

JavaScript

prompt 是什麼？

`prompt()` 是 JavaScript 提供的 **瀏覽器內建對話框**，用來讓使用者輸入資料。

```
let name = prompt("請輸入你的名字：");
```

- `prompt()` 會跳出一個輸入框。
- 它會回傳使用者輸入的字串，或是 `null`（如果使用者按了取消）。

```
let age = prompt("你幾歲？");  
console.log("你輸入的年齡是：" + age);
```

String Interpolation

String interpolation（字串插值）是用來把變數或運算結果**嵌入到字串中**的一種方法。

JavaScript 使用 **template literals**（**樣板字串**）搭配 **反引號**（```）來達成字串插值。

```
let message = `Hello, ${name}!`;
```

- 使用 **反引號** ``` 包住整段字串。
- 變數或運算式放在 `${}` 中，就能插入到字串裡。

```
let name = prompt("你的名字是？");  
let greeting = `哈囉，${name}，歡迎來到我們的網站！`;  
console.log(greeting);
```

String Methods

取得字串的長度（字元數）。

```
let str = "Hello";  
console.log(str.length); // 5
```

轉成全部大寫 / 全部小寫。

```
let str = "Hello";  
console.log(str.toUpperCase()); // "HELLO"  
console.log(str.toLowerCase()); // "hello"
```

是否包含某段文字，回傳 `true` 或 `false`。

```
let str = "JavaScript is fun";  
console.log(str.includes("Script")); // true
```

回傳子字串第一次/最後一次出現的位置，找不到回傳 `-1`。

```
let str = "hello world";  
console.log(str.indexOf("o")); // 4  
console.log(str.lastIndexOf("o")); // 7
```

切出子字串，不會改變原本的字串。

```
let str = "abcdef";  
console.log(str.slice(1, 4)); // "bcd"
```

和 `slice` 類似，但不接受負數。

```
let str = "abcdef";  
console.log(str.substring(1, 4)); // "bcd"
```

用新字串取代指定部分。

```
let str = "I like cats";  
console.log(str.replace("cats", "dogs")); // "I like dogs"
```

```
let str = "ha ha ha";  
console.log(str.replaceAll("ha", "ho")); // "ho ho ho"
```

移除前後空白字元。

```
let str = " Hello ";  
console.log(str.trim()); // "Hello"
```

依照指定分隔符，回傳陣列。

```
let str = "a,b,c";  
let arr = str.split(",");  
console.log(arr); // ["a", "b", "c"]
```

取得特定位置的字元或 Unicode 編碼。

```
let str = "Hello";  
console.log(str.charAt(1)); // "e"  
console.log(str.charCodeAt(1)); // 101
```

判斷字串是否以指定開頭或結尾。

```
let str = "JavaScript";  
console.log(str.startsWith("Java")); // true  
console.log(str.endsWith("Script")); // true
```

一些 JavaScript 的「神奇運算」範例

```
console.log(20 + "5"); // "205" → 數字會被轉成字串再串接
```

✓ JS 中的 + 是「加法或字串串接」，只要其中一個是字串，就會變成字串。

```
console.log("20" - 5); // 15 → "20" 會被轉成數字
console.log("20" / 2); // 10
console.log("10" * 3); // 30
```

✓ 除了 +，其他運算符如 -, *, / 會自動將字串轉為數字（若能轉換）

```
console.log(5 + true); // 6 → true 被轉成 1
console.log(5 + false); // 5 → false 被轉成 0
```

```
console.log(null + 1); // 1 → null 被當作 0
console.log(undefined + 1); // NaN → undefined 無法轉為數字
```

JavaScript 的「隱式轉型」規則

表達式類型	結果	原因
"5" + 3	"53"	字串 + 數字 → 字串串接
"5" - 3	2	"5" 轉數字 → 數學運算
true + 1	2	true 變 1
false + 1	1	false 變 0
null + 1	1	null 變 0
undefined + 1	NaN	undefined 轉型失敗，結果為 NaN
[] + 1	"1"	空陣列轉空字串，然後做字串串接
[1] + 1	"11"	[1] 轉成 "1" 字串後再加上 "1"
{ } + 1	[object Object]1（非嚴格模式）	{ } 被當成物件，轉成字串後再加

在開發時避免踩雷的建議

1. 總是使用明確型別轉換

```
Number("5"); // 5
String(10); // "10"
```

```
Boolean(0); // false
```

2. 避免混用不同型別做運算

3. 使用 `===` 而非 `==` 比較值

- `===`：嚴格比較（值與型別都相同才回 `true`）
- `==`：會自動轉型比較，容易造成錯誤

```
0 == false // true 😱  
0 === false // false ✅  
+"42" //42  
-"42" //-42
```

✅ ?? Nullish Coalescing Operator

?? 用來提供「預設值」，但只在左邊是 `null` 或 `undefined` 時才會使用右邊的值。

不同於 `||`（邏輯或），`??` 不會把 `0`、`false`、`"` 視為空值。

```
let result = a ?? b;
```

若 `a` 是 `null` 或 `undefined`，結果就會是 `b`，否則就是 `a`。

```
let name = null;  
let username = name ?? "預設名稱";  
console.log(username); // 預設名稱
```

```
let count = 0;  
let finalCount = count ?? 10;  
console.log(finalCount); // ✅ 0（不會被替代）  
  
let count2 = count || 10;  
console.log(count2); // ⚠️ 10（因為 0 被當成 false）
```

- 使用者輸入為空時給預設值
- 避免不小心把 `0`、`false`、空字串 `""` 當成「空值」

✓ Arrow Function

// 傳統寫法

```
function add(a, b) {  
  return a + b;  
}
```

// 箭頭函數寫法

```
const add = (a, b) => a + b;
```

功能	寫法
無參數	<code>() => {...}</code>
一個參數	<code>x => {...}</code>
多個參數	<code>(x, y) => {...}</code>
單行運算，省略 <code>{}</code> 和 <code>return</code>	<code>x => x * 2</code>
回傳物件（加括號）	<code>() => ({ name: "James" })</code>

項目	Arrow Function	傳統 Function
<code>this</code> 綁定	不綁定自己的 <code>this</code>	有自己的 <code>this</code>
語法簡潔	✓	✗
適合回呼函數	✓	✓
建構函數用法	✗ 無法用 <code>new</code>	✓ 可用 <code>new</code>

✓ 常見的 `Math` 函式一覽

方法	功能說明	範例
<code>Math.abs(x)</code>	回傳絕對值	<code>Math.abs(-5)</code> → 5
<code>Math.round(x)</code>	四捨五入	<code>Math.round(4.6)</code> → 5
<code>Math.floor(x)</code>	無條件捨去（向下取整）	<code>Math.floor(4.9)</code> → 4
<code>Math.ceil(x)</code>	無條件進位（向上取整）	<code>Math.ceil(4.1)</code> → 5
<code>Math.trunc(x)</code>	去除小數點（僅保留整數部分）	<code>Math.trunc(4.9)</code> → 4
<code>Math.max(a, b...)</code>	取最大值（可多個參數）	<code>Math.max(1, 7, 3)</code> → 7
<code>Math.min(a, b...)</code>	取最小值	<code>Math.min(1, 7, 3)</code> → 1

<code>Math.random()</code>	產生 0~1 之間的隨機小數	<code>Math.random()</code> → 例如 0.38
<code>Math.pow(x, y)</code>	計算 x 的 y 次方	<code>Math.pow(2, 3)</code> → 8
<code>Math.sqrt(x)</code>	開根號	<code>Math.sqrt(25)</code> → 5
<code>Math.cbrt(x)</code>	開立方根	<code>Math.cbrt(27)</code> → 3
<code>Math.sign(x)</code>	傳回數字的正負 (1, -1 或 0)	<code>Math.sign(-20)</code> → -1

```
let num = Math.floor(Math.random() * 10) + 1;
console.log(num); // 1 ~ 10 之間的隨機整數
```

Array 的常見方法

`array.length = 2` //array的長度可以改!

方法	功能	範例
<code>includes()</code>	是否包含某元素	<code>[1, 2, 3].includes(2)</code> → true
<code>indexOf()</code>	回傳元素位置 (找不到是 -1)	<code>[1, 2, 3].indexOf(3)</code> → 2
<code>find()</code>	回傳符合條件的第一個元素	<code>[1,2,3].find(x => x > 1)</code> → 2
<code>findIndex()</code>	回傳符合條件的第一個索引	<code>[1,2,3].findIndex(x => x > 1)</code> → 1

方法	功能	範例
<code>forEach()</code>	逐一執行，不回傳新陣列	<code>arr.forEach(x => console.log(x))</code>
<code>map()</code>	建立新陣列 (轉換每個元素)	<code>[1,2].map(x => x*2)</code> → [2,4]
<code>filter()</code>	篩選符合條件的元素	<code>[1,2,3].filter(x => x>1)</code> → [2,3]
<code>reduce()</code>	累加、累乘等自訂累計運算	<code>[1,2,3].reduce((a,b)>=>a+b,0)</code> → 6

方法	功能	範例
<code>push()</code>	從尾端加入元素	<code>arr.push(4)</code>
<code>pop()</code>	從尾端移除元素	<code>arr.pop()</code>
<code>shift()</code>	從前端移除元素	<code>arr.shift()</code>
<code>unshift()</code>	從前端加入元素	<code>arr.unshift(0)</code>
<code>splice()</code>	插入、刪除或取代元素 (會改原陣列)	<code>arr.splice(1, 1)</code> 移除第 2 個
<code>sort()</code>	排序 (需小心數字排序)	<code>arr.sort((a, b) => a - b)</code>
<code>reverse()</code>	反轉陣列順序	<code>arr.reverse()</code>

方法	功能	範例
<code>slice()</code>	切出新陣列（不含結尾）	<code>arr.slice(1,3)</code>
<code>concat()</code>	合併陣列	<code>[1].concat([2,3])</code> → <code>[1,2,3]</code>
<code>join()</code>	用指定字串合併為一個字串	<code>[1,2].join("-")</code> → <code>"1-2"</code>

Array Destructuring

//基本語法

```
let arr = [10, 20, 30];
let [a, b, c] = arr;
```

```
console.log(a); // 10
console.log(b); // 20
console.log(c); // 30
```

//忽略某些值

```
let [first, , third] = [1, 2, 3];
console.log(third); // 3
```

//剩餘運算子

```
let [head, ...rest] = [1, 2, 3, 4];
console.log(head); // 1
console.log(rest); // [2, 3, 4]
```

```
let [a = 1, b = 2] = [];
console.log(a); // 1
console.log(b); // 2
```

//與函式作搭配

```
function getScores() {
  return [90, 80, 70];
}

let [math, eng, sci] = getScores();
```

Ways to access object

```
//使用「點記號」  
let person = { name: "James", age: 25 };  
console.log(person.name); // "James"
```

```
//使用「中括號」[]（適用於變數或動態 key）  
let key = "age";  
console.log(person[key]); // 25
```

Object Destructuring

```
//可以一次從物件中「取出多個屬性變數」  
let person = { name: "Alice", job: "Engineer", city: "Taipei", gender: "woman" };  
  
let { job, city } = person;  
console.log(job); // "Engineer"  
console.log(city); // "Taipei"
```

```
//剩餘運算子  
let { job, city, ...remainingProperties } = person;  
console.log(remainingProperties); // { name: "Alice", gender: "woman" }
```

```
//解構 + 更改變數名  
let { job: occupation } = person;  
console.log(occupation); // "Engineer"
```

```
delete person.city;  
console.log(person); // { name: "Alice", job: "Engineer" }
```

注意：`delete` 會直接改變原物件

🟪 Check if object has a property

```
console.log("name" in person); // true
console.log("age" in person); // false
```

```
person.hasOwnProperty("job"); // true
```

//不推薦

```
if (person.name !== undefined) { ... }
```

🟢 ?. (Optional Chaining Operator)

//安全地存取深層屬性，避免錯誤（undefined 時不報錯）

```
let user = {
  profile: {
    name: "John"
  }
};
```

```
console.log(user.profile?.name); // "John"
console.log(user.address?.city); // undefined（不會報錯）
```

//可搭配函數、陣列、物件

```
user.sayHi?.(); // 如果函數存在才執行
user.friends?.[0]; // 安全地取陣列第 0 項
```

✅ Object in JS ↔ Element in HTML

在 JavaScript 中，HTML 元素有一個 `.classList` 屬性，它是一個 類似陣列的物件

方法名稱	功能說明	回傳值	用途範例
<code>.add("className")</code>	加入指定的 class (如果還沒存在)	無 (undefined)	加上 <code>.active</code> 或 <code>.filled</code> 樣式

<code>.remove("className")</code>	移除指定的 class (如果存在)	無 (undefined)	拿掉 <code>.active</code> 或 <code>.filled</code> 樣式
<code>.contains("className")</code>	檢查該元素是否包含指定的 class	<code>true</code> / <code>false</code>	判斷是否要顯示某個樣式或狀態
<code>.toggle("className")</code>	有就移除，沒有就加入 (自動切換)	<code>true</code> (加上) <code>false</code> (移除)	做按鈕開關、喜歡愛心、暗黑模式等切換效果
<code>.toggle("className", condition)</code>	根據布林值強制切換 class	<code>true</code> / <code>false</code>	進階條件控制 toggle 行為



多個 class 可同時 `.add("a", "b")` 或 `.remove("a", "b")`

✓ JSON 是什麼？

JSON 全名是 **JavaScript Object Notation (JavaScript 物件表示法)**，是一種用來儲存與傳遞資料的格式。

它的格式長得很像 **JavaScript** 的物件，但實際上是一種純文字格式 (**string**)。



```
{
  "name": "James",
  "age": 25,
  "isStudent": true,
  "skills": ["JavaScript", "Python"]
}
```

這就是一個「JSON 物件」，它長得像 JS，但注意：

1. 所有 **key (鍵)** 都要用雙引號包住
2. 只能用基本資料型別：字串、數字、布林、null、陣列、物件

🔧 JSON 可以幹嘛？(用途)

用途	說明
🔄 資料交換格式 (最常用)	前端與後端溝通常用 JSON 傳資料
📁 儲存設定檔、資料檔	比如 <code>package.json</code> 、 <code>db.json</code>

 API 回傳格式	幾乎所有 Web API 都用 JSON 當資料格式
 前端模擬假資料 (mock data)	開發階段可用 JSON 模擬後端傳來的資料

//字串 → JavaScript 物件

```
let jsonStr = '{"name":"James","age":25}';
let obj = JSON.parse(jsonStr);
console.log(obj.name); // "James"
```

//JavaScript 物件 → 字串

```
let person = { name: "James", age: 25 };
let str = JSON.stringify(person);
console.log(str); // '{"name":"James","age":25}'
```

常見錯誤	原因
用單引號包 key	JSON 格式規定只能用雙引號
包含函數	JSON 無法儲存 function 或 undefined
不合法的逗號	末尾不能多加逗號 (和 JS 不一樣)

🟡 ES6 以後的「陣列迴圈」

//for...of (簡潔地遍歷陣列、字串)

```
let arr = ["a", "b", "c"];
for (let val of arr) {
  console.log(val); // a, b, c
}
```

//for...in (取出物件或陣列的鍵名)

```
let obj = { name: "Alice", age: 25 };
for (let key in obj) {
  console.log(key);    // name, age
  console.log(obj[key]); // Alice, 25
}
```

Higher-order Functions

forEach()

◆ **用途：**對陣列每一項執行指定動作，不回傳新陣列（只做事情）

◆ **語法：**

```
javascript
複製編輯
array.forEach((item, index, array) => {
  // 對每個 item 做某件事
});
```

◆ **範例：**

```
javascript
複製編輯
let arr = [1, 2, 3];
arr.forEach((n) => console.log(n * 2));
// 輸出：2, 4, 6
```

◆ **特點：**不能 `break`、不能 `return` 結果

map()

◆ **用途：**建立一個**新陣列**，每個元素為原始元素處理後的結果

◆ **語法：**

```
javascript
複製編輯
let newArr = array.map((item, index, array) => {
  return 處理後的值;
});
```

◆ **範例：**

javascript

複製編輯

```
let numbers = [1, 2, 3];
let squares = numbers.map(x ⇒ x * x);
console.log(squares); // [1, 4, 9]
```

filter()

◆ **用途：**建立一個**新陣列**，內容為符合條件的原始元素

◆ **語法：**

javascript

複製編輯

```
let newArr = array.filter((item, index, array) ⇒ {
  return 條件; // 回傳 true 才會留下來
});
```

//最常用的寫法只用一個參數

```
let nums = [1, 2, 3, 4, 5];
```

```
let even = nums.filter(n ⇒ n % 2 === 0); // 只用 element (n)
console.log(even); // → [2, 4]
```

//根據「索引」過濾

```
let words = ["a", "b", "c", "d"];
```

```
let skipFirst = words.filter((word, index) ⇒ index > 0);
console.log(skipFirst); // → ["b", "c", "d"]
```

//根據整個陣列的狀況來判斷

```
let nums = [3, 7, 9, 12];
```

```
let aboveAverage = nums.filter((num, i, arr) => {  
  let avg = arr.reduce((a, b) => a + b) / arr.length;  
  return num > avg;  
});  
console.log(aboveAverage); // → [9, 12]
```

reduce()

◆ **用途**：將陣列「**累積成單一值**」（常用來加總、乘積、合併物件）

◆ **語法**：

```
javascript  
複製編輯  
let result = array.reduce((accumulator, current, index, array) => {  
  return 累積邏輯;  
}, 初始值);
```

◆ **範例**（加總）：

```
javascript  
複製編輯  
let nums = [1, 2, 3];  
let sum = nums.reduce((acc, curr) => acc + curr, 0);  
console.log(sum); // 6
```

sort()

◆ **用途**：排序陣列（會**改變原陣列**）

◆ **語法**：

```
javascript  
複製編輯  
array.sort((a, b) => a - b); // 升序  
array.sort((a, b) => b - a); // 降序
```

◆ 範例：

```
javascript  
複製編輯  
let nums = [3, 1, 2];  
nums.sort((a, b) => a - b);  
console.log(nums); // [1, 2, 3]
```

◆ 注意：若不寫 compare function，會以「字串」排序（錯誤結果）

every()

◆ 用途：全部都符合條件時，回傳 `true`，否則回傳 `false`

◆ 語法：

```
javascript  
複製編輯  
array.every((item) => 條件);
```

◆ 範例：

```
javascript  
複製編輯  
let nums = [2, 4, 6];  
let allEven = nums.every(x => x % 2 === 0);  
console.log(allEven); // true
```

some()

◆ 用途：只要有一個符合條件，就回傳 `true`，否則為 `false`

◆ 語法：

```
javascript  
複製編輯  
array.some((item) => 條件);
```


◆ 範例：

```
javascript
複製編輯
let nums = [1, 3, 4];
let hasEven = nums.some(x => x % 2 === 0);
console.log(hasEven); // true
```

DOM基礎

querySelector()

◆ 功能：選取第一個符合 CSS 選擇器的元素

◆ 語法：

```
document.querySelector("選擇器");
```

◆ 範例：

```
let title = document.querySelector("h1");
let btn = document.querySelector(".submit");
let input = document.querySelector("#username");
```

querySelectorAll()

◆ 功能：選取所有符合的元素（**NodeList**，類陣列）

◆ 語法：

```
document.querySelectorAll("選擇器");
```

◆ 範例：

```
let items = document.querySelectorAll(".todo-item");
items.forEach(item => console.log(item.textContent));
```

getElementById()


◆ 功能：根據「id」選取單一元素

◆ 語法：

```
document.getElementById("id名稱");
```

◆ 範例：

```
let header = document.getElementById("main-title");
```

◆  注意：不用加 #，只能選一個，速度比 `querySelector()` 快

`createElement()`


◆ 功能：建立一個新的 DOM 元素，可加入網頁中

◆ 語法：

```
let newTag = document.createElement("tagName");
```

◆ 範例：


```
let newP = document.createElement("p");
newP.textContent = "這是一段新文字";
document.body.appendChild(newP); // 加入到網頁中
```

非常棒的問題！

`innerHTML`、`innerText`、`textContent` 這三個屬性看起來很像，但其實差異蠻大，掌握這些可以讓你精準操作 DOM 中文字與內容。以下是詳細說明、比較與範例：

innerHTML & innerText & textContent

`innerHTML`

- 讀取或設定 HTML 元素的「整個內容（包括 HTML 標籤）」
- 可以插入、解析 HTML 標籤（ 也會執行 script）

```
<div id="box">Hello <strong>World</strong></div>
```

```
let box = document.getElementById("box");
console.log(box.innerHTML);
// "Hello <strong>World</strong>"
```

```
box.innerHTML = "<p>新段落</p>";
```

使用時機：

- 需要插入 HTML 標籤時
- 取整段 HTML 原始碼內容

innerText

- 讀取或設定**可見的「純文字」**內容
- 會受到 CSS 的 `display: none` 或 `visibility: hidden` 影響（隱藏就抓不到）
- 自動格式化（例如換行）

```
<div id="box">Hello <strong>World</strong></div>
```

```
console.log(box.innerText);  
// "Hello World"
```

使用時機：

- 需要顯示給使用者看的內容
- 會考慮樣式、排版（適合模擬「看到的畫面」）

textContent

- 讀取或設定**所有文字內容**（不管有沒有被隱藏）
- **不解析 HTML 標籤**，也不格式化
- `❤` 要在textContent改的話要直接寫"♥"，否則無法顯示

```
<div id="box">Hello <strong>World</strong></div>
```

```
console.log(box.textContent);  
// "Hello World"
```

 **比較總表：**

屬性	是否包含 HTML 標籤	是否受 CSS 影響	是否保留格式 (換行)	使用建議
<code>innerHTML</code>	✅ 會	❌ 不影響	❌ 無格式	插入或取得 HTML 原始碼
<code>innerText</code>	❌ 只取顯示的純文字	✅ 有影響	✅ 自動換行	取畫面上「看得到」的文字
<code>textContent</code>	❌ 只取純文字	❌ 不影響	❌ 沒有換行	取原始純文字內容最快速

✅ Event Methods

`addEventListener()`

◆ **功能：**為元素註冊事件監聽器 (listener)，當事件發生時執行指定的函式。

◆ **語法：**

```
element.addEventListener("事件類型", 處理函式);
```

◆ **範例：**

```
let btn = document.querySelector("button");
btn.addEventListener("click", () => {
  alert("你點到我了!");
});
```

常見錯誤

```
scaryStoryBtn.addEventListener("click", displayStory("scary")); //❌ 會直接執行
scaryStoryBtn.addEventListener("click", () => displayStory("scary")); //✅
```

`removeEventListener()`

◆ **功能：**移除先前透過 `addEventListener()` 綁定的事件。

◆ **注意：**只能移除同一個函式 (不可用匿名函式)

◆ **範例：**

```
function greet() {
  alert("Hi!");
}
```

```
btn.addEventListener("click", greet);  
// 移除監聽  
btn.removeEventListener("click", greet);
```

各種事件的觸發對象與應用

事件類型	說明	常見用途
click	滑鼠點擊	按鈕、開關
mouseover	滑鼠移入元素	hover 效果、提示
mouseout	滑鼠移出元素	還原樣式
keydown	按鍵按下（可偵測方向鍵）	遊戲控制、即時輸入檢查
keyup	放開鍵盤	表單驗證、輸入完成提示
submit	表單送出	驗證輸入、阻止預設送出
DOMContentLoaded	DOM 結構完成（不等圖片、影片）	

change vs input 差異？

事件	適用情境	觸發時機
change	下拉選單、checkbox、blur 後	改變值 + 離開輸入框 or 選項改變
input	即時輸入偵測（打字時）	使用者每輸入一字就會觸發

event.preventDefault() 是什麼？

它是 JavaScript 中的一個方法，用來阻止事件的「預設行為」。

HTML 元素本來就會有某些「預設行為」：

元素	預設行為範例
	會跳轉到超連結
<form>	會重新整理頁面並送出資料
<input type="submit">	提交整個表單
拖曳事件	預設可能觸發瀏覽器拖拉功能

preventDefault() 就是拿來 攔住這些預設行為，改由我們自己用 JS 處理。

```
//基本語法
element.addEventListener("事件", function(e) {
  e.preventDefault(); // 阻止預設行為
});
```

```
<form id="myForm">
  <input type="text" required>
  <button type="submit">送出</button>
</form>
```


```
//阻止表單提交（最常見）
let form = document.getElementById("myForm");

form.addEventListener("submit", function(event) {
  event.preventDefault(); // 表單不會真的送出
  alert("表單送出前先檢查喔！");
});
```

Event Bubbling

事件冒泡是指當你在某個元素上觸發事件時，該事件會從內層元素一路往外層傳遞，直到 document 為止。

點擊小按鈕 → 冒泡到父層 `div` → 再冒到 `body` → 最後到 `document`

就像泡泡往上冒一樣 

```
<div id="outer">
  <button id="inner">Click me</button>
</div>
```

```
document.getElementById("outer").addEventListener("click", () => {  
  console.log("外層被點了！");  
});  
  
document.getElementById("inner").addEventListener("click", () => {  
  console.log("內層被點了！");  
});
```


► 點按按鈕時：

```
cpp  
複製編輯  
內層被點了！  
外層被點了！ // 因為事件冒泡到了 outer
```

```
//阻止冒泡  
event.stopPropagation();
```

Event Delegation

事件委派是指：只在「父元素」上綁事件監聽器，當子元素被點擊時，透過冒泡機制由父層「代為處理」事件。

- 可以大幅減少 `addEventListener()` 的數量 
- 特別適用於動態產生的子元素
- 效能更好、更整齊

```
html  
複製編輯  
<ul id="list">  
  <li>Apple</li>  
  <li>Banana</li>  
  <li>Cherry</li>
```

```
</ul>
```

```
const list = document.getElementById("list");

list.addEventListener("click", (event) => {
  if (event.target.tagName === "LI") {
    console.log("你點了：" + event.target.textContent);
  }
});
```

✓ 雖然我們沒幫每個 `` 綁事件，但冒泡讓 `ul` 接收到了點擊事件！

✓ 常見 DOM 操作屬性的方法

`setAttribute(name, value)`

◆ 功能：新增或更新一個屬性

◆ 語法：

```
element.setAttribute("屬性名", "值");
```

◆ 範例：

```
let link = document.querySelector("a");
link.setAttribute("href", "https://example.com");
link.setAttribute("target", "_self");
```

`getAttribute(name)`

◆ 功能：讀取元素上指定屬性的值

◆ 語法：

```
let value = element.getAttribute("屬性名");
```


◆ 範例：

```
let target = link.getAttribute("target");
console.log(target); // "_self"
```

`removeAttribute(name)`

◆ 功能：從元素上移除屬性

◆ 語法：

```
element.removeAttribute("屬性名");
```

◆ 範例：

```
link.removeAttribute("target");
```

`hasAttribute(name)`

◆ 功能：檢查元素上是否存在某屬性（回傳 `true` 或 `false`）

◆ 語法：

```
element.hasAttribute("屬性名");
```

◆ 範例：

```
if (link.hasAttribute("href")) {
  console.log("有超連結！");
}
```

實用應用情境

需求	實作方式
設定圖片的 <code>src</code>	<code>img.setAttribute("src", "pic.jpg")</code>
表單欄位加上 <code>disabled</code>	<code>input.setAttribute("disabled", "")</code>
取得使用者輸入的 <code>value</code> （非 DOM 屬性）	<code>input.getAttribute("value")</code>
移除按鈕的 <code>disabled</code> 屬性	<code>button.removeAttribute("disabled")</code>

用法： `element.classList`

每個 HTML 元素都有一個 `classList` 屬性，它是一個 **可操作的類別清單**（不是字串）
你可以透過它來 **增減 class 名稱**，控制元素的樣式。

`add()`

：新增 class

```
element.classList.add("className");
```

```
let box = document.querySelector("#box");  
box.classList.add("active");
```

✅ 如果原本就有這個 class，不會重複加上。

`remove()`

：移除 class

```
element.classList.remove("className");
```

```
box.classList.remove("active");
```

`toggle()`

：有就移除，沒有就新增（開關切換）

```
element.classList.toggle("className");
```

```
box.classList.toggle("active");
```

💡 ⚠️ 特別適合「開關」功能，例如展開/收起、亮燈/關燈

✅ `element.style` 是什麼？

它是每個 DOM 元素都內建的屬性，用來**直接設定該元素的行內樣式**（inline style）。

```
element.style.樣式名稱 = "值";
```

1. 只能設定該元素的行內樣式（`style=""`）。
2. CSS 屬性中的 連字號（-）需改成駝峰式（`camelCase`）命名。

✓ 時間控制函式

`setTimeout()`

功能：延遲執行某段程式碼一次

```
setTimeout(callback, delayInMs);
```

- `callback`：要延遲執行的函式
- `delayInMs`：延遲時間（單位是毫秒）

```
setTimeout(() => {  
  console.log("3 秒後才看到這句話");  
}, 3000);
```

| 🕒 執行完 3 秒後，才印出訊息

`setInterval()`

功能：固定間隔重複執行程式碼

```
setInterval(callback, intervalInMs);
```

- `intervalInMs`：每隔幾毫秒執行一次

```
setInterval(() => {  
  console.log("每 2 秒印一次");  
}, 2000);
```

停止 `setTimeout`

```
let timeoutId = setTimeout(() => {  
  console.log("這段本來會出現");  
}, 3000);
```

`clearTimeout(timeoutId);` // 馬上取消，不會出現

停止 `setInterval`

```
let intervalId = setInterval(() => {  
  console.log("不斷重複中...");  
}, 1000);
```

// 5 秒後停止

```
setTimeout(() => {  
  clearInterval(intervalId);  
}, 5000);
```