Javascript Basics

by Fin Chen

Javascript

- "JavaScript" is an ECMAScript variant managed by Mozilla. All non-Mozilla implementations using JavaScript are actually implementing ECMAScript, rather than Javascript. "JavaScript" support generally is meant to describe support for ECMA-262, edition 3, though some—notably Chrome, but also Opera 12 and Internet Explorer 10—target ECMA-262, edition 5.

Javascript 基礎

- 結構
  - 變數、型別、值
  - 運算式、運算子
  - 流程控制
  - 物件
- 陣列
  - 函式
  - 正規表示法
  - 類別與模組

Javascript 基礎

```
#建立專案目錄並以sublime text2開啟

~$ mkdir myFirstApp
~$ e
```

```
#hello.js

var name = "Fin";
console.log("Hi, I am " + name);
```

```
#執行hello.js

~$ node hello
~$ nodemon hello.js
```

- 練習： 寫一個顯示你自己名字的程式

Javascript 結構

- 大小寫不同
- 註解：
  - //單行註解
  - /* 多行註解 */
  - 善用註解
- 分號

Variable 變數

引號

分號結尾

```
var name = "Fin";
```

宣告

變數名稱

變數值

Variable 變數

```
var name = "Fin";
name = name + " Chen";
console.log(name);
```

Variable 變數

- 變數可以不用宣告直接用，直譯器會自動幫你宣告
- 但會遇到很多狀況，所以還是先宣告再使用
- 範例：

```
//noname.js

console.log(name);
```

- 命名原則：camelCase, 簡單清楚

Type 型別

- Number
- String
- Boolean
- null & undefined

- Array
- Object
- Function
- RegExp

Number 數字

- 整數： 0, 3, 100
- 16進位： 0xCAFE911, 0xff
- 8進位： 0377
- 浮點數： 3.14, .33333, 6.02e23, 1.478e-32

Number 操作

- Math 提供許多常用的運算函式

```
Math.pow(2,53) // => 9007199254740992: 2 to the power 53
Math.round(.6) // => 1.0: round to the nearest integer
Math.ceil(.6) // => 1.0: round up to an integer
Math.floor(.6) // => 0.0: round down to an integer
Math.abs(-5) // => 5: absolute value
Math.max(x,y,z) // Return the largest argument
Math.min(x,y,z) // Return the smallest argument
Math.random() // Pseudo-random number x where 0 <= x < 1.0
Math.PI // π: circumference of a circle / diameter
Math.E // e: The base of the natural logarithm
Math.sqrt(3) // The square root of 3
Math.pow(3, 1/3) // The cube root of 3
Math.sin(0) // Trigonometry: also Math.cos, Math.atan, etc.
Math.log(10) // Natural logarithm of 10
Math.log(100)/Math.LN10 // Base 10 logarithm of 100
Math.log(512)/Math.LN2 // Base 2 logarithm of 512
Math.exp(3) // Math.E cubed
```

String 字串

- 用雙引號或單引號
- 反斜線為逸出字元（escaping character）
- 合法的字串：
  - "", 'TESTING', "3.14"
  - "name='fin'", "第一行\n第二行"

String 字串

- 常用逸出字元
  - \t: tab
  - \n: 換行
  - \", \': 引號
  - \\: 反斜線
  - \uXXXX: unicode字元

String Methods

- charAt
- concat
- contains
- indexOf, lastIndexOf
- slice, substr, substring
- split
- match, replace, search >> RegExp

String 範例

```
var s = "hello, world" // Start with some text.
s.charAt(0) // => "h": the first character.
s.charAt(s.length-1) // => "d": the last character.
s.substring(1,4) // => "ell": the 2nd, 3rd and 4th characters.
s.slice(1,4) // => "ell": same thing
s.slice(-3) // => "rld": last 3 characters
s.indexOf("l") // => 2: position of first letter l.
s.lastIndexOf("l") // => 10: position of last letter l.
s.indexOf("l", 3) // => 3: position of first "l" at or after 3
s.split(", ") // => ["hello", "world"] split into substrings
s.replace("h", "H") // => "Hello, world": replaces all instances
s.toUpperCase() // => "HELLO, WORLD"
```

String 範例
(ECMAScript 5)

- ECMAScript 5中，可以把字串當成陣列來操作

```
s = "hello, world";
s[0] // => "h"
s[s.length-1] // => "d"
```

String

- 練習：
  一函式wordCounter(str)，回傳值為此str裡面有多少個以空白區分的字。範例： "Hi, I'm 32 years old and weight 120 lbs" >> 9
- tips: array.length可以取得陣列大小
- ```
  function wordCounter(str) {
      var result;
      return result;
  }
  wordCounter()
  ```
  ```

Boolean

- 通常是比較後的結果
- 或是用在判斷式中
  - 會被判別為false的值
    - false, 0, NaN, "", undefined, null
  - 其他都是true
  - ! 為not: !false === true

Boolean

```
var myName = 'Fin';
var hisName = 'Ben';
var equal = myName === hisName;
console.log("Is " + myName + " and " + hisName + " equal? " + equal);

var emptyArray = [];
var emptyObject = {};
var emptyString = "";
console.log(!!emptyArray, !!emptyObject, !!emptyString);
```

雙重否定轉換為boolean

null & undefined

- null: 變數值為空
- undefined: 變數未被宣告 or 無此值 or 尚未初始化
  - null == undefined == false
  - null !== undefined
  - typeof null === 'object'
  - typeof undefined === 'undefined'

null & undefined

```
var noValue;
var someObject = {};
console.log(noValue, typeof someObject.prop !== 'undefined');
```

Type Conversion

- 內建的Type Conversion常導致無法預期的結果
- 盡量使用 ===/!== 而非 ==/!=

Variable Scope

- Javascript的變數範圍是以function來界定

```
var scope = "global"; // Declare a global variable
function checkscope() {
    var scope = "local"; // Declare a local variable with the same name
    return scope; // Return the local value, not the global one
}

console.log(checkscope()); // => "local"
```

Variable Scope

- 變數的尋找是由內而外

```
scope = "global"; // 編譯器會自動宣告變數
function checkscope() {
    scope = "local"; // local端沒有此變數，往上一層找
    myscope = "local"; // 沒有宣告的話會變成全域變數
    var myLocalScope = "local";
    return [scope, myscope]; // Return two values.
}

console.log(myLocalScope); //這是checkscope的local變數，看不到
console.log(scope, myscope); //因為兩個都是全域變數，可以直接取值
```

Variable Hoist

- 編譯器會把變數宣告拉到變數範圍的最前面，好讓整個變數範圍都知道有這個變數存在
- 所以....會有以下狀況

```
var scope = "global";
function f() {
    console.log(scope); // Prints "undefined", not "global"
    var scope = "local"; // Variable initialized here, but defined
everywhere
    console.log(scope); // Prints "local"
}
```

Variable Hoist

- 正確做法： 所有變數請一定在範圍的最前面作宣告

```
var scope = "global";
function f() {
   this.scope;
    var scope;
    console.log(scope); // 仍然會輸出undefined, 但從程式碼就看得出來
    scope = "local"; // 先宣告，之後再指派變數值
    console.log(scope); // Prints "local"
}
```

Expression 運算式

- 能夠讓直譯器理解並執行的程式片段

```
//primary expression
"hello" | 1.23
/pattern/
true | false | null | this
i | undefined

//object and array
var ary = [1,2,3];
var obj = {x:1, y:2};

//function
var square = function(x) { return x*x; }
```

Expression 運算式

```
//property access
var o = {x:1, y:{z:3}};
var a = [o,4,[5,6]];
o.y;
a[0].x;

//invocation expression
f(0);
Math.max(x,y,z);

//object creation
new Date();
new Point(2,3);
```

Operator 運算子

- 算術運算子(Arithmetic Operator)
- 指派運算子(Assignment Operator)
- 位元運算子(Bitwise Operator)
- 比較運算子(Comparison Operator)
- 邏輯運算子(Logical Operator)
- 字串運算子(String Operator)
- 特殊運算子(Special Operator)

Operator 優先順序

結合律	運算子
右	new
左	[] () .
右	++ -- ~ - +(正負號) ! delete typeof void
左	* / %
左	+ - (加減, 字串連接)
左	<< >> >>>
左	< <= > >= <>
左	== != === !==
左	&
左	^
左	|
左	&&
左	||
右	? :
右	= += -= *= /= %= &= |= ^= <<= >>= >>>=
左	,

Operator 算術運算子

範例	名稱	說明
+a	正號	實際上無作用
-a	負號	將a正負變號
a + b	加法	a和b的總和
a - b	減法	a減去b
a * b	乘法	a乘上b
a / b	除法	a除以b
a % b	取餘數	a除以b的餘數(餘數的正負號與a相同)
++a	前置增值	a先+1，再回傳
a++	後置增值	a先回傳，再+1
--a	前置減值	a先-1，再回傳
a--	後置減值	a先回傳，再-1

Operator 指派運算子

範例	說明
a += b	a = a + b
a -= b	a = a - b
a *= b	a = a * b
a /= b	a = a / b
a %= b	a = a % b
a <<= b	a = a << b
a >>= b	a = a >> b
a &= b	a = a & b
a |= b	a = a | b
a ^= b	a = a ^ b

Operator 位元運算子

範例	名稱	說明
a & b	交集(And)	a和b的位元皆為1的部份為1
a | b	聯集(Or)	a或b的位元其中一方為1的部份為1
a ^ b	互斥(Xor)	a和b的位元只有其中一方為1的部份為1
~ a	補數(Not)	a為1的部份為0，為0的部份為1
a << b	左移	a往左移動b個位元(意同a乘以2的b次方)
a >> b	帶正負號右移	a往右移動b個位元，並保留原始正負號(意同a除以2的b次方之整數)
a >>> b	補0右移	a往右移動b個位元，無視正負號，左邊的位元一律補0

Operator 比較運算子

範例	名稱	說明
a == b	相等	如果a和b的值相等則為true
a === b	完全相等	如果a和b的值與型別都相等則為true，物件、陣列和函式必須是同一個物件才為true
a != b	不相等	如果a和b的值不相等則為true
a !== b	不完全相等	如果a和b的值或型別不相等則為true
a < b	小於	如果a小於b則為true
a > b	大於	如果a大於b則為true
a <= b	小於等於	如果a小於等於b則為true
a >= b	大於等於	如果a大於等於b則為true

Operator +

- number + number: 加法運算
- string/object + any: any轉成字串後結合
- 其他: 轉換成數字再執行加法運算
- null: 數字: 0, 字串: null
- undefined: 數字: NaN, 字串: undefined
- NaN + any number = NaN

Operator +

```
1 + 2 // => 3: addition
"1" + "2" // => "12": concatenation
"1" + 2 // => "12": concatenation after number-to-string
1 + {} // => "1[object Object]": concatenation after object-to-string
true + true // => 2: addition after boolean-to-number
2 + null // => 2: addition after null converts to 0
2 + undefined // => NaN: addition after undefined converts to NaN
1 + 2 + " blind mice"; // => "3 blind mice"
1 + (2 + " blind mice"); // => "12 blind mice"
```

Operator

- 練習：簡化下列算是

```
 ((4 >= 6) || ("grass" != "green"))
&& !(((12 * 2) == 144) && true)
```

Statement

- 宣告
- 條件
- 迴圈
- 跳脫

Statement 宣告

- var

```
var i;
var greeting = "hello" + name;
var x = 2.34, y = Math.cos(0.75), r, theta; var x = 2, y = x*x;
var f = function(x) { return x*x }, y = f(x);

for(var i = 0; i < 10; i++) console.log(i);
for(var i = 0, j=10; i < 10; i++,j--) console.log(i*j);
for(var i in o) console.log(i);
```

- function

```
function hypotenuse(x, y) {
    return Math.sqrt(x*x + y*y); // return is documented in the next section
}

function factorial(n) { // A recursive function if (n <= 1) return 1;
    return n * factorial(n - 1);
}
```

Statement 條件

- if, else if
- switch

Statement 條件

- if

```
i = j = 1;
k = 2;
if (i == j)
    if (j == k)
        console.log("i equals k");
else
    console.log("i doesn't equal j"); // WRONG!!
```

- 如何修正?
- 建議：任何if,else後面都要接{}

Statement 條件

- else if

```
if (n == 1) {
    // Execute code block #1
}
else if (n == 2) {
    // Execute code block #2 }
else if (n == 3) {
    // Execute code block #3
}
else {
    // If all else fails, execute block #4
}
```

Statement 條件

- switch

```
switch(n) {
    case 1: // Start here if n == 1
        // Execute code block #1. break;
    case 2:
        // Execute code block #2. break;
    case 3:
        // Execute code block #3. break;
    default:
        // Execute code block #4. break;
}
```

- tip: 沒有任何條件滿足時才會執行default，default所在位置不影響switch判斷。

Statement 廻圈

- while
- do while
- for
- for/in

Statement 迴圈

- while

```
var count = 0;
while (count < 10) {
    console.log(count);
    count++;
}
```

Statement 迴圈

- do while

```
function printArray(a) {
    var len = a.length, i = 0;
    if (len == 0)
        console.log("Empty Array");
    else {
        do {
            console.log(a[i]);
        } while (++i < len); }
    }
}
```

- 確定此迴圈至少執行一次時才使用

Statement 迴圈

- for

```
for(var count = 0; count < 10; count++) console.log(count);
```

```
for(var i = 0, j = 10 ; i < 10 ; i++, j--)
    sum += i * j;
```

- for/in: 用在列舉物件屬性時

```
for(var p in o) // Assign property names of o to variable p
    ßconsole.log(o[p]); // Print the value of each property
```

```
var o = {x:1, y:2, z:3};
var a = [], i = 0;
for(a[i++] in o) /* empty */;
```

Statement 跳脫

- break
- continue
- return
- throw
- try/catch/finally

Statement 跳脫

- break

```
switch(n) {
    case 1: // Start here if n == 1
        // Execute code block #1.
        break;
    case 2:
        // Execute code block #2.
        break;
    case 3:
        // Execute code block #3.
        break;
    default:
        // Execute code block #4.
        break;
}
```

```
for(var i = 0; i < a.length; i++) {
    if (a[i] == target) break;
}
```

Statement 跳脫

- continue

```
for(i = 0; i < data.length; i++) {
    if (!data[i]) continue; // Can't proceed with undefined data
    total += data[i];
}
```

- return

```
function display_object(o) {
    if (!o) return;
    console.dir(o);
    return;
}
```

Statement 跳脫

- throw

```
function factorial(x) {
    // If the input argument is invalid, throw an exception!
    if (x < 0) throw new Error("x must not be negative");
    // Otherwise, compute a value and return normally
    for(var f = 1; x > 1; f *= x, x--) /* empty */ ;
    return f;
}
```

Statement 跳脫

- try/catch/finally

```
function factorial(x) {
    var f = 1;
    try {
        if (x < 0) throw new Error("x must not be negative");
        for(; x > 1; f *= x, x--) /* empty */ ;
        return f;
    }
    catch(e) {
        if(e instanceof Error) {
            console.log(e);
            return undefined;
        }
    }
    finally {
        console.log("Exit function");

    }
}
```

Statement Practice

- 練習：
  寫一函式，輸入一正整數，會由小到大列出所有小於此正整數的偶數，並以逗號間隔。非正整數以try/catch處理並顯示錯誤訊息。例如：
  getSmallerEvens(10)印出2,4,6,8，
  getSmallerEvens("aha")印出"argument error"

Statement 其他

- with
- debugger
- "use strict"

Object 物件

- 基本的物件宣告：
  - `var empty = {};`
  - `var point = { x:0, y:0 };`
- 物件式宣告：
  - `var a = new Object();`（不建議使用）
  - `var b = new Date();`
  - `var c = new RegExp("js");`

Object 物件屬性

```
//get properties from object

var book = {
    author: {firstname: "J.R.R.", lastname: "Tolkien"},
    "main title": "The Lord of the Rings"
};
var author = book.author; // Get the "author" property of the book.
var name = author.lastname; // Get the "surname" property of the author.
var title = book["main title"]; // Get the "main title" property of the book.
console.log(name, title);

//set properties of object
book["sub title"]= "The Fellowship of the Ring";
book.price = 131;
console.log(book["main title"], book["sub title"]);
```

Object 存取屬性

```
//get properties from object

var book = {
    author: {firstname: "J.R.R.", lastname: "Tolkien"},
    "main title": "The Lord of the Rings"
};

var subtitle = book.subtitle;

//error
var len = book.subtitle.length;

// A concise and idiomatic alternative to get subtitle length or undefined var
len = book && book.subtitle && book.subtitle.length;

console.log(subtitle, len);
```

Object delete

- 移除物件屬性
- 全域物件無法移除

```
//delete: remove props from an object

var book = {
    author: {firstname: "J.R.R.", lastname: "Tolkien"},
    "main title": "The Lord of the Rings"
};

delete book.author; // The book object now has no author property.
delete book["main title"]; // Now it doesn't have "main title", either.

console.dir(book);
```

Object 測試屬性

- in

```
var o = { x: 1 }
"x" in o; // true: o has an own property "x"
"y" in o; // false: o doesn't have a property "y"
"toString" in o; // true: o inherits a toString property
```

- hasOwnProperty

```
var o = { x: 1 }
o.hasOwnProperty("x"); // true: o has an own property x
o.hasOwnProperty("y"); // false: o doesn't have a property y
o.hasOwnProperty("toString"); // false: toString is an inherited property
```

- propertyIsEnumerable

```
var o = inherit({ y: 2 });
o.x = 1;
o.propertyIsEnumerable("x"); // true: o has an own enumerable property x
o.propertyIsEnumerable("y"); // false: y is inherited, not own
Object.prototype.propertyIsEnumerable("toString"); // false: not enumerable
```

Object 列舉屬性

- 列出此物件的所有屬性

```
// 在for loop裡的in只會列出enumerable的屬性
// ex: 前例的toString不會出現在loop中

for(var i in o) {
    if (foo.hasOwnProperty(i)) {
        console.log(i);
    }
}
```

```
// 列出物件的屬性，且非函式

for(var i in o) {
    if (foo.hasOwnProperty(i) && typeof o[i] !== 'function') {
        console.log(i);
    }
}
```

Object

- 練習：
  建立一friends物件，裡面包含兩個屬性bill & steve。兩個屬性皆為物件，各包含了自己的firstName(字串), lastName(字串), number(陣列)。

- 練習二：
  試著在friends底下加入一個turner，俱有與bill相似的屬性。並刪除steve。

Array 陣列

- 基本的陣列宣告：
  - `var myArray = [];`
  - `var myArray2 = [1, {}, true];`
- 物件式宣告：（不建議使用）
  - `var a = new Array();`
  - `var b = new Array(5);`
  - `var c = new Array(4, "1", true);`

Iterating Array

```
//針對一般陣列
for(var i = 0; i < a.length; i++) {
  if (!(i in a)) continue ; // 跳過未定義的index
  // loop body here
}
```

```
//針對sparse array
for(var index in sparseArray) {
  var value = sparseArray[index];
  // Now do something with index and value
}
```

Array Methods

- join
- reverse
- sort
- concat
- slice
- splice
- push, pop
- unshift, shift
- toString, toLocaleString

Array Methods

```
var a = [1, 2, 3];
a.join();
a.join(" ");
a.join("");

var b = new Array(10);
b.join('-')
```

```
var a = [1,2,3];
a.reverse().join() // => "3,2,1" and a is now [3,2,1]
```

Array Methods

```
//sort with alphabetical order
var a = new Array("banana", "cherry", "apple");
a.sort();
var s = a.join(", "); // s == "apple, banana, cherry"
```

```
var a = [1,2,3];
a.reverse().join() // => "3,2,1" and a is now [3,2,1]
```

```
var a = [33, 4, 1111, 222];
a.sort(); // Alphabetical order: 1111, 222, 33, 4
a.sort(function(a,b) { // Numerical order: 4, 33, 222, 1111
  return a-b; // Returns < 0, 0, or > 0, depending on order
});
a.sort(function(a,b) {return b-a}); // Reverse numerical order
```

Array Methods

```
var a = [1,2,3];
a.concat(4, 5)
a.concat([4,5]);
a.concat([4,5],[6,7])
a.concat(4, [5,[6,7]])
```

```
var a = [1,2,3,4,5];
a.slice(0,3); // Returns [1,2,3]
a.slice(3); // Returns [4,5]
a.slice(1,-1); // Returns [2,3,4]
a.slice(-3,-2); // Returns [3]
```

```
//slice可以拿來複製陣列
var a = [1,2,3,4,5];
var b = a.slice(0);
b[4] = 0;
console.log(a, b); //[1,2,3,4,5] [1,2,3,4,0]
```

Array Methods

```
var stack = [];  // stack: []
stack.push(1,2); // stack: [1,2] Returns 2
stack.pop(); // stack: [1] Returns 2
stack.push(3); // stack: [1,3] Returns 2
stack.pop(); // stack: [1] Returns 3
stack.push([4,5]); // stack: [1,[4,5]] Returns 2
stack.pop() // stack: [1] Returns [4,5]
stack.pop(); // stack: [] Returns 1
```

```
var a = []; // a:[]
a.unshift(1); // a:[1] Returns: 1
a.unshift(22); // a:[22,1] Returns: 2
a.shift(); // a:[1] Returns: 22
a.unshift(3,[4,5]); // a:[3,[4,5],1] Returns: 3
a.shift(); // a:[[4,5],1] Returns: 3
a.shift(); // a:[1] Returns: [4,5]
a.shift(); // a:[] Returns: 1
```

Array Methods
(ECMAScript 5)

- forEach()
- map()
- filter()
- every(), some()
- reduce(), reduceRight()
- indexOf(), lastIndexOf()

Array Methods
(ECMAScript 5)

```
// forEach()
// array iterator

var data = [1,2,3,4,5]; // An array to sum
// Compute the sum of the array elements
var sum = 0; // Start at 0
data.forEach(function(value) { sum += value; }); // Add each value to sum
sum // => 15

// Now increment each array element
data.forEach(function(v, i, a) { a[i] = v + 1; });
console.log(data); // => [2,3,4,5,6]
```

```
//map
//計算並產生一個新的陣列

a = [1, 2, 3];
b = a.map(function(x) { return x*x; }); // b is [1, 4, 9]
```

Array Methods
(ECMAScript 5)

```
//filter

a = [5, 4, 3, 2, 1];
smallvalues = a.filter(function(x) { return x < 3 }); // [2, 1]
everyother = a.filter(function(x,i) { return i%2==0 }); // [5, 3, 1]
```

```
//every
//檢查是否每個元素都符合條件

a = [1,2,3,4,5];
a.every(function(x) { return x < 10; }) // => true: all values < 10.
a.every(function(x) { return x % 2 === 0; }) // => false: not all values even.
```

```
//some
//檢查是否有元素符合條件

a = [1,2,3,4,5];
a.some(function(x) { return x%2===0; }) // => true a has some even numbers.
a.some(isNaN) // => false: a has no non-numbers.
```

Array Methods
(ECMAScript 5)

```
//array.reduce(callback, [initialValue])
//callback = function(previousValue, currentValue, index, array)

var a = [1,2,3,4,5]
var sum = a.reduce(function(x,y) { return x+y }, 0); // Sum of values
var product = a.reduce(function(x,y) { return x*y }, 1); // Product of values
var max = a.reduce(function(x,y) { return (x>y)?x:y; }); // Largest value
```

```
//reduceRight()

var a = [2, 3, 4]
// Compute 2^(3^4). Exponentiation has right-to-left precedence

var big = a.reduceRight(function(accumulator,value) {
                                return Math.pow(value,accumulator);
                            });
```

- 練習：使用reduce把陣列[1,2,3,4,5]變成字串12345

Array Methods
(ECMAScript 5)

```
//array.lastIndexOf(searchElement[, fromIndex])

a = [0,1,2,1,0];
a.indexOf(1) // => 1: a[1] is 1
a.lastIndexOf(1) // => 3: a[3] is 1
a.indexOf(3) // => -1: no element has value 3
```

```
// Find all occurrences of a value x in an array a and return an array
// of matching indexes

function findall(a, x) {
    var results = [], // The array of indexes we'll return
        len = a.length, // The length of the array to be searched
        pos = 0; // The position to search from
    while(pos < len) { // While more elements to search...
        pos = a.indexOf(x, pos); // Search
        if (pos === -1) break; // If nothing found, we're done.
        results.push(pos); // Otherwise, store index in array
        pos = pos + 1; // And start next search at next element
    }
    return results; // Return array of indexes
}
```

Array Practice

- 寫一個函式，接收三個參數 between(str, start, end)
  str為一字串，start為開始字串，end為結束字串；回傳為start與end中間的值。舉例: between("hola [hello] bon", "[", "]") 應回傳 "hello"
- tips: 使用String.indexOf以及Array.slice

Function 函式

- 函式是物件的一種
- 輸入參數(argument) > 計算(context) > 輸出結果(return)

Function 宣告

- 基本函式宣告：

```
function factorial(x) {
    if (x <= 1) return 1;
    return x * factorial(x-1);
}
```

- 需注意函式名稱也會被hoist

Function 呼叫

- 直接呼叫

```
var probability = factorial(5)/factorial(13);

var strict = (function() { return !this; }());
```

```
var calculator = { // An object literal
    operand1: 1,
    operand2: 1,
    add: function() {
        // Note the use of the this keyword to refer to this object.
        this.result = this.operand1 + this.operand2; }
};
calculator.add(); // A method invocation to compute 1+1.
calculator.result // => 2
```

Function 呼叫

- 建構式

```
var probability = factorial(5)/factorial(13);

var strict = (function() { return !this; }());
```

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

var bob = new Person("Bob Dylan", 72);
bob.name; //"Bob Dylan
bob.age; //72
```

Function 呼叫

- call, apply

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    this.greet = function(greeter) {
        return "Hi, " + greeter + ". I'm " + this.name;
    }
};

var bob = new Person("Bob Dylan", 72);
var hans = new Person("Hans Zimmer", 55);
bob.greet.call(hans, "Fin");
hans.greet.apply(bob, ["Fin"]);
```

Function 參數

- 參數可以是任意數，用arguments來取

```
function max(/* ... */) {
    var max = Number.NEGATIVE_INFINITY;
    // Loop through the arguments, looking for, and remembering, the biggest.
    for(var i = 0; i < arguments.length; i++)
        if (arguments[i] > max) max = arguments[i];
    // Return the biggest
    return max;
}
var largest = max(1, 10, 100, 2, 3, 1000, 4, 5, 10000, 6); // => 10000
```

Function 參數

- 常用pattern

```
function newObj(arg, option) {
    var defaultOption = {
        option1: value1,
        option2: value2,
        option3: value3
    }

    //combine defaultOption and option

    //use arg and option to run newObj
}
```

Function 調用

- 宣告為匿名函式並執行，不占用namespace

```
(function(x,y) { // mymodule function rewritten as an unnamed expression
    // Module code goes here.
}(1,2)); // end the function literal and invoke it now.
```

- bind

```
function f(y) { return this.x + y; };
var o = { x : 1 };
var g = f.bind(o);
g(2);
```

Function 建構式

- 既然function是物件，那麼可以new Function嗎？
  - 可以，但千萬不要用...

Function

- 練習一：
  一函式 greaterThanTen，只接收一個參數a（數字），並回傳一個函式B，函式B只接受一個參數b（同樣數字），如果b>a則為true, b<=a為false

- var greaterThanTen = greaterThan(10);
  console.log(greaterThanTen(11));

- 練習二：
  寫一個函式power(base, exponent)，不使用Math物件，計算base的exponent次方。
  exponent為整數

Class 類別

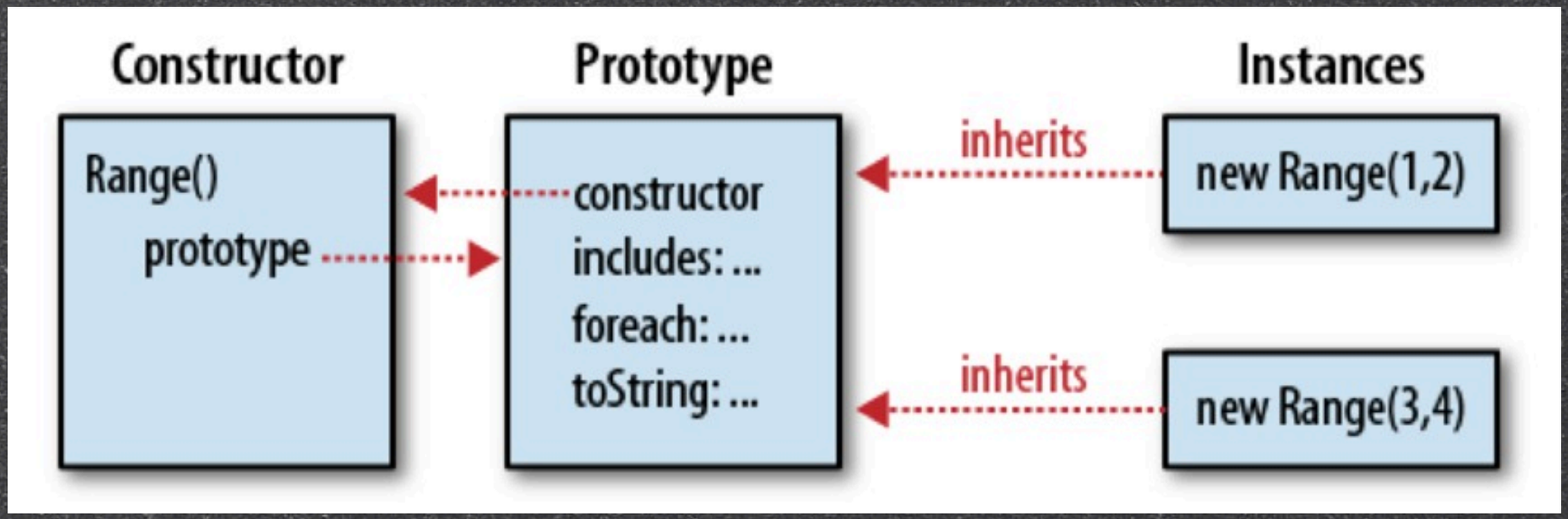
- A Simple Javascript class:

```
function Range(from, to) {
    // These are noninherited properties that are unique to this object.
    this.from = from;
    this.to = to;
}

Range.prototype = {
    constructor: Range,
    includes: function(x) { return this.from <= x && x <= this.to; },
    foreach: function(f) {
        for(var x = Math.ceil(this.from); x <= this.to; x++) f(x); },
    toString: function() { return "(" + this.from + "..." + this.to + ")"; }
};

// Here are example uses of a range object
var r = new Range(1,3); // Create a range object
r.includes(2); // => true: 2 is in the range
r.foreach(console.log); // Prints 1 2 3
console.log(r.toString()); // Prints (1...3)
```

Class 類別


Constructor
Prototype
Instances
Range()
prototype
constructor
includes: ...
foreach: ...
toString: ...
inherits
new Range(1,2)
inherits
new Range(3,4)

Class 類別

- Class fields/methods: 直接加在constructor
- Instance method: 加在prototype
- Instance property: 加在instance

Class 類別

```
function Complex(real, imaginary) {
    if (isNaN(real) || isNaN(imaginary))
        throw new TypeError();
    this.r = real;
    this.i = imaginary;
}

Complex.prototype.add = function(that) {
    return new Complex(this.r + that.r, this.i + that.i);
};

Complex.prototype.mul = function(that) {
    return new Complex(this.r * that.r - this.i * that.i, this.r * that.i +
this.i * that.r);
};

Complex.prototype.mag = function() {
    return Math.sqrt(this.r*this.r + this.i*this.i);
};

Complex.prototype.neg = function() { return new Complex(-this.r, -this.i); };
```

Class 類別

```
Complex.prototype.toString = function() {
    return "{" + this.r + "," + this.i + "}";
};

// Test whether this Complex object has the same value as another.
Complex.prototype.equals = function(that) {
    return that != null &&
        that.constructor === Complex &&
        this.r === that.r && this.i === that.i;
};


Complex.ZERO = new Complex(0,0);
Complex.ONE = new Complex(1,0);
Complex.I = new Complex(0,1);

Complex.parse = function(s) {
    try { // Assume that the parsing will succeed
        var m = Complex._format.exec(s); // Regular expression magic return new
Complex(parseFloat(m[1]), parseFloat(m[2]));
    } catch (x) { // And throw an exception if it fails
        throw new TypeError("Can't parse '" + s + "' as a complex number.");
    }
};

Complex._format = /^\{([^,]+),([^}]+)\}$/;
```

- 如果要把toString改成{1, 5i}這種格式要如何改

Class 類別

- 寫一個Cartesian類別
  - obj1 = new Cartesian(x1,y1), obj2 = new Cartesian(x2,y2)
  - obj1.distance() 計算與原點的距離
  - obj1.distance(obj2) 計算兩點距離
  - obj1.x, obj2.y 取得x,y
  - obj1.toString() = {x, y}
  - obj1.furthur(obj2) = true 如果距離原點較遠
  - Cartesian.ORIGIN 為原點

RegExp 正規表示法

- 用來搜尋字串，比對是否符合規則
- 不同語言的正規表示法有些微功能上的不同
- `var pattern = /s$/;`
- `var pattern = new RegExp("s$");`

RegExp 正規表示法

\0	null
\t	tab
\n	newline
\v	vertical tab
\f	form feed
\r	carriage return
\uxxxx	unicode character
\	逸出字元，ex: \?, \\, \.

RegExp 正規表示法

[...]	含有[]裡面的任何字元其一
[^...]	不含有[]裡面的任何字元其一
.	任何"一個"字元（不包含換行符號）
\w	等同[a-zA-Z0-9_]
\W	等同[^a-zA-Z0-9_]
\s	任何空白字元，包含\t, \v
\S	任何非空白字元
\d	[0-9]
\D	[^0-9]

RegExp 正規表示法

{n, m}	符合前一物件至少n次，至多m次
{n,}	符合前一物件至少n次
{n}	符合前一物件剛好n次
?	符合前一物件0或1次
+	符合前一物件1次以上
*	符合前一物件0次以上

RegExp 正規表示法

|	符合左邊或右邊的物件
(...)	群組，可與{}，*等結合
(?:...)	群組但不儲存
\n	與第n個群組相同
^	含有[]裡面的任何字元其一
$	不含有[]裡面的任何字元其一

RegExp 正規表示法

i	不分大小寫
g	全域搜尋
m	多行模式，^代表每行開始，$代表每行結束

RegExp in String

- search
- replace
- match
- split

```
console.log("JavaScript".search(/script/i));

console.log("jAvaSCRipt claZZ".replace(/javascript/gi, "JavaScript"));
console.log("hello _there_ and _here_".replace(/_(.*?)_/g, "<div>$1</div>"));

var url = /(\w+):\/\/([\w.]+)\/(\S*)/;
var text = "https://www.facebook.com/thingsaboutwebdev";
var result = text.match(url);
console.log(result);

var words = 'How are you doing, john?'.split(/[\s,\?\.]+/);
console.log(words);
```

RegExp 物件

- exec: 執行結果像是String.match，但如果有 g flag，則可以連續執行取得多筆結果

```
var pattern = /Java/g;
var text = "JavaScript is more fun than Java!";
var result;
while((result = pattern.exec(text)) != null) {
   console.log("Matched '" + result[0] + "'" +
               " at position " + result.index +
               "; next search begins at " + pattern.lastIndex);
}
```

- test: boolean test

```
var pattern = /java/i;
pattern.test("JavaScript"); // Returns true
```

常用 RegExp

- 限制英文字母/数字: /^[a-zA-Z0-9]*$/
- 日期(YYYY/MM/DD): /^((19|20)?[0-9]{2}[- /.](0?[1-9]|1[012])[- /.](0?[1-9]|[12][0-9]|3[01]))*$/
- Email: /^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4})*$/
- IP: /^((?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))*$/
- Password: /^(?=^.{6,}$)((?=.*[A-Za-z0-9])(?=.*[A-Z])(?=.*[a-z]))^.*$/
- VISA Card#: /^(4[0-9]{12}(?:[0-9]{3})?)*$/

RegExp

- 練習：

function extractDate(string)

輸入一字串，搜尋第一個DD/MM/YYYY的樣式，並回傳Date物件(利用new Date(DD, MM, YYYY)建立)。注意DD, MM有可能是個位或雙位數。

Coding Conventions

- camelCase, 建構式UpperCamelCase
- tab/space?
- 永遠使用大括號，大括號內縮排，開始的大括號接在同一行，結束的大括號自己一行
- 一定要加分號
- 空白行：程式碼明確段落使用
- 區段開頭宣告變數

Things not to do

- with
- eval
- forget semicolon
- == and !=