# JavaScript 基礎

系統開發部 程式設計科

# JavaScript 簡介

- JavaScript 是一種廣泛用於用戶端 Web 開發的腳本語言，
  常用來給 HTML 網頁添加動態功能，比如響應使用者的各種操作。
  最初受 Java 啟發而開始設計的，目的之一就是「看上去像 Java」，
  因此語法上有類似之處，一些名稱和命名規範也借自 Java。

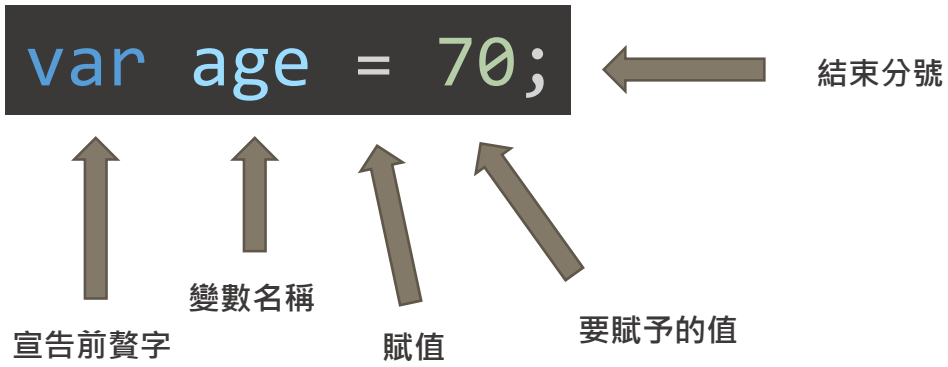
- 基本特點：
  - Browser 端語言
  - 物件導向
  - 是一種解釋性程式語言（代碼不進行預編譯）。
  - 主要用來向 HTML 頁面添加互動行為。
  - 可以直接嵌入 HTML 頁面，也可以寫成單獨的 .js 檔案（有利於結構和行為的分離）。

# 宣告

注意：不能重複宣告同一個變量

js變量命名限制：
1.只能包含：字母、數字、$、底線
2.首字不能為數字

var age的區塊為宣告(declare);
age = 70的部分為賦值;
當為變量時，宣告跟賦值是可以拆開的，也就是等同於
var age;
age = 70;




不再建議使用var來宣告變量

- 在宣告上有幾個需要注意的事項：
  - 需要加上 var 的前贅字，沒有加的話在某些情況會有意外的結果
  - 給予變數名稱
  - 等號代表把右側的值賦予給左側的變數

在js的的世界裡，儲存數據的兩種方式，一種是儲存成變量(Variable)；另一種則是儲存成常量(Constant)。

變量宣告與賦值可分開(也可一起)，常量則必須宣告賦值同時做；變量在過程中值可以變，常量則不行。

變量宣告：var、let

常量宣告：const

# 弱型別

- 在看 JavaScript 的宣告時，會發現不論什麼型別，全部都是用 var 前贅字宣告，
  也就是大家稱 JavaScript 為弱型別語言的原因，如果拿它的宣告跟 Java 比較的話：

| Java | JavaScript |
| --- | --- |
| String test = "test"; | var test = 'test'; |
| int i = 1; | var i = 1; |
| boolean flag = true; | var flag = true; |

- Java 在宣告時，前面就會加上型別，也就是變數的型別在最一開始就會確定下來。
  也就是說 JavaScript 的變數沒有型別、值才有型別。

# 基本型別

# 基本型別

- 到 ES6 為止，總共有 6 種基本型別 string、number、boolean、undefined、null、symbol，課程中會對前五種做基本的介紹，第六種有興趣可以自行查詢。

| 型別 | 宣告樣式 |
| --- | --- |
| string | var string = 'test'; |
| number | var number = 1; |
| boolean | var flag = true; |
| undefined | var foo = undefined; |
| null | var bar = null; |
| symbol | var baz = Symbol(1); |

# 基本型別 - 字串

- JavaScript 中沒有字元型態，以單引號或雙引號來包括單一字元，都是字串型態。

```
var str1 = 'test';
var str2 = "test";

console.log(str1);
console.log(str2);
```

```
test
test
```

➢ console.log(…) 的語法：能幫忙把 () 內的結果給打印出來

- 而字串的拼接都是使用 '+' 來進行拼接：

```
console.log(str1 + str2);
```

```
testtest
```

# 基本型別 - 字串

可用三種方法表示：單引號(')、雙引號(")、反引號(`)，反引號可用來嵌入變量

- 建議在 JavaScript 程式中採用單引號，而將雙引號保留給 HTML 使用，避免跳脫文字的麻煩。

例如：var html = '<input type="text" value="defalut">';

例外：var text =  "It's time";

```
let age = 25;
console.log(`I am ${age} years old`); // I am 25 years old
```

➢ 跳脫字元：\ ( 反斜線 )

# 基本型別 - 數字

number能表示的數值範圍約莫在正負2的53次方之間(-9007199254740991~9007199254740991)，若想表達高精度的大整數，用number處理會運算錯誤，建議改用bigint(可表示任意長度的整數，只需在結尾加n)

- JavaScript 中的數字並沒有整數與浮點數的區分，這些寫法全都是 number 型別：

```
var number1 = 1;
var number2 = 0.1;
var number3 = -5;
```

```
console.log(9999999999999999); // 10000000000000000
console.log(9999999999999999 - 9999999999999998); // 2
```

```
console.log(9999999999999999n); // 9999999999999999n
console.log(9999999999999999n - 9999999999999998n); // 1n
```

```
console.log(typeof 9999999999999999n); // bigint
```

- 在進行運算上單純的使用 + - * / 即可

```
var num1 = 3;
var num2 = 2;

console.log(num1 + num2);
console.log(num1 * num2);
console.log(num1 / num2);
console.log(num1 - num2);
```

```
5
6
1.5
1
```

number又可分為常規數字與非常規數字，
非常規數字包括NaN, Infinity

```
var a = 1/0;
console.log(1/0); // Infinity
console.log(typeof a); // number
```

啥時會出現NaN呢？
例如：
var a = 'a';
console.log(Number(a));(意思是將a轉型為number)
→NaN

- number 型別還有一個特殊值「NaN」，也就是 not a number。

```
var notanumber = NaN;
```

# 基本型別 - 混合運算

計算次方(用**表示)：
console.log(2 ** 3); // 8

- JavaScript 是弱型別語言，在做混合運算時會出現轉型上的問題。

  當使用 - * /進行處理，會先將兩邊運算元轉換為數值再進行運算，

  若有一方轉換失敗，就會得到 NaN。

```
1 - undefined
NaN
```

NaN的資料型別仍為number

```
'1' * '2'
2
```

- +亦為串接運算子，如果兩邊有其中一項是字串，就會變成字串的拼接：

```
1 + '1'
"11"
```

例如：
console.log('4' + undefined);
→4undefined

console.log(4 + undefined);
→NaN

# 小試身手

```
console.log(1 + 2 * 3); // 7
console.log((1 + 2) / 3); // 1
console.log(2 - '1'); 1
console.log(2 + '1'); 21
console.log(2 * 's'); NaN
```

# 基本型別 - boolean

- boolean 的宣告，內容值只有分成 'true' 及 'false' :

```
var bool1 = true;
var bool2 = false;

console.log(bool1);
console.log(bool2);
```

```
true
```

```
false
```

# 基本型別 - null、undefined

- null 與 undefined 的宣告，直接寫 null 及 undefined 即可：

```
var test1 = null;
var test2 = undefined;

console.log(test1);
console.log(test2);
```

```
null
```

```
undefined
```

- 一個變數如果沒有給予初始值，預設上會給予 undefined。

```
var defaultVal;
console.log(defaultVal);
```

```
undefined
```

# 小試身手

```
var x;
var y = x;
console.log(y);
```

undefined

# 基本型別 - false value

- 在轉型上，JavaScript 裡有五個值轉型成 boolean 會是 false，其他都是 true，
  所以有人會稱他們為 false family：false、 0、NaN、''、null、undefined

注意：只有數字0是false，像'0'就是true

```
console.log(Boolean(0));
console.log(Boolean(NaN));
console.log(Boolean(''));
console.log(Boolean(null));
console.log(Boolean(undefined));
```

```
false
false
false
false
false
```

# Hoisting

- 當在宣告出一個變數前就先使用它，在 JavaScript 中是合法的：

```
console.log(a);

var a = 1;
undefined
```

- 可以看到在 console.log(…) 後才宣告出變數 a，而 JavaScript 並不會拿到 ReferenceError，而是會拿到 undefined，原因在於 Hoisting 的效果，這段程式碼對於 JavaScript 來說就像如此，所有使用 var 宣告的變數，都會被優先拉到最上頭，然而賦值的順序並不會更改：

```
var a;

console.log(a);

a = 1;
```

注意：這邊只有把宣告給拉上去，賦值並沒有拉上去

# 基本型別 - 邏輯運算

- JavaScript 在做邏輯運算 &&（and）及 ||（or）時，會先將各個變數轉換成 boolean 值，按照 and 及 or 的運算回傳最後比較出來的變數：

```
var x = null;
var y = 'test';
var z = 99;

console.log(x || y || z);
console.log(x && y && z);
```

```
test
```

```
null
```

補充：

```
var x = true;
var y = 'test';
var z = 99;
console.log(x || y || z);
console.log(x && y && z);

→true
→99
```

# 比較運算子

== 和 === 在比較物件時行為是一樣的

- 在 JavaScript 中有兩個相等運算子 == 與 ===，都可以判斷值或物件參考是否相同

- == ：兩邊轉換為同一型別，再比較是否相等。 equal
- === ：只要兩邊型態不一，就會判斷為 false。 strict equal

字符串之間也可進行比較，根據該字符在unicode編碼中所對應的數字進行比較(逐字符比較)：
console.log('a'.charCodeAt(0)); // 97
console.log('b'.charCodeAt(0)); // 98
console.log('a' > 'b'); // false

```
console.log('1' == 1);
console.log('1' === 1);
```

true

false

若數字想與字符串進行比較，會先將字符串轉為數字再進行比較，假設字符串為'10'，就好比先執行Number('10')->10，之後再進行比較，若轉換失敗則會變成NaN

- 不相等運算子：!= 與 !==

```
var obj1 = { a: 1 };
var obj2 = { a: 1 };

obj1 === obj2;  // console.log: false
```

你可以觀察到，就算 obj1 和 obj2 的屬性名稱與值都相同，但互相比較的結果卻是 false。

這是因為每個物件都是獨立存在的實體，兩者的記憶體位置並不相同。在比較 物件型別 時，比較的是記憶體位置，而非值。

# 流程控制 - if-else

- if - else 是基本的流程控制程式碼，在小括號內部放入會回傳 boolean 值的運算 ( 或是直接放入變數，讓它自行幫忙轉型成 boolean )，JavaScript 會優先執行第一個判斷式為 true 的 block :

```
var score = 75;

if(score >= 90){
    console.log('優');
}else if(score < 90 && score >= 80){
    console.log('甲');
}else if(score < 80 && score >= 70){
    console.log('乙');
}else if(score < 70 && score >= 60){
    console.log('丙');
}else{
    console.log('不及格');
}
```

```
乙
```

- if 可以單獨存在。
- else 非必要。

# 小試身手

- 在判斷式中，所有東西都可以轉換為布林值。

```
var value = '0';
if(value){
    console.log('true');
}else{
    console.log('false');
}
```

```
true
```

# 流程控制 – for 迴圈

- for 迴圈則會進行計數式的事情，小括號內會放入三段敘述句：

  第一段為起始條件、

  第二段為繼續執行條件（若為 true 時則會繼續執行以下的 coding block）、

  第三段為每次執行後都會做的事情

```
for(var i = 0;i < 3;++i){
    console.log(i);
}
```

```
0
1
2
```

# 流程控制 – for 迴圈

- 而除此之外，還有 break 與 continue 可以使用，它們分別代表直接終止迴圈以及跳過當次的迴圈：

```
for(var i = 0;i < 3;++i){
    if(i == 1){
        break;
    }
    console.log(i);
}
```

```
0
```

```
for(var i = 0;i < 3;++i){
    if(i == 1){
        continue;
    }
    console.log(i);
}
```

```
0
2
```

# 流程控制 - while 迴圈

- while 迴圈也是執行重複性的事情，小括號內部只要為 true 就會執行裡面的 block，
  不過與 for 迴圈相比偏向單純的條件式而不是計數式：

```
var randomNum = 0;
while(randomNum < 0.8){
    randomNum = Math.random();
    console.log(randomNum);
}
```

```
0.04679780058830340 6
```

```
0.90522015703627 7
```

Math.random()
→會回傳一個偽隨機小數 (pseudo-random) 介於 0 到 1 之間(包含 0，不包含 1)，大致符合數學與統計上的均勻分佈 (uniform distribution)

- 使用上也有 break 與 continue 的語法可以使用，概念上跟 for 迴圈是完全一樣的。

# 練習 1 - 1

- 分別用 for loop 與 while 印出數字 1~9

```
for (var i = 1; i <= 9; i++) {
    console.log(i);
}
```

# 練習 1 - 2

- 印出 1~100 中是 3 但不是 5 的倍數

```
for (var i = 1; i <= 100; i++) {
    if (i % 3 === 0 && i % 5 !== 0) {
        console.log(i);
    }
}
```

# 物件

# 物件

- 在 JavaScript 中只要不是基本型別的就會是物件，

  無論是再來要提到的陣列或是函式，它們都是物件的一種。

神奇的地方是：基本型別也會「暫時被包裝成物件」，用以方便使用方法

例如：
let str = "hello";
console.log(str.length); // 5 !

- 而一般物件的宣告形式如下：

```
var obj1 = {};
var obj2 = new Object();
```

- 在像 Java 這樣的強型別語言中，通常都會使用 new 關鍵字來創造物件（上面第二種），

  不過 JavaScript 提供了 object literal（上面第一種）的寫法，讓創造物件在語法上更為方便。

# 物件

- 物件其實是個屬性與值的群集（Collection），而結構看起來就像下面的形式：

| Key | Value |
| --- | --- |
| Name | Jack |
| Age | 20 |
| Role | PG |

- 它會有一組一組的 **key - value pair**，類似表格每列都有欄位和其對應的值，
  而值的部分可以放入各種型別的值（字串、數字、布林值...）

```
var data = {
    Name:'Jack',
    Age:20,
    Role:'PG'
}
```

- Key 會自動視為字串。
- Key 重複的話，value 會後蓋前。

# 物件

輕量級資料交換格式。其內容由屬性和值所組成，因此也有易於閱讀和處理的優勢。

- 而物件也可以放到物件裡面，也就出現了所謂的JSON資料結構：

```
var jsonObj = {
    key1: 'key1',
    key2: 2,
    key3: true,
    key4: {
        key41: 'key41',
        key42: 3
    }
}
```

- 取值語法上有.及[]可以使用：

```
console.log(jsonObj.key1);
```

key1 這邊原本印錯

```
console.log(jsonObj['key2']);
```

2 這邊原本印錯

  - 差別在使用 [] 內是可以允許空白、包含.、接受變數

但須注意的是[]這種取法，裡面放的必須是字串

# 物件

注意：物件沒有for...of的語法

- 若想迭代出內部的 key - value 值，可以使用 for ... in 語法：

```
var obj = {key1:123, key2:true};
for(var key in obj){
    console.log(key + ':' + obj[key]);
}
```

```
key1:123
key2:true
```

注意：for ... in 物件的語法，拿到的是物件的key

- 除了一開始就選擇將值設定好，也可以後續再補上，設值語法一樣可以使用 . 及 [] ：

```
var obj = {};

obj['key1'] = 123;
obj.key2 = true;

console.log(obj);
```

```
▶ {key1: 123, key2: true}
```

補充：
判斷一個key是否在obj內可直接用key in obj做判斷，回傳布林值

# 小試身手

注意：JavaScript 物件裡的 key 如果不是字串或中括號包起來的變數，會直接當成字面量名稱。

```
var key1 = 'test';
var obj = {'key':key1, key1:'key'};
for(var tempKey in obj){
    console.log(tempKey + ':' + obj[tempKey]);
}
```

```
key:test
key1:key
```

```
var key = 'test';
var obj = {'key':key, key:'key'};
for(var tempKey in obj){
    console.log(tempKey + ':' + obj[tempKey]);
}
```

```
key:key
```

# 陣列

- 陣列的宣告語法如下，也是一樣分成 Array literal 以及 new 的寫法：

```
var arr1 = [];
var arr2 = new Array();
```

- 結構看起來就像：


- 若想要一開始宣告時就給予值，可以用下面的方法來宣告，跟物件的形式基本上類似：

```
var arr = [1,2,3];
console.log(arr);
▶ (3) [1, 2, 3]
```

# 陣列

- 陣列也是物件的一種：


[1,2,3]

| Key | Value |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |

{0:1, 1:2, 2:3, length:3}

- 陣列的 Key 就是索引值 index
- 每個陣列都有一個屬性 length

# 陣列

- 而若想塞值到Array中，可以一個個進行push，或是使用屬性設定的方式：

index放字串會自動轉為數字(但不建議這麼做)

```
var arr = [];

arr.push(0);
arr['1'] = true;
arr[2] = '123';
console.log(arr);
▶ (3) [0, true, "123"]
```

- 如果中間有空值，預設上會給予undefined：

```
var arr = [];
arr[1] = true;

console.log(arr[0]);
undefined
```

# 陣列

- 迭代陣列的內容時，通常會使用一般的 for 迴圈進行迭代，
  而除此之外還有另一種 for...of 的寫法，兩種都可以達到目的：

```
var arr = [0, 1, 2];
for(var i = 0; i < arr.length; i++){
    console.log(arr[i]);
}

for(var val of arr){
    console.log(val);
}
```

```
0
1
2
0
1
2
```

陣列亦可以用for in的寫法，但須注意的是取出來的為index而非value

```
for (var idx in arr) {
    console.log(arr[idx]);
}
```

# 小試身手

```
var array1 = [];
var array2 = array1;
array2.push('123');
console.log(array1.length);
```

1

# 練習 2 - 1

- 給定以下資訊，請建立一個包含這些資訊的物件

| key | 值 | 說明 |
|---|---|---|
| name | 幸運兒 | 姓名 |
| age | 20 | 年齡 |
| lottory | 1, 5, 29, 41, 37, 15 | 樂透號碼 |

```
var obj = {



};
```

```
var obj = { name: "幸運兒" };
obj["age"] = 20;
obj.lottery = [1, 5, 29, 41, 37, 15];

console.log(obj);
```

# 練習 2 - 2

- 給定一個物件紀錄全班原本的成績　var origin = {'王大明': 89,　'張小美': 97,　'李鐵柱': 18,　'謝小天': 22};

- 老師不小心改錯考卷，最後決定每個人加5分，滿分100，印出加分後每個人的結果：

印出結果：

```
王大明：94
張小美：100
李鐵柱：23
謝小天：27
```

```
var origin = {
    王大明: 89,
    張小美: 97,
    李鐵柱: 18,
    謝小天: 22,
};

for (var key in origin) {
    origin[key] = (origin[key] <= 95) ? (origin[key] + 5) : 100;
}

console.log(origin);
```

# 函式

# 函式

- 對於要重複執行的內容，可以使用 function 定義函式。

- 函式也是一種物件，宣告上的形式：

```
function func(){
    return 'test';
}
```

```
function cal(a,b,c){
    return a + b - c;
}
```

  - 可以利用 return; 語法終止函式。

- 若要執行函式，要在函式的名稱後面加上 ()，如果沒有小括號，就只是單純的取得物件的內容：

```
console.log(func);
console.log(func());
ƒ func(){
    return 'test';
}
test
```

```
console.log(cal(3,2,1));
4
```

  - func 加上 () 運算子才表示執行函式。

# 函式

- 對於 JavaScript 來說，函式名稱只要相同就會被覆寫，無關乎參數的數量：

```
function func(a, b){
    console.log('two parameter');
}

function func(a){
    console.log('one parameter');
}

func(10, 20);
```

```
one parameter
```

補充：函式可以給傳入參數一個預設值，在用戶沒有傳入值時就會將預設值拿來用

```
function SayHi(user = 'Visitor'){
   console.log(`Hi, ${user}`);
}

SayHi('Steven'); // Hi, Steven
SayHi(); // Hi, Visitor
```

- 可以看到這邊即使多傳入參數，JavaScript 在解析上也沒有問題。

  如果少傳入參數 JavaScript 會給予 undefined。 少傳入的那個參數等於傳了一個undefined

# 函式

- 函式宣告（Function Declaration）：

  function func(){}

  優先載入

```
func1('1');

function func1(str){
  console.log('func1='+str);
}
```

func1=1

- 函式運算式（Function Expressions）：

  var func = function(){};

  執行到才會載入

```
func2('1');

var func2 = function(str){
  console.log('func2='+str);
};
```

func2 is not a function

因為使用var來宣告變數，根據hoisting，等於在最前面宣告了var func2，所以才會沒跑出is not defined這種結果

# 小試身手

```
var func = function (str){
  console.log('func = '+str);
};

function func(str, str2){
  console.log('func override = '+str);
};

func('1');
```

func = 1

# 檢查型別 - typeof

return的資料型態是string

- JavaScript 是弱型別語言，因此在檢查型別上有其他語法可以幫忙做到，其中一個是 typeof，它可以檢查出七種型別：number、string、boolean、undefined、symbol、object、function，其他像是 array 是檢查不出來的：

```
var a = 1;
var b = '1';
var c = true;
var d = undefined;
var e = Symbol(1);
var f = {};
var g = function(){};

var h = null;
```

```
console.log(typeof a);
console.log(typeof b);
console.log(typeof c);
console.log(typeof d);
console.log(typeof e);
console.log(typeof f);
console.log(typeof g);

console.log(typeof h);
```

```
number
string
boolean
undefined
symbol
object
function
object
```

- 這邊將所有的例子全部條列出來，前面七種沒有太大的問題，而第八個 null 這邊條列出來的是 object，是個較特別的部分。

```
var i = [1, 2, 3];
console.log(typeof i);
→object
```

# 檢查型別 - instanceof

- 另一種檢查型別的方式是用 instanceof：

```
var obj = {};
var arr = [];

console.log(obj instanceof Object);
console.log(arr instanceof Array);
console.log(arr instanceof Object);
```

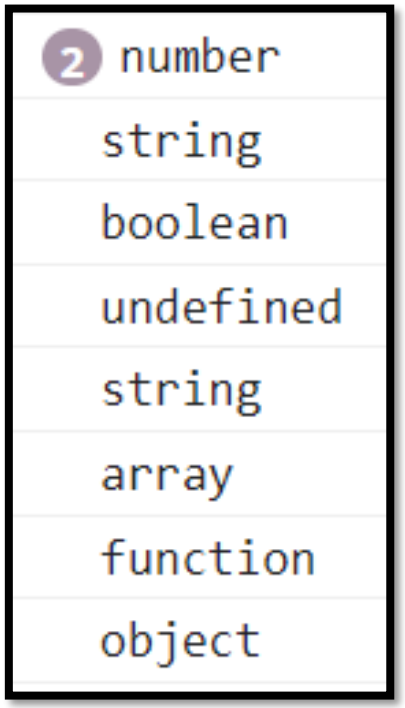
```
true
true
true
```

- 可以看到因為 array 也是物件的一種，因此 arr instanceof Object 也會拿到 true。

# 練習 3 - 1

- 判斷並印出陣列內的物件型別，如果不是基本型別，需印出是 Object 還是 Array

```
var arr = [6, 5, '草莓大福', false, undefined, '抹茶拿鐵', [], function(){}, {}];
```




```
for (var value of arr) {
    if (value instanceof Array) {
        console.log("array");
    } else {
        console.log(typeof value);
    }
}
```

```
for (var value of arr) {
    console.log(value instanceof Array ? "array" : typeof value);
}
```

# 範圍

# Scope

- 在使用 var 宣告時是有個基本 Scope 的，而這個基本 Scope 是 function，一般的 if 、for 並不能達到切分的效果：

```
for(var i = 0;i < 3;++i){

}
console.log(i);
```

```
3
```

```
function func(){
    var j = 'test';
    console.log(j);
}
func();
console.log(j);
```

```
test
▶Uncaught ReferenceError: j is not defined
    at <anonymous>:6:13
```

如果把var拿掉的結果

```
function func(){
   j = 'test';
   console.log(j);
}
func();
console.log(j);

→test
→test
```

# this

- JavaScript 中的 this 所代表的物件，然而究竟是指向哪個物件，是依賴於呼叫的當下：

```
var obj = {
    func: func,
    test: 'test'
};

function func(){
    console.log(this);
    console.log(this.test);
}

func();
obj.func();
```

```
▶ Window {window: Window, self: Win
undefined
▶ {test: "test", func: f}
test
```

- this 對於物件的綁定大致上可以分成四種，這邊會介紹較為簡單的兩種綁定，一種是在呼叫時前面沒有加上任何的物件，這時預設上會指向全域物件 window，另一種則是看前面是由哪個物件呼叫，那就會指向哪個物件。

# 區域變數 v.s. 全域變數

- 區域變數：
  - 在函式中透過 var 宣告的變數
  - 無法在函式外其他地方調用
  - 當函式結束工作後，變數即失效

- 全域變數：
  - 函式中未使用 var 宣告的變數、或函式外無論有無用 var 宣告的變數
  - 表示為 window 的屬性，等同於 window.???
  - 可於整個網頁範圍內調用
  - 網頁關閉時，變數亦失效。

- 如果全域與區域中有同名的變數，則區域會暫時覆蓋全域。

```
function func() {
    x = 10;
    var y = 20;
}

func();

console.log(x);
console.log(y);
```

```
10
y is not defined
```

# 小試身手

```
var a = 10;
if (a < 100){
    var b = 20;
}
console.log(b);
```

20

```
var func = function(){
    var str1 = 'funcString';
    var func2 = function(){
        console.log('str1 = ' + str1);
    };
    func2();
};
func();
```

str1 = funcString

# 小試身手

```
str = 'string';

var func = function(){
    var str = 'funcString';
    console.log('func get window.str = ' + window.str);
    console.log('func get str = ' + str);
};

console.log('before = ' + str);
func();
console.log('after = ' + str);
```

```
func get window.str = string
func get str = funcString
before = string
after = string
```

# 函式延伸

# 函式

- 另外在 JavaScript 中會稱函式為 first-class function，

  原因在於函式與一般物件並無相異，可以當作參數傳入到另一個 function，也可以被當作回傳值傳出來：

```
function testFunc(){
    return function(){
        console.log('sayHello');
    }
}

var sayHello = testFunc();
sayHello();
```

```
sayHello
```

# 回呼函式 Callback function

- 所謂的回呼函式其實就是＂將函式當作另一個函式的參數＂。
- 用來維持函式的功能專一，能夠讓使用者廣泛的重複使用。

```
var exchange = function(NT, CCY){
    switch (CCY) {
        case 'USD':
            return NT / 30;

        case 'CNY':
            return NT / 5;

        default:
            alert('該幣別無法交易');
    }
};
exchange(3000, 'USD'); //100
exchange(100, 'CNY'); //20
```

V.S.

```
var exchange = function(NT, callback){
    return callback(NT);
}

var USD = function(NT){
    return NT / 30;
};

var CNY = function(NT){
    return NT / 5;
};

exchange(3000, USD); //100
exchange(100, CNY); //20
```

# 回呼函式 Callback function

- 學校提供處理考試結果的通用方法

```
var testResult = function(grade, passAction, unPassAction){
    if(grade >= 60){
        passAction();
    }else{
        unPassAction();
    }
};
```

- 數學老師決定

```
testResult(mathGrade
    , function(){
        // 簡訊飲料兌換卷
    }, function(){
        // 寄 mail 通知家長
});
```

# new

- JavaScript 中的 new 關鍵字是使用 function 來創造一個新的物件，
  本質上還是執行 function 中的內容，而 new 在 function 中會做幾件事情：
  - 創造一個無中生有的物件 {}
  - 將this 指向此物件(JavaScript中的第三種this綁定)
  - 將此物件回傳出去

在function沒有回傳值或回傳值不是物件的情況下才會這麼做，否則只會回傳return後的那個物件

```
var a;
var func1 = function(){
    a = 0;
};

console.log('a->' + a);
```

undefined

```
var b;
var func2 = new function(){
    b = 0;
};

console.log('b->' + b);
```

0

左邊這段倒數第二句加上func1()，結果就會和右邊一樣

➢ function 也有點 Java 的 class 概念。

# new

- 看看以下兩段程式碼，為什麼第二段 Mary 的程式碼會有錯誤？

```
function Person(name, age){
    this.name = name;
    this.age = age;
}

var Jason = new Person('Jason', 25);
console.log(Jason.name);
console.log(Jason.age);
```

```
Jason
25
```

```
var Mary = Person('Mary', 24);
console.log(Mary.name);
console.log(Mary.age);
```

退回最預設的this綁定，因此此時，window.age = 24

```
▶Uncaught TypeError: Cannot read property 'name' of undefined
    at <anonymous>:2:18
```

# new

- 比較下面幾段程式碼的回傳值，並想看看會有怎樣的結果：

```
function Person1(name, age){
    this.name = name;
    this.age = age;
}

function Person2(name, age){
    this.name = name;
    this.age = age;
    return 1;
}

function Person3(name, age){
    this.name = name;
    this.age = age;
    return {};
}
```

```
var person1 = new Person1('Jason', 25);
var person2 = new Person2('Jason', 25);
var person3 = new Person3('Jason', 25);
```

各person裡儲存的值
person1
→Person1 {name: 'Jason', age: 25}

person2
→Person2 {name: 'Jason', age: 25}

person3
→{}

# 函式

閉包的概念

- 用 function 來模擬一個 Java 的 ArrayList 類別

```
var datas = new ArrayList();
datas.add('AAA');
datas.get(0); // AAA
```

```
var ArrayList = function(){
    var list = [];

    var add = function(data){
        list.push(data);
    };
    var get = function(index){
        return list[index];
    };

    var functions = {
        'add':add,
        'get':get
    };

    return functions;
}
```

```
var ArrayList = function(){
    var list = [];

    return {
        'add': function(data){
            list.push(data);
        },
        'get': function(index){
            return list[index];
        }
    };
};
```

優化後寫法

# 課後練習

# 課後練習 1：HashMap

1. 編輯 HashMap.js
   - 模擬 Java  HashMap 類別
   - 提供 put、keys、contains、get、clear 等方法。

2. practice1.html 無須異動，打開就顯示如下畫面：

```
contains key1 : true
get key1 : test1
contains key4 : false
keys length before clear : 2 [ key1; key2; ]
keys length after clear : 0
```

```
var HashMap = function() {
    var obj = {};

    return {
        put: function(key, value) {
            obj[key] = value;
        },
        keys: function() {
            arr = [];
            for(var k in obj){
                arr.push(k);
            }
            return arr;
        },
        contains: function(key) {
            return key in obj;
        },
        get: function(key) {
            return obj[key];
        },
        clear: function() {
            obj = {};
        }
    };
};
```

# 課後練習 2：practice2.html

```
(Object){
    firstName   =   (string) Dark
    lastName    =   (string) Liu
    sex     =   (string) M
    weight    =   (number) 80
    married    =   (boolean) true
    workExp    =   (Array)[
      0   =   (Object){
        name    =   (string) cathay life insurence
        during    =   (number) 8
      }
      1   =   (Object){
        name    =   (string) cathay united bank
        during    =   (number) 1
      }
    ]
    skill    =   (Array)[
      0   =   (string) java
      1   =   (string) SQL
      2   =   (string) jsp
      3   =   (string) javascript
    ]
  }
```

- 試著解析存在頁面上的 bigObj 物件，依下列指示輸出。
  - 若讀取到 String, Number, Boolean 請輸出並換行：
    (type) value
  - 若讀取到 Object 請輸出並換行：
    {
    key = value,
    key = value
    }
  - 若讀取到 Array 請輸出並換行：
    [
    0 = value,
    1 = value
    ]
- Hint:
  - 輸出 <br>來換行
  - 使用 function 遞迴呼叫來解析。

# Thank You