# Reference

1. 從ES6開始的JavaScrip學習生活 (<https://eyesofkids.gitbooks.io/javascript-start-from-es6/content/>)

# 7/24

1. JaveScrip特殊的原始資料類型
   1. null
   2. undefined
   3. Symbol

null會用來當作一種運算後的特殊情況回傳值。而未定義(undefined)則常使用的是加上typeof運算符的判斷方式。

1. JavaScript採用了與其他程式語言不同的邏輯運算的回傳值設計，我們把這兩個運算符稱為"短路求值(Short-circuit)"的運算符。實際上JavaScript中，在經過邏輯與(&&)與邏輯或(||)的運算後，它的回傳值是 - 最後的值(Last value)

邏輯或(Logical OR)(||)運算符在運算時，如果當第1個運算子為"falsy"時，則回傳第2個運算子。否則，將會回傳第1個運算子。

console.log('foo' || 'bar') // 'foo'

console.log(false || 'bar') // 'bar'

console.log( 0 || '' || 5 || 'bar') //5

console.log(false || null || '' || 0 || NaN || 'Hello' || undefined)

1. 二個等號(==)時稱為相等(equal)，它是比較運算符號，用於比較兩個變數or常數的其中的值是否相等。它的反義符號是(!=)。

三個等號(===)時稱為(identity)，它也是比較運算符號，用於比較兩個變數or常數是否完全相等(包含類型與值)。它的反義符號是(!==)。

1. switch有一種講法是相當於多個if...else的組合語句，不過這個理解是有些問題。實際上，它最常被使用的是在完全一致相等值(===)比較的情況下。

# 7/25

1. for...of語句 (類似foreach)

for...of語句是新的ES6語句，可以用於可迭代物件上，取出其中的值，可迭代物件包含陣列、字串、Map物件、Set物件等等。簡單的範例如下:

let aArray = [1, 2, 3]

for (let value of aArray) {

console.log(value)

}

let aString = "abcd";

for (let value of aString) {

console.log(value);

}

1. Javascrip的函式可分為兩種  
   1. 匿名函式 function() {}

2. 具名稱函式 function foo() {}

匿名函式並沒有函式的名稱，通常用來當作一個指定值，指定給一個變數/常數，被指定後這個變數/常數名稱，就成了這個函式的名稱。

//函式定義 - 使用有名稱的函式

function sum(a, b){

return a+b

}

//函式表達式 - 常數指定為匿名函式

const sum = function(a, b) {

return a+b

}

1. ES6中有一種新式的函式語法，稱為"箭頭函式(Arrow Function)"，使用肥箭頭符號(Flat Arrow)(=>)，它是一種匿名函式的縮短寫法，下面這個寫法相當於上面的sum函式定義:

const sum = (a, b) => a + b

1. 在javascrip函式可做為傳入參數

在JavaScript中函式也可以當作實際傳入參數的值，將一個函式傳入到另一個函式中作為參數值，而且在函式的最後也可以回傳函式。這種函式的結構稱之為"高階函式(Higher-order function)"，是一種JavaScript程式語言的獨特特性。

習慣上，因為函式在定義時，它的傳入參數並沒辦法限定資料類型，所以當要定義傳入參數將會是個函式時，通常會用fn或func作為傳入參數名稱，以此作為辦別。

const addOne = function(value){

return value + 1

}

const addOneAndTwo = function(value, fn){

return fn(value) + 2

}

console.log(addOneAndTwo(10, addOne)) //13

1. 不固定傳入參數(Variadic)與其餘(rest)參數

下列範例中，函式sum函定義了三個傳入參數，但如果真正在呼叫函式時傳入的參數值(arguments)語定義的參數數量不同，會發生什麼情況？

沒有預設值的時候會視為undefined值，而多出來的情況，是會被直接略過。

function sum(x, y, z) {

return x+y+z

}

console.log(sum(1, 2, 3)) //6

console.log(sum(1, 2)) //NaN

console.log(sum(1, 2, 3, 4)) //6

console.log(sum('1', '2', '3')) //123

console.log(sum('1', '2')) //12undefined

console.log(sum('1', '2', '3', '4')) //123

1. 在ES6中加入了其餘參數(rest parameters)的新作法，它使用省略符號(ellipsis)(...)加在傳入參數名稱前面，其餘參數的傳入值是一個標準的陣列值，以下是一個範例:

function sum(...value) {

let total = 0

for (let i = 0 ; i< value.length; i++){

total += value[i]

}

return total

}

console.log(sum(1, 2, 3)) //6

console.log(sum(1, 2)) //3

console.log(sum(1, 2, 3, 4)) //10

console.log(sum('1', '2', '3')) //123

console.log(sum('1', '2')) //12

console.log(sum('1', '2', '3', '4')) //1234

1. 內部(巢狀)函式

函式中的語句中，還可以包含其他的函式，這稱為內部函式(inner)，或是巢狀函式(nested)的結構。以下為一個簡單的範例:

function addOuter(a, b) {

function addInner() {

return a + b

}

return addInner()

}

addOuter(1, 2) //3

1. 提升(Hoisting)

JavaScript語言中的一種執行階段時的特性，也是一種隱性機制。不過，沒先定義與指定值就使用，這絕對是個壞習慣是吧？變數/常數沒指定好就使用，結果一定是不是你要的。

var、let和const會被提升其定義，但指定的值不會一併提升上去，像下面這樣的程式碼:

console.log(x) //undefined

var x = 5

console.log(y) //undefined

let y = 5

最後的結果出乎意料，竟然只是沒指定值的undefined，而不是程式錯誤。實際上這程式碼裡的變數被提升(Hoisting)了，相當於:

var x

console.log(x)

x = 5

let y

console.log(y)

y = 5

1. 提升(Hoisting)特性整理
   1. 所有的定義(var, let, const, function, function\*, class)都會被提升
   2. 使用函式定義時，在函式區塊中的這些定義也會被提升到該區塊的最前面
   3. 當函式與變數/常數同名稱而提升時，函式的優先程度高於變數/常數。
2. 陣列的宣告方式

const aArray = []

const bArray = [1, 2, 3]

console.log(aArray.length) //0

aArray[0] = 1

aArray[1] = 2

aArray[2] = 3

aArray[2] = 5

console.log(typeof aArray) // object

console.log(aArray) // [1,2,5]

console.log(aArray.length) //3

console.log(aArray[3]) //undefined

1. 陣列易混淆的宣告方式

//這是錯誤示範

const aArray = [10]

實際上這相當於:

const aArray = []

aArray[0] = 10