



JavaScript

講師：劉國良





Scripting Language

手稿語言(英語:Scripting language)是為了縮短傳統的「編寫、編譯、連結、執行」(edit-compile-link-run)過程而建立的電腦編程語言。早期的手稿語言經常被稱為批次處理語言或作業控制語言。一個指令碼通常是直譯執行而非編譯。指令碼語言通常都有簡單、易學、易用的特性,目的就是希望能讓程式設計師快速完成程式的編寫工作。

一些通用動態語言,比如 Perl,從一門手稿語言發展成了更通用的程式語言,由於「直譯執行,記憶體管理,動態」等特性,它們仍被歸類為手稿語言。

JavaScript 直到現在仍然是網頁瀏覽器內的主要程式語言,它的 ECMAScript 標準化保證了它成為流行的通用嵌入性語言。

[\[維基百科\]](#)





JavaScript 誕生由來

1994 年, Mosaic 的主要開發人員創立了 Netscape 公司, 該公司的目標是取代 NCSA Mosaic 成為世界第一的網頁瀏覽器。在四個月內, 已經佔據了四分之三的瀏覽器市場, 並成為 1990 年代網際網路的主要瀏覽器。

網景預見到網路需要變得更動態。公司的創始人馬克·安德森認為 HTML 需要一種語言, 讓網頁設計師和兼職程式設計師可以很容易地使用它來組裝圖片和外掛程式之類的元件, 且程式碼可以直接編寫在網頁標記中。

1995 年網景跟昇陽合作, 計劃在 Netscape Navigator 中支援 Java, 這時網景內部產生激烈的爭論, 後來網景決定發明一種與 Java 搭配使用並且語法上類似的輔助手稿語言。這個決策導致排除了採用現有的語言, 例如 Perl、Python、Tcl 和 Scheme。為了在其他競爭提案中捍衛 JavaScript 這個想法, 公司需要有一個可以運作的原型。艾克在 1995 年 5 月僅花了 十天 時間就把原型設計出來了。最初命名為 Mocha, 1995 年 9 月在 Netscape Navigator 2.0 的 Beta 版中改名為 LiveScript, 同年 12 月, Netscape Navigator 2.0 Beta 3 時被重新命名為 JavaScript。當時網景公司與昇陽電腦公司組成的開發聯盟為了讓這門語言搭上 Java 這個程式語言「熱詞」, 因此將其臨時改名為 JavaScript, 日後這成為大眾對這門語言有諸多誤解的原因之一。

[\[維基百科\]](#)





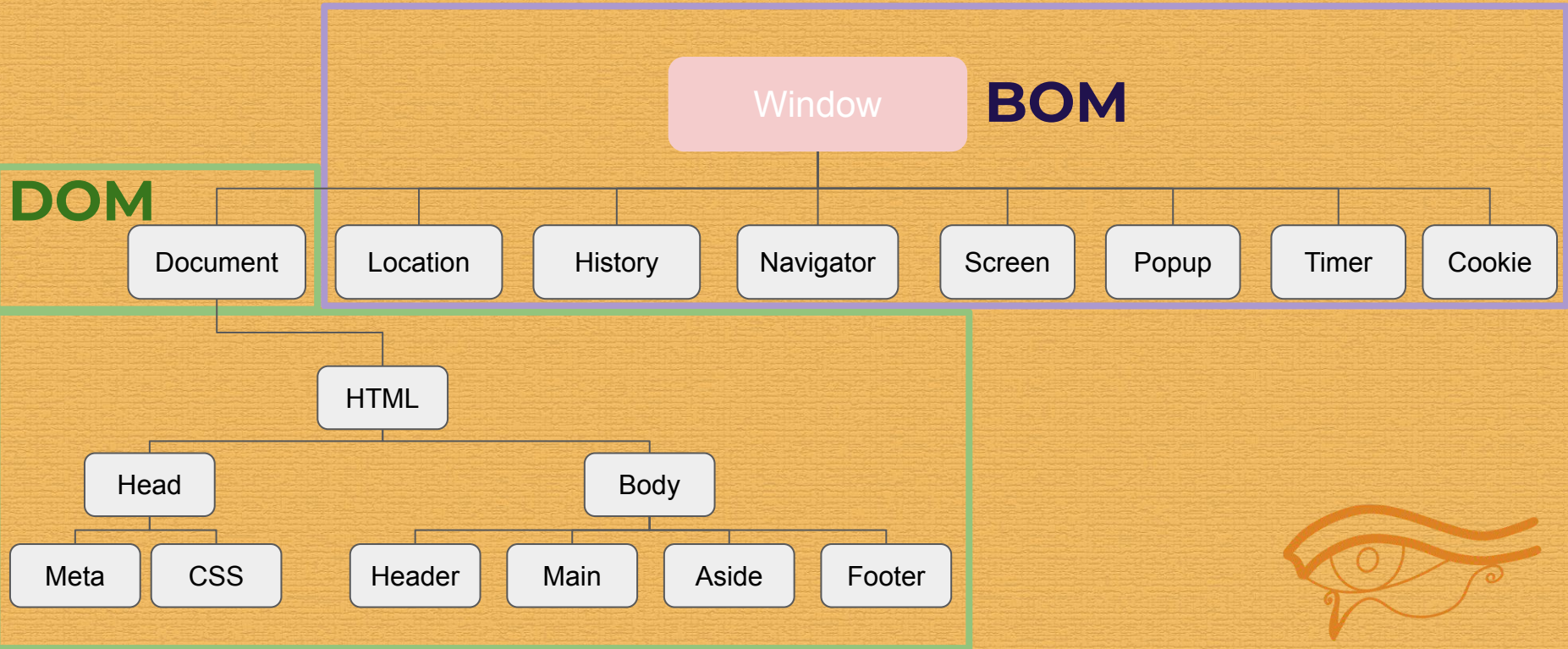
JavaScript

一般來說，完整的JavaScript包括以下幾個部分：

1. ECMAScript，描述了該語言的語法和基本物件。
2. 文件物件模型(DOM)，描述處理網頁內容的方法和介面。
3. 瀏覽物件模型(BOM)，描述與瀏覽器進行互動的方法和介面。



BOM & DOM



JavaScript 寫在哪？

```
<!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>
  <!-- HTML 寫在這裡 -->
  <h1>Hello World</h1>

  <script>
    //JavaScript 寫在這裡
    console.log('HELLO WORLD!');
  </script>
</body>
</html>
```




console 檢查工具

學習一門新語言的時候，最好能有一個翻譯來當橋樑。

任何時候都可以利用呼叫 `console.log(string, variable)` 請電腦印出結果給我們看，
未來撰寫程式時，是 debug 除錯非常好用的工具。

還沒變得很複雜之前，我們先單純簡單的測試一下 console 工具：

```
console.log('Hi');  
console.log('Hi 某某人 你好');
```





DOM 控制

DOM (Document Object Model) 是指整個網頁內容視為一個樹狀結構，從中可以取得任意的節點(node)，每個節點都包含著一個物件(object)，我們可以利用 DOM 的方法來改變網頁結構、css 樣式、甚至是網頁內容。

選取 DOM 的方法：

```
document.getElementById('html_id');  
document.getElementsByClassName('html_class');  
document.getElementsByTagName('h1');  
document.querySelector('#html_id');  
document.querySelectorAll('#html_id, .html_class, h1');
```



修改HTML標籤裡的內容

```
const h1 = document.querySelector('h1');
```

```
h1.innerHTML = 'new H1 content';
```

```
const box = document.querySelector('.box');
```

```
box.innerHTML = '<div class="insideBox">456</div>';
```


修改HTML標籤裡的CSS

```
const h1 = document.querySelector('h1');
```

```
h1.style.color = 'red';
```

```
const box = document.querySelector('.box');
```

```
box.style.backgroundColor = 'lightblue';
```

```
box.childNodes[0].style.backgroundColor = 'lightpink';
```


宣告變數 Variable

當我們在寫程式的過程中，需要一個儲存資料的容器時，我們會宣告一個變數，跟程式說「請幫我把這個變數的內容儲存起來」，未來有需要時只要呼叫變數名字，就可以取得當時儲存的內容。

ES 5

```
var myName = 'Eason';  
console.log('Hello', myName);
```

ES 6

```
let yourName = 'Márquez';  
console.log('Hi', yourName, ", I'm ", myName);  
  
const noReAssignment = 10;  
console.log('Can't re-assignment variable', noReAssignment);
```




資料型別

JavaScript 屬於弱型別語言，造成其他語言的工程師開發困擾。所謂弱型別，就是指可以隨時修改變數儲存時的資料型別。那資料型別又是什麼呢？其實指的正是儲存資料有哪些格式：

ECMAScript 2021 標準定義了 8 種資料型別：

1. String
2. Number
3. Boolean
4. Null
5. Undefined
6. BigInt (ES2020 當數值會超過 2^{53} 時使用 BigInt)
7. Symbol (ES6 新定義的型別)
8. Object 型別

其中前 7 種稱為原始型別 (primitive type)，
與第 8 種物件型別儲存方式設計不一樣。





String 字串

1. 用來儲存文字型別的資料。使用雙引號"..."或單引號('...') 包住字串前後代表該文字內容範圍。
2. 字串相加, 可以使用 + 運算符來合併兩個字串的內容。
3. ES6 新增一個使用反引號(`...`)來包住字串的用法, 更方便合併字串、帶入變數。
4. 我們可以使用 typeof 來判斷資料型別

```
const myName = 'Eason';  
console.log('myName:', myName);  
console.log('typeof:', typeof myName);
```

```
const myGreet = "Hello";  
console.log(myGreet + myName);
```

```
const newStringUsingBacktick = `${myGreet}, I'm ${myName}.`; //ES6 之後可使用反引號  
console.log('newStringUsingBacktick:', newStringUsingBacktick);
```



Number 數字

Number.MAX_VALUE : 可表示的最大正數。 $(2^{53} = 2 ** 53)$

Number.MAX_SAFE_INTEGER : JavaScript 中 IEEE-754 雙精度範圍間的最大整數 $(2^{53} - 1)$ 。

Number.MIN_VALUE: 可表示的最小、最接近 0 的數字 $(= 5e-324 = 5 \times 10^{-324})$ 。

```
const maxValue = Number.MAX_VALUE;  
console.log('maxValue:', maxValue);
```

```
const maxSafeInteger = Number.MAX_SAFE_INTEGER;  
console.log('maxSafeInteger:', maxSafeInteger);
```

```
const minValue = Number.MIN_VALUE;  
console.log('minValue:', minValue);
```



Boolean 布林

布林值: True or False, 通常用在「判斷條件」是否要執行某行程式。

例如: 如果下雨, 我就叫外送, 不然我就出門去吃早餐。

```
const isRaining = false;

if (isRaining) {
  console.log('call Uber Eats.');
```



```
}
else{
  console.log('go out for breakfast.');
```



```
}
```



Null 空值 (發音/nəl/)

通常用在建立變數的時候, 我還不確定該資料型別使用。

ps:通常最大的疑問是把它跟「數字 0」, 以及「未定義變數(undefined)」搞混。

```
let myData = null;
if (myData) {
  console.log('myData1:', myData);
}
else{
  console.log('I got no data.');
```



```
myData = 'OK, I got some data from server now.';
if (myData) {
  console.log('myData2:', myData);
}
```

undefined 未定義

從頭到尾根本沒有宣告該變數名稱。

```
const myName = 'Eason';  
console.log('myName:', myName);  
  
console.log('yourName:', yourName);
```


非 0 值 / 0 / null / undefined

Non-zero value



null



0



undefined



BigInt 大整數

用於表示大於 2^{53} 的整數的功能。(ES2020 新增)

BigInt 是透過在一個數值後加上 n ，例如 $10n$ ，或呼叫 `BigInt()` 所生成的。

警告：Number 和 BigInt 不能混和計算，他們必須被轉換到同一個型別。

```
const myBigInt = 9007199254740991n;  
console.log('myBigInt:', myBigInt);  
  
const myBigInt2 = BigInt(9007199254740991);  
console.log('myBigInt2:', myBigInt2);  
  
const myNumber = 100;  
console.log('myBigInt + myNumber:', myBigInt + myNumber);
```


Object 物件 - 取值的方法

當資料複雜的時候，單純資料型別已經無法完整存放所有內容，我們會用 Object 這資料型別來儲存大筆資料，並且在需要的時候又可以方便呼叫資料內容來運用。

Object 資料儲存方式是 **key : value** 成雙成對放在**大括號**裡面。當我需要呼叫資料時只要有 key 即可得到與他配對的 value 值。

```
const myObj = { 'key1' : 'value1', 'key2':'value2', 'key3':'value3' };
```

//如果我想取得 value1, 可以呼叫 myObj 變數, 並且使用中括號裡面放 key (名稱記得加引號)

```
console.log('value1:', myObj['key1'] );
```

//另一種取得 value1 的方法, 可以直接用「點表示法」來呼叫 Object 裡的 key

```
console.log('value1:', myObj.key1 );
```

Object 物件 - 中括號表示法

使用中括號比較麻煩，為什麼需要這方法呢？

有時候我們需要使用「變數」當做 key 名稱，在撰寫程式當時還不知道叫什麼，，這時就只能使用中括號表示法才能使用變數當 key 名稱來呼叫取得 value。

```
const myObj = { 'key1' : 'value1', 'key2': 'value2', 'key3': 'value3' };
```

//如果我想取得對應 value，但我還不知道 key 的名稱，他可能是從伺服器來的某一筆字串資料。

```
const keyFromServer = 'key1';
```

//取得 key 之後，再去問 myObj 有沒有對應的 value，如果有就印出來

//注意：這邊使用的是變數，而非剛剛的字串，所以中括號裡面沒有引號

```
console.log('value1:', myObj[keyFromServer] );
```




Object 物件新增

如果今天想新增一個 key : value ...

```
const myObj = { 'key1': 'value1', 'key2': 'value2', 'key3': 'value3' };
```

//如果我想修改 value1, 我可以在左邊直接呼叫它, 並且等於右邊賦予新值即可。

```
myObj.key4 = 'value4';
```

```
console.log('myObj:', myObj );
```





Object 物件修改

如果今天想修改其中一個 key ...

```
const myObj = { 'key1': 'value1', 'key2': 'value2', 'key3': 'value3' };
```

//如果我想修改 value1, 我可以在左邊直接呼叫它, 並且等於右邊賦予新值即可。

```
myObj.key1 = 'New value1';
```

```
console.log('myObj:', myObj );
```





Object 物件刪除

如果今天想刪除其中一個 value ...

```
const myObj = { 'key1': 'value1', 'key2': 'value2', 'key3': 'value3' };
```

//如果我想刪除 key1, 我可以使用 delete 指令即可。

```
delete myObj.key1;
```

```
console.log('myObj:', myObj );
```



Object 物件資料

如果今天想列出所有的 key 或 value ...

`Object.keys()` ES5

`Object.values()` ES2017(ES8)

`Object.entries()` ES2017(ES8)

```
const myObj = { 'key1' : 'value1', 'key2':'value2', 'key3':'value3' };
```

//如果我想拿到所有的 keys values, 可以使用物件原型 Object 裡的 keys, values 方法來取得。

```
console.log('myObj:', Object.keys(myObj));
```

```
console.log('myObj:', Object.values(myObj));
```

//物件原型裡還有另一個方法更厲害, 可以一次同時取得 keys & values。

```
console.log('myObj:', Object.entries(myObj));
```

//如果我想知道 Object 裡的資料有幾筆, 可以再加上 .length 取得長度。

```
console.log('myObj:', Object.entries(myObj).length);
```


綜合練習

如果今天想來一點...麥當勞？

```
<div id="mainMeal">
  <button>1號餐</button>
  <button>2號餐</button>
  <button>3號餐</button>
</div>

<div id="sideMeal">
  <button>A</button>
  <button>B</button>
  <button>C</button>
</div>

<div id="result">
  <p>主餐你點的是 :<span id="mainMealResult"></span> </p>
  <p>副餐你點的是 :<span id="sideMealResult"></span></p>
  <p>總價格 :<span id="totalPrice"></span></p>
</div>
```

綜合練習

先準備資料的部份

```
<script>

    const mainMealMenu = { '1號餐': { 'name': '大麥克', 'price': 72 }, '2號餐': { 'name': '雙層牛肉吉事堡',
    'price': 62 }, '3號餐': { 'name': '嫩煎雞腿堡', 'price': 82 } };

    const sideMealMenu = { 'A': { 'name': '中薯+飲料', 'price': 55 }, 'B': { 'name': '冰旋風+飲料', 'price':
    85 }, 'C': { 'name': '麥克雞塊+薯條+飲料', 'price': 100 } };

    const mainMeal = document.querySelector('#mainMeal');
    const sideMeal = document.querySelector('#sideMeal');
    const mainMealResult = document.querySelector('#mainMealResult');
    const sideMealResult = document.querySelector('#sideMealResult');
    const totalPrice = document.querySelector('#totalPrice');

    let mainMealPrice = 0;
    let sideMealPrice = 0;

</script>
```


綜合練習

主餐的功能邏輯

```
mainMeal.addEventListener('click', function (event) {  
  console.log('event target:', event.target.innerText);  
  const mealName = event.target.innerText;  
  console.log('data:', mainMealMenu[mealName]);  
  const mealDetail = mainMealMenu[mealName];  
  mainMealPrice = mealDetail.price;  
  
  mainMealResult.innerHTML = `${mealDetail.name}, 價格: ${mealDetail.price}`;  
  totalPrice.innerHTML = mainMealPrice + sideMealPrice;  
})
```



綜合練習

副餐的功能邏輯

```
sideMeal.addEventListener('click', function (event) {  
  console.log('event target:', event.target.innerText);  
  const mealName = event.target.innerText;  
  console.log('data:', sideMealMenu[mealName]);  
  const mealDetail = sideMealMenu[mealName];  
  sideMealPrice = mealDetail.price;  
  
  sideMealResult.innerHTML = `${mealDetail.name}, 價格: ${mealDetail.price}`;  
  totalPrice.innerHTML = mainMealPrice + sideMealPrice;  
})
```





Array 陣列

Array 陣列，是使用**中括號**、並按**順序**來存放或讀取資料，類似列表(list) 的物件(Object)，它們的原型(Prototype) 擁有方法(methods) 來執行遍歷和或修改。

```
const myArray = [ 'My value1', 'My value2', 'My value3'];
```

//如果我想取得對應 value1, 我只需要用中括號+索引值(index), index 是從 0 開始計算。

```
console.log('value1:', myArray[0] );
```

```
console.log('value2:', myArray[1] );
```

```
console.log('value3:', myArray[2] );
```

//如果我想取得陣列長度, 就是使用 .length 方法。

```
console.log('陣列長度:', myArray.length);
```



Array 新增

如果今天想新增一個 value ...

```
const myArray = [ 'My value1', 'My value2', 'My value3'];
```

//如果我想新增 value4, 我可以使用陣列的 push 方法, 把值加進去。

```
myArray.push('value4');  
console.log('myArray:', myArray );
```

//當然也可以直接用中括號+索引值的方式去新增, 但要小心 index 錯誤可能會產生 bug 問題。

```
myArray[6] = 'value5';  
console.log('myArray:', myArray);  
console.log('陣列長度:', myArray.length);
```




Array 修改

如果今天想修改其中一個 value ...

```
const myArray = [ 'My value1', 'My value2', 'My value3'];
```

//如果我想修改 value1, 我可以在左邊直接呼叫它, 並且等於右邊賦予新值即可。

```
/data
```

```
console.log('myArray:', myArray );
```



綜合練習

製作行政區查詢系統

```
<label>請選擇行政區</label>
```

```
<select id="city">
```

```
  <option value="-1" style="display: none;">請選擇</option>
```

```
  <option value="0">基隆市</option>
```

```
  <option value="1">台北市</option>
```

```
  <option value="2">新北市</option>
```

```
</select>
```

```
<div id="result"></div>
```


綜合練習

先準備好行政區的資料

```
<script>

  const cityArray = ["基隆市", "台北市", "新北市"];

  const districtArray = ["'中正區'", "'信義區'", "'仁愛區'", "'中山區'", "'安樂區'", "'暖暖區'", "'七堵區'", "'松山區'", "'信義區'", "'大安區'", "'中山區'", "'中正區'", "'大同區'", "'萬華區'", "'文山區'", "'南港區'", "'內湖區'", "'士林區'", "'北投區'", "'板橋區'", "'三重區'", "'中和區'", "'永和區'", "'新莊區'", "'新店區'", "'土城區'", "'蘆洲區'", "'汐止區'", "'樹林區'", "'淡水區'"];

  const citySelector = document.querySelector('#city');
  const result = document.querySelector('#result');

</script>
```

綜合練習

開始寫互動的功能

```
citySelector.addEventListener('change', function (event) {  
    console.log('event target:', event.target);  
    const cityValue = event.target.value;  
    const cityName = cityArray[cityValue];  
    console.log('data:', districtArray[cityValue]);  
    const districDetail = districtArray[cityValue];  
  
    result.innerHTML = `

### 你選擇的是 : ${cityName}</h3> <p>行政區資料如下 : ${districDetail}</p>`; })


```


Flask

```
import flask
from flask import jsonify
from flask_cors import CORS

app = flask.Flask(__name__)
app.config["JSON_AS_ASCII"] = False
app.config["DEBUG"] = True
CORS(app, resources={r"/*": {"origins": ["http://127.0.0.1:5500", "*"]}})

menuData = {
    '1號餐': { 'name': '456大麥克', 'price': 72 },
    '2號餐': { 'name': '雙層牛肉吉事堡', 'price': 62 },
    '3號餐': { 'name': '嫩煎雞腿堡', 'price': 82 },
    'A': { 'name': '中薯+飲料', 'price': 55 },
    'B': { 'name': '冰旋風+飲料', 'price': 85 },
    'C': { 'name': '麥克雞塊+薯條+飲料', 'price': 100 }
}
```

```
@app.route('/', methods=['GET'])
def home():
    return "<h1>Welecome to my flask  
server.</h1>"

@app.route('/menu', methods=['GET'])
def menu():
    return jsonify(menuData)

app.run()
```

Fetch (ajax)

```
<script>

  fetch('http://127.0.0.1:5000/menu' )

    .then(r => r.json())

    .then(data => {

      console.log('data from server', data)

      let allBtnDoms = '';

      for (let i = 0; i < Object.keys(data).length; i++) {

        allBtnDoms += `<button>${Object.keys(data)[i]}</button>`

      }

      allMenuArea.innerHTML = allBtnDoms;

    })

</script>
```




Symbol

推出 Symbol 資料型別最主要的目的就是處理資料可能會有重複的問題，當資料內容可能重複出現，但又希望每筆資料都能獨立存在的時候，就需要使用 Symbol。

每筆 Symbol 建立的值，都是獨一無二的存在。

```
const string1 = 'Eason';  
const string2 = 'Eason';  
console.log('is string1 === string2? :', string1 === string2);  
  
const symbol1 = Symbol('Eason');  
const symbol2 = Symbol('Eason');  
console.log('is symbol1 === symbol2? :', symbol1 === symbol2);
```





運算子

時間的關係，這邊僅列出一些常用的運算子講解其概念

1. 算術運算子
2. 賦值運算子
3. 比較運算子
4. 邏輯運算子
5. 條件(三元)運算子



算數運算子

名稱	寫法	範例
加、減、乘、除	10 + 2, 10 - 2, 10 * 2, 10 / 2	((2+ 3) * 10) - 5
餘數	10 % 3	
指數	10 ** 2	
增加1	x++	let x = 10; console.log('x++:', x++);
減少1	x--	let x = 10; console.log('x--:', x--);
利用運算子轉型	+'10'	let x = '10'; console.log('+'x:', +x);



綜合練習

製作計算機 CSS

<style>

```
.calculator {  
  width: 400px;  
  padding: 10px;  
  background-color: #ccc;  
}  
  
.display {  
  min-height: 100px;  
  background-color: #eee;  
  margin: 10px;  
  font-size: 2rem;  
  word-break: break-all;  
}
```

</style>

```
.btns {  
  display: flex;  
  flex-wrap: wrap;  
}  
  
.btn {  
  width: 80px;  
  height: 80px;  
  margin: 10px;  
  background-color: #fff;  
  display: flex;  
  justify-content: center;  
  align-items: center;  
}  
  
.btn0 {  
  width: 180px;  
}
```


綜合練習

製作計算機 HTML

```
<div class="calculator">
  <div class="display"></div>
  <div class="btns">
    <div class="btn">1</div>
    <div class="btn">2</div>
    <div class="btn">3</div>
    <div class="btn">+</div>
    <div class="btn">4</div>
    <div class="btn">5</div>
    <div class="btn">6</div>
    <div class="btn">-</div>
    <div class="btn">7</div>
    <div class="btn">8</div>
    <div class="btn">9</div>
    <div class="btn">*</div>
    <div class="btn btn0">0</div>
    <div class="btn">=</div>
    <div class="btn">/</div>
  </div>
</div>
```

綜合練習

製作計算機 JS

```
<script>
const calc = document.querySelector('.calculator');
const display = document.querySelector('.display');

let inputAll = '';

calc.addEventListener('click', function (event) {
  const eventTarget = event.target;
  if (!event.target.classList.contains("btn")) return;

  if (event.target.innerHTML === "=") {
    try {
      if (eval(inputAll)) {
        display.innerHTML = `${eval(inputAll)}`;
        inputAll = '';
      }
    }
    catch {
      display.innerHTML = '';
      inputAll = '';
    }
  }
  else {
    inputAll += event.target.innerHTML;
    display.innerHTML = inputAll;
  }
});
</script>
```




賦值運算子

JavaScript 的等號跟數學等號意義不同，這裡的等號比較像是「變成右邊的 值」的意思。

名稱	意義	簡寫
賦值	$x = y$	$x = y$
加法	$x = x + y$	$x += y$
減法	$x = x - y$	$x -= y$
乘法	$x = x * y$	$x *= y$
除法	$x = x / y$	$x /= y$
餘數	$x = x \% y$	$x \% = y$
指數	$x = x ** y$	$x ** = y$



比較運算子

得到的結果為布林值。

名稱	寫法	範例
大於	$x > y$	<code>10 > 5 //true</code>
小於	$x < y$	<code>10 < 5 //false</code>
小於等於	$x \leq y$	<code>10 \leq 5 //false</code>
等於	$x == y$	<code>'10' == 10 //true</code>
嚴格等於	$x === y$	<code>'10' === 10 //false</code>
不等於	$x != y$	<code>'10' != 10 //false</code>
嚴格不等於	$x !== y$	<code>'10' !== 10 //true</code>



邏輯運算子

得到的結果通常是布林 值，但特殊情況下我們也會用這方法得到其他類別的資料 ...

名稱	寫法
AND 而且	true && false && true
OR 或者	true false true
NOT 反轉	!true, !false
利用NOT轉型	!!0, !!null, !!", !!undefined, !!NaN

當我們後期很熟悉邏輯運算子之後，常常會把它運用在程式語法之中。我們稱之為短路解析。利用 AND 可以快速判斷資料是否存在，利用 OR 可以設定預設值。



三元運算子 (判斷) ? if true... : if false...

一般來說，寫程式常常會遇到一些需要判斷的地方，如果為 true，才繼續往下寫，我們會用 if 來作為判斷：

```
const x = 10;  
if ( x > 10) {  
    console.log('x > 10');  
}  
else  
{  
    console.log('x <= 10');  
}
```

但上面寫起來會變得很長，這時候，我們可以用三元運算子來做簡化：

```
const x = 10;  
console.log( (x > 10) ? 'x > 10' : 'x <= 10' );
```


函式 function

函式可以說是 javascript 組成的基本要素之一，一個函式本身就是一段功能，可用來重複執行某個任務或計算的語法。目前前端框架也都是使用 Functional Programming 設計。

定義函式，是用一系列的函式關鍵詞以及符號組成，例如：

- 1.開頭用 `function` 當做宣告。
- 2.後面帶著一個函式名稱：`square` (函式名稱可以自定)。
- 3.小括號裡面是呼叫 `square` 函式的時候可以帶入的參數 (參數名稱可自定)。
- 4.大括號裡面代表你想執行的任務、或計算。
- 5.最後，結束函式後想回傳什麼結果，可以寫在 `return` 後面。

```
function square(number) {  
  return number * number;  
}
```



函式表達式 function expression

現在還有一個新的函式宣告方式，叫做函式表達式 (function expression)
簡單來說，就是變成宣告一個變數，讓該變數等於一個函式。
以後呼叫這個變數，就等同於是呼叫函式。

```
const square = function (number) {  
  return number * number;  
}
```

```
const x = square(2); // x = 4  
const y = square(3); // y = 9
```





變數作用域 Scope

隨著程式功能複雜度上升，變數間相互影響的機率也上升。

因此，一位好的工程師必須盡量將變數控制在某個範圍內，避免影響到其他人的程式。

可簡單區分為：全域變數、以及區域變數。

簡而言之就是在哪些範圍內可以呼叫到這個變數、或修改它。

```
<script>
var x = 10;
let y = 20;

function add (){
  return x + y;
}

console.log (add());
</script>
```




變數作用域 Scope

通常我們會比較希望使用被限制在某個區域範圍的變數，避免影響過大，所以都會利用呼叫函式的時候，順便帶入當時的參數進到函式 內使用，如此一來，這個變數就只會在函式 內作用，並不會影響外部的變數，我們稱之為區域變數。

```
<script>
var x = 10;
let y = 20;

function add ( x, y){
  return x + y;
}

console.log ( add (3, 4) );
</script>
```



變數作用域 Scope

宣告變數時的位置也很重要，寫在函式裡面就是區域變數。

```
<script>
var x = 10;
let y = 20;

function add ( x, y){
  let x = 5;
  let y = 2;
  return x + y;
}

console.log ( add () );
</script>
```

變數作用域 Scope

總結: 如果自己範圍內有該變數, 會先以自己的區域變數為主。

如果自己這層沒有, 則會再往上一層尋找, 直到找到、或完全找不到 (undefined) 為止。

```
let x = 10;  
  
function level1 () {  
  let x = 5;  
  function level2 () {  
    let x = 1;  
    console.log('x:', x);  
  }  
  level2();  
}  
  
level1();
```


閉包 closure

閉包是函式以及該函式被宣告時所在的作用域環境 (lexical environment) 的組合。

```
function myClosure (x, funcName){  
  
  return function insideClosureFunc (){  
    x = x - 1;  
    console.log(funcName, ' now x:', x);  
  }  
}  
  
const newFunc1 = myClosure(10, 'newFunc1');  
newFunc1();  
  
const newFunc2 = myClosure(100, 'newFunc2');  
newFunc2();
```

事件綁定 addEventListener

當我們在寫程式的過程中，有時候會需要 **與使用者互動** 的功能，譬如：針對點擊按鈕後做出反應。希望使用者點擊之後，執行某段程式碼的話，我們會這樣寫：

```
const loginBtn = document.querySelector('.loginBtn');
```

```
loginBtn.addEventListener( 'click', function(){
```

```
    //要執行的程式碼寫 function 裡面
```

```
    alert('login btn clicked');
```

```
});
```

呼叫 JavaScript 內建的 alert 函式
可以跳出一個視窗提醒使用者注意

預習：函式的基本架構

```
function () {  
    //要執行什麼功能寫裡面  
}
```

寫函式的用意是未來需要一直做重複的事情時，只需要再呼叫這個函式，不用重複寫好幾次程式碼。

綜合練習

使用者點擊按鈕之後，H1 文字變紅色、box 變成寬 500px ,高 500px。

```
<button class="changeBtn">Change Btn</button>
<h1>Hello World</h1>
<div class="box" style="width:100px;height:100px;background:lightblue;"></div>
```

```
const changeBtn = document.querySelector('.changeBtn');
const h1 = document.querySelector('h1');
const box = document.querySelector('.box');

changeBtn.addEventListener( 'click', function(){
    //要執行的程式碼寫 function 裡面
    //我希望1: H1 文字變紅色
    //我希望2:box 變成寬 500px, 高 500px
});
```

字串 常用方法 (Method)

寫法	說明
charAt(索引值)	取得某位置的字元
charCodeAt(索引值)	取得某位置字元的字碼
concat(字串)	字串串接, 通常會使用 + 運算子更簡單
indexOf(字串, [索引值])	尋找該字串出現的位置, 找不到回傳 -1
lastIndexOf(字串, [索引值])	從後面開始尋找字串出現的位置
slice(索引值1, 索引值2)	從索引值1 到 索引值2 建立一個新字串回傳
split(分割符號)	依照某特定字, 分割字串, 產生一個陣列
substr(索引值, 字元數)	從索引值的位置, 擷取字元數長度
substring(索引值1, 索引值2)	類似slice, 但索引值1可以大於索引值2。
toLowerCase()	轉小寫
toUpperCase()	轉大寫

陣列 常用方法 (Method)

寫法	說明
concat(陣列)	與某陣列串接，建立一個新的陣列
indexOf(字串, [索引值]),lastIndexOf(字串, [索引值])	取得元素值的位置，找不到則回傳 -1
join([連接符號])	將陣列裡的元素用連結符號串成一個字串
pop()	彈出，從尾端取出一個元素
push()	推入，從尾端新增一個元素
shift()	左移，從前端取出一個元素
unshift()	右移，從前端加入一個元素
sort()	排序
reverse()	反轉排序
slice(索引值1, 索引值2)	從索引值1到索引值2(不包含)複製陣列
splice(索引值, 刪除數量 [, 加入])	可插入、刪除、或替換陣列 內的元素

迴圈

自定義迴圈規則，包含三個條件，使用分號隔開：

- 1.宣告一個索引值，可指定從多少開始。
- 2.會重複判斷是否繼續執行的條件。
- 3.每次結束後執行的變化。

注意：必須讓這三個條件組合成一個會結束的情況，否則這個迴圈會一直無窮無盡執行下去。

//自定參數迴圈

```
for (let i = 0 ; i < 5 ; i ++ ) {  
  console.log('i:', i );  
}
```

//使用陣列資料來跑迴圈

```
const myArray = ['a', 'b', 'c', 'd'];  
for (let i = 0 ; i < myArray.length ; i++){  
  console.log('myArray[' + i + ']', myArray[i] );  
}
```


陣列方法迴圈

javascript 針對陣列，有內建許多跑迴圈的方法可以使用。

```
const myArray = ['a', 'b', 'c', 'd'];  
//使用陣列內建方法 forEach 來跑迴圈, 不會回傳新陣列  
myArray.forEach (function(element, index){  
    console.log('index', index);  
    console.log('element', element);  
})  
//使用陣列內建方法 map 來跑迴圈, 會回傳一個新的陣列  
const myNewArray = myArray.map (function(element, index){  
    return index + element;  
})
```

陣列方法迴圈

javascript 針對陣列，有內建許多跑迴圈的方法可以使用。

```
const myArray = [1, 2, 3, 4];  
//使用陣列內建方法 filter 來跑迴圈，會回傳依照某條件篩選後的新陣列  
const myNewArrayByFilter = myArray.filter (function(element, index){  
    return element > 2;  
})  
  
//使用陣列內建方法 reduce 來跑迴圈，回傳一個全部累加後的資料  
const myNewArrayByReduce = myArray.reduce((accumulator, currentValue, index, originalArray) => {  
    accumulator += currentValue;  
    return accumulator;  
}, initialValue)
```

ps: 如果沒有累加前的初始值，initialValue 可以為 0。

Object 迴圈

可以使用之前說的 `Object.entries(myObj)`，列出所有資料的格式是陣列，利用陣列資料跑回圈。

```
const userInfoObj = {name : 'Fabio', age: '23', birth: ' 20 April'};
//我們先檢查 data 陣列內容
const datasArray = Object.entries(userInfoObj);
console.log('datasArray:', datasArray)
//確定是陣列之後，後面就可以接續使用陣列的方法跑回圈，得到所有 key value 的資料了
Object.entries(userInfoObj).forEach (function(element, index){
  console.log('element', element);
})
//陣列裡如果資料還是陣列，我們會稱之為二維陣列，可以再跑一次回圈
Object.entries(userInfoObj).forEach (function(element, index){
  element.forEach( function (item, index){
    console.log('item', item);
  })
})
```



Date time 日期時間

我們可以透過 JavaScript 取得目前系統的時間。

```
const now = new Date();
console.log("now", now);

// 時間格式轉成 ISO8601 標準時間格式
console.log("now", now.toISOString());

// 字串格式轉 ISO8601 標準時間格式
const today_good = new Date("2022-03-23");
console.log("today", today_good.toISOString());

// 錯誤的字串格式轉 ISO8601 標準時間格式
const today_bad = new Date("2022/03/23");
console.log("today2", today_bad.toISOString());
```



Date time 日期時間

我們可以透過 JavaScript 取得目前系統的時間。

```
// 或者是使用帶著時間的標準時間格式的字串比較安全
```

```
const dayTime = new Date("2022-03-23T19:29:07.597Z");
```

```
console.log("dayTime", dayTime.toISOString());
```

```
// 如果不轉 ISO8601 的話, 會是 local 當地時間(GMT+8)
```

```
const dayTime2 = new Date("2020-02-02T13:00:00");
```

```
console.log("dayTime2", dayTime2);
```

```
// new Date().toLocaleString('zh-TW', {timeZone: 'Asia/Taipei'});
```

```
// new Date().toLocaleString('en-US', {timeZone: 'America/New_York'});
```

```
// 或是指定時區的轉換 ISO8601 標準時間格式
```

```
console.log(
```

```
  "now",
```

```
  now.toLocaleString("en-US", { timeZone: "Asia/Taipei" })
```

```
);
```

Date time 日期時間

我們可以透過 JavaScript 取得目前系統的時間。

```
// 取得本地時間與標準時區的時間差
```

```
console.log('diff time:', new Date().getTimezoneOffset() / 60);
```

```
// 取得各別時間參數
```

```
let d = new Date();
```

```
console.log('new Date, d:', 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.getMilliseconds());
```

```
console.log(d.getTime());
```




Date time 日期時間

如果是經驗不足的新手，也可以考慮使用前輩製作的套件，站在巨人的肩膀上前進。

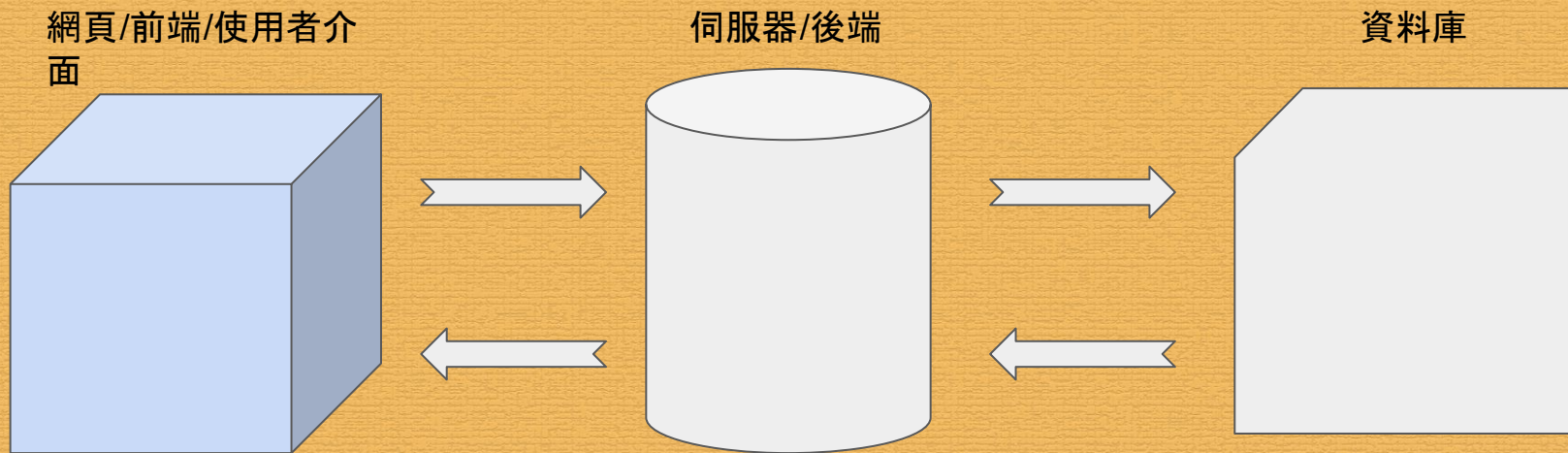


DAY.JS



Ajax

前後端傳遞資料時使用的技術，依據不同的傳輸協定會設定不同的參數，取得伺服器的資料。



Ajax

原生 JavaScript 內建最新的 ajax 技術為 Fetch。

傳送請求最基本的方法有：GET, POST

其他複雜一點還有：PUT, DELETE, CONNECT, TRACE, PATCH, HEAD, OPTIONS 等。

```
//使用 GET 的方式，通常是用來快速簡易的跟伺服器取得資料用，沒有特別指定方法的話就是使用GET Method。
```

```
fetch('https://jsonplaceholder.typicode.com/posts/1')  
  .then((response) => response.json())  
  .then((json) => console.log(json));
```

```
fetch('https://jsonplaceholder.typicode.com/posts')  
  .then((response) => response.json())  
  .then((json) => console.log(json));
```

Ajax

POST 通常用來傳送較為隱私的資訊，例如使用者註冊、登入的帳號密碼。

```
fetch('https://jsonplaceholder.typicode.com/posts', {  
  method: 'POST',  
  body: JSON.stringify({  
    userName: 'ChaosMint',  
    userPassworld: '12345678',  
  }),  
  headers: {  
    'Content-type': 'application/json; charset=UTF-8',  
  },  
})  
  .then((response) => response.json())  
  .then((json) => console.log(json));
```


Ajax

PUT 通常用來修改更新資訊。

```
fetch('https://jsonplaceholder.typicode.com/posts/1', {  
  method: 'PUT',  
  body: JSON.stringify({  
    userId: '101',  
    userName: 'ChaosMango',  
    userPassworld: 'gas!!#fasdfga 31',  
  }),  
  headers: {  
    'Content-type': 'application/json; charset=UTF-8',  
  },  
})  
  
  .then((response) => response.json())  
  .then((json) => console.log(json));
```



Ajax

DELETE 通常用來刪除資訊，使用時一定要確定，如果資料庫沒有備份，刪除就回不來了。

```
fetch('https://jsonplaceholder.typicode.com/posts/1', {  
  method: 'DELETE',  
});
```

