

# JavaScript 基礎



林新德

shinder.lin@gmail.com

參考專案 <https://bitbucket.org/lsd0125/mfee26-js/>

- 1. JavaScript 常數與變數
- 2. 運算子與基本類型
- 3. 流程控制
- 4. 資料集合類型: **Object**, **Array**
- 5. 自訂函式與箭頭函式
- 6. 定時觸發函式
- 7. 代表時間點的 **Date** 物件
- 8. **Math** 相關功能
- 9. **window** 物件與其子物件
- 10. 事件處理
- 11. 錯誤處理 **try/catch**
- 12. 表單與表單欄位
- 13. **Regular Expression** ( 補充教材 )
- 14. **AJAX** ( 補充教材 )

# 1.1 JavaScript 程式語言

- 直譯式 ( **Script** ) 程式語言。
- 功能為，為網頁提供互動的功能，執行環境為瀏覽器 ( 前端 )。
- 也是網站後端的程式語言 ( **Node.js** )。
- 其標準為 **ECMA** 所制訂，標準稱為 **ECMAScript** ( 簡稱 **ES** )。
- **ES5** 可為大部份的瀏覽器所執行。
- **Chrome** 目前可支援大部份 **ES6+** 的功能。

## 1.2 Chrome developer tools

- 在網頁上按滑鼠右鍵，於跳出的選單上點選「檢查」，可開啟 **developer tools**（開發人員工具）。
- 「**Elements**」可查看 **DOM**。
- 「**Console**」可用來做 **JavaScript** 除錯。
- 「**Network**」可查看 **HTTP** 的 **requests** 和 **responses** 狀況。
- 「**Sources**」可查看該頁面使用的資源（各類型的檔案）。
- 「**Application**」頁面暫存及**cookies** 等資料查看。

## 1.3 動態新增頁面標籤內容

```
<script>
    document.write('<h2>123</h2>');
</script>
```

- `<script></script>` 可以放在哪些地方？

```
<script>
    alert('hello');
    document.write('<h2>ABC</h2>');
</script>
```

- 注意 `alert()` 的特性
- `confirm()` 和 `prompt()` 功能為何？

## 1.4 常數表示法

```
<script>
  // console.log(023); // 舊的 8進位用法，不建議使用
  console.log(0o23);    // 8進位
  console.log(0x23);    // 16進位
  console.log(0b1111);  // 2進位
  console.log(2e5);      // 科學表示法
  console.log(2E+5);
  console.log(Number.MAX_SAFE_INTEGER);
  console.log(Number.MAX_VALUE);
</script>
```

## 1.5 變數宣告

```
// 會變成 window 的屬性
var age = 10;
console.log(age);

// 不會變成 window 的屬性
let name = 'Shinder'; // ES6
console.log(name);

// 不會變成 window 的屬性
const cc = '不能用設定改變'; // ES6
console.log(cc);
```

- 1. `var` 和 `let` 宣告變數有何不同？

- 2. `let` 和 `const` 宣告變數有何不同？



## 1.6 取得標籤元素

```
<body>
  <div id="info"></div>
  <script>
    const el = document.getElementById('info');
    el.innerHTML = '<h2>AAAA</h2>';
  </script>
</body>
```

# 1.7 DOM

- 文件物件模型 ( **Document Object Model, DOM** ) 是 **HTML** 文件的程式介面。
- 它提供了一個文件 ( 樹 ) 的結構化表示法，並定義讓程式可以存取並改變文件架構、風格和內容的方法。
- 參考 [https://www.w3schools.com/jsref/dom\\_obj\\_all.asp](https://www.w3schools.com/jsref/dom_obj_all.asp)



## 2.1 基本類型

- 基本類型包含：Number、Boolean、String。
- 可使用 `typeof` 運算子查看變數的基本類型

```
'use strict'; // 使用嚴謹模式，變數必須宣告才能使用
var a = 10, b;
console.log(typeof a); // 'number'
console.log(typeof true); // 'boolean'
console.log(typeof 'hi'); // 'string'
console.log(typeof 0.01); // 'number'
console.log(typeof b); // 'undefined'
console.log(typeof null); // 'object'
```

- `obj.constructor.name` // ES6 查看類型的名稱

## 2.2 算術運算子

- + ( 加 )
- - ( 減 )
- \* ( 乘 )
- / ( 除 )
- % ( 求餘數 )
- 先乘除後加減，可以用**小括號**提高運算順序（和數學上的概念相同）。
- $0.1 + 0.2$

## 2.3 關係運算子

- 結果為布林值 ( **true** 或 **false** )，和數學上的概念不同。
- $a < b$  ( **a 是否** 小於 **b** )
- $a \leq b$  ( **a 是否** 小於等於 **b** )
- $a > b$  ( **a 是否** 大於 **b** )
- $a \geq b$  ( **a 是否** 大於等於 **b** )
- $a == b$  ( **a 是否** 等於 **b** )
- $a != b$  ( **a 是否** 不等於 **b** )
- $a === b$  ( **a 是否** 等於 **b**，嚴謹，類型必須相同 )
- $a !== b$  ( **a 是否** 不等於 **b**，嚴謹 )

## 2.4 邏輯運算子

- `!` ( `not` ) 非，單元運算子，優先權最高。
- `&&` ( `and` ) 且，快捷運算子，以布林值概念判斷 ( **Truthy**, **Falsy** )，結果不一定為布林值。
- `||` ( `or` ) 或，快捷運算子，以布林值概念判斷，結果不一定為布林值。
- 看成 **true** 的值稱作 **Truthy**
- 看成 **false** 的值稱作 **Falsy**

- 請在 **console** 測試：

```
console.log( !! -1 );  
console.log( true || false );  
console.log( 3 || 6 );  
console.log( 7 && 8 );  
console.log( 0 && 2 );
```



## 2.5 三元運算子

- ( 判斷式 ) ? ( 真時回傳值 ) : ( 假時回傳值 )
- `console.log( 5 > 8 ? 'a' : 'b' );`

運算子優先權：  
單元 > 算術 > 關係 > 邏輯 > 設定

## 2.6 基本型別轉換

- 轉換為數值（通常為字串轉數值）：

- `+` 轉換為數值
- `parseInt(s)` 字串轉換為整數
- `parseFloat(s)` 字串轉換為浮點數
- `Number(s)` 轉換為數值（較少用）

- 轉換為布林值：

- `!!n`
- `Boolean(n)`（較少用）

- 轉換為字串：

- `n + ""`（左右兩個運算元，只要一個為字串，即為字串串接）
- `n.toString()`
- `String(n)`（較少用）

\*\* 十進位數值如何轉換為十六進位的字串？

## 2.7 字串的標示方式

- 傳統方式使用單引號或雙引號標示。
- **ES6** 可以使用反引號標示，而且可以在中間換行。

```
<div id="info"></div>
<script>
  const info = document.querySelector('#info');
  let name = 'Enes Kanter';
  const str = `土耳其裔中鋒坎特 ( ${name} ) <br>
               本季在紐約尼克陣中逐漸被冷落`;
  info.innerHTML = str;
</script>
```

- 字串的跳脫表示法

```
console.log("You say \"Yes\", I say \"No\".");  
console.log('You say "Yes", I say "No".');  
console.log('You say \\Yes\\, \\n I say \\No\\'.');
```

[https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/String#跳脫符號](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/String#跳脫符號)

## 2.8 字串常用方法

String的方法	說明
charAt (索引)	取得某位置的字元。
charCodeAt (索引)	取得某位置字元的字碼。
concat (字串)	字串串接。通常使用 + 運算子。
indexOf (子字串, [索引])	子字串出現的位置。
lastIndexOf (子字串)	子字串最後出現的位置。
slice (索引B, 索引E)	從索引B到E ( 不包含 ) 建立一個新字串回傳。
split (分割符號)	以分割符號 ( 字串 ) 切割產生陣列。
substr (索引B, 字元個數)	依字元個數，從索引B取得建立一個新字串回傳。
substring (索引B, 索引E)	同slice()。
toLowerCase ()	轉換成小寫字母回傳。
toUpperCase ()	轉換成大寫字母回傳。

屬性：length ( 字串長度，字元數量 )

## 2.9 querySelector()

- <https://developer.mozilla.org/zh-TW/docs/Web/API/Document/querySelector>
- `querySelector()`, `querySelectorAll()` 利用CSS 選擇器去取得 DOM 裡的元素。
- `querySelector()` 只會選到第一個符合的元素。
- `querySelectorAll()` 才可以選到所有符合的元素（回傳類型 `NodeList`）。

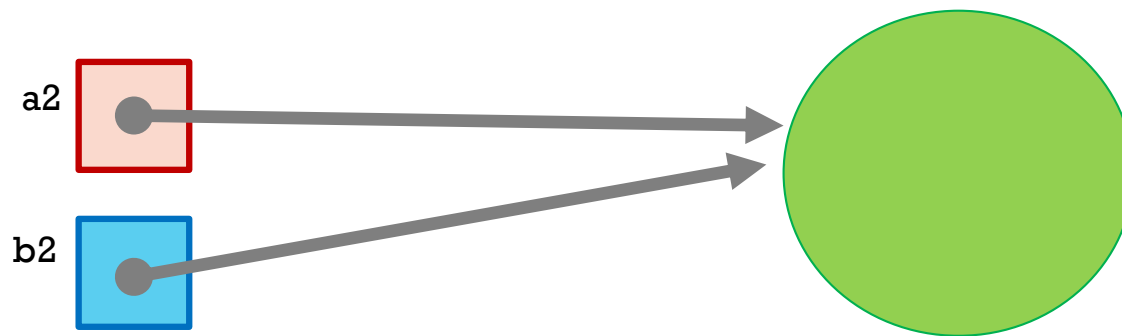
```
// 取得符合選擇器條件的第一個元素
document.querySelector('#my_id');

// 取得符合選擇器條件的所有元素
document.querySelectorAll('.my_class');
```

```
<table>
  <tr>
    <td>1</td>
    <td class="myclass">2</td>
    <td>3</td>
  </tr>
  <tr>
    <td>4</td>
    <td class="myclass">5</td>
    <td>6</td>
  </tr>
</table>
<script>
  const td1 = document.querySelector('td');
  const td2 = document.querySelectorAll('td');
  const td3 = document.querySelector('.myclass');
  const td4 = document.querySelectorAll('.myclass');
</script>
```

請在 **console** 查看 **td1** 等變數的內容

參照 ( reference ) 概念



\* `a2` 和 `b2` 變數內容為參照

`a` 10

`b` 5

\* `a` 和 `b` 變數內容為值 ( `Number`, `Boolean`, `String` 類型或 `undefined`, `null` )





# 3 流程控制

- if 選擇敘述
- if/else
- if/else 巢狀
- switch/case 選擇敘述
- for 迴圈
- while 迴圈
- do/while 迴圈
- for/in 迴圈 ( 使用在 **Object** 或 **Array** 類型的物件 )
- for/of 迴圈 ( 使用在 **Array** 類型的物件 )

## 3.1 if

- **if**：依條件執行或不執行某程式區塊（**block**）。

```
if(條件式) {  
    // 條件式為 true 時執行  
}
```

- **if/else**：依條件選擇執行第一個區塊或第二個區塊。

```
if(條件式) {  
    // 條件式為 true 時執行  
} else {  
    // 條件式為 false 時執行  
}
```

```
if(條件式一) {  
    // 條件式一為 true 時執行  
} else {  
    if(條件式二) {  
        // 條件式二為 true 時執行  
    } else {  
        // 皆為 false 時執行  
    }  
}
```

```
if(條件式一) {  
    // 條件式一為 true 時  
  
    if(條件式二) {  
        // 條件式二為 true 時執行  
    } else {  
        // 條件式二為 false 時執行  
    }  
} else {  
    // 條件式一為 false 時  
}
```

```
if(條件式一) {  
    // 條件式一為true時執行  
} else {  
    if(條件式二) {  
        // 條件式二為true時執行  
    } else {  
        if(條件式三) {  
            // 條件式三為true時執行  
        } else {  
            // 皆為false時執行  
        }  
    }  
}
```

```
if(條件式一) {  
    // 條件式一為true時執行  
} else if(條件式二) {  
    // 條件式二為true時執行  
} else if(條件式三) {  
    // 條件式三為true時執行  
} else {  
    // 皆為false時執行  
}
```

## 3.2 switch/case

- 比對以選擇開始執行的位置

```
switch (變數) {  
    case 值一:  
        // 變數為值一時  
        break;  
    case 值二:  
        // 變數為值二時  
        break;  
    case 值三:  
        // 變數為值三時  
        break;  
    default:  
        // 都不是時執行  
}  

```

```
<input type="number" id="inp" min="0" max="100" />
<button onclick="func()">click</button>
<p id="info"></p>
<script>
    const inp = document.querySelector("#inp");
    function func() {
        let score = +inp.value;
        let s = parseInt(score / 10);
        let g;
        if (s >= 0 && s <= 10) {
            switch (s) {
                case 10:
                case 9:
                    g = "A"; break;
                case 8:
                    g = "B"; break;
                case 7:
                    g = "C"; break;
                case 6:
                    g = "D"; break;
                default:
                    g = "F";
            }
        } else {
            g = "分數超出範圍";
        }
        info.innerHTML = g;
    }
</script>
```

## 3.3 for 迴圈

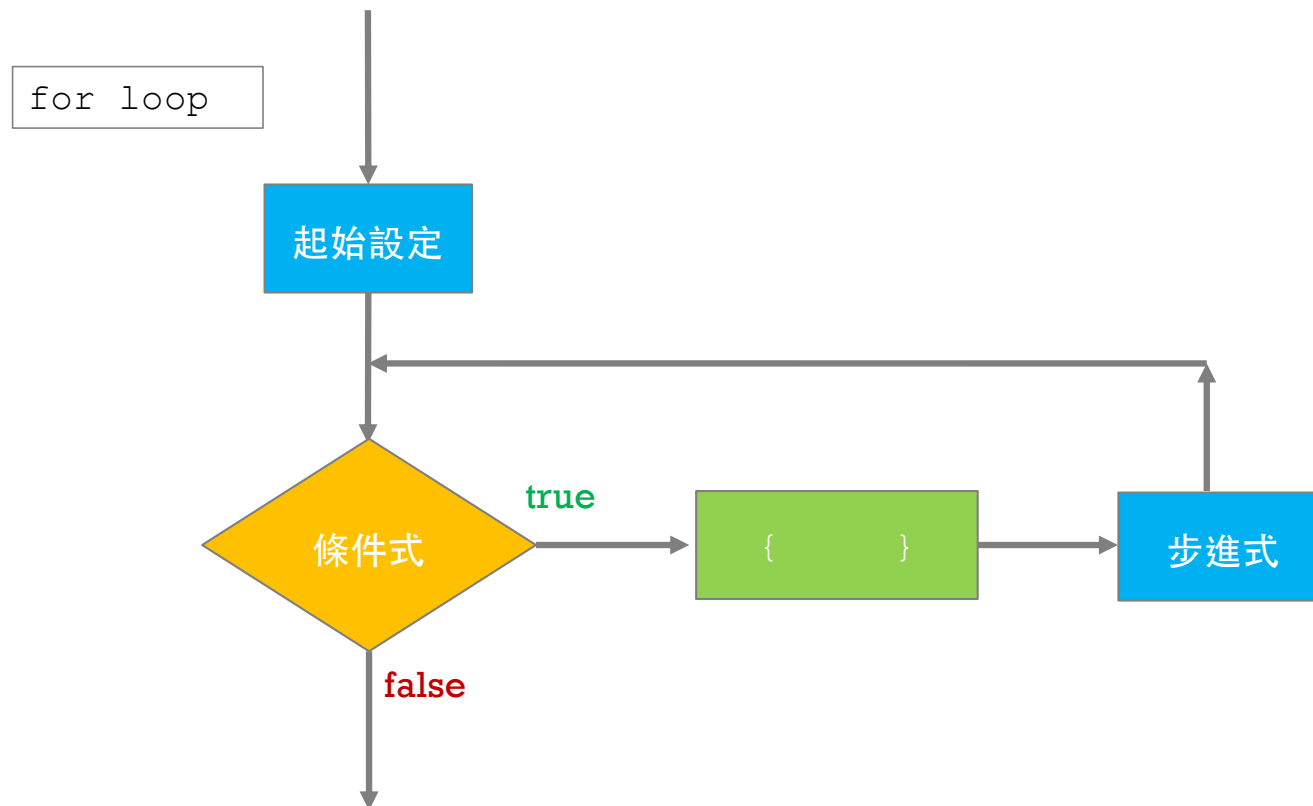
- 通常會有控制變數。注意，小刮號裡的兩個分號一定要有。

```
for(起始式 ; 條件式 ; 步進式) {  
    // 迴圈內容  
}
```

- 通常控制變數的值從 0 開始，小於「執行次數的值」。

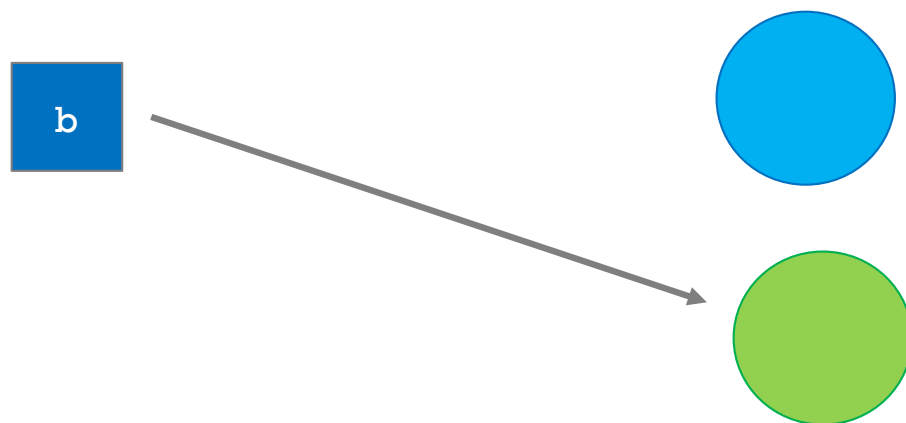
```
for(let i=0; i<8; i++) {  
    console.log(i);  
}
```





```
<style>
  .rect {
    position: relative;
    width: 800px;
    height: 600px;
    background-color: rgb(248, 232, 232);
    border: 1px solid black;
  }
  .ball {
    position: absolute;
    width: 50px;
    height: 50px;
    border-radius: 50%;
    background-color: rgb(245, 52, 52);
    color: white;
    font-weight: 800;
    line-height: 50px;
    text-align: center;
    border: 1px solid black;
  }
</style>
```

```
<div class="rect"></div>
<script>
  const rect = document.querySelector(".rect");
  for (let i = 0; i < 10; i++) {
    const b = document.createElement("div");
    b.className = "ball";
    b.innerHTML = i + 1;
    b.style.left = 60 * i + "px";
    b.style.top = "50px";
    rect.appendChild(b);
  }
</script>
```



\* **b** 變數會指向不同的 **DOM** 元素

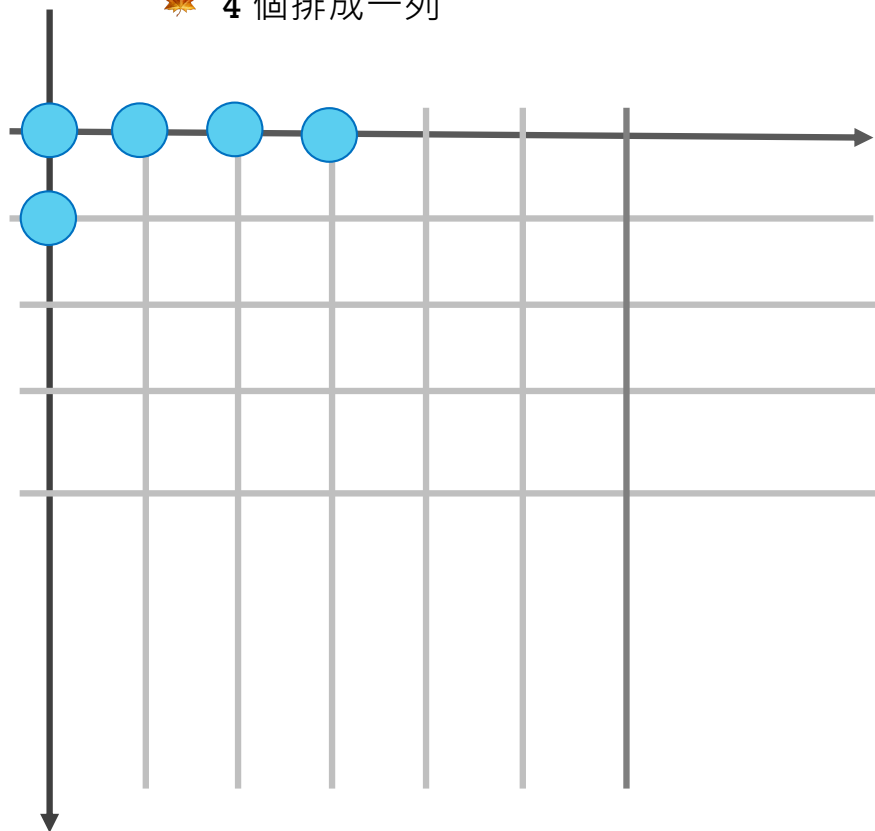
// 另一種作法

```
for(i=0; i<16; i++){
    let left = i*50, txt = i+1;
    rect.innerHTML += `<div class="ball"
                        style="left: ${left}px; top: 100px;">${txt}</div>`;
}
```

// 巢狀迴圈

```
let b, i, k, n=1;
for(k=0; k<6; k++) {
    for (i = 0; i < 10; i++, n++) {
        b = document.createElement('div');
        b.className = 'ball';
        b.innerHTML = n;
        b.style.left = i * 50 + 'px';
        b.style.top = k*60 + 'px';
        rect.appendChild(b);
    }
}
```

🍁 4 個排成一列



(x, y)

(0, 0)

(1, 0)

(2, 0)

(3, 0)

(0, 1)

(1, 1)

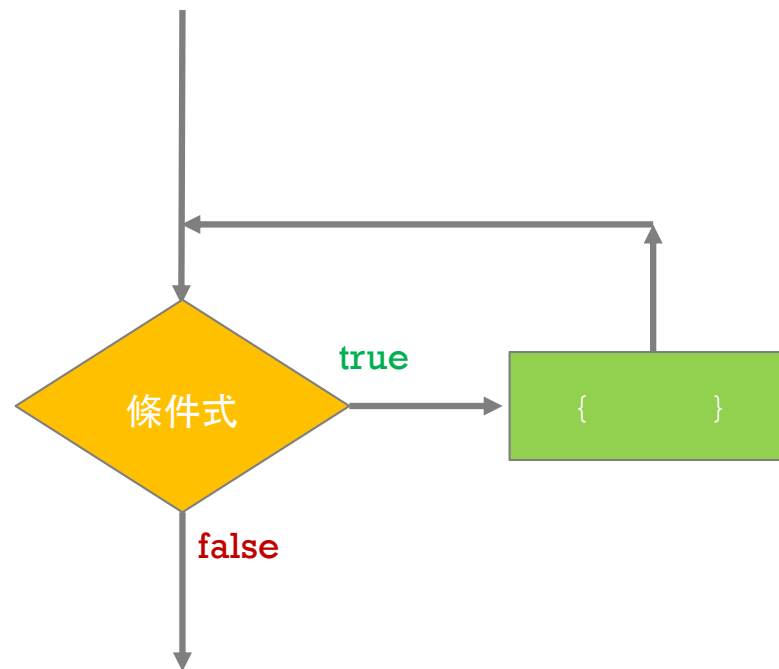
```
// 4 個一列
const rect = document.querySelector('#rect');
let b, i;
for(i=0; i<13; i++){
    let left = (i % 4) * 50,
        top = parseInt(i / 4) * 60,
        txt = i+1;
    rect.innerHTML += `<div class="ball"
        style="left: ${left}px; top: ${top}px;">${txt}</div>`;
}
```

## 3.4 while 迴圈

```
while (條件式) {  
    // 迴圈內容  
}
```

- **while** 迴圈和 **for** 迴圈可以互換。

while loop



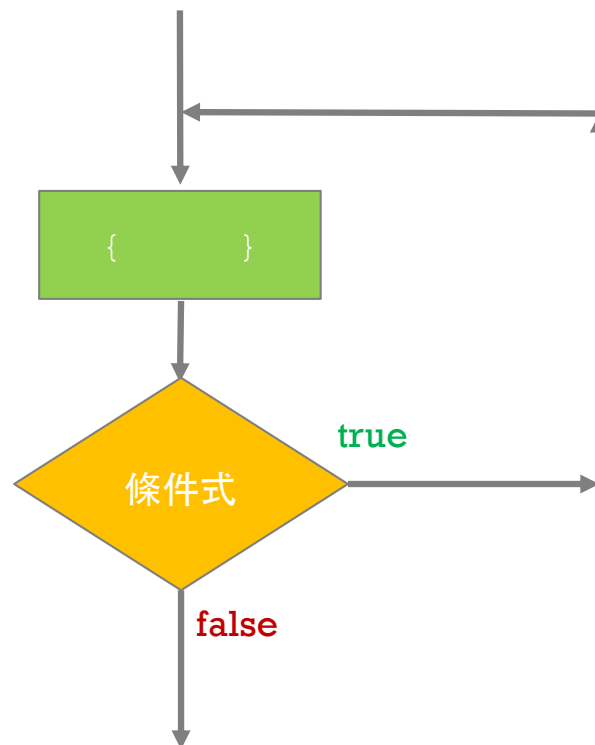


## 3.5 do/while 迴圈

- 先執行一次再說。
- 此迴圈很少用到。

```
do {  
    // 迴圈內容  
} while (條件式);
```

do/while loop





## 4 較複雜的資料類型

- 不是此三個類型的個體 ( `Number`, `Boolean`, `String` ) , 而且不為 `undefined` 或 `null` 者為**複雜類型** ( **參照**類型 ) 。
- **Object** : 一個名稱對應一個值的資料集合 ( `key-value pair` ) 。
- **Array** : 有順序的資料集合 。

## 4.1 Object 類型

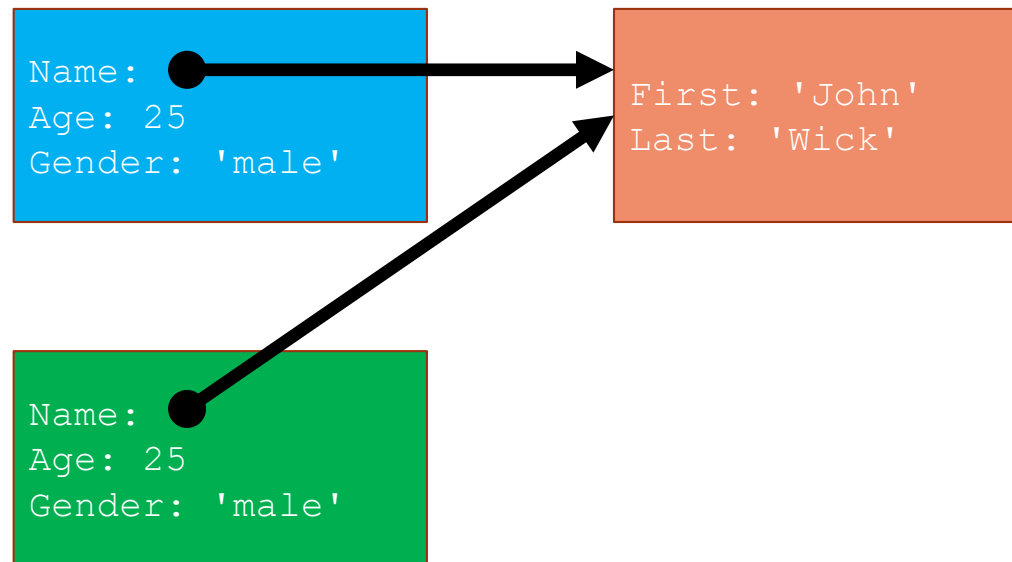
- Object 為鍵值對 ( key-value pair ) 的資料型態。
- 一個屬性名稱對應一個屬性值。
- 也稱為hash table。
- 「屬性名稱」為字串，定義時不應重複，否則會發生覆蓋。
- 「屬性名稱」也可以是 Symbol 類型 ( ES6 ) 。
- Object 的功能通常用來當作「字典」。
- 其 key-value pair 的 value 也可以是 function，而成為有「方法」的物件。
- Object 的特性是一個屬性名稱對應一個屬性值，重點不是順序，無需理會名稱的順序。

```
const obj1 = {a: 12, d: 'dog', b: true};
console.log(obj1);
const obj2 = {a:12, d: 'dog', b: true, a: 56}; // 重複key
console.log(obj2);
```

```
const obj = {a: 12, d: 'dog', b: true};
const obj2 = {
  "a": 12,
  "d": 'dog',
  "b": true,
};
obj['m'] = 'monkey'; // 中括號表示法
obj.t = 'tiger';     // 點表示法
delete obj.d;        // 用 delete 刪除
console.log(obj);
```

比較「中括號表示法」與「點表示法」

展開複製為淺層複製



## 4.2 for/in 迴圈

- for/in 列舉的變數，取得的是「屬性名稱」（類型為 **String**）。

```
<div class="rect"></div>
<script>
  const rect = document.querySelector(".rect");
  const person = {
    age: 25,
    name: "David",
    gender: "male",
  };
  for (let s in person) {
    rect.innerText += `${s}: ${person[s]}\n`;
  }
</script>
```



## 4.3 Array 類型

- **Array** 為有序的資料集合，以索引為取值的 **key**。
- 索引值從 0 開始。
- 索引值的內部運作是以「字串」的方式運作。

```
const ar = [12, 'abc', 77, -6];
for(let i=0; i<ar.length; i++) {
    console.log( i + ':' + ar[i] );
}
ar['3'] = 100;
for(let s in ar) {
    console.log( s + ':' + (typeof s) + '::' + ar[s] );
}
```

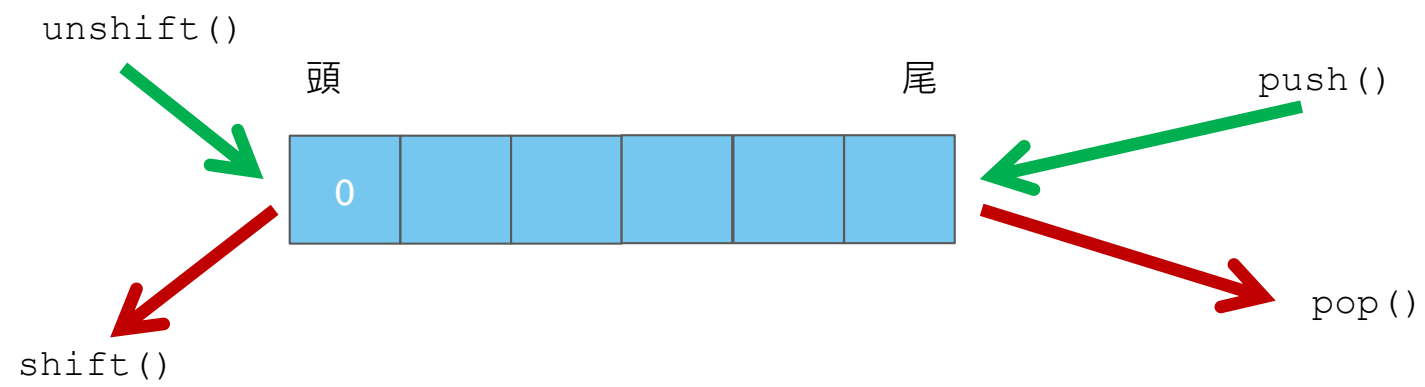
# for/of 迴圈

- **for/of** 用在可迭代的 ( **iterable** ) 物件上 ( 例如 **Array** ) 。
- 其變數可以一個、一個取得物件裡的元素。
- **for/of** 不能使用在 **Object** 上。

```
<div id="rect"></div>
<script>
  const rect = document.querySelector('#rect');
  const ar = [12, 'abc', 77, -6];
  for(let v of ar) {
    rect.innerHTML += v + '<br>';
  }
</script>
```

\*\* 標示紅色表示該方法會讓陣列本身改變

Array的方法	說明
concat (陣列)	和某陣列串接建立新的陣列。
indexOf (元素值, [索引])	取得元素值的索引，若找不到該元素值回傳 -1。
join ([黏著符號])	陣列以黏著符號串接為字串。
lastIndexOf (元素值)	由尾端取得元素值的索引，若找不到該元素值回傳 -1。
pop ()	彈出，從尾端取出一元素值。
push (元素值)	推入，從尾端加入一元素值。
reverse ()	陣列順序反轉。
shift ()	從前端取出一元素值。
slice (索引B, 索引E)	由索引B到索引E複製部份陣列 ( 不包含索引E )。
sort ()	排序。
splice (索引, 刪除量, [加入])	插入、刪除、替換元素。
unshift (元素值)	從前端加入一元素值。



- 排序是以字串 ( unicode字碼 ) 為順序。

```
const str = '到底需要日曬多久才能幫助人體獲得足夠的維生素D';
const ar = str.split('');
ar.sort();
console.log(ar);
const br = [35, 6, 78, 12, 54, 9];
br.sort();
console.log(br);
```

- 自訂排序規則，數值由小到大：

```
const br = [35, 6, 78, 12, 54, 9];
br.sort(function (a, b) {
    return a - b;    // 負值對調
});
console.log(br);
```

## ▪ splice() 的用法

```
<div id="rect"></div>
<script>
  const rect = document.querySelector('#rect');
  const ar = ["a", "b", "c", "d", "e", "f", "g", "h", "i"];
  let br = ar.slice();

  // 移除
  rect.innerHTML += br.splice(2, 3) + '<br>';
  rect.innerHTML += br + '<br>';

  br = ar.slice();
  // 新增, 插入元素
  rect.innerHTML += br.splice(2, 0, 'bill', 'peter') + '<br>';
  rect.innerHTML += br + '<br>';

  br = ar.slice();
  // 取代
  rect.innerHTML += br.splice(2, 2, 'bill', 'peter') + '<br>';
  rect.innerHTML += br + '<br>';
</script>
```

## ▪ `forEach()` 的用法

```
<div id="rect"></div>
<script>
  const rect = document.querySelector('#rect');
  const people = [
    { name: 'david', age: 25, id: 'A006' },
    { name: 'bill', age: 27, id: 'A009' },
    { name: 'peter', age: 23, id: 'A011' },
  ];
  people.forEach(function(val, ind){
    rect.innerHTML += ind + ': ' + val.name + '<br>';
  });
</script>
```

## 4.4 陣列的複製

- 淺層複製 ( 單層複製 )
- 可使用展開運算子 ( ... )

```
const ar = [12, 99, 'aaa', ['bill', 25, 'male'] ];
const br = ar;
const cr = ar.slice(); // first level copy
const dr = [...ar];     // first level copy (spread operator)
ar[3][0] = 'flora';
ar[0] = 100;

console.log('ar:', ar);
console.log('br:', br);
console.log('cr:', cr);
```



## 4.5 Destructuring and Spread

```
const p = {  
  name: 'bill',  
  age: 25,  
  gender: 'male'  
};  
const ar = [12, 99, 'aaa'];  
let first_name = 'shin';  
let last_name = 'lin';  
let {name, age} = p;           // 展開設定，解構賦值  
let p2 = {first_name, last_name};  
let ar2 = [1, 2, ...ar, 7];    // 展開陣列  
let [a, b] = ar;               // 展開設定，解構賦值
```



# 5 自訂函式

- 基本定義方式（傳統方式）：

```
function 函式名(形式參數列) {  
    // 內容  
    return 回傳值;  
}
```

```
// 簡單函式  
function myFunc() {  
    console.log('hi');  
}  
myFunc();
```

```
// 使用參數
function myFunc(a, b) {
  console.log('a:' + a);
  console.log('b:' + b);
  console.log('arguments:' + arguments);
  console.log(JSON.stringify(arguments));
}
myFunc(7, 9, 12);

// ES6
function myFunc2(a, ...b) {
  console.log('a:' + a);
  console.log('b:' + b);
}
myFunc2(7, 9, 12);

// ES6 形式參數可以設定預設值
```

```
// 複雜類型為參數時  
function func(ar) {  
    ar.push('abc');  
}  
const bc = [5, 6, 7];  
func(bc);  
console.log(bc);
```

- 參數為複雜類型時，機制同設定 ( = ) 。
- 回傳值為複雜類型時，機制同參數傳遞和設定 ( = ) 。
- 回傳值只能有一個，若要回傳多個值可以為 **Array** 或 **Object** 類型。
- 函式可以看成是特殊的物件。

## 5.1 區域變數與全域變數

- 全域變數指的是在頂層範圍（不包含在任何 **function** 裡），宣告的變數。
- 用 **var** 宣告全域變數時，該變數會變成 **window** 物件的屬性。用 **let** 宣告則否。
- 區域變數的領域（**scope**）只存在函式裡面，函式以外的範圍，看不到該變數。
- 形式參數是區域變數。
- 當某個區域在使用某個變數時，在區域內找不到該變數，則會往外一層區域尋找。

## 5.2 以變數設定方式建立函式

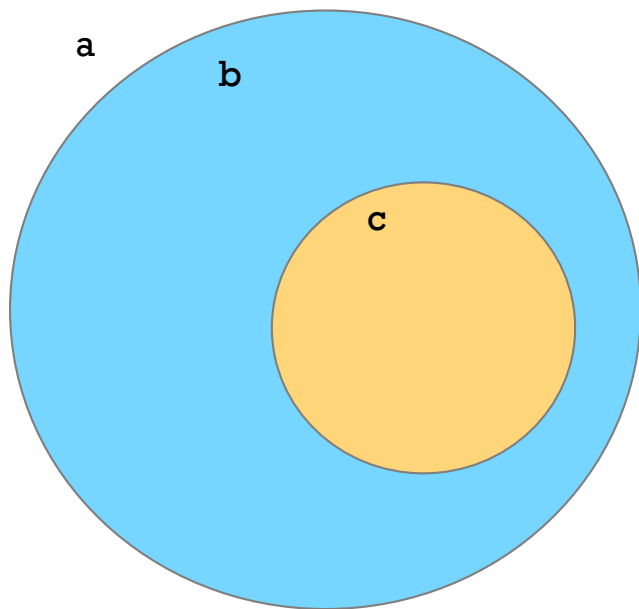
```
const func = function (name='Bill', age=20) {  
  //return {name: name, age: age};  
  return {name, age}; // ES6  
};  
console.log(func('John', 30));  
console.log(func('Kevin'));
```

```
// 直接執行的匿名函式
// 區域變數
const ar = [4, 5, 6];
(function() {
    const ar = [9, 8, 7];
    while(ar.length) {
        console.log( ar.pop() );
    }
})();
console.log(ar);
```

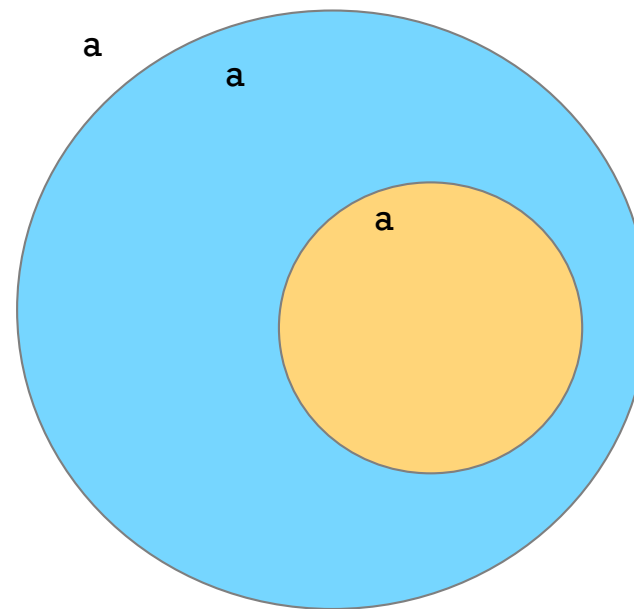
```
const a = 1;
(function() {
    const b = 2;
    (function() {
        const c = 3;
        console.log('a =', a);
        console.log('b =', b);
        console.log('c =', c);
    })();
})();
```



不同變數領域 (scope) 的不同名稱的變數



不同變數領域 (scope) 的同名變數



## 5.3 Arrow functions ( 箭頭函式, ES6 )

```
function f1(name, age) {  
}  
  
let f2 = (name, age) => {  
};  
  
const f3 = ()=>{  
};  
  
const f4 = n => n*n;  
  
const f5 = function(n) {  
    return n*n;  
};
```



## 6. 間隔時間觸發函式

- `setTimeout()`：間隔一段時間後觸發一次

```
console.log( new Date() );  
setTimeout(() => {  
    console.log( new Date() );  
}, 3000);
```

- `setInterval()`：間隔一段時間後重複觸發

```
function traceTime() {  
    console.log(count, new Date() );  
    if(--count <= 0)  
        clearInterval(iid);  
}  
var iid = setInterval(traceTime, 1000);  
var count = 6;
```

```
// 使用 let 宣告控制變數，使每一輪執行都有獨立的變數
for(let i=0; i<7; i++){
    setTimeout(function(){
        console.log(i);
    }, i*1000);
}
```



# 7. 表示時間點的物件 **Date**

```
const d = new Date();
console.log(d);
console.log(d.getFullYear());
console.log(d.getMonth()); // from 0 to 11
console.log(d.getDate());
console.log(d.getDay());
console.log(d.getHours());
console.log(d.getMinutes());
console.log(d.getSeconds());
console.log(d.getTime()); // 1970年至今的毫秒數
console.log(Date.now()); // 1970年至今的毫秒數
console.log(new Date('2023-09-20')); // 以格林威治時間解析, GMT
console.log(new Date('2023/09/20')); // 以當地時間解析
```

```
<div id="rect">電子時鐘</div>
<script>
  const rect = document.querySelector('#rect');
  const run = () => {
    let now = new Date;
    rect.innerHTML = now.toString().split(' ')[4];
    setTimeout(run, 1000);
  };
  run();
</script>
```



## ■ 秒針

```
<div class="clock">
  <div class="hand">
    <div class="hand-sec"></div>
  </div>
</div>
<script>
  const sec_hand = document.querySelector(".hand");
  //sec_hand.style.transform = "rotate(30deg)";
  const runClock = () => {
    const now = new Date();
    sec_hand.style.transform =
      `rotate(${now.getSeconds() * 6}deg)`;
    setTimeout(runClock, 1000);
  };
  runClock();
</script>
```

```
<style>
  .clock {
    position: relative;
    width: 600px;
    height: 600px;
    border-radius: 50%;
    background-color: lightcyan;
    border: 1px solid black;
  }
  .hand {
    position: absolute;
    left: 300px;
    top: 300px;
  }
  .hand-sec {
    position: absolute;
    width: 2px;
    height: 300px;
    left: -1px;
    top: -300px;
    background-color: red;
  }
</style>
```



## 8. 數學物件

Math的常用方法	說明
<code>abs(x)</code>	求絕對值。
<code>atan2(y, x)</code>	三角函數反正切（垂直距離和水平距離求角度）。
<code>ceil(x)</code>	大於等於 <b>x</b> 的最小整數。
<code>cos(x)</code>	三角函數餘弦。
<code>floor(x)</code>	小於等於 <b>x</b> 的最大整數。
<code>max(x, y, z, ..., n)</code>	最大值。
<code>min(x, y, z, ..., n)</code>	最小值。
<code>pow(x, y)</code>	<b>x</b> 的 <b>y</b> 次方。
<code>random()</code>	0 到 1 之間的亂數（大於等於 0, 小於 1）。
<code>round(x)</code>	四捨五入求整數。
<code>sin(x)</code>	三角函數正弦。

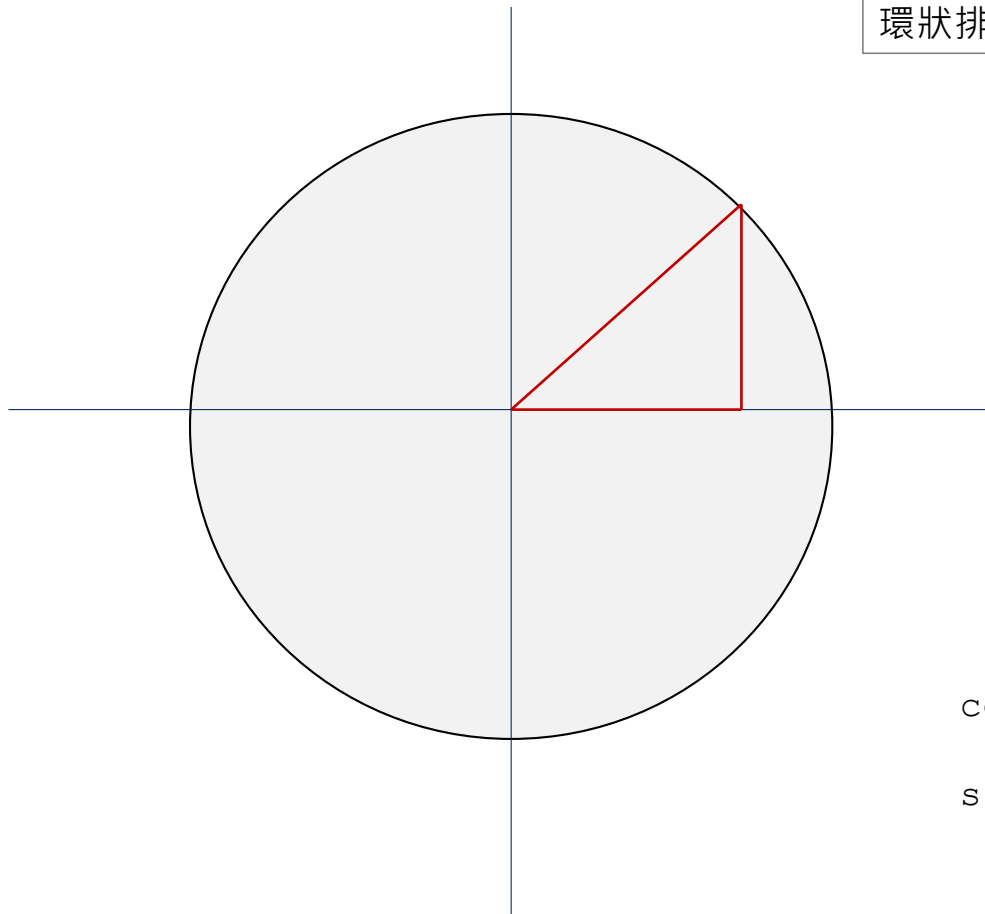
```
<div class="rect">隨機圓點</div>
<script>
  const rect = document.querySelector(".rect");

  for (let i = 0; i < 1000; i++) {
    const b = document.createElement("div");
    b.className = "ball";
    const size = 10 + Math.floor(Math.random() * 21);
    const x = Math.floor(Math.random() * 800);
    const y = Math.floor(Math.random() * 600);

    b.style.backgroundColor = `hsl(${bgc},100%,50%)`;
    b.style.left = x + "px";
    b.style.top = y + "px";
    b.style.height = b.style.width = size + "px";
    rect.appendChild(b);
  }
</script>
```

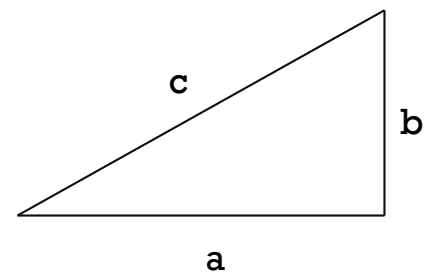
```
<style>
  .rect {
    position: relative;
    width: 800px;
    height: 600px;
    background-color: lightcyan;
    border: 1px solid black;
  }
  .ball {
    position: absolute;
    width: 20px;
    height: 20px;
    border-radius: 50%;
    background-color: red;
    text-align: center;
    border: 1px solid black;
  }
</style>
```

環狀排列



$$\cos \theta = ?$$

$$\sin \theta = ?$$



```
<div class="rect">環狀排列</div>
<script>
  const rect = document.querySelector(".rect");
  let b;
  const ballNum = 12;
  const angUnit = (Math.PI * 2) / ballNum;

  for (let i = 0; i < ballNum; i++) {
    b = document.createElement("div");
    b.className = "ball";
    b.innerHTML = i + 1;
    b.style.left =
      400 - 25 + Math.cos(i * angUnit - Math.PI / 3) * 260 + "px";
    b.style.top =
      300 - 25 + Math.sin(i * angUnit - Math.PI / 3) * 260 + "px";
    rect.appendChild(b);
  }
</script>
```

\* 圓心的 (400, 300), 半徑為 260, ball 的半徑為 25



## 9. window 物件

- **window** 的子物件：
  - **navigator** : 提供瀏覽器版本相關資訊
  - **screen** : 提供螢幕顯示尺寸相關資訊
  - **history** : 處理頁面上一頁或下一頁歷史記錄
  - **location** : 提供 **url** 相關資訊及頁面的重新導向。
  - **document** : 頁面對應 **DOM** 的物件



window 物件的常用方法	說明
alert()	顯示警示對話框。
blur()	讓視窗失焦。
clearInterval()	清除重複觸發某函式。
clearTimeout()	清除觸發一次某函式。
close()	關閉視窗。
confirm()	顯示確認對話框。
focus()	讓視窗取得焦點。
print()	執行列印。
prompt()	顯示詢問對話框。
setInterval()	間隔一段時間後重複觸發函式。
setTimeout()	間隔一段時間後觸發一次函式。

Document的常用屬性	類型 ( Chrome )	說明
URL	String	網址。
cookie	String	<b>Cookies</b> 。
doctype	DocumentType	文件類型。
domain	String	網域名稱。
forms	HTMLCollection	表單集合。
head	HTMLHeadElement	<b>Head</b> 元素。
images	HTMLCollection	圖片集合。
links	HTMLCollection	連結集合。
referrer	String	從哪兒來。
title	String	標頭名稱。

Document的常用方法	說明
<code>getElementById()</code>	依 <b>id</b> 取得元素。
<code>getElementsByClassName()</code>	依 <b>css</b> 類別名稱取得元素。
<code>getElementsByName()</code>	依 <b>name</b> 屬性名稱取得元素。
<code>getElementsByTagName()</code>	依標籤名稱取得元素。
<b><code>querySelector()</code></b>	依選擇器字串取得一個元素。(ES5)
<b><code>querySelectorAll()</code></b>	依選擇器字串取得多個元素。(ES5)



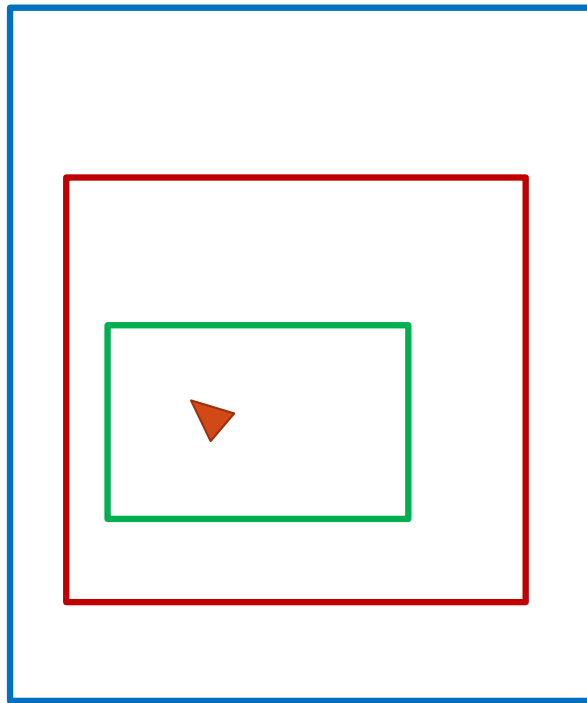
# 10. 事件處理

標籤的事件處理器	說明
onclick	單擊滑鼠左鍵。
ondblclick	雙擊滑鼠左鍵。
onmousedown	按下滑鼠左鍵時。
onmousemove	滑鼠在元素上移動時。
onmouseover	滑鼠移入元素時。
onmouseout	滑鼠移開元素時。
onmouseup	放開滑鼠左鍵時。
onkeydown	按下按鍵時，用於<body>。
onkeypress	按住按鍵時重複觸發，用於<body>。
onkeyup	放開按鍵時，用於<body>。
onload	內容載入後，使用於<body>、<object>。
onresize	文件大小改變時。
onscroll	文件捲動時。
onblur	失焦時，用於表單內的元素。
onchange	內容改變時，用於<input>、<select>、<textarea>
onfocus	取得焦點時，用於表單內的元素。
onreset	重置時，用於表單。
onselect	選取部份內容時，用於<input>、<textarea>。
onsubmit	送出表單時。

```
<button onclick="dosomething(event)">Hello</button>
<script>
  const btn = document.querySelector('button');
  function dosomething(evt) {
    console.log('1');
  }
  btn.onclick = function() {
    console.log('2');
  };
  btn.addEventListener('click', function() {
    console.log('3');
  });
  btn.addEventListener('click', function() {
    console.log('5');
  });
  const listener = (event) => {
    console.log(event);
  };
  btn.addEventListener('click', listener);
</script>
```

🍀 onclick 只能選一種使用

## 10.1 事件浮出模型



## 10.1 事件浮出模型

```
<div class="rect">
  <div class="ball"></div>
</div>
<script>
  const rect = document.querySelector(".rect");
  const ball = document.querySelector(".ball");

  const handler = (event) => {
    console.log("target:", event.target);
    console.log("currentTarget:", event.currentTarget);
    console.log(event.eventPhase);
  };

  ball.addEventListener("click", handler);
  rect.addEventListener("click", handler);
  document.addEventListener("click", handler);
  window.addEventListener("click", handler);
</script>
```

```
<style>
  .rect {
    position: relative;
    width: 800px;
    height: 600px;
    background-color: lightcyan;
    border: 1px solid black;
  }
  .ball {
    position: absolute;
    width: 150px;
    height: 150px;
    border-radius: 50%;
    background-color: yellow;
    text-align: center;
    border: 1px solid black;
    left: 100px;
    top: 100px;
  }
</style>
```



- `<a>` 標籤的假連結
- 在 `<a>` 標籤的 `href` 屬性使用 JavaScript

```
<a href="javascript: do_something()">text</a>
```

```
<a href="javascript: fn()">Hello</a>
<script>
  function fn(){
    console.log('href');
  }
  document.querySelector('a').onclick = function(event){
    console.log('onclick');
    event.preventDefault();
  }
</script>
```



# 11. try/catch

```
let str1 = '{"a": 123}';  
let str2 = '{a: 123}';  
let obj;  
try {  
    obj = JSON.parse(str2);  
} catch(error) {  
    //console.log(error);  
    console.log(error.message);  
    console.log(error.stack);  
} finally {  
    console.log('一定會被執行');  
}
```



## 12. 表單

```
<form name="form1" method="post" action="">

  <label for="user">User:</label>
  <input type="text" name="user" id="user"><br>

  <label for="password">Password:</label>
  <input type="password" name="password" id="password"><br>

  <input type="submit">
</form>
```

- 取得表單的元素

```
document.form1;  
document.forms[0];  
  
document.forms[0].user.value;  
document.forms[0].elements;  
document.forms[0].elements[0];  
document.forms[0].elements[0].value;  
document.forms[0].elements['user'].value;  
document.forms[0].password.value;  
document.forms[0].password.setAttribute('type', 'text');  
document.forms[0].password.setAttribute('type', 'password');
```



## 13. 正規表示法 ( 補充教材 )

- 正規表示法 ( **Regular Expression** 或稱「正則表達式」 ) 的目的是做文字的比對和尋找，在文字處理上非常重要。
- 現在，**JavaScript** 和其它許多程式語言也都支援正規表示法。**JavaScript** 裡使用的是 **RegExp** 物件。
- **RegExp** 物件可以搭配 **String** 物件的 **match**、**replace**、**search** 和 **split** 方法一起使用。
- **RegExp** 物件可以直接使用「/」包裹的方式定義。
- 練習場：<https://regex101.com/>

```
var re1 = /\sbe\s/i;  
var re2 = new RegExp( '\\sbe\\s', 'i');
```



```
var str = "b be bEAch bead Beaker BEAN bee being abbey abet";  
var re = /\sbe/ig; // remove 'g' and try again  
console.log(str.search(re));  
console.log(str.match(re));  
console.log(str.replace(re, "***"));  
console.log(str.split(re));
```

# 單一字元表示法

表示法	說明	範例
\d	數字0~9	/\d\d/ 符合者為 '22' ; '2c' 則不符合
\D	「非」數字	/\D\D/ 符合者為 'ac' ; '2c'則不符合
\s	一個空白 ( space )	/a\sbar/ 符合者為 'a bar' ; 'abar' 則不符合
\S	「非」空白	/a\Sbar/ 符合者為 'a-bar' ; 'abar' 和 'a bar' 不符合
\w	字母、數字或底線 ( _ )	/c\w/ 符合者為 'c7' ; 'c#' 和 'c-' 不符合
\W	「非」字母、數字或底線	/c\W/ 符合者為 'c%' ; 'ca' 和 'c_' 不符合
.	任何字元 ( 不包含換行 )	/a../ 符合者可為 'a12' 、 'ap+' 、 'a##'
[]	中括號中任一字元	/b[ae]d/ 符合者可為 'bad' 、 'bed'
[^]	不包含中括號中任一字元	/b[^ae]d/ 符合者可為 'b-d' 、 'bod' ; 'bad' 和 'bed' 不符合

# 多字元表示法

表示法	說明	範例
*	重複0次或多次	/lo*p/ 符合者可為 'lp'、'lop'、'loop'、'looop'
?	重複0次或1次	/lo?p/ 符合者為 'lp'、'lop'
+	重複1次或多次	/lo+p/ 符合者可為 'lop'、'loop'、'looop'
{n}	重複n次	/ba{2}d/ 符合者為 'baad'
{n, }	重複n次或以上	/ba{2,}d/ 符合者可為 'baad'、'baaad'
{n, m}	重複n次至m次之間	/ba{1,2}d/ 符合者為 'bad'、'baad'

- 上表裡的表示符號又稱為「貪婪計量子 ( Greedy quantifiers )」，會儘量找尋較長的字串。例如表示式為「lo\*」，當搜尋的對象為 "looop" 時，搜尋到的會是 "looo"，而不是 "loo"、"lo" 或 "l"。
- 「貪婪計量子」後面接個「?」時，會變成「自閉計量子 ( Reluctant quantifiers )」儘量找尋較短的字串。例如表示式為「lo+?」，當搜尋的對象為 "looop" 時，搜尋到的會是 "lo"，而不是 "loo" 或 "looo"。

# 位置及其它表示法

表示法	說明	範例
^	字首	/^pos/ 符合者可為 'pose' ; 'apos' 不符合
\$	字尾	/ring\$/ 符合者為 'spring' ; 'ringer' 不符合
	或	/jpg png/
()	子表示法	/img\.(jpg png)/

- **RegExp** 物件有兩個方法 **exec** 和 **test** 。
- **exec** 方法通常是用來搜尋字串中符合字模的子字串。
- **test** 方法是用來測試字串是否符合字模。

```
var str = "b bEAch bead Beaker";  
var re = /\sbe/ig;  
var obj;  
while ( obj = re.exec(str) ) {  
    console.log( obj );  
    console.log(re.lastIndex + ' -----');  
}
```



## 14. AJAX ( 需要 WEB SERVER )

- 使用 XMLHttpRequest

```
function doAjax() {  
    var xhr = new XMLHttpRequest();  
    xhr.onload = function() {  
        document.getElementById('info').innerHTML = this.responseText;  
    };  
    xhr.open('GET', 'hello_ajax.txt', true); // 第三個參數為 async  
    xhr.send();  
}
```

- 使用 `fetch()`
- [https://developer.mozilla.org/zh-TW/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/zh-TW/docs/Web/API/Fetch_API/Using_Fetch)

```
<script>
function doAjax() {
    fetch('hello_ajax.txt')
        .then(response=>{
            return response.text();
        })
        .then(txt=>{
            document.getElementById('info').innerHTML = txt;
        });
}
</script>
```



# 參考網站

- <https://www.w3schools.com/>
- <https://developer.mozilla.org/zh-TW/docs/Web>

*Thank you for listening*

