

# Intro to JavaScript

**Ryan Chung**





# 概要


- 語法簡介
- 瀏覽器Console輸出
- 函數
- 物件

# JavaScript 簡介


- 小道消息
  - JavaScript語言是由Brendan Eich在Netscape工作時所設計出來的，聽說他只花了11天的時間就完成。
- 微軟很早就開始支持JS!
  - IE3版本(1996年)
- 分工角色
  - HTML 畫面元件結構
  - CSS 樣式、排版
  - JavaScript 互動、程式邏輯




# 開發環境


**Visual Studio Code**

[Docs](#)
[Updates](#)
[Blog](#)
[API](#)
[Extensions](#)
[FAQ](#)




[Download](#)

Version 1.42 is now available! Read about the new features and fixes from January.

## Code editing. Redefined.

Free. Built on open source. Runs everywhere.

[Download for Windows](#)  
Stable Build

[Other platforms and Insiders Edition](#)

By using VS Code, you agree to its [license and privacy statement](#).

EXTENSIONS: MARKETPLACE

@sort:installs

Python

2019.6.24221

4.9M

4.5

Linting, Debugging (multi-threaded, ...)

Microsoft

Install

GitLens — Git sup...

9.8.5

23.1M

5

Supercharge the Git capabilities buil...

Eric Amodio

Install

C/C++

0.24.0

23M

3.5

C/C++ IntelliSense, debugging, and ...

Microsoft

Install

ESLint

1.9.0

21.9M

4.5

Integrates ESLint JavaScript into VS ...

Dirk Baeumer

Install

Debugger for Ch...

4.11.6

20.6M

4

Debug your JavaScript code in the C...

Microsoft

Install

Language Supp...

0.47.0

18.7M

4.5

Java Linting, Intellisense, formatting, ...

Red Hat

Install

vscode-icons

8.8.0

17.2M

5

Icons for Visual Studio Code

VSCo Icons Team

Install

Vetur

0.21.1

17M

4.5

Vue tooling for VS Code

Pine Wu

Install

C#

1.21.0

15.6M

4

C# for Visual Studio Code (powered ...)

Microsoft

Install

File

Edit

Selection

View

Go

Debug

Terminal

Help

src > JS

serviceWorker.js

create-react-app - Visual Studio Code - In...

JS App.js

JS index.js

JS serviceWorker.js

```

39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
        
```

product

productSub

removeSiteSpecificTrackingException

removeWebWideTrackingException

requestMediaKeySystemAccess

sendBeacon

serviceWorker (property) Navigator.serviceWork...

storage

storeSiteSpecificTrackingException

storeWebWideTrackingException

userAgent

vendor

```

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
        
```

1: node

You can now view create-react-app in the browser.

Local:

http://localhost:3000/

On Your Network:

http://10.211.55.3:3000/

Note that the development build is not optimized.

master

0

0

0

Ln 43, Col 19

Spaces: 2


UTF-8

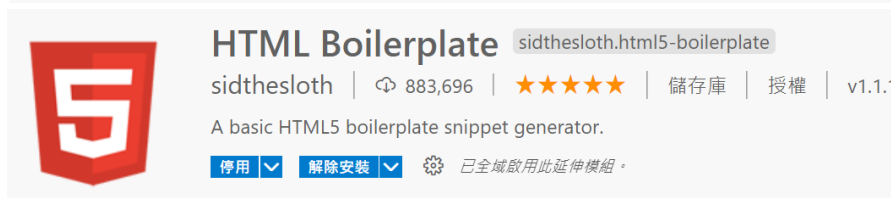
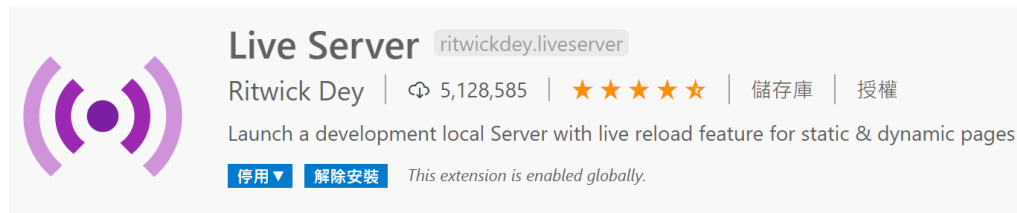
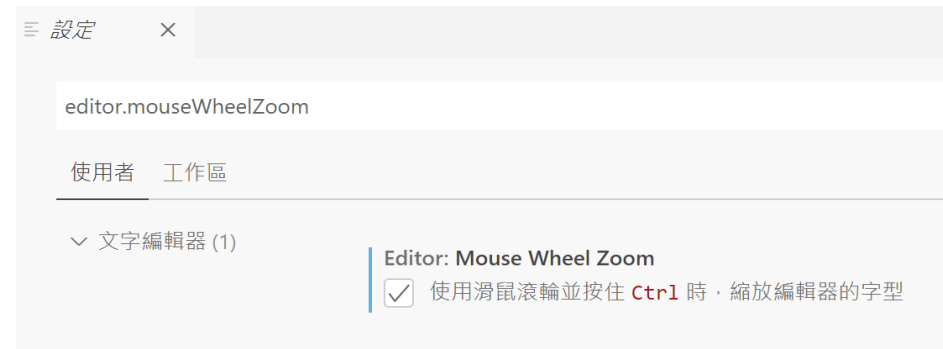
LF

JavaScript

<https://code.visualstudio.com/>

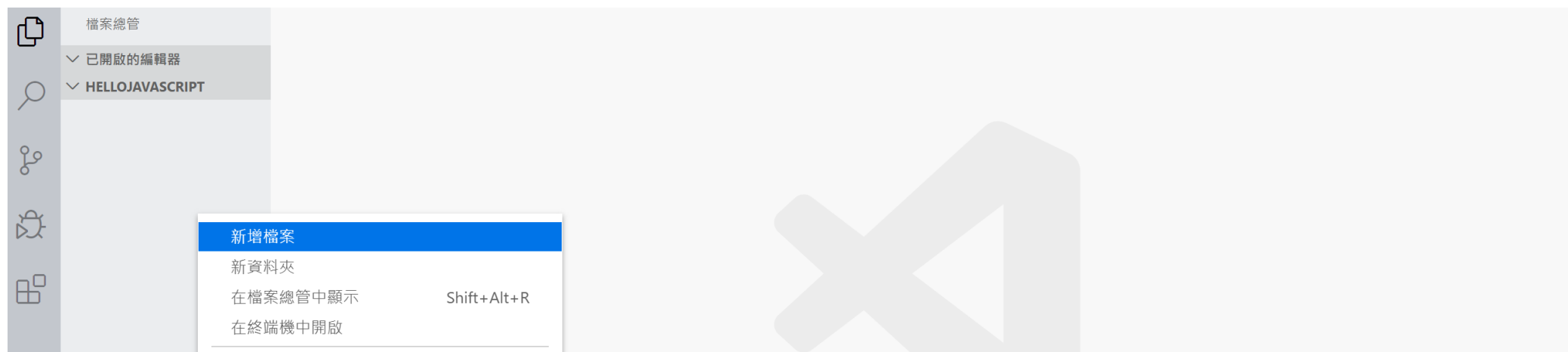
# 安裝擴充套件

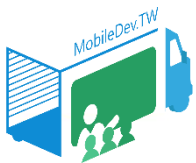
- 按下左邊 Extensions圖示  或 Ctrl + Shift + X
  - Chinese (Traditional) Language Pack for Visual Studio Code
  - Live Server
  - HTML Boilerplate
- 設定Ctrl+ 鼠標滾軸控制編輯器字號
  - editor.mouseWheelZoom
- 設定編輯時自動儲存
  - 檔案 -> 自動儲存



# 第一個 JS 專案

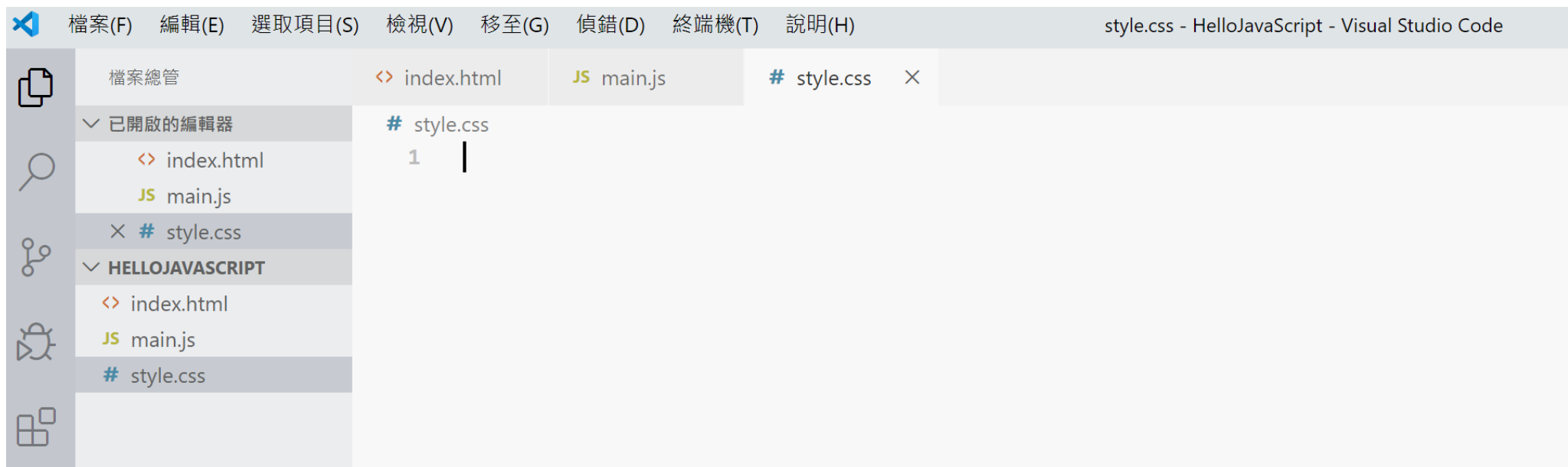
- 在左邊區塊點擊右鍵，新增檔案，命名為index.html





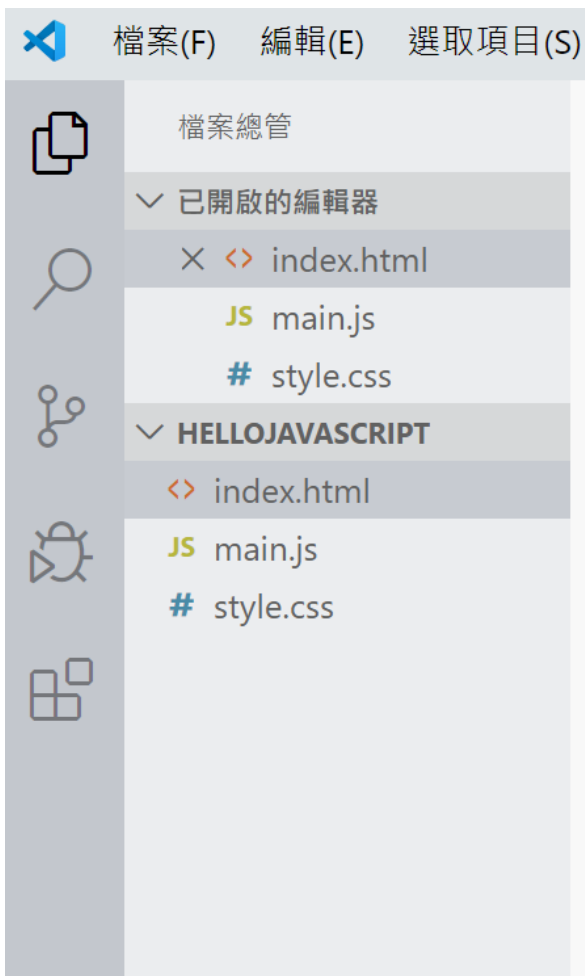
# 第一個 JS 專案

- 在左邊點擊右鍵，新增檔案，再增加兩個檔案
  - main.js
  - style.css



# 第一個 JS 專案

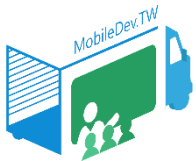
- 回到index.html，編輯檔案如下



```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title></title>
  <link rel="stylesheet" href="style.css">
  <script src="main.js"></script>
</head>
<body>

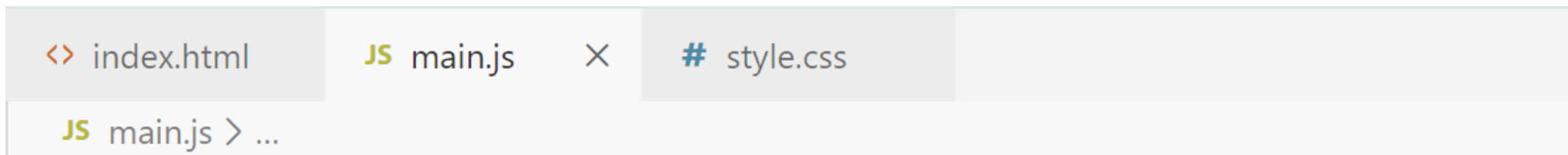
</body>
</html>
```



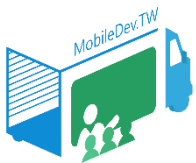


# 第一個 JS 專案

- 再到main.js，編輯檔案如下

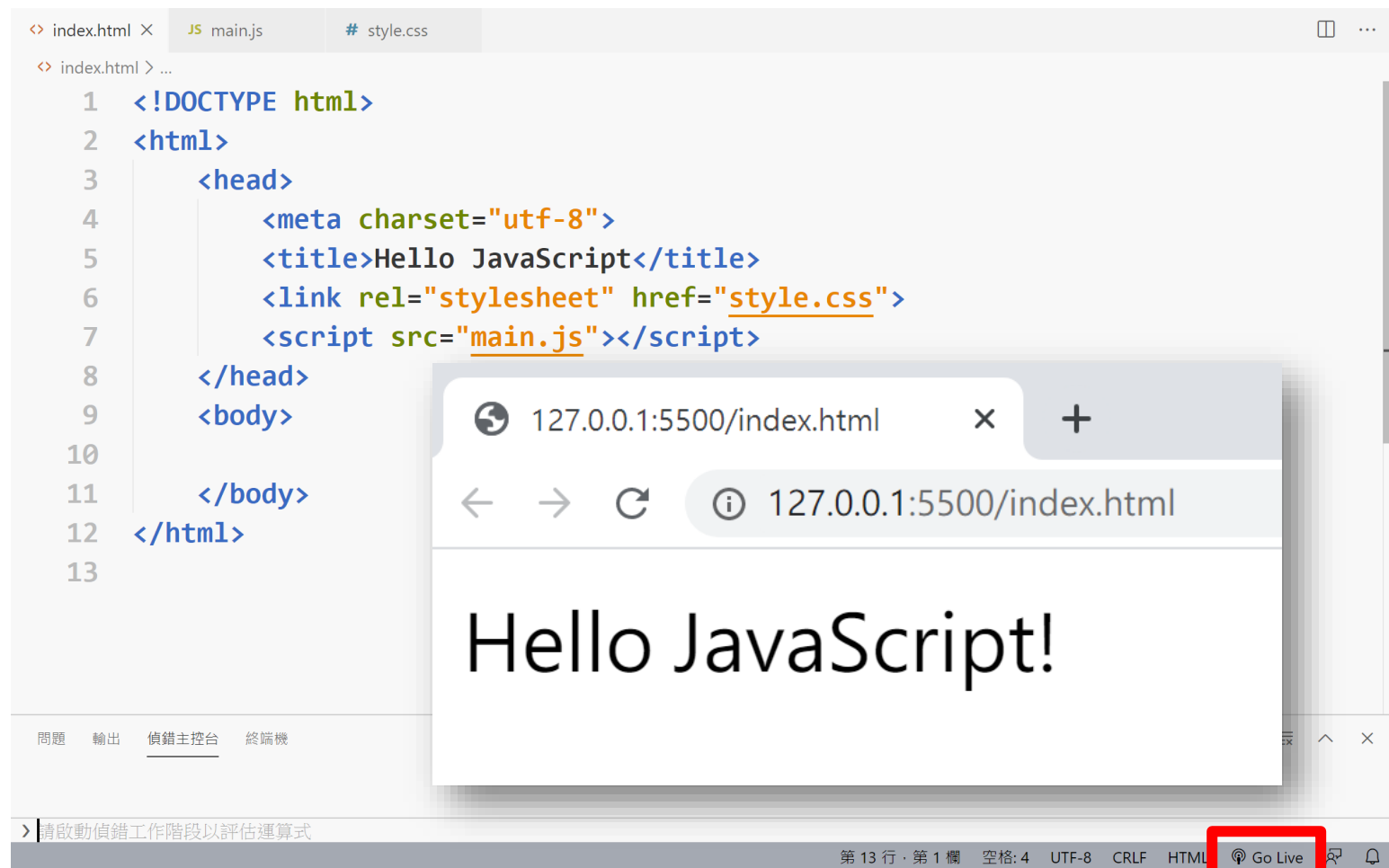


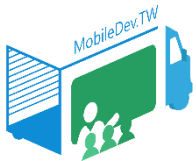
```
window.onload = function(){  
    document.write("Hello JavaScript!");  
};
```



# 測試執行

- 按下右下方的Go Live
- 或在左邊檔案區 index.html右鍵單擊 Open with Live Server





# 什麼是JavaScript

- 增加HTML頁面上的互動性
- 是一種scripting language
- 功用
  - 增加網頁上的動態效果
  - 可以根據特定事件對應執行動作
  - 可以讀取與改寫HTML組件
  - 可以拿來進行窗體驗證
  - 偵測使用者的瀏覽器
  - 建立與存取cookie

# 擺放位置

## 1. 放在<body>..</body>

- 一般要直接執行的JavaScript會放在</body>前面
- 先讓整個頁面的HTML組件加載

## 2. <head>...</head>

- 特定事件觸發才做的函數內容會放在<head>section

## 3. 獨立成一個檔案

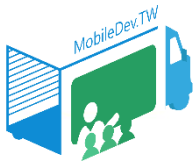
<head> <script src="xxx.js"> </script> </head>

```
<script type="text/javascript">
```

```
//就寫在這裡面
```

```
document.write("<p>" + Date() + "</p>");
```

```
</script>
```



# 陳述句 / 註解方式

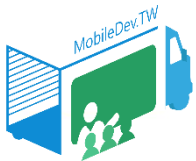
- 順序執行
- 大小寫有區分
- 通常以分號結尾

- 單行 //

- 多行

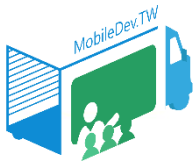
/\* Comments more than  
one line

\*/



# 變數

- 變數名稱
  - 大小寫有區分
  - 第一個字必須是英文字母或底線
- 變數宣告
  - var (逐漸減少使用)
  - let **NEW** (建議使用)
  - const **NEW** (建議使用)



# 變數範圍 Variables Scope

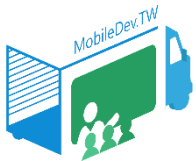
- 全域 Global Scope
  - 整個網頁程序執行都可以使用
- 函數內 Function Scope
  - 只在函數範圍內可以使用
- 區塊內 Block Scope **NEW** (ECMAScript 2015後)
  - 只在區塊範圍內可以使用
  - var 不支援



# 跳出窗口

- Alert box
  - `alert("sometext");`
- Confirm box
  - `confirm("sometext");`
  - OK : true or Cancel : false
- Prompt box
  - `prompt("sometext","defaultvalue");`
  - 取得使用者輸入的值進行動作





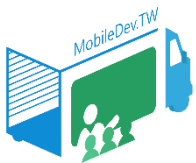
# 練習：點擊文字取得字元數

127.0.0.1:5500 顯示

JavaScript有10個字元

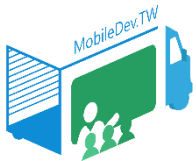
確定

- HTML
- CSS
- JavaScript



# index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <ul>
      <li>HTML</li>
      <li>CSS</li>
      <li>JavaScript</li>
    </ul>
    <script src="main.js" async defer></script>
  </body>
</html>
```



# main.js

```
window.onload = function(){  
    document.onclick = function(e){  
        alert(e.target.innerHTML + "有" + e.target.innerHTML.length + "個字元");  
    }  
}
```



127.0.0.1:5500/index.html

- HTML
- CSS
- JavaScript

127.0.0.1:5500 顯示

JavaScript有10個字元

確定

# Confirm Box 確認

127.0.0.1:5500 顯示

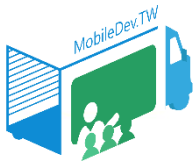
你真的確定你想要取消這個服務嗎？

確定

取消

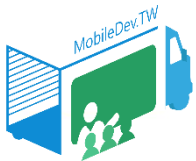
服務已取消

繼續使用本服務



# index.html

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title></title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>XXXXX</h1>
  <!--
  <ul>
    <li>HTML</li>
    <li>CSS</li>
    <li>JavaScript</li>
  </ul>
  -->
  <script src="main.js" async defer></script>
</body>
</html>
```



# main.js

```
/*
window.onload = function(){
    document.onclick = function(e){
        alert(e.target.innerHTML + "有" + e.target.innerHTML.length + "個字元");
    }
}*/

let confirmAnswer = confirm("你真的確定你想要取消這個服務嗎?");
let thisH1 = document.getElementsByTagName("h1")[0];
if(confirmAnswer){
    thisH1.innerHTML = "服務已取消";
}else{
    thisH1.innerHTML = "繼續使用本服務";
}
```

# Prompt Box

127.0.0.1:5500 顯示

小明家裡有三個小孩，他兩個哥哥叫張一、張二、第三個小孩叫什麼？

確定

取消

張三

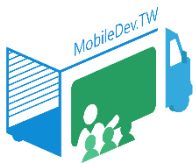
小明

李四

那小明是誰？

聰明

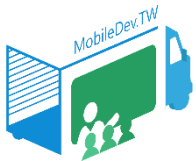
你想多了



# index.html

```
<!DOCTYPE html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1 id="Response">XXXXX</h1>
    <script async defer src="main.js"></script>
  </body>
</html>
```

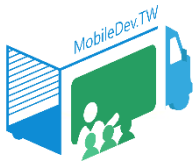




# main.js

```
let promptAnswer = prompt("小明家裡有三個小孩，他兩個哥哥叫張一、張二，請問第三個小孩叫什麼?", "張三");
let thisH1 = document.getElementById("Response");

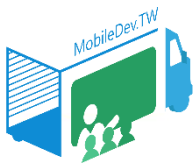
switch(promptAnswer){
  case "張三":
    thisH1.innerHTML = "那小明是誰?";
    break;
  case "小明":
    thisH1.innerHTML = "聰明";
    break;
  default:
    thisH1.innerHTML = "你想多了";
}
```



# 函數

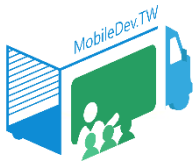
- 執行時機
  - 直接被呼叫
  - 事件發生時觸發
  - Callback
- 語法

```
function functionname(var1, var2, ..., varX)  
{  
  some code  
}
```



# 函數的呼叫 – index.html

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title></title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1 id="Response">XXXXXX</h1>
  <button>Click Me</button>
  <script async defer src="main.js"></script>
</body>
</html>
```

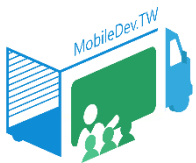


# 函數的呼叫 – main.js

```
function showAlert(){  
    thisH1.innerHTML = "Hello!";  
}
```

```
let thisButton = document.getElementsByTagName("Button")[0];  
let thisH1 = document.getElementsByTagName("h1")[0];  
thisButton.onclick = function(){  
    showAlert();  
};
```

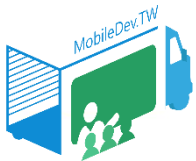




# Break and Continue in For loop

- break : 離開循環
- continue : 跳過這一圈

0	<pre>for(let i=0;i&lt;10;i++){   if(i==3){     break;   }   console.log(i); }</pre>	0	<pre>for(let i=0;i&lt;10;i++){   if(i==3){     continue;   }   console.log(i); }</pre>	0
1		1		1
2		2		2
3		2		4
4	>	>	>	5
5				6
6				7
7				8
8				9
9				>
>				



# For...in

- 巡訪對象中的每一個屬性與方法

```
let person = {  
  firstName: "Ryan",  
  lastName: "Chung",  
  height: 178  
};
```

```
for(x in person){  
  console.log(person[x]);  
}
```

Ryan

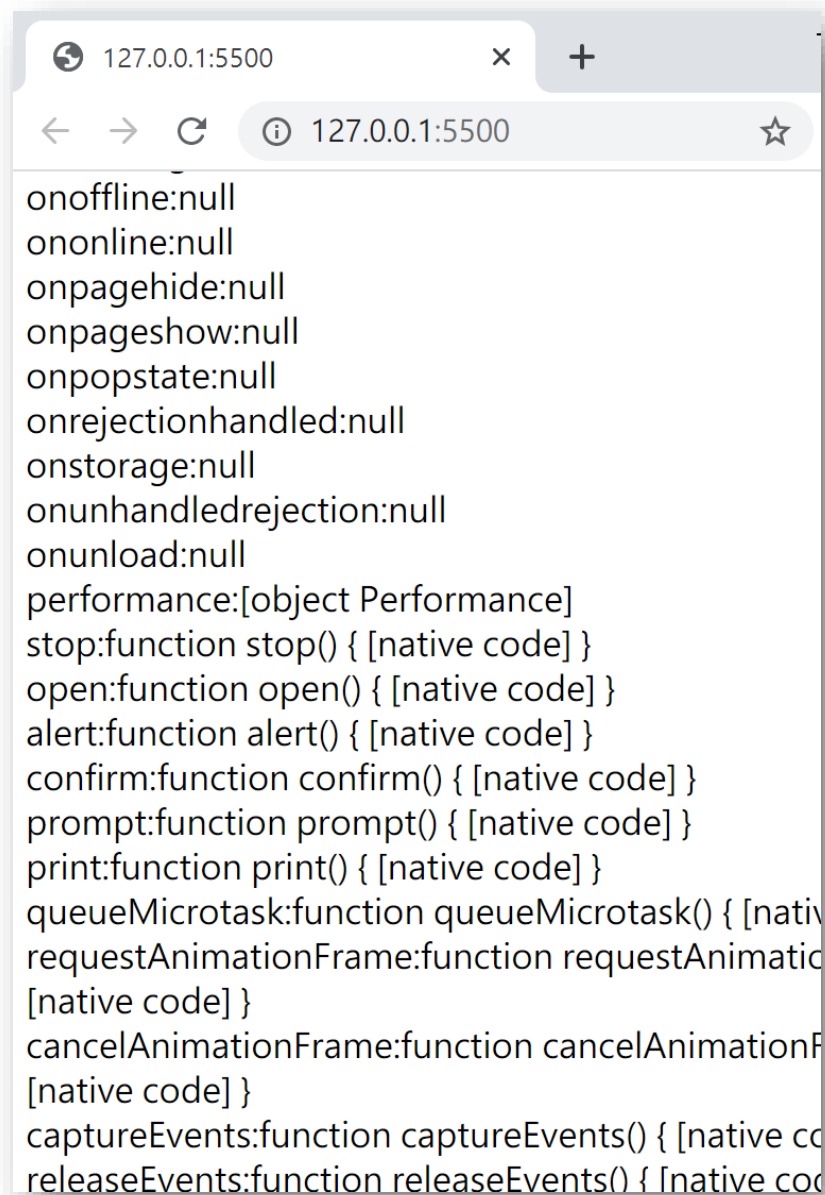
Chung

178



# For..in Lab

- 巡訪window物件



```
127.0.0.1:5500
onoffline:null
ononline:null
onpagehide:null
onpageshow:null
onpopstate:null
onrejectionhandled:null
onstorage:null
onunhandledrejection:null
onunload:null
performance:[object Performance]
stop:function stop() { [native code] }
open:function open() { [native code] }
alert:function alert() { [native code] }
confirm:function confirm() { [native code] }
prompt:function prompt() { [native code] }
print:function print() { [native code] }
queueMicrotask:function queueMicrotask() { [native code] }
requestAnimationFrame:function requestAnimationFrame() { [native code] }
cancelAnimationFrame:function cancelAnimationFrame() { [native code] }
captureEvents:function captureEvents() { [native code] }
releaseEvents:function releaseEvents() { [native code] }
```



# 事件觸發方法與常見事件

- 事件是JavaScript可以偵測得到的一種動作
- HTML元件上的動作會觸發JavaScript的事件
  - 按下按鈕元件 --> onClick事件
  - 網頁頁面載入 --> onLoad事件
  - 滑鼠游標在某個元件上 --> onMouseOver事件

發生什麼事？要做什麼因應措施？





# 事件觸發處理方法1.HTML Attribute

- 直接加入在HTML的屬性中

```
20<script>
21  function showAlert()
22  {
23      alert("hi");
24  }
25</script>
26</head>
27<body>
28  <h1 align="center">Hello Alert</h1>
29  <hr>
30  <div onclick="showAlert()">Click Me</div>
31</body>
```



# 事件觸發處理方法2.Event Function

- 使用JavaScript語法，定義onclick要執行的動作

```
20 <script>
21     function showAlert()
22     {
23         alert("hi");
24     }
25 </script>
26 </head>
27 <body>
28     <h1 align="center">Hello Alert</h1>
29     <hr>
30     <div>Click Me</div>
31     <script>
32         document.getElementsByTagName("div")[0].onclick=
33         showAlert;
34     </script>
```

A purple arrow originates from the `showAlert()` function definition (lines 21-24) and points to the `showAlert` property in the `onclick` assignment (lines 32-33), illustrating the link between the function and the event handler.



# 事件觸發處理方法3.Event Listener

- 監聽該元件，是否有XX事件發生

```
20 <script>
21     function showAlert()
22     {
23         alert("hi");
24     }
25 </script>
26 </head>
27 <body>
28     <h1 align="center">Hello Alert</h1>
29     <hr>
30     <div>Click Me</div>
31 <script>
32     document.getElementsByTagName("div")[0].
33     addEventListener("click",showAlert);
34 </script>
```

# 常見滑鼠/鍵盤事件

事件	描述
onclick	點擊
ondblclick	雙擊
onmousedown	在該元件上按下滑鼠按鍵
onmousemove	在該元件上移動游標
onmouseover	滑鼠游標進入該元件範圍
onmouseout	滑鼠游標離開該元件範圍
onmouseup	在該元件上釋放滑鼠按鍵

事件	描述
onkeydown	實體鍵盤被按下
onkeypress	有字元被輸入
onkeyup	實體鍵盤被放開



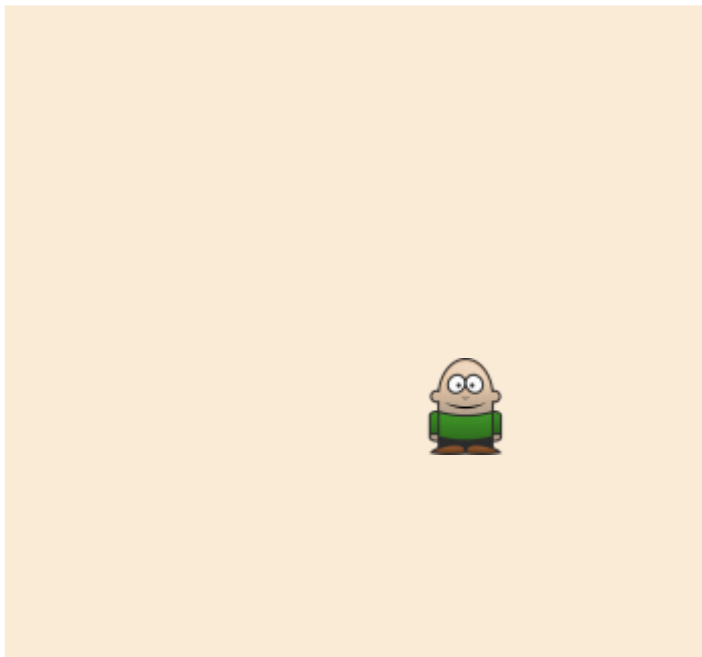
# 框架/對象事件

事件	描述
onload	文件、對象被載入
onscroll	文件捲動
onabort	取消載入
onerror	載入錯誤
onresize	大小被改變
onunload	離開該頁面、進入新頁面、重新整理

# 視窗事件

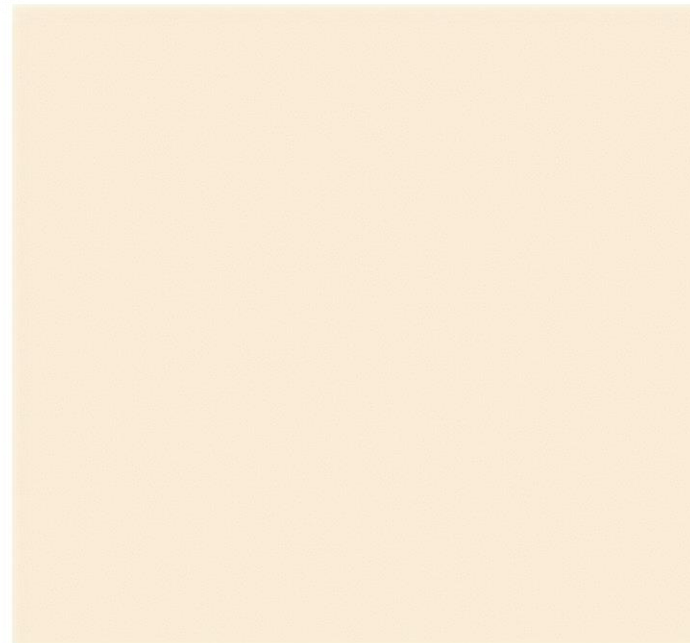
事件	描述
onblur	用戶離開聚焦在該項目
onchange	內容被改變(選項勾選、下拉選單...)
onfocus	用戶聚焦在該項目
onreset	恢復預設資料
onselect	選取了一些文字
onsubmit	送出

# 練習：小人物回家



## 你進來了

你在裡面走來走去

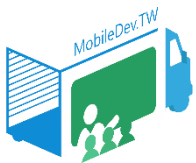




# index.html

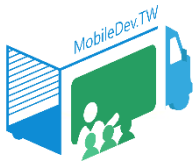
```
<!DOCTYPE html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div></div>
    <h1></h1>
    <p></p>
    <script async defer src="main.js"></script>
  </body>
</html>
```





# style.css

```
div{  
    width: 30%;  
    height: 200px;  
    background-color: antiquewhite;  
    cursor: url(man.png), auto;  
}  
p{  
    height: 30px;  
    font-size: large;  
}
```



# main.js

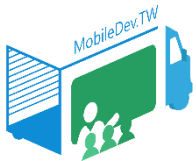
```
let thisH1 = document.getElementsByTagName("h1")[0];
let thisP = document.getElementsByTagName("p")[0];
let thisDiv = document.getElementsByTagName("div")[0];

function mouseIn(){
    thisH1.innerHTML="你進來了";
}

function mouseOut(){
    thisH1.innerHTML="你出去了";
    thisP.innerHTML="";
}

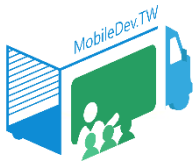
function mouseMove(e){
    thisP.innerHTML="你在裡面走來走去。位置 : "+e.clientX+", "+e.clientY;
}

thisDiv.addEventListener("mouseover", mouseIn);
thisDiv.addEventListener("mouseout", mouseOut);
thisDiv.addEventListener("mousemove", mouseMove);
```



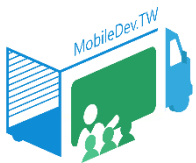
# String 字串對象

- 屬性
  - `text.length` : 取得字串長度
- 方法
  - `text.big()` : 將字串字體放大
  - `text.charAt(x)` : 回傳x位置的字元
  - `text.concat(string2,string3,...)` : 串接字串
  - `text.indexOf(string)` : 回傳第一個出現該字串的位置
  - `text.lastIndexOf(string)` : 回傳最後一個出現該字串的位置
  - `text.replace("subString","newString")` : 字串替換
  - `text.slice(begin, end)` : 取出部份字串 (可輸入一個負值從後面取x字元)
  - `text.split(separator)` : 依特定符號進行切割，並放入陣列
  - `text.substr(begin,length)` : 取出部份字串
  - `text.substring(from,to)` : 取出部份字串
  - `text.toLowerCase()` : 全部轉小寫



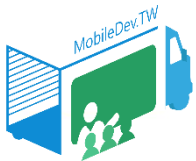
# String Lab

- 字符串 Hello World!
  - 秀出字串長度
  - 找到World的位置
  - 依空格切割，秀出Hello與World!



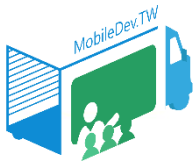
# index.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <title></title>
6      </head>
7      <body>
8          <h1>Hello World!</h1>
9          <script async defer src="main.js"></script>
10     </body>
11 </html>
```



# main.js

```
1 let thisH1 = document.getElementsByTagName("h1")[0];
2 thisH1.addEventListener("click", showAlert);
3
4 function showAlert(){
5     alert("字串長度 : "+thisH1.innerHTML.length+"\n"+
6         "World在第"+thisH1.innerHTML.indexOf("World")+"位置"+"\\n"+
7         "第一個字"+thisH1.innerHTML.split(" ")[0]+"\\n"+
8         "第二個字"+thisH1.innerHTML.split(" ")[1]);
9 }
```



# Date 日期時間物件

- 取得目前時間
  - `var d=new Date();`
  - `d.getFullYear()` 目前年
  - `d.getMonth()` 目前月(0~11)
  - `d.getDate()` 目前日(1~31)
  - `d.getDay()` 目前周(0~6)
  - `d.getHours()` 目前時(0~23)
  - `d.getMinutes()` 目前分(0~59)
  - `d.getSeconds()` 目前秒(0~59)

取得目前日期

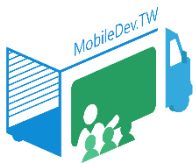
`d.toLocaleDateString()`

取得目前時間

`d.toLocaleTimeString()`

取得目前日期加時間

`d.toLocaleString()`



# 陣列物件

- 建立陣列

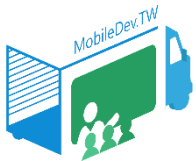
```
let myFriends = new Array();  
myFriends[0]="John";  
myFriends[1]="Mary";  
myFriends[2]="David";
```

```
let myFriends =  
new Array("John", "Mary", "David");
```

```
let myFriends = ["John", "Mary", "David"];
```

- 存取陣列

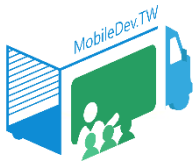




# Array 陣列物件

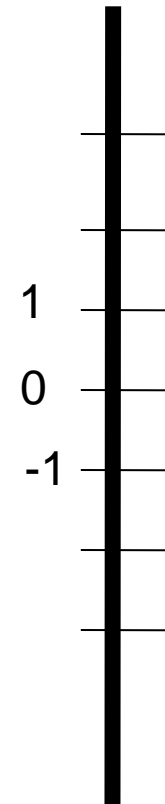
- 將陣列的各元素以特定的連接符號組合成字串
  - `arrayName.join(separator)`

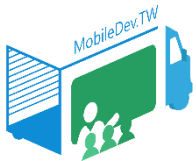
```
//陣列的三種建立方式
/*
let myFriends = new Array();
myFriends[0]="John";
myFriends[1]="Mary";
myFriends[2]="David";
*/
//let myFriends = new Array("John","Mary","David");
let myFriends = ["John","Mary","David"];
//console.log(myFriends[1]);
//把陣列的各元素組合成一個字串
console.log(myFriends.join(" and "));
```



# Math 數學物件

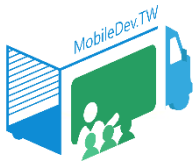
- `Math.PI` : PI值
- `Math.abs(number)` : 取絕對值
- `Math.floor(number)` : 向下整數
- `Math.ceil(number)` : 向上整數
- `Math.round(number)` : 取整數(四捨五入)
- `Math.max(number,number,number,...)` : 最大值
- `Math.random()` : 產生隨機數 (介於0 ~ 1之間)





# window Object

- 在瀏覽器中開啓一個窗口即建立一個視窗物件
- 屬性
  - `window.closed` : 該視窗關閉即為true
  - `window.name` : 該視窗名稱
  - `history` (物件) : 記錄下用戶在該視窗所去過的網址
  - `navigator` (物件) : 瀏覽器相關資訊
  - `document` (物件) : 當該視窗載入一份HTML文件時即產生



# window Object

- 方法

- 跳出視窗

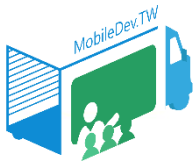
- alert()
    - confirm()
    - prompt()

- 定時器

- setInterval()
    - setTimeout()
    - clearInterval()
    - clearTimeout()

- 開新視窗

- window.open(URL, name, specs)
  - URL : 顯示網頁
  - name : 視窗名稱
  - specs : 規格
    - height
    - width
    - left
    - top



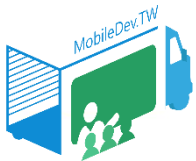
# document Object

- 屬性

- domain : 傳回目前文件所在的域名
- title : 傳回目前文件定義的title
- URL : 傳回目前文件的完整網址路徑
- cookie : 傳回目前文件的cookie資訊

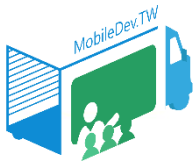
- 方法

- getElementById() : 存取第一個id名稱相符的元件
- getElementsByName() : 存取所有name相符的元件
- getElementsByTagName() : 存取所有該標籤名稱的元件
- write() : 寫入文件
- writeln() : 寫入文件並帶上換行符號



# document Object

- Collections 篩選
  - `anchors[ ]` 找到所有頁面上的anchor
  - `forms[ ]` 找到所有頁面上的form
  - `images[ ]` 找到所有頁面上的image
  - `links[ ]` 找到所有頁面上的link



# Document Object Model

- W3C標準
- 用來存取HTML或XML文件
- Core DOM
  - 任何結構化文件的標準模型
- XML DOM
  - XML文件的標準模型
- HTML DOM
  - HTML文件的標準模型

## HTML DOM

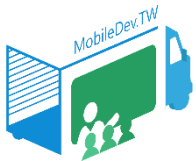
- HTML的標準物件模型
- HTML的標準程式界面
- 可於各種平台、語言使用
- 用來取得、改變、新增或刪除HTML元件



# Node 節點

- HTML文件中，所有的事物都是一個節點
- 整個文件：文件節點
- HTML元件：元件節點
- HTML元件中的文字：文字節點
- HTML中的屬性：屬性節點
- HTML中的註解：註解節點





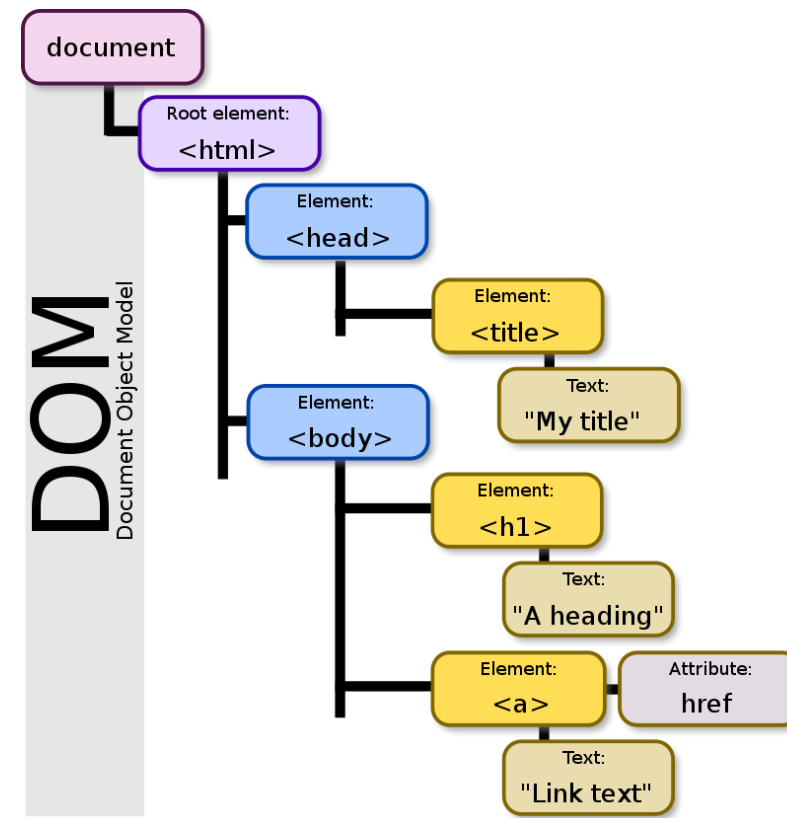
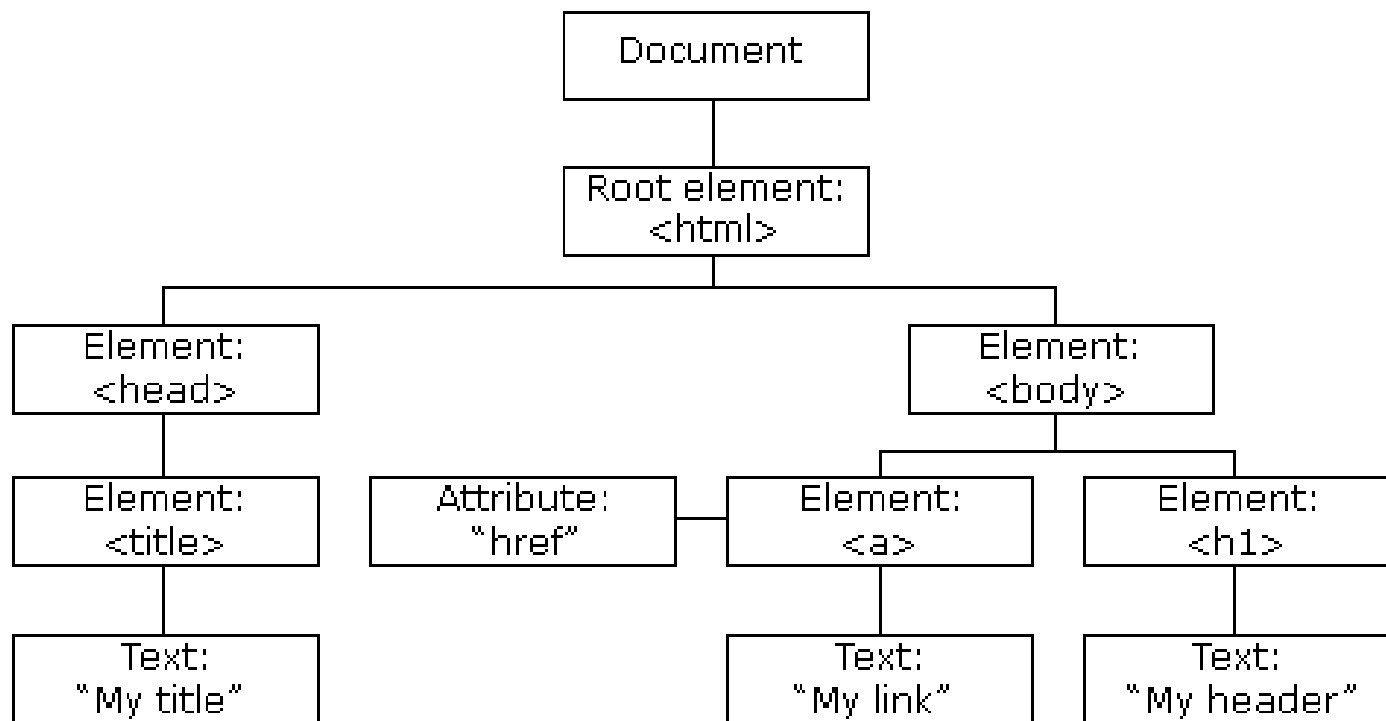
# 節點分析

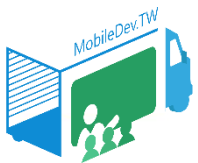
- 根節點：`<html>`
- `<html>` 節點有兩個子節點：`<head>` 與 `<body>`
- `<title>` 節點有一個 `text` 子節點：DOM Tutorial

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

# 樹狀結構

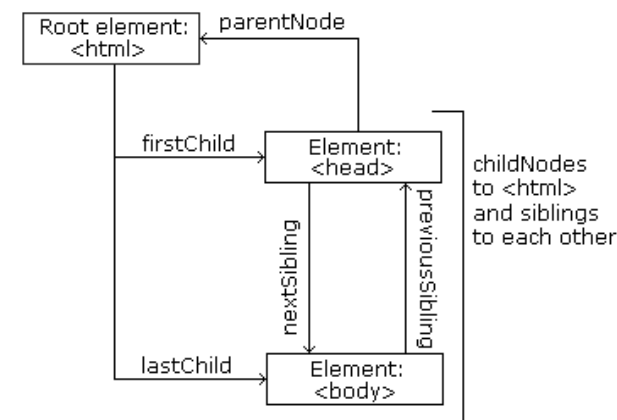
- HTML DOM把HTML文件當成一棵節點樹

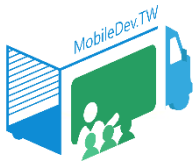




# 樹狀結構：父母、小孩與兄弟姊妹

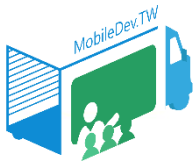
- 最上層的節點稱之為根 (root)
- 除了根節點外，每一個節點都有一個父節點
- 一個節點可以有任意數量的子節點
- 沒有子節點的節點稱之為葉 (leaf)
- 有相同父節點的節點稱之為兄弟(sibling)





# HTML DOM屬性

- innerHTML : 文字值
- nodeName : 名稱
- nodeValue : 值
- parentNode : 父節點
- firstChild : 第一個子節點
- lastChild : 最後一個子節點
- nextSibling : 緊鄰的兄弟節點
- nodeType : 元件型態



# HTML DOM屬性

- nodeName

- 元件節點的節點名稱即為標籤名稱(會變成大寫)
- 屬性節點的節點名稱即為屬性名稱
- 文字節點的節點名稱為#text
- 文件節點的節點名稱為#document

- nodeValue

- 組件節點的節點值為null
- 文字節點的節點值即為文字本身
- 屬性節點的節點值即為屬性值

- nodeType

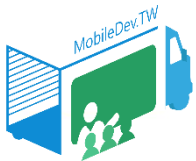
1 : Element 組件

2 : Attribute 屬性

3 : Text 文字

8 : Comment 註解

9 : Document 文件



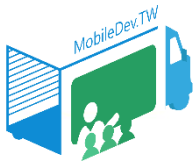
# HTML DOM

- 方法

- getElementById(id)
- getElementsByTagName(name)
- appendChild(node)
- removeChild(node)
- getAttribute(attributeName)

- Collection

- attributes[ ] 回傳該元件的所有屬性組成一個陣列
- childNodes[ ] 回傳該元件的所有子節點組成一個陣列



# 改變元件

- 改變元件的屬性值
  - `document.body.bgColor="yellow";`
- 改變元件內的文字
  - `document.getElementById("p1").innerHTML="Hi!";`
- 改變元件的樣式
  - `document.body.style.color="blue";`
  - `document.body.style.backgroundImage`

# 自定義物件

- [方法一]直接建立一個新物件

```
personObj = new Object();
personObj.firstname = "John";
personObj.lastname = "Doe";
personObj.age = 50;
personObj.eyecolor = "blue";
personObj.smile=function()
{
    document.write("^___^");
};

personObj.smile();

document.write(personObj.firstname+"<br />");
```



# 自定義物件

- [方法二]直接建立一個新物件(寫在一起)

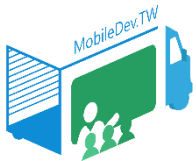
```
personObj={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue",  
smile:function(){document.write("-_-|||")}};  
  
personObj.smile();  
  
document.write(personObj.lastname+"<br />");
```

# 自定義物件

- [方法三]先宣告，再使用

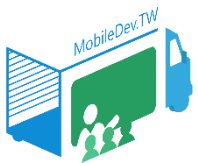
```
function person(firstname, lastname, age, eyecolor) {  
    this.firstname = firstname;  
    this.lastname = lastname;  
    this.age = age;  
    this.eyecolor = eyecolor;  
    this.smile=function(){document.write("^0^")} ;  
}
```

```
var myFather=new person("John","Doe",50,"blue");  
myFather.smile();  
document.write(myFather.lastname+"<br />");
```

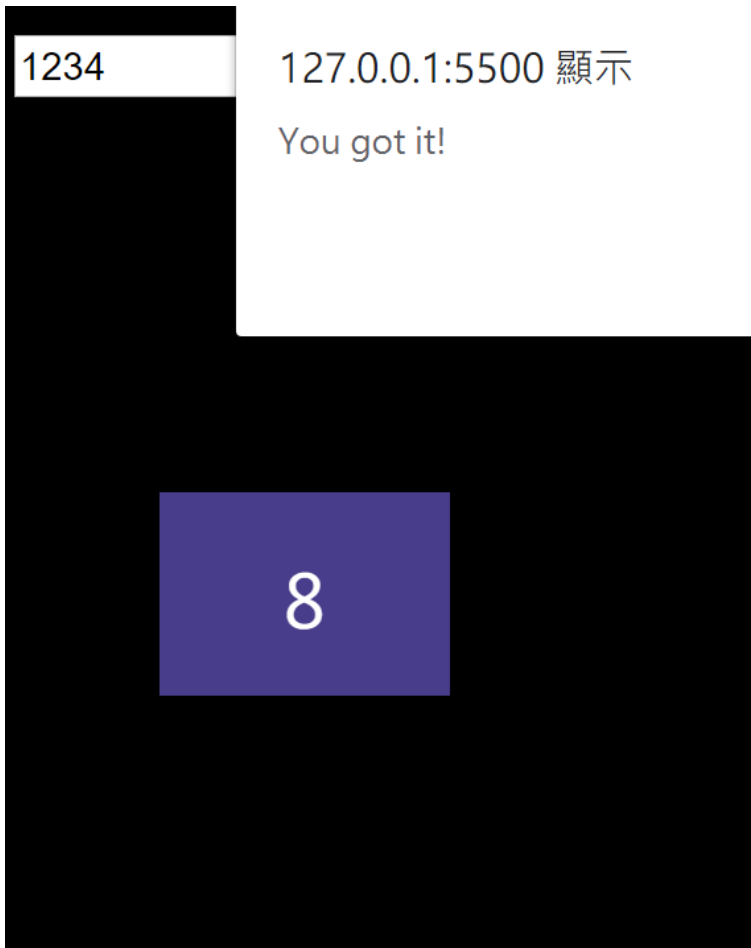


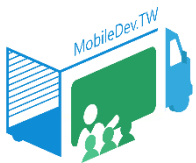
# 定時器

- `setTimeout()` : 特定時間之後做什麼事情(一次)
- `clearTimeout()` : 解除`setTimeout`
- `setInterval()` : 每隔多少時間之後做什麼事情(反覆)
- `clearInterval()` : 解除`setInterval`



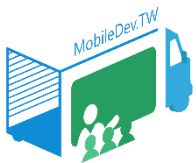
# 練習：密碼鎖炸彈





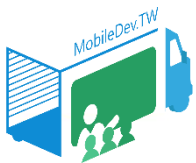
# index.html

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title></title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <input id="userInput" type="text">
  <button>Enter</button>
  <p id="hint"></p>
  <div id="timer"></div>
  <script async defer src="main.js"></script>
</body>
</html>
```



# style.css

```
body{
    background-color: black;
}
div{
    width: 100px;
    height: 50px;
    margin: 100px 50px;
    text-align: center;
    padding-top: 20px;
    color: white;
    font-size: 25px;
    background-color: darkslateblue;
}
p{
    color: red;
    height: 20px;
}
```



# main.js

```
let timer = document.getElementById("timer");
let userInput = document.getElementById("userInput");
let hint = document.getElementById("hint");
let button = document.getElementsByTagName("button")[0];
let count = 10;

timer.innerHTML = count;
button.addEventListener("click", checkPassword);
let myVar = setInterval(myTimer, 1000);

function myTimer(){
    count--;
    timer.innerHTML = count;
    if(count==0){
        hint.innerHTML="Game Over!";
        clearInterval(myVar);
    }
}

function checkPassword(){
    hint.innerHTML="";
    if(parseInt(userInput.value)==1234){
        alert("You got it!");
        clearInterval(myVar);
    }else{
        hint.innerHTML="Try again!";
    }
    userInput.value = null;
}
```



# JavaScript 綜整

- 基本認識
  - 簡介、功能、擺放位置、註解
- 程式運作
  - 變數、函數、迴圈、事件觸發
- 物件導向
  - 內建物件、DOM、自定義物件

