>", arr[i][j], "</td>");
10    }
11
12    document.write("</tr>");
13 }
14
15 document.write("</table>");
16
17 }
```

## 函式 (function)

函式是構成 JavaScript 的基本要素之一

一個函式本身就是一段 JavaScript 程序—包含用於執行某一個任務或計算的語法  
對於重複執行的內容，可使用 function 定義函式，讓程式的每個部份都可以共用

分為：

- 內建函式 (build-in functions)
- 自訂函式 (user-defined functions)

將一段具有某種特定功能的程式碼另外包裝成獨立的函式之優點：

- 減少程式碼的重複性
- 增進程式碼再利用
- 可讀性增加
- 易於維護

### 內建函式

一般內建

| -----      | -----                                                                |
|------------|----------------------------------------------------------------------|
| parseInt() | 將輸入的字串轉成整數                                                           |
| isNaN()    | 檢查參數是否非數字 ex:'123' -> false 2021/12/12 -> true (false代表是數字，true代表不是) |
| eval()     | 計算字串，並執行JavaScript 代碼                                                |



參閱：11\_function.html 的 eval()

```
1  var str = "var a=10; var b=20";
2  eval(str);
3  document.write("a : ", a, "<br>");
4
5
```

👉 參閱 : 11\_function.html 的 **isNaN()**

```
1 document.write("isNaN(326) : ", isNaN(326), "<br>"); //false
2 document.write("isNaN('326') : ", isNaN("326"), "<br>"); //false
3 document.write("isNaN('abcd') : ", isNaN("abcd"), "<br>"); //true
4 document.write("isNaN('2015/12/12') : ", isNaN("2015/12/12"), "<br>"); //true
5
6
```

### 與物件結合

又稱物件的方法，像是前面提到的 **Math** 物件

## 自訂函式

**function** 是函數的關鍵字

程式讀到 **function** 就之後等等要讀的是一個函式

### 函式的基本語法

```
1 // 創建函式囉! !
2 function 函式名字(){
3
4     要做些什麼事情呢.....
5
6 }
7
8 函式名字(); // 呼叫函式執行
9
10
```

沒有參數，也沒有傳回值

👉 參閱 : 11\_function.html 的 沒有參數，也沒有傳回值

```
1
2 function sayHello() {
3     document.write("Hello~~~", "<br>");
4 }
5
6 sayHello();
7
```

有參數, 沒有傳回值

👉 參閱 : 11\_function.html 的有參數, 沒有傳回值

```
1 | 
2 | function sayHelloToSomeone(name) {
3 |   document.write(`Hello, ${name}, 您好~~~`, "<br>");
4 | }
5 | sayHelloToSomeone("文文");
6 | 
```

有參數, 有傳回值

function搭配 return 的使用方式

👉 參閱 : 11\_function.html 的有參數, 有傳回值/有參數, 有傳回值2

```
1 | 
2 | function needyourname(name) {
3 |   return 'hi~' + name;
4 | }
5 | document.write(needyourname("文文"), "<br>");
6 | 
```

```
1 | 
2 | function test(num) {
3 |   let all = num * 10;
4 |   return all;
5 | }
6 | document.write(test(5));
7 | 
```

題目 :

還記得我們有算過8000秒等於幾小時幾分鐘幾秒嗎？

我們改用function return 的方式呈現 ~

```
1 | function time(t) {
2 |   .
3 |   .
4 |   .
5 |   return ~ ~ ~
6 | }
7 | document.write(time("8000"));
8 | 
```

## 參數給預設值：ES6

我們在設計程式時，要開始思考一下程式有沒有可能會有突發狀況???  
因此，假設我們像剛剛那樣需要傳一個值給function，但...我的值忘記傳了！  
來看！

```
1 | function showMsg(msg) {
2 |   document.write(msg, "<br>"); //undefined
3 |
4 |
5 |
6 | showMsg();
7 |
```

沒接到值會 **undefined**

為了避免以上的問題，我們可以透過參數給 **預設值** 的方式

👉 參閱：[11\\_function.html](#) 的參數給預設值：ES6

```
1 | function showMsg(msg = "Hi~") {
2 |   document.write(msg, "<br>");
3 |
4 |
5 | showMsg("早安~");
6 | showMsg();
7 |
```

## 函式宣告 ( Function Declaration )

以上的函式的表現方式呢～我們可以稱它為 **函式宣告** 或 **具名函式**  
具名函式是以 **function statement** 的方式被建立，有 **hoist** 現象  
還記得～ **var** 也有 **hoist** 現象嗎？

所以～可以讓程式先讀到呼叫，再讀到 **function**  
方便～～～

```
1 | sayHellohihihi();
2 |
3 |
4 | function sayHellohihihi() {
5 |   document.write("不要這樣哈囉吧～<br>");
6 |
7 | }
```

## 函式運算式 ( Function Expressions )

這類的通常會是 **匿名函式**

在函式運算式中，函式要等待解釋器執行到該行敘述時才會執行，這表示在解讀到他之前，我們都無法去呼叫他

### 匿名函式

👉 參閱 : 11\_function.html 的函式運算式

```
1 // 貫名函式
2 function say_hi() {
3     document.write("哈囉~<br>");
4 }
5 say_hi();
6
7
8 // 匿名函式
9 let say_hi_hi = function() {
10     document.write("哈囉哈~<br>");
11 };
12 say_hi_hi();
13
14
15
```

這邊我們可以看到，匿名函式我們會宣告一個變數，這個變數透過等於(=)裝進一個 **function**，只是這個**function**，就可以不用寫函式名字了，現在這個變數就可以指向這個 **function**囉~

📢 补充 :

那我想要在函式運算式中的那個 **function** 後面加上一個名字是可以的嗎？

答案是~可以

但這個名字只在「自己函式的區塊內」有效

有興趣的人可以再去找資料，我們這裡不深談這個

只是希望大家不要誤會他不能再加上名字

接著我們來看~剛剛說"具名函式有hoist現象"，"函式運算式中，函式要等待解釋器執行到該行敘述時才會執行，這表示在解讀到他之前，我們都無法去呼叫他"

👉 參閱 : 11\_function.html 的函式運算式2

```
1 // sayHellohi();
2 // Uncaught SyntaxError: Identifier 'sayHellohi' has already been declared
3
4
5 let sayHellohi = function() {
6   document.write("哈囉~<br>");
7 }
8 sayHellohi();
9
```

如果要傳參數可以如下：

👉 參閱 : 11\_function.html 的函式運算式

```
1
2 let hi_u_hi = function (name) {
3   document.write("哈囉~" + name + "<br>");
4 }
5 hi_u_hi("Sara");
6
```

## 箭頭函式 ( Arrow Functions )

箭頭函式(Arrow Functions)是ES6標準中，最受歡迎的一種新語法。

它會受歡迎的原因是好處多多，而且沒有什麼副作用或壞處，只要注意在某些情況下不要使用過頭就行了。

大致上有以下幾點：

- 語法簡單
- 可以讓程式碼的可閱讀性提高

👉 參閱 : 11\_function.html 的函式運算式

```
1  let myFunction = () => {
2    document.write("換個方式哈囉~<br>");
3  };
4  myFunction();
5
6  let myU = (name) => {
7    document.write("哈囉~" + name + "<br>");
8  };
9  myU("Fan");
10
11
```

🔥 注意：

因為時間關係，我們不會深討論箭頭函式～

延伸閱讀:[https://eyesofkids.gitbooks.io/javascript-start-from-es6/content/part4/arrow\\_function.html](https://eyesofkids.gitbooks.io/javascript-start-from-es6/content/part4/arrow_function.html)

## 函式特性

- 本身是一個物件
- 可以重覆被定義
- 不支援overloading (覆蓋問題)
- 要避免跟瀏覽器的全域方法取同個名字，會覆寫掉喔QQ
- 可以用 arguments 來取得呼叫此函式時的所有實際引數資料
- 可以設定參數預設值

### 參數 (arguments) and 引數 (parameter)

```
function sum(a, b, c){  
    let total = 0;  
    for(var i = 0; i < arguments.length ;i++){  
        total += arguments[i];  
    }  
    return total;  
}  
document.write(`${sum(2,4,6,8,10,12)}<br>`);
```

參數 parameter  
引數 arguments

👉 參閱 : 11\_function.html 的 arguments

```
1 //具名函式  
2  
3 function sum(a, b, c) {  
4     let total = 0;  
5     for (var i = 0; i < arguments.length; i++) {  
6         total += arguments[i];  
7     }  
8     return total;  
9 }  
10  
11 document.write(`${sum(2, 4, 6, 8, 10, 12)}<br>`); //42  
12  
13  
14 //函式運算式  
15  
16 let sum2 = function (a, b, c) {  
17     let total = 0;  
18     for (var i = 0; i < arguments.length; i++) {  
19         total += arguments[i];  
20     }  
21     return total;  
22 };  
23  
24 document.write(`${sum2(2, 4, 6, 8, 10, 12)}<br>`); //42  
25  
26  
27
```

🔥 注意 :

ES6 的箭頭函式 (Arrow Function) 沒有提供 arguments

## callback function 回調函式

- 把函式當作另一個函式的參數，透過另一個函式來呼叫它
- Callback function 跟一般的函式沒什麼不同呢？差別在於被呼叫執行的時機
- 可以控制多個函式間執行的順序

👉 參閱：[11\\_function.html 的callback function](#)

### callback function

```
1
2  function A_function(x) {
3    document.write("這是A<br>");
4    x();
5    //x參數等於B_function;
6    //因此 x(); 就是等於 B_function();
7  };
8
9  function B_function() {
10   document.write("這是B<br>");
11 };
12
13 A_function(B_function);
14
```

### 箭頭函式的樣子

```
1
2  let A_function = (x) => {
3    document.write("這是A<br>");
4    x();
5    //x=B_function;
6    //x();=B_function();
7  };
8
9  let B_function = () => {
10   document.write("這是B<br>");
11 };
12
13 A_function(B_function);
14
```

## 範例

```
1 function total_2(num1, num2) {
2     document.write(num1 + num2 + '<br>');
3 }
4
5
6 function total_1(num1, num2) {
7     document.write(num1 * num2 + '<br>');
8 }
9
10 function c_function(num1, num2, callback) {
11     var a = num1;
12     var b = num2;
13     callback(a, b);
14 }
15
16 c_function(3, 5, total_1);
17 //都是呼叫c_function 但走乘法，結果是 15
18
19 c_function(3, 5, total_2);
20 //都是呼叫c_function 但走加法，結果是 8
21
```

## callback function 優點

- 讓開發者可以自由撰寫要執行的（函式）動作
- 可以確保該（函式）動作是最後執行的（無論是異步還是同步）

## callback function 需注意

當函式之間的相互依賴牽扯過多，可能會產生的 **Callback Hell**，會導致維護起來非常不便喔！



```
1 function hell(win) {
2     // for listener purpose
3     return function() {
4         loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5             loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6                 loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7                     loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8                         loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9                             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                                    loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                                        loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                                            async.eachSeries(SCRIPTS, function(src, callback) {
14                                                loadScript(win, BASE_URL+src, callback);
15                                            });
16                                        });
17                                    });
18                                });
19                            });
20                        });
21                    });
22                });
23            });
24        });
25    };
26}
```

## 變數的有效範圍 (Scope)

"function"是切分變數有效範圍的最小單位

### 全域變數

- 宣告在函式外的變數皆為全域變數
- 函式外 未宣告 直接使用之變數也為全域變數
- 全域變數指的是全域物件的屬性

👉 參閱 : 12\_function2.html 的變數的有效範圍

```
1 //全域物件的「屬性」?
2 var a = 100;
3 document.write(window.a)
4
5
```

### 區域變數

- 區域變數是在進入函式時被建立出來的，只能在函式中使用，函式結束時會將此 變數空間收回
- 在函式內的都是區域變數
- 函式中的參數也是區域變數
- 函式中使用變數時，會先以函式內開始找，找不到才會出去找

### 觀察

👉 參閱 : 12\_function2.html 的變數的有效範圍1

```
1
2 var a;
3 a = 10;
4 var b = 20;
5 c = 30;
6
7 function test(m, n) {
8     var d = 40;
9     a = 100;
10    d = 400;
11    e = m + n;
12 }
13 test(1, 2);
14
15 document.write(`a:${a},b:${b},c:${c},d:${d},e:${e}`);
16 //Uncaught ReferenceError: d is not defined
17
```

## 練習

👉 參閱 : 12\_function2.html 的變數的有效範圍2

題目：小心hoisting

```
1  var x = 1;
2
3
4  function hoisting_scope(y) {
5      document.write(x + "<br>"); //??
6
7      var x = 100;
8      return x + y;
9  };
10
11 document.write(hoisting_scope(50) + "<br>"); //??
12 document.write(x + "<br>"); //??
```

## 立即函式

語法:

```
1
2  (function( ){
3      做什麼事呢~
4  }
5  )()
```

範例

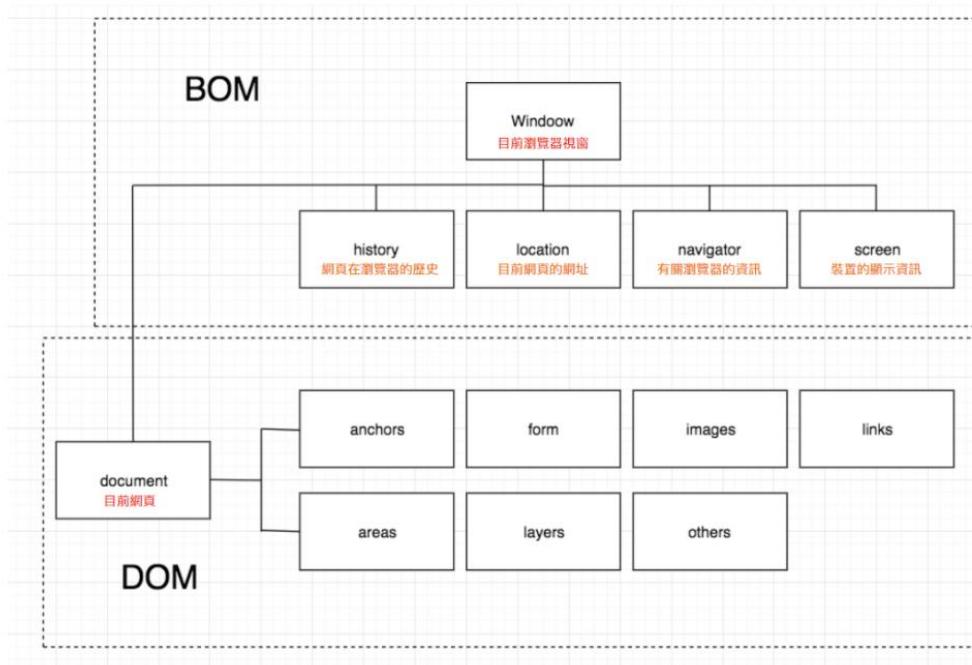
👉 參閱 : 12\_function2.html 的立即函式

```
1
2  let num = '123';
3  (function () {
4      let num2 = '456';
5      document.write(` ${num}&${num2}`); //123&456
6  })();
7  document.write(` ${num}&${num2}`);
8  //Uncaught ReferenceError: num2 is not defined
9
```

## BOM & DOM

### BOM (Browser Object Model · 瀏覽器物件模型)

- 是 **瀏覽器** 所有功能的核心，與網頁的內容無關。
- 核心其實是 `window` 物件。而 `window` 物件提供的屬性主要為 `document`、`location`、`navigator`、`screen`、`history`.....。
- 在瀏覽器裡的 `window` 物件扮演著兩種角色：
  - ECMAScript 標準裡的「全域物件」(Global Object)
  - JavaScript 用來與瀏覽器溝通的窗口



## Window 物件

### Window 物件.方法

| 方法                         | 作用                                               |
|----------------------------|--------------------------------------------------|
| alert(訊息)                  | 顯示訊息對話盒                                          |
| confirm(訊息)                | 顯示訊息對話盒，並要求使用者選擇是或否                              |
| prompt(訊息,[預設值])           | 顯示訊息對話盒，並要求使用者輸入資料                               |
| open(URL,視窗名稱)             | 開啟一個新的視窗                                         |
| close()                    | 關閉視窗                                             |
| focus()                    | 取得焦點                                             |
| blur()                     | 去除焦點                                             |
| setInterval(指定程式,<br>間隔時間) | 設定指定的程式，<br>經過間隔時間之後會被週期性地重覆被執行                  |
| clearInterval(計時器編號)       | 取消setInterval所設定的重覆執行某函式的工作                      |
| setTimeout(指定程式,<br>間隔時間)  | 設定指定的程式，經過間隔時間之後會被執行一次<br>( 時間單位為毫秒，會回傳計時器編號 ) ) |
| clearTimeout(計時器編號)        | 取消setTimeout所設定的要執行某函式的工作                        |

### window 物件.方法

window 物件下的成員，window 是可以省略不打的

👉 參閱：[13\\_window.html](#)

```
1 // window.alert('我在這!');  
2 alert('我在這!');
```

```
1 let ans = confirm('Are you sure? ');  
2 if(ans){  
3     document.write("耶~");  
4 }else{  
5     document.write("好吧");  
6 }  
7  
8
```

```
1 let psn = prompt("請輸入書號");  
2 document.write(psn);  
3  
4
```

## window 物件.屬性

| 屬性名         | 作用                |
|-------------|-------------------|
| name        | 指定視窗的名稱           |
| document    | 文件物件              |
| history     | 歷史物件(回上一頁)        |
| location    | 位置物件              |
| screen      | 螢幕物件              |
| console     | 主控台物件             |
| innerHeight | 視窗內的高度 ( 含捲軸 )    |
| innerWidth  | 視窗內的寬度 ( 含捲軸 )    |
| outerHeight | 視窗的高度 ( 含捲軸、工具列 ) |
| outerWidth  | 視窗的寬度 ( 含捲軸、工具列 ) |
| pageXOffset | 文件左上角被捲出的寬度       |
| pageYOffset | 文件左上角被捲出的高度       |

👉 參閱 : 14\_window1.html

```
1 | document.write("innerWidth:" + window.innerWidth + "<br>");  
2 | document.write("innerHeight:" + window.innerHeight + "<br>");  
3 | document.write("outerWidth:" + window.outerWidth + "<br>");  
4 | document.write("outerHeight:" + window.outerHeight + "<br>");  
5 |  
6 |  
7 |
```

innerWidth:729  
innerHeight:491  
outerWidth:1473  
outerHeight:1102  
location:http://127.0.0.1:5501/14\_window1.html  
location:http://127.0.0.1:5501/14\_window1.html

### **location** 物件.屬性

```
1 | document.write("location:" + location + "<br>");  
2 | document.write("location:" + location.href + "<br>");  
3 |  
4 | location.href = 'https://www.google.com/';  
5 | // 跳轉去google  
6 |
```

### **location** 物件.方法

```
1 | // 重新頁面  
2 | location.reload();  
3 |  
4 |
```

### **history** 物件.屬性

```
1 | //取得目前瀏覽歷史的總數  
2 | document.write("history.length" + history.length);  
3 |  
4 |
```

### **history** 物件.方法

```
1 | //同於用戶在瀏覽器上點選「上一頁」按鈕  
2 | history.back();  
3 | history.go(1);  
4 |  
5 |
```

```
1 | //同於用戶在瀏覽器上點選「下一頁」按鈕  
2 | history.forward();  
3 | history.go(-1);  
4 |  
5 |
```

```
1 | //重新整理  
2 | history.go(0);  
3 | location.reload();  
4 |  
5 |
```

### 範例

透過from表單傳送資料，假裝資料錯誤回到上一頁的範例

👉 參閱：[15\\_history.html](#) [15\\_history2.html](#)

- ▶ [15\\_history.html](#)
- ▶ [15\\_history2.html](#)

## navigator 物件

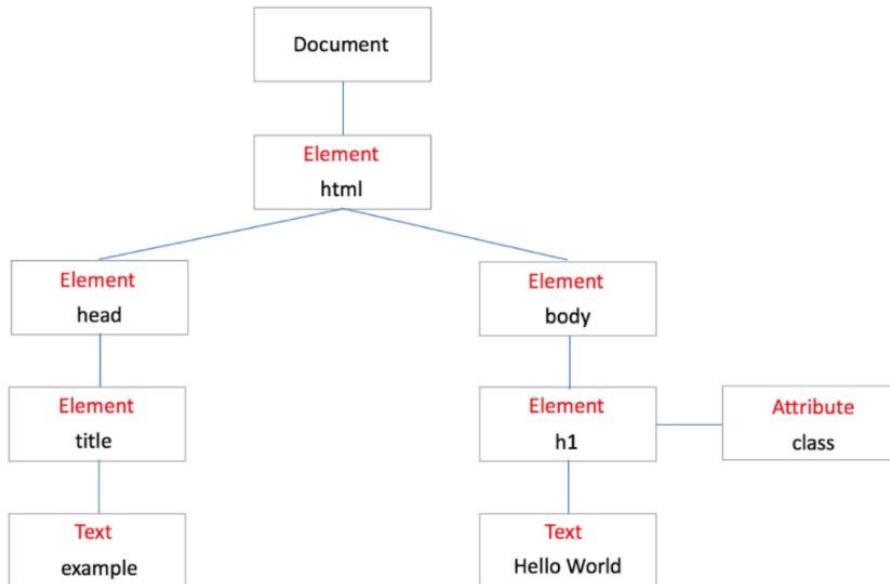
| 屬性名稱            | 說明         |
|-----------------|------------|
| appCodeName     | 瀏覽器代碼      |
| appName         | 瀏覽器名稱      |
| appVersion      | 瀏覽器版本      |
| userAgent       | 使用者代理器     |
| platform        | 瀏覽器所在的作業平台 |
| browserLanguage | 瀏覽器所使用的語系  |

👉 參閱 : 16\_navigator.html

```
1 | document.write("<span>appName : </span>", window.navigator.appName, "<br>");
2 | document.write("<span>appVersion : </span>", navigator.appVersion, "<br>");
3 | document.write("<span>user-agent : </span>", navigator.userAgent, "<br>");
4 | document.write("<span>platform: </span>", navigator.platform, "<br>");
5 |
6 |
```

## Document 物件(Document Object Model，文件物件模型)

- 是一個以樹狀結構來表示 HTML 文件的模型，而組合起來的樹狀圖，我們稱之為「DOM Tree」。



- 我們把 HTML 每個節點的關係攤來看，就好像是一棵樹一樣。在最根部的地方，就是 document。往下可以延伸出一個個的 HTML 標籤。
- 此時JavaScript就可以藉由DOM去存取並改變HTML架構、樣式和內容的方法，JavaScript 就是透過 DOM 提供的 API 來對 HTML 做存取與操作。

**🔥 注意：**

BOM: JavaScript 與瀏覽器溝通的窗口，不涉及網頁內容。

DOM: JavaScript 用來控制網頁的節點與內容的標準。

## document 物件.屬性

| 屬性                        | 說明                    |
|---------------------------|-----------------------|
| document.title            | 設置文檔標題等價於HTML的title標籤 |
| document.bgColor          | 設置頁面背景色               |
| document.fgColor          | 設置前景色(文本顏色)           |
| document.linkColor        | 未點擊過的鏈接顏色             |
| document.alinkColor       | 激活鏈接(焦點在此鏈接上的)的顏色     |
| document.vlinkColor       | 已點擊過的鏈接顏色             |
| document.URL              | 設置URL屬性從而在同一窗口打開另一網頁  |
| document.fileCreatedDate  | 文件建立日期，只讀屬性           |
| document.fileModifiedDate | 文件修改日期，只讀屬性           |
| document.fileSize         | 文件大小，只讀屬性             |
| document.cookie           | 設置和讀出cookie           |
| document.charset          | 設置字符集 簡體中文:           |

```
1 | 
2 |   document.title = 'title';
3 | 
```

title

X



參閱 : 17\_document.html

### document 物件.方法

| 方法                                  | 說明                                 |
|-------------------------------------|------------------------------------|
| getElementById("idName")            | 透過idName抓取 ( 一個 )                  |
| getElementsByClassName("className") | 透過className抓取 ( 回傳多個 ,<br>就算只有一個 ) |
| getElementsByTagName("tagName")     | 透過標籤名稱抓取 ( 回傳多個 ,<br>就算只有一個 )      |
| getElementsByName("tagName")        | 透過name抓取 ( 回傳多個 ,<br>就算只有一個 )      |
| querySelector(css選擇器)               | 透過css選擇器抓取 ( 回傳一個 )                |
| querySelectorAll(css選擇器)            | 透過css選擇器抓取 ( 回傳多個 )                |
| createElement('tagName')            | 創建新元素                              |
| createTextNode()                    | 創建新文字節點                            |

### document 物件.方法 ( 抓取某元素 )

在javascript中dom的操作十分重要，還記的我們說明到JS能做什麼嗎?javascript要跟使用者互動~我們會透過javascript來控制、改變、新增、修改、移除...html上的標籤或是css樣式等等

所以我們要來學習如何抓到或是指向某個標籤元素

- ID選擇器

```
1 | 
2 |   document.getElementById("id_name");
3 |   document.querySelector("#id_name");
4 | 
```

#### 範例

```
1 | <div id="root">idName取在重要的東西上，id我們讓他唯一</div>
```

```
1 | 
2 |   console.log(document.getElementById('root'));
3 |   console.log(document.querySelector('#root'));
4 | 
```

- class選擇器

```
1 | document.getElementsByClassName("class_name");
2 | document.querySelectorAll(".class_name");
3 | document.querySelector(".class_name");
4 |
5 |
```

#### 範例1 一個classname

```
1 | <div class="byClassName">單一個class</div>
```

```
1 |
2 | console.log(document.getElementsByClassName('byClassName')[0]);
3 | console.log(document.querySelectorAll('.byClassName')[0]);
4 | console.log(document.querySelector('.byClassName'));
5 |
```

#### 範例2 多個同名的classname

```
1 |
2 | <div class="box">box1</div>
3 | <div class="box">box2</div>
4 | <div class="box">box3</div>
5 | <div class="box">box4</div>
6 | <div class="box">box5</div>
7 |
```

```
1 |
2 | console.log(document.getElementsByClassName('box'));
3 | console.log(document.querySelectorAll('.box'));
4 |
5 | console.log(document.getElementsByClassName('box')[2]);
6 | console.log(document.querySelectorAll('.box')[2]);
7 |
8 | console.log(document.getElementsByClassName('box').length);
9 | console.log(document.querySelectorAll('.box').length);
10 |
```

看到陣列的樣子~~~就表示我們可以透過迴圈的方式取值!

```
1  for (var i = 0; i < document.getElementsByClassName('box').length; i++) {  
2      console.log(document.getElementsByClassName('box')[i]);  
3  }  
4  
5  for (var i = 0; i < document.querySelectorAll('.box').length; i++) {  
6      console.log(document.querySelectorAll('.box')[i]);  
7  }  
8  
9
```

- 屬性選擇器

```
1  document.querySelectorAll("input[name='abc']");
2  //選取所有input name屬性為abc的<input>元素
3
4
5  document.querySelectorAll("div[data-type]");
6  //獲取所有帶自定義屬性data-type的<div>元素
7
```

### 範例

```
1  <form>
2      <label>男<input type="radio" name="gender" value="1" checked></label>
3      <label>女<input type="radio" name="gender" value="2"></label>
4      <label>其他<input type="radio" name="gender" value="3"></label>
5
6  </form>
7
```

```
1
2  console.log(document.getElementsByName('gender').length)
3  console.log(document.querySelectorAll("input[name='gender']").length)
4  console.log(document.querySelector("input[name='gender']:checked").value)
5
```

## 存取 HTML 元素物件的內容

我們已經會抓標籤元素了嘛~那我們要開始對他們動些手腳了  
這邊會針對非表單欄位元素，有起始標籤和終止標籤之元素來存取 HTML 元素物件的內容

- `innerHTML`: 元素內的不管是標籤或是文字都可以弄進來
- `innerText`: 元素內的文

### 觀察

👉 參閱 : 18\_innerhtmlandinnertext.html

```
1 <div id="app">
2   <p>這裡有一個p標籤</p>
3 </div>
4
```

```
1 let app = document.querySelector('#app').innerHTML;
2 let app_text = document.querySelector('#app').innerText;
3 console.log(app);
4 console.log(app_text);
5
```

接著我們反過來~我們用字串的方式，把html結構和文字渲染到頁面上

```
1 let html = `
2 <hr>
3   <p>換掉p標籤</p>
4 <hr>
5 `
6
7 document.querySelector('#app').innerHTML = html;
8
```

```
1 let html = `
2 <hr>
3   <p>換掉p標籤</p>
4 <hr>
5 `
6
7 document.querySelector('#app').innerText = html;
8
```

再來～我們要來看一些表單欄位元素  
單欄位元素可以透過 **value** 屬性取得或設定輸入盒的值

👉 參閱 : 19\_value.html

```
1 | <form>
2 |   輸入 : <input class="inputtext" type="text" value="打字打字打字"><br>
3 |   <label>男<input type="radio" name="gender" value="1" checked></label>
4 |   <label>女<input type="radio" name="gender" value="2"></label>
5 |   <label>其他<input type="radio" name="gender" value="3"></label>
6 |
7 | </form>
8 | 
```

```
1 | console.log(document.querySelector(".inputtext"))
2 | console.log(document.querySelector(".inputtext").value);
3 | 
```

```
1 | console.log(document.querySelector("input[name='gender']:checked").value)
2 |
3 | 
```

## 改變HTML元素物件外觀的樣式

透過 「物件. **style** .樣式屬性名稱 = 怎樣怎樣的值」，可取得或改變標籤物件的css

👉 參閱 : 20\_style.html

```
1 | <p class="changeP">這是p標籤</p>
2 |
3 |
```

```
1 | document.querySelector('.changeP').style.color = "pink";
2 | document.querySelector('.changeP').style.backgroundColor = "black";
3 |
4 |
```

```
<p class="changeP">這是p標籤</p>
document.querySelector('.changeP').style.color = "pink";
document.querySelector('.changeP').style.backgroundColor = "black";
```

原本css是  
XXX-XXX  
這種寫法的  
都要改為  
駝峰的寫法喔

## 新增或移除特定屬性

| 屬性或方法                     | 說明           |
|---------------------------|--------------|
| classList.add(class名稱)    | 增加元素class的屬性 |
| classList.remove(class名稱) | 刪除元素class的屬性 |
| setAttribute(屬性,值)        | 增加元素某屬性的值    |
| getAttribute(屬性)          | 取得元素某屬性的值    |

👉 參閱 : 21\_classList.html

```
1 |     <br>
2 |
3 |
```

```
1 |
2 |     .eye{
3 |         border: 4px solid rgb(0, 255, 17);
4 |
5 |     }
6 |
7 |     .eyeborder{
8 |         border: 1px solid red;
9 |
10| }
```

```
1 |     document.querySelector('.eye').classList.add('eyeborder');
```

我們加上一個classname

```

```

css有先後順序，有衝突的時候，後來的classname會蓋掉之前的

```
1 |     document.querySelector('.eye').setAttribute('src','./img/hidden.svg');
2 |     document.write(document.querySelector('.eye').getAttribute('src'));
```

透過"setAttribute"修改或新增標籤的屬性

## setInterval / setTimeout 計時器

| 方法            | 描述                                 |
|---------------|------------------------------------|
| setInterval   | 週期性的執行一個函數(function)或代碼，秒數為重複執行的時間 |
| clearInterval | 取消調用 setInterval 設置的重複執行動作         |
| setTimeout    | 在指定的延遲時間後調用一個函數或代碼片段，秒數為延遲執行的時間    |
| clearTimeout  | 取消由 setTimeout() 設置的 timeout       |

### 語法

```
1 | var timerId = setInterval( function(){ } | ' code ' , 毫秒 ) ;
2 | clearInterval( timerId ) ;
```

```
1 | var timerId2 = setTimeout( function(){ } | ' code ' , 毫秒 ) ;
2 | clearTimeout( timerId2 ) ;
```

1000毫秒=1秒

### 範例

👉 參閱 : 22\_setInterval.html

```
1 | <div id="divNum" style="
2 |   color:rgb(255, 255, 255);
3 |   width:300px;
4 |   height:150px;
5 |   background-color:#333;
6 |   font-size:50px;
7 |   text-align:center;">5
8 |
9 | </div>
10 | 
```

```
1  
2 var num = 5;//1.設一個變數，我要從5開始倒數  
3 var timerId;//2.設一個變數，準備啟動或取消我的計時器  
4  
5 //5.設置每一秒要做什麼事情的function  
6 function countDown() {  
7     num--; //6.先將倒數減一  
8  
9     if (num == 0) { //8.數數到0的時候就是最後了！  
10         num = "Happy new year~~";//9.0的時候要渲染新年快樂  
11         clearInterval(timerId);//10.0的時候不用再有計時器了，清掉  
12     }  
13  
14     // 7.減一之後我要渲染到頁面上  
15     document.querySelector("#divNum").innerText = num;  
16 }  
17  
18 //3. 設置一個function，啟動計時器  
19 function init() {  
20     timerId = setInterval(countDown, 1000);  
21 }  
22 init();//4.呼叫設計計時器的function  
23
```

👉 參閱 : 23\_setTimeout.html

```
1 <div id="divNum" style="color:rgb(255, 255, 255); width:300px; height:150px; background-color:#333; font-size:50px; text-align:center;">5  
2 </div>  
3  
4  
5  
6  
7  
8  
9  
10
```

```
1  var num = 5; //1.設一個變數，我要從5開始倒數
2  var duration = 1000;//2.設一個變數，第一是1000毫秒
3
4
5 //5.設置每一秒要做什麼事情的function
6 function countDown(){
7     num --; //6.先將倒數減一
8     if( num == 0){
9         //10.0的時候就渲染新年快樂
10        document.querySelector("#divNum").innerText = "Happy new year~~";
11    }else{
12        //7.將數字渲染到頁面上
13        document.querySelector("#divNum").innerText = num;
14        //8.毫秒加上1000毫秒，這樣才可以下一個會多一秒之後再執行
15        duration += 1000;
16        //9.啟動計時器
17        setTimeout( countDown, duration);
18    }
19}
20
21 //3. 設置一個function，啟動計時器
22 function init(){
23     setTimeout(countDown, duration);
24 }
25
26 init(); //4.呼叫設計計時器的function
27
```

🔥 注意：

忽然!!!我們會發現 `setTimeout()` 與 `setInterval()` 好像有點問題???簡單來說，當主要執行緒內工作的時間太久，就勢必會延遲執行!

但我們可以透過先計算 client 的時間與 server 端的時間差等等去比對修正，不過這樣的誤差永遠都會存在...

## 物件

- 封裝資料，重複使用
- 可裝載程式碼，以執行操作行為
- 將資料物件化可方便傳遞

### 語法

```
1 | var point = new Object();
2 | var point = {};
3 |
4 |
```

### 實字標示法建立物件

```
1 | let obj = {
2 |   name: 'shang',
3 |   skill: ['html', 'css', 'js'],
4 |   say: function () {
5 |     return 'hi~';
6 |   }
7 | }
8 |
9 |
10| document.write(obj.name + '<br>');
11| document.write(obj['name'] + '<br>');
12| document.write(obj.skill[0] + '<br>');
13| document.write(obj.say() + '<br>');
14|
```

### 建構子語法建立物件

```
1 | // let obj2 = new Object();
2 | let obj2 = {};
3 | obj2.name = 'jack';
4 | obj2.skill = ['vue', 'react', 'laravel'];
5 | obj2.say = function () {
6 |   return 'hello';
7 | };
8 | document.write(obj2.name + '<br>');
9 | document.write(obj2.skill[2] + '<br>');
10| document.write(obj2.say() + '<br>');
11|
12|
13|
14|
15| obj2.name = 'amy';//改掉囉！
16| document.write(obj2.name + '<br>');
17| console.log(obj2);
18|
```

```
▼ Object ⓘ
  name: "amy"
  ▶ say: f ()
  ▶ skill: (3) ["vue", "react", "laravel"]
  ▶ proto : Object
```

## 建構子表示法建立物件

```
1 function obj3(name, skill, say) {
2     this.name = name;
3     this.skill = skill;
4     this.say = function () {
5         return say
6     };
7 }
8
9
10 let who = new obj3('sara', ['git', 'gulp', 'webpack'], 'QQ');
11 let who2 = new obj3('sara2', ['git2', 'gulp2', 'webpack2'], 'QQ2');
12
13 document.write(who.name + '<br>');
14 document.write(who2.name + '<br>');
15 document.write(who.skill[1] + '<br>');
16 document.write(who.say() + '<br>');
17
```

this指向某個建構子 (constructor) 所建立的物件

## for in 物件

```
1
2 var john = {
3     FirstName: "Chan",
4     LastName: "Tak-Wai",
5 };
6
7 for (var p in john) {
8     document.write(p + ":" + john[p] + "<br>");
9 }
10
```

FirstName:Chan  
LastName:Tak-Wai

## this 關鍵字

- this 是 function 執行時，自動生成的一個內部物件
- 隨著 function 執行場合的不同，this 所指向的值，也會有所不同
- 在大多數的情況下，this 代表的就是呼叫 function 的物件 (Owner Object of the function)
- 經常使用在函式和物件之中
- 指"本人的~"，如果沒有會往外層找

👉 參閱 : 25\_this.html

### 全域範圍的函式

```
1  function windowSize() {
2      return this.innerWidth;
3  }
4  document.write(windowSize() + "<br>");  

5
6 //這裡的this指向window
7
8
```

### 全域範圍的變數

```
1  var width = 100;
2  var shape = {
3      width: 500
4  };
5
6  function whichWidth() {
7      return this.width;
8      //this 在自己這裡找不到width，所以往外找
9  }
10
11 document.write(whichWidth() + "<br>");//100
12
13
```

### 物件的方法

```
1  var shape = {
2      width: 600,
3      getotherwidth: function () {
4          return this.width
5      }
6  };
7
8  document.write(shape.getotherwidth() + "<br>");//600
9
10
```

## 函式運算式轉化為方法

### 函式運算式轉化為狀況1

```
1  var width = 200;
2  var shape = {
3      width: 400,
4  };
5  var getotherwidth = function () {
6      return this.width
7  }
8  shape.getwidth = getotherwidth;
9
10 // 實際這段 shape.getwidth = getotherwidth;
11 // 會讓 shape 物件變成以下
12 // var shape = {
13 //     width: 400,
14 //     getwidth: function () {
15 //         return this.width
16 //     }
17 // };
18
19 document.write(shape.getwidth() + "<br>");//400
20
21
```

### 函式運算式轉化為狀況2

```
1
2  var width = 200;
3  var shape = {
4      width: 400,
5  };
6  var getotherwidth = function () {
7      return this.width
8  }
9
10 document.write(getotherwidth() + "<br>");//200
11
```

### 函式運算式轉化為狀況3(箭頭函式this有嚴格模式)

```
1
2  var width = 200;
3  var shape = {
4      width: 400,
5      getwidth: () => {
6          return this.width
7      }
8  };
9  document.write(shape.getwidth() + "<br>");//200
10
```

## 函式運算式轉化為狀況4(箭頭函式this有嚴格模式)

```
1  var width = 200;
2  var shape = {
3      width: 400,
4  };
5  var getotherwidth = () => {
6      return this.width
7  }
8  shape.getwidth = getotherwidth;
9  document.write(shape.getwidth() + "<br>");//200
10
11
```



因為時間關係，我們不會深討論箭頭函式～

延伸閱讀:[https://eyesofkids.gitbooks.io/javascript-start-from-es6/content/part4/arrow\\_function.html](https://eyesofkids.gitbooks.io/javascript-start-from-es6/content/part4/arrow_function.html)

---

## 傳值 & 傳址

指的是電腦記憶體中的東西，與程式的參照傳遞互動的模式

### 傳值 Call by value

基本型別就是使用「傳值」

當我們創造變數並給值時，變數會指向值在電腦記憶體中的位置，若我們以這個值為參照，指定另一個變數指向這個值時，電腦會在記憶體中新增(複製)一個新值，讓後來的這個變數指向新的值。

```
1  var a = 10;
2  var b = 10;
3
4  document.write('a === b:', a === b, '<br>'); // true
5
6
7  var c = a;
8  c++;
9
10 document.write("a:", a, '<br>'); // 10
11 document.write("c:", c, '<br>'); // 11
12
13
```

## 傳址 Call by reference

物件型別就是使用「傳址」

當我們創造變數並給值(物件)時，變數會指向物件在電腦記憶體中的位置，若我們以這個物件為參照，指定另一個變數指向這物件，這個變數就會指向電腦記憶體中同樣的物件，不會有新的物件在記憶體中被創造出來。

```
1 var obj1 = { v1: 1 };
2 var obj2 = { v1: 1 };
3
4 document.write("obj1 === obj2:", obj1 === obj2); // false.
```

因為比較的是“**記憶體位置**”，而非值

👉 參閱 : 26\_callby.html

```
1 var obj1 = { v1: 1 };
2 var obj2 = { v1: 1 };
3
4 document.write("obj1 === obj2:", obj1 === obj2, '<br>');
5 // false
6 // 記憶體位置不同
7
8
```

```
1 var obj1 = { v1: 1 };
2 var obj2 = obj1;
3
4 obj1.v1 = 0;
5 document.write(obj2.v1, '<br>'); // 0
6
7 obj2.v2 = 2;
8 document.write(obj1.v2, '<br>'); // 2
9
10 //改動誰就會跟著給動呢~~~
11
12 //修改任意物件屬性時，另一邊的屬性也會更動
13 //這是因為兩個變數指向相同的記憶體位置
14 //並沒有新的物件被複製出
15
16 document.write(obj1 === obj2, '<br>'); // true
17
18
```

## 複製物件

- 淺拷貝

如果有巢狀或多維，它們還是傳址。因為深層的物件或陣列還是傳址，不會完全複製一份。

物件：Object.assign()、...展開運算子

陣列：slice()、concat()、forEach、push()、map()、filter()、...展開運算子

- 深拷貝

深拷貝就是完全複製一份，不會有共用記憶體的問題。

利用 JSON 方法：先轉 JSON 格式，再轉回來。

-物件轉字串:JSON.stringify(物件變數)

-字串轉物件:JSON.parse(字串)

## forEach

```
1 const array1 = ['a', 'b', { c: 'tibame' }];
2
3 array1.forEach(function (element) {
4     console.log(element)
5 });
6
7 array1.forEach(function (element, index) {
8     console.log(element, index)
9 });
10 array1.forEach(function (element, index, arr) {
11     console.log(element, index, arr)
12 });
13
14 array1.forEach(element => console.log(element));
15 array1.forEach((element, index) => console.log(element, index));
16 array1.forEach((element, index, arr) => console.log(element, index, arr));
17
18
```

```
1 const array1 = ['a', 'b', { c: 'tibame' }];
2 const array2 = [];
3 array1.forEach(function (element) {
4     array2.push(element)
5 });
6
7 // array1 與 array2 已經是兩個不同的物件
8 array1[0] = "ddd";
9 console.log(array1[0], array2[0]); //ddd a
10 console.log(array1[0] === array2[0]); //false
11
12 array1[2].c = "TIBAME"
13 console.log(array1[2], array2[2]); //{c: "TIBAME"} {c: "TIBAME"}
14 console.log(array1[2] === array2[2]); //true
15
16
```

## map

```
1 const array1 = [1, 4, 9, 16, { c: 'tibame' }];
2 array1.map(x => console.log(x));
3 array1.map(function (x) {
4     console.log(x);
5 });
6
7 array1.map(function (x, index, arr) {
8     console.log(x, index, arr);
9 });
10
11
```

```
1 const array1 = [1, 4, 9, 16, { c: 'tibame' }];
2 const map1 = array1.map(x => x);
3 //map()會分配記憶體空間儲存新陣列並具備回傳值
4 //所以不用像forEach()要搭配push來產生另一個陣列
5
6 const map1 = array1.map(x => x);
7 console.log(map1);
8 console.log(map1 === array1); //false
9 map1[0] = 11111;
10 console.log(map1[0], array1[0]); //11111 1
11 console.log(map1[0] === array1[0]); //false
12
13 map1[4].c = "TIBAME";
14 console.log(map1[4], array1[4]); //{c: "TIBAME"} {c: "TIBAME"}
15 console.log(map1[4] === array1[4]); //true
16
```

## map比較forEach

```
1 const array1 = [1, 4, 9, 16];
2 const map1 = array1.map(x => x);
3 const map2 = array1.forEach(y => y * 2);
4
5 console.log(map1); // [1, 4, 9, 16]
6 console.log(map2); // undefined
7
```

## ...展開運算式

可以比較一下前面有提到的 arguments

先來看一下怎麼使用

```
1 function sum (...data){  
2     let total = 0;  
3     for (let i = 0; i < data.length; i++) {  
4         total += data[i]  
5     }  
6     return total;  
7 }  
8 console.log(sum(10, 20)); //30  
10
```

```
1 function sum2(a, b, ...data) {  
2     console.log('a的值為: ' + a); //10  
3     console.log('b的值為: ' + b); //20  
4     console.log('data的值為: ' + data); // [30,40,50]  
5 }  
6 sum2(10, 20, 30, 40, 50); // 會印出30  
7  
8
```

還記得箭頭函是沒有arguments~但可以用...展開運算式

```
1 let sum_a = (...data) => {  
2     let total = 0;  
3     for (let i = 0; i < data.length; i++) {  
4         total += data[i]  
5     }  
6     return total;  
7 }  
8 console.log(sum_a(10, 20)); // 會印出30  
10
```

```
1 let array1 = [1, 4, 9, 16, { c: 'tibame' }];  
2 let array2 = [...array1];  
3 console.log(array2)  
4 console.log(array1 === array2); // false  
5 array1[0] = 11111;  
6 console.log(array1[0], array2[0]); // 11111 1  
7 console.log(array1[0] === array2[0]); // false  
8 array1[4].c = "TIBAME";  
9 console.log(array1[4], array2[4]); // {c: "TIBAME"} {c: "TIBAME"}  
10 console.log(array1[4] === array2[4]); // true  
11  
12
```

## 事件

### 事件驅動程式設計

- 事件：  
使用者再瀏覽網頁過程中由系統或使用者引發的一些狀況  
如：網頁下載完畢、使用者點擊按鈕
- 事件處理程序：  
針對所發生的事件予以適切的回應，撰寫相關的程式碼，即為事件處理程序
- 主要概念：  
物件 + 事件 + 事件處理程序  
由哪個物件？+ 發生什麼事情？+ 該做哪些事情？

### 設定事件處理器

事件處理器必須是函式物件

方法：

- 在HTML標籤中以屬性指定事件處理器<button on事件名稱 "js...">  
老語法不建議用，但框架是用這樣的觀念
- 使用JS程式碼註冊事件處理器  
物件.on事件名稱 = “事件處理器”
- 使用 addEventListener() 註冊事件處理器  
addEventListener(事件名稱, 事件處理器, useCapture)  
removeEventListener(事件名稱, 事件處理器, useCapture)

| 關鍵字        | 說明                               |
|------------|----------------------------------|
| event      | 必須。描述事件名稱的物件。使用click來取代onclick。  |
| function   | 必須。描述事件觸發後執行的函數。                 |
| useCapture | 預設是false(在冒泡階段執行)。true(在捕捉階段執行)。 |

## 事件種類

| Attribute  | Description                                                                     |
|------------|---------------------------------------------------------------------------------|
| click      | 按下鼠標（通常是按下主按鈕）時觸發。                                                              |
| dblclick   | 在同一個元素上雙擊鼠標時觸發。                                                                 |
| mousedown  | 按下鼠標鍵時觸發。                                                                       |
| mousemove  | 當鼠標在一個節點內部移動時觸發。                                                                |
| mouseout   | 鼠標離開一個節點時觸發。                                                                    |
| mouseover  | 鼠標進入一個節點時觸發。                                                                    |
| mouseup    | 釋放按下的鼠標鍵時觸發。                                                                    |
| mousewheel | Deprecated. Use the onwheel attribute instead                                   |
| wheel      | 滾動鼠標的滾輪時觸發。                                                                     |
| load       | 頁面或圖像加載完成                                                                       |
| resize     | 調整視窗或框架的大小                                                                      |
| scroll     | 當滾動元素的滾動條時發生該事件                                                                 |
| keydown    | 按下鍵盤時觸發。                                                                        |
| keypress   | 按下有值的鍵時觸發，即按下Ctrl、Alt、Shift、Meta這樣無值的鍵，這個事件不會觸發。對於有值的鍵，按下時先觸發keydown事件，再觸發這個事件。 |
| keyup      | 鬆開鍵盤時觸發該事件。                                                                     |
| drag       | Script to be run when an element is dragged                                     |
| dragend    | 在拖動操作結束時運行的腳本                                                                   |
| dragenter  | Script to be run when an element has been dragged to a valid drop target        |
| dragleave  | Script to be run when an element leaves a valid drop target                     |
| dragover   | 將元素拖到有效放置目標上時要運行的腳本                                                             |
| dragstart  | 在拖動操作開始時運行的腳本                                                                   |
| drop       | 拖放被拖動元素時要運行的腳本                                                                  |

## 事件的寫法

👉 參閱 : 29\_event.html

在HTML標籤中以屬性指定事件處理器(不建議)

```
1 <div class="box_color_c"></div>
2 <button id="btn"
3 onclick="document.querySelector('.box_color_c').style.backgroundColor = 'gray'; "
4 click
5 </button>
```

```
1 .box_color_c {
2     width: 200px;
3     height: 200px;
4     border: 1px solid gray;
5 }
```

在HTML標籤中以屬性指定事件處理器，再搭配使用JS程式碼註冊事件處理器

```
1 <div class="box_color_c"></div>
2 <button id="btn" onclick='show_c()'>click</button>
```

```
1
2 function show_c() {
3     let color = `rgb(${Math.floor(Math.random() * 256)},
4                     ${Math.floor(Math.random() * 256)},
5                     ${Math.floor(Math.random() * 256)})`;
6
7     document.querySelector('.box_color_c').style.backgroundColor = color;
8 }
9
```

使用JS程式碼註冊事件處理器

```
1 <div class="box_color_c"></div>
2 <button id="btn">click</button>
```

```
1
2 function show_c() {
3     let color = `rgb(${Math.floor(Math.random() * 256)},
4                     ${Math.floor(Math.random() * 256)},
5                     ${Math.floor(Math.random() * 256)})`;
6
7     document.querySelector('.box_color_c').style.backgroundColor = color;
8 }
9
10 document.querySelector('#btn').onclick = show_c;
11
```

## 註冊事件處理器(推推👉)

```
1 <div class="box_color_c"></div>
2 <button id="btn">click</button>

1
2 function show_c() {
3     let color = `rgb(${Math.floor(Math.random() * 256)},
4                     ${Math.floor(Math.random() * 256)},
5                     ${Math.floor(Math.random() * 256)})`;
6
7     document.querySelector('.box_color_c').style.backgroundColor = color;
8 }
9
10 document.querySelector('#btn').addEventListener('click', show_c);
11
12
13
```

## 註冊事件處理器(匿名)

```
1 <div class="box_color_c"></div>
2 <button id="btn">click</button>

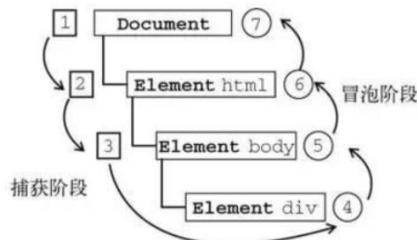
1
2 document.querySelector('#btn').addEventListener('click', function(){
3     let color = `rgb(${Math.floor(Math.random() * 256)},
4                     ${Math.floor(Math.random() * 256)},
5                     ${Math.floor(Math.random() * 256)})`;
6
7     document.querySelector('.box_color_c').style.backgroundColor = color;
8 });
9
```

## 註冊事件處理器的 useCapture

useCapture預設是false

- false:在冒泡階段執行
- true:在捕捉階段執行

什麼是冒泡階段跟捕獲階段呢?



### 事件冒泡 (dubbed bubbling)

事件冒泡我們從字面意思理解就是當用戶行為觸發我們頁面的定義好的事件後，會有一個由內到外的一個冒泡過程，而不是一下子就命中事件綁定的元素

### 事件捕獲 (event capturing)

事件捕獲與冒泡恰恰相反，當滑鼠點擊或者觸發dom事件時，瀏覽器會從根節點開始由外到內進行事件傳播，即點擊了子元素，如果父元素通過事件捕獲方式註冊了對應的事件的話，會先觸發父元素綁定的事件

👉 參閱 : 31\_event\_bubble\_capture.html

```
1 <body id="myBody">
2   <p>1111111111111</p>
3   <p>1111111111111</p>
4   <p>1111111111111</p>
5
6   <form id="myForm">
7     <p>這裡是表單</p>
8     <p>這裡是表單</p>
9     <p>這裡是表單</p>
10    <input id="btn" type="button" value="click me">
11      <p>這裡是表單</p>
12      <p>這裡是表單</p>
13      <p>這裡是表單</p>
14    </form>
15
16  </body>
17
```

```
1 form {
2   background-color: #FFC
3 }
4
5
```

```
1  function btn_click(e) {
2      window.alert("button 被按");
3  }
4
5  function myForm_click() {
6      window.alert("myForm 被按");
7  }
8
9  function myBody_click() {
10     window.alert("myBody 被按");
11 }
12
13 function init() {
14     document.querySelector("#btn").addEventListener("click", btn_click, false);
15     document.querySelector("#myForm").addEventListener("click", myForm_click, false);
16     document.querySelector("#myBody").addEventListener("click", myBody_click, false);
17 }
18
19 window.addEventListener('click',init);
20
```

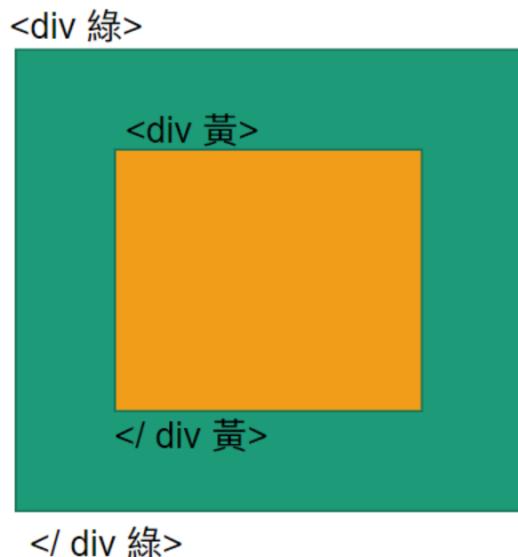
接著試著把 useCapture 的 false 改成 true

```
1  function init() {
2      document.querySelector("#btn").addEventListener("click", btn_click, true);
3      document.querySelector("#myForm").addEventListener("click", myForm_click, true);
4      document.querySelector("#myBody").addEventListener("click", myBody_click, true);
5
6  }
7
```

## e.stopPropagation()

等等，就算知道大部分我們都使用預設的false也就是冒泡事件~但是!!!  
html結構跟事件綁定還是很有可能會出現冒泡事件啊!

像是如下圖



將黃色綠色各綁定一個 click 事件  
當我點擊黃色的 div 時會同時觸發綠色的 div 的事件  
因為他們重疊在一起，所以點擊的時候就會一起觸發  
這現象我們稱作 **冒泡事件**

好!

那我們回到剛剛的範例



參閱 : 31\_event\_bubble\_capture.html

我們將button的click的事件中加入 **e.stopPropagation()**

```
1 | function btn_click(e) {
2 |   e.stopPropagation();
3 |   window.alert("button 被按");
4 |
5 |
6 |}
```

從此不再一直冒泡下去了  
這就是阻止冒泡事件

但!等等!!! **e** ? 我沒有傳參數啊？哪來的東西？

當事件被觸發，瀏覽器就會為這個事件創造一個物件，稱為 **事件物件(event object)**  
事件處理器可透過第一個參數來取得“事件物件”

👉 參閱 : 32\_e.html

```
1 | <button id="btn">click</button>
2 |
3 |
```

```
1 | function btnclick(e) {
2 |   console.log(e);
3 | }
4 | document.querySelector('#btn').addEventListener('click',btnclick);
5 |
6 |
```

我們來看看這個 e 會是什麼東西的！

可以看到這個 e 有很多關於事件物件的屬性或方法

```
▼ MouseEvent {isTrusted: true, screenX: 1970, screenY: -142, clientX: 36, clientY: 15, ...} ⓘ
  altKey: false
  bubbles: true
  button: 0
  buttons: 0
  cancelBubble: false
  cancelable: true
  clientX: 36
  clientY: 15
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 1
  eventPhase: 0
  fromElement: null
  isTrusted: true
  layerX: 36
  layerY: 15
  metaKey: false
  movementX: 0
  movementY: 0
  offsetX: 26
  offsetY: 5
  pageX: 36
  pageY: 15
▶ path: (5) [button#btn, body, html, document, Window]
  relatedTarget: null
  returnValue: true
  screenX: 1970
  screenY: -142
  shiftKey: false
▶ sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: false}
▶ srcElement: button#btn
▶ target: button#btn
  timeStamp: 8899.85499996692
▶ toElement: button#btn
  type: "click"
▶ view: Window {window: Window, self: Window, document: document, name: "", location: Location, ...}
  which: 1
  x: 36
  y: 15
▶ __proto__: MouseEvent
```

## 事件物件

| 屬性或方法             | 說明                                         | 例子                       |
|-------------------|--------------------------------------------|--------------------------|
| target            | 傳回觸發事件的DOM物件                               | e.target                 |
| preventDefault()  | 取消事件的預設行為，但不會影響事件的傳遞                       | e.preventDefault()       |
| stopPropagation() | 終止事件的傳導，但不會影響事件的預設行為                       | e.stopPropagation()      |
| type              | 傳回事件名稱的字串                                  | e.type                   |
| screenX / screenY | 傳回事件觸發時滑鼠相對於螢幕的位置座標                        | e.screenX /<br>e.screenY |
| clientX / clientY | 傳回事件觸發時滑鼠相對於視窗的位置座標                        | e.clientX /<br>e.clientY |
| offsetX / offsetY | 傳回事件觸發時滑鼠相對於來源標籤的位置座標                      | e.offsetX /<br>e.offsetY |
| keyCode           | 傳回事件觸發時鍵盤按下鍵的unicode值                      | e.keyCode                |
| altKey            | 事件觸發時鍵盤alt鍵是否被點擊，傳回布林值                     | e.altKey                 |
| ctrlKey           | 事件觸發時鍵盤ctrl鍵是否被點擊，傳回布林值                    | e.ctrlKey                |
| shiftKey          | 事件觸發時鍵盤shfit鍵是否被點擊，傳回布林值                   | e.shiftKey               |
| button            | 事件觸發時滑鼠的按鍵 ( 0: left, 1:middle, 2: right ) | e.button                 |
| fromElement       | mouseover和mouseout事件觸發時從哪一個標籤移出            | e.fromElement            |
| toElement         | mouseover和mouseout事件觸發時移入哪一個標籤             | e.toElement              |

### e. preventDefault()

作用是 **停止事件的默認動作**

例如有時候我們會利用連結的來當作按鈕，他本身DOM就擁有連結的功能，但是有時候我們會為他新增類似click的事件，而只要在該觸發的事件中加入e.preventDefault()，就不會在執行他默認的動作，也就是不會再執行「**連結到某個網址**」這個動作



參閱 : 33\_preventDefault.html

```
1 <a id="myAnchor"
2   href="https://www.w3schools.com/jsref/event_preventdefault.asp">
3   Go to W3Schools
4   </a>
5
6
```

```
1 document.getElementById("myAnchor").addEventListener("click", function(e) {
2   e.preventDefault();
3 });
4
5
```

## e.target

指向觸發事件的 DOM 物件

👉 參閱 : 34\_target.html

```
1 <div class="box">第一個</div>
2 <div class="box">第二個</div>
3 <div class="box">第三個</div>
4 <div class="box">第四個</div>
5
6
```

```
1 let allbox = document.querySelectorAll('.box');
2 allbox.forEach(element => element.addEventListener('click', function (e) {
3   console.log(e.target);
4   console.log(e.target.innerText);
5 }));
6
7
```

## 其他事件的範例

### change 事件

👉 參閱 : 35\_event\_change.html

```
1 <form>
2   <select name="month"></select>
3 </form>
4 <br>
5 <h2 class="show">1月</h2>
6
7
```

```
1 function which_month() {
2   let val = document.querySelector("select[name='month']").value;
3   console.log(val)
4   document.querySelector('.show').innerText = val + '月';
5 }
6
7
8
9
10 function init() {
11   let month = '';
12   for (var i = 1; i < 13; i++) {
13     month += `<option value="${i}">${i}月</option>`;
14   }
15   document.querySelector("select[name='month']").innerHTML = month;
16   document.querySelector("select[name='month']").addEventListener('change', which_month);
17 }
18
19 window.addEventListener('load', init);
20
```

## keyboard 事件

👉 參閱 : 36\_keyboardEvent.html

```
1 function memId_keydown(e) {  
2     console.log("down keyCode:", e.keyCode);  
3     //key code  
4 }  
5 window.addEventListener('keydown', memId_keydown);  
6  
7
```

```
1 function memId_keyup(e) {  
2     console.log("up  keyCode:  ", e.keyCode);  
3 }  
4 window.addEventListener('keyup', memId_keyup);  
5  
6
```

```
1 function &nbsp;memId_keypress(e)&nbsp;{  
2     &nbsp;&nbsp;&nbsp;console.log("press &nbsp;keyCode: &nbsp;",&nbsp;e.keyCode);  
3 }  
4 window.addEventListener('keypress',memId_keypress);  
5  
6
```

## drag 事件

👉 參閱 : 37\_dragevent.html

```
1 <div id="leftbox">  
2       
3 </div>  
4  
5 <div id="rightbox"></div>  
6  
7
```

```
1 #leftbox {  
2     border: 5px dotted green;  
3     width: 280px;  
4     height: 280px;  
5     float: left;  
6     margin: 5px;  
7     padding: 5px;  
8 }  
9  
10 img {  
11     width: 100%;  
12 }  
13  
14 #rightbox {  
15     border: 5px dotted red;  
16     width: 280px;  
17     height: 280px;  
18     float: left;  
19     margin: 5px;  
20     padding: 5px;  
21 }  
22  
23
```

```

1  let image, rightbox, leftbox, data;
2
3
4  function doFirst() {
5      //先跟HTML畫面產生關聯，再建事件聆聽功能
6      image = document.querySelector(".image");
7      image.addEventListener("dragstart", startDrag);
8
9      rightbox = document.querySelector("#rightbox");
10     rightbox.addEventListener("dragover", p_dragover);
11     rightbox.addEventListener("drop", dropped);
12
13     leftbox = document.querySelector("#leftbox");
14     leftbox.addEventListener("dragover", p_dragover);
15     leftbox.addEventListener("drop", dropped);
16 }
17
18 function startDrag(e) {
19     data = ``;
20 }
21
22 function dropped(e) {
23     e.preventDefault();
24     if (e.target != leftbox && e.target != rightbox) {
25         e.target.parentNode.innerHTML = data;
26     } else {
27         e.target.innerHTML = data;
28     }
29     image.addEventListener("dragend", endDrag);
30 }
31
32 function endDrag(e) {
33     e.target.parentNode.innerHTML = '';
34     doFirst();
35 }
36
37 function p_dragover(e) {
38     e.preventDefault();
39 }
40
41 window.addEventListener("load", doFirst);
42

```

`window.addEventListener("load", doFirst);`

- `window.onload` 方法 :

該方法用於在網頁載入完畢後立刻執行的操作，即當html載入完畢後，立刻執行某個方法等。

- 為什麼使用 `window.onload`?

因為JS中的函式方法等需要在HTML文件渲染完成才可以使用，如果沒有渲染完成，此時的DOM樹是不完整的，這樣JS檔案就可能報出"undefined"錯誤。

## 練習

題目1：

參閱 : time\_s.html

變成透過input 我輸入秒數他幫我算

多少秒 ?

0小時2分5秒

題目2：

參閱 : slimg\_s.html

點小圖換大圖



## DOM tree

DOM節點，節點通常分成以下四種：

- Document

就是指這份文件，也就是這份 HTML 檔的開端，所有的一切都會從 Document 開始往下進行。

- Element

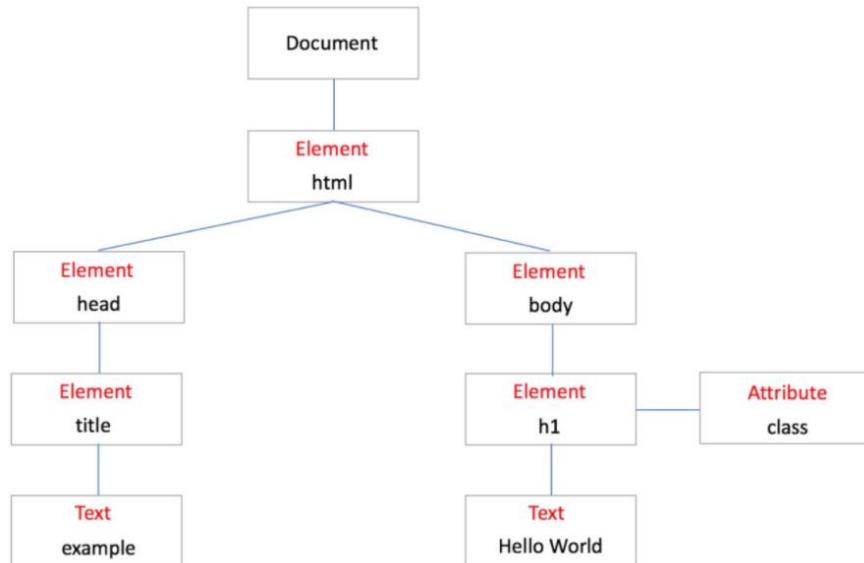
就是指文件內的各個標籤，因此像是 `<div>`、`<p>` 等等各種 HTML Tag 都是被歸類在 Element 裡面。

- Text

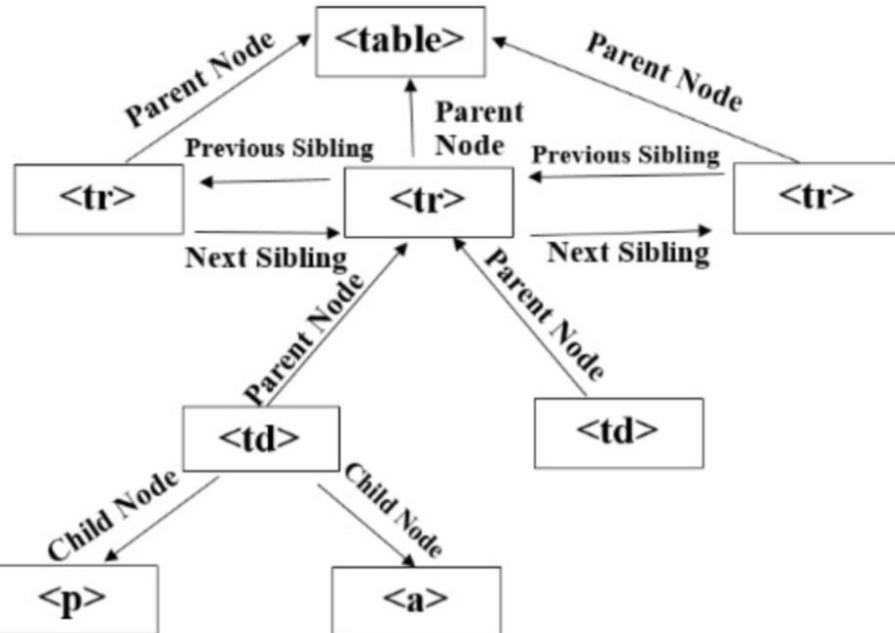
就是指被各個標籤包起來的文字，舉例來說在 `<h1>Hello World</h1>` 中，Hello World 被 `<h1>` 這個 Element 包起來，因此 Hello World 就是此 Element 的 Text

- Attribute

就是指各個標籤內的相關屬性。



DOM 為樹狀結構，樹狀結構最重要的觀念就是 Node 彼此之間的關係



- 父子關係(Parent and Child)

簡單來說就是上下層節點，上層為 Parent Node，下層為 Child Node。

- 兄弟關係(Siblings)

簡單來說就是同一層節點，彼此間只有 Previous 以及 Next 兩種。

## DOM 節點屬性

| 屬性              | 描述                                                                      |
|-----------------|-------------------------------------------------------------------------|
| parentNode      | 以Node的形式返回當前節點的父節點。如果沒有父節點，則為null。                                      |
| childNodes      | 以Node[]的形式存放當前節點的子節點。如果沒有子節點，則返回空陣列。                                    |
| children        | 取得指定element下所有子元素的HTML集合物件(HTMLCollection)但是不包含text nodes跟comment nodes |
| firstChild      | 以Node的形式返回當前節點的第一個子節點。如果沒有子節點，則為null。                                   |
| lastChild       | 以Node的形式返回當前節點的最後一個子節點。如果沒有子節點，則為null。                                  |
| previousSibling | 以Node的形式返回緊挨當前節點、位於它之前的兄弟節點。如果沒有這樣的節點，則返回null。上一個兄弟節點                   |
| nextSibling     | 以Node的形式返回當前節點的兄弟下一個節點。如果沒有這樣的節點，則返回null。下一個兄弟節點                        |
| nodeName        | 節點的名字，Element節點則代表Element的標記名稱。                                         |

## 參閱 :38\_domAtt.html

```
1 <div>
2   <h2>hello world</h2>
3   <p class="box">welcome to tibame!</p>
4   <p>say something</p>
5 </div>
6
7
8 <!-- <div><h2>hello world</h2><p>welcome to tibame!</p><p>say something</p></div> -->
9
```

```
1 console.log(document.querySelector('div'))
2 // console.log(document.querySelector('div').nodeName)//DIV
3 // console.log(document.querySelector('div').children)//HTMLCollection(3)&nbsp;[h2, p, p]
4 // console.log(document.querySelector('div').childNodes)//NodeList(7)&nbsp;[text, h2, text, p, text, p, text]
5 // console.log(document.querySelector('div').children[0])//<h2>hello world</h2>
6 // console.log(document.querySelector('div').childNodes[0])//#text 還有！！換行會有文字節點喔！空白的麻煩
7 // console.log(document.querySelector('h2').nextSibling)//#text
8 // console.log(document.querySelector('h2').parentNode)
9
```

## DOM 文件.方法

| 方法                      | 描述                    |
|-------------------------|-----------------------|
| <b>createElement()</b>  | 用指定的標記名建立新的Element節點。 |
| <b>createTextNode()</b> | 用指定的文字建立新的TextNode節點。 |

## DOM 節點.方法

| 方法                               | 描述                                                         |
|----------------------------------|------------------------------------------------------------|
| <b>append()</b>                  | 插入到元素內到最後一個子元素                                             |
| <b>appendChild(childNode)</b>    | "將指定的子節點加到node的子節點清單中tr.appendChild(td)"                   |
| <b>insertBefore(aNode,bNode)</b> | "將指定的節點插到node之下，特定節點之前<br>tr.insertBefore(newTd,targetTd)" |
| <b>removeChild(childNode)</b>    | "將子節點從node移除tr.removeChild(td)"                            |
| <b>replaceChild(aNode,bNode)</b> | "將此node的子節點中，<br>某特定節點換成另一個節點tr.replaceChild(aNode,bNode)" |
| <b>hasChildNode()</b>            | "傳回一個布林值，指出此node是否有子節點tr.hasChildNode()"                   |

## 範例

👉 參閱 : 39\_domarr2.html

```
1 <button id="btn">add</button>
2 <div id="app"></div>
3
4
1 function show() {
2     let app = document.querySelector('#app');
3     let newNode = document.createElement('div');
4     let newNodeT = document.createTextNode('我在這');
5
6     newNode.setAttribute('class', 'box')
7     newNode.appendChild(newNodeT)
8     app.appendChild(newNode);
9 }
10 function init() {
11     document.querySelector('#btn').addEventListener('click', show)
12 }
13
14 window.addEventListener('load', init);
15
16
```

這裡我們比較一下append跟appendChild喔！

```
1 function show() {
2     let app = document.querySelector('#app');
3     let newNode = `我在這`;
4     app.append(newNode)//以Node物件或DOMstring(基本上是文字)的形式新增元素
5     //app.appendChild(newNode) // 接受一個Node物件//所以如果是文字他不接受喔！
6
7 }
8 function init() {
9     document.querySelector('#btn').addEventListener('click', show)
10 }
11
12 window.addEventListener('load', init);
13
```

## 加上移除

```
1 <button id="btn">add</button>
2 <button id="btnDelete">Delete</button>
3 <div id="app"></div>
4
5
```

```
1 let app = document.querySelector('#app');
2 //移除'我在這'
3 function bye() {
4     if(document.querySelectorAll('.box').length!=0){ //如果有'我在這'我才移除
5         app.removeChild(app.lastChild); //移除最後一個'我在這'
6     }
7 }
8 //新增'我在這'
9 function show() {
10    let newNode = document.createElement('div');
11    let newNodeT = document.createTextNode('我在這');
12    newNode.setAttribute('class', 'box')
13    app.appendChild(newNode).appendChild(newNodeT);
14 }
15 function init() {
16     document.querySelector('#btn').addEventListener('click', show); //新增
17     document.querySelector('#btnDelete').addEventListener('click', bye); //移除
18 }
19
20 window.addEventListener('load', init);
```

## 字串物件

| 屬性或方法                         | 說明                                                         | 例子                    |
|-------------------------------|------------------------------------------------------------|-----------------------|
| length                        | 字串長度                                                       | str.length            |
| charAt(索引值n)                  | 返回索引值n的字元，如果不在 0 ~ str.length - 1之間，則返回一個空字串。              | str.charAt(3)         |
| indexOf(字串s, 開始索引值n)          | 返回 s 在字串中首次出現的位置，從索引值 n 位置開始查詢，如果 s 不存在，則返回 -1。            | str.indexOf('l')      |
| lastIndexOf(字串s, 開始索引值n)      | 返回 s 在字串中最後出現的位置，從索引值 n 位置開始向前查詢，如果 s 不存在，則返回 -1。          | str.lastIndexOf('l')  |
| substring(索引值n, 索引值m)         | 返回從索引值 n 到索引值 (m - 1) 之間的字元。若結束索引值m省略，則表示從索引值 n 位置一直擷取到最後。 | str.substring(1)      |
| replace(字串r, 字串s)             | 將字串中的字串 r 替換成字串 s                                          | str.replace('a', 'e') |
| split(字串s, 個數n)               | 以字串 s 切割字串，並返回n個子字串的陣列                                     | str.split('@')        |
| trim()                        | 返回一個去除開頭與結尾的所有空白字元字串                                       | str.trim()            |
| toLowerCase() / toUpperCase() | 返回一個將字串轉換成小寫 / 大寫的字串                                       | str.toUpperCase()     |
| search(正規表示式)                 | 查詢字串與一個正規表示式是否匹配。成功，則返回首次匹配項的索引；否則，返回 -1                   | str.search(/java/)    |

👉 參閱 : 40\_string\_C.html

```
1 var str = "abcdeabcde";
2 var obj = new String(123);
3 document.write("length : ", str.length, "<br>"); //10
4 document.write("toUpperCase() : ", str.toUpperCase(), "<br>"); //ABCDEABCDE
5
6 document.write("typeof str : ", typeof str, "<br>"); // string
7 document.write("typeof obj : ", typeof obj, "<br>"); // object
8
9
10 document.write("indexOf('cd') : ", str.indexOf("cd"), "<br>"); //2
11 document.write("indexOf('cdd') : ", str.indexOf("cdd"), "<br>"); // -1 找不到
12
13 document.write("lastIndexOf('cd') : ", str.lastIndexOf("cd"), "<br>"); //7 找到最後的'cd'
14 document.write("lastIndexOf('cdd') : ", str.lastIndexOf("cdd"), "<br>"); // -1
15
16 document.write("match('cd') : ", str.match("cd"), "<br>"); //cd
17 document.write("match('cdd') : ", str.match("cdd"), "<br>"); //null
18
19 document.write("charAt(3) : ", str.charAt(3), "<br>"); //d
20
21 document.write("substr(2,5) : ", str.substr(2, 5), "<br>"); //cdeab 從陣列二 含自己五個
22 document.write("substr(2) : ", str.substr(2), "<br>"); //cdeabcde 從陣列二 含自己後面都有
23 document.write("substring(2,5) : ", str.substring(2, 5), "<br>"); //cde 從陣列二 含自己到陣列五之前
24
25 document.write("replace('de','QQQ') : ", str.replace("de", "QQQ"), "<br>"); //de被替換成QQQ
26 document.write("replace(/de/g,'QQQ') : ", str.replace(/de/g, "QQQ"), "<br>"); //全部的de都換成QQQ
```

```

1  var url = "www.lib.ncu.edu.tw";
2  var arr = url.split(".");
3  for (let i in arr) {
4      document.write(i, " : ", arr[i], "<br>");
5  }
6
7

```

## 時間物件

- 建立日期物件為今天的日期

```
var now = new Date();
```

- 建立日期物件("西元年,月-1,日")

```
var day1 = new Date(2020, 4, 21);
```

- 建立日期物件("西元年,月-1,日,時,分,秒")

```
var day2 = new Date(2000, 2, 21, 12, 10, 30);
```

| 屬性或方法         | 說明                                                         | 例子                         |
|---------------|------------------------------------------------------------|----------------------------|
| getTime()     | 回傳自1970/01/01至今的毫秒數                                        | now.getTime()              |
| getFullYear() | 回傳完整的西元年 (如：2021)                                          | now.getFullYear()          |
| getYear()     | 若介於1900~1999年間，<br>則回傳西元年的後兩碼，<br>否則回傳完整的西元年               | now.getYear()              |
| getMonth()    | 回傳月份 (0 ~ 11)                                              | now.getMonth()             |
| getDay()      | 回傳星期幾 (0 ~ 6) (0:星期天)                                      | now.getDay()               |
| getDate()     | 回傳日期 (1 ~ 31)                                              | now.getDate()              |
| getHours()    | 回傳小時 (0 ~ 23)                                              | now.getHours()             |
| getMinutes()  | 回傳分 (0 ~ 59)                                               | now.getMinutes()           |
| getSeconds()  | 回傳秒 (0 ~ 59)                                               | now.getSeconds()           |
| setTime(毫秒數)  | 重新設定日期物件的時間。<br>傳入自1970/01/01的毫秒數。<br>回傳自1970/01/01的毫秒數    | now.setTime(1572795693920) |
| setFullYear() | 重新設定日期物件的完整西元年 (如：<br>2004)                                | now.setFullYear(2004)      |
| setYear()     | 重新設定日期物件。<br>若介於1900~1999年間，<br>則需傳入西元年的後兩碼。<br>否則傳入完整的西元年 | now.setYear(23)            |
| setMonth()    | 重新設定日期物件的月份 (0 ~ 11)                                       | now.setMonth(3)            |
| setDate()     | 重新設定日期物件的日期 (1 ~ 31)                                       | now.setDate(23)            |
| setHours()    | 重新設定日期物件的小時 (0 ~ 23)                                       | now.setHours(1)            |
| setMinutes()  | 重新設定日期物件的分 (0 ~ 59)                                        | now.setMinutes(23)         |
| setSeconds()  | 重新設定日期物件的秒 (0 ~ 59)                                        | now.setSeconds(23)         |

👉 參閱 : 41\_date.html

```
1  /* new Date() */
2  var dayTime = new Date();
3  document.write("<b>new Date() 今天的 Date Object :</b><br/>" + dayTime + "<br/><br/>");
4
5  var dayTime2 = new Date('2021-01-22');
6  document.write("<b>new Date() 定義一個Date Object :</b><br/>" + dayTime2 + "<br/><br/>");
7
8
9  /* getFullYear */
10 document.write("<b>getFullYear 年份 :</b><br/>" + dayTime.getFullYear() + "<br/>");
11 /* getMonth */
12 // document.write(
13 //   "<b>getMonth 月份 :</b><br/>" + dayTime.getMonth() + "<br/>"
14 // );
15 document.write("<b>getMonth 月份 :</b><br/>" + (dayTime.getMonth() + 1) + "<br/>");
16 /* getDate */
17 document.write("<b>getDate 日期 :</b><br/>" + dayTime.getDate() + "<br/><br/>");
18
19 /* setHours、setMinutes、setSeconds */
20 dayTime.setHours(20);
21 dayTime.setMinutes(50);
22 dayTime.setSeconds(33);
23 document.write("<b>setHours、setMinutes、setSeconds 設定時間 :</b><br/>" + dayTime + "<br/><br/>");
```

## 練習

題目：彈窗

參閱：[alert\\_w\\_s.html](#)

```
function fs_alert() {  
    //打顯示你的彈窗  
    //按下確認可以關閉你的彈窗  
}  
  
function init() {  
    //為你的按鈕註冊事件  
}  
// 程式準備開始  
window.addEventListener('load', init);
```



題目：to do list

參閱：[todolist\\_s\\_1.html](#) / [todolist\\_s\\_2.html](#)

