

JavaScript

網頁程式設計

DARREN YANG 楊德倫

目次

1. 測試及除錯工具.....	1
2. 常數和變數宣告.....	8
3. 運算子.....	14
4. 字串.....	19
5. 取得標籤元素.....	21
6. 流程控制.....	26
7. Object 物件.....	35
8. Array 陣列	37
9. JSON.....	46
10. 函式的定義.....	49
11. 時間與計時器.....	57

課程範例網址：

https://github.com/telunyang/javascript_iii_bd

1. 測試及除錯工具

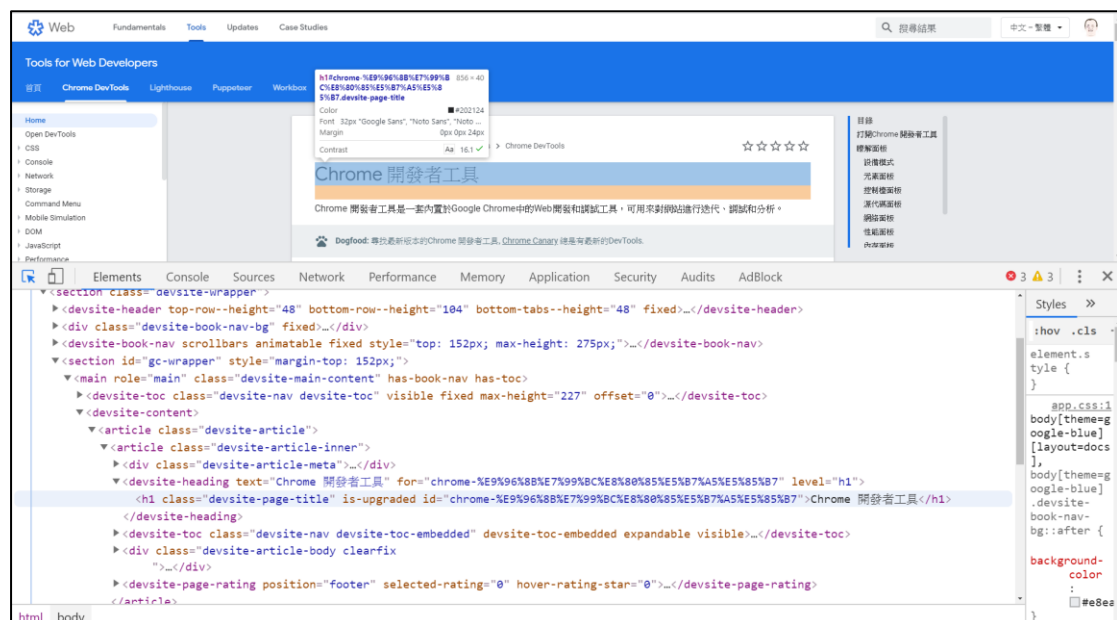
開啟開發工具

- F12

檢查元素

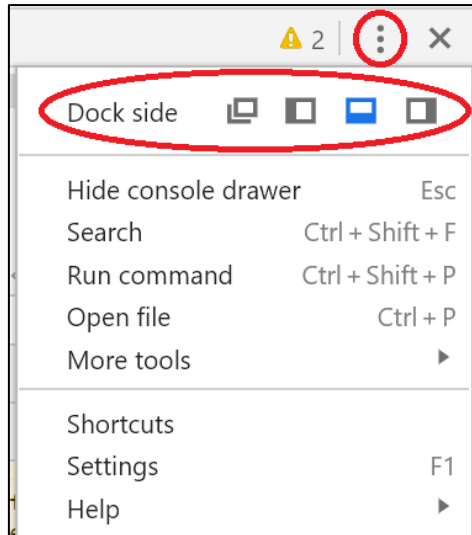


圖：檢查元素按鈕

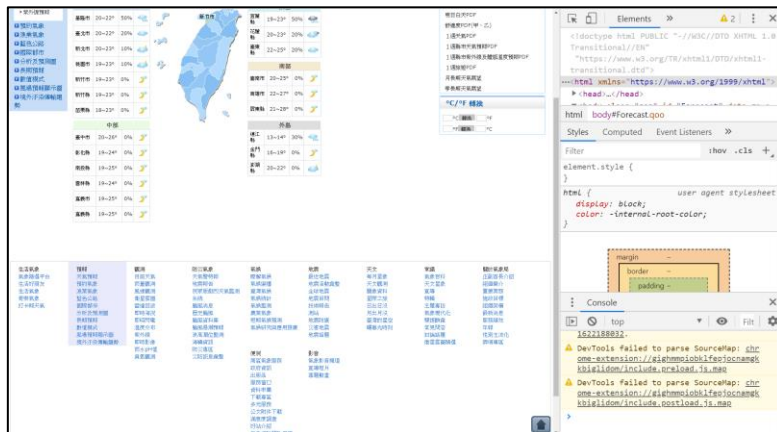


(圖) 檢查元素

開發工具的位置

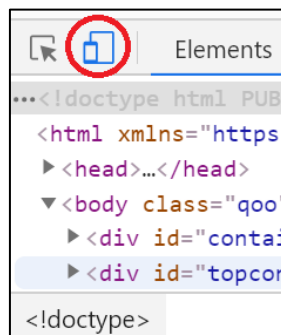


(圖) 按下三個點的圖示，也可以選擇 dock side

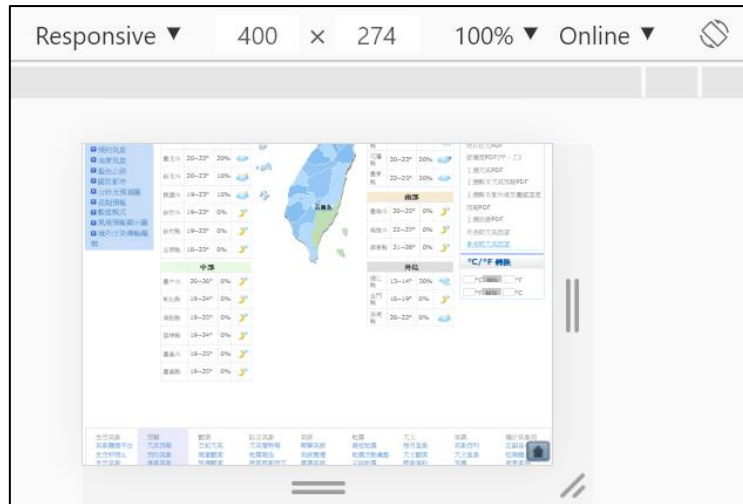


(圖) 切換 dock side，從下方到右側

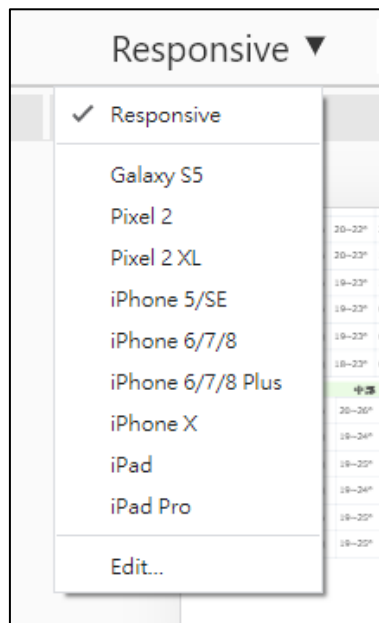
切換裝置



(圖) 等同按下切換裝置工具欄



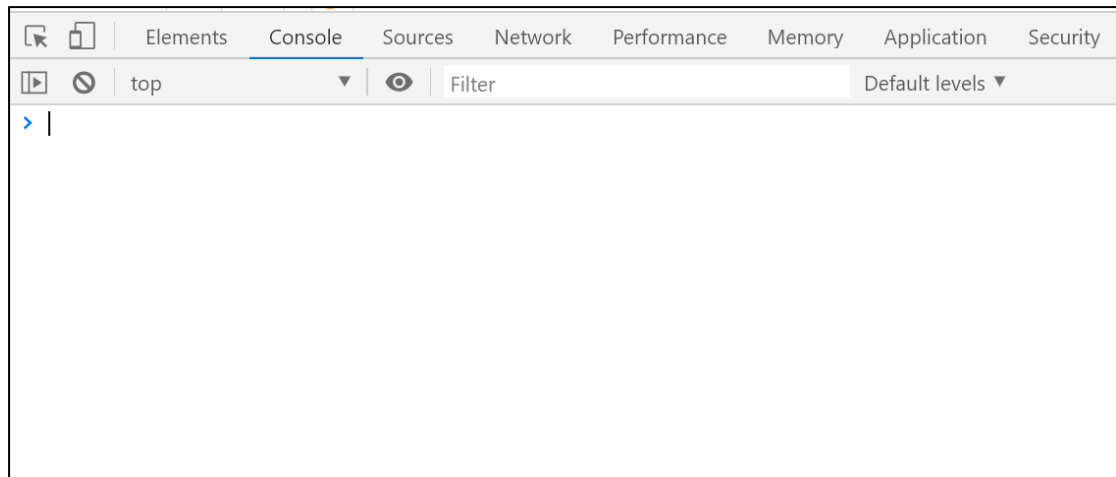
(圖) 可選擇不同的行動裝置，或自訂寬高，來顯示網頁



(圖) 選擇裝置來觀看網頁

Console 面板

我們可以使用 Console 面板，進行 JavaScript 的開發測試與除錯。



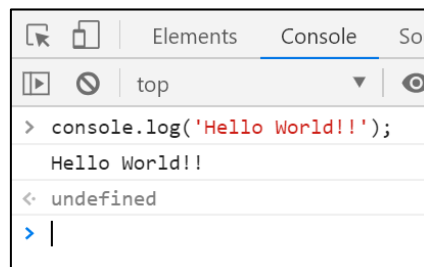
(圖) Console 面板

常用快速鍵：

- Ctrl + R 或 F5 刷新頁面
- Ctrl + F5 清除快取後，刷新頁面(重新從伺服器端請求下載 HTML)
- Ctrl + L 清除 Console
- Shift + Enter 在 Console 中斷行(或多行)

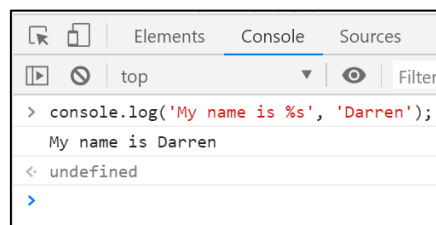
以下是使用的範例：

- 使用 console.log 直接輸出



(圖) 輸出結果

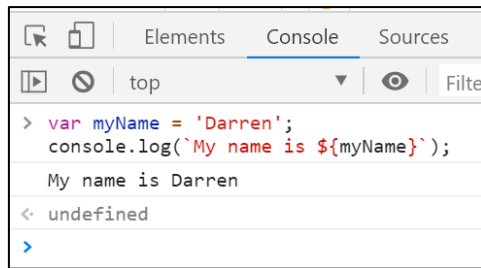
- 使用 %s 輸入字串



(圖) 第二個字串引數值，可以帶入 %s 當中

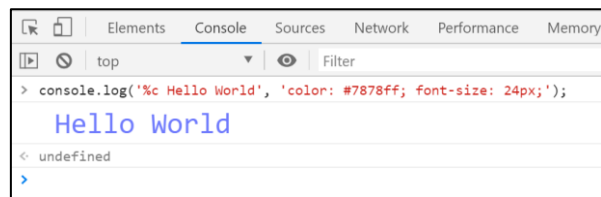
備註: 若需要換行，可以在輸入第一行以後，按下 Shift + Enter，即可換行

- 使用 es6 的 template strings



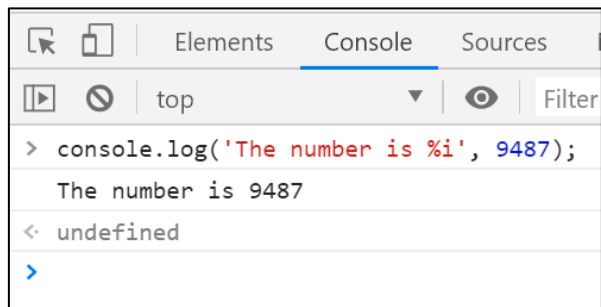
(圖) 透過 template 進行字串輸出

- 使用 %c 輸出時，加入樣式 (css style)



(圖) 設定 css style 的文字輸出

- 使用 %i 伴隨字串格式來輸出數字的值



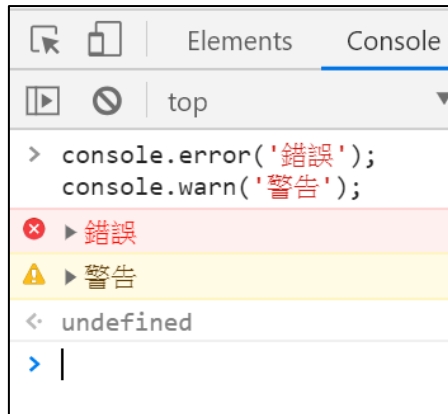
(圖) 輸出數字的值

常見的 % (格式化字串所使用的符號)：

- %s: 輸入字串
- %i 或 %d: 輸入數值
- %c: css style

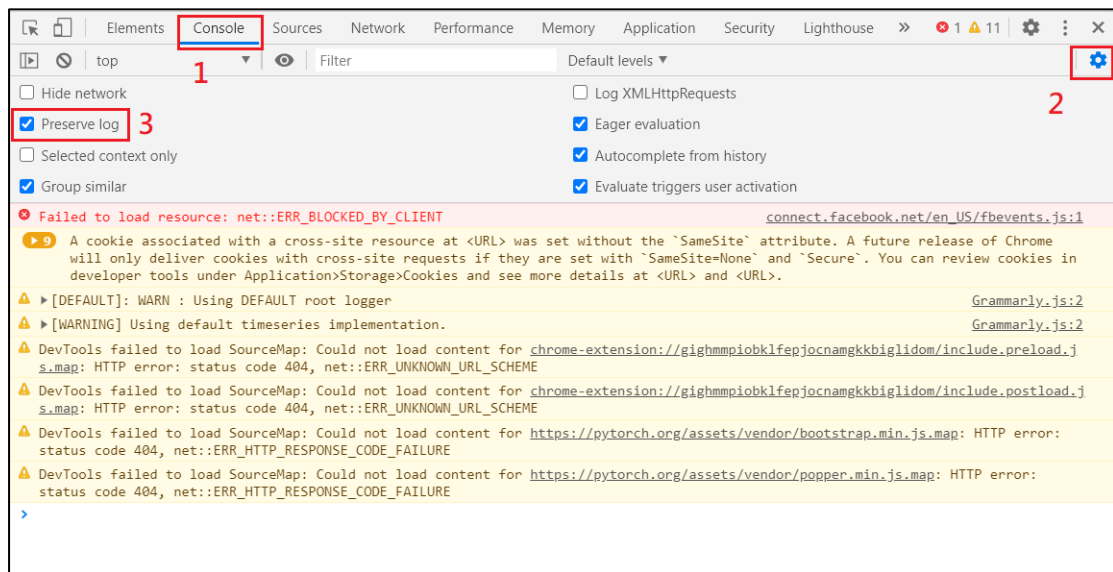
若有特殊的訊息(例如錯誤、警告等)要顯示出來，可以使用 console.error、console.warn：

- console.error() JavaScript 出現錯誤訊息時，會出現紅色的文字訊息與圖示
- console.warn() JavaScript 出現警告訊息時，會出現黃色的文字訊息與圖示



(圖) 顯示結果會包括圖示

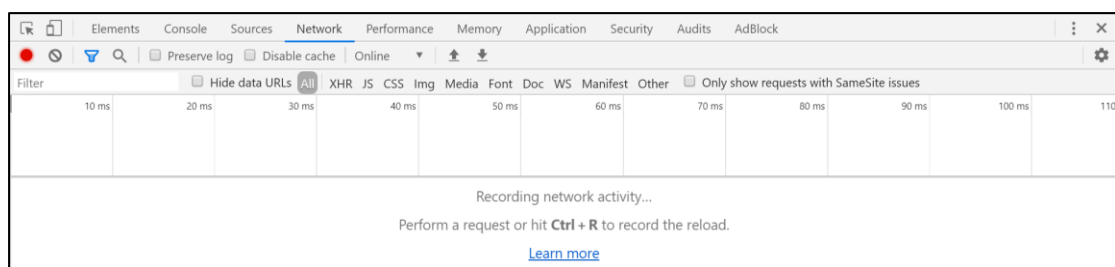
我們可以開啟 Preserve log，讓每次刷新頁面時，還能保有先前的輸出訊息。這個部分很重要，因為未來在使用非同步傳輸（Ajax、Fetch）結合 Web API 的時候，在資料傳輸上有問題，可以在刷新頁面、重新執行程式碼後，保留先前的輸出結果，讓我們在 debug 上更為便利。



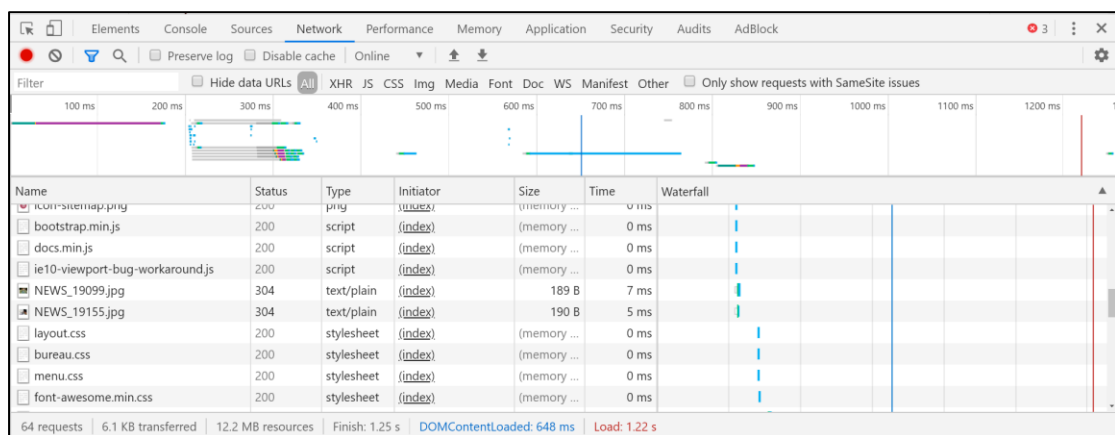
圖：開啟 Preserve log

Network 面板

Network 面板會顯示出所有網路請求的詳細訊息記錄，包括狀態、資源類型、大小、所需時間、HTTP request header 和 response header 等等，明確找出哪些請求比預期還要耗時，並加以調整，是優化網頁的重要工具。



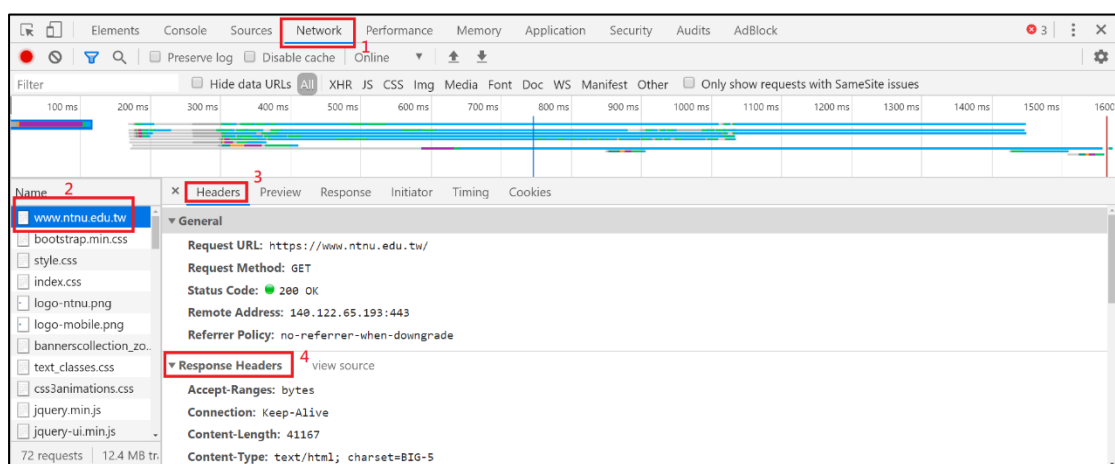
(圖) Network 面板會記錄任何的網路活動



(圖) 記錄網頁讀取的資訊與下載順序

我們可以透過 Headers，來了解網頁請求的狀況。開啟 Headers 的流程為：

1. 開啟 Network 面板
2. Ctrl + R 或是 F5 刷新頁面
3. 點選左側的檔案名稱
4. 點選 Headers



(圖) 觀看檔案的 Headers 內容

Request Headers (請求標頭, 參考維基百科)

標頭欄位	說明	範例
Accept	能夠接受的回應內容類型（Content-Types）。	Accept: text/plain
Cookie	之前由伺服器通過 Set-Cookie 傳送的一個超文字傳輸協定 Cookie	Cookie: _ga=GA1.3.1322956465.1572335045;loc ale=zh_TW; _gid=GA1.3.1110994946.1584940974; _gat_gtag_UA_141775379_1=1
Content-Type	請求多媒體類型（用於 POST 和 PUT 請求中）	Content-Type: application/x-www-form-urlencoded
User-Agent	瀏覽器的瀏覽器身分標識字串	User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/21.0

2. 常數和變數宣告

我們可以透過變數名稱的宣告，來為「值」（文字、數值、運算結果、暫存資料等）取個名字。我們使用關鍵字 `var`、`let`、`const` 來宣告變數，例如 `var myName`，而所謂「關鍵字」，是指在程式語言中具有意義的單詞，通常不適合拿來作為變數命名使用。原則上，變數的賦值，通常由右往左，我們稱之為「賦值運算子」（Assignment operators）。

變數命名的規則，有很多種方式，例如駱駝命名法、匈牙利命名法等。駱駝命名法是一種慣例，看起來像駱駝的駱峰，變數以小寫字母開頭，除了第一個單詞外，其它單詞的首個字母都大寫，例如 `numberOfCandies`，或是我們上方的 `myName`，或是 `numberOfDays`。課程中，原則上使用駱駝命名法，但沒有特別強制規定，你也可以自由選擇自己慣用的方式。

var、let 和 const

`var`、`let`、`const` 都是宣告變數的關鍵字，`var` 跟 `let` 都是宣告一般自訂變數，而 `const` 是用來宣告自訂常數（顧名思義，常數原則上不可修改它的值）。

範例 2-1.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    var myName = 'Darren';
    console.log('我的名字: ' + myName);

    let myAge = 18;
    console.log('我的年紀: ' + myAge);

    const PI = 3.1415926;
    console.log('圓周率 = ' + PI);
  </script>
</head>
<body>

</body>
</html>
```

const 關鍵字用於「常數」的宣告，有別於我們先前看到的「變數」，原則上不宜重覆賦值，也常用大寫的方法與變數加以區隔，例如 const MAX_LENGTH、DB_HOST、DB_PASSWORD 等。

識別字的規則

識別字 (identifier) 為程式語言中依程式需求自行定義的名稱，舉凡程式中所用的各種名稱都屬於識別字，內建物件 (built-in object) 及 DOM 其內所用的名稱也屬於識別字，自行定義名稱時應該避免與其相衝突，同時，識別字也不可與保留字 (reserved words) 的名稱相同。

JavaScript 定義識別字可用任何 Unicode 符號，但是習慣上仍是以英文二十六的大小寫字母為主，另加上數字、底線符號及 dollar sign，如下

_	\$											
a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z
A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9			

範例 2-2-1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    var _b;
    var $a;
    var $3;
    var 3cats;
  </script>
</head>
<body>

</body>
</html>
```

補充說明

我們另外補充說明「**保留字**」。在 JavaScript 中，有些識別字是保留給程式語言用的關鍵字，不能拿來作為變數名稱或是函式名稱。

我們應該避免使用 JavaScript 關鍵字：

abstract	arguments	boolean	break	byte
case	catch	char	class*	const
continue	debugger	default	delete	do
double	else	enum*	eval	export*
extends*	false	final	finally	float
for	function	goto	if	implements
import*	in	instanceof	int	interface
let	long	native	new	null
package	private	protected	public	return
short	static	super*	switch	synchronized
this	throw	throws	transient	true
try	typeof	var	void	volatile
while	with	yield		

我們應該避免使用 JavaScript 的物件、屬性和方法名稱：

Array	Date	eval	function	hasOwnProperty
Infinity	isFinite	isNaN	isPrototypeOf	length
Math	NaN	name	Number	Object
prototype	String	toString	undefined	valueOf

我們在之後會應用到 Window 物件，我們也要避免使用到它的屬性和方法：

alert	all	anchor	anchors	area
assign	blur	button	checkbox	clearInterval
clearTimeout	clientInformation	close	closed	confirm

constructor	crypto	decodeURI	decodeURIComponent	defaultStatus
document	element	elements	embed	embeds
encodeURIComponent	encodeURIComponent	escape	event	fileUpload
focus	form	forms	frame	innerHeight
innerWidth	layer	layers	link	location
mimeType	navigate	navigator	frames	frameRate
hidden	history	image	images	offscreenBuffering
open	opener	option	outerHeight	outerWidth
packages	pageXOffset	pageYOffset	parent	parseFloat
parseInt	password	pkcs11	plugin	prompt
propertyIsEnumerable	radio	reset	screenX	screenY
scroll	secure	select	self	setInterval
setTimeout	status	submit	taint	text
textarea	top	unescape	untaint	window

我們應該避免使用 HTML 的事件名稱：

onblur	onclick	onerror	onfocus
onkeydown	onkeypress	onkeyup	onmouseover
onload	onmouseup	onmousedown	onsubmit

var 和 let 的主要差異

Javascript 透過 var 關鍵字宣告的變數，具有抬升 (Hoisting) 的性質，不易掌握變數的生命週期，同時 var 所宣告的全域變數，會成為 window 物件的

屬性，我們日後會在課堂中提到。近年的 Javascript 版本，增加了 let 與 const 關鍵字，解決了 var 關鍵字的抬升問題（Hoisting）。

我們來看看 var 在區域變數中的特性：

一般來說，我們會這樣宣告變數、初始化變數（給變數初始值）、輸出變數內容

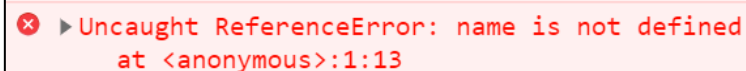
```
var name = 'Darren';  
console.log(name);  
//輸出 Darren
```

換個方式

```
console.log(name);  
var name = 'Darren';  
//這裡會輸出 undefined（Console 面板會不顯示輸出）
```

若是這樣子呢

```
console.log(name);  
let name = 'Darren';  
//應該輸出 undefined，卻拋出 ReferenceError: name is not defined 的訊息
```



（圖）參考錯誤: name 變數沒有被定義

這是因為 Javascript 在使用 var 宣告變數時，會將變數抬升（hoisting）到作用域（程式區塊）的前面，作用域內外的每一行都有機會使用到。

在 JavaScript 中有八種主要的型別：

- 三種基本型別：
 - 布林(Boolean)
 - 數值(Number)
 - 字串(String)
- 兩種複合的型別：
 - 陣列(Array)
 - 物件(Object)
- 兩種簡單型別：
 - 空值(null)

- 未定義(undefined)
- 一種特殊型別：
 - 函式(Function)

3. 運算子

算術運算子

JavaScript 可以進行加 (+)、減 (-)、乘 (*)、除 (/) 的基本運算，傳統的數學計算概念，也可以在程式當中運作，例如先乘除、後加減等等的概念；關於下面的基本運算，我們稱為「算數運算子」(Arithmetic operators)。

運算子	範例
加 (+)	$1 + 2 = 3$
減 (-)	$3 - 1 = 2$
乘 (*)	$3 * 4 = 12$
除 (/)	$8 / 2 = 4$
取餘數 (%)	$12 \% 5 = 2.$
遞增 (++)	假如 x 是 3，那 ++x 將把 x 設定為 4 並回傳 4，而 x++ 會回傳 3，接著才把 x 設定為 4。
遞減 (--)	假如 x 是 3，那 --x 將把 x 設定為 2 並回傳 2，而 x-- 會回傳 3，接著才把 x 設定為 2。
(一元運算子) 減號 (-)	假如 x 是 3，-x 回傳 -3。
(一元運算子) 加號 (+)	+"3" 回傳 3；+true 回傳 1。
指數運算子 (**)	$2 ** 3$ 回傳 8。

範例 3-1.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
```



```

    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        let x = 10;
        let y = 5;

        console.log(x + y); //15
        console.log(x - y); //5
        console.log(x * y); //50
        console.log(x / y); //2
        console.log(x % y); //0

        x++;
        console.log(x); //11

        y--;
        console.log(y); //4

        console.log(y**3); //64
    </script>
</head>
<body>

</body>
</html>

```

關係運算子

以相互比較的方式，來呈現布林邏輯運算結果，稱之為「關係運算子」或「比較運算子」，例如常常提到的「大於 >、大於等於 >=、等於 ==（或 ===）、小於 <、小於等於 <=、不等於 !=」等概念。

範例 3-2.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">

```

```
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Document</title>
<script>
    let price_a = 100;
    let price_b = 50;
    let price_c = 100;

    console.log(price_a > price_b); //true
    console.log(price_a < price_b); //false
    console.log(price_a == price_b); //false
    console.log(price_a >= price_b); //true
    console.log(price_a <= price_b); //false

    console.log(price_a > price_c); //false
    console.log(price_a < price_c); //false
    console.log(price_a == price_c); //true
    console.log(price_a >= price_c); //true
    console.log(price_a <= price_c); //true
</script>
</head>
<body>

</body>
</html>
```

補充說明

「==」 vs. 「===」 相等運算子

有時候閱讀程式設計教課書，在討論是否等價這件事，有著不同的寫法，如同上面的「==」與「===」，兩個的差別在於：

「x == y」：若是型態不相等，變數會先強制轉換成相同的型態，再進行嚴格比對。

```
console.log(1 == 1); // 輸出 true
```

```
console.log("1" == "1"); // 輸出 true
```

```
console.log(1 == "1"); // 輸出 true
console.log("1" == 1); // 輸出 true
```

「`===`」：兩個型態不相等，不轉換型態，直接嚴格比對。最常用的等價邏輯判斷方式，若為 `true`，意味著兩個值相同，類型也相等。

```
console.log(1 === 1); // 輸出 true
console.log("1" === "1"); // 輸出 true
console.log(1 === "1"); // 輸出 false
console.log("1" === 1); // 輸出 false
```

邏輯運算子

布林提供了 `true` (真) 與 `false` (假) 的判斷值，透過邏輯運算子 (`&&`、`||`、`!`) 來進行操作。

範例 3-3-1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    let a = true;
    let b = false;
    console.log(a && b);
    //輸出 false

    let x = true;
    let y = false;
    console.log(x || y);
    // 輸出 true

    let z = true;
    let result = !z;
    console.log(result);
```

```

        // 輸出 false
    </script>

</head>
<body>

</body>
</html>

```

賦值運算子的種類，常見的有以下幾種，提供給大家參考：

名稱	簡化後運算子	說明
賦值	<code>x = y</code>	<code>x = y</code>
加法賦值	<code>x += y</code>	<code>x = x + y</code>
減法賦值	<code>x -= y</code>	<code>x = x - y</code>
乘法賦值	<code>x *= y</code>	<code>x = x * y</code>
除法賦值	<code>x /= y</code>	<code>x = x / y</code>
餘數賦值	<code>x %= y</code>	<code>x = x % y</code>
指數賦值	<code>x **= y</code>	<code>x = x ** y</code>
左移賦值	<code>x <<= y</code>	<code>x = x << y</code>
右移賦值	<code>x >>= y</code>	<code>x = x >> y</code>
無號右移賦值	<code>x >>>= y</code>	<code>x = x >>> y</code>
位元 AND 賦值	<code>x &= y</code>	<code>x = x & y</code>
位元 XOR 賦值	<code>x ^= y</code>	<code>x = x ^ y</code>
位元 OR 賦值	<code>x = y</code>	<code>x = x y</code>

範例 3-3-2.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        //宣告變數
        let x = 20;
        let y = 10;
    </script>

```

```

        //將 x 加上 y 的結果，再賦值到 x 當中
        x += y; //可以寫成 x = x + y;
        console.log(x);

        //變數初始化
        x = 2, y = 8;
        y *= x; //可以寫成 y = y * x;
        console.log(y);

        //變數初始化
        x = 15, y = 4;
        x %= y; //可以寫成 x = x % y;
        console.log(x);
    </script>
</head>
<body>

</body>
</html>

```

4. 字串

字串的標示方式

- 透過兩個「'」將字串圍起來
- 透過兩個「"」將字串圍起來
- 透過 Template Literals 或 Template Strings 合併字串

範例 4-1.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">

```

```
<title>Document</title>
<script>
    //宣告字串
    let strName = 'Darren';

    //用單引號圍住字串的方式輸出
    console.log('Hello World! ' + strName + '...');

    //用雙引號圍住字串的方式輸出
    console.log("I love javascript! " + strName + "...");

    //用 template strings 的方式輸出
    console.log(`My name is ${strName} ...`);

    //輸出 你好! 我是 Darren Yang
    let fname = 'Darren';
    let lname = 'Yang';
    console.log(`你好! 我是 ${fname} ${lname}`);

    //若是嵌入變數計算，可以這麼做
    let a = 5;
    let b = 10;
    console.log(`${a + b}, ${2 * a + b}`);
</script>

</head>
<body>

</body>
</html>
```

字串的跳脫表示法

範例 4-2.html
<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta http-equiv="X-UA-Compatible" content="IE=edge"></pre>

```

    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        //跳脫字元(\)： 為了避免字串當中有單引號
        console.log('Did you talk to Mary\'s brother?');

        /**
         * 其它種類的跳脫字元
         * \t： 縮排，等同於 tab 鍵
         * \r： Enter (return)
         * \n： 斷行，代表建立新的一行(new line)
         */
        console.log("嗨!\t\t\t\t 近來好嗎?\n 好久不見");
    </script>

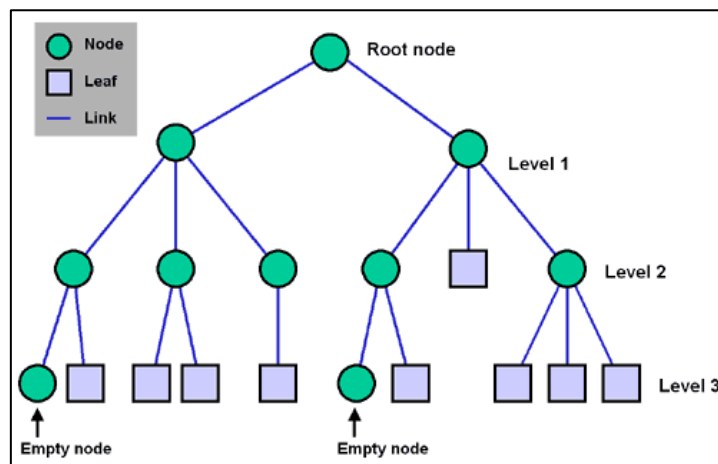
</head>
<body>

</body>
</html>

```

5. 取得標籤元素

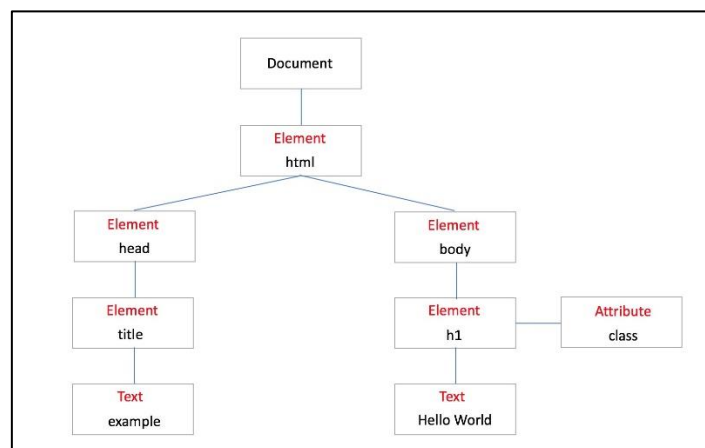
我們來解析一下 DOM。



(圖) DOM 的樹狀結構，裡面是由節點、樹葉、連結所組成

在 DOM 中，每個元素(element)、文字(text) 等等都是一個節點(node)，而節點通常分成以下四種：

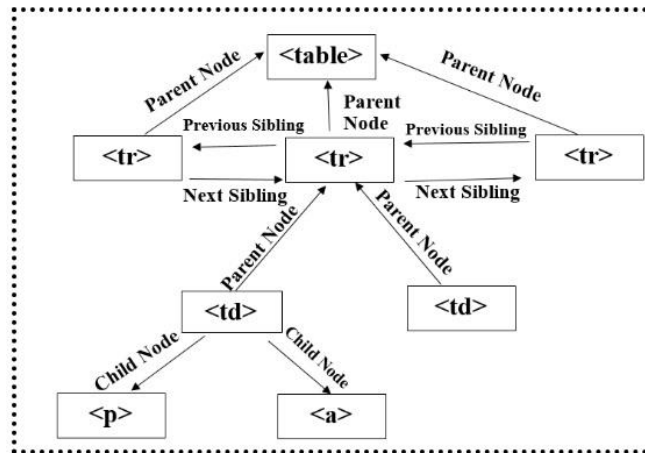
- Document
 - Document 就是指這份文件，也就是這份 HTML 檔的開端，所有的一切都會從 Document 開始往下進行。
- Element
 - Element 就是指文件內的各個標籤，因此像是 <div>、<p> 等等各種 HTML Tag 都是被歸類在 Element 裡面。
- Text
 - Text 就是指被各個標籤包起來的文字，舉例來說在 <h1>Hello World</h1> 中，Hello World 被 <h1> 這個 Element 包起來，因此 Hello World 就是此 Element 的 Text
- Attribute
 - Attribute 就是指各個標籤內的相關屬性，例如 class、id、data-* 等。



(圖) DOM 圖解

由於 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，這邊可以分成以下兩種關係：

- 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。



(圖) table 元素 - 節點間的關係

補充說明

window 物件（可以想成瀏覽器本身）代表了一個包含 DOM 文件的視窗，其中的 document 屬性指向了視窗中載入的 Document 物件：

- 代表所有在瀏覽器中載入的網頁，也是作為網頁內容 DOM 樹（包含如 <body>、<table> 與其它的元素）的進入點。
- 提供了網頁文件所需的通用函式，例如取得頁面 URL 或是建立網頁文件中新的元素節點等。

以下為常用的 DOM API：

- document.getElementById('idName')
 - 找尋 DOM 中符合此 id 名稱的元素，並回傳相對應的 element。
- document.getElementsByTagName('tag')
 - 找尋 DOM 中符合此 tag 名稱的所有元素。
- document.getElementsByClassName('className')
 - 找尋 DOM 中符合此 class 名稱的所有元素。
- document.querySelector(selectors)
 - 利用 selector 來找尋 DOM 中的元素，並回傳相對應的第一個 element。
- document.querySelectorAll(selectors)
 - 利用 selector 來找尋 DOM 中的所有元素。

補充說明		
選擇器	例子	例子描述
<u>.class</u>	. intro	選擇 class="intro" 的所有元素。
<u>.class1.class2</u>	. name1.name2	選擇 class 屬性中同時有 name1 和 name2 的所有元素。
<u>.class1 .class2</u>	. name1 .name2	選擇作為類名 name1 元素後代的所有類名 name2 元素。
<u>#id</u>	#firstname	選擇 id="firstname" 的元素。
<u>*</u>	*	選擇所有元素。
<u>element</u>	p	選擇所有 <p> 元素。
<u>element.class</u>	p.intro	選擇 class="intro" 的所有 <p> 元素。
<u>element,element</u>	div, p	選擇所有 <div> 元素和所有 <p> 元素。
<u>element element</u>	div p	選擇 <div> 元素內的所有 <p> 元素。
<u>element > element</u>	div > p	選擇父元素是 <div> 的所有 <p> 元素。
<u>[attribute]</u>	[target]	選擇帶有 target 屬性的所有元素。
<u>[attribute=value]</u>	[target=_blank]	選擇帶有 target="_blank" 屬性的所有元素。
<u>[attribute~=value]</u>	[title~=flower]	選擇 title 屬性包含單詞 "flower" 的所有元素。

補充說明		
選擇器	例子	例子描述
<u><code>[attribute^=value]</code></u>	<code>a[href^="https"]</code>	選擇其 href 屬性值以 "https" 開頭的每個 <a> 元素。
<u><code>[attribute\$=value]</code></u>	<code>a[href\$=".pdf"]</code>	選擇其 href 屬性以 ".pdf" 結尾的所有 <a> 元素。

參考資料：

CSS 選擇器參考手冊

https://www.w3school.com.cn/cssref/css_selectors.asp

querySelector()

範例 5-1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>
  <p class="myClass">問世間，</p>
  <p class="myClass">情是何物，</p>
  <p class="myClass">直教生死相許。</p>

  <script>
    //這裡是回傳第一個 p，是一種 element
    let p = document.querySelector('p');
    p.style.backgroundColor = '#7878ff';
  </script>
</html>
```

querySelectorAll()

範例 5-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>
  <p class="myClass">問世間，</p>
  <p class="myClass">情是何物，</p>
  <p class="myClass">直教生死相許。</p>

  <script>
    let p = document.querySelectorAll('p');

    //可以使用類似陣列的操作方式，提供索引值來存取
    p[0].style.backgroundColor = '#ff0000';
    p[1].innerHTML = '<span style="color: #00ff00">...情是何
物...</span>';
    p[2].style.lineHeight = '24px';
    p[2].style.backgroundColor = '#0000ff';
    p[2].style.color = '#ffffff';
  </script>
</body>
</html>
```

6. 流程控制

我們把條件與迴圈，稱之為「流程控制」或「控制結構」，對於任何程式，都扮演著重要的角色。它們讓你定義的特定條件，控制在何時、何種頻率來執行哪些部分的程式碼。

選擇敘述

if 陳述句、if ... else 陳述句、if...else 陳述句鍵，它是依據條件做二選一區塊執行時，所使用的陳述句：

範例 6-1-1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    let condition = false;
    if (condition) {
      //若條件為真，則執行這個區塊的程式碼
      console.log('Hello World!');
      document.write('Hello World!');
    } else {
      //若條件為假，則執行這個區塊的程式碼
      console.log('May you have a nice day!');
      document.write('May you have a nice day!');
    }
  </script>
</head>
<body>

</body>
</html>
```

範例 6-1-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```

<title>Document</title>
<script>
    let number = 7;
    if (number > 10) {
        console.log(1);
    } else if (number === 7) {
        console.log(2);
    } else if (number > 5) {
        console.log(3);
    } else if (number === 6) {
        console.log(4);
    }
</script>
</head>
<body>

</body>
</html>

```

範例 6-1-3.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        let bool = (1 > 2) ? true : false;
        console.log(bool);
        // 輸出 false

        let x = 3;
        let y = 4;
        let answer = (x > y) ? 'x is greater than y' : 'x is less tha
n y';
        console.log(answer);
    </script>
</head>
<body>
</body>
</html>

```

```
        // 輸出 x is less than y
    </script>
</head>
<body>

</body>
</html>
```

switch 判斷

範例 6-1-4.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        let number = 7;
        switch(number){
            case '7':
                console.log('string 7');
                break;

            case 7:
                console.log('number 7');
                break;

            case 'number':
                console.log('string number');
                break;

            default:
                console.log('string default');
        }
        // 輸出 number 7
    </script>
```

```

</head>
<body>

</body>
</html>

```

迴圈

while 迴圈結合了條件判斷的概念，符合條件，則繼續執行區塊內的程式，直到條件不成立，才跳出程式區塊。以下提供範例來說明：

while 迴圈

範例 6-2-1.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    let count = 1;
    while(count <= 7){
      console.log(count);
      count++;
    }
    // 輸出 1 2 3 4 5 6 7 (console.log 會斷行)
  </script>
</head>
<body>

</body>
</html>

```

for 迴圈

說明

```

for(init; condition; increment) {
  //statements
}

```



```
}
```

init 是變數初始宣告的地方

condition 是變數狀態與判斷條件

increment 是變數賦值的方式

```
for(let i = 0; i < 10; i++) {  
    console.log('The value is ' + i);  
}  
// 輸出 The value is 0 .... The value is 9
```

迴圈內部也可以使用迴圈，通稱為「巢狀迴圈」

```
for(let i = 1; i <= 9; i++) {  
    for(let j = 1; j <= 9; j++){  
        console.log(i + ' * ' + j + ' = ' + (i*j));  
    }  
}
```

for 的無限迴圈形式：

```
for(;;){  
    console.log('Hello World!');  
}
```

while 的無限迴圈形式

```
while(1){ // 將 1 改成 true 亦同  
    ...  
}
```

範例 6-2-2.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-  
scale=1.0">
```

```

<title>Document</title>
<script>
    for(let i = 0; i < 10; i++) {
        console.log('The value is ' + i);
    }
    // 輸出 The value is 0 .... The value is 9
</script>
</head>
<body>

</body>
</html>

```

範例 6-2-3.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        for(let i = 1; i <= 9; i++) {
            for(let j = 1; j <= 9; j++){
                //在 Console 面板輸出結果
                console.log(`${i} * ${j} = ${i*j}`);
            }
        }

        for(let i = 1; i <= 9; i++) {
            for(let j = 1; j <= 9; j++){
                //在網頁輸出結果
                document.write(`${i} * ${j} = ${i*j} &nbsp;`);
            }
            document.write('<br />');
        }
    </script>

```

```
</head>
<body>

</body>
</html>
```

break 和 continue

通常我們使用 break 來停止、跳出迴圈運作，直接往 for 或 while 迴圈 block 結尾以後的程式區塊繼續執行；使用 continue 直接跳往下一個索引值的步驟繼續執行。

範例 6-3-1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    //i 到 4 的時候，跳出迴圈
    for(let i = 1; i <= 9; i++) {
      if(i == 4){
        break;
      }
      console.log(`i = ${i}`);
    }

    /**
     * j 為 4 的時候，只跳出內部迴圈，
     * 但外部迴圈依然會繼續執行，
     * 只有內部迴圈的 j 走到 4 時候，
     * 自行跳出
     */
    for(let i = 1; i <= 9; i++) {
      for(let j = 1; j <= 9; j++){
        if(j == 4){
          break;
        }
      }
    }
  </script>

```

```

        }
        console.log(`i = ${i}, j = ${j}`);
    }
}
</script>
</head>
<body>

</body>
</html>

```

範例 6-3-2.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        let count = 0;
        while(count <= 9){
            //count 遞增到 5 的時候，跳出迴圈
            if(count == 5){
                break;
            }
            console.log(`count = ${count}`);
            count++; //千萬記得要遞增，不然會變成無限迴圈
        }

        //i 為 5 的時候略過，直接往下一個 i (即是 6) 繼續執行
        for(let i = 1; i <= 9; i++) {
            if(i == 5){
                continue;
            }
            console.log(`i = ${i}`);
        }
    }

```

```

    //count 為 5 的時候略過，直接往下一個 count (即是 6) 繼續執行
    count = 0;
    while(count <= 9){
        count++;
        if(count == 5){
            continue;
        }
        console.log(`count = ${count}`);
    }
</script>
</head>
<body>

</body>
</html>

```

7. Object 物件

物件使用字串作為屬性來存取不同元素，這個字串，叫作「鍵」(Key)或是屬性 (property)，它所指向的元素叫作「值」(Value)。通常把這兩個合在一起，稱為「鍵值對」(Key-Value pair)，最主要的目的，在於儲存更多特定對象的資訊

建立物件

範例

物件的宣告：

```
let obj = {};
```

也有人這麼寫

```
let obj = new Object();
```

範例 7-1.html

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```

<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Document</title>
<script>
    let cat = {
        legs: 4,
        name: 'Doraemon',
        color: 'blue',
    };

    //使用「[ ]」來存取屬性
    console.log( cat['legs'] ); // 輸出 4
    console.log( cat['name'] ); // 輸出 Doraemon

    //使用「.」來存取屬性或函式
    console.log( cat.legs ); // 輸出 4
    console.log( cat.color ); // 輸出 blue

    //增加物件屬性：
    cat['eye_color'] = 'black';
    cat.habbitt = 'plays with Nobita';

    //輸出物件內容
    console.log(cat);
</script>

</head>
<body>

</body>
</html>

```

for/in 迴圈

for 迴圈的形式，常見還有其它幾種。若是要取得物件/陣列當中的「索引（index）/鍵（key）」，可以使用「for(property/key/index in dataSet) {...}」（就是

for/in 可以查看 Object 類型物件的屬性和屬性值)。

範例 7-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    let obj = {
      fname: 'Darren',
      lname: 'Yang',
      age: null,
      lineId: 'telunyang'
    };
    for(let attr in obj) {
      console.log(`The property in object variable is ${attr}`)
    }
    //輸出 The property in object variable is fname
    //輸出 The property in object variable is lname
    //輸出 The property in object variable is age
    //輸出 The property in object variable is lineId
  </script>
</head>
<body>

</body>
</html>
```

8. Array 陣列

Array 類型的特點

在沒有使用陣列的情況下，我們需要這樣記錄資料：

```
let name1 = "Alex";  
let name2 = "Bill";  
let name3 = "Cook";  
let name4 = "Darren";  
.  
.  
.  
let name9999 = 'Somebody';
```

上面這種列表會變得很不好用，假設每一個人的名字都需要一張紙來記錄，這要浪費多少紙張？於是我們可以使用類似「清單」概念的陣列，將所有名字都記錄在同一張紙上，這樣就簡單多了。

建立陣列

範例

陣列的宣告：

```
let arr = [];
```

也有人這麼寫

```
let arr = new Array();
```

範例

宣告陣列時，建立初始值：

```
let arr = ['Alex', 'Bill', 'Cook', 'Darren'];
```

有時候為了排版好看，會將陣列中的元素，透過鍵盤的 Enter，讓每個元素對齊：

```
let arr = [  
    'Alex',  
    'Bill',  
    'Cook',  
    'Darren'  
];
```


很重要的觀念是，陣列中每一個值的索引（index），都是從「0」開始。

索引	0	1	2	3
值	'Alex'	'Bill'	'Cook'	'Darren'
概念	{0: 'Alex'}	{1: 'Bill'}	{2: 'Cook'}	{3: 'Darren'}

範例 8-1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    //一般來說，陣列的索引，從 [0] 開始，到 [n - 1] 結束。
    let arr = ['Alex', 'Bill', 'Cook', 'Darren'];
    console.log( arr[0] ); // 輸出 Alex
    console.log( arr[3] ); // 輸出 Darren
    console.log( arr[4] ); // 輸出 Undefined
  </script>
</head>
<body>

</body>
</html>
```

範例 8-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
```

```
<script>
//一、初始化陣列
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];

//設定（新增）元素：
listOfName[4] = 'Ellen'; // 指定索引位置來新增元素
console.log(listOfName);
//輸出 [ 'Alex', 'Bill', 'Cook', 'Darren', 'Ellen' ]

//二、重新初始化陣列
listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];

//使用 push()，將資料加到陣列尾端：
listOfName.push('Ellen');
listOfName.push('Fox');
console.log(listOfName);
//輸出 ["Alex", "Bill", "Cook", "Darren", "Ellen", "Fox"]

//三、重新初始化陣列
listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];

//修改元素：
listOfName[0] = 'Allen';
listOfName[2] = 'Carl';
console.log(listOfName[0]); // 輸出 Allen
console.log(listOfName[2]); // 輸出 Carl
console.log(listOfName);
// 輸出 [ 'Allen', 'Bill', 'Carl', 'Darren' ]

//四、重新初始化陣列
listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];

//刪除尾端元素，使用 pop()，將會刪除陣列尾端的資料：
```

```

listOfName.pop();
console.log(listOfName); // 輸出 [ 'Alex', 'Bill', 'Cook' ]
listOfName.pop();
console.log(listOfName); // 輸出 [ 'Alex', 'Bill' ]

//五、重新初始化陣列
listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];

//刪除尾端元素，並取出放置在新的變數中
let name = listOfName.pop();
console.log(name); // 輸出 Darren

//六、重新初始化陣列
listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];

//刪除陣列第一個元素，使用 shift():
listOfName.shift();
console.log(listOfName); // 輸出 [ 'Bill', 'Cook', 'Darren' ]

//七、重新初始化陣列
listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];

//有時候從尾端刪除的陣列資料，需要放在陣列前端，使用 unshift():
let name = listOfName.pop(); // 從陣列尾端刪除 Darren
listOfName.unshift(name); // 將刪除的陣列尾端資料放到陣列前端
console.log(listOfName); // 輸
出 [ 'Darren', 'Alex', 'Bill', 'Cook' ]
</script>
</head>
<body>

</body>
</html>

```

補充說明

二維陣列：

```
let arr = [  
  ['a0', 'a1', 'a2', 'a3'],  
  ['b0', 'b1', 'b2'],  
  ['c0', 'c1', 'c2', 'c3', 'c4'],  
];  
console.log( arr[0][1] ); // 輸出 a1  
console.log( arr[2][4] ); // 輸出 c4
```

它的概念如下表格：

二維陣列	0	1	2	3	4
0	a0	a1	a2	a3	
1	b0	b1	b2		
2	c0	c1	c2	c3	c4

左側索引代表每一列的陣列資料，上方索引代表每一列當中特定欄位的位置。

建立二維陣列：

```
let arr1d = [];  
for(let i = 1; i <= 9; i++){  
  //先建立一維陣列  
  arr1d.push(i);  
}  
  
let arr2d = [];  
for(let j = 1; j <= 9; j++){  
  //連續新增先前建立的一維陣列，便可成為二維陣列  
  arr2d.push(arr1d);  
}  
  
console.log(arr2d);
```

輸出:

```
[  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],  
  [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]  
]
```

範例 8-3.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta http-equiv="X-UA-Compatible" content="IE=edge">  
  <meta name="viewport" content="width=device-width, initial-  
scale=1.0">  
  <title>Document</title>  
  <script>  
    //二維陣列  
    let arr = [  
      ['a0', 'a1', 'a2', 'a3'],  
      ['b0', 'b1', 'b2'],  
      ['c0', 'c1', 'c2', 'c3', 'c4'],  
    ];  
  
    console.log( arr[0][1] ); // 輸出 a1  
    console.log( arr[2][4] ); // 輸出 c4  
  
    //建立二維陣列：  
    let arr1d = [];  
    for(let i = 1; i <= 9; i++){
```

```

        //先建立一維陣列
        arr1d.push(i);
    }

    let arr2d = [];
    for(let j = 1; j <= 9; j++){
        //連續新增先前建立的一維陣列，便可成為二維陣列
        arr2d.push(arr1d);
    }
    console.log(arr2d);
</script>
</head>
<body>

</body>
</html>

```

補充說明：

- 陣列、字串具有「**序列化**」的特性，例如陣列 `arr = ["Alex", "Bill"]`；可透過 `arr[0]` 來取得 "Alex"；字串 `str = "人生好難"`；可透過 `str[2]` 來取得「好」。
- 針對這種序列化的結構，若是要搜尋當中是否有相等的值，可以使用「**.indexOf()**」，來與序列化資料當中的值進行比較，若有相等的值，則回傳該值位於序列化結構的「索引」（0 到 n-1，找不到的話，回傳 -1）。

範例 8-4.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        //宣告一個陣列
        let arr = ["Alex", "Bill"];
    </script>

```

```

    //宣告一個字串
    let str = "人生好難";

    //判斷 Bill 是否存在於 arr 當中
    if( arr.indexOf("Bill") !== -1 ){
        alert(`有找到 Bill`);
    } else {
        alert(`沒找到 Bill ...`);
    }

    //判斷「好」是否存在於 str 當中
    if( str.indexOf("好") !== -1 ){
        alert(`有找到「好」`);
    } else {
        alert(`沒找到「好」 ...`);
    }
</script>
</head>
<body>

</body>
</html>

```

for/of

透過迴圈取得 array 內部的值。

範例 8-5.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        //初始化陣列
        let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
    </script>

```

```

        //透過 for / of 的格式，迭代取得陣列各個元素所代表的值
        for(let value of listOfName){
            console.log(`${value}`);
        }
    </script>
</head>
<body>

</body>
</html>

```

9. JSON

JSON (JavaScript Object Notation, JavaScript 物件表示法，讀作/¹dʒeɪ sən/), 是一種輕量級的資料交換語言，該語言以易於讓人閱讀的文字為基礎，用來傳輸由屬性、值組成的資料物件。JSON 資料格式與語言無關。即便它源自 JavaScript，但目前很多程式語言都支援 JSON 格式資料的生成和解析。JSON 的官方 MIME 類型是 application/json，副檔名是 .json。

大致要注意的是 JSON：

- 名稱為字串，必須用"雙引號"包起來。
- 值可以是"雙引號"包括的字串，或者是數字、布林值、null、物件、陣列。

JSON 轉換

說明

將物件轉成 json

```
let strJson = JSON.stringify( objJson );
```

將 json 轉成物件 (將 json 字串轉換成 JavaScript 的原生類型)

```
let objJson = JSON.parse(strJson);
```

JSON.parse()

範例 9-1.html

```

<!DOCTYPE html>
<html lang="en">
<head>

```



```
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Document</title>
<script>
//輸出 json 內容
let str = `{
  "firstName": "Darren",
  "lastName": "Yang",
  "gender": "male",
  "age": 18,
  "address": {
    "streetAddress": "台北市復興南路一段 390 號 2 樓、3 樓",
    "city": "Taipei"
  },
  "phoneNumber": [
    {
      "type": "office",
      "number": "6631-6666"
    },
    {
      "type": "fax",
      "number": "6631-6598"
    }
  ]
}`;

//將 json 轉成物件
let obj = JSON.parse(str);

//讀取 obj 物件屬性 firstName
console.log(obj['firstName'])

//讀取 obj 物件屬性 address 裡面的物件屬性 streetAddress
console.log(obj['address']['streetAddress']);

//將 obj 物件屬性 phoneNumber 當中的陣列資料（裡面是物件）逐一列出
```

```

    for(let o of obj['phoneNumber']){
        console.log('=====');
        console.log(o['type']);
        console.log(o['number']);
    }
</script>
</head>
<body>

</body>
</html>

```

JSON.stringify()

範例 9-2.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        //物件初始化
        let objPerson = {
            name: "Bill",
            age: 25,
            hairColor: 'black',
            skinColor: 'beige'
        };

        //新增屬性
        objPerson.hadShoes = true;
        objPerson.isWoken = true;

        //修改屬性
        objPerson.age = 26;
    </script>

```

```

        //刪除屬性
        delete objPerson.skinColor;

        //將物件轉成 JSON
        let strJson = JSON.stringify(objPerson);

        //顯示 JSON 內容
        console.log(strJson);
    </script>
</head>
<body>

</body>
</html>

```

10. 函式的定義

函式（Function，又稱函數），是把程式碼集合在一起，以便能夠重複使用它們的一種方法。原則上，函式是有名字的（函式名稱）。

基本型

範例 10-1.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        //一、建立基本函式
        function say() {
            console.log('Hello World!');
        }
    </script>

```

```
//執行函式
say();

//二、帶有參數的函式
function greet01(name){
    console.log('Hello, ' + name);
}

//執行函式
greet01('Alex');

//三、帶有多個參數的函式
function greet02(greeting, name){
    console.log(greeting + ', ' + name);
}

//執行函式
greet02('Hi', 'Alex');

//四、有回傳值的函式
function getMessage01(){
    return 'Good job!';
}

//回傳函式執行結果
console.log( getMessage01() );

//五、帶有參數，同時回傳值的函式
function getMessage02(name){
    return 'Good job! ' + name;
}
```

```

        //回傳函式執行結果
        console.log( getMessage02('Bill') );
    </script>
</head>
<body>

</body>
</html>

```

回呼函式 (call back)

稱為回呼函數、回調函式，為一種「延續傳遞風格」(Continuation-passing style) 的函式程式寫法，它的對比的是前面我們所提供的基本函式範例（直接風格，Direct style）。回呼函數可以將特定函式作為傳遞參數，在該函式中呼叫執行，將原本應該在該函式中回傳的值，交給下一個函式來執行。

範例

建立一個 say 函式，其中的第二個參數也是函式：

```

//主程式
say('Darren Yang', function(result){
    console.log(result);
});

//建立 say 函式
function say(name, callback_function){
    //say 函式會處理特定程式碼後，透過 callback_function 把結果或訊息回傳到主程式
    callback_function(`Hi, [${name}] ... how have you been ?`);
}

```

1. 主程式呼叫 say 函式的時候，除了第 1 個參數 name，在第 2 個參數放置一個 callback_function 函式，一起傳遞到 say 函式。
2. say 程式區塊中，使用主程式傳遞到 say 函式當中的 name 跟 callback_function 參數。
3. 經過處理，將結果作為 callback_function 函式的參數，再透過 callback_function 送回主程式，變成主程式的第二個參數。
4. 此時 callback_function 展開變成一般的函式「function(result) { ... }」，此時該函式的「result」就是在 say 函式中處理的結果，被 callback_function 作為參數，帶回主程式。

流程

1. 想像主程式原先的樣子：

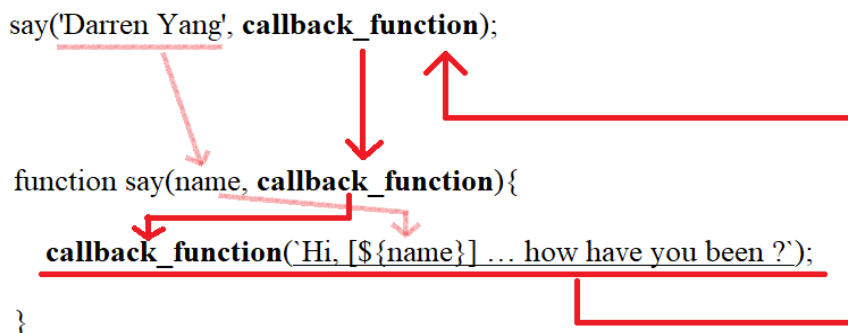
```
say('Darren Yang', callback_function);
```

2. 想像建立 say 函式的樣子：

```
function say(name, callback_function){  
    callback_function(`Hi, [${name}] ... how have you been ?`);  
}
```

3. 經過一連串的傳遞與處理：

```
say('Darren Yang', callback_function);  
  
function say(name, callback_function){  
    callback_function(`Hi, [${name}] ... how have you been ?`);  
}
```

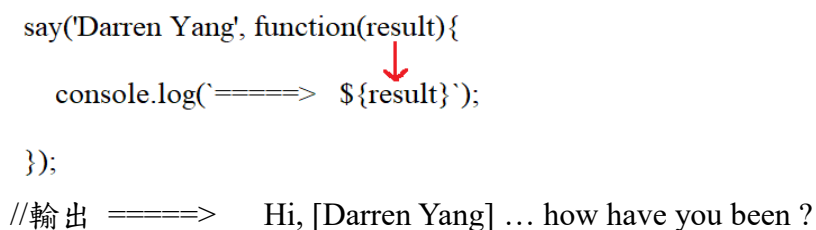


4. 主程式變成這個樣子，result 是「Hi, xxx ... how have you been ?」的文字處理結果，可以在「{ ... }」當中使用：

```
say('Darren Yang', function(result){ ... });
```

5. 我們可以自訂「function(result) { ... }」的內容：

```
say('Darren Yang', function(result){  
    console.log('=====>  ${result}');  
});  
  
//輸出 =====>   Hi, [Darren Yang] ... how have you been ?
```



使用回呼函數的目的，在於確保程式運作流程的明確性（另一種說法是指移交程式執行的控制權），例如一個回呼函式用於讀取檔案內容，主程式為了確保檔案內容被完整讀取出來，使用回呼函數，等待回呼函數執行完畢後，再透過主程式帶入的函式參數，將檔案內容回傳到主程式當中。

範例 10-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    //建立回呼函式
    function say(name, callback_function){
      callback_function(`Hi, [${name}] ... how have you been ?`);
    }

    //執行回呼函式
    say('Darren', function(str){
      alert(str);
    });
  </script>
</head>
<body>

</body>
</html>
```

匿名函式

把一個函數複製給變數，而這個函數是沒有名字的，即匿名函數。

範例 10-3.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
```

```
//基本跟一般函式無異，只是把函式帶到變數中，變數即為函式名稱
let say01 = function() {
    console.log('Hello World!');
}
say01();

//帶參數
let say02 = function(name) {
    console.log('Hello, ' + name);
}
say02('Alex');

//有回傳值
let say03 = function(name) {
    return 'Hello, ' + name;
}
console.log( say03('Bill') );
</script>
</head>
<body>

</body>
</html>
```

補充說明
<p>IIFE (Immediately Invoked Function Expression) 是一個定義完馬上就執行的 JavaScript function 。</p> <p>IIF 起手式</p> <pre>(function(){ console.log('Hello World!'); })();</pre> <p>帶參數的 IIFE</p> <pre>(</pre>


```
function (name) {  
    alert('Good job! ' + name);  
}  
>('Darren');
```

他又稱為 Self-Executing Anonymous Function，也是一種常見的設計模式，包含兩個主要部分：第一個部分是使用 Grouping Operator () 包起來的 anonymous function。這樣的寫法可以避免裡面的變數污染到 global scope。

第二個部分是馬上執行 function 的 expression ()，JavaScript 引擎看到它就會立刻轉譯該 function。

範例 10-4.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-  
scale=1.0">  
    <title>Document</title>  
    <script>  
        //顯示在 Console 面板  
        (  
            function () {  
                console.log('Hello World!');  
            }  
        )();  
  
        //顯示在 <body> 當中  
        (  
            function () {  
                document.write('Good job!');  
            }  
        )();  
  
        //跳出訊息  
        (  

```

```

        function (name) {
            alert('Good job! ' + name);
        }
    )('Darren');
</script>
</head>
<body>

</body>
</html>

```

箭頭函式

範例 10-5.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <script>
        //匿名函式風格的箭頭函式
        let say = () => {
            console.log('Hello World!');
        }
        say();

        //帶參數的箭頭函式
        let greet = (name) => {
            return 'Hello, ' + name;
        };
        console.log( greet('Bill') );

        //帶兩個參數的箭頭函式
        let sum = (num1, num2) => num1 + num2;

```

```

        console.log( sum(10, 20) );

        //帶參數，未加 () 的箭頭函式
        let getValue = value => value + ',' + value;
        console.log( getValue('Ha') );

        //箭頭函式版本 IIFE
        (() => {
            let strName = "Beryl";
            console.log(`Hello, ${strName}`);
        })();
    </script>
</head>
<body>

</body>
</html>

```

11. 時間與計時器

Date 物件

在 Javascript 的互動應用中，經常使用「Date 物件」當中的類時間函式，來判斷、比較、計算時間的差距，或是使用特定的時間格式，例如 timestamp、ISO 8601 等。

範例 11-1.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>

```

```

<script>
    //宣告 Date 物件：
    let date = new Date();

    //字串變數，預設空值
    let strTime = '';

    //串接時間
    strTime += date.getFullYear() // 2021，西元年
    strTime += date.getMonth() + 1 // 月份，5 月時執行會出現 4，記得加 1
    strTime += date.getDate() // 26，當月 26 號
    strTime += date.getHours(); // 幾時，例如 14 時
    strTime += date.getMinutes(); // 幾分，例如 58 分
    strTime += date.getSeconds(); //幾秒，例如 53 秒

    //輸出結果
    console.log(strTime);
</script>
</head>
<body>

</body>
</html>

```

補充說明

若是時間物件的函式（方法）回傳，只有一位數的結果（例如六月只回傳 6，而非 06），我們可以用一些小技巧來輸出自訂的格式。

```

// getMonth() 範圍 0 到 11，所以要加 1
let month = (new Date().getMonth() + 1);

// 數值小於 10，則左側加個 '0'
if( month < 10 ){
    month = '0' + month;
}

console.log(month); // 輸出 05

```

範例 11-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
let date = new Date();

let year, month, day, hour, minute, second;

// 2021, 西元年
year = date.getFullYear()

// 月份, 5 月時執行會出現 4, 比正常值少 1, 記得加 1
if( (date.getMonth() + 1) < 10 ) {
  month = '0' + (date.getMonth() + 1);
} else {
  month = (date.getMonth() + 1);
};

// 25, 當月 25 號
if( date.getDate() < 10 ) {
  day = '0' + date.getDate();
} else {
  day = date.getDate();
};

// 幾時, 例如 14 時
if( date.getHours() < 10 ) {
  hour = '0' + date.getHours();
} else {
  hour = date.getHours();
};
```

```

// 幾分，例如 58 分
if( date.getMinutes() < 10) {
    minute = '0' + date.getMinutes();
} else {
    minute = date.getMinutes();
};

//幾秒，例如 53 秒
if( date.getSeconds() < 10) {
    second = '0' + date.getSeconds();
} else {
    second = date.getSeconds();
};

console.log(`${year}-${month}-${day} ${hour}:${minute}:${second}`);
</script>
</head>
<body>

</body>
</html>

```

setTimeout 用法

用於在指定的毫秒數後，執行 1 次函式，後方參數會作為函式參數使用：

```
setTimeout(函式, 毫秒數[, 參數 1, 參數 2, ... 參數 N]);
```

```
let instanceId = setTimeout(函式, 毫秒數[, 參數 1, 參數 2, ... 參數 N]);
```

例如

```

setTimeout( function () {
    alert('Hello World');
}, 3000 );

```

範例 11-3-1.html

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Document</title>
<script>
//三秒後跳出訊息 Hello World
setTimeout( function () {
    alert('Hello World');
}, 3000 );

//自訂函式後，使用 setTimeout 來調用自訂函式
function getMessage01(){
    alert('Good job!');
}

//十秒後跳出訊息 Good job!
setTimeout(getMessage01, 10000);

//也可以自帶參數
function getMessage02(greeting, name){
    alert(`${greeting}, ${name}`);
}
setTimeout(getMessage02, 18000, 'Hi', 'Darren');
</script>
</head>
<body>

</body>
</html>
```

setInterval 用法

與 setTimeout 不同（僅執行 1 次），setInterval 方法可以週期性地（一樣指定毫秒）執行自訂的函式。

執行 setInterval()

```
setInterval(function(){ alert("Hello"); }, 3000);
```

執行 setInterval()，並返回實體 id

```
let instanceId = setInterval(function(){ alert("Hello"); }, 3000);
```

停止 setInterval

```
clearInterval(myVar);
```

範例 11-3-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>
  <button onclick="startCount()" id="btn_start">開始累加</button>
  <input type="text" id="txt" value="">
  <button onclick="stopCount()" id="btn_end">結束累加</button>

  <script>
    //累加數字的變數
    let countNum = 0;

    //作為 clearInterval() 用的變數
    let t;

    //累加用的函式
    function count() {
      //將當前的 count 值，放到 id=txt 的元素 value 屬性當中
      document.querySelector("input#txt").value = countNum;

      //進行數字的累加
      countNum = countNum + 1;
```



```

    }

    //開始累加
    function startCount() {
        //週期性在指定毫秒後，進行累加
        t = setInterval(function(){ count() }, 1000);

        //將開始按鈕設定為不可用
        document.querySelector("button#btn_start").setAttribute("disabled", "");

        //將結束鈕按移除不可用的屬性
        document.querySelector("button#btn_end").removeAttribute("disabled");
    }

    //結束累加
    function stopCount() {
        //結束 setInterval
        clearInterval(t);

        //將開始鈕按移除不可用的屬性
        document.querySelector("button#btn_start").removeAttribute("disabled");

        //將結束按鈕設定為不可用
        document.querySelector("button#btn_end").setAttribute("disabled", "");
    }
</script>
</body>
</html>

```

Math 物件

Math 是一個擁有數學常數及數學函數（非函式物件）屬性及方法的內建物件。

屬性	說明
Math.PI	一個圓的圓周和其直徑比值，約為 3.14159。

方法	說明
Math.abs(x)	回傳 x 的絕對值。

Math.ceil(x)	回傳不小於 x 的最小整數值。
Math.floor(x)	回傳不大於 x 的最大整數值。
Math.log10(x)	回傳以 10 為底，x 的對數值。
Math.log2(x)	回傳以 2 為底，x 的對數值。
Math.pow(x, y)	回傳 x 的 y 次方，也就是 xy 。
Math.random()	回傳一個 0 到 1（不足 1）之間的隨機值。
Math.round(x)	回傳 x 的四捨五入值。
Math.sqrt(x)	回傳 x 的正平方根。

範例 11-3-3.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script>
    //隨機數
    console.log( Math.random() );
    console.log( Math.random() );
    console.log( Math.random() );
    console.log( Math.random() );

    //隨機選擇餐廳
    let arrMeal = [ '麥當勞', '肯德基', '摩斯漢堡', '頂呱呱', '漢堡王' ];
    let idxRandom = Math.floor( Math.random() * 5 );
    document.write(`我可以選擇的餐廳有: <br />`);

    for(let i = 0; i < arrMeal.length; i++){
      document.write('● ' + arrMeal[i] + '<br />');
    }

    document.write(`今天中餐，我選擇 ${arrMeal[idxRandom]}`);
  </script>
</head>
<body>

```

```
</body>
```

```
</html>
```