

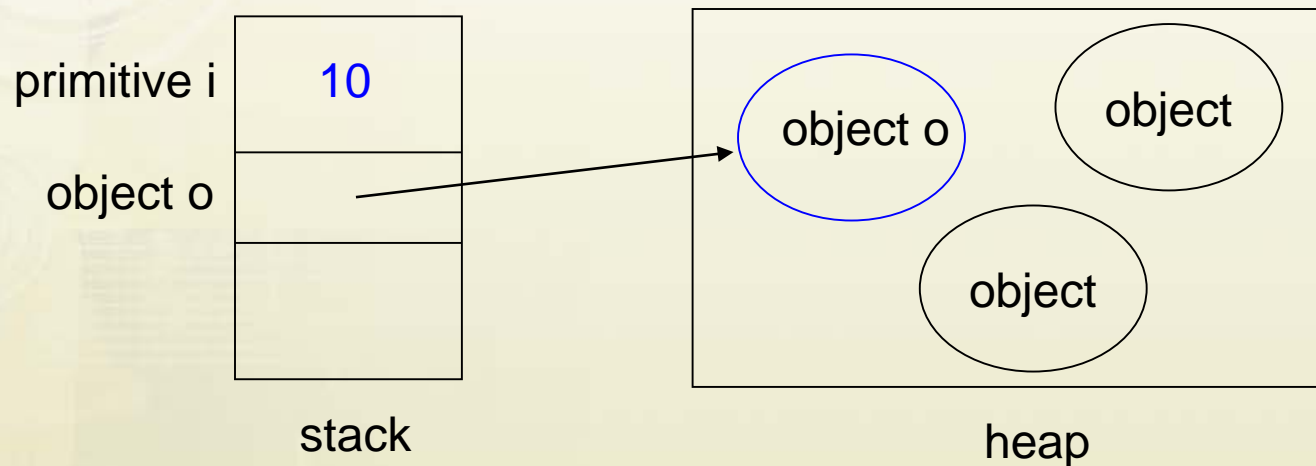
JavaScript Reference Types

Primitive versus Reference Types

- primitive types
 - number, string, boolean, undefined, null
 - primitive values are stored on the **stack** memory
- reference types
 - group of data and function definitions
 - also called **object definition**
 - each object has some Property and Method
 - e.g. document.write()
document is a object, write() is the method of document
 - are stored on the **heap** memory

Primitive versus Reference Types

- when a object is assigned to a variable, only the memory address of object are stored to variable



Reference Types

- Primitive Wrapper Types
 - Number, Boolean, String
- Native Reference Types
 - Array Type
 - Date Type, Math Type, ...
 - Function Type
 - Object Type
- Custom Types (User Defined Object)

Primitive Wrapper Types

- Every time a primitive value is read, an object of the corresponding primitive wrapper type is created behind the scenes
- The Wrapper objects have useful properties and methods, e.g. `toString()`.
- e.g.

```
var s = "hello";  
console.log(s.toUpperCase()); // HELLO
```

String

- e.g [String.html](#) 示範字串物件的各種方法與屬性
- note: `substr(m, n)` 和 `substring(m, n)` 的差異。如果 `text = "我願是千萬條江河"`，`text.substr(3,5)`會傳回 "千萬條江河"(第4個字元開始，取5個字元)，`text.substring(3,5)`會傳回 "千萬"(第4個字元開始，第5個字元結束)。
- 有關於字串的比對，只提到了 `indexOf()` 和 `lastIndexOf()` 兩個方法，事實上 JavaScript 對於字串的比對和代換有許多強大的功能，例如 `search`、`match`、`replace` 等函數，這些功能統稱「通用表示法」，後面會仔細介紹。

String Properties and Methods

性質或方法	說明
length	傳回字串的長度
toUpperCase()	換成大寫字母
toLowerCase()	換成小寫字母
concat()	字串並排（等效於使用加號）
charAt(n)	抽出第 n 個字元（ n=0 代表第一個字元）
charCodeAt(n)	抽出第 n 個字元（ n=0 代表第一個字元），並轉換成 Unicode
substr(m, n)	傳回一個字串，從位置 m 開始，且長度為 n
substring(m, n)	傳回一個字串，從位置 m 開始，結束於位置 n-1
indexOf(str)	尋找子字串 str 在原字串的第一次出現位置
lastIndexOf(str)	尋找子字串 str 在原字串的最後一次出現位置

String Properties and Methods

性質或方法	說明	等效的HTML標籤
big()	增大字串的字型	<big>...</big>
small()	減小字串的字型	<small>...</small>
blink()	閃爍字串（不適用於 IE 瀏覽器）	<blink>...</blink>
bold()	變粗體	...
italics()	變斜體	<i>...</i>
fixed()	變等寬字體	<tt>...</tt>
strike()	槓掉字串	<strike>...</strike>
sub()	變下標	_{...}
sup()	變上標	^{...}
fontcolor()	設定字串的顏色	...
fontsize()	設定字串的字型大小	...

Number

- Returns different object values to their numbers. If the value cannot be converted to a legal number, NaN is returned. If no argument is provided, it returns 0.
 - `Number(true)` → 1, `Number(false)` → 0
 - `Number(null)` → 0
 - `Number("011")` → 11
 - `Number("0xf")` → 15
 - `Number("string")` → NaN
 - `Number("")` → 0
- e.g. [Number.html](#)

Number methods

函式格式	說明
<code>x.toString([radix])</code>	將數值 <code>x</code> 轉成特定基底 <code>radix</code> 的字串。
<code>x.toFixed(n)</code>	將數值 <code>x</code> 轉成小數點以下 <code>n</code> 位有效數字的小數點表示法。
<code>x.toExponential(n)</code>	將數值 <code>x</code> 轉成小數點以下 <code>n</code> 位有效數字的科學記號表示法。
<code>x.toPrecision(n)</code>	將數值 <code>x</code> 轉成共具有 <code>n</code> 位有效數字。

Number Properties

■ some useful properties

常數表示法	說明
Number.MIN_VALUE	傳回能在 JScript 中表示最接近零的數字。大約等於 5.00E-324。
Number.MAX_VALUE	傳回能在 JScript 中表示的最大值。大約等於 1.79E+308。
Number.NEGATIVE_INFINITY	傳回一個能在 JScript 中表示、且比最大負數 (-Number.MAX_VALUE) 還要小的值。
Number.POSITIVE_INFINITY	傳回能在 JScript 中表示且大於最大數 (Number.MAX_VALUE) 的值。
Number.NaN	一個特殊值，可指出算術運算式的傳回值不是一個數字。

Number toString() 範例

- e.g. [NumberRadix.html](#) 利用 toString() 函式，顯示不同基底的表示方式

```
document.writeln("<td>" + x.toString());
```

```
document.writeln("<td>" + x.toString(2));
```

```
document.writeln("<td>" + x.toString(8));
```

```
document.writeln("<td>" + x.toString(16));
```

- 如果 toString() 沒有參數，預設以十進位為底

Note: toString() Example

- 利用 toString() 將各種物件轉成字串 [toString.html](#)

物件	toString() 的結果
Array（陣列）	將 Array 的元素轉換為字串，形成以逗號串連起來的結果，此結果與 Array.toString() 和 Array.join() 得到的結果相同
Boolean（布林）	如果布林值為 True，會傳回 "true"；否則會傳回 "false"
Date（日期）	傳回顯示日期的文字形式
Error（錯誤）	傳回包含錯誤訊息的字串
Function（函數）	傳回函數的定義
Number（數字）	傳回數字的文字表示法
String（字串）	傳回 String 物件的值
自訂物件	傳回 "[object Object]"

Reference Types - Array Type

Array Type

- arrays are ordered lists of data
- each item of array can hold any type of data
- first way to create array
 - e.g. `var myArray = new Array();` // 產生一個空的陣列
 - e.g. `var myArray = new Array(10);`
 - create an array containing **ten items** each item gets **undefined value**
 - e.g. `var myArray = new Array("item1", "item2", "item3");`
 - the new operator can be omitted
- 2nd way to create array
 - `var myArray = [];`
 - `var myArray = [,,,,,]`
 - create 5 items in Firefox, 6 items in IE, not recommended
 - e.g. `var myArray = ["item1", "item2", "item3"];`

Array Index

- 要使用陣列變數時，需先宣告，但可以不用設定陣列的元素個數。我們可以使用索引 (Index) 來存取每一個元素的值，索引從 0 開始，例如陣列 myArray 的第一個元素為 myArray [0]，第五個元素為 myArray [4]，依此類推。

- e.g. [array/arrayList.html](#)

```
var myArray = new Array();
```

```
myArray[0] = "This is a test"; // 加入第 1 個元素
```

```
myArray[1] = 3.1415926; // 加入第 2 個元素
```

```
myArray[2] = "The last element"; // 加入第 3 個元素
```


Array Listing

- (續上例) 列出陣列中元素

```
myArray = ["教務處", "學務處", "總務處"];
```

```
for (prop in myArray)
```

```
    document.write("<br>myArray[" + prop + "] = " +  
myArray[prop]);
```

- 說明

- JavaScript 會把 0, 1, 2 當成是陣列物件的性質
- 在取用陣列的元素時，還是必須使用 myArray[2] 或是 myArray["2"] 等，而不能使用 myArray.2。

Array length property

- The length property sets or returns the number of elements in an array.
 - 也就是陣列的 length 屬性可讀也可寫
- 修改 length 屬性可以更改陣列大小
 - 減少 length 則 (new_length-1) 之後的元素將被刪除
 - 增加 length 則增加 (old_length-1) 之後的元素，型態為 undefined
 - e.g. [arrayLength.html](#)

用陣列方式存取物件屬性

- (續上例)

```
for (prop in document)
```

```
    document.write("<br>document." + prop + " = " +  
document[prop]); </script>
```

- 說明

- 使用 document.xyz 或 document["xyz"] 來存取屬性 xyz，得到的結果是相同的。

[note] arrays with named indexes

- JavaScript does not support arrays with named indexes.
 - In JavaScript, arrays always use numbered indexes.
 - If you use a named index, JavaScript will redefine the array to a standard object.
-
- [note] Arrays with named indexes are called associative arrays (or hashes).

Array methods

方法	說明
concat()	傳回一個由兩個或兩個以上陣列並排而成的新陣列
join()	傳回一個字串值，它是由陣列中的所有元素串連在一起所組成，並且用特定的分隔字元來分隔
pop()	移除陣列的最後一個元素，並將它傳回
push()	附加新元素到陣列尾部，並傳回陣列的新長度
reverse()	傳回一個元素位置反轉的陣列
shift()	移除陣列的第一個元素，並將它傳回
slice()	傳回陣列的一個區段
splice()	移除陣列中的元素，並依需要在原位插入新元素，然後傳回被刪除的元素
sort()	傳回一個元素已排序過的陣列
toString()	傳回一個物件（或陣列）的字串表示法
unshift()	在陣列開始處插入指定的元素，並傳回此陣列

Array toString(), join()

- 輸出陣列物件時，toString() 會將陣列轉換成由逗號隔開的字串，或使用 join() 來指定輸出的格式
- e.g. [arrayJoin.html](#)
 - 程式碼重點
 - document.writeln("myPet.toString()="+myPet.toString()+"
");
 - document.writeln("myPet.join()="+myPet.join()+"
");
 - document.writeln("myPet.join(',')="+myPet.join(',')+"
");
 - document.writeln("myPet.join('+')="+myPet.join('+')+"
");
- 說明
 - 從範例可以看出，myPet.toString() 和 myPet.join() 得到的結果是一樣的。

Array split(), concat()

- 將字串拆成陣列：split()

- 陣列的串接：concat()

- e.g. [arrayConcat.html](#)

- 程式碼重點

- ```
array1=str1.split('、'); // 將字串拆成陣列
```

- ```
array2=str2.split('、'); // 將字串拆成陣列
```

- ```
array3=array1.concat(array2); // 串接兩個陣列
```

- 說明

- 字串物件由 split() 中的字串"、"分割成陣列。

- Concat 可以使兩陣列結合在一起形成新的陣列

# Array splice()

- 置換陣列中元素

- e.g. [arraySplice.html](#)

- 程式碼重點

```
myPet = ["鼠", "牛", "虎", "兔", "龍", "蛇", "Cat",
"Bird", "狗", "豬"];
```

```
myPet.splice(6, 2, "馬", "羊", "猴", "雞");
```

- 說明

- splice(index,howmany,element1,...,elementX)
  - delete from index to index+howmany-1, then  
insert the element1, ..., elementX



# Array sort(), reverse()

- 陣列中元素的排序，可使用陣列方法 `sort()` 和 `reverse()`
- e.g. [arraySort1.html](#)
- 說明
  - 要注意的是 `sort()` 會先將數值轉成字串，再進行字串的排序。
  - 若要進行數值的排序，必須自訂比較函數，並將此函數傳進 `sort()`，稍後說明。

# Array pop(), push(), shift(), unshift()

- 陣列元素的運作：pop(), push(), shift(), unshift()
  - pop(), push() 類似堆疊，於陣列後方操作
  - shift(), unshift() 於陣列啟始處操作
- e.g. [arrayPop.html](#)

- 程式碼重點

```
popped = myPet.pop();
elementCount = myPet.push("龍", "蛇");
shifted = myPet.shift();
myPet.unshift("馬", "羊");
```

# Array Iteration Methods

- `Array.forEach()`
- `Array.map()`
- `Array.filter()`
- `Array.reduce()`, `Array.reduceRight()`
- `Array.every()`
- `Array.some()`
- `Array.indexOf()`, `Array.lastIndexOf()`
- `Array.find()`, `Array.findIndex()`
- [https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

# Reference Types - Math Type

數學物件

# Math

- e.g. [math.html](#) 示範由數學物件的一些 properties and methods

| 方法        | 說明                       |
|-----------|--------------------------|
| abs(x)    | 取一個數 x 的絕對值              |
| ceil(x)   | 傳回大於輸入值 x 的最小整數          |
| floor(x)  | 傳回一個比輸入值 x 小的最大整數        |
| log(x)    | 計算以 e (2.71828) 為底的自然對數值 |
| exp(x)    | 傳回以 e (2.71828) 為底的冪次方值  |
| pow(a, n) | 計算任意 a 的 n 次方            |
| sqrt(x)   | 求出一個數 x 的平方根             |
| round(x)  | 四捨五入至整數                  |

# Math methods

| 方法                     | 說明               |
|------------------------|------------------|
| <code>max(a, b)</code> | 傳回兩個數 a, b 中較大的數 |
| <code>min(a, b)</code> | 傳回兩個數 a, b 中較小的數 |
| <code>random()</code>  | 隨機產生一個介於 0~1 的數值 |
| <code>sin(x)</code>    | 正弦函數             |
| <code>cos(x)</code>    | 餘弦函數             |
| <code>tan(x)</code>    | 正切函數             |
| <code>asin(x)</code>   | 反正弦函數            |
| <code>acos(x)</code>   | 反餘弦函數            |
| <code>atan(x)</code>   | 反正切函數            |

# Math Example

- [mathRandom.html](#) 利用亂數選擇字串陣列的元素

- 程式碼重點

```
index = Math.floor(Math.random()*text.length);
```

- 說明

- `Math.random()` 會傳回一個介於 0 和 1 之間的亂數。
- 因此 `Math.random()*text.length` 會產生一個介於 0 和 `text.length` 之間的亂數（帶有小數）。
- 最後，`Math.floor(Math.random()*text.length)` 會產生一個介於 0 和 `text.length-1` 之間的整數（包含頭尾），所以可以用來選取 `text` 陣列中的一個元素。

# Reference Types - Date



# Date Example -- date.html

- new Date() creates a new date object with the current date and time
- Date objects are **static**. The computer time is ticking, but date objects are not.
  - 也就是說 Date 物件內容不會隨時間而改變

# Date methods

| 方法               | 說明                               |
|------------------|----------------------------------|
| toString()       | 以標準字串來表示日期物件                     |
| toLocaleString() | 以地方字串（依作業系統而有所不同）來表示日期物件         |
| getFullYear()    | 取得年份                             |
| getMonth()       | 取得月份（需注意：0 代表一月，因此例如若是八月，結果就是 7） |
| getDate()        | 取得日期                             |
| getHours()       | 取得時數                             |
| getMinutes()     | 取得分鐘數                            |
| getSeconds()     | 取得秒數                             |
| getDay()         | 取得星期數（例如若是星期四，結果就是 4）            |

# Function

# Function

- a block of code designed to perform a particular task.
- benefits:
  - re-used
    - save code: code is repeated many times with only a few minor modifications
  - modular design: programming in a modular style
  - black-box
    - given a function, we need only know "what it does".  
There is no need to know "how it does".  
( maybe we do care if it does the job efficiently. )
  - team collaboration

# First-Class Function

- A programming language is said to have First-Class functions when functions in that language are treated like any other variable.

For example, in Javascript, a function

- can be passed as an argument to other functions,
- can be returned by another function
- can be assigned as a value to a variable.

# Function Definition 1/2

- Syntax

```
function functionName(Arguments) {
 statements
 ...
 return (return_value) // 非必要
}
```

- 括號裡的引數 (Input Arguments)，可以沒有；若有多個則以逗號分開。
- 若有需要，函數最後可用 **return** 來傳回值 (數值、字串，或其他型態的資料) 至呼叫此函數的程式。

# Function Definition 2/2

- 函數的定義，通常寫在 `<head>` 及 `</head>` 之間，以確保 HTML 主體在被呈現前，所有相關的 JavaScript 函數都已被載入，並隨時可被執行
- 定義函數並不代表函數的執行，只有在程式中呼叫函數的名稱後，才會執行該函數。
- 一般來說，我們希望函數的定義出現的位置和它被呼叫之處能越接近越好，以方便程式管理，在這種情況下，只要函數定義出現在其被呼叫之前即可[note: function hoisted]

# Function Definition - Example

■ e.g. [function.html](#)

```
<script type="text/javascript">
```

```
 function show(name, message) {
 alert(name + ", " + message);
 }
```

```
 show("John", "Hi");
 show("Boy", "Wake up!!");
 show("Girl", "Wake up!!");
```

```
 function max(x, y) { if (x < y) return y; else return x; }
```

```
 var a=1, b=2;
```

```
 var c=max(a, b);
```

```
</script>
```



# Example - Array 的自訂比較函數

陣列的自訂比較函數必須：

- 具有兩個參數 e.g. `myComp(arg1, arg2)`
- `return` 一個數值來表示兩個參數間的關係
  - $<0$ , 表示 `arg1` 排在 `arg2` 之前
  - $=0$ , 表示 `arg1` 等於 `arg2`
    - no guarantee of stability
  - $>0$ , 表示 `arg1` 排在 `arg2` 之後
  - 如果 `return` 一個非數值，do nothing! (no error)
- e.g. `arraySort2.html`  
`function comparisonFunction(a, b){ return(a-b); }`

# Call by Value or Call by Reference

- All function arguments in ECMAScript are passed by value
- Q: What happens if function changes its argument value?
- for primitive types - nothing
  - strings, numbers, boolean values, and null are unchanged in the original variable.
  - called by value
  - the actual value of variable is passed to function argument

# Call by Value or Call by Reference

- for reference type, the variable store reference values, if object arguments are passed, the address of object be passed. So it is equivalent to be passed by reference
  - called by reference (called by address)
  - A function can change the properties of an object

# Variable Scope 1/4

變數可因其有效範圍的不同，分成兩類

- Local variables (區域/局部 變數)
  - **var** operator makes the variable local to the scope in which it is defined. (see next slide)
    - 必須在變數第一次使用時加上 **var**
- Global variables (全域變數)
  - if **var** operator is omitted, means to define a global variable ( is not recommended )
  - 在整個程式中都可以看的見、而且每一個函數都可以用的變數。

# Variable Scope 2/4

so the story is

- variable defined outside the function is global variable, no matter use operator **var**
- variable defined in the function
  - is local variable if use operator **var**
  - is global variable if do not use operator **var** (is not recommended)
- 如果在函數內有一個變數名稱與全域變數名稱相同，則區域變數優先權 (precedence) 高於全域變數
- 區域變數有助於降低記憶體浪費，提高系統使用效率
- 請養成用 **var** 宣告變數的好習慣

# Variable Scope Example

- e.g. [variableScope.html](#)

- 程式碼重點

function 內： `var x=5; // 局部變數`  
`y=8; // 全域變數`  
function 外： `x = 10; // 全域變數`  
`y = 10; // 全域變數`

- 說明

- 函數內的變數 `x` 和外面的 `x` 雖然名稱一樣，但是用 `var` 宣告，執行函數後外面的變數 `x` 值不受影響。
  - 函數內的變數 `y` 和函數外面 `y` 的名稱一樣，且執行函數完後，外面的變數 `y` 會變成 8。
  - 為了減少除錯的時間，所有函數的內部變數，在第一次使用時最好加上 `var`，以確認其有效範圍只在此函數內。

# Variable Scope Types

There are four (technically three) types of scope in javascript:

- window scope
  - These variables can be used anywhere in your script at any time. (global variable)
- function scope
  - These variables are only available inside of a particular function.
- class (arguably same as function)
  - These variables are only available to members of a particular class.
- block (new in ECMAScript 6) [註]
  - These variables only exist within a particular code block (i.e., between a set of '{' and '}').



# Block Scope

```
for (var i=1; i<10; i++) { ... }
console.log(i); // you will get 10
```

- In Javascript, the variable `i` still exists outside the for loop

```
for (let i=1; i<10; i++) { ... }
console.log(i); // undefined
```

- Block-Level Scope: the variable `i` will be destroyed after the loop execution



# Block Scope - let, const

- The let and const block bindings introduce lexical scoping to JavaScript. These declarations are **not hoisted** and only exist within the block in which they are declared. you cannot access variables before they are declared, even with safe operators such as typeof.
- If you use let or const in the global scope, a new binding is created in the global scope but **no property is added to the global object**. e.g. :

```
var hi = "Hi!";
console.log(window.hi); // work!
let hi = "Hi!";
console.log(window.hi); // error!
```

# Function Arguments array

- in addition to set the variable `argu1`, `argu2...`, all parameters are stored in variable **arguments**, which is an array
- `arguments.length` is parameter count
- `arguments[i]` is the *i*th parameter value
  - `function functionName(argu1, argu, ....) {  
statements; ...  
}`
- 所以在Javascript中呼叫函數時，如果超過函數定義的引數個數時，仍然可執行。

# Function Arguments Example

- e.g. [arguments.html](#)

```
function sum() {
 let i,total=0;
 for (i=0; i<arguments.length; i++) {
 total = total + arguments[i];
 }
 return total;
}

document.write(sum(10,10)+"
");
document.write(sum(10,20,30)+"
");
```

# recursive function 遞迴函數

- recursion is a technique where a function invokes itself.

- ```
function functionName(arguments){  
    if (...) // 結束條件  
        functionName(arguments); // recursive call  
}
```

- 函數呼叫自己，直到某個條件達成時才停止。
- 如果沒有結束條件，這個函數就會永無止盡的呼叫，形成無窮遞迴。
- 遞迴易設計可以精簡程式碼，但耗CPU資源(但在某些特殊演算法上效果較迴圈佳)。

Classification of Recursion

- Direct Recursion
 - e.g. factorial $n! = n * (n-1)!$
- Indirect Recursion
 - mutual recursion: function A call function B, then function B call function A.
e.g. Cyclic Hanoi Towers

recursive function -- factorial.html

- computer the factorial of n in recursion
 - ```
function factorial(n){
 if (n<=1) return(1);
 return (n*factorial(n-1)); // invoke factorial() again
}
document.write("5! = ", factorial(5))
```
- This kind of recursion is called linear recursion, because the stack grows linearly with n.

# recursive function -- fibonacci.html

- computer the n'th Fibonacci number in recursion
  - ```
function fibonacci(n){  
    if (n<=1) return(1);  
    return (fibonacci(n-1)+ fibonacci(n-2));  
}  
document.write("f10 = ", fibonacci(10))
```
- This kind of recursion is called tree recursion, because the run-time stack grows and shrinks like a tree.

Recursion versus Iteration

	Recursion	Iteration
pro	easy to design	fast
con	time and space overhead	hard to design

- factorial and Fibonacci have obvious iterative solution. Iteration is better solution for them.
- how about MaCarthy's 91 function:

$f(n) = n-10$, if $n > 100$
 $= f(f(n+11))$, otherwise

$f(100) = f(f(111)) = f(101) = 91$
 $f(91) = f(f(102)) = f(92) = f(93) = \dots = f(100)$

- recursion or iteration?

$f(n) = n-10$, if $n > 100$
 $= 91$, otherwise

Recursion versus Iteration

- usually, time is more important than space. we can allocate more memory (tabulation) to store intermediate data, it can improve the recursion performance. e.g.

combination $c(m,n) = c(m-1,n)+c(m-1,n-1)$
 $= (c(m-2, n)+c(m-2,n-1))+(c(m-2,n-1)+c(m-2,n-2))$,
It's not necessary to compute $c(m-2,n-1)$ twice.

Function Definition more 1/2

- 1. typical way
 - `function functionName(argu1, argu2,) { ... }`
 - e.g. `function sum(x, y) { return (x+y); }`
 - loaded into the scope before code execution
- 2. function expression
 - `var functionName=function(argu1, argu2,) { ... }`
 - e.g. `var sum=function(x, y) { return (x+y); };`
 - create anonymous function, then assign to a variable
 - unavailable until the function expression are executed

Function Definition more 2/2

- 3. defined by **Function** object constructor
 - `var functionName = new Function(argu1String, argu2String, ..., bodyString);`
 - e.g. `var show = new Function("alert('hello!')");`
 - e.g. `var sum = new Function("x", "y", "return (x+y)");`
 - This way is **not recommended**.
- 4. **Arrow Function** [ES6]
 - short syntax of expressions function. e.g.
`var sum = (x,y) => x+y;`
is equivalent to
`var sum = function(x,y) {return x+y;}`

Function Definition more [note]

- unusual way

```
var functionName = function  
  anotherName(listOfVariableNames) { function-body  
};
```

- In this particular case, because the function is being assigned, and not defined normally, the name `anotherName` can be used by the code inside the function to refer to the function itself, but the code outside the function cannot see it at all

Arrow Function Limitation

- No this, super, arguments object
- Cannot be called with new
 - Arrow functions do not have a `[[Construct]]` method.
 - no `new.target` bindings
- No prototype
 - since you can't use `new` on an arrow function, there's no need for a prototype.

Function Concept Summary

- function is an object.
- each function is an instance of the Function type.
- function name is a simple reference to function object.
- function name can be reassigned to new function object


Function Hoisted

- no matter where functions definition are declared, they get hoisted to the top of program block behind the scenes. (include function name and function definition)
- but for the function expression, function implementation do not get hoisted. (only the variable get hoisted.)
- e.g. [hoisted.html](#)

Object & Prototype Linkage

What is Object

- An object is a collection of related data(property) and/or functionality(method)
- e.g.

Object	Properties	Methods
	<code>car.name = Fiat</code> <code>car.model = 500</code> <code>car.weight = 850kg</code> <code>car.color = white</code>	<code>car.start()</code> <code>car.drive()</code> <code>car.brake()</code> <code>car.stop()</code>

- accessed the object's properties and methods using dot notation.

Object Literal

- can create object with an object literal

```
var student = {name:"Timmy", id:"0200102"};
```

- 這種物件表示法稱為

JSON (JavaScript Object Notation)

- (註: JSON要求屬性名稱一律要使用引號，JavaScript 不要求)
- 使用 object literal 的好處是可以指定含有空格的屬性，但是若要存取此屬性，則必須使用「物件["屬性"]」的方式來進行，而不能使用「物件.屬性」的方式。

built-in Object type

- or create object with an built-in Object() type
 - Object() is the fundamental reference type
 - all objects are instances of Object type

- create an instance of Object, e.g.

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

- e.g. [object.html](#), customize object:

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

```
student.name = "Timmy";
```

```
student.id = "0200102";
```

先產生一個原型物件，然後增加 property，產生自訂物件。

[註] in operator

- 以 in 運算子測試物件的欄位是否存在
 - in 除了搭配 for 使用外，也可以用來檢查物件是否含有某屬性，如果這個屬性存在，就會回傳 true，不存在則回傳 false。

- e.g. [object.html](#)

```
if (field in student)
```

```
    document.write(field + " is a field of  
student<br>");
```

Factory Pattern

- 使用函數搭配原型 Object 產生物件
- e.g. [factory.html](#)

```
function student(name, id) {  
    var o = new Object();  
    o.name = name;  
    o.id = id;  
    return o;  
}  
  
var student1 = student("Timmy", "0200102");
```

Constructor Pattern

- e.g. [constructor.html](#)

```
function Student(name, id) {
```

```
    this.name = name;
```

```
    this.id = id;
```

```
    this.display = function () { ... };
```

```
}
```

```
var student1 = new Student("Timmy", "0200102");
```

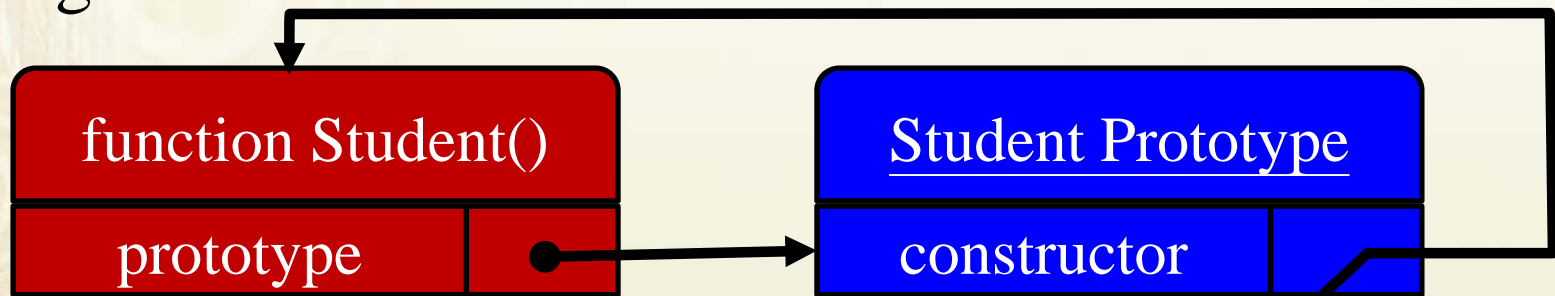
- 說明
 - 使用 Constructor (建構子) 概念來產生物件
 - this 代表 new 所產生的物件

Constructor Pattern's Imperfection

- The major downside to constructors is that methods are created once for each instance.
 - e.g. each instance of Circle get its own instance of area() that output the circle area.
- one of solutions is to move the function definition outside of the constructor.
 - cons: create function in the global scope
- another solution is to use prototype pattern

Prototype Object of Function

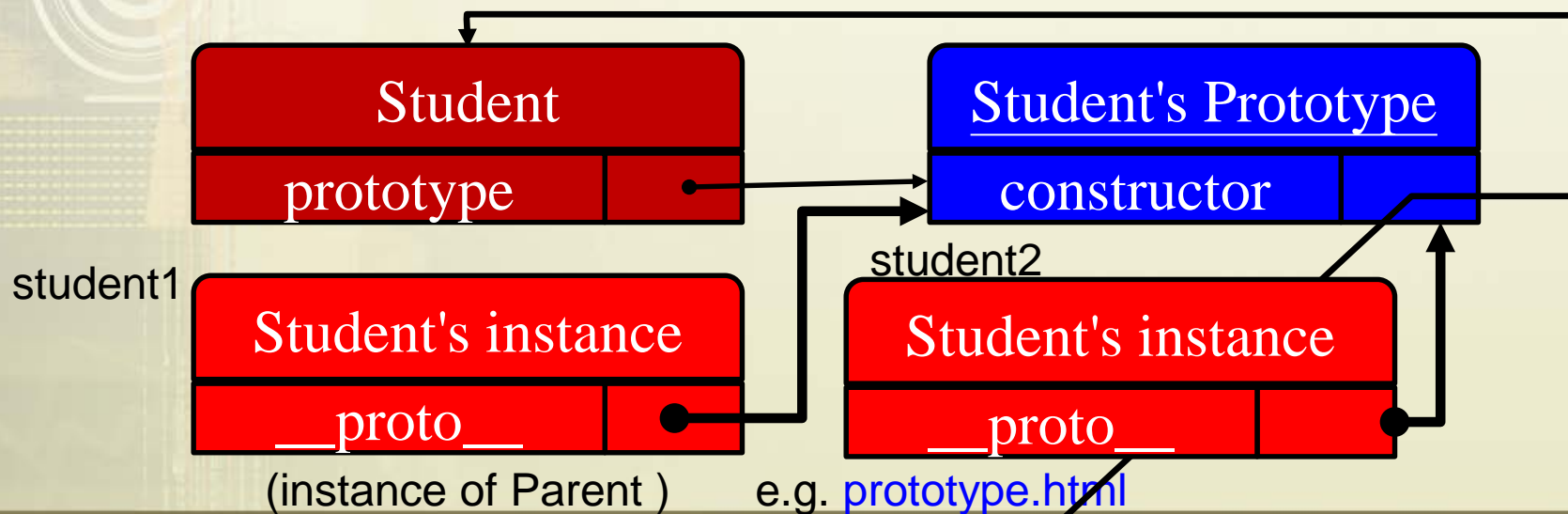
- each function is created with a prototype property, which refer to a Prototype object
- e.g.



```
function Student(...) { ... };  
Student.prototype.display = function () { ... };  
Student.prototype.extension = "others";
```


How prototype work

- every instance has an internal pointer to the constructor's prototype object.
 - this pointer may be named `__proto__`, dependent on browser
 - so all instances share the prototype object



How prototype work

- Whenever a property is accessed, the search begins on the object instance itself. If not found, continues to search the `__proto__` object.
- This is how prototypes are used to share properties and methods among multiple object instance.

How prototype work [note]

- e.g.

```
var foo = function (){ };  
foo.hi = function(){console.log('foo.hi');};  
foo.prototype.hi = function(){console.log('prototype.hi');};  
foo.hi(); // foo.hi  
var f = new foo();  
f.hi(); // prototype.hi
```

- `hasOwnProperty()` determines if a property exists on the instance. It returns true only if the property on the instance itself, not on prototype.

e.g. `f.hasOwnProperty("hi")`

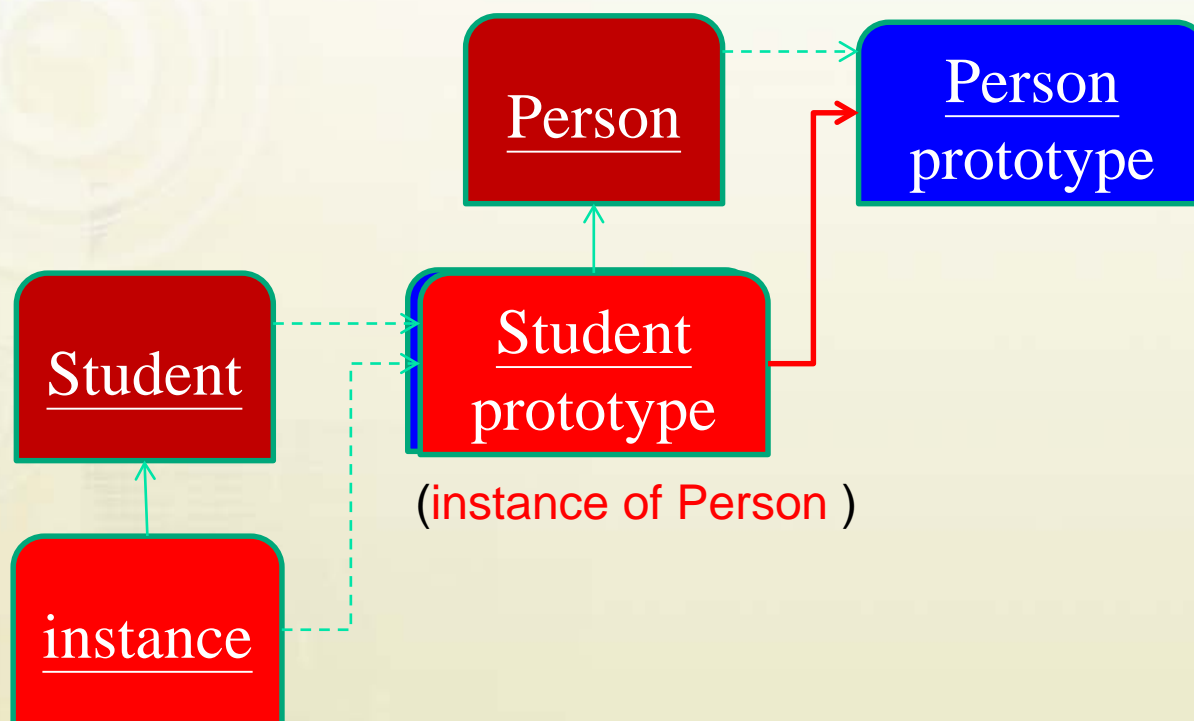
- e.g. `if (hi in f)`

- `in` operator returns true if the property is accessible, means the property can be on the instance or on the prototype.

prototype chain

- 可利用 prototype 特性模擬物件的繼承

e.g. [prototypeChain.html](#)



Class [ES6]

- e.g. [class.html](#)
- A class body can only contain methods, but not properties
- Class declarations are not hoisted
- Class be invoked only via new, not via function call
- two ways to define a class:
class declarations and *class expressions*.

Exercise

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Test_your_skills:_Object-oriented_JavaScript