

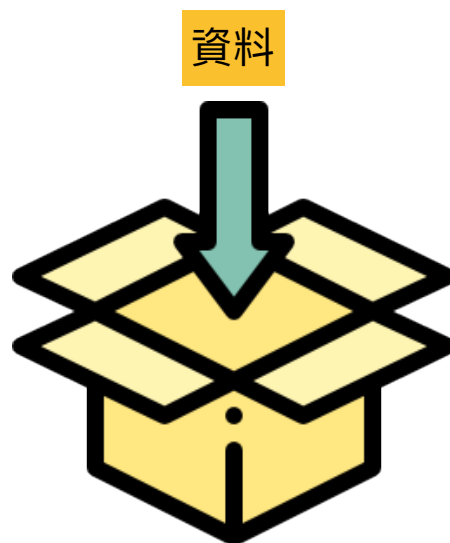
# ES6七大新特性

let / const





變數 - 有「盒子」的概念，把資料放在裡面，然後標註上一個名稱作為標籤



Icon made by Elias Freepik(<http://www.freepik.com/>) from [www.flaticon.com](http://www.flaticon.com)



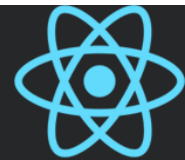
宣告時使用 **let** 關鍵字，一開始宣告時要養成給定初始值的好習慣

let a = 1

關鍵字一定要全小寫英文

變數的名稱(識別名)，在這個程式碼文件中是唯一的

這不是「等於」這是  
「指定」或「給定」的  
意思

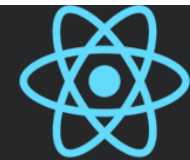


變數的資料類型，會隨著給定的值而變動，這稱為「動態資料類型」或是「鬆散(弱)資料類型」的特性

```
let a = 1  
a = 'hello'
```

變數 a 的資料類型現在是「數字」

變數 a 的資料類型現在是「字串」



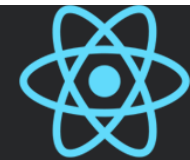
宣告時使用 **const** 關鍵字，常數規定宣告時一定要給定初始值，而且之後不能再次作給定運算

**const b = 1**

關鍵字一定要全小寫英文

常數的名稱(識別名)，在這個程式碼文件中是唯一的

這不是「等於」這是  
「指定」或「給定」的  
意思



常數是一種「**具有固定值的變數**」，一開始宣告時給定值後，就不能再次作給定的運算，所以它的資料類型一開始就決定好了

```
const b = 1
```

```
b = 'hello'
```

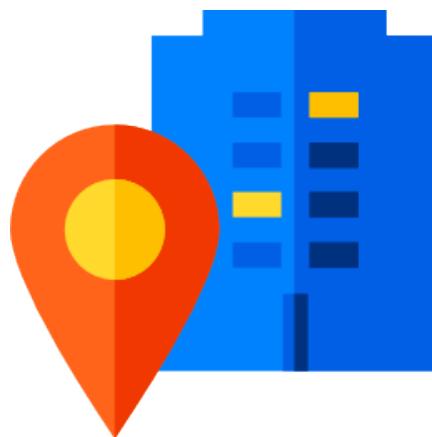
常數 b 的資料類型現在是「數字」

作再次指定的運算，這行會造成錯誤





常數可以用於複合型的資料類型，例如 **物件** 或 **陣列** 的宣告，因為裡面記錄的是「**地址**」，類似於某大樓的地址，如果改變裡面的成員並不會更動到地址



Icon made by Elias Freepik(<http://www.freepik.com/>) from [www.flaticon.com](http://www.flaticon.com)



❓ 範例中的 d 陣列經過 c 陣列改變成員值後，此時裡面的成員是什麼？

```
const c = [1, 2]
```

```
const d = c
```

```
c[0] = 3
```

→ 常數 c 是一個陣列，裡面有兩個成員

→ 常數 d 指定為 c 陣列，d 的地址與 c 相同

→ 改變 c 陣列中的成員值





常數也可以用於函式表達式的宣告，函式表達式只能使用 **const** 宣告

```
const foo = function( ) { }
```

註：JS中的函式宣告有兩種，一種是函式定義語法(簡稱FD)，一種是函式表達式語法(簡稱FE)。它們會被使用在不同的場合，而且某些特性不太相同。



常數也可以用於函式表達式的宣告，函式表達式只能使用 **const** 宣告

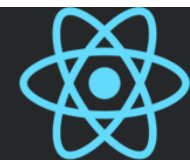
```
const foo = function( ) { }
```

註：JS中的函式宣告有兩種，一種是函式定義語法(簡稱FD)，一種是函式表達式語法(簡稱FE)。它們會被使用在不同的場合，而且某些特性不太相同。

## 變數&常數撰寫風格建議



- ✓ 優先使用常數宣告(const)
- ✓ 使用變數宣告(let)時，宣告時就要指定初始值，如果不確定是什麼類型的值，可以使用null
- ✓ 在函式或程式碼文件的最前面宣告變數(常數)
- ✓ 一行語句宣告一個變數(常數)
- ✓ 把let的宣告放在一起，const的宣告放在一起



- ✓ 區域作用域(block) vs 函式作用域(function)
- ✓ for圓括號中的let變數仍然是在區塊作用域
- ✓ for迴圈中的let變數會作重新綁定
- ✓ let與const的提升(hoisting)，以及TDZ(暫時死區)

註：以上都是比較ES6之前，單只用var來宣告變數的情況。

# ES6七大新特性

## 箭頭函式





箭頭函式語法是「**函式表達式**」的簡短語法，但它與函式原本的宣告方式有些特性上的不同

```
const foo = function(x) { return x + 1 }
```

```
const foo = (x) => x + 1
```



箭頭函式語法是使用肥箭頭符號( $\Rightarrow$ )，區分左邊的函式傳入參數，與右邊的函式主體，語法可以再進行省略：

1. 只有單一個傳入參數時，可再省略圓括號( $()$ )
2. 函式主體中只有單一行回傳的表達式(or值)，可省略花括號( $\{\}$ )

```
const foo = x => x+1
```

```
const foo = x => { x+1 }
```

這兩種語句是不同的

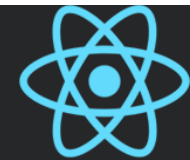


JSX語法裡搭配使用箭頭函式，因為標記有太多列時，可以使用圓括號(())作為撰寫時的分行語法，執行時仍然是同一行(自動加return的語法)

```
const HelloWorld = (props) => (  
  <div>  
    <h1>{props.text}</h1>  
  </div>  
)
```



## 箭頭函式 vs 一般函式



	箭頭函式	一般函式
預設特性	沒有arguments物件	有arguments物件
建構函式	不能使用	可以使用
this	詞法上綁定(Lexical this) 由週邊的作用域所決定	由呼叫函式的擁有者物件 (Owner)決定
call/apply/bind可否覆蓋this	不行	可以



## 箭頭函式不能/不建議使用的情況

- ➡️ ✓ 用物件字面文字定義物件時，物件中的方法
- ➡️ ✓ 在物件的prototype屬性中定義的方法
- ➡️ ✓ DOM事件處理的監聽者(事件處理函式)
- ➡️ ✓ 建構函式

註：以上指有用到this的情況，或是與this有關。

註：箭頭函式中沒有一般函式定義中的隱藏arguments物件，也不能作為constructor使用。



- ➡️ ✓ callback(回調, 回呼)優先使用箭頭函式
- ➡️ ✓ 雖然箭頭函式的左邊(傳入參數)只有一個時可以省略圓括號(()), 但建議還是不論幾個都用圓括號框起來
- ➡️ ✓ 避免合併使用箭頭函式與其他的比較運算符(>=, <=), 會造成閱讀不便與混亂
- ➡️ ✓ 肥箭頭符號的前後要加一個空格, 不要黏在一起。另外, 不要直接在符號前後換行

# ES6七大新特性

函式傳入預設值





## 函式傳入參數預設值語法

取代原本使用falsy與邏輯或(||)設定函式預設值的語法，預設值可以是值、表達式、物件、陣列、函式或this等等

```
function foo(x = 1) {}
```

註：只有在傳入參數值為 **undefined** 或不存在時，才會使用預設值。注意 **null** 仍被視為有值的情況。



## 函式傳入參數預設值語法

取代原本使用falsy與邏輯或(||)設定函式預設值的語法，預設值可以是值、表達式、物件、陣列、函式或this等等

```
function foo(x = 1) {}
```

註：只有在傳入參數值為 **undefined** 或不存在時，才會使用預設值。注意 **null** 仍被視為有值的情況。

# ES6七大新特性

解構賦值





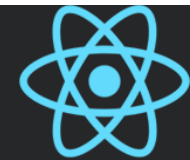
## 解構賦值(Destructuring Assignment)語法

解構賦值是”**解析結構+指定值運算**”的意思。是專門設計給**物件**與**陣列**使用的指定值語法，以”**鏡子**”般的對映樣式，提取物件與陣列中的成員值。設計此語法的目的是讓程式碼更簡短與提高閱讀性。

```
const [a, b] = [1, 2]
const {a: x, b: y} = {a: 1, b: 2}
const {a, b} = {a: 1, b: 2}
```

註：可使用指定預設值、可搭配函式傳入參數與其餘運算符使用





- ✓ 總是使用const宣告來作解構賦值
- ✓ 解構賦值的樣式中不要包含空樣式(空物件或空陣列)
- ✓ 在函式的傳入參數或回傳值中作解構賦值時，優先使用物件

# ES6七大新特性

展開與其餘運算符





## 展開與其餘運算符(Spread Operator & Rest Operator)語法

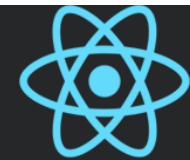
**展開運算符** - 展開一個陣列為一個個的獨立值，用於”陣列字面”與”函式呼叫”

```
const c = [...arr, b]  
f(...arr)
```

**其餘運算符** - 集合所有剩餘的值，組合成一個陣列。用於”函式傳入參數定義”與”解構賦值”

```
function f(...a) { }  
const [a, ...b] = [1, 2, 3]
```

註：其餘參數在傳入參數定義中，必定是位於最後一位，並且在參數中只能有一個其餘參數。



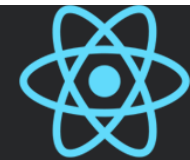
- ✓ 不要使用函式中的arguments物件，總是使用其餘參數語法來取代它
- ✓ 不要在展開運算符與其餘運算符後面有空格
- ✓ 用展開運算符來作拷貝陣列，取代函式的`apply`與陣列的`concat`的語法

# ES6七大新特性

類別



## 類別(Class)語法



原型物件導向的語法糖，轉換ES5語法後為函式。目的是提供一種用於自訂類型、物件的繼承與擴充時，更簡便的語法。

```
class MyClass extends Parent{  
  constructor( ) {  
    super( )  
  }  
}
```

註：類別是React用於撰寫元件的主要語法，注意其中有部份是超出ES6標準(ES7+)的語法。



- ✓ 在命名類別時，使用大駝峰(PascalCase)命名方式
- ✓ 撰寫自訂的toString()方法時要確保它是可以運作的，而且不會產生副作用
- ✓ 不要使用JavaScript中的getters/setters，可能有不預期的副作用，不易測試與維護

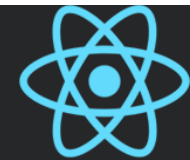
# ES6七大新特性

模組系統





## 模組(Module)語法



組織與管理程式碼檔案的重要語法，搭配npm工具，方便使用與導入現有模組。

```
// 模組輸出的檔案
```

```
export default MyClass
```

```
// 模組導入的檔案
```

```
import MyClass from './MyClass'
```

註：最新的瀏覽器內建支援情況都尚未完整，要先用如babel工具進行編譯/打包才能使用。



- ✓ 不要使用萬用符(wildcard, \*)作導入模組
- ✓ 從一個位置只作一次導入
- ✓ 導入語句都放在程式碼檔案中其他語句的上面
- ✓ 如果在模組中只需要單一個輸出，優先使用預設(default)輸出的語法