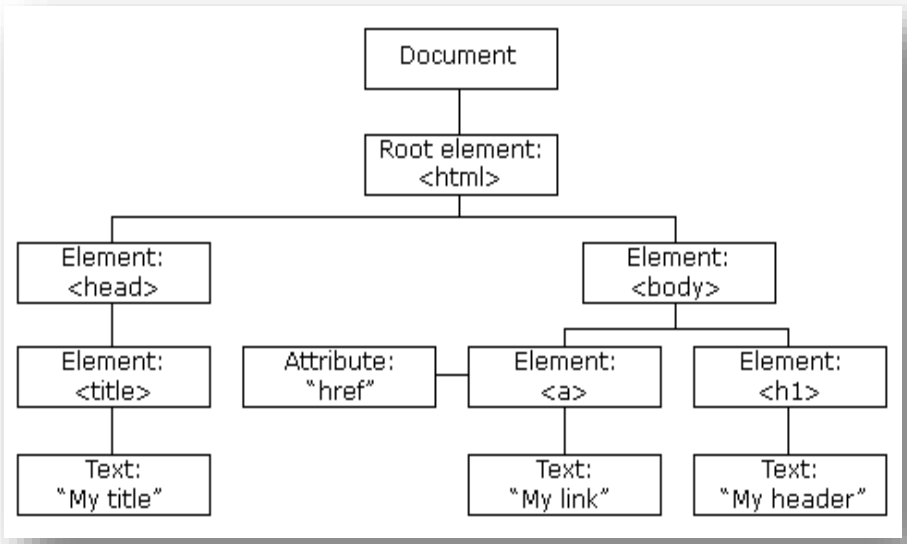
目 錄

0 JavaScript vs DOM

- DOM(Document Object Model/文件物件模型)是一個表達、存取網頁中物件的介面(API/Application Programming Interface)。
- DOM 是全球資訊網協會(World Wide Web Consortium，W3C)所制定的一套標準，大部分的瀏覽器開發廠商都會遵循並採用這套標準。
- 有了 DOM， JavaScript 就可以和網頁上的 HTML 進行互動，並修改 HTML 內容。
- DOM 將一個 HTML 網頁的資料結構以樹狀(tree)圖來表達，樹狀圖中的每一塊狀項目稱為節點(nodes)。
- HTML DOM 樹狀圖範例




```
<html>
  <head>
      <title>My title</title>
  </head>
  <body>
      <a href="t1.html">My link</a>
     <h1>My header</h1>
  </body>
</html>
```

0.1 入門範例(T00-01.html)

T00-01.html


```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>事件處理</title>
    <style>
        .main-box {
            margin: 50px auto;
            width: 400px;
            height: 200px;
            padding: 50px;
            box-shadow: 5px 5px 10px #999;
            border: 1px solid #fff;
            text-align: center;
            font-size: 1.3em;
        }
    </style>
</head>
<body>
    <div class="main-box">
        <div id="div3">
            <h3>簡易四則計算器</h3>
            <input type="text" size="10" id="number1" value="20" />
            <input type="text" id="op" value="+" style="width:20px; text-align:center;" />
            <input type="text" size="10" id="number2" value="10" />
            <button onclick="calculate()">=</button>
            <input type="text" size="10" id="result" />
        </div>
        <script>
            function calculate() {
                var s1 = document.getElementById('number1').value;
                var s2 = document.getElementById('number2').value;
```

```
                var n1 = parseInt(s1);
                var n2 = parseInt(s2);
                var op = document.getElementById('op').value;
                var n3;
                switch (op) {
                    case '+':
                        n3 = n1 + n2;
                        break;
                    case '-':
                        n3 = n1 - n2;
                        break;
                    case '*':
                    case 'x':
                        n3 = n1 * n2;
                        break;
                    case '/':
                        n3 = n1 / n2;
                        break;
                    default:
                        n3 = '運算符號無效!';
                }
                document.getElementById('result').value = n3;
            }
        </script>
    </div>
</body>
</html>
```

1 JavaScript 基本語法

1.1 識別字的規則

- 識別字(Identifier)是一個程式碼中的字元序列(a sequence of characters)，用以定義變數(variable)、函式(function) 或屬性(property)。
- 在 JavaScript 中，識別字只能包含 Unicode 的字母、數字、"_"、"$"，區分大小寫但不能以數字開頭。
- 識別字不同於字串，字串是資料(data)而識別字是程式碼(code)的一部分 。

1.2 變數與常數宣告

1.2.1 變數宣告

- 變數乃用以暫存資料的電腦記憶體中的一塊空間。
- JavaScript 的所有變數皆須宣告，變數宣告方式主要有二：
  - ✓ var var1[= value1] [, var2 [= value2]] [, ..., varN [= valueN]];
  - ✓ let var1 [= value1] [, var2 [= value2]] [, ..., varN [= valueN]];
    - ✧ var1, var2, …, varN：變數名稱。
    - ✧ value1, value2, …, valueN：變數的初始值，可以是任何合法的運算式(expression)。
  - ✓ 以 var 和 let 宣告變數的主要差異
    - ✧ 重複宣告
      - ◆ let 變數在同一個區塊內不能重複宣告。
      - ◆ var 變數在同一個區塊內則可重複宣告。
    - ✧ 可視範圍 (scope)
      - ◆ let 宣告的變數具有區塊可視範圍。
      - ◆ var 宣告的變數不具有區塊可視範圍，但可分成：
        - ● 全域變數(global variable)
          A JavaScript global variable is declared outside the function or declared with window object. (var 宣告的全域變數會變成 window 的屬性)。
          It can be accessed from any function。
        - ● 區域變數(local variable)
          Local variables are variables that are defined within functions.
          They have local scope, which means that they can only be used within the functions that define them.
  - ✓ 使用範例：

```
<script>
    {
        var n1 = 100; //全域變數
        let n2 = 100; //區塊變數
        document.write(`n1=${n1} n2=${n2}<br/>`);//print: n1=100 n2=100
        //註:`${}`: 樣板字面值(Template literals)是允許嵌入運算式的字串字面值(string literals）
    }
    document.write(`n1=${n1}<br/>`);//print: n1=100
    //document.write(`n2=${n2}<br/>`); //Uncaught ReferenceError: n2 is not defined
    function f1() {
        var n3 = 100; //區域變數
        let n4 = 100; //區塊變數
        document.write(`n3=${n3} n4=${n4}<br/>`); //print n3=100 n4=100
    }
    //document.write(`n3=${n3}<br/>`); //Uncaught ReferenceError: n3 is not defined
     //document.write(`n4=${n4}<br/>`); //Uncaught ReferenceError: n4 is not defined
```

```
        f1();
    </script>
```

- 範例相關 API
  - document.write(markup)
    The Document.write() method writes a string of text to a document stream.(在元素內新增內容)
    - Parameters
      - markup：A string containing the text to be written to the document.

1.2.2 常數宣告

- Constants (常數)有點像使用 let 所宣告的變數，具有區塊可視範圍。常數不能重複指定值，也不能重複宣告。
- 常數宣告方式：
  - const name1 = value1 [, name2 = value2 [, ... [, nameN = valueN]]];
    - nameN：常數的名稱。
    - valueN：常數的值，可以是任何合法的運算式(expression)。
- 使用範例：

```
const PI = 3.14159;
```

1.3 JavaScript 的資料型別

1.3.1 基本資料型別(Primitive Data Type)

- JavaScript 支援的基本資料型別
  - ✓ 字串型別(String)
    - ✧ 字串型別代表一種字元序列，通常用來表示一組文字資料
    - ✧ 字串內每個字元對應 16-bit 之正整數
    - ✧ 字串型別之值須以單引號或雙引號或反引號括起來，例如"Hello"或'Hello'或`Hello`
  - ✓ 數值型別(Number)
    - ✧ Number 型別之值只有 64 位元浮點數 (包含整數)，例如：

      var n1=10;

      var n2=10.5;
  - ✓ 布林型別(Boolean)
    - ✧ 布林型別之值只有 true 、false
  - ✓ Null 型別
    - ✧ Null 型別之值只有 null
    - ✧ null 表示不存在的物件
  - ✓ 未定義型別(Undefined)
    - ✧ 未定義型別之值只有 undefined
    - ✧ 一個被宣告而沒有初始化的變數有 undefined 值
- 使用範例：

```
<script>
    var v1 = 100;     //Number
    var v2 = true;    //Boolean
    var v3 = "Hello"; //String
</script>
```

1.3.2 Object

- JavaScript 支援的另一種資料型別：Object
- 詳細說明請參考後面 Object 相關章節。

1.4 資料型別的轉換

1.4.1 字串轉換為 Number

- parseInt(string, radix);
  - ✓ 標準內建函式之一
  - ✓ 能將輸入的字串轉成整數。
  - ✓ 參數
    - ✧ string
      待轉成數字的字串。若 string 參數類型不是字串的話，會先將其轉成字串（相當於先執行 ToString 再執行 parseInt）空白值會被忽略。
    - ✧ radix
      - ◆ 從 2 到 36，能代表該進位系統的數字。
      - ◆ 例如說指定 10 就等於指定十進位。
  - ✓ 回傳值
    - ✧ 藉由給定字串作轉換後的數字。
    - ✧ 若第一個字元無法轉換為數字，則回傳 NaN (全域屬性)。
- parseFloat(string)
  - ✓ 標準內建函式之一
  - ✓ 能將輸入的字串轉成浮點數。
  - ✓ 參數
    - ✧ string
    - ✧ 需要被解析成為浮點數的值。
  - ✓ 回傳值
    - ✧ 給定值被解析成浮點數。如果給定值不能被轉換成數值，則回傳 NaN (全域屬性)。
- 使用範例：

```
<script>
    var v1 = "123.456";
    document.write(`v1=${parseInt(v1)} v1=${parseFloat(v1)}`); // v1=123    v1=123.456
</script>
```

1.4.2 Number 轉換為字串

- obj.toString()
  - ✓ 傳回一個表示該(Number)物件的字串。
- numObj.toFixed([digits])
  - ✓ 使用定點小數表示法（fixed-point notation）來格式化數字。
  - ✓ 參數
    - ✧ digits
      - ◆ Optional.
      - ◆ The number of digits to appear after the decimal point; this may be a value between 0 and 20, inclusive, and implementations may optionally support a larger range of values.
      - ◆ If this argument is omitted, it is treated as 0.
  - ✓ 回傳值
    - ✧ 一個代表以定點小數表示法（fixed-point notation）格式化數字後的字串。
- 使用範例：

```
<script>
    var v1 = 123.456;
    document.write(`v1=${v1.toString()} v1=${v1.toFixed(2)}`); // v1=123.456    v1=123.46
</script>
```

1.5 運算子

1.5.1 算術運算子

- 算術運算子列表

var y=5;

運算子	說明	範例	結果
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6
--	Decrement	x=--y	x=4

- 運算子優先順序

運算子	說明
. [] ()	Field access Array indexing Function calls
++ -- - (負) ! typeof (回傳 data type)	Unary operators
* / %	Multiplication, division, modulo division
+ - +	Addition, subtraction, string concatenation
< ,<= >, >=	Less than, less than or equal Greater than, greater than or equal
==, != ===, !==	Equality, inequality Strict equality, strict inequality
&&	Logical AND
||	Logical OR
?:	Conditional
=	Assignment

OP=	Assignment with operation (such as += and &=)

- 使用範例：

```
<script>
    var v1 = 100;
    var v2 = 123.45;
    var v3 = v1 + 20 - v2 * 5 / 3 + "px";
    document.write(v3);    // -85.75px
</script>
```

1.5.2 關係運算子

- 關係運算子列表

var x=5

運算子	說明	範例
==	is equal to	x==5 is true x=='5' is true
===	is exactly equal to (value and type)	x===5 is true x==='5' is false
!=	is not equal	x != '5' is false
!==	is not exactly equal to	x !=='5' is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

- 使用範例：

```
<script>
    var v1 = 100;
    var v2 = 123.45;
    document.write(v1>v2<50); //true
</script>
```

1.5.3 邏輯運算子

- 關係運算子列表

x=6; y=3;

運算子	說明	範例
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

- 使用範例：

```
<script>
    var v1 = 100;
    var v2 = 123.45;
    document.write(v1 < v2 && v1++ != 100);    //false
```

```
</script>
```

1.5.4 條件運算子

```
var lang = "CHINESE"
var greeting= (lang=="CHINESE")?"歡迎":"Welcome";
//當 lang=="CHINESE"為真，則 greeting 被指定"歡迎"，否則 greeting 被指定"Welcome "
```

1.5.5 指派運算子

x=10; y=5;

運算子	說明	相當於	結果
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

1.6 流程控制

1.6.1 選擇性敘述

1.6.1.1 if 敘述 (One-Way selection) (T01-01.html 1/3-A)

```
if (condition)
{
    程式碼被執行(當條件式為 true)
}
```

- ✧ condition 為 0、''、null、undefined、NaN 時，視為 false；
- ✧ condition 為其他值時，則視為 true。

■ 使用範例：T01-01.html 1/3-A


```
<div id="p1">
    <div class="main-box">
        <h3>
            <script>
                var gender = "Male";
                if (gender == "Male") {
                    document.write("男性網友您好!");
                }
                document.write(`<hr/>`);

                (略)
            </script>
        </h3>
    </div>
</div>
```

1.6.1.2 if...else 敘述 (Two-Way Selection/二選一) (T01-01.html 1/3-B)

```
if (condition)
{
    程式碼被執行(當條件式為 true)
}
else
{
```

程式碼被執行(當條件式為 false)
}

■ 使用範例：T01-01.html 2/3


```
<div id="p1">
    <div class="main-box">
        <h3>
            <script>
                (略)
                if (gender == "Male") {
                    document.write("先生您好!");
                }
                else {
                    document.write("女士您好!");
                }
                 document.write(`<hr/>`);

                (略)
            </script>
        </h3>
    </div>
</div>
```

1.6.1.3 switch 敘述 (T01-01.html 1/3-C)

```
switch(n)
{
    case 1:
        execute code block 1
        break;
      case 2:
        execute code block 2
        break;
      default:
        code to be executed if n is different from case 1 and 2
}
```

■ 使用範例：T01-01.html 1/3-C

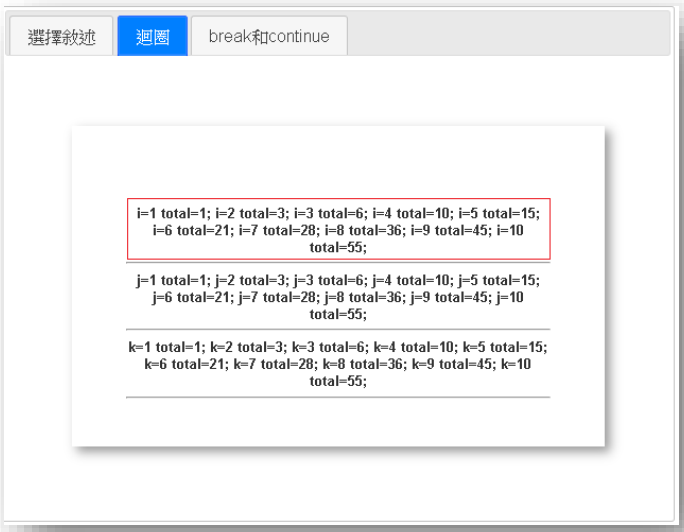

```
<div id="p1">
    <div class="main-box">
        <h3>
            <script>
                (略)
                var birthPlace="台北";
                switch (birthPlace)
                {
                    case "台南":
                        document.write("出生地:台南");
                        break;
                    case "台中":
                        document.write("出生地:台中");
                        break;
                    case "台北":
                        document.write("出生地:台北");
                        break;
                    default:
                        document.write("出生地:N/A");
                        break;
                }
            </script>
        </h3>
    </div>
</div>
```

1.6.2 重複性敘述

1.6.2.1 for 迴圈 (T01-01.html 2/3-A)

```
for ([let] v=起值; v<=終值; v=v++增值){
    程式碼被執行
}
```

- 使用範例：T01-01.html 2/3-A

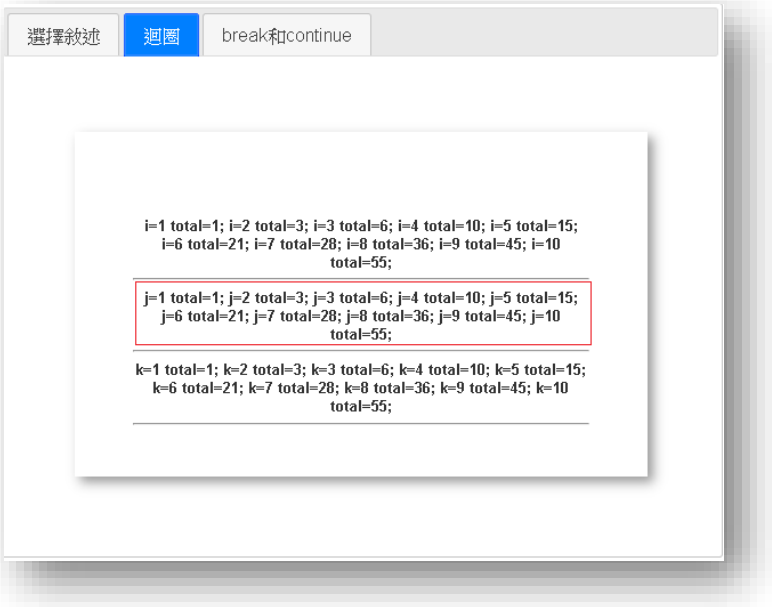



```
<div id="p2">
    <div class="main-box">
        <h5>
            <script>
                var s = "";
                var total = 0;
                for (let i = 1; i <= 10; i++) {
                    total += i;
                     s += `i=${i} total=${total};\t`;
                }
                document.write(`${s}<hr/>`);
                (略)
            </script>
        </h5>
    </div>
</div>
```

1.6.2.2 while 迴圈 (T01-02.html 2/3-B)

```
while (condition) {
    程式碼被執行(當條件式為 true)
}
```

■ 使用範例：T01-01.html 2/3-B

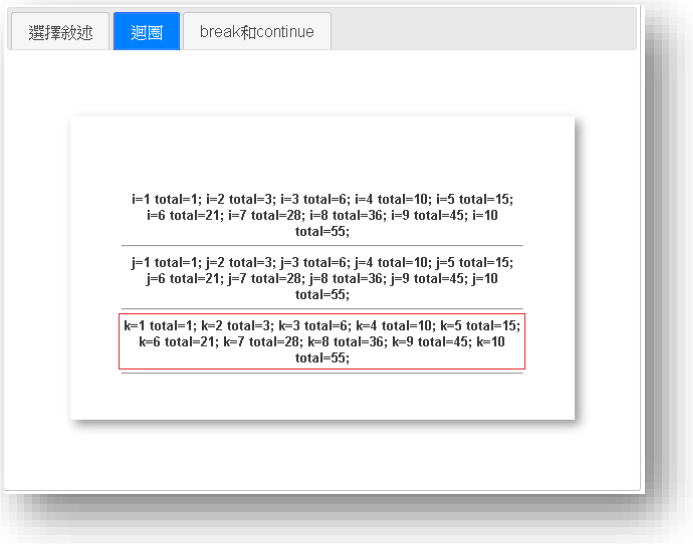



```
<div id="p2">
    <div class="main-box">
        <h5>
            <script>
                (略)
                  var s = "";
                  var total = 0;
                  var j = 1;
                  while (j <= 10) {
                        total += j;
                           s += `j=${j++} total=${total};\t`;
                  }
                  document.write(`${s}<hr/>`);
                  (略)
            </script>
        </h5>
    </div>
</div>
```

1.6.2.3 do while 迴圈 (T01-01.html 2/3-C)

```
do {
    程式碼被執行
 }
while (條件式/condition);
```

- 使用範例：T01-01.htm l2/3-C

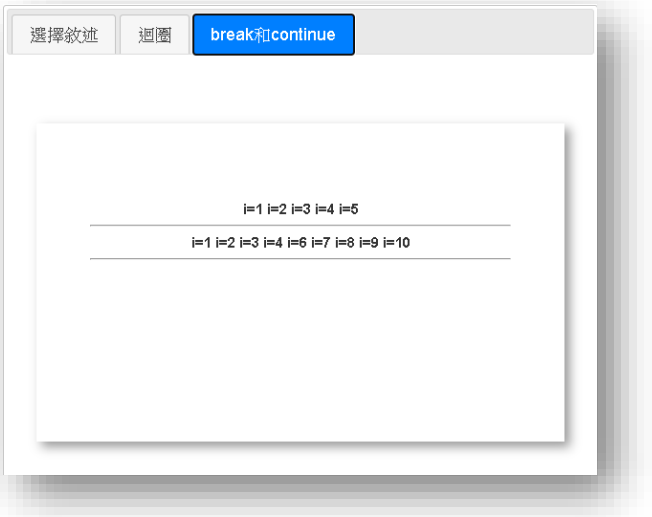



```
<div id="p2">
    <div class="main-box">
        <h5>
            <script>
                (略)
                var s = "";
                var total = 0;
                var k = 1;
                do {
                    total += k;
                    s += `k=${k++} total=${total};\t`;
                }
                while (k <= 10);
                document.write(`${s}<hr/>`);
            </script>
        </h5>
    </div>
</div>
```

1.6.3 break 和 continue (T01-01.html 3/3)

- break 敘述會中斷目前的迭代或 switch，並將程式流程轉到被中斷之敘述後的敘述。
- continue 敘述會中斷目前的迭代，並繼續執行迴圈的下一回合迭代。

- 使用範例：T01-1.html 3/3




```
<div id="p3">
    <div class="main-box">
        <h5>
            <script>
                var s = "";
                for (let i = 1; i <= 10; i++) {
                    if (i > 5)
                        break;
                    s += `i=${i}\t`;
                }
                document.write(`${s}<hr/>`);
                var s = "";
                for (let i = 1; i <= 10; i++) {
                    if (i == 5)
                        continue;
                    s += `i=${i}\t`;
                }
                document.write(`${s}<hr/>`);
            </script>
        </h5>
    </div>
</div>
```

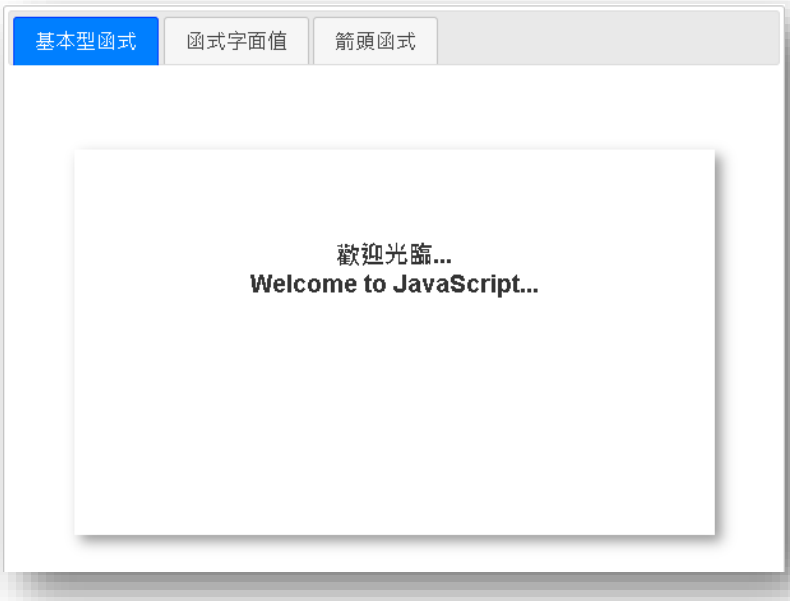
2 函式的定義

2.1 基本型函式 (T02-01.html 1/3)

- 一個基本型函式的定義由一系列的函式關鍵詞組成， 依次為：
  - ✓ 函式的名稱。
  - ✓ 包圍在括號()中，並由逗號區隔的一個函式**參數列表**(a list of parameters)。
  - ✓ 包圍在大括號{}中，用於定義函式功能的一些 JavaScript 敘述(statements)。
- 例如：

```
function greeting(message) {
    console.log(`${message}`);
}
```

- 使用範例：T02-01.html 1/3




```
<div id="p1">
    <div class="main-box">
        <h3>
            <script>
                var s = "";
                function greeting(message) {
                    s += `${message}<br/>`;
                }
                greeting("歡迎光臨...");
                greeting("Welcome to JavaScript...");
                document.write(`${s}`);
            </script>
        </h3>
    </div>
</div>
```

2.2 函式字面值 (T02-01.html 2/3)

- 函式字面值(Function Literal / Function Expression)是一種匿名函式(Anonymous Function)。

- 定義函式字面值的語法：

```
let myFunction = function ([param1[, param2[, ..., paramN]]]) {
    statements;
};
```

  - ✓ Parameters
    - ✧ paramN Optional
      - ◆ The name of an argument to be passed to the function.
    - ✧ statements Optional
      - ◆ The statements which comprise the body of the function.
- 使用範例：T02-01.html 2/3


```
<div id="p2">
    <div class="main-box">
        <h3>
            <script>
                var s = "";
                var f1 = function (message) { s += `${message}<br/>` };
                f1("歡迎光臨!!!");
                (function (message) { s += `${message}<br/>` })("Welcome to JavaScript!!!");
                document.write(`${s}`);
            </script>
        </h3>
    </div>
</div>
```

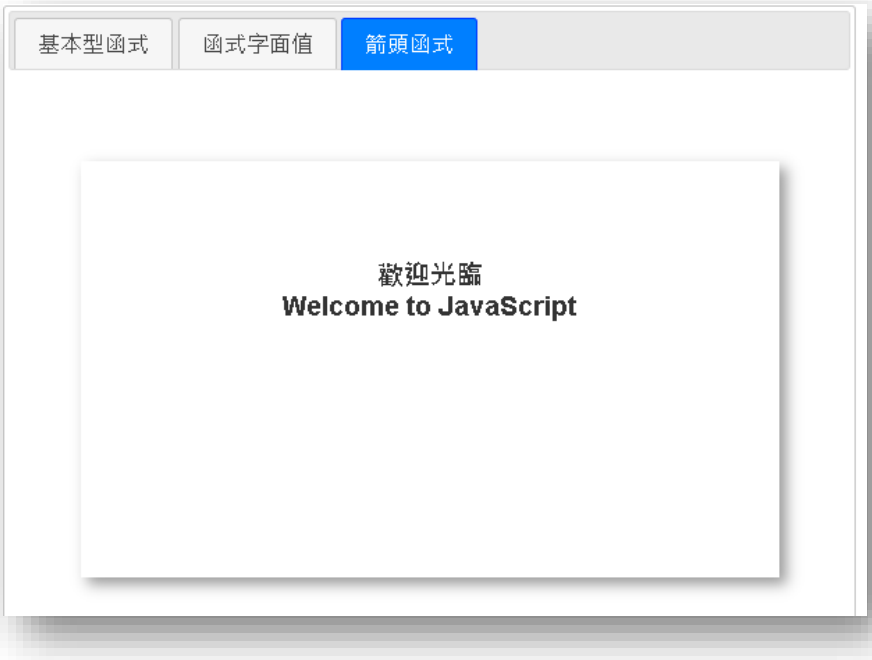
2.3 箭頭函式 (T02-01.html 3/3)

- 箭頭函式有比 Function Literal 還簡短的語法。
- 它適用於非物件方法的函式，但不能被用作建構子函式(constructor)。
- 箭頭函式也是一種匿名函式。
- 箭頭函式的結構：

```
( parameter-list ) => {        Statement[s];      }
```

- ✓ 參數只有一個時，(parameters)的小括號可以省略
- ✓ 當主體只含一個敘述時，左右大括號可以省略(但分號也必須拿掉)
- ✓ 左右大括號省略時，如果唯一的敘述是 return value;，則 return 關鍵字與分號必須省略

■ 使用範例：T02-01.html 3/3




```
<div id="p3">
    <div class="main-box">
        <h3>
            <script>
                var s = "";
                var f1 = (message) => { return `${message}<br/>`; };
                var f2 = message => `${message}<br/>` ;
                s += f1("歡迎光臨");
                s += f2("Welcome to JavaScript");
                document.write(`${s}`);
            </script>
        </h3>
    </div>
</div>
```

3 物件建立與存取

3.1 物件導論

- JacaScript 的"物件"可以與真實生活中的物件做類比，其概念可以用生活中有形的物體來做理解。
- 在 JavaScript 裡，"物件"是一個擁有自己的屬性(properties)、方法(methods)的獨立的實體，這些實體預設繼承了 JavaScript 內建的 Object 建構子函式的相關的屬性與方法(例如: toString()方法)。

3.1.1 建立特定類別(class)的物件的步驟 (T03-01.html 1/3-A)

- **定義建構子函式(constructor)**

```
function Dog(name, age) {
   //定義屬性
   this.name = name;
   this.age = age;
   //定義方法
   this.cry = function () { return `${this.name}汪汪汪...`; }
}
```

- **使用 new 建立物件實體**

```
var dog1 = new Dog('小黑', 3);
```

- **存取物件的屬性與方法**

```
document.write(`dog1.name=${dog1.name} dog1.age=${dog1.age} dog1.cry()=${dog1.cry()}<br/>`);
```

3.1.2 建立 Object 類別(class)的物件的步驟 (T03-01.html 1/3-B)

- **使用 new 建立物件實體**

```
var o1 = new Object();
```

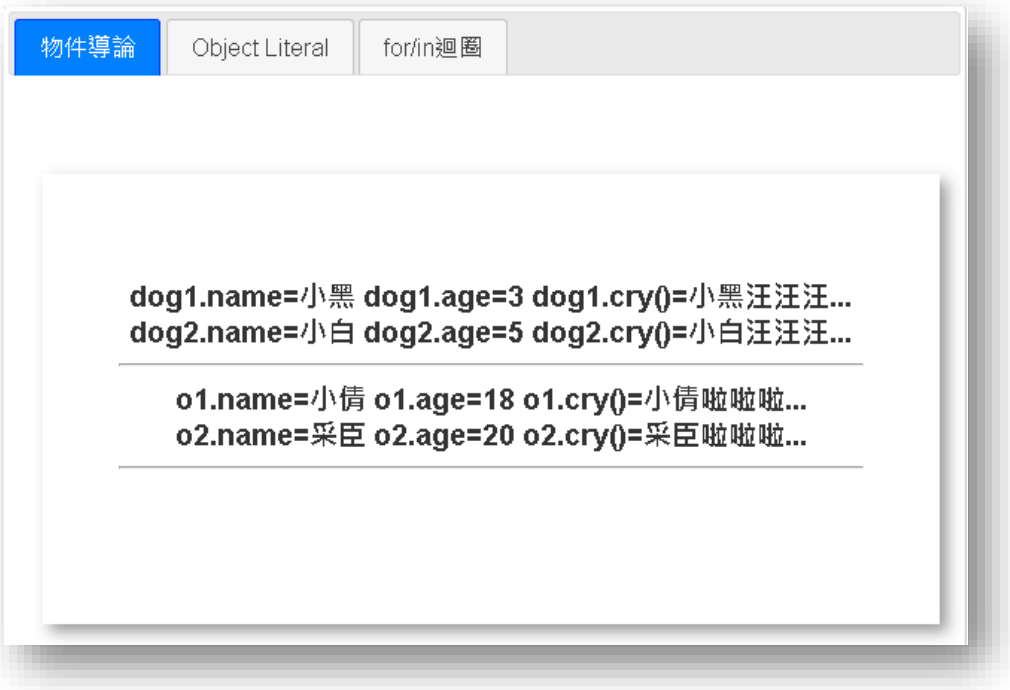
- **添加物件實體的屬性與方法**

```
 o1.name = '小倩';
 o1.age = 18;
 o1.sing = function () {
                    return `${this.name}啦啦啦...`;
             };
```

- **存取物件的屬性與方法**

```
document.write(`o1.name=${o1.name} o1.age=${o1.age} o1.cry()=${o1.sing()}<br/>`);
```

- 使用範例：T03-1.html 1/3




```
<div id="p1">
    <div class="main-box">
        <h3>
            <script>
            <script>
                //A. 建立特定類別(class)的物件
                //定義建構子函式(constructor)
                function Dog(name, age) {
                    //定義屬性
                    this.name = name;
                    this.age = age;
                    //定義方法
                    this.cry = function () { return `${this.name}汪汪汪...`; }
                }
                //建立物件實體
                var dog1 = new Dog('小黑', 3);
                var dog2 = new Dog('小白', 5);
                //存取物件的屬性與方法
                document.write(`dog1.name=${dog1.name} dog1.age=${dog1.age}
                                dog1.cry()=${dog1.cry()}<br/>`);
                document.write(`dog2.name=${dog2.name} dog2.age=${dog2.age}
                                dog2.cry()=${dog2.cry()}<hr/>`);
                //-------------------------------------------//
                //B. 建立Object類別(class)的物件
                //建立物件實體
                var o1 = new Object();
                //添加物件實體的屬性與方法
                o1.name = '小倩';
                o1.age = 18;
```

```
            o1.sing = function () {
                return `${this.name}啦啦啦...`;
            };
            //建立物件實體
            var o2 = new Object();
            //添加物件實體的屬性與方法
            o2.name = '采臣';
            o2.age = 20;
            o2.sing = function () {
                return `${this.name}啦啦啦...`;
            };
            //存取物件的屬性與方法
            document.write(`o1.name=${o1.name} o1.age=${o1.age}
                                o1.cry()=${o1.sing()}<br/>`);
            document.write(`o2.name=${o2.name} o2.age=${o2.age}
                                o2.cry()=${o2.sing()}<hr/>`);
        </script>
    </h3>
</div>
</div>
```

3.2 Object Literal (T03-01.html 2/3)

- Object Literal(Object literal/expression/initializer)合法的格式：
  - ✓ var obj1 = { name: "mary", age: 25 };
  - ✓ var obj2 = { "name": 'mary', "age": 25 };
  - ✓ var obj3 = { 'name': "mary", 'age': 25 };

- 使用範例：T03-01.html 2/3


```
<div id="p2">
```

```
<div class="main-box">
    <h3>
        <script>
            var obj1 = { name: "mary", age: 25 };
            var obj2 = { "name": 'mary', "age": 25 };
            var obj3 = { 'name': "mary", 'age': 25 };
            document.write(`obj1=${JSON.stringify(obj1)}<hr/>
                            obj2=${JSON.stringify(obj2)}<hr/>
                            obj3=${JSON.stringify(obj3)}<hr/>`);
        </script>
    </h3>
</div>
</div>
```

✓ JSON.stringify()方法使用說明，請參考附錄。

3.3 for/in 迴圈 (T03-01.html 3/3)

■ 用於物件的 for/in 迴圈

```
for ([let] property in object) {
    statement[s];
}
```

■ 使用範例：T03-01.html 3/3


```
<div id="p3">
    <div class="main-box">
        <h3>
            <script>
                va r s = "";
                var object = { name: "peter", age: 35 }
                for (let prop in object) {
                    s += `${prop}=${object[prop]}\t`;
                }
                document.write(`${s}`);
```

```
            </script>
        </h3>
    </div>
</div>
```

4 陣列建立與存取

4.1 陣列導論 (T04-01.html 1/3)

- JavaScript 中的 Array 類別被用於建構陣列。陣列為高階(high-level)、似列表(list-like)的物件。
- 陣列在 Javascript 裡面並沒有固定的長度與型別。
- 建立陣列：
  - ✓ **使用 Array 建構子函式(constructor)：**

    new Array(element0, element1[, ...[, elementN]])

    new Array(arrayLength)
    - ✧ 參數
      - ◆ elementN

        JavaScript 陣列會以傳入的元素進行初始化。
      - ◆ arrayLength
        - ● 如果傳遞給 Array 建構函數的唯一參數是 0 和 2^{32}-1（含）之間的整數，將回傳一個新的 JavaScript 陣列，其長度被設定為這個數字。
        - ● 如果參數是任何其他數值，將拋出 RangeError 異常。
  - ✓ **使用 Array Literal(Array expression) ：**

    [element0, element1, ..., elementN]
    - ✧ 使用範例請參考 4.2。
- 以元素之索引存取特定位置之陣列元素

```
var arr1 = new Array(1, 2, 3);
console.log(`arr1[0]=${arr1[0]}`); //print: arr1[0]=1
```

- 擷取陣列的元素個數

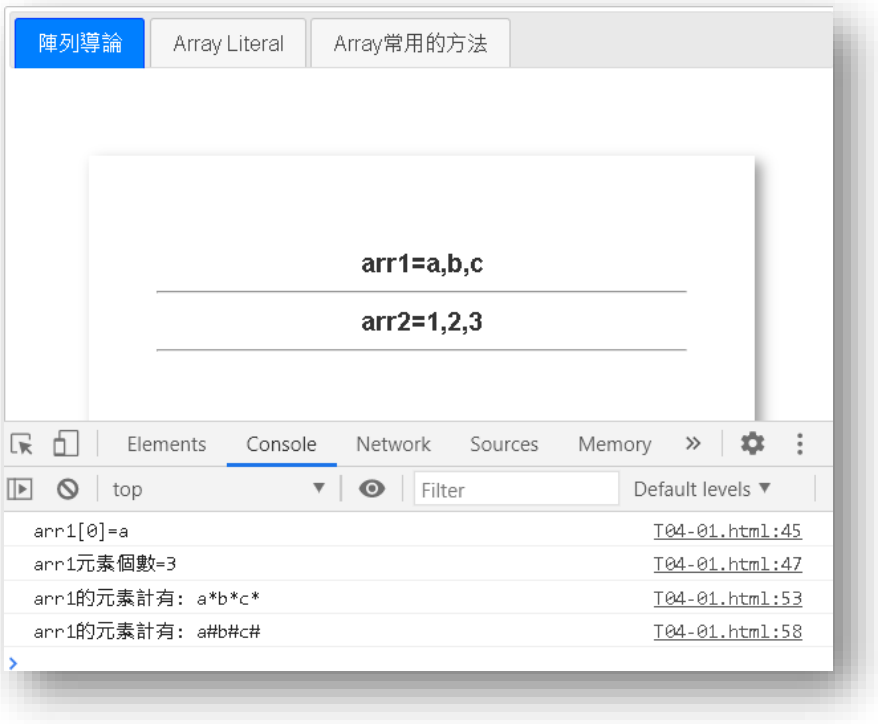
```
console.log(`arr1元素個數=${arr1.length}`); //print: arr1元素個數=3
```

- 巡覽陣列的每個元素

```
var s = '';
for (let i = 0; i < arr1.length; i++) {
        s += arr1[i] + '*';
}
console.log(`arr1的元素計有: ${s}`); //arr1的元素計有: 1*2*3*
s = '';
for (let i in arr1) {
        s += arr1[i] + '#';
}
console.log(`arr1的元素計有: ${s}`); //arr1的元素計有: 1#2#3#
```

```
** 用於陣列的 for/in 迴圈**
for ([let] index in array) {
   statement[s];
}
```

- 使用範例：T04-01.htm 1/3




```
<div id="p1">
    <div class="main-box">
        <h3>
            <script>
                var arr1 = new Array("a", "b", "c");
                var arr2 = new Array(3);
                arr2[0] = 1
                arr2[1] = 2;
                arr2[2] = 3;
                document.write(`arr1=${arr1.toString()}<hr/>`);
                document.write(`arr1=${arr2.toString()}<hr/>`);
                //以元素之索引存取特定位置之陣列元素
                console.log(`arr1[0]=${arr1[0]}`); //print: arr1[0]=a
                //擷取陣列的元素個數
                console.log(`arr1元素個數=${arr1.length}`); //print: arr1元素個數=3
                //巡覽陣列的每個元素
                var s = '';
                for (let i = 0; i < arr1.length; i++) {
                    s += arr1[i] + '*';
                }
                console.log(`arr1的元素計有: ${s}`); //arr1的元素計有: 1*2*3*
                s = '';
                for (let i in arr1) {
                    s += arr1[i] + '#';
                }
                console.log(`arr1的元素計有: ${s}`); //arr1的元素計有: 1#2#3#
            </script>
```

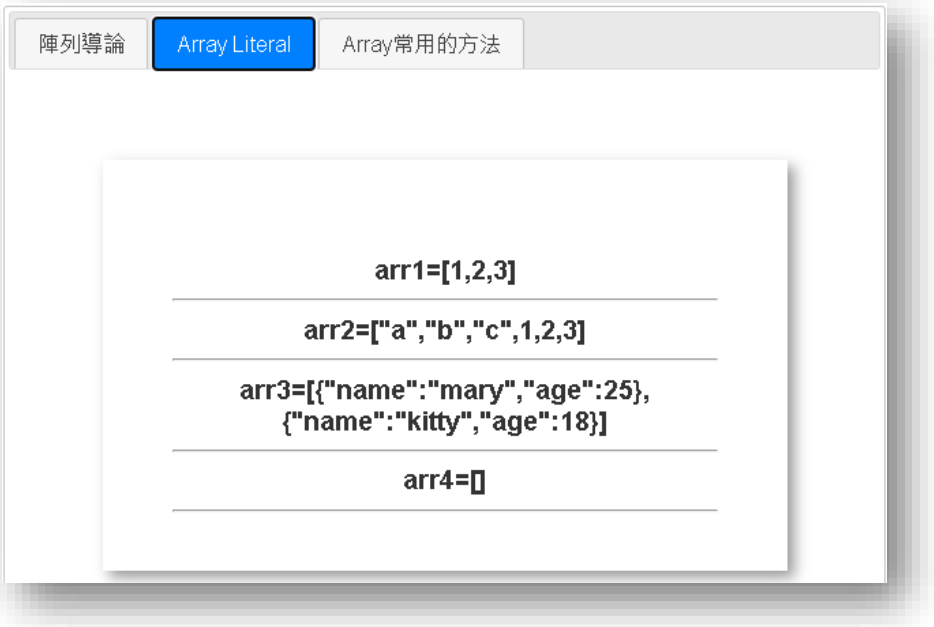
```
            </h3>
        </div>
    </div>
```

4.2 Array Literal (T04-01.html 2/3)

- Array Literal(Array expression)合法的格式

```
var arr1 = [1, 2, 3];
var arr2 = ["a", "b", "c", 1, 2, 3];
var arr3 = [{name:'mary',age:25}, {name:'kitty',age:18}];
var arr4 = [];
```

- 使用範例：T04-01.html 2/3



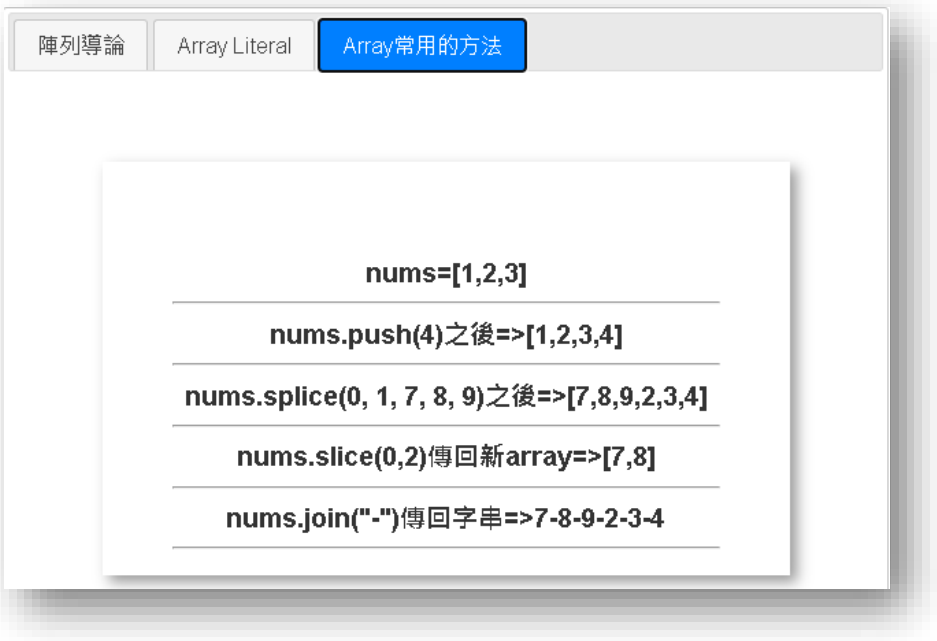

```
<div id="p2">
    <div class="main-box">
        <h3>
            <script>
                var arr1 = [1, 2, 3];
                var arr2 = ["a", "b", "c", 1, 2, 3];
                var arr3 = [{name:'mary',age:25}, {name:'kitty',age:18}];
                var arr4 = [];
                document.write(`arr1=${JSON.stringify(arr1)}<hr/>
                                arr2=${JSON.stringify(arr2)}<hr/>
                                arr3=${JSON.stringify(arr3)}<hr/>
                                arr4=${JSON.stringify(arr4)}<hr/>`);
            </script>
        </h3>
    </div>
</div>
```

4.3 Array 常用的方法 (T04-01.html 3/3)

- arr.push(element1[, ...[, elementN]])
  - ✓ push() 方法會添加一個或多個元素至陣列的末端，並且回傳陣列的新長度。
  - ✓ 參數
    - ✧ elementN
      欲添加至陣列末端的元素。
  - ✓ 回傳值
    - ✧ 呼叫此方法之物件的新 length 屬性值。
- arr.pop()
  - ✓ pop() 方法會移除並回傳陣列的最後一個元素。此方法會改變陣列的長度。
  - ✓ 回傳值
    - ✧ 自陣列中移除的元素；若陣列為空，則為 undefined。
- arr.join([separator])
  - ✓ join() 方法會將陣列中所有的元素連接、合併成一個字串，並回傳此字串。
  - ✓ 參數
    - ✧ separator 選擇性
      - ◆ 用來隔開陣列中每個元素的字串。
      - ◆ 如果必要的話，separator 會自動被轉成字串型態。
      - ◆ 如果未傳入此參數，陣列中的元素將預設用英文逗號（「,」）隔開(相當於 arr.toString( ))。
      - ◆ 如果 separator 是空字串，合併後，元素間不會有任何字元。
  - ✓ 回傳值
    - ✧ 一個合併所有陣列元素的字串。
    - ✧ 假如 arr.length 為 0，將回傳空字串。
- arr.slice([begin[, end]])
  - ✓ slice() 方法會回傳一個新陣列物件，為原陣列選擇之 begin 至 end（不含 end）部分的淺拷貝（shallow copy）。而原本的陣列將不會被修改。
  - ✓ 參數
    - ✧ begin 選擇性
      - ◆ 自哪一個索引（起始為 0）開始提取拷貝。
      - ◆ 假如 begin 為 undefined，則 slice 會從索引 0 開始提取。
    - ✧ end 選擇性
      - ◆ 至哪一個索引（起始為 0）之前停止提取。slice 提取但不包含至索引 end。
      - ◆ 若省略了 end，則 slice 會提取至陣列的最後一個元素（arr.length）。
      - ◆ 假如 end 大於陣列的長度，slice 會提取至陣列的最後一個元素（arr.length）。
  - ✓ 回傳值
    - ✧ 一個包含提取之元素的新陣列。
- array.splice(start[, deleteCount[, item1[, item2[, ...]]]])
  - ✓ splice() 方法可以藉由刪除既有元素並／或加入新元素來改變一個陣列的內容。
  - ✓ 參數
    - ✧ start
      - ◆ 陣列中要開始改動的元素索引（起始為 0）。
      - ◆ 若索引大於陣列長度，則實際開始的索引值會被設為陣列長度。
    - ✧ deleteCount 選擇性
      - ◆ 一個表示欲刪除的原陣列元素數量的整數。
      - ◆ 若 deleteCount 為 0 或是負數，則不會有元素被刪除。
    - ✧ item1, item2, ... 選擇性
      - ◆ 從 start 開始，要加入到陣列的元素。 如果你沒有指定任何元素，則 splice() 只會依照 start 和 deleteCount 刪除陣列的元素。
  - ✓ 回傳值

- ✧ 一個包含被刪除的元素陣列。
- ✧ 如果只有一個元素被刪除，依舊是回傳包含一個元素的陣列。 倘若沒有元素被刪除，則會回傳空陣列。

■ 使用範例：T04-02.html 3/3




```
<div id="p3">
    <div class="main-box">
        <h3>
            <script>
                var nums = [1, 2, 3];
                document.write(`nums=${JSON.stringify(nums)}<hr/>`);
                nums.push(4);
                document.write(`nums.push(4)之後=>${JSON.stringify(nums)}<hr/>`);
                nums.splice(0, 1, 7, 8, 9);
                document.write(`nums.splice(0, 1, 7, 8, 9)之後=>${JSON.stringify(nums)}<hr/>`);
                var nums2 = nums.slice(0,2);
                document.write(`nums.slice(0,2)傳回新array=>${JSON.stringify(nums2)}<hr/>`);
                var s = nums.join("-");
                document.write(`nums.join("-")傳回字串=>${s}<hr/>`);
            </script>
        </h3>
    </div>
</div>
```

5 事件處理

5.1 HTML 事件處理器屬性 (T05-01.html 1/2)

- 許多 DOM 元素可被設定「接受」(accept)，或稱為「監聽 」(listen)事件，並在發生時執行處理事件的程式碼。
- 事件處理器(Event-handlers)通常會使用 EventTarget.addEventListener() 來被連結或稱為附加至各個 HTML 元素(例如 <button>、 、 等)，且此方式一般也是用來取代舊的 HTML 事件處理器屬性(attributes)。
- 常見的 HTML 事件處理器屬性如下：
  - ✓ Window Event Attribute(應用於<body>元素)
    - ✧ onload
      - ◆ Fires after the page is finished loading
  - ✓ Form Event Attribute
    - ✧ onsubmit
      - ◆ Fires when a form is submitted
    - ✧ onblur:
      - ◆ Fires the moment that the element loses focus
    - ✧ onchange
      - ◆ Fires the moment when the value of the element is changed
    - ✧ oncontextmenu
      - ◆ Script to be run when a context menu is triggered
    - ✧ onfocus
      - ◆ Fires the moment when the element gets focus
    - ✧ oninput
      - ◆ Script to be run when an element gets user input
  - ✓ Keyboard Event Attribute
    - ✧ onkeydown
      - ◆ Fires when a user is pressing a key
    - ✧ onkeypress
      - ◆ Fires when a user presses a key
      - ◆ 當按鍵不是特殊鍵(modifier key)，例如 Ctrl 、Alt……等等，會送出 keypress event。
    - ✧ onkeyup
      - ◆ Fires when a user releases a key
  - ✓ Mouse Event Attribute
    - ✧ onclick
      - ◆ Fires on a mouse click on the element
    - ✧ ondblclick
      - ◆ Fires on a mouse double-click on the element
    - ✧ onmousedown
      - ◆ Fires when a mouse button is pressed down on an element
    - ✧ onmousemove
      - ◆ Fires when the mouse pointer is moving while it is over an element
    - ✧ onmouseout
      - ◆ Fires when the mouse pointer moves out of an element
    - ✧ Onmouseover
      - ◆ Fires when the mouse pointer moves over an element
    - ✧ onmouseup
      - ◆ Fires when a mouse button is released over an element

- ✧ Onwheel
  - ◆ Fires when the mouse wheel rolls up or down over an element

■ 使用範例：T05-01.html 1/2


```
<!DOCTYPE html>
<html>
<head>
    <title>事件處理</title>
    <link rel="icon" href="./favicon.ico" />
    <link href="stylesheets/jquery-ui/base/jquery-ui.min.css" rel="stylesheet" />
    <style>
        (略)
        textarea {
            width: 300px;    height: 80px;
        }
        .message-pane {
            width: 300px;     height: 150px;
            overflow: auto; margin: auto;
        }
    </style>
    <script src="javascripts/jquery-3.6.0.min.js"></script>
    <script src="javascripts/jquery-ui.min.js"></script>
    <script type="text/javascript">
        $(function () {
            $("#tabs").tabs();
        });
    </script>
</head>
<body>
    <div id="tabs">
```

```
<ul>
    <li><a href="#p1">HTML事件處理器屬性</a></li>
    <li><a href="#p2">使用JavaScript API附加事件處理器</a></li>
</ul>
<div id="p1">
    <div class="main-box">
        請在下面的元件上用滑鼠點擊一下
        <textarea id="ta1" onmousedown="myEventHandler(event)"
                  onmouseup="myEventHandler(event)"
                  onclick="myEventHandler(event)"
                  onmouseover="myEventHandler(event)"
                  onmouseout="myEventHandler(event)"
                  ></textarea>
        <div class="message-pane" id="div1"></div>
        <script>
            var div1 = document.getElementById('div1');
            function myEventHandler(e) {
                div1.innerHTML += `${e.type}...<br/>`;
            }
        </script>
    </div>
</div>
(略)
</div>
</body>
</html>
```

- 範例相關 API
  - ✧ document.getElementById(id)

    The Document method getElementById() returns an Element object representing the element whose id property matches the specified string.

    Since element IDs are required to be unique if specified, they're a useful way to get access to a specific element quickly.

    - Parameters
      - id

        The ID of the element to locate. The ID is case-sensitive string which is unique within the document; only one element may have any given ID.
    - Return value
      - An Element object describing the DOM element object matching the specified ID, or null if no matching element was found in the document.
  - ✧ Element.innerHTML

    The Element property innerHTML gets or sets the HTML contained within the element.

    - const content = element.innerHTML; (擷取元素的內容並指派給 content 常數)
    - element.innerHTML = htmlString; (設定元素的內容為 htmlString)

5.2 使用 JavaScript API 附加事件處理器 (T05-01.html 2/2)

- target.addEventListener(type, listener [, options]);
- target.addEventListener(type, listener [, useCapture]);
  - ✓ The EventTarget method addEventListener() sets up a function that will be called whenever the specified event is delivered to the target.

- ✓ EventTarget is a DOM interface implemented by objects that can receive events and may have listeners for them.
- ✓ Common targets are Element, Document, and Window, but the target may be any object that supports events (such as XMLHttpRequest).
- ✓ Parameters
  - ✧ type
    - ◆ A case-sensitive string representing the event type(事件類型) to listen for.
  - ✧ listener
    - ◆ JavaScript function.
  - ✧ options Optional
    - ◆ An options object specifies characteristics about the event listener. The available options are:
      - ● capture
        A Boolean indicating that events of this type will be dispatched to the registered listener before being dispatched to any EventTarget beneath it in the DOM tree.
      - ● once
        A Boolean indicating that the listener should be invoked at most once after being added. If true, the listener would be automatically removed when invoked.
      - ● passive
        A Boolean which, if true, indicates that the function specified by listener will never call preventDefault(). If a passive listener does call preventDefault(), the user agent will do nothing other than generate a console warning.
- ✓ Return value
  - ✧ Undefined

■ 使用範例：T05-01.html 2/2


```
<!DOCTYPE html>
<html>
<head>
```

```
    (略)
</head>
<body>
    <div id="tabs">
        <ul>
            <li><a href="#p1">HTML事件處理器屬性</a></li>
            <li><a href="#p2">使用JavaScript API附加事件處理器</a></li>
        </ul>
        (略)
        <div id="p2">
            <div class="main-box">
                請在下面的元件上按下任意按鍵
                <textarea id="ta2"></textarea>
                <div class="message-pane" id="div2"></div>
                <script>
                    var ta2 = document.getElementById('ta2');
                    var div2 = document.getElementById('div2');
                    ta2.addEventListener('keydown', function (event) {
                        div2.innerHTML += `code=${event.code}
                            key=${event.key} type=${event.type}...<br/>`;
                        //event.code:  傳回事件對應的按鍵的代碼(字串)
                        //event.key:  傳回事件對應的按鍵的值(字串)
                    });
                    ta2.addEventListener('keypress', function (event) {
                        div2.innerHTML += `code=${event.code}
                            key=${event.key} type=${event.type}...<br/>`;
                    });
                    ta2.addEventListener('keyup', function (event) {
                        div2.innerHTML += `code=${event.code}
                            key=${event.key} type=${event.type}...<br/>`;
                    });
                </script>
            </div>
        </div>
        (略)
    </div>
</body>
</html>
```

6 時間與計時器

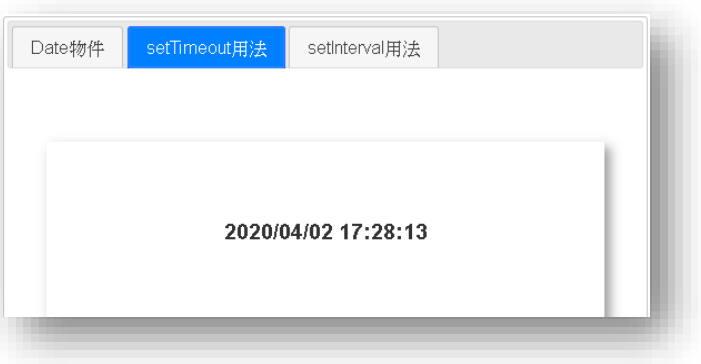
6.1 Date 物件 (T06-01.html 1/3)

- Date 建構函式可用來建立一個 JavaScript Date 物件來指向某一個時間點。
- Date 物件是基於世界標準時間（UTC）1970 年 1 月 1 日開始的毫秒數值來儲存時間。
- 常用的 Date 建構函式
  - ✓ new Date();
  - ✓ new Date(year, month[, day[, hour[, minutes[, seconds[, milliseconds]]]]]);
    - ✧ 參數
      - ◆ year
        表示年份的整數。當數值落在 0 到 99 之間，表示 1900 到 1999 之間的年份。
      - ◆ month
        表示月份的整數。由 0 開始（一月）到 11 （十二月）。
      - ◆ day
        選用。表示月份中第幾天的整數值。
      - ◆ hour
        選用。表示小時數的整數值。
      - ◆ minute
        選用。表示分鐘數的整數值。
      - ◆ second
        選用。表示秒數的整數值。
      - ◆ millisecond
        選用。表示毫秒數的整數值。
- 常用的 Date 物件方法
  - ✓ .getFullYear()
    回傳本地時間的年份（ 以 4 位數表現）。
  - ✓ .getMonth()
    回傳本地時間的月份（0-11）。
  - ✓ .getDate()
    回傳本地時間月份中的日期（1-31）。
  - ✓ .getHours()
    回傳本地時間的小時（0-23）。
  - ✓ .getMinutes()
    回傳本地時間的分鐘數（0-59）。
  - ✓ .getSeconds()
    回傳本地時間的秒數（0-59)。
  - ✓ .getMilliseconds()
    回傳本地時間的毫秒數（0-999）。
  - ✓ .getDay()
    回傳本地時間星期中的日子（0-6）。
  - ✓ .getTime()
    回傳由 1970-01-01 00:00:00 UTC 開始，到代表時間經過的毫秒數（以負值表示 1970 年之前的時間）。
- 使用範例：T06-01.html 1/3


```
<div id="p1">
    <div class="main-box">
        <h3>
            <script>
                var now = new Date();
                var s = `${now.getFullYear()}/${now.getMonth() + 1}/${now.getDate()}
                        ${now.getHours()}:${now.getMinutes()}:${now.getSeconds()}`;
                document.write(`${s}`);
            </script>
        </h3>
    </div>
</div>
```

6.2 setTimeout 用法 (T06-01.html 2/3)

- setTimeout(function, milliseconds, param1, param2, ...) (指定時間過後執行指定的函式一次)
  - ✓ The setTimeout() method calls a function or evaluates an expression after a specified number of milliseconds.
  - ✓ Parameters
    - ✧ function
      - ◆ Required.
      - ◆ The function that will be executed
    - ✧ Milliseconds
      - ◆ Optional.
      - ◆ The number of milliseconds to wait before executing the code.
      - ◆ If omitted, the value 0 is used
    - ✧ param1, param2, ...
      - ◆ Optional. Additional parameters to pass to the function
    - ✧ Return Value:
      - ◆ A Number, representing the ID value of the timer that is set.
      - ◆ Use this value with the clearTimeout() method to cancel the timer
- 使用範例：T06-01.html 2/3




```
<div id="p2">
    <div class="main-box">
        <h3>
            <span id="span1"></span>
            <script>
                function formatDigit(digit) {
                    return ("0" + digit).slice(- 2); //使用說明請參考附錄
                }
                function displayNowTime() {
                    var now = new Date();
                    var y = now.getFullYear(), m = formatDigit(now.getMonth() + 1),
                        d = formatDigit(now.getDate()), h = formatDigit(now.getHours()),
                        M = formatDigit(now.getMinutes()), s = formatDigit(now.getSeconds());
                    var s = `${y}/${m}/${d} ${h}:${M}:${s}`;
                    document.getElementById("span1").innerHTML = s;
                    setTimeout(function () { displayNowTime(); }, 1000);
                }
                setTimeout(    function () {      displayNowTime();    }, 0);
            </script>
        </h3>
    </div>
</div>
```

6.3 setInterval 用法 (T06-01.html 3/3)

- setInterval(function, milliseconds, param1, param2, ...) (間隔時間重複執行指定的函式)
  - ✓ The setInterval() method calls a function or evaluates an expression at specified intervals (in milliseconds).
  - ✓ Parameters
    - ✧ function
      - ◆ Required.
      - ◆ The function that will be executed
    - ✧ milliseconds
      - ◆ Required.
      - ◆ The intervals (in milliseconds) on how often to execute the code.
      - ◆ If the value is less than 10, the value 10 is used
    - ✧ param1, param2, ...
      - ◆ Optional.
      - ◆ Additional parameters to pass to the function

- ✧ Return Value:
  - ◆ A Number, representing the ID value of the timer that is set.
  - ◆ Use this value with the clearInterval() method to cancel the timer

■ 使用範例：T06-01.html 3/3


```
<div id="p3">
    <div class="main-box">
        <h3>
            <span id="span2"></span>
            <script>
              function displayCurrentTime() {
                  var now = new Date();
                  var y = now.getFullYear(), m = formatDigit(now.getMonth() + 1),
                      d = formatDigit(now.getDate()), h = formatDigit(now.getHours()),
                      M = formatDigit(now.getMinutes()), s = formatDigit(now.getSeconds());
                  var s = `${y}/${m}/${d} ${h}:${M}:${s}`;
                  document.getElementById("span2").innerHTML = s;
              }
              setInterval(function () { displayCurrentTime(); }, 1000);
            </script>
        </h3>
    </div>
</div>
```

7 附錄

7.1 使用 Chrome「開發者工具」測試與除錯

7.1.1 Console 面板

- 在 Google Chrome 瀏覽器進行 JavaScript 程式除錯時，可以使用「開發人員工具」中的 Console 面板。
- 當網頁內的 JavaScript 程式出現錯誤時，Chorme 會在 Console 上顯示紅色的錯誤訊息(error message)。
- 網頁程式開發人員也可以使用 console.log()等 API 輸出程式當中的運算式的執行結果，以進行程式偵錯。

7.1.2 Network 面板

- 當用戶端的網頁欲傳送 HTTP 請求(Request)給網頁伺服器並等候其回應(Response)時，在程式的除錯上通常會使用「開發人員工具」的 Network 面板來查看 HTTP 請求與 HTTP 回應的標頭(headers)標頭與內容。
- 例如
  - ✓ 使用者在瀏覽器端瀏覽 **Appendix1.html** 頁面(程式碼如下)時，在姓名欄位輸入「小倩」後送出 HTTP 請求

```
<form action="WebServer1.aspx" method="get">
    <label>
        姓名 <input type="text" id="text1" name="username" size="20" value="小倩" />
    </label>
    <input type="submit" value="送出" />
</form>
```

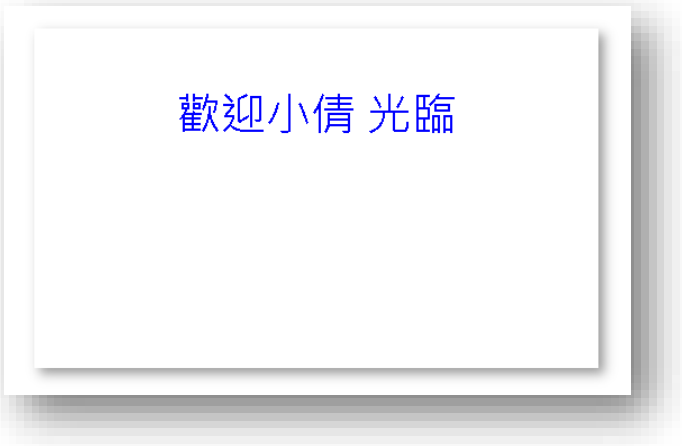
  - ✓ 瀏覽器實際上會發送下列 HTTP「請求訊息 (Request Message)」

```
GET /WebServer1.aspx?username=%E5%B0%8F%E5%80%A9 HTTP/1.1
Host: localhost:55224
(略)
```

  - ✓ 網頁伺服器在接到 HTTP 請求後會回應如下之 HTTP「回應訊息(Response Message)」

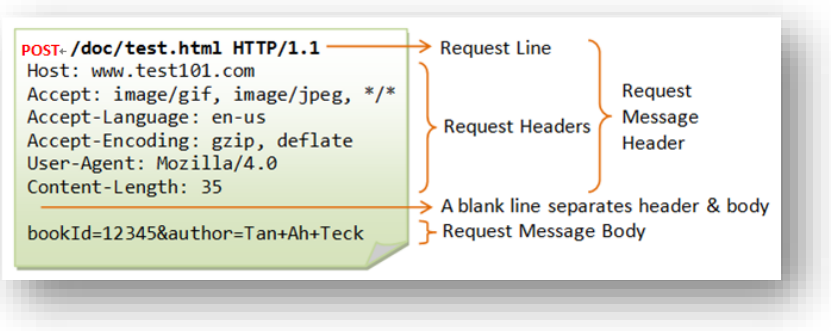
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Date: Tue, 27 Apr 2021 04:10:17 GMT
Content-Length: 421
(略)

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Welcome</title>
    <style>
        .main-box {
            margin: 50px auto;
            width: 400px;
            height: 200px;
            padding: 50px;
            box-shadow: 5px 5px 10px #999;
            border: 1px solid #fff;
            text-align: center;
            font-size: 40px;
            color: blue;
        }
    </style>
</head>
<body>
    <div class="main-box">歡迎小倩  光臨</div>
</body>
</html>
```

✓ 最後瀏覽器在接到 HTTP 回應後將渲染(Render)如下之頁面



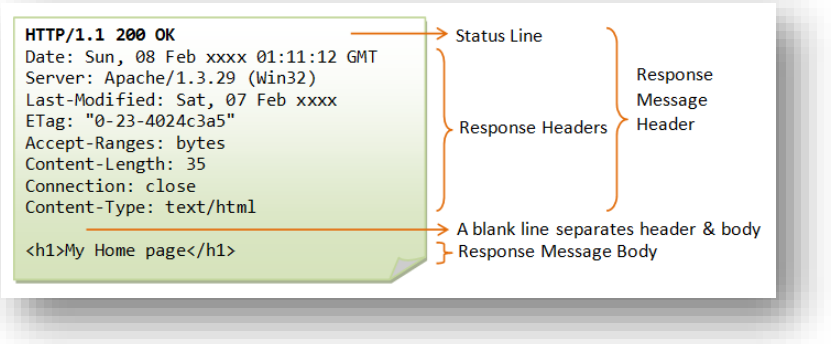
7.1.2.1 HTTP Request Message 與 Response Message 之格式

■ HTTP Request Message 格式



Request-Line = Method Request-URI HTTP-Version

- HTTP Response Message 格式



Status-Line = HTTP-Version Status-Code Reason-Phrase

7.1.2.2 表單資料進行 URL encoding (Percent-encoding)的規則

- The alphanumeric characters "a" through "z", "A" through "Z" and "0" through "9" remain the same.
- The special characters ".", "-", "*", and "_" remain the same.
- The space character " " is converted into a plus sign "+". (註: %20 亦可)
- All other characters are unsafe and are first converted into one or more bytes using some encoding scheme. Then each byte is represented by the 3-character string "%xy", where xy is the two-digit hexadecimal representation of the byte.

例如：

<input type='text' name='username' value='小甜甜 A9.*_-' /> 經過 URL encoding (字元編碼：utf-8)之後產生之表單資料如下：

username=%E5%B0%8F%E7%94%9C%E7%94%9C+A9.*_-

7.2 JSON

7.2.1 導論

- JSON(JavaScript Object Notation)是一種輕量級的資料交換語言，以文字為基礎，且易於讓人閱讀。
- 儘管 JSON 是 Javascript 程式語言的一個子集，但 JSON 是獨立於語言的文字格式，並且採用了類似於 C 語言家族的一些習慣。
- JSON 的官方 MIME 類型是 application/json，副檔名是.json。
- JSON 支援的資料型別有：
  - ✓ Number
  - ✓ String
    - ✧ 須以雙引號包圍
  - ✓ Boolean
  - ✓ Array
    - ✧ 須使用 JavaScript 的 Array literal 語法表示
  - ✓ Object
    - ✧ 須使用 JavaScript 的 Object literal 語法表示
    - ✧ 屬性名稱須以雙引號包圍
  - ✓ null
- JSON 最常用兩個資料結構：
  - ✓ Object
  - ✓ Array
- JSON 格式範例：

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "height_cm": 197.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

7.2.2 JavaScript 常用 JSON API

- JSON.parse(jsonString)
  - ✓ 解析 JSON 字串並傳回解析後所得的 JavaScript 的 String、Number、Boolean、Null、Object 或 Array 值
  - ✓ 參數：
    - ✧ jsonString：well-formed JSON 字串
  - ✓ 範例：

```
<script>
    var jsonStr = `[{ "name": "mary", "age": 25 },{ "name": "kitty", "age": 18 }]`;
    var ary = JSON.parse(jsonStr);
    var s = "";
    for (let index in ary) {
        for (let prop in ary[index]) {
            s += `${prop}=${ary[index][prop]} `;
        }
        s += `<hr/>`;
    }
    document.write(`${s}`);
</script>
```

- JSON.stringify(javascriptValue)
  - ✓ 將 JavaScript value 轉成 JSON 格式字串
  - ✓ 參數：
    - ✧ JavaScipt Value (String、Number、Boolean、Null、Object、Array)
  - ✓ 回傳值：
    - ✧ JSON 格式字串
  - ✓ 範例：

```
<script>
    var array = [{ name: 'mary', age: 25 }, { name: 'kitty', age: 18 }];
     document.write(`JSON.stringify(array)=${JSON.stringify(array)}<hr/>`);
</script>
```

7.3 字串(String)常用方法

7.3.1 str.substring(indexStart[, indexEnd]) 取子字串

- The substring() method returns the part of the string between the start and end indexes, or to the end of the string.
- Parameters
  - ✓ indexStart
    - ✧ The index of the first character to include in the returned substring.
    - ✧ Any argument value that is less than 0 or greater than stringName.length is treated as if it were 0 and stringName.length, respectively.
  - ✓ indexEnd Optional
    - ✧ The index of the first character to exclude from the returned substring.
- Return value
  - ✓ A new string containing the specified part of the given string.

7.3.2 str.split([separator]) 將字串切割成字串陣列

- The split() method divides a String into an ordered set of substrings, puts these substrings into an array, and returns the array. The division is done by searching for a pattern; where the pattern is provided as the first parameter in the method's call.
- Parameters
  - ✓ separator Optional
    - ✧ The pattern describing where each split should occur.
    - ✧ The separator can be a simple string or it can be a regular expression.
    - ✧ If separator is omitted or does not occur in str, the returned array contains one element consisting of the entire string.
- Return value
  - ✓ An Array of strings, split at each point where the separator occurs in the given string.

7.3.3 str.charAt(index) 取指定位置的單一字元的字串

- The String object's charAt() method returns a new string consisting of the single UTF-16 code unit located at the specified offset into the string.
- Parameters
  - ✓ index
    - ✧ An integer between 0 and 1-less-than the length of the string.
    - ✧ If the index cannot be converted to the integer or no index is provided, the default is 0, so the first character in of str is returned.
- Return value
  - ✓ A string representing the character (exactly one UTF-16 code unit) at the specified index.
  - ✓ If index is out of range, charAt() returns an empty string.

7.3.4 str.indexOf(searchValue [, fromIndex]) 取指定子字串的位置

- The indexOf() method returns the index within the calling String object of the first occurrence of the specified value, starting the search at fromIndex. Returns -1 if the value is not found.
- Parameters
  - ✓ searchValue
    - ✧ The string value to search for.
  - ✓ fromIndex Optional
    - ✧ An integer representing the index at which to start the search. Defaults to 0.
- Return value
  - ✓ The index of the first occurrence of searchValue, or -1 if not found.

7.3.5 str.slice(beginIndex[, endIndex]) 取字串切片

- The slice() method extracts a section of a string and returns it as a new string, without modifying the original string.
- Parameters
  - ✓ beginIndex
    - ✧ The zero-based index at which to begin extraction. If negative, it is treated as str.length + beginIndex. (For example, if beginIndex is -3 it is treated as str.length - 3.)
    - ✧ If beginIndex is greater than or equal to str.length, slice() returns an empty string.
  - ✓ endIndex Optional
    - ✧ The zero-based index before which to end extraction. The character at this index will not be included.
    - ✧ If endIndex is omitted, slice() extracts to the end of the string. If negative, it is treated as str.length + endIndex. (For example, if endIndex is -3 it is treated as str.length - 3.)
- Return value
  - ✓ A new string containing the extracted section of the string.

7.3.6 使用範例

```
<script>
    var s = "Welcome to JavaScript!";
    document.write(`s=${s}<br/>`);
    document.write(`s.substring(0,7)=${s.substring(0, 7)}<br/>`);
    document.write(`s.split(' ')=${s.split(' ')}<br/>`);
    document.write(`s.charAt(0)=${s.charAt(0)}<br/>`);
    document.write(`s.indexOf("Java")=${s.indexOf("Java")}<br/>`);
    document.write(`"012".slice(-2)=${"012".slice(-2)}<br/>`);
</script>
```

s=Welcome to JavaScript!
s.substring(0,7)=Welcome
s.split(' ')=Welcome,to,JavaScript!
s.charAt(0)=W
s.indexOf("Java")=11
"012".slice(-2)=12