* Callback function

Function can be another function’s variable => we call this function “Callback function”

* Execution context 執行環境

JS的每個檔案在開啟時都先經過執行環境

執行環境有2個階段:

CreationPhase(初始階段)、ExecutionPhase(執行階段)

* CreationPhase:

就像打基礎一樣，先有這些才能執行，在這個階段JS會做幾件事:

1. Windows物件被生成
2. 執行範圍被定義(scope chain)，依循著閉包(closure)原則
3. “this”這個關鍵字被生成(指向window object)
4. 程式碼中的function declaration以及var variable被儲存到記憶體當中(Hoisting)

【p.s let、const 不會被存進去】

硬log let const會跳Reference Error

硬log var 會跳undefine

因為var已經被存進去，只是還未進到executionphrase被assign

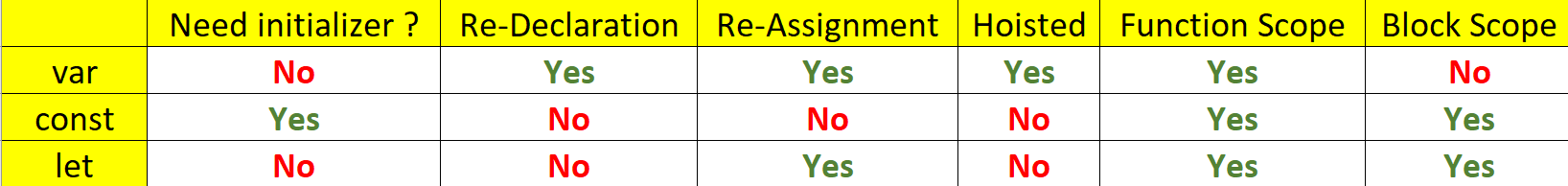
【p.s arrow function不會被存進去】

* ExecutionPhase:

1. 程式碼由上到下逐行執行

* Scope : Global scope、Function scope、Block scope
* Var let const

要不要先assin 是否重複宣告 是否重複賦值 是否提升



* Re-declaration 重新宣告

        var greeting = "say Hi";

        var greeting = "say Hello instead";

* Re-Assignment 重新賦值

        let greeting = "say Hi";

        greeting = "say Hello instead";

* Hoisted

Just like  var, let declarations are hoisted to the top.

Unlike var which is initialized as undefined, the let keyword is not initialized.

So if you try to use a let variable before declaration, you'll get a Reference Error.

其實var、let、const都會被hoisted, 但是只有var會在執行階段(creation phase)被JS納入記憶體內

* Lexical scoping

**內層區塊可以存取定義在外層區塊的變數。反過來說，外層區塊沒辦法存取內層區塊的變數。**

let name="Tom"

function printname(){

    console.log(name);

};

printname(); //Tom

像這樣，在執行printname()時要log出name的變數值，但是function內卻沒有name這個變數被定義，於是JS就會自動向外找，找到global scope有個name變數，所以log出來的結果是Tom.

* Block scope

*//var是一個很麻煩的函數，他有function-scope【func內定義的var,外面不能用】*

*//但是沒有block-scope(for-loop, if statement會影響 var的變數數值)*

        var x =1 ;

*for*(var x=0;x<10;x++){

            console.log(x);

*//在這邊for loop 會從0開始跳 0,1,2,3,...,9*

*//在跳到10的那一瞬間，x不符合for loop的規則*

*//log在印到9就停止，但是此時此刻 x=10*

*//這是為什麼外面的x=10*

        }

*//JS生成階段 會先知道有個var 宣告一個變數叫做x*

*//再來會一行一行往下執行*

*//先執行for-loop，執行的結果印出x=0~9，x的最終值為10*

*//由於var "不擁有" Block scope特性(裡面執行的x會被外面的看見)*

*//這也是為什麼外面的x =10!!*

        console.log(`The outside x=`+x);//10

* Closure(閉包)

<https://shubo.io/javascript-closure/>

https://shubo.io/javascript-closure/#lexical-environment-%E8%AA%9E%E5%BD%99%E7%92%B0%E5%A2%83

***在JavaScript中，即使在外層區塊已經回傳的狀況下，只要內層區塊還保留著一份參考，那麽外層區塊的環境不會隨著回傳而消失，我們依然可以存取外層環境中的變數。***

## Lexical Environment (語彙環境)

JavaScript 中，每段函數及區塊 (code block，也就是大括號 {} 圍起來的範圍) 都會對應到一個稱為 Lexical Environment (語彙環境) 的資料結構。

一段 code 要存取一個變數的時候，會先在自己的 environment record 裡面找，找不到會再往外層的 Lexical Environment 找，直到 global Lexical Environment 為止。

* closure經典例子:

只要還保有func 裡面設定的const 能夠保存下去(記憶當下的執行環境)，故重複執行時也會經由closure特殊性質，會有不同結果

const varGlobal = 'x'

function outer(paramOuter){

  const varOuter ='y'

  function inner(paramInner){

    const varInner ='z'

    console.log(varGlobal) *//x =>因為Lexical scoping，會向外找varGlobal*

    console.log(varOuter) *//y =>因為Lexical scoping，會向外找varOuter*

    console.log(varInner) *//z =>優先找到function scope內的varInner*

    console.log(paramOuter) *//a =>因為下面assign func=outer('c'), 所以印出帶入outer的'c'*

    console.log(paramInner) *//b =>因為func就代表inner這個函數，func('b')也就代表inner帶入的值是'b'*

  }

*return* inner *//function也是個物件，可以把物件丟出去!*

}

const func = outer('a'); *//這邊outer()回傳的會是inner這個 function物件*

*// console.log(func); //[Function: inner]*

func('b');

* closure模擬私有變數設定

*// 用閉包 (Closure) 模擬物件導向中的私有成員 (Private Member)*

*// 我們可以用 closure 的特性，模擬物件導向中的私有成員 (private member)。這個方法有時又被稱作 Module Pattern。*

*// 下面這個範例，我們創造一個 counter 物件，並提供三個方法存取物件內部的 count 變數。*

function makeCounter() {

    let count = 0;

    function changeBy(val) {

      count += val;

    }

*return* {

      increment: function() {

        changeBy(1)

      },

      decrement: function() {

        changeBy(-1)

      },

      value: function() {

*return* count

      }

    };

  };

  const counter = makeCounter();

  console.log(counter.value()); *// 0*

  counter.increment();

  counter.increment();

  console.log(counter.value()); *// 2*

  counter.decrement();

  console.log(counter.value()); *// 1*

*//   因為 closure 的特性，counter 物件的三個方法 increment()、decrement() 和 value() 能夠存取同一個語彙環境 (Lexical Environment)，所以這三個方法能夠存取 makeCounter() 中的同一個 count 變數及 changeBy() 函式。*

*// 透過呼叫這三個方法，我們能夠改變或讀取隱藏起來的 count 變數。*

*// 值得注意的是，除非透過 counter 物件上的 increment()、decrement() 或 value() 方法，我們沒辦法直接存取其內部的 count 變數。因此這裏的 count 就相當於物件導向中的私有成員變數 (private member)。*

Async與sync 異步與同步

* 要將函式改為異步

中使用了計時器APIsetTimeout會把傳入的回調函式進行異步執行，也就是先移到工作佇列中，等執行主執行緒的呼叫堆疊空了，在某個時間回到主執行緒再執行。所以即使它的時間設定為0秒，裡面的回調函式並不是立即執行，而是會暫緩(延時)執行的一種回調函式，一般稱為異步回調函式。

* Spread sheet

//Also Spread operator can concate array

let friends1 = ["Johane", "Jasmin", "Mary"];

//let friends2 = ["John", "Tom", "Mark"];

//let friends = [...friends1, ...friends2];

let newfriend = "Anthen";

let friends = [...friends1, ...newfriend]; 你也可以用Spread sheet 串接只有一個String

唯一不同的是，你如果只是傳接單一個String, 前面可不用加上Spread sheet

console.log(friends); //這麼一來就會串接成一個array了，要注意順序，寫前面的放前面+

* Array的刪除某個特定值:

透過filter方法

Ex:

幾個關鍵要注意

1. 你必須讓原本的arr被re-assignment(重新賦值)

arr = arr.filter()

1. filter()這個method裡面要放入一個(判別式)，.filter()會自動遍歷你array裡的每一個元素，

只要元素符合判別式，該元素會被回傳，反之刪除!

let arr = ["aaa", "bbb", "ccc"]; //假設我想刪除陣列中的”bbb”

arr = arr.filter((e) => { //re-assignment

*return* e !== "bbb"; //條件式注意: 1. 記得要寫return

2. 條件判別為true就會被留下

});

console.log(arr);

同樣的效果!，但filter()中的callback function做省略 (沒有大括號、沒有return)

let arr = ["aaa", "bbb", "ccc"];

arr = arr.filter((e) => e !== "bbb"); //

console.log(arr);

套一個function 可以自由的輸入想刪除的值!

let arr = ["aaa", "bbb", "ccc"];

function doFilter(data) { //

  arr = arr.filter((e) => {

*return* e !== data;

  });

*return* arr;

}

console.log(doFilter("ccc"));

* 邏輯運算子(&& ||)的特別用法

let a = NaN;

let b = 100;

a = 10;

console.log(a && b);

*/\*\**

*> &&如何運作*

*ex: a && b && c*

*從左到右有三個變數，分別是a,b,c*

*&&代表的是and*

*把這三個(a && b && c)寫在一起的意思會是:*

*從左到右三個變數的布林回傳值如果都是true=>回傳最後一個值(c)*

*一但有其中一個布林回傳值是false的時候=>回傳第一個FALSE的值*

*應用例子:*

*let a = NaN;*

*let b = 100;*

*console.log(a && b); //NaN*

*應用例子2:*

*假設data是一個API回傳的資料(.json)，如果資料庫找不到這筆資料*

*這個data回傳值就會是null*

*這種時候就可以使用 && 串接*

*可以省去寫一大堆if...else declaration*

*{data && data.map((d) => {return d})}*

*//上面這行代表:*

*如果data是null, 回傳null*

*如果data有值，就用.map()這個method去做遍歷*

*> ||如何運作*

*基本上就是跟&&相反：從左邊到右邊，回傳第一個是truthy的值，若全部皆為falsy，則回傳最後一個值。*

*延伸例子1*

*const list = JSON.parse(localStorage.getItem('someKey')) || []*

*延伸例子2*

*const numbers = null*

*const numbersTwo = [1,2,3]*

*function multiplyTen (numbers) {*

*return numbers !== null && numbers.length ? numbers.map(number => number \* 10) : []*

*}*

*multiplyTen(numbers) // []*

*multiplyTen(numbersTwo) // [10,20,30]*

*\*/*

\*有關array.map()

原則map()要放入一個Callback function執行

以下程式碼示範如何使用帶有一個參數的函式來操作 map。這個參數會自動地逐一取出原始陣列各個元素來使用。

var numbers = [1, 4, 9];

var doubles = numbers.map(function(num) {

↑這邊可以看到map中的callback func，其參數num就是代表每個被迭代的元素(有點類似for-loop中的i 那樣的感覺，代表每一筆資料)

return num \* 2;

});

// doubles 現在是 [2, 8, 18]

// numbers 仍然是 [1, 4, 9]

https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\_Objects/Array/map

範例

把一個數字陣列轉換成對應的開根號後的數字陣列

以下的程式碼把一個數字陣列(array of numbers) 轉換成一個 新的以該數字陣列裡的一個個數做開根號計算的數字陣列.

var numbers = [1, 4, 9];

var roots = numbers.map(Math.sqrt); //map會return一個新的array

// roots 現在是 [1, 2, 3]

/\* numbers 還是 [1, 4, 9]，這證明了 map() 不會去變動到 numbers 的值，

map 內部是做了 immutable 的機制，Array.prototype 底下的這些高階函式

大多都具有這樣函數式編程裡非常注重的特性 - immutable，不會去改變資料

來源本身原有的值

\*/

Copy to Clipboard

使用 map 將陣列中的物件變更格式

以下程式碼取出一陣列，將其中物件變更格式後建立為一個新的陣列並傳回。

var kvArray = [{key: 1, value: 10},

{key: 2, value: 20},

{key: 3, value: 30}];

var reformattedArray = kvArray.map(function(obj) {

var rObj = {};

rObj[obj.key] = obj.value;

return rObj;

});

// reformattedArray 現在是 [{1: 10}, {2: 20}, {3: 30}],

// kvArray 仍然是：

// [{key: 1, value: 10},

// {key: 2, value: 20},

// {key: 3, value: 30}]

Copy to Clipboard

使用帶參數的函式將一數字陣列進行對應