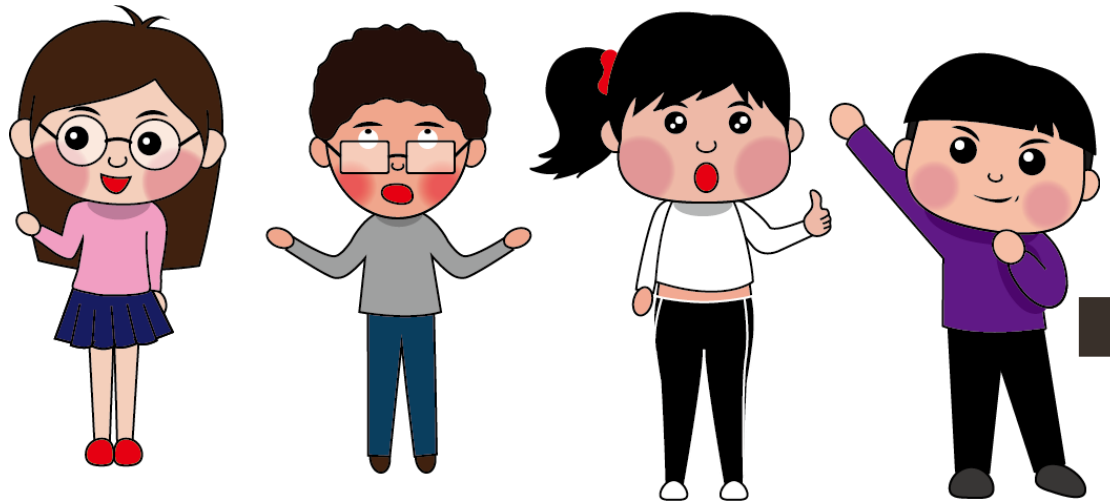
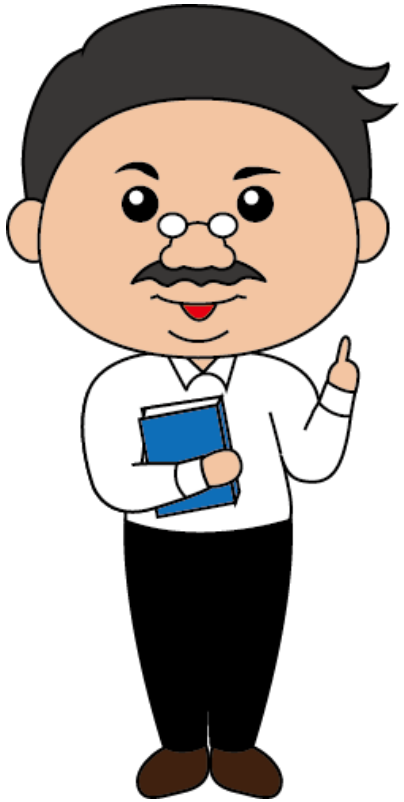
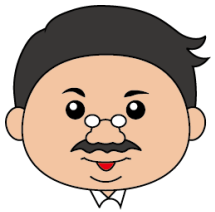
CH04
程式設計





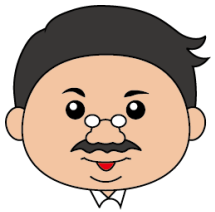
4-1
結構化程式設計實作

- 4-1-1　循序結構
- 4-1-2　重複結構
- 4-1-3　選擇結構
- 4-1-4　遞迴結構



4-1-1　循序結構

- 程式語言的執行是由上而下一行一行的執行，有些程式語言每一行程式必須以分號「；」結束。
- 依序執行是程式的基本運作方式，稱為**循序結構(Sequence Structure)**。



4-1-1 循序結構

```
<script>
var summary;  //宣告變數
summary = 1+100;
summary = summary * 100 / 2;
alert(summary);
</script>
```

宣告summary變數，「//」代表註解，不會執行

↓

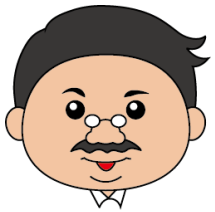
先進行1+100，然後將101指定給summary

↓

等號右側先進行101×100 / 2，再指定給summary

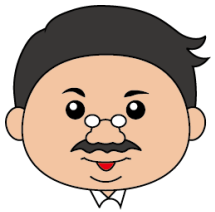
↓

顯示出結果(5050)



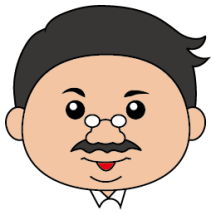
4-1-1　循序結構

- 每行程式執行中的「=」並不是數學的等號(左右相等)，而是指將「=」右邊的執行結果指定(設定)給左邊的變數。



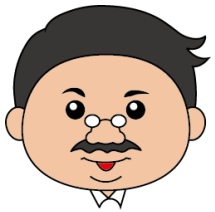
4-1-1　循序結構

- 另外程式中的乘號以*號代替，運算符號旁是否有空隔並不影響程式的執行「　summary=summary+1　」與「summary = summary + 1」是一樣的，summary是一個變數，變數是一個記憶體空間，用於儲存程式運作中的資料。



4-1-2 重複結構

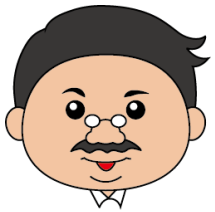
- 電腦的優點是可快速且重複的運算，因此每一種程式語言必然提供重複執行的語法，稱之為**重複結構**(Repetition Structure)，又稱為迴圈結構(Loop Structure)。
- 迴圈依迴圈結束的判斷方式，分成**計數式迴圈**(Count Controlled Loop)及**條件式迴圈**(Conditional Loop)。



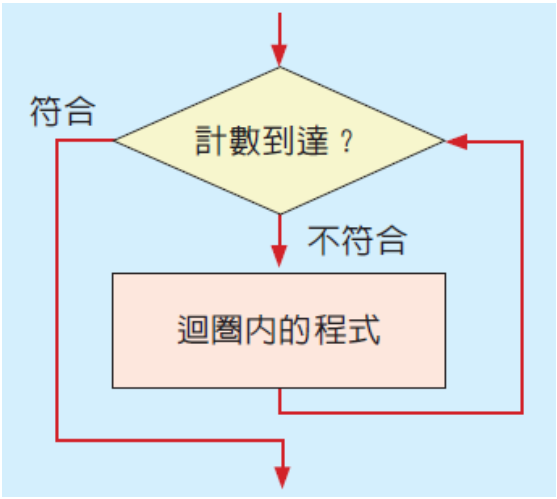
4-1-2　重複結構

計數式迴圈

- 計數式迴圈的執行次數，在迴圈開始時就決定了，下圖分別是類似於C語言(C-like)及VB程式語言的計數式迴圈語法，迴圈次數由變數i所決定，此例是1~100，共100次迴圈。

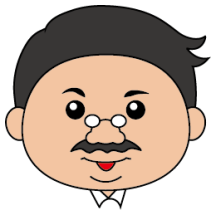


4-1-2　重複結構



C-like語法	VB語法

for(i=1 ; i<=100 ; i++)
{
　//迴圈內執行的程式
}
	For i=1 to 100 step 1
	　REM 迴圈內執行的程式
	　'迴圈執行的程式
	next

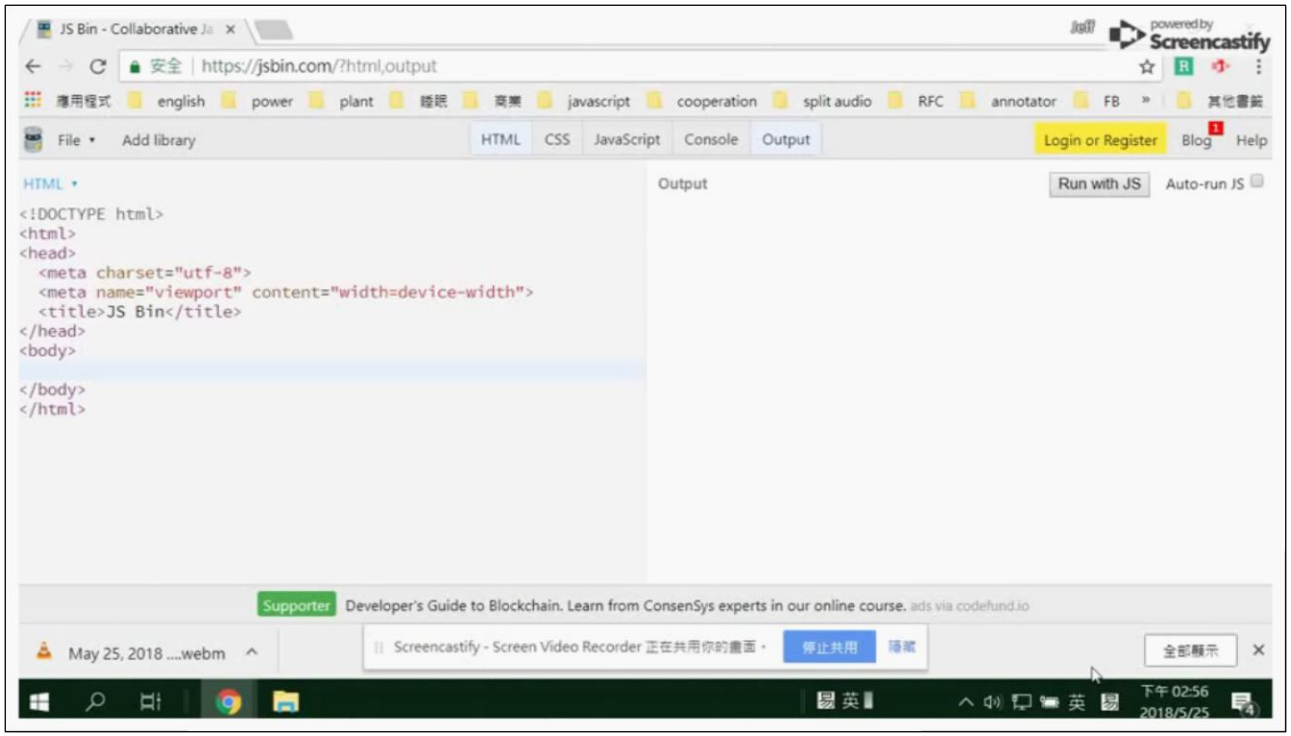


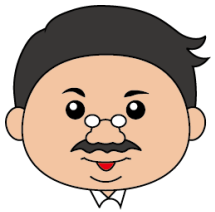
4-1-2　重複結構

■ 實作練習1：

計算10! (1×2×3……×9×10) 的值

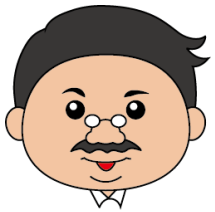
操作影片





4-1-2　重複結構

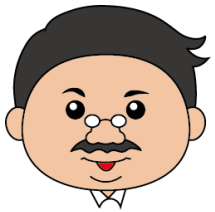
```
1. var n=10;
2. var summary=1;
3. for(i=1;i<=n;i++)
4. {
5.       summary=summary*i;
6. }
7. alert("10!="+summary);
8.
9. //alert;程式碼分大小寫,I為小寫
```



4-1-2　重複結構

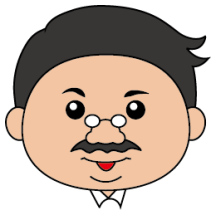
程式說明

- 第1、2行：宣告一個n及summary的變數，並設定10給n，設定1給summary。
- 第3行：設定一個計數式迴圈，迴圈從i=1開始執行，在i小於或等於10的條件下繼續迴圈。i++是指每次迴圈執行後，i自動+1，i++的意思等於程式碼i=i+1。



4-1-2　重複結構

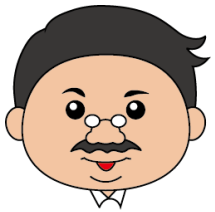
- 第4行：左大括號代表迴圈程式碼開始。
- 第5行：summary是執行的結果，「summary=summary*i」是把「summary×i」之後，再存回至summary變數。
- 第6行：右大括號代表迴圈程式碼結束。



4-1-2　重複結構

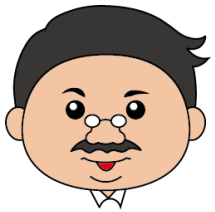
- 第7行：顯示文字「10! = 」再串接summary的值，此程式中「+」代表字串的串接，而非加法。

  alert是顯示訊息的函式。JavaScript是類似於C語言的網頁使用者端(client side)程式，程式碼中字母大小寫有區分。alert不可打成ALERT。



4-1-2　重複結構

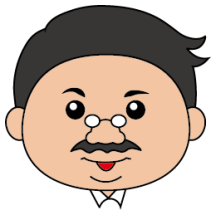
- 第8行：空行及空白鍵不影響執行。
- 第9行：雙斜線：「//」代表註解符號，程式不會執行「//」及其後的文字註解。



4-1-2 重複結構

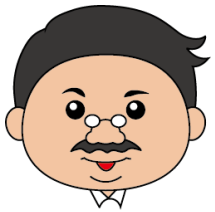
執行說明

1. JsBin是一個線上編寫HTML+CSS+JavaScript程式的界面。
2. 網頁左側的HTML區可以編寫網頁內容，其中可設定物件及其識別碼(id)，以供JavaScript控制用。例如：「document.getElementById("show_area").innerHTML=summary;」，即可將此例的執行結果，顯示在show_area這個物件中。



4-1-2　重複結構

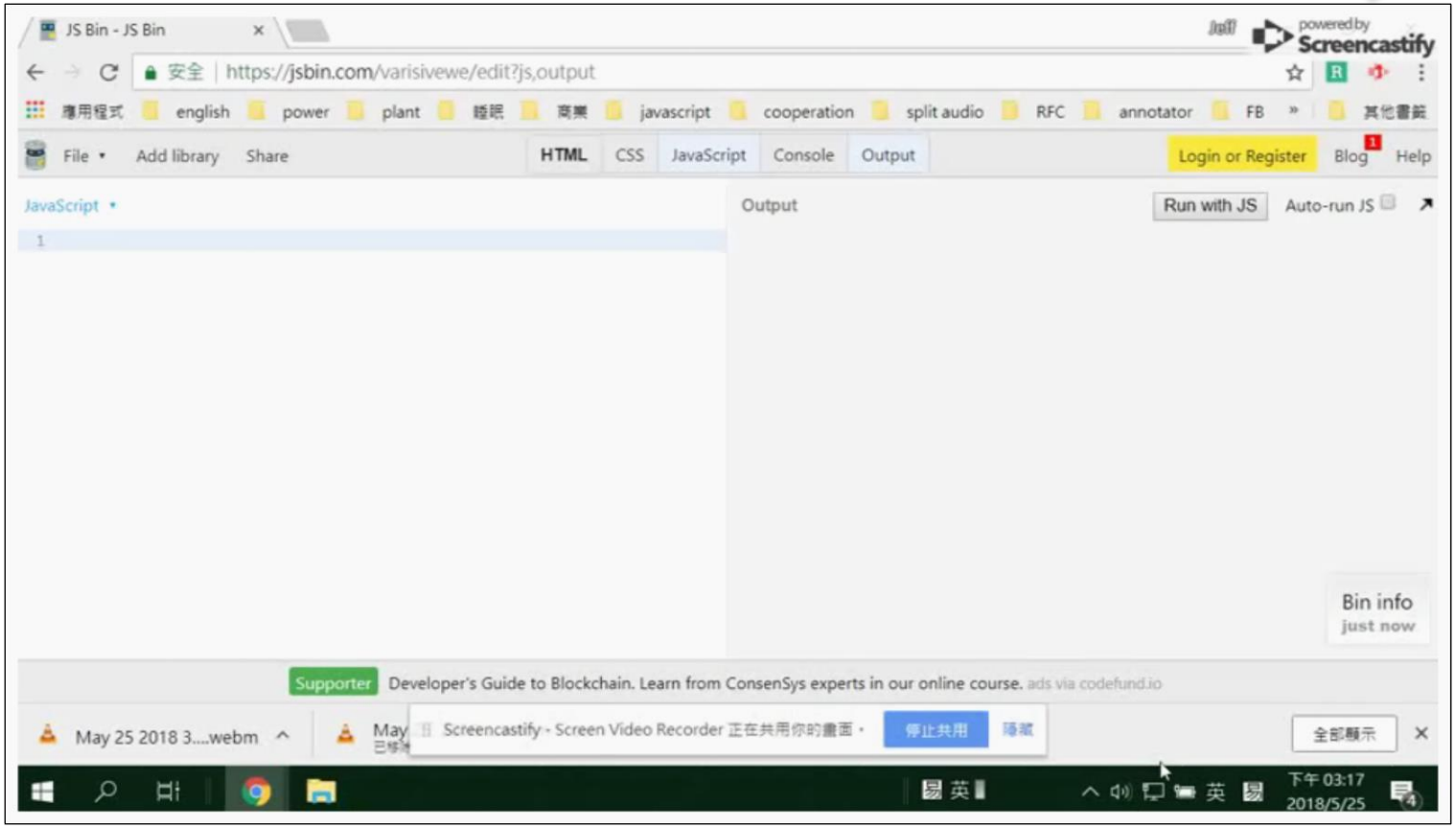
3. CSS可設定網頁各物件的樣式。
4. 右側Output區會顯示HTML+CSS+JavaScript的整個執行結果。
5. Console區可以觀看程式測試的輸出情況。

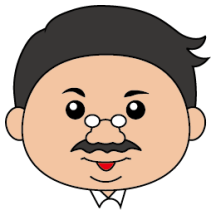


4-1-2　重複結構

- 實作練習2：列出陣列的值

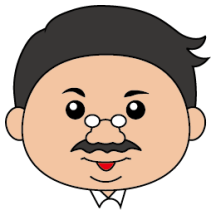
操作影片





4-1-2　重複結構

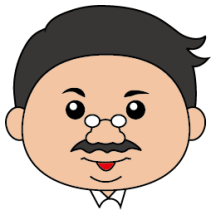
```
1. var array=[1,3,5,7,9,"a","b","c"];


2. for(i=0;i<array.length;i++)
3.
4.   console.log(array[i]);
5. //不用{}
```



4-1-2　重複結構

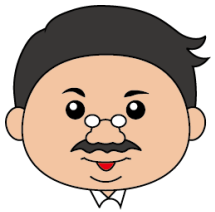
程式說明

- 第1行：宣告一個陣列變數，名稱為array，陣列內容(元素)共8個(索引值0~7)，每個內容以逗號「,」隔開，字串以雙引號「"」括住。
- 第3行：設定一個固定迴圈，迴圈從i=0開始執行，在i小於array.length下繼續執行，array.length是指array陣列的長度，也就是此array的空間大小(8)，i每次+1。



4-1-2　重複結構

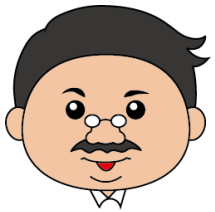
- 第4行：此處沒有代表迴圈開始的左大括號，表示此迴圈的程式碼只有一行(第5行)，此時迴圈內的程式無需用左右大括號括住。



4-1-2 重複結構

- 第5行：每次迴圈在Console中印出array[i]的內容。i是陣列的索引，例如：第一次迴圈array[0]，此陣列空間的內容為1。Console是指程式的控制台，一般用於觀看程式的執行結果。

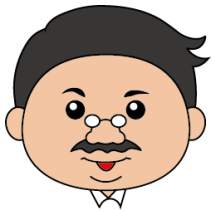
i	0	1	2	3	4	5	6	7
array	1	3	5	7	9	a	b	c



4-1-2　重複結構

條件式迴圈

- 條件式迴圈的迴圈執行次數不確定，條件符合下，迴圈會不斷的執行。
- C語言(C-like)及VB條件式迴圈的語法。

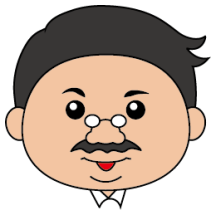


4-1-2　重複結構

C-like語法	VB語法

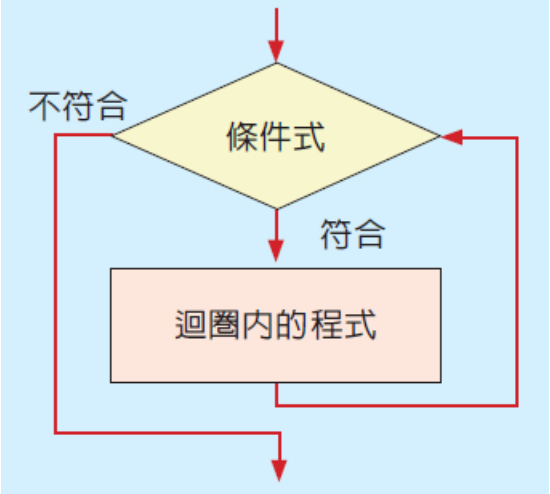
while(條件式)
{
　//迴圈內執行的程式
}
	While條件式
	　'迴圈內執行的程式
	End while

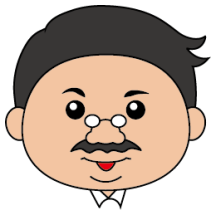
do
{
　//迴圈內執行的程式,至少執行一次
}while(條件式)
	Do
	　'迴圈內執行的程式,至少執行一次
	Loop while 條件式



4-1-2　重複結構

- while迴圈會先判斷條件是否成立，成立後再執行迴圈中的程式，do迴圈則會先執行一次迴圈再判斷條件式是否成立，所以至少執行一次迴圈中的程式。

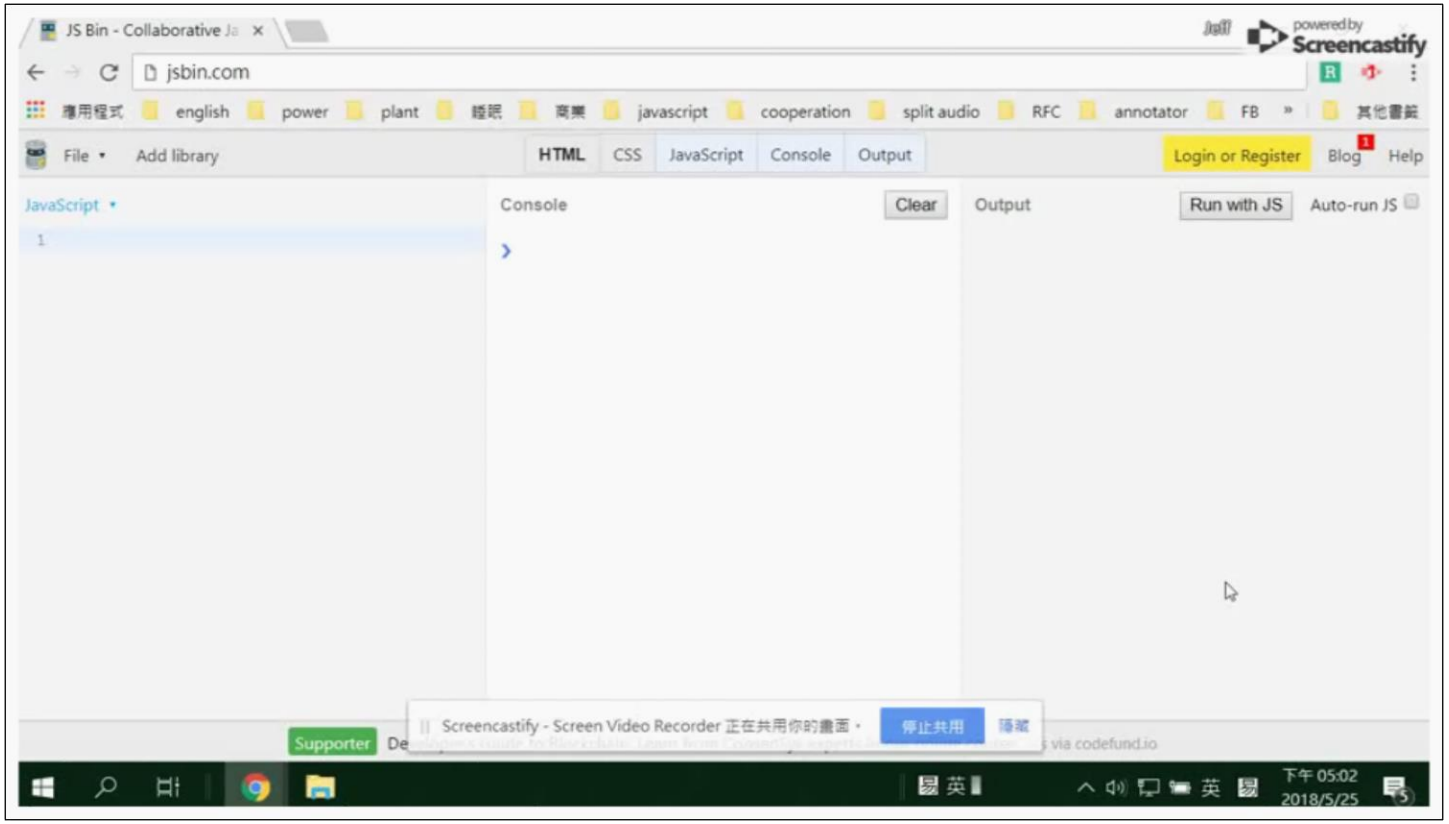


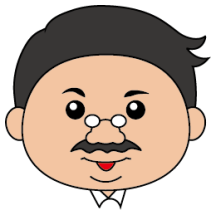


4-1-2　重複結構

- 實作練習3：(while)列出每一個陣列元素值，直到負數值出現

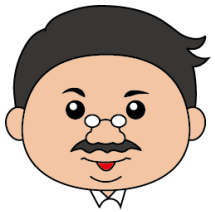
操作影片





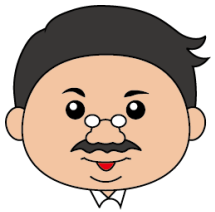
4-1-2 重複結構

程式	說明
1 var data = [1,3,5,7,-9,11,13,15];	宣告一個data陣列及8個值
2 var i=0;	宣告一個索引變數
3	
4 while (data[i]>0)	data[i]>0是迴圈的判斷式
5 {	
6 console.log(data[i]);	在控制台印出非負數值
7 i++;	陣列索引每次加1
8 }	
9 console.log(data[i]);	在控制台印出第一個負數值



4-1-2　重複結構

- 使用條件式迴圈，必須注意條件變數必須在迴圈中改變，否則會形成無窮迴圈，造成程式當機。
- 以此例來說，如果在陣列中完全沒有負數， 迴圈將會一直執行，陣列索引變數i會持續增加而超出陣列長度，將造成程式錯誤。

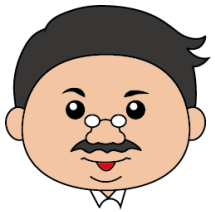


4-1-2　重複結構

- 我們可以適當修改條件式為：

  「data[i]>0 && i<data.length」

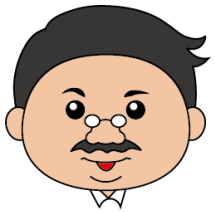
- 其中「&&」是「而且」的邏輯運算子，必須「data[i]>0」而且「i<data.length」，程式才會繼續執行迴圈，任何一個條件不符合，程式即會跳出迴圈，以免形成無窮迴圈而當機。



4-1-2 重複結構

- 「>=」、「<=」、「==」及「!=」四項稱為關係運算子，「||」及「&&」為邏輯運算子。

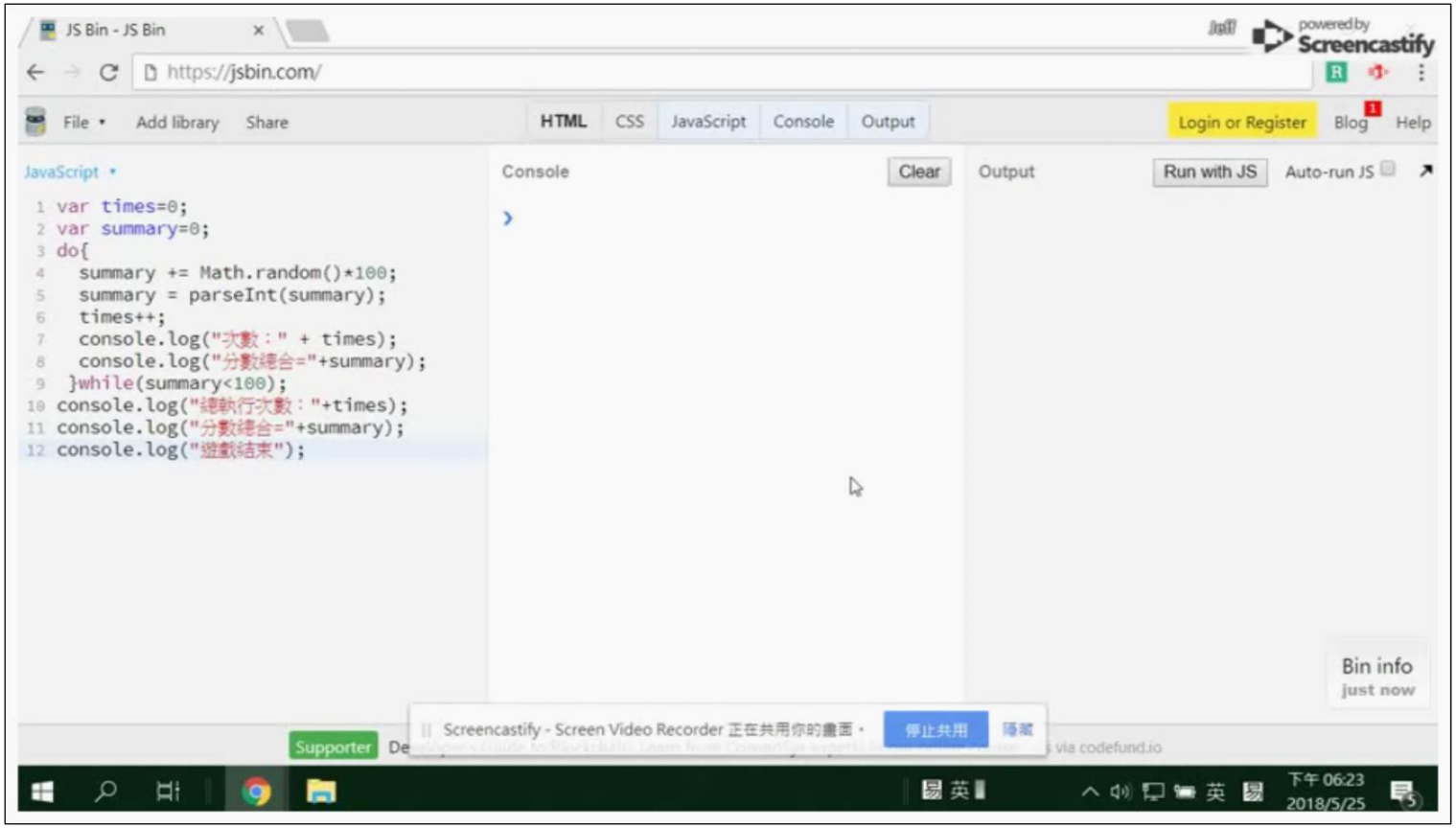
C-like	VB	功能	C-like條件式範例	範例說明
>=	>=	大於等於	while(a >= b) {}	假如a大於或等於b，就執行迴圈
<=	<=	小於等於	while(a <= b) {}	假如a小於或等於b，就執行迴圈
==	=	等於	while(a == b) {}	假如a等於b，就執行迴圈
!=	<>	不等於	while(a != b) {}	假如a不等於b，就執行迴圈
||	Or	或	while(a>0 || b>0) {}	假如a>0或b>0，就執行迴圈
&&	And	及	while(a>0 && b>0) {}	假如a>0而且b>0，就執行迴圈

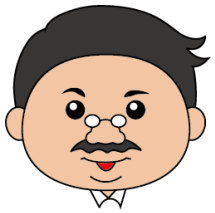


4-1-2 重複結構

操作影片

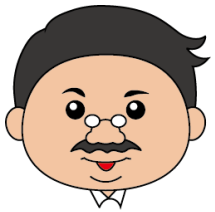
- 實作練習4：(do while)數字遊戲





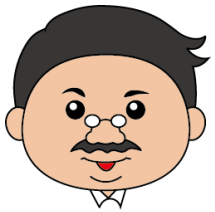
4-1-2　重複結構

程式	說明
1 var times=0;	宣告執行次數的變數times
2 var summary=0;	宣告總合分數的變數summary
3 do{	do 迴圈內至少執行一次
4　summary += Math.random()*100;	Math.random是數學亂數函式，產生0~0.999
5　summary = parseInt(summary);	parseInt是取整數，將summary小數去掉
6　times++;	執行次數加1
7　console.log("次數：" + times);	印出目前的次數
8　console.log("分數總合="+summary);	印出分數總合
9　} while(summary<100);	判斷總合分數，小於100則繼續迴圈
10 console.log("總執行次數："+times);	
11 console.log("分數總合="+summary);	
12 console.log("遊戲結束");	



4-1-3 選擇結構

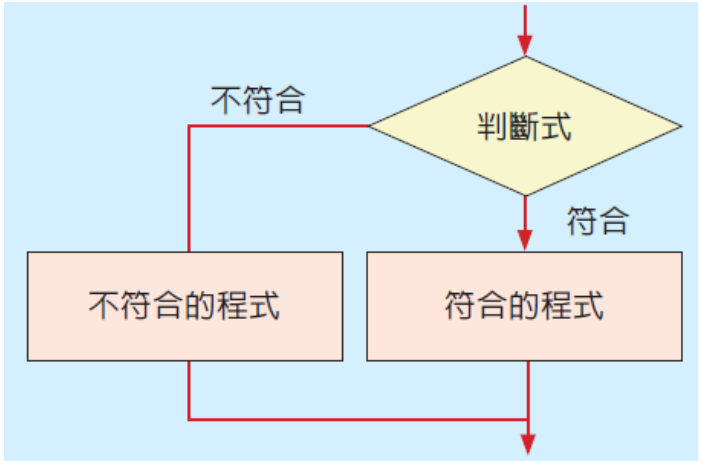
- **選擇結構**(Selection Structure)用條件式來做判斷，根據結果的真、假，來選擇執行哪些動作。
- 一般程式語言有兩種選擇結構的語法，一個是if，另一個是switch(或select case)，if適用於一種狀況的判斷，switch適合於多種條件下的選擇。

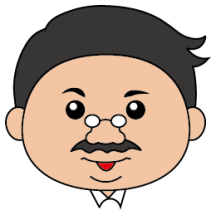


4-1-3 選擇結構

if

- if只有一個判斷式，當判斷式條件成立時執行某些程式，條件不成立時執行另外的程式，也可以在條件不成立時，不執行任何程式。



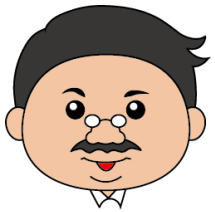


4-1-3　選擇結構

C-like語法	VB語法

if(判斷式)
{
　//判斷式成立時的程式
}
	If　判斷式　then
	　‘判斷式成立時的程式
	end if

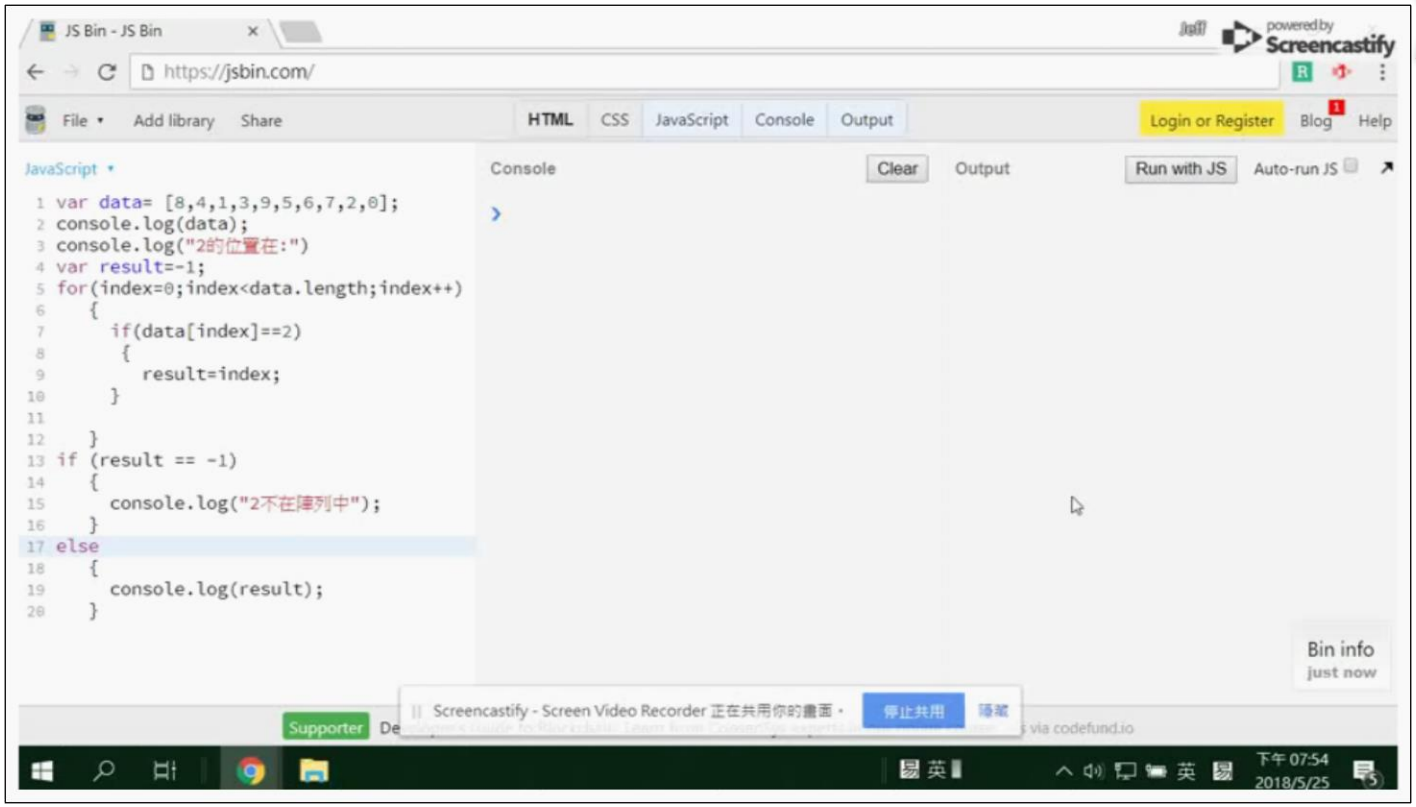
if(判斷式)
{
　//判斷式成立時的程式
}
else
{
　//判斷式不成立時的程式
}
	If　判斷式　then
	　‘判斷式成立時的程式
	else
	　‘判斷式不成立時的程式
	end if

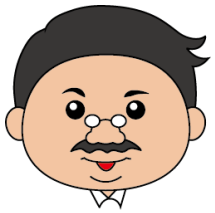


4-1-3　選擇結構

- 實作練習5：循序搜尋陣列中某一元素的值，印出該值所在的陣列索引值

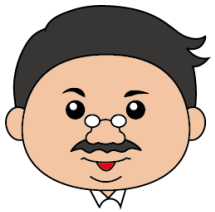
操作影片





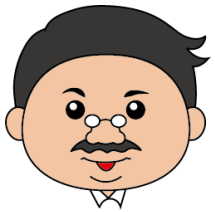
4-1-3 選擇結構

程式碼	程式說明
1 var data= [8,4,1,3,9,5,6,7,2,0];	宣告一個陣列變數data
2 console.log(data);	在Console中印出陣列內容
3 console.log("2的位置在:")	在Console中印出："2的位置在:"的字串
4 var result=-1;	宣告result變數，做為查詢結果
5 for(index=0;index<data.length;index++)	index是陣列索引
6 {	
7 if(data[index]==2)	data[index]==2是判斷式，兩個=，不是一個
8 {	判斷成立後的程式開始
9 result=index;	找到後將索引位置設定給result
10	



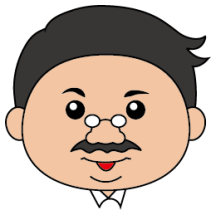
4-1-3 選擇結構

程式碼	程式說明
1 var data= [8,4,1,3,9,5,6,7,2,0];	宣告一個陣列變數data
2 console.log(data);	在Console中印出陣列內容
3 console.log("2的位置在:")	在Console中印出："2的位置在:"的字串
4 var result=-1;	宣告result變數，做為查詢結果
5 for(index=0;index<data.length;index++)	index是陣列索引
6 {	
7 if(data[index]==2)	data[index]==2是判斷式，兩個=，不是一個
8 {	判斷成立後的程式開始
9 result=index;	找到後將索引位置設定給result
10	



4-1-3 選擇結構

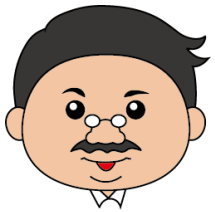
程式碼	程式說明
11 }	判斷成立後的程式結束
12 }	
13 if (result == -1)	判斷result是否等於-1，-1代表未找到
14 {	
15 console.log("2不在陣列中");	判斷式成立的程式(未找到)
16 }	
17 else	
18 {	
19 console.log(result);	判斷式不成立的程式(找到，印出位置)
20 }	



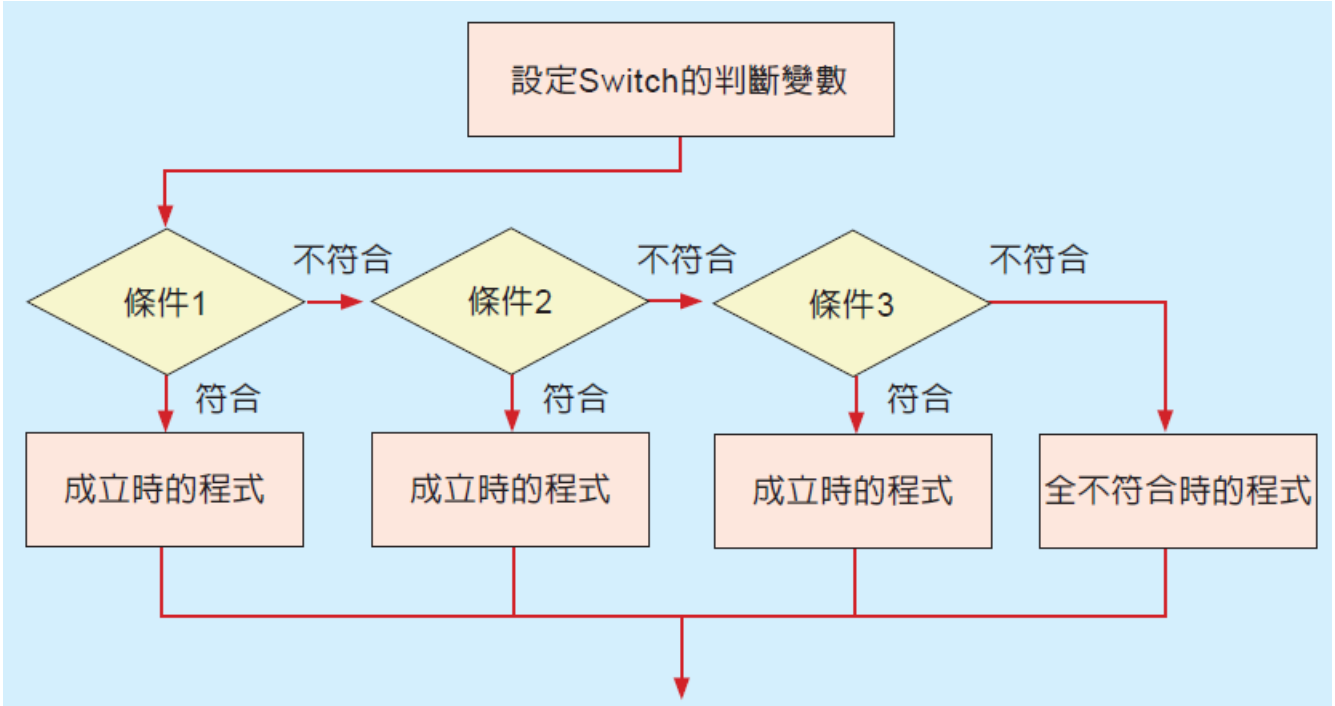
4-1-3 選擇結構

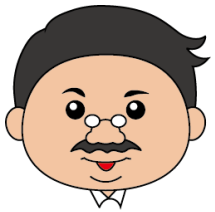
switch

- Switch針對不同的情況，有數個判斷條件，符合其中一個條件，就執行該條件下的程式，所有條件都不成立時，就執行預設(default)的程式。



4-1-3 選擇結構

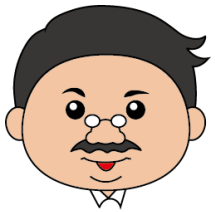

設定Switch的判斷變數
條件1
不符合
條件2
不符合
條件3
不符合
符合
符合
符合
成立時的程式
成立時的程式
成立時的程式
全不符合時的程式



4-1-3 選擇結構

C-like語法	VB語法

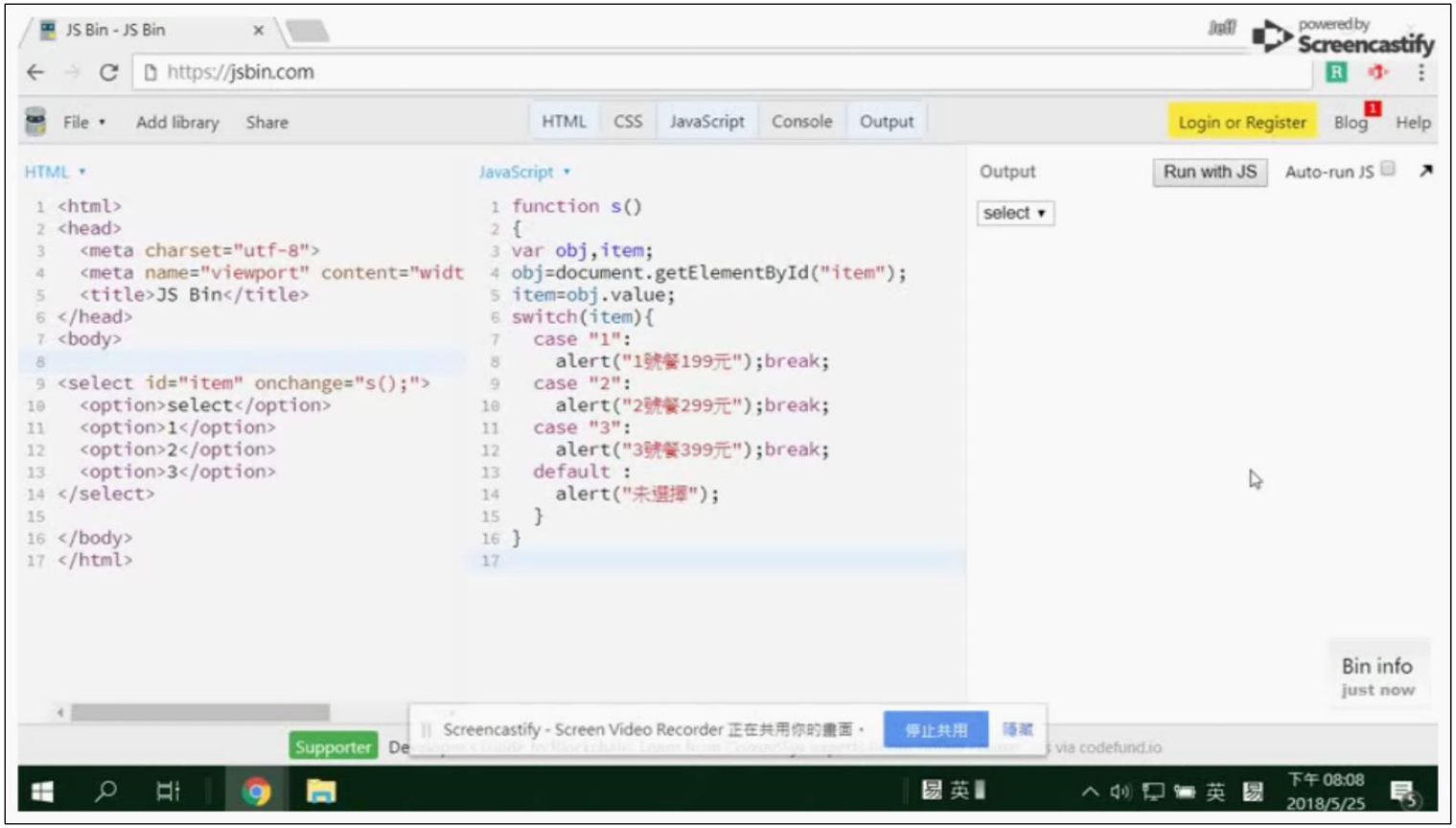
switch(判斷的變數)
{
case 1:
//條件1成立時的程式
break; //條件程式結束
case 2:
//條件2成立時的程式
break;
case 3:
//條件3成立時的程式
break;
default:
//條件全不符合時的程式
}
	Select case判斷的變數
	Case 1
	‘條件1成立時的程式
	Case 2
	‘條件2成立時的程式
	Case3
	‘條件3成立時的程式
	Case else
	‘條件全不符合時的程式
	End select

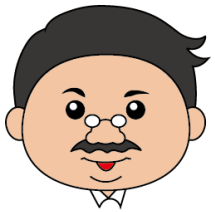


4-1-3　選擇結構

操作影片

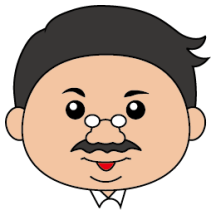
- 實作練習6：123點餐





4-1-3 選擇結構

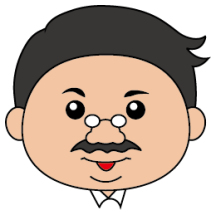
```
1.  function s()
2.  {
3.  var obj,item;
4.  obj=document.getElementById("item");
5.  item=obj.value;
6.  switch(item){
7.    case "1":
8.      alert("1號餐199元");break;
9.    case "2":
10.     alert("2號餐299元");break;
11.   case "3":
12.     alert("3號餐399元");break;
13.   default :
14.     alert("未選擇");
15.   }
16. }
```



4-1-3　選擇結構

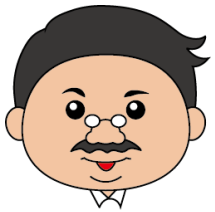
程式說明

- 第1行：函式名稱s。
- 第2行：函式的程式開始。
- 第3行：宣告obj及item兩個變數。
- 第4行：將網頁元件item設定給obj物件變數。document是指網頁文件，getElevmentById的功能是依物件的id來抓取到該物件。



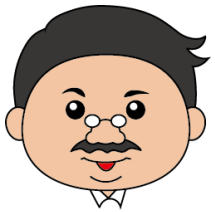
4-1-3　選擇結構

- 第5行：將obj物件的值(使用者點選的)，設定給item變數。
- 第6行：switch結構以item變數為判斷條件。
- 第7行：item="1"的情況。



4-1-3　選擇結構

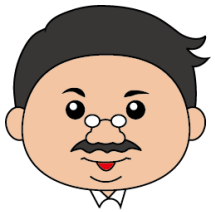
- 第8行：顯示 "1號餐199元" 的文字，「break;」語法代表這種情況的程式結束，「break;」也可以獨立成一行。
- 第13行：代表所有條件都不符合的情況。
- 第14行：都不符合條件(使用者選擇「select」)時，顯示"未選擇"的字樣。



4-1-4　遞迴結構

分治法

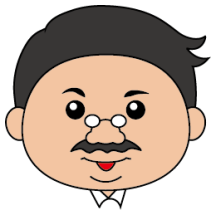
- 分治法(Divide and Conquer)是很重要的演算法，它其實就是將一個大問題分解成一個個小問題，解決所有的小問題之後，自然就完成了大問題的解答。



4-1-4 遞迴結構

- 以n!為例，要算出5!的答案，要先算出4!的解答，而要算出4!，也要先算出3!的答案，這就是分治法的觀念。

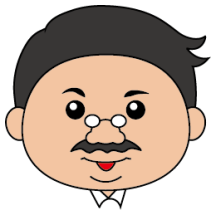
$$5! = 5 \times 4! = 5 \times (4 \times 3 \times 2 \times 1)$$
$$4! = 4 \times 3!$$
$$3! = 3 \times 2!$$
$$2! = 2 \times 1!$$
$$1! = 1$$



4-1-4 遞迴結構

遞迴法

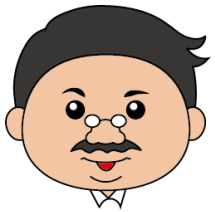
- **遞迴法(Recursive)是指函式呼叫函式本身的程式執行方式，遞迴法可以讓程式更簡潔，減少一些變數的宣告，一些複雜演算法的實現常採用遞迴法，讓演算法更易懂，然而因遞迴必須讓函式不斷的自我呼叫，一般而言，遞迴法比迴圈法的程式執行效率低。**
- **分治法常使用遞迴來實現。**



4-1-4 遞迴結構

- 遞迴法與迴圈法的比較

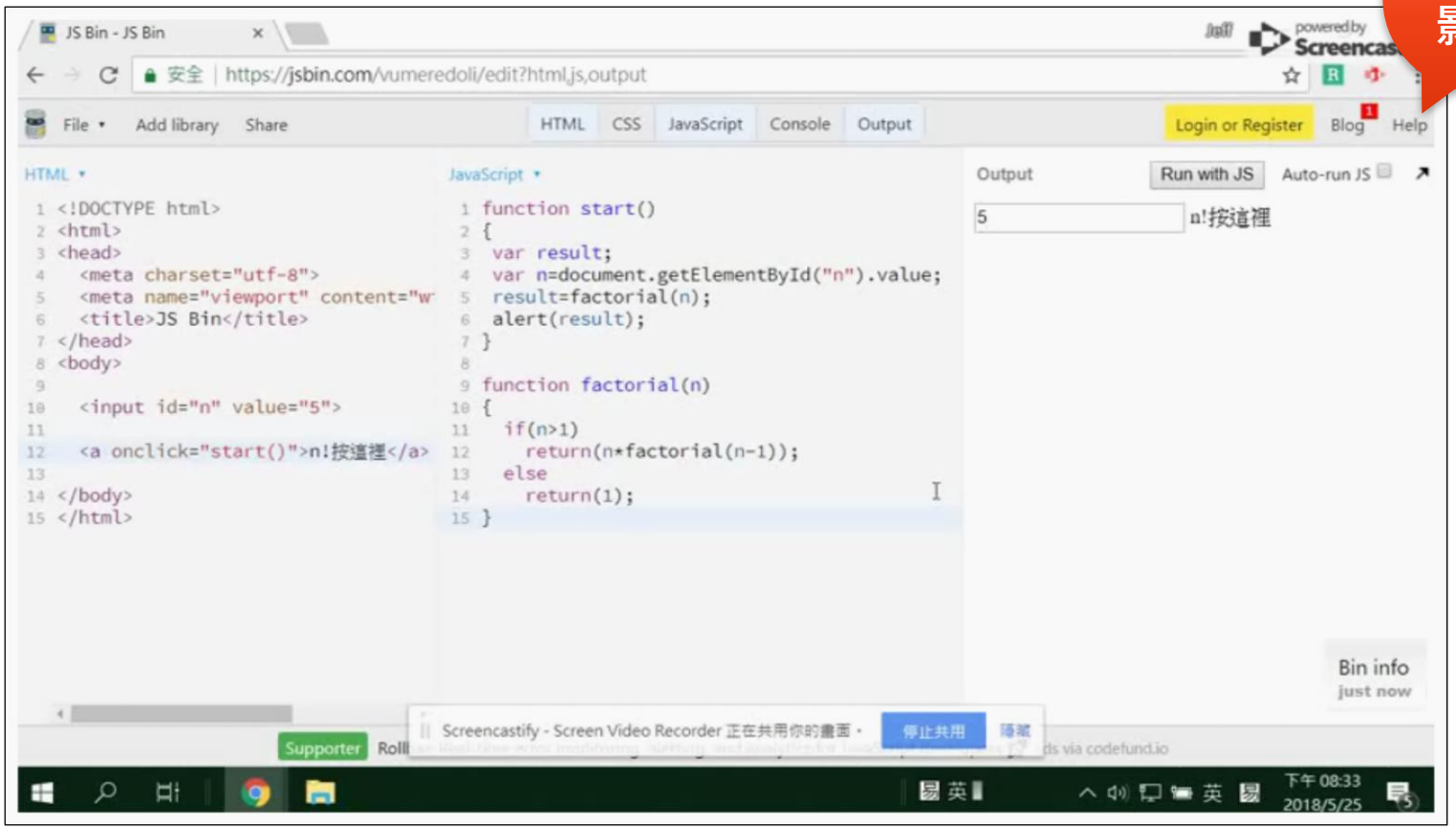
遞迴法	迴圈法
1 alert(recursive(100));	1 alert(sum(100));
2.	2
3 function recursive(counter)	3 function sum(counter)
4 {	4 {var sum_value=0;
5 if(counter>1)	5 for(i =0;i<=counter;i++)
6 return(counter +recursive(counter-1));	6 sum_value += i
7 else	7 return sum_value;
8 return(1);	8 }
9 }	

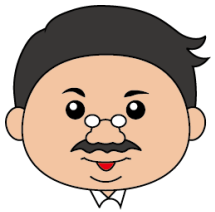


4-1-4 遞迴結構

- 實作練習7：遞迴計算n!(1×2×3……×n-1×n)的值

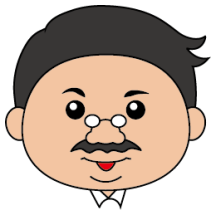
操作影片





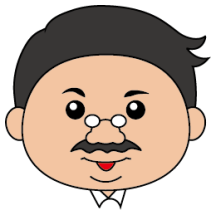
4-1-4 遞迴結構

程式	說明
function start()	超連結按鈕的事件程序，名稱：start
{	
var result;	設定一個計算結果的變數：result
var n= document.getElementById("n").value;	設定變數n，並取得輸入框的值(value)
result = factorial(n);	執行階層函式factorial(n)，並傳回結果給result
alert(result);	顯示結果
}	
function factorial(n)	階層函式：factorial(n)，n是傳入要計算的階層數



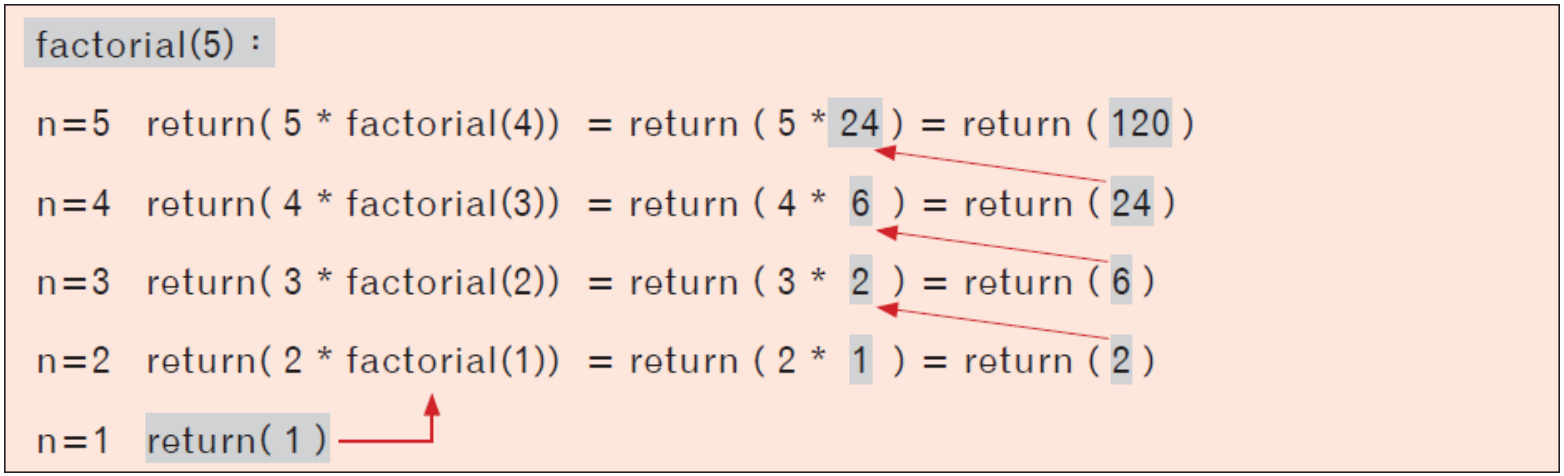
4-1-4 遞迴結構

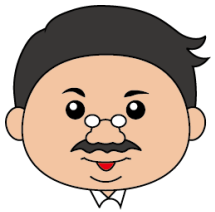
程式	說明
function factorial(n)	階層函式：factorial(n)，n是傳入要計算的階層數
{	
if(n>1)	判斷n是否大於1
return(n*factorial(n-1));	是：再遞迴執行factorial(n-1)。傳入n-1 當factorial(n-1)傳回後，再傳回n*factorial(n-1)的計算結果給原呼叫的程式
else	否則(n小於等於1)
return(1);	回傳1，程式結束
}	



4-1-4 遞迴結構

- 此例的呼叫及傳回情況

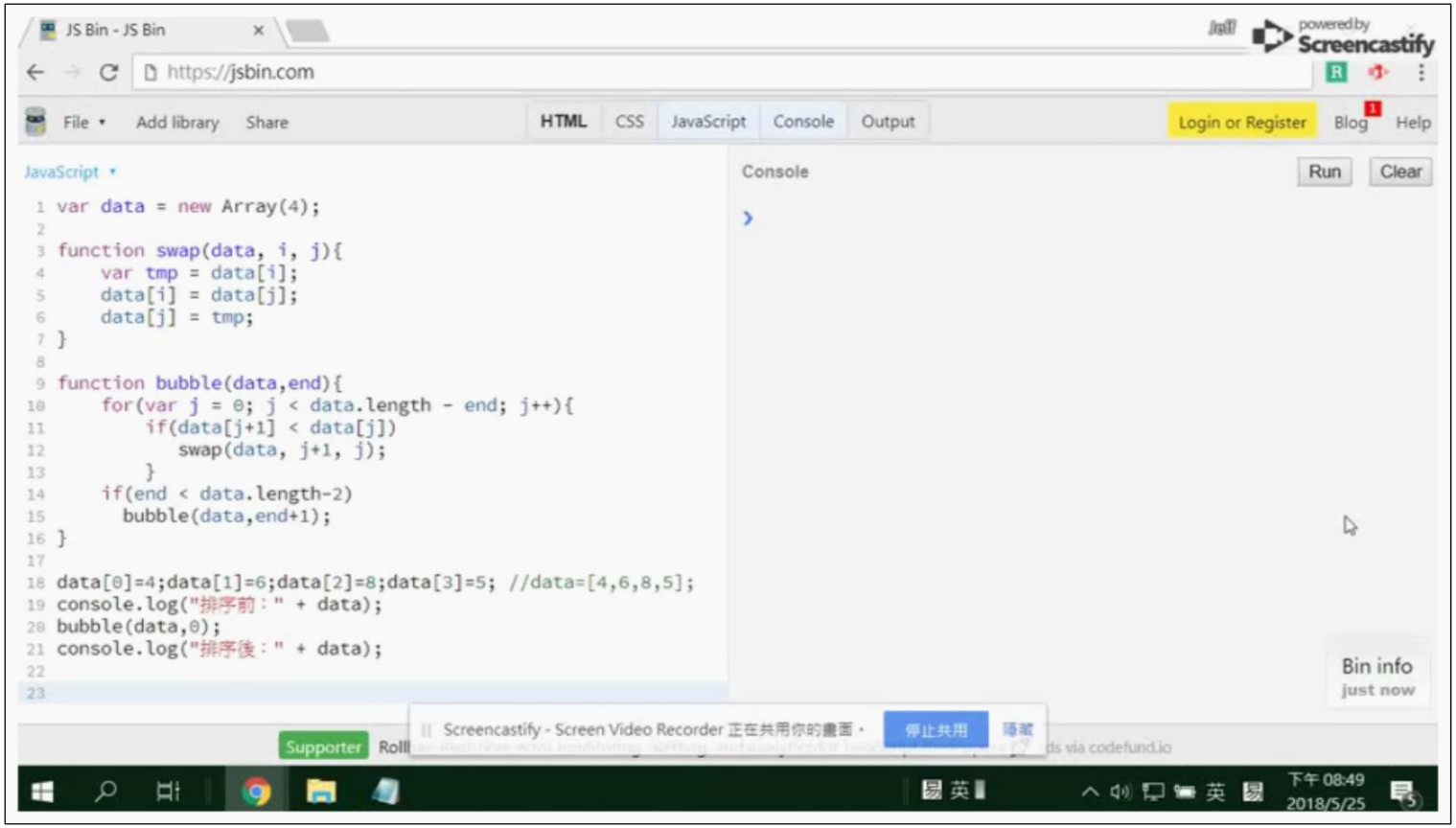


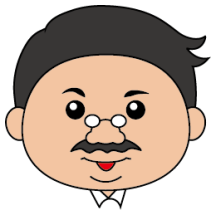


4-1-4　遞迴結構

操作影片

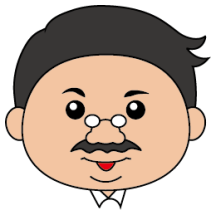
■ 實作練習8：遞迴法氣泡排序





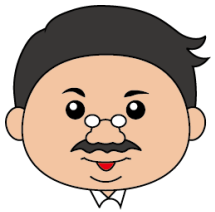
4-1-4　遞迴結構

- 氣泡排序每一回合會找出未排序資料中的最大值，因此第1次遞迴後，最大值之外剩下的就是下次遞迴的未排序資料，而每次遞迴做的工作都相同，只有待排序資料的結束位置不同而已，因此，在此例中程式設定一個end的變數，並傳入遞迴函式中。



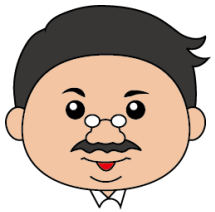
4-1-4　遞迴結構

		end=2	end=1	end=0
未排序資料	4	6	8	5
第1次遞迴	4	6	5	8
第2次遞迴	4	5	6	8
第3次遞迴	4	5	6	8



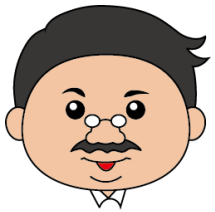
4-1-4 遞迴結構

程式	說明
1 var data = new Array(4);	宣告一個四個元素的陣列
2	
3 function swap(data, i, j){	交換陣列中兩個元素的資料
4 var tmp = data[i];	
5 data[i] = data[j];	
6 data[j] = tmp;	
7 }	
8	
9 function bubble(data,end){	氣泡排序函式，傳入陣列及end變數
10 for(var j = 0; j < data.length - end; j++){	執行迴圈找出未排序資料的最大值
11 if(data[j+1] < data[j])	假如左側元素較右側元素大



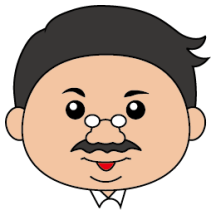
4-1-4 遞迴結構

12 swap(data, j+1, j);	交換陣列元素（大者往右移）
13 }	
14 if(end < data.length-2)	透過end變數判斷是否仍有未排序資料
15 bubble(data,end+1);	遞迴呼叫氣泡排序
16 }	
17	
18 data[0]=4;data[1]=6;data[2]=8;data[3]=5;	將4 6 8 5四個值存入陣列中
19 console.log("排序前：" + data);	在Console中印出"排序前："及串接陣列資料
20 bubble(data,0);	呼叫氣泡排序，傳入data陣列及end=0
21 console.log("排序後：" + data);	在Console中印出"排序後："及串接陣列資料



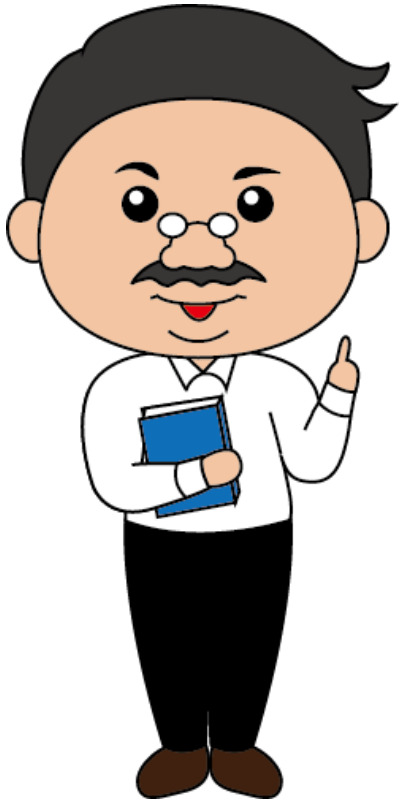
4-1-4　遞迴結構

- 這個程式會從第18行開始執行，並存入4個待排序值，在執行前會在Console中印出排序前data陣列中的值，排序完成後再印出排序後陣列中的值做確認。
- 這個程式只能替固定的陣列值進行排序，從程式訓練的角度來說，可以思考如何讓程式有更多的功能，進一步的練習改進。



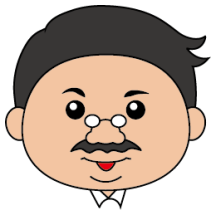
4-1-4　遞迴結構

- 例如：可以在HTML區新增一個輸入文字框(input tag)，讓使用者可以自行輸入待排序的資料，即可學習如何從文字框的輸入值中，分離出每個數字，並將這些數字存入陣列中，這樣就可以讓使用者自行輸入待排序值，程式即可有更多變化，也有利於程式執行結果的確認。



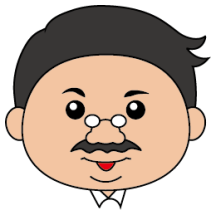
4-2 模組化程式設計實作

- 4-2-1　物件導向程式設計
- 4-2-2　JavaScript與物件導向



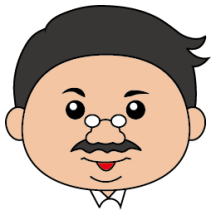
4-2-1　物件導向程式設計

- **物件導向程式設計(Object Oriented Programming, OOP)可視為一種程式開發的方式，程式設計師透過物件屬性的設定、物件事件的觸發控制及事件的反應程序，來設計程式系統。**

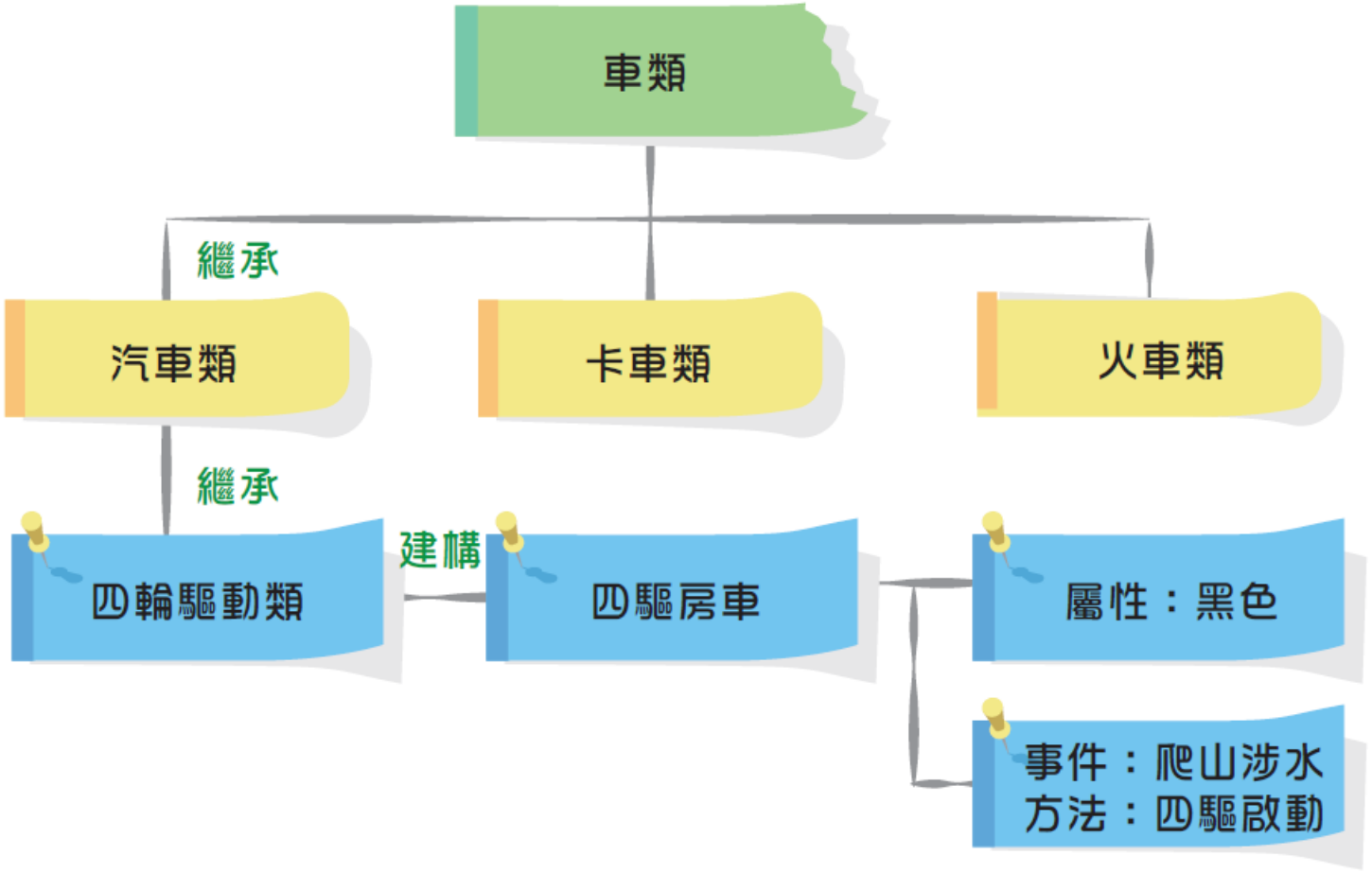


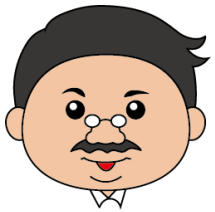
4-2-1 物件導向程式設計

- 在程式設計中，實體物件是從**類別**(class)建構而來。
- 例如：車子這個類別下，有汽車、卡車等子類別繼承車子類別而來，再根據汽車類別建構出一般實際的汽車或四輪驅動車。



4-2-1 物件導向程式設計



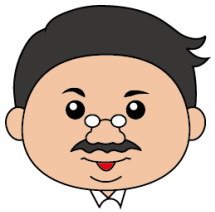


4-2-1　物件導向程式設計

物件導向三大特性

封裝(Encapsulation)

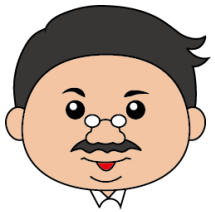
- 物件的方法及屬性必須被封裝成一個個體，外部程式只能透過封裝保留的界面，來設定物件的屬性或執行物件方法。封裝讓物件不被外部程式所破壞或影響，物件成為獨立個體。



4-2-1　物件導向程式設計

繼承(Inheritance)

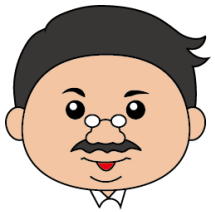
- 子類別可透過繼承獲得父類別的相關屬性與方法(功能)，也可擴充或修改成自己的屬性與方法。以四輪驅動車繼承自汽車類別為例，它繼承了一般前輪驅動的功能，也可以再建立新的後輪驅動功能。



4-2-1　物件導向程式設計

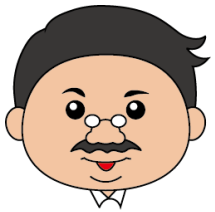
多型

- 簡單說就是繼承自同一類物件的方法，可以執行不同的功能。例如：四輪驅動車的驅動方法(四輪驅動.move)繼承自汽車類，但是「四輪驅動.move()」的方法與「汽車.move()」方法在動力輸出上是不同的。



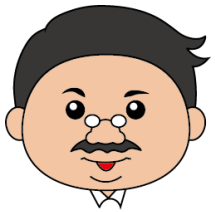
4-2-1 物件導向程式設計

- 這種改寫自父類別繼承下來的方法，稱為覆載多型(Overriding Polymorphism)，此英文中的over是指覆蓋的意思。
- 另外同一物件還可以透過傳遞不同的參數(Parametric)而執行不同的回應程式，稱為多載多型(Overloading Polymorphism)，這裡的over是指更多的意思。



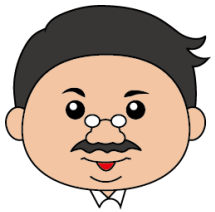
4-2-1 物件導向程式設計

- 舉例來說，四輪驅動車繼承自「汽車.move()」的方法，因此「四輪驅動.move()」的方法可以保留原汽車類的前輪驅動方法，但是「四輪驅動.move(1)」的方法，將四輪驅動車改為後輪驅動，「四輪驅動.move(2)」方法為四輪同時驅動。



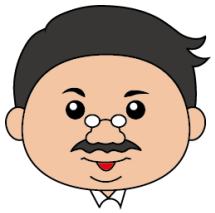
4-2-2 JavaScript與物件導向

- JavaScript是一種物件導向程式語言，具備物件導向封裝、繼承及多型等三大特性。
- JavaScript比較特別，它不採用一般物件導向語言class此關鍵字，而是以函式(Function)來建構類別，另外JavaScript不用宣告資料型別，資料型別的轉換由解譯器(瀏覽器等)自行處理。



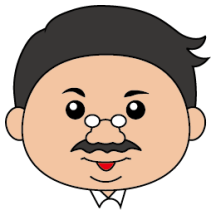
4-2-2 JavaScript與物件導向

- 例如：p="123"代表的是字串，指定p=123時，p的資料型別隨即轉換為整數，程式設計師對資料的處理比較方便，但程式速度比較慢。
- 下列程式是一個JavaScript的類別，它的建構方式類似函式，此例是建構一個堆疊結構：



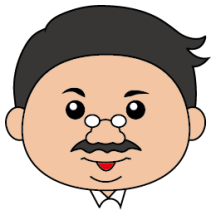
4-2-2 JavaScript與物件導向

```
function stack()
{
   var array = [];
   this.push = function(item)
                    {
                       array.push(item);
                    }
this.pop = function()
                    {
return array.pop();
                    }
    This.array=array;
}
```



4-2-2 JavaScript與物件導向

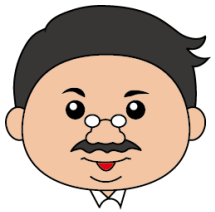
- 類別名稱：**stack**。
- 類別的方法：**push**及**pop**。
- 類別的屬性：**array**，資料結構為陣列「**[]**」。
- 當**stack**類別完成後，在**JavaScript**中即可用以下語法來建構出**my_stack**這個實體物件。



4-2-2 JavaScript與物件導向

var my_stack = new stack();

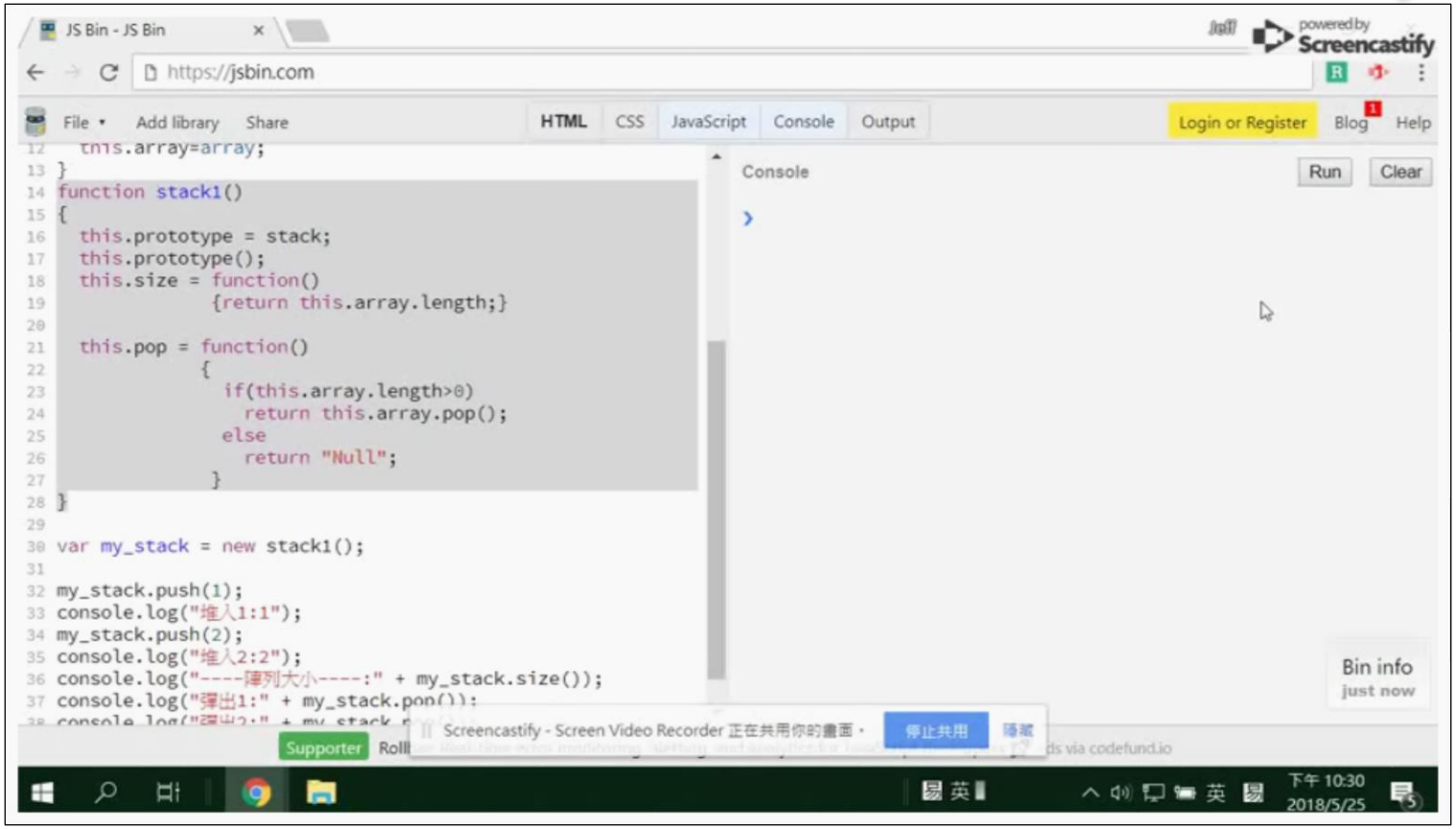
- 當實體物件建立後，我們即可使用此物件的方法，也可以讀取或設定物件的屬性，物件的採用方法是「物件.方法()」，「物件.屬性」，例如：
- my_stack.push("堆入");←→執行push的方法，傳入參數為字串：「堆入」。
- var my_array = my_stack.array;←→讀出物件中的array陣列給my_array變數。

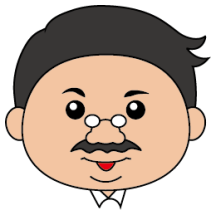


4-2-2 JavaScript與物件導向

- 實作練習9：建立一個堆疊的類別，並建立其物件實體

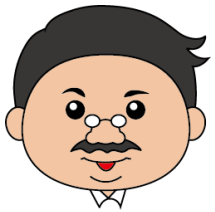
操作影片





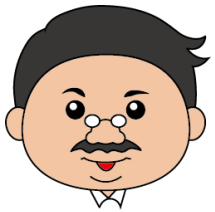
4-2-2 JavaScript與物件導向

程式	說明
1 function stack()	建立一個類別（函式），名稱為stack
2 {	
3 var array = [];	宣告一個陣列，名稱為array，預設值為空值
4 this.push = function(element)	建立push方法，this代表建立類別的物件
5 {	
6 array.push(element);	向array陣列堆入一個元素，值為element
7 }	
8 this.pop = function()	建立pop方法，this代表建立類別的物件
9 {	
10 return array.pop();	從array陣列彈出一個元素
11 }	
13 this.array=array;	建立一個array的屬性，值等於原array陣列



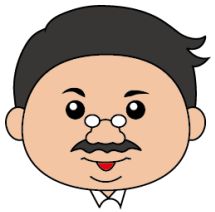
4-2-2 JavaScript與物件導向

14 }	
15 var my_stack = new stack();	建立一個stack類別的實體物件，名稱：my_stack
16 my_stack.push("數字1");	執行my_stack.push方法，堆入一個字串：『數字1』
17 console.log("堆入1:數字1");	在console中印出：『堆入1:數字1』
18 my_stack.push("數字2");	執行my_stack.push方法，堆入一個字串：『數字2』
19 console.log("堆入2:數字2");	在console中印出：『堆入2:數字2』
20 console.log("彈出1:" + my_stack.pop());	在console中印出：『彈出1:』串接my_stack.pop()彈出的值：『數字2』
21 console.log("彈出2:" + my_stack.pop());	在console中印出：『彈出2:』串接my_stack.pop()彈出的值：『數字1』
22 console.log("彈出3:" + my_stack.pop());	在console中印出：『彈出3:』串接my_stack.pop()彈出的值：『undefined』



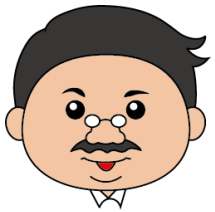
4-2-2 JavaScript與物件導向

- 這個程式JavaScript透過建立函式的方式建立類別(1~14行，stack類別)，再透過宣告(第15行)建構一個實體物件：my_stack，這個物件繼承自stack類別，並擁有這個類別的方法與屬性，其中第13行array是這個物件的屬性，本例中我們尚未使用。



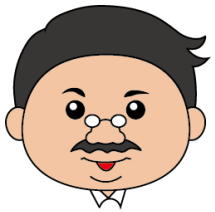
4-2-2 JavaScript與物件導向

- 程式中我們先堆入「數字1」，再堆入「數字2」，但先彈出的是「數字2」，其後才是「數字1」，這符合堆疊先進後出的要求。



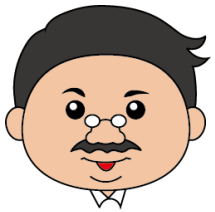
4-2-2 JavaScript與物件導向

- 第22行的彈出結果為什麼是「undefined」呢？其中文意思是「未定義」，這個堆疊在我們彈出兩次後，陣列就空了，此時再執行my_stack.pop()方法，JavaScript程式雖然不會出錯，但是會傳回undefined。



4-2-2 JavaScript與物件導向

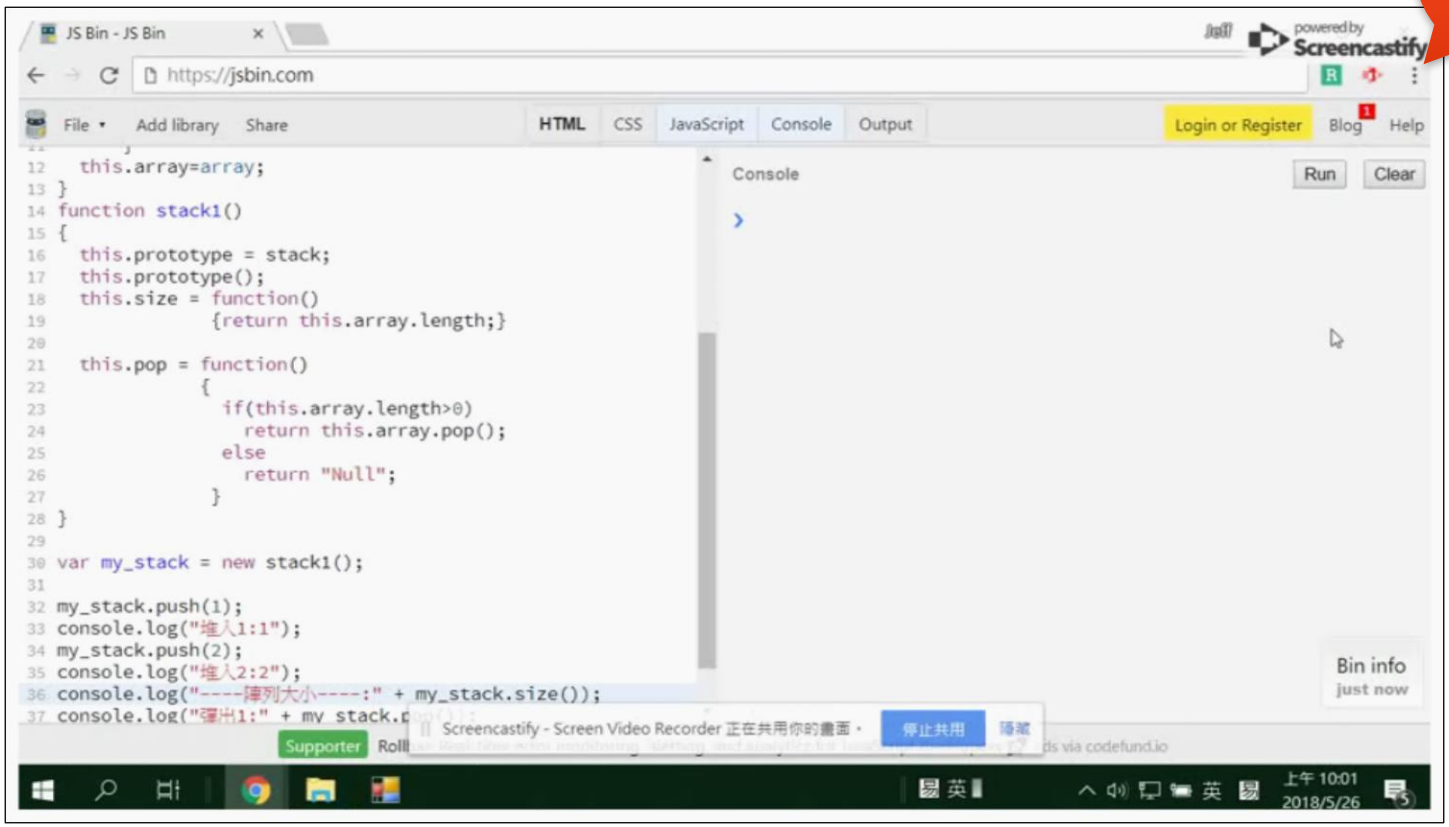
- 這是JavaScript的優異之處，有些程式語言則會出現程式錯誤而停止執行，因此這個程式並不夠好，最好是在執行pop方法之前，先判斷堆疊物件(陣列)是否已空，如果是就不執行pop方法。

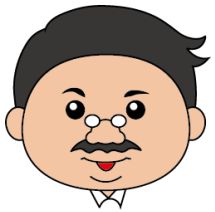


4-2-2 JavaScript與物件導向

- 實作練習10：繼承類別，新增子類別的方法並覆載原父類別的方法

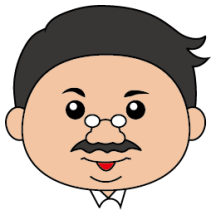
操作影片





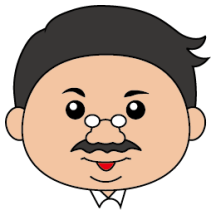
4-2-2 JavaScript與物件導向

- stack1類別繼承自stack類別的語法是利用第16及17行：
- this.prototype = stack;←→設定繼承類別，不可以寫成this.prototype = new stack();
- this.prototype();←→載入原型程式



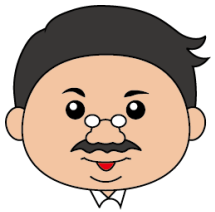
4-2-2 JavaScript與物件導向

- prototype的中文意思是原型，設定物件的原型，即是繼承自原型。
- 此例中stack1類別繼承自stack類別，兩者皆有pop(彈出)的方法，而根據物件導向子類別會覆載父類別的方法，因此my_stack物件是stack1類別的實體，my_stack.pop方法將取代原stack類別中相同名稱pop的方法。



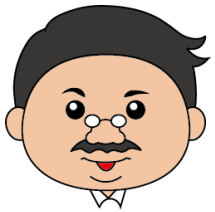
4-2-2 JavaScript與物件導向

- stack1.pop()方法與stack.pop方法的差異，在於stack1.pop多增加了判斷堆疊是否為空堆疊的判斷(第23~26行)，空堆疊就不執行彈出的動作，直接回傳「Null」，而不是由JavaScript解譯器傳回「undefined」。



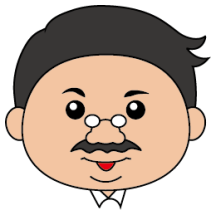
4-2-2 JavaScript與物件導向

程式	說明
1 ~13 function stack() { …… }	父類別stack的程式
14 function stack1()	建立一個類別（函式），名稱為stack1
15 {	
16 this.prototype= stack;	設定此類別原型（繼承）為stack類別
17 this.prototype();	執行原型的程式
18 this.size = function()	建立size方法，this代表建立類別的物件
19 { return this.array.length; }	This.size方法功能為回傳堆疊的大小（長度）
21 this.pop = function()	建立pop方法，取代stack類別的方法
22 {	
23 if(this.array.length>0)	判斷堆疊的大小是否大於0(不是空堆疊)



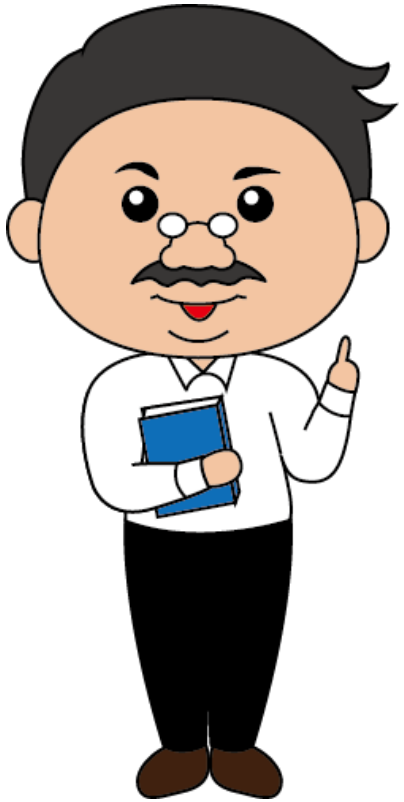
4-2-2 JavaScript與物件導向

24 return this.array.pop();	是則執行堆疊彈出
25 else	否則
26 return "Null";	回傳"Null" – 空值
27 }	
28 }	
30 var my_stack = new stack1();	宣告一個my_stcak變數，為stack1類別的物件
32 my_stack.push("數字1");	my_stack堆入『數字1』
33 console.log("堆入1:數字1");	在console中印出：『堆入1:數字1』
34 my_stack.push("數字2");	my_stack物件堆入一個字串：『數字2』



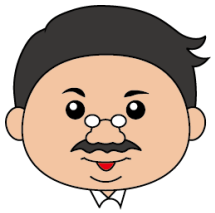
4-2-2 JavaScript與物件導向

35 console.log("堆入2:數字2");	console中印出：『堆入2:數字2』
36 console.log("----陣列大小----："+ my_stack.size());	在console中印出：『----陣列大小----：』串接目前堆疊空間的大小：2
37 console.log("彈出1:" + my_stack.pop());	在console中印出：『彈出1:』串接my_stack.pop()彈出的值：『數字2』
38 console.log("彈出2:" + my_stack.pop());	在console中印出：『彈出2:』串接my_stack.pop()彈出的值：『數字1』
39 console.log("彈出3:" + my_stack.pop());	在console中印出：『彈出3:』串接my_stack.pop()彈出的值：『undefined』



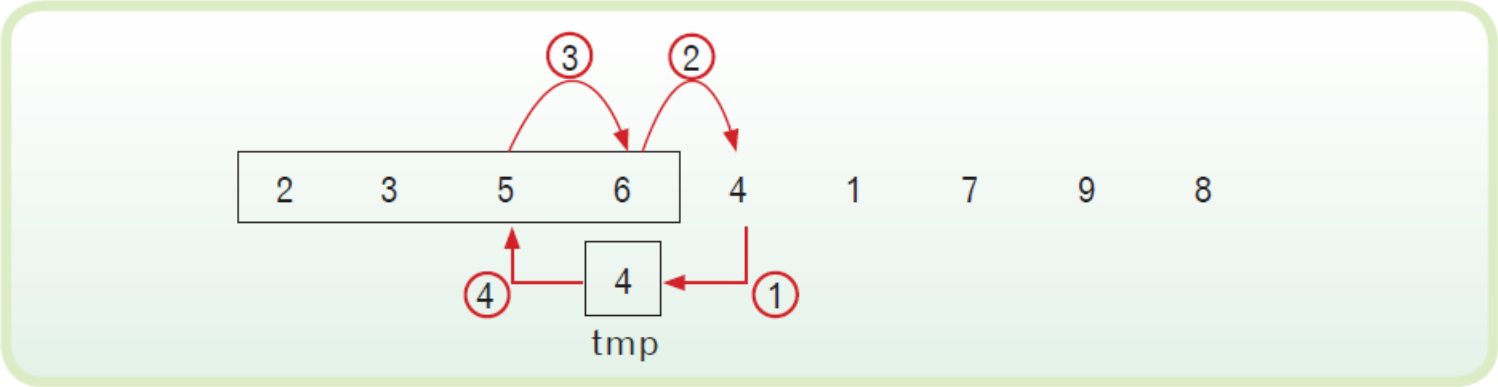
4-3 基本演算法的程式設計實作

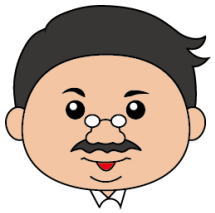
- 4-3-1　插入排序法
- 4-3-2　選擇排序法
- 4-3-3　二元搜尋法



4-3-1　插入排序法

- 插入排序每一次會將一個未排序資料插入到已排序完成的資料中。
- 2 3 5 6 是已排序完成的資料，而4是一個未排序資料，即將插入到3之後；原來5的位置。數字位置移動的順序如圈內數字，tmp是一個暫存變數。

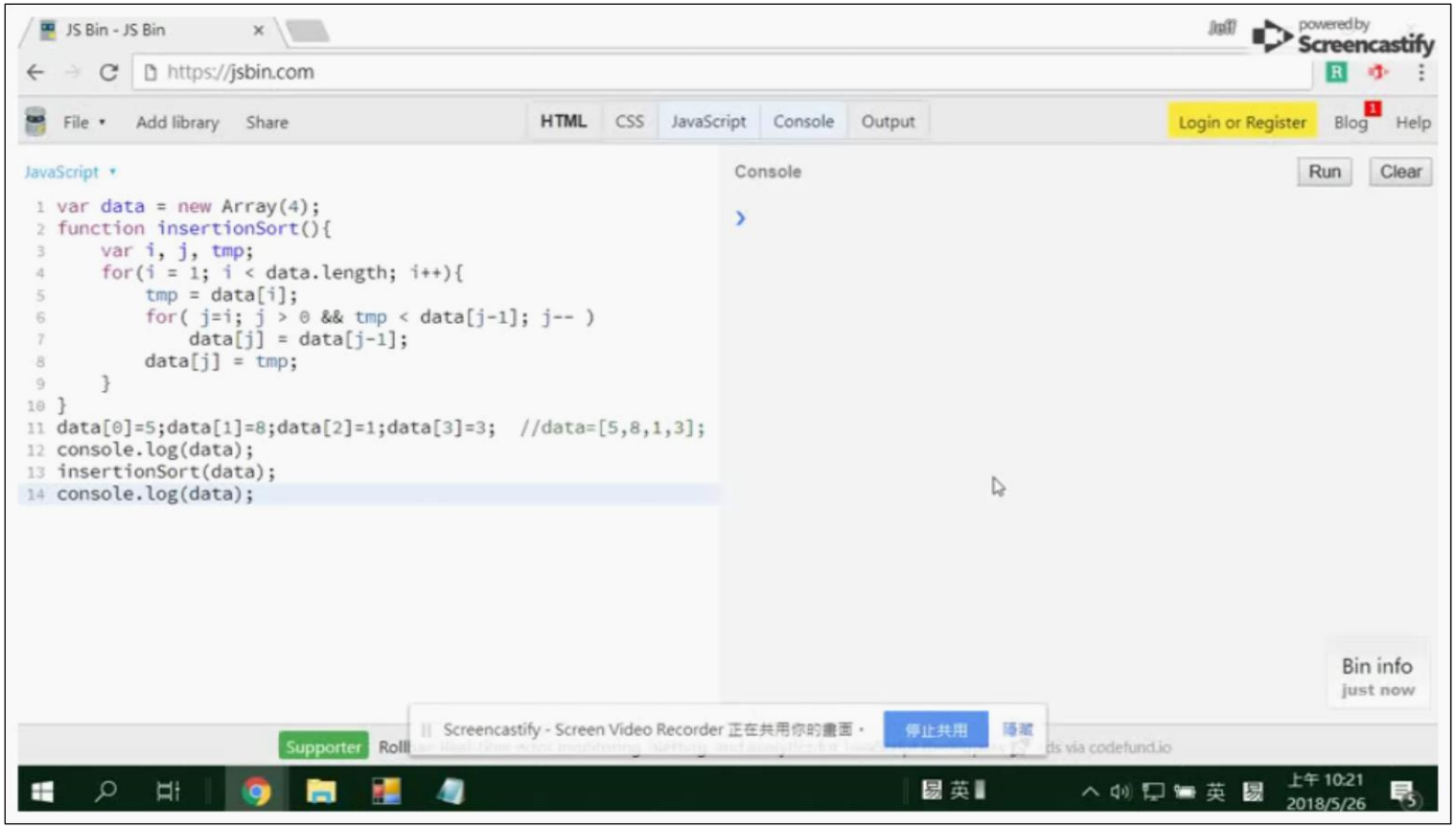


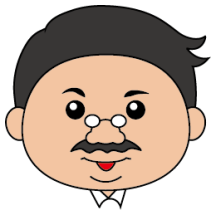


4-3-1　插入排序法

操作影片

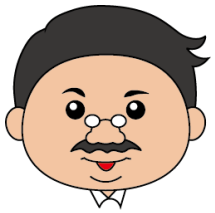
- 實作練習11：插入排序法





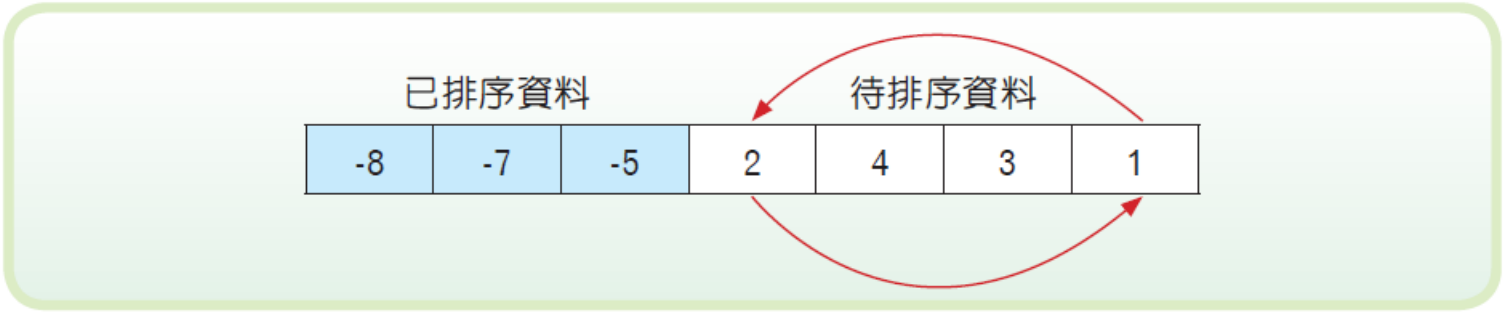
4-3-1　插入排序法

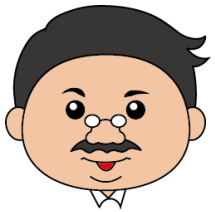
程式	說明
1 var data = new Array(4);	宣告四個元素名為data的陣列（全域變數）
2 function insertionSort(){	插入排序的函式
3　　var i, j, tmp;	
4　　for(i = 1; i < data.length; i++){	i是未排序的資料，從1開始至陣列長度減1
5　　tmp = data[i];	先將第i個未排資料放在暫存區，並騰出空間
6　　for(j=i; j > 0 && tmp < data[j-1]; j--)	從陣列中第i個往前找，直到待排資料data[j-1]<tmp
7　　data[j] = data[j-1];	已排資料往右移
8　　data[j] = tmp;	將待排資料插入至已排資料中
9　　}	
10 }	
11 data[0]=5;data[1]=8;data[2]=1;data[3]=3;	將待排資料 5 8 1 3 四個數字存入陣列中
12 console.log(data);	在console中印出排序前的陣列資料
13 insertionSort(data);	進行插入排序
14 console.log(data);	在console中印出排序後的陣列資料



4-3-2 選擇排序法

- 選擇排序法會在待排序陣列資料中找出最小值，然後與待排序陣列資料的第一個資料做交換。

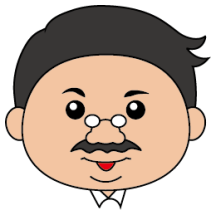




4-3-2 選擇排序法

- 如何找出待排序中最小值呢？其方法是將待排資料的第一個數字的陣列索引值(i)存入最小值索引變數(min=i)，然後此待排數字與其它待排數字做比較，比較小，就將其陣列索引值存入最小值的變數(min=j)，演算法如下：

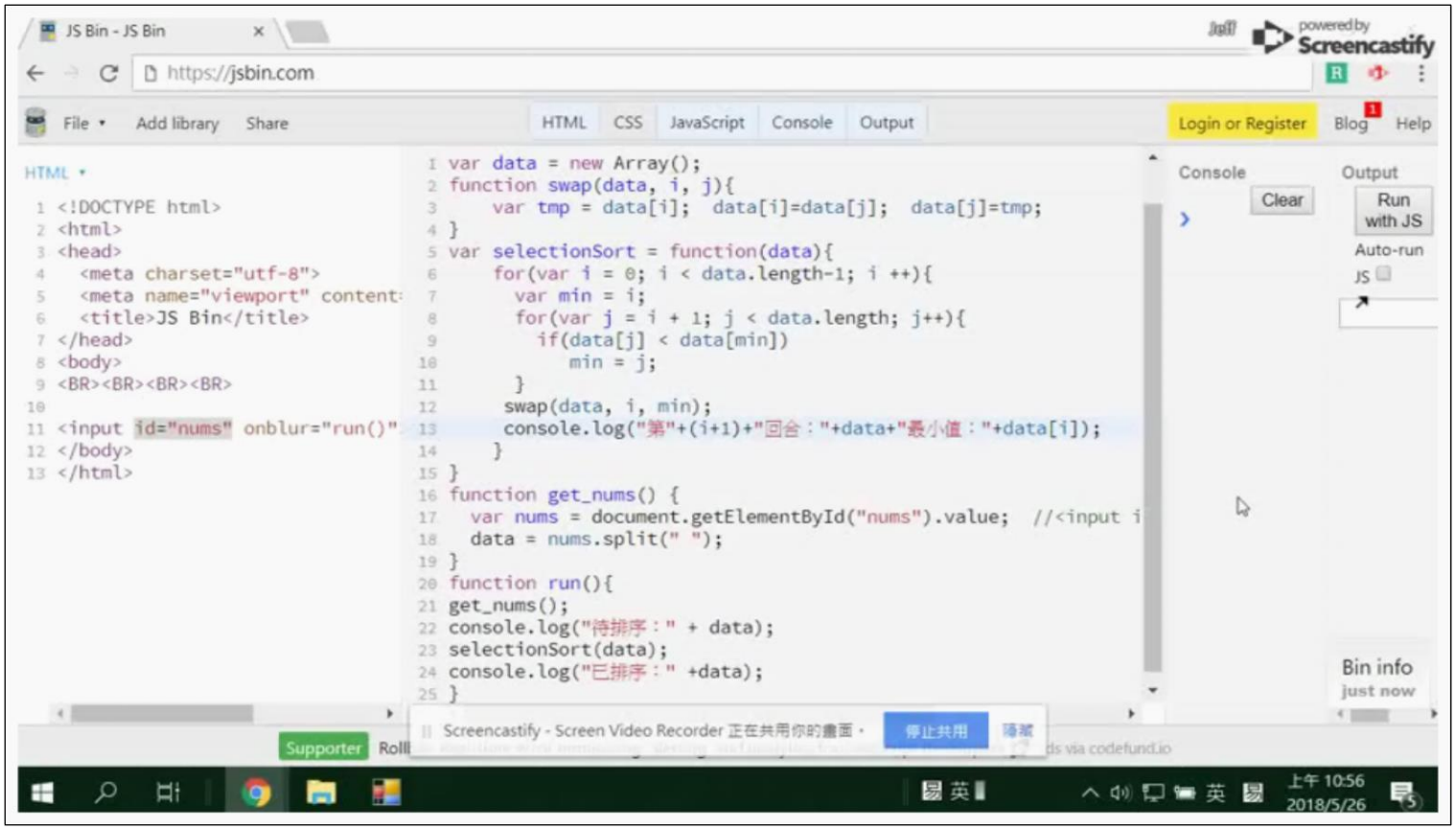
```
var min = i;  //將目前待排序數字的索引值寫入min變數
for(var j = i + 1; j < data.length; j++){  //其餘待比較的數字從陣列i+1~陣列最後一個
if(data[j] < data[min])  //假如此待比較數字比目前最小值還要小（執行下一行）
min = j;  //陣列索引值寫入變數min
}
```

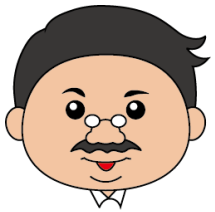


4-3-2　選擇排序法

操作影片

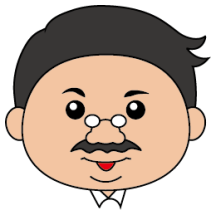
■ 實作練習12：選擇排序法





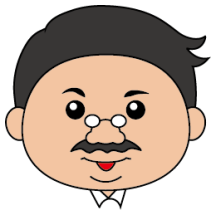
4-3-2 選擇排序法

程式	說明
1 function swap(data, i, j){	將陣列索引i與j的值交換的函式
2 var tmp = data[i]; data[i]=data[j]; data[j]=tmp;	
3 }	
4 var selectionSort = function(data){	
5 for(var i = 0; i < data.length-1; i ++){	改變待排序數字的範圍，最後一個不用排
6 var min = i;	先將此次待排序的最小值的索引假設為i
7 for(var j = i + 1; j < data.length; j++){	從陣列i+1至陣列的最後一個元素
8 if(data[j] < data[min])	如果data[j]的值比最小值（data[min]）還小
9 min = j;	最小值索引改為j



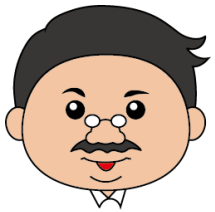
4-3-2 選擇排序法

10 }	
11 swap(data, i, min);	將此回合待排序資料的第一個數字data[i]與data[min]交換位置
12 console.log("第"+(i+1)+"回:"+data+"最小值:"+data[i]);	在console區印出此回合的最小值
13 }	
14 }	
15 function get_nums() {	將文字輸入框的數字串分割後傳入data陣列中
16 var nums = document.getElementById("nums").value;	取得nums物件的值（數字串），並指定給nums變數
17 data = nums.split(" ");	將nums字串變數以空白鍵為區隔符號進行分割，並存入data陣列中



4-3-2 選擇排序法

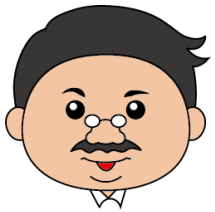
18 }	
19 function run(){	文字框失去焦點，執行的事件程序
20 get_nums();	進行數字串分割
21 console.log("待排序：" + data);	在Console區印出排序前的待排數字
22 selectionSort(data);	進行選擇排序
23 console.log("已排序：" +data);	在Console區印出排序後的排序數字
24 }	



4-3-2 選擇排序法

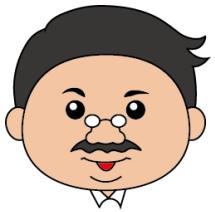
傳值或傳位址呼叫*

- 在此例中我們並未宣告data陣列，第一次使用data陣列是在get_nums函式中，而在事件程序run函式及各函式中皆使用到它，而且大家使用的都是同一個data陣列。
- 在get_nums函式中未宣告data，則這個data陣列會被視為全域變數。



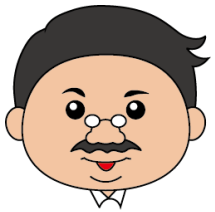
4-3-2 選擇排序法

- 而在一般程式呼叫中，有傳值呼叫(call by value)及傳位址呼叫(call by address or reference)兩種，傳值呼叫是指將值傳進函式，在函式中的變數與原呼叫變數無關，函式中該變數值的改變，並不影響原變數的值。
- 以下列JavaScript程式為例，c_value被傳入test函式，但並不影響c_value的值，c_value仍然為3。



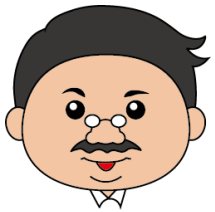
4-3-2 選擇排序法

```
var c_value = 3;
test(c_value);
alert(c_value);          //顯示3 , 值沒有改變
function test(c_value){
c_value = c_value+1; //test函式中的c_value已改變為4
}
```



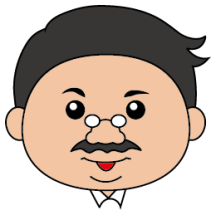
4-3-2 選擇排序法

- 傳位址呼叫是指將變數位址傳進函式，在函式中的變數與原呼叫變數是同一個，函式中該變數值的改變，原變數的值就跟著改變。
- 以下列JavaScript程式為例，test(c_value)將c_value陣列傳入test函式，JavaScript的陣列自動屬於傳位址呼，c_value[0]的值在test()函式中被改變為4。



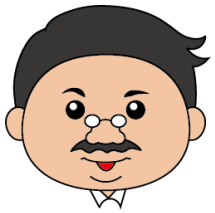
4-3-2 選擇排序法

```
var c_value = new Array();     // Array≠array
c_value[0]=3;
test(c_value);
alert(c_value[0]);                    //顯示4 , c_value[0]的值改變了
function test(c_value){
c_value[0] = c_value[0]+1;     //test函式中的c_value[0]已改變為4
}
```



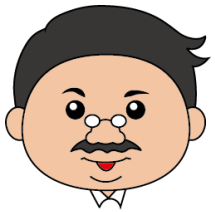
4-3-2　選擇排序法

- 下列程式碼則是VB.net的程式，VB.net的呼叫皆必須指定傳值(ByVal)或傳位址(ByRef)。
- 此例中test(ByVal　c_value)採傳值呼叫，因此Start_click副程式呼叫test函式後，c_value值並沒有改變。



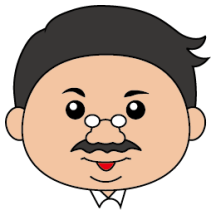
4-3-2 選擇排序法

```
Private Sub Start_Click(ByVal sender As System.Object)
Dim c_value As Integer = 3              '宣告一個整數變數
MessageBox.Show(test(c_value))          '顯示4
MessageBox.Show(c_value)                '顯示3 , 值沒有改變
End Sub
Function test(ByVal c_value)
c_value += 1
Return c_value                          '回傳c_value
End Function
```



4-3-3 二元搜尋法

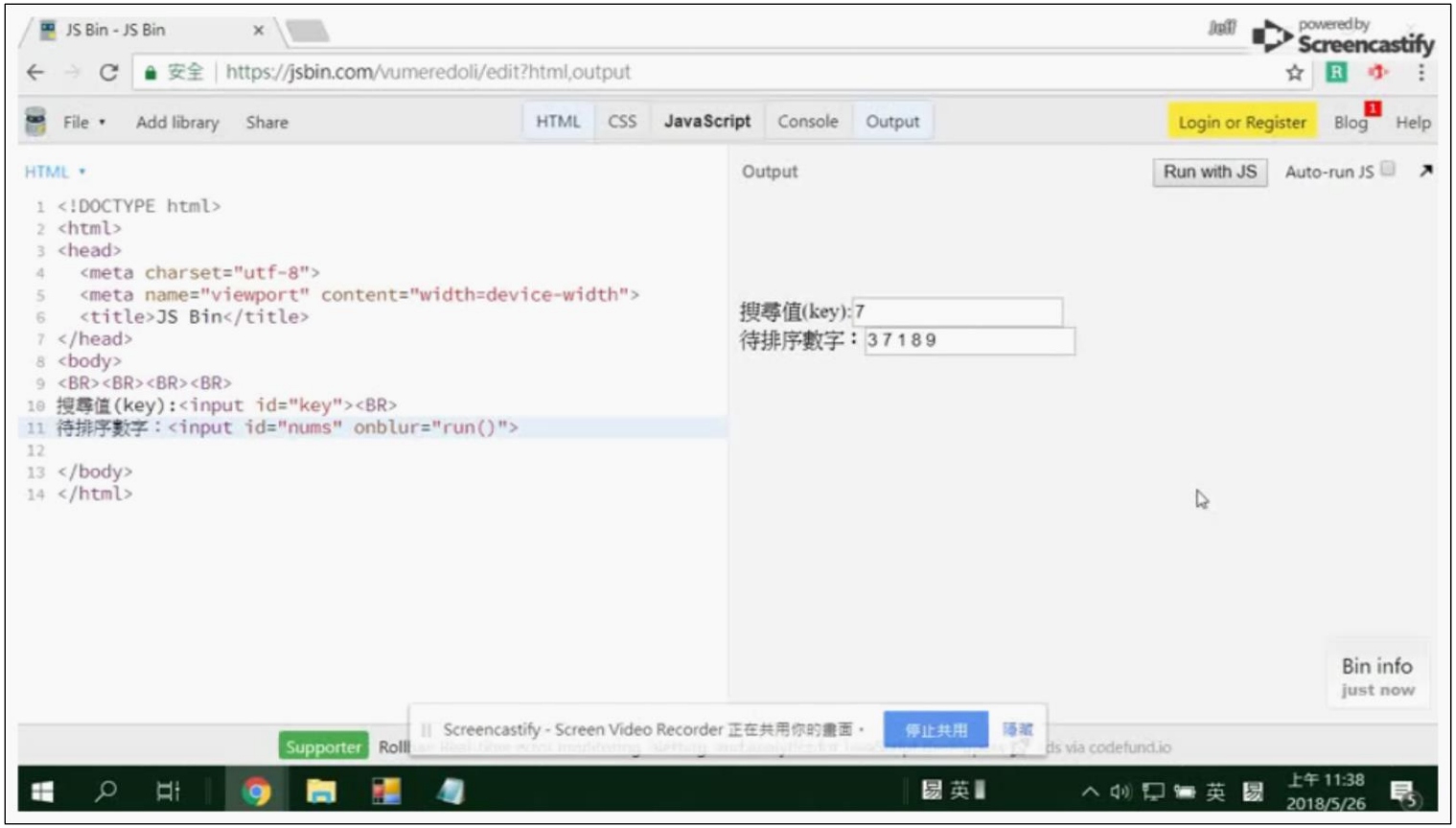
- 二元搜尋的基本要求，是待搜尋的資料已排列完成，每回合取得一個中間值，如果中間值不是搜尋值(Key)，再由中間值的大小，決定往左或往右進行下一回合的搜尋。

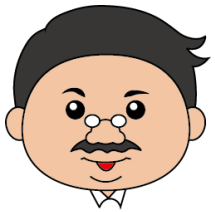


4-3-3 二元搜尋法

操作影片

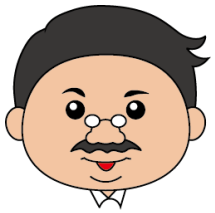
- 實作練習13：選擇排序+二元搜尋





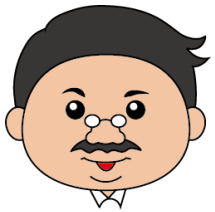
4-3-3 二元搜尋法

```
function swap(data, i, j){
    var tmp = data[i];  data[i]=data[j];  data[j]=tmp;
}
var selectionSort = function(data){
    for(var i = 0; i < data.length-1; i ++){
      var min = i;
      for(var j = i + 1; j < data.length; j++){
        if(data[j] < data[min])
            min = j;
      }
```



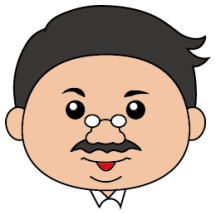
4-3-3 二元搜尋法

```
    swap(data, i, min);
    console.log("第"+(i+1)+"回:"+data+"最小值:"+data[i]);
   }
}
function get_nums() {
  var nums = document.getElementById("nums").value;
  data = nums.split(" ").map(function(item) { return parseInt(item);});
}
function BinarySearch(data, key) {
   var left = 0;
   var right = data.length - 1;
```



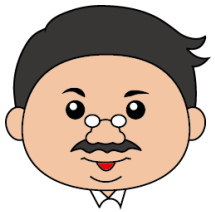
4-3-3 二元搜尋法

```
var middle = 0;

while(left <= right){
    middle = parseInt((left+right)/2);
    if(data[middle] == key)
        return middle;
    else if(data[middle] < key)
        left = middle + 1;
    else
        right = middle - 1;
}
```



4-3-3 二元搜尋法

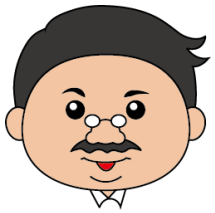
```
    return -1;
}
function run(){
 get_nums();
 console.log("待排序：" + data);
 selectionSort(data);
 console.log("已排序：" +data);
 var key = document.getElementById("key").value;
 var result =BinarySearch(data,key);
 console.log(key+"在第:"+(result+1)+"位置");
}
```



4-3-3 二元搜尋法

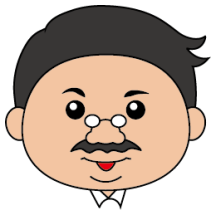
程式Debug

- 這個程式我們進行了兩次測試，第一次「3 7 1 8 9」排序及搜尋的位置都正確，但第二次「13 7 1 8 19」的排列順序：「1 13 19 7 8」，13及19被排在7及8之前的結果是錯的。
- 在程式設計中，除錯(Debug)是程式設計的一環，有時除錯花費的時間，甚至超過程式編碼(Coding)。



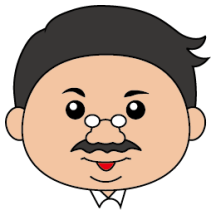
4-3-3 二元搜尋法

- 程式設計師必須根據程式的執行結果，以及測試界面指出的錯誤行數，進行程式除錯。
- 找出錯誤通常建立在程式設計師的經驗，初學者通常經由除錯的過程累積經驗。



4-3-3 二元搜尋法

- 1 13 19排在前面，7 8 排在後面，有經驗的程式設計師，應該就可以猜測到是程式將它們當成字串在排序。
- 這個錯誤的源頭，在於get_num()函式將文字框內的數字分割並存入data陣列時(第17行)，存入的是字串，而非數字，因此排序時1開頭的數字(實際上是字串)都會比較小，才會造成「13<7」的情況。



4-3-3 二元搜尋法

```
15 function get_nums() {
16 var nums = document.getElementById("nums").value;
17 data = nums.split(" ");
18 }
```

- 我們可以採用以下語法，取代上方第17行的程式，在存入陣列時，將字串轉成整數，這個程式錯誤即可解決。

```
data = nums.split(" ").map(function(item) { return parseInt(item);});
```