Ch5 集合(Collection) 物件

1 學習目標

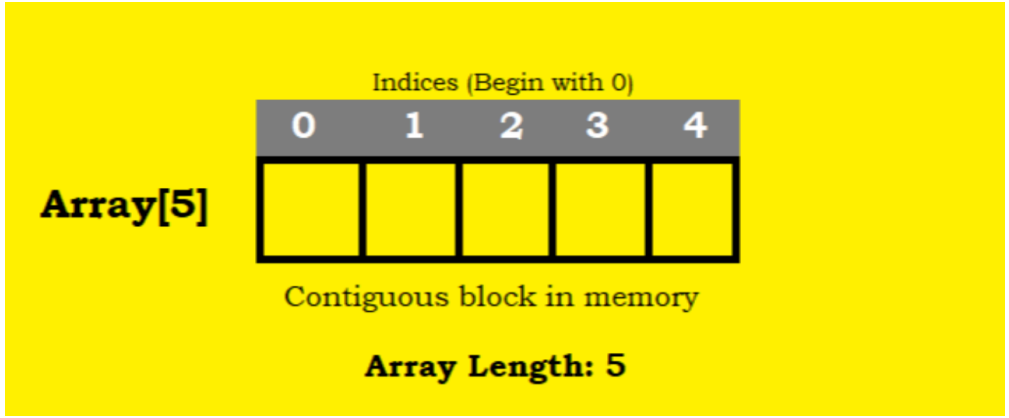
- 了解集合物件的特性
- 陣列(Array)的宣告及操作
- Set 與 Map 的宣告及操作

2 集合物件

- 集合物件是用來儲存多個值的資料結構
- 例如:
  - 陣列 (Array): 存放多個值，使用索引(index)來存取, 值可重複
  - Set: 存放不重複的值, 使用索引來存取
  - Map: 存放鍵值對 (key-value pair), 使用 key 來存取 值
- 他們都是物件參考型別(Object Reference Type)
  - stack 中儲存物件的參考(reference)位址
  - heap 中儲存物件的實際資料, 可動態擴展

3 陣列 (Array)

- 以連續的記憶體空間儲存多個值
- 值的型別可以不同，不一定要全部是同一型別



4 使用情境

- 存放同學的成績: 用陣列儲存班上同學的考試分數，方便計算平均分數和排序
- 購物車商品清單: 儲存使用者選購的商品，每個元素可以是商品物件，包含名稱、價格等資訊
- 歷史紀錄追蹤: 記錄使用者最近的操作或瀏覽記錄，可以使用陣列來實作「上一步」或「下一步」功能

5 陣列的基本操作

- 建立陣列
- 訪問陣列元素
- 修改陣列元素（添加、刪除和更新）
- 獲取陣列的長度
- 遍歷陣列(拜訪陣列的每個元素)
- 陣列的排序和搜尋
- 陣列的切片和拼接

6 建立陣列

有三種建立陣列的方法:

1. 陣列文字(Array Literal)
2. Array() 建構子
3. Array.of() 方法

陣列文字(Array Literal)

- 使用文字來描述陣列及其內容
- 經常在建立空陣列或事先已知值的清單時使用。

Ex: 有 red, green, blue 三個顏色的陣列

```
var colors = ['red', 'green', 'blue'];
```

Ex: 建立空陣列

```
var emptyArray = [];
```

Array() 建構子

- Array literal 的限制
  - 無法一次建立很長的陣列
  - e.g. 建立一個長度為 100 的陣列
- 使用 Array() 建構子來建立陣列
  - 可提供 "數字" 表示陣列的長度
  - 可提供 "一個或多個值"，表示陣列的內容

Ex: 建立長度為 5 的陣列, 以儲存顏色

- 注意: 內容沒有初始化

```
var colors = new Array(5);
console.log(colors); // [ <5 empty items> ]
```

Ex: 儲存 5 個顏色的陣列, 5 個顏色分別為: red, green, blue, yellow, purple

```
var colors = new Array('red', 'green', 'blue', 'yellow', 'purple');
console.log(colors); // [ 'red', 'green', 'blue', 'yellow', 'purple' ]
```

Syntax

JS

```
new Array()
new Array(element1)
new Array(element1, element2)
new Array(element1, element2, /* …, */ elementN)
new Array(arrayLength)

Array()
Array(element1)
Array(element1, element2)
Array(element1, element2, /* …, */ elementN)
Array(arrayLength)
```

Array() 的簽名

- 多個引數時, 使用這些引數來初始化陣列
- 單一引數時，
  - 如果是數字，則表示陣列的長度
  - 如果不是數字，則表示陣列的內容

Array() 的常見陷阱

x_arr 和 y_arr 的意義相同嗎？

```
var x_arr = new Array(5);
var y_arr = new Array("5")
```

▶ 參考答案

Array.of()

- 如果明確地要使用一串資料(a list values)來建立陣列，則使用 Array.of() 方法
  - 明確的展現你的意圖
- Array.of() 方法會將所有引數視為陣列的內容

Ex: 建立一個學生成績的陣列, 只有一個元素 80 分

```
var scores = Array.of(80);
console.log(scores); // [ 80 ]
```

補充

另有 Array.from() 方法

- 將一個 array-like 物件(類陣列物件)或可迭代物件(iterable object)轉換為陣列
- 後續章節會介紹 array-like 物件和可迭代物件

7 訪問陣列元素 (Accessing Array Elements)

使用整數的索引值

- 使用索引來訪問陣列的元素
- 索引符號是方括號 []
- 索引值是必需是 整數及非負數
- 陣列的索引從 0 開始
- 如果索引超出範圍，則返回 undefined

取得陣列的元素

Ex. 取得顏色陣列的第2個元素

```
var colors = ['red', 'green', 'blue'];
console.log(colors[1]); // green
```

Ex. 取得顏色陣列的最後一個元素

```
var colors = ['red', 'green', 'blue'];
console.log(colors[colors.length - 1]); // blue
```

Ex. 索引值超出範圍

```
var colors = ['red', 'green', 'blue'];
console.log(colors[3]); // undefined
```

非整數的索引值: 成為 Array 的屬性

- 陣列是物件
- JS 中，允許動態的新增物件屬性
- 取得物件屬性的語法：
  - 物件名稱.屬性名稱
  - 物件名稱["屬性名稱"]
- 當使用非整數的索引值時，會變成使用物件屬性的語法。

Ex. 使用非整數的索引值

```
var colors = ['red', 'green', 'blue'];
// 3.5 會轉成字串 "3.5"
colors[3.5] = 'yellow';
colors[-1] = 'purple';
```

會產生以下 colors 物件

```
{
    0: "red",
    1: "green",
    2: "blue",
    '3.5': "yellow",
    '-1': "purple"
}
```

8 陣列元素的: add, delete, update

- 可以新增、刪除或更新陣列的元素

新增或更新元素

- 使用指派運算子 (=) 來新增或更新元素
- 索引值超出範圍時，會自動擴展陣列的大小
  - 若有跳號的情況，則會在中間的元素補上 undefined
- 否則, 會更新指定索引的元素

Ex. 新增元素: 索引值超出範圍，並有跳號的情況

```
var colors = ['red', 'green', 'blue'];
colors[5] = 'yellow';
console.log(colors); // [ 'red', 'green', 'blue', <2 empty items>, 'yellow' ]
```

Ex. 更新元素: 更新第一個元素為 black

```
var colors = ['red', 'green', 'blue'];
colors[0] = 'black';
console.log(colors); // [ 'black', 'green', 'blue' ]
```

加到陣列的最後面的位置

方法 1: 使用 length 屬性取得陣列的長度

```
var colors = ['red', 'green', 'blue'];
colors[colors.length] = 'yellow';
console.log(colors); // [ 'red', 'green', 'blue', 'yellow' ]
```

方法 2: 使用 push() 方法

```
var colors = ['red', 'green', 'blue'];
colors.push('yellow');
console.log(colors); // [ 'red', 'green', 'blue', 'yellow' ]
```

Q: 那個程式碼看起來比較優雅？明確表達你的意圖？

刪除元素

- 使用 delete 運算子來"刪除"元素
- `delete` 並沒有真正的刪除元素
  - 只是將該元素的值設為 undefined
  - 陣列的長度不會改變

Ex. 刪除元素 (陣列長度不變)

```
var colors = ['red', 'green', 'blue'];
delete colors[1];
console.log(colors); // [ 'red', <1 empty item>, 'blue' ]
```

delete 運算子的副作用

- delete 運算子會使陣列變得稀疏，因為它不會改變陣列的長度。
- 如果你要移除元素，包含它的位置，請使用陣列的 `splice()` 方法。
  - 避免陣列變得稀疏
- `splice()` 方法用來添加(insert)、更新(update)和刪除(delete)陣列中的元素。
  - 會改變陣列的長度，避免陣列變得稀疏
  - splice 中文意指接合或連接。

See MDN web docs: Array.prototype.splice() - JavaScript | MDN

9 遍歷陣列(拜訪陣列的每個元素)

- 遍歷陣列的每個元素是常見的陣列操作
- 常見情境:
  - 印出陣列的每個元素
  - 計算陣列的總和
  - 陣列中的每個元素加 1

三種遍歷陣列的方法

- `for` loop: 傳統的方式, 要自己管理 counter
- `for/of` loop: 較新的方式, 會自動管理 counter
- `forEach()` method: 使用 Iterator 物件來遍歷陣列

for loop

有一個 colors 陣列，裡面有 red, green, blue 三個顏色，如何印出每個顏色？

```
let colors = ['red', 'green', 'blue'];
```

使用 for loop, 你需要自己管理 counter (或 index) 的值

```
for (let i = 0; i < colors.length; i++) {
    console.log(colors[i]);
}
```

Q: 有沒有更簡潔的方式？

for/of loop

- `for/of` loop 是 ES6 新增的語法
- 會自動管理 counter (或 index) 的值
- 但你需要一個變數來存放被拜訪(當前)的元素(visited element or current element)

重寫上面的程式碼:

```
for (let color of colors) {
    console.log(color);
}
```

取得 index 的值 (使用 for/of loop 的時候)

Q: 使用 for/of loop 的時候，如何取得 index 的值？

- index 由 JS 自動管理
  A: 使用 `Array` 物件的 `entries()` 方法, 回傳陣列元素的 [index, value] 陣列
- 註: `entries()` 回傳一個 Iterator (迭代器) 物件, 會在後面章節介紹

印出陣列元素的 index 及 value

修改上面的程式碼，使印出顏色的 index 及 value

```
for (let [index, color] of colors.entries()) {
    console.log(index, color);
}
```

Quick Practice

有以下的陣列, 請印出每個元素的 index 及 value:

```
let revengers = ['ironman', 'thor', 'hulk', 'black widow', 'hawk eye'];
```

使用 for/of loop 遍歷陣列。

▶ 參考答案

forEach() method

Q: 先前的 for/of loop 的 block 中的內容如果要重複使用，該怎麼辦？

有以下兩個陣列, 印出每個元素:

```
let colors = ['red', 'green', 'blue'];
let fruits = ['apple', 'banana', 'orange'];
```

如果使用 for/of loop 的話，會有重複的 block:

```
for (let color of colors) {
    console.log(color);
}
for (let fruit of fruits) {
    console.log(fruit);
}
```

將重複的 block 抽出來變成 function 重覆使用

```
function printElement(el) {
    console.log(el);
}
```

搭配 forEach() method, 套用此函數到每個元素:

```
colors.forEach(printElement);
fruits.forEach(printElement);
```

forEach(callback) method

- forEach(callback) method 是陣列物件提供 iterative method
- 接受一個函數作為參數, 應用此函數到每個元素
- 遍歷過程中可對每個元素套用函數，並產生新的陣列
  - 原來的陣列不會改變
- 這種設計思維的方式來自於「函數式編程」(Functional Programming)

Quick Practice

有以下的陣列, 我們期望將內容轉成大寫並印出來:

```
let colors = ['red', 'green', 'blue'];
let fruits = ['apple', 'banana', 'orange'];
```

使用 forEach() method 完成要求。

Hints:

- 先定義一個函數，將傳入的參數轉成大寫並印出來
  - 使用 `String` 物件的 `toUpperCase()` 方法轉成大寫
- 接著使用 forEach() method 套用此函數到每個元素

▶ 參考答案

ForEach() 的 callback function 的簽名

若只想要元素的值，則只需要一個參數

```
function callback(element) {
    // ...
}
```

若還需要 index 的值，則需要兩個參數

```
function callback(element, index) {
    // ...
}
```

若還需要陣列本身的參考，則需要三個參數

```
function callback(element, index, array) {
    // ...
}
```

10 陣列的進階操作方法

陣列物件提供許多方法來操作陣列:

- 新增和替換元素: `push()`, `unshift()`, `splice()`
- 移除元素: `pop()`, `shift()`, `splice()`
- 切片: `slice()`
- 尋找元素: `indexOf()`, `find()`
- 排序: `sort()`, `reverse()`
- 陣列串接: `concat()`, `join()`

push() and pop()

情境: 自陣列的尾端新增或移除元素

- `push()` 方法: 在陣列的尾端新增一個或多個元素
- `pop()` 方法: 移除陣列的尾端元素

Ex. 現有 5 個人在排隊，請將 6 個人加入隊伍中

```
let queue = ['A', 'B', 'C', 'D', 'E'];
queue.push('F');
console.log(queue); // [ 'A', 'B', 'C', 'D', 'E', 'F' ]
```

Ex. 現有 6 個人在排隊，請將最後一個人移除

```
let queue = ['A', 'B', 'C', 'D', 'E', 'F'];
queue.pop();
console.log(queue); // [ 'A', 'B', 'C', 'D', 'E' ]
```

unshift() and shift()

情境: 自陣列的**前端**新增或移除元素

- `unshift()` 方法: 在陣列的前端新增一個或多個元素
- `shift()` 方法: 移除陣列的前端元素(整個元素的位置會往前移動, 第0個元素從陣列中移除)
    - 回傳被移除的元素

Ex. 現有 5 個人在排隊, 第一個人已完成服務，後面的人要往前移動

```
let queue = ['A', 'B', 'C', 'D', 'E'];
let complete = queue.shift();
console.log(complete); // A
console.log(queue); // [ 'B', 'C', 'D', 'E' ]
```

Ex. 剛離開的人因故回來了，請將他加入隊伍的前端

```
let queue = ['B', 'C', 'D', 'E'];
```

Quick Practice

1. 有三個人, 分別為 Jack, Tom, Mary, 排隊買票，用一陣列描述。
2. Jack 已經買完票了，請將他移除並印出他的名字。更改隊伍的順序。
3. Emily 來了，請將她加入隊伍的尾端。
4. Sophia 來了，是 VIP，請將她加入隊伍的前端。
5. 印出隊伍的順序。

▶ 參考答案

splice(): 新增、更新或刪除元素

- 一個可用於在陣列中新增、移除和替換元素的通用方法。
- 優點:
  - 不會產生稀疏的陣列，因為會改變陣列的長度
- 小心:
  - 會改變原始陣列的內容

`splice()` 方法對陣列進行的操作：

1. 移除: 從 `start` 索引開始移除 `deleteCount` 個元素。
2. 新增: 在 `start` 索引處插入 `item1, item2, ...` 等元素。
   - `start` 索引後的原始元素會向右移動。
3. 更新: 當 `deleteCount` = 1 時，更新 `start` 索引處的元素。

`splice()` 方法會修改原始陣列，並回傳:

- 被移除的元素的陣列
- 如果沒有移除任何元素，則回傳空陣列

splice() 在特定位置新增元素

- 有 months 陣列 `months = ['Jan', 'March', 'April', 'June'];`
- 缺了二月，請將二月加到 Jan 和 March 之間

```
let months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
console.log(months); // [ 'Jan', 'Feb', 'March', 'April', 'June' ]
```

Syntax:

```
array.splice(start, deleteCount, item1, item2, ...);
```

- `start`: 開始位置的索引值
- `deleteCount`: 要刪除的元素數量
- `item1, item2, ...`: 要新增的多個元素清單

splice() 更新特定的元素

- 更新 Feb 及 March 為 February 及 March
- 沒有直接的更新，必須先刪除再新增

```
let months = ['Jan', 'Feb', 'Mar', 'Apr', 'Jun'];
months.splice(1, 2, 'February', 'March');
console.log(months); // [ 'Jan', 'Febuary', 'March', 'April', 'June' ]
```

splice() 刪除特定的元素

- 刪除 Apr 及 Jun
- 注意: 會改變原始陣列的內容

```
let months = ['Jan', 'Feb', 'Mar', 'Apr', 'Jun'];
removedElm = months.splice(3, 2);
console.log(months); // [ 'Jan', 'Feb', 'Mar' ]
console.log(removedElm); // [ 'Apr', 'Jun' ]
```

Quick Practice

給定以下的陣列: `['A', 'B', 'C']`,請插入值 `'1'`, `'2'` 使其變成 `['A', '1', '2', 'B', 'C']`。

▶ 參考答案

slice() 取出某個範圍的元素

- `slice(start, end)` 方法用來取出陣列中某個範圍的元素
- 回傳一個新的陣列
- 不會改變原始陣列的內容

Syntax:

```
slice()  // 取出整個陣列
slice(start)  // 取出從 start 開始到陣列的最後一個元素
slice(-start) // 取出後面的 n 個元素 (從倒數 start 的位置取到最後一個元素)
slice(start, end) // 取出從 start 開始到 end - 1 的元素
```

由**前往後**及由**後往前**的 index

```
read from start --->
   0     1     2     3     4
|     |     |     |     |     |
|  S  |  L  |  I  |  C  |  E  |
|     |     |     |     |     |
  -5    -4    -3    -2    -1
<--- read from reverse
```

Example

```
// 從 index 2 開始取到最後一個元素
const fruits = ["Apple", "Banana", "Orange", "Mango", "Pineapple"];

const tropical = fruits.slice(2);
console.log(tropical); // ['Orange', 'Mango', 'Pineapple']

// 從倒數第 2 個元素開始取到最後一個元素
const lastTwo = fruits.slice(-2);
console.log(lastTwo); // ['Mango', 'Pineapple']
```

Quick Practice

有以下的 URL 字串 `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice`

使用 split('/') 方法將字串切割成陣列，並使用 slice() 方法取出:

1. `Array` 及 `slice` 的部分。
2. domain name 及語系 的部分: `developer.mozilla.org` and `en-US`

▶ 參考答案

... 展開運算子 將陣列轉成值清單

- `...` 運算子(三個點)是 ES6 新增的語法
- 也稱為 展開運算子(spread operator)
- 可以將陣列轉換為清單值

`['A', 'B', 'C']` 轉換為 `'A', 'B', 'C'`

使用情境 1

- 將陣列轉換為函數的引數

將 ['1', '2', '3'] 插入到 陣列 ['D', 'E', 'F'] 的 D 和 E 之間。
使用 splice() 方法

```
let arr1 = ['1', '2', '3'];
let arr2 = ['D', 'E', 'F'];
arr2.splice(1, 0, ...arr1);
console.log(arr2); // [ 'D', '1', '2', '3', 'E', 'F' ]
```

如果沒有展開運算子，會將整個陣列當成一個值

```
let arr1 = ['1', '2', '3'];
let arr2 = ['D', 'E', 'F'];
arr2.splice(1, 0, arr1);
console.log(arr2); // [ 'D', [ '1', '2', '3' ], 'E', 'F' ]
```

使用情境 2

- 在 Array Literal 中使用展開運算子, 將某個陣列的內容加入到另一個陣列中

```
let arr1 = ['1', '2', '3'];
let arr2 = ['D', 'E', 'F', ...arr1];
console.log(arr2); // [ 'D', 'E', 'F', '1', '2', '3' ]
```

Concat() 串接兩個陣列內的元素

Scenario: 有多個陣列, 想要將他們的元素串接在一起, 變成一個陣列

- 使用 `concat()` 方法
- `concat()` 方法會回傳一個新的陣列
- 不會改變原始陣列的內容

syntax:

```
concat()  //  回傳目前的陣列的 copy
concat(value1)  //
concat(value1, value2)
concat(value1, value2, /* …, */ valueN)
```

- value 可以是陣列或值

```
let arr1 = ['A', 'B', 'C'];
let arr2 = ['D', 'E', 'F'];
```

join() 串接陣列內的元素

- `join()` 方法用來將陣列中的元素串接成一個字串
- 不會改變原始陣列的內容
- 回傳串接後的字串

Syntax:

```
join() // 使用預設的逗號分隔
join(separator) // 使用指定的分隔符號串接
```

Ex. 將陣列中的元素串接成一個字串, 用 '-' 分隔

```
let strs = Array.from('hello');
let str = strs.join('-');
console.log(str); // h-e-l-l-o
```

排序元素

- `sort()` 將陣列中的元素進行排序, 預設由小到大
- `reverse()` 將陣列中的元素反轉

注意:

1. 會改變原始陣列的內容
2. 預設會將內容轉成字串進行排序
    - 若要使用其他的排序方式，則需要提供一個比較函數

Sort syntax:

```
sort()
sort(compareFn)
```

排序字串

將陣列中的內容由小到大排序

```
let arr = [1, 100, 2, 12 , 21]
arr.sort();
console.log(arr); // [ 1, 100, 12, 2, 21 ]
```

- 會將內容轉成字串進行排序

排序數字

- 若要使用數字進行排序，則需要提供一個比較函數
  - a > b 時，回傳正數,
  - a < b 時，回傳負數,
  - a = b 時，回傳 0

```
function compValue(a, b) {
    return a - b;
}

let arr = [1, 100, 2, 12 , 21]
arr.sort(compValue);
console.log(arr); // [ 1, 2, 12, 21, 100 ]
```

Quick Practice

將字串 'Hello World' 反轉成 'dlroW olleH'

Hints:

- 將字串轉成 char array
- 使用 reverse() 方法反轉陣列
- 使用 join() 方法將陣列轉成字串

▶ 參考答案

11 陣列的迭代方法 (Iterative Methods)

迭代方法 (Iterative Methods)

- 陣列（Array）、Set、Map 等可迭代對象所提供的方法
- 他們會自動拜訪每個元素，並執行使用者提供的 callback 函數
- 迭代方法背後的維:
  - 使用函數轉換每一個原始的元素，得到一個新的陣列或集合
  - 不會改變原始的陣列或集合

為什麼要使用迭代方法？

- 迭代方法（Iteration Methods） 提供了一種更簡潔、更直覺、更可讀的方式來遍歷和處理陣列數據。
- 相比於傳統的 for 迴圈或 while 迴圈，迭代方法讓代碼更清晰、簡短，並且減少了錯誤的可能性。

Ex: 串接多個迭代方法完成工作

過濾陣列中的偶數，並將每個偶數平方

```
const numbers = [1, 2, 3, 4, 5, 6];
const squaredEvens = numbers
    .filter(num => num % 2 === 0)
    .map(num => num ** 2);

console.log(squaredEvens); // [4, 16, 36]
```

使用 for loop 的話，會變得冗長不直觀

```
const numbers = [1, 2, 3, 4, 5, 6];
let squaredEvens = [];
for (let i = 0; i < numbers.length; i++) {
    if (numbers[i] % 2 === 0) {
        squaredEvens.push(numbers[i] ** 2);
    }
}
console.log(squaredEvens); // [4, 16, 36]
```

迭代方法 概覽

- 過濾與尋找元素:
  - `filter()`: 過濾陣列中的元素
  - `find()`: 找到陣列中的第一個符合條件的元素
  - `findIndex()`: 找到陣列中的第一個符合條件的元素的索引值
  - `some()`: 判斷陣列中是否有至少一個元素符合條件
  - `every()`: 判斷陣列中是否所有元素都符合條件
- 元素轉換:
  - `map()`: 將陣列中的每個元素轉換為新的值
- 對每個元素執行操作:
  - `forEach()`: 對陣列中的每個元素執行操作
- 將多個元素變成一個值:
  - `reduce()`: 將陣列中的所有元素轉換為一個值(e.g. 加總)

12 過濾與尋找元素

filter() 過濾陣列中的元素

情境: 陣列中存放學生成績, 找出大於 70 的成績

思維: 迭代動作與過濾邏輯分開

設計一函數，傳入一個數字，判斷是否大於 70

```
function isPass(score) {
    return score > 70;
}
```

- 將 `isPass()` 函數傳入 `filter()` 方法中，過濾出大於 70 的成績

```
let scores = [60, 70, 80, 90];
let passScores = scores.filter(isPass);
console.log(passScores); // [ 80, 90 ]
```

使用 lambda 函數精簡程式碼

如果 `isPass()` 函數只會用在 `filter()` 方法中，可以使用 lambda 函數來精簡程式碼

```
let scores = [60, 70, 80, 90];
let passScores = scores.filter(score => score > 70);
console.log(passScores); // [ 80, 90 ]
```

lambda 函數概述

- Lambda 函數是一種匿名函數(沒有名稱的函數)
- `=>` 的左邊是參數，右邊是函數的主體
- 若參數只有一個，則可以省略括號
- 若只有一個運算式，則可以省略大括號和 return 關鍵字

運作過程

- `filter()` 方法會遍歷陣列中的每個元素
- 對每個元素執行傳入的 `isPass()` 函數
  - 如果 `isPass()` 函數回傳 true，則將該元素加入到新的陣列中
  - 如果回傳 false，則不加入
- `filter()` 方法會回傳一個新的陣列，包含所有符合條件的元素
- 不會改變原始陣列的內容

every() 所有元素是否都符合條件

情境: 學生的成績是否都大於 60 呢?

- 使用 `every()` 方法, 回傳 boolean 值

```
let scores = [60, 70, 80, 90];
let isAllPass = scores.every(score => score > 60);
console.log(isAllPass); // false
```

Quick Practice

有以下的銷售金額, 是不是所有金額都大於 1000 呢?
如果不是, 請印出所有小於 1000 的金額

▶ 參考答案

13 元素轉換

情境

調整學生的成績，每位學生加 5 分

原始成績: `scores = [60, 70, 80, 90];`

新的成績: `newScores = [65, 75, 85, 95];`

每個元素的運算邏輯: `x => x + 5`

- 傳入 x, 回傳 `x + 5`

map() 方法

- `map()` 方法用來將陣列中的每個元素轉換為新的值
  - 回傳一個新的陣列

```
let scores = [60, 70, 80, 90];
let newScores = scores.map(score => score + 5);
console.log(newScores); // [ 65, 75, 85, 95 ]
```

14 對每個元素執行操作

情境

將學生成績格式化輸出: `成績 xx, 及格/不及格`

- 原始成績: `scores = [60, 70, 80, 90];`
- 輸出:
  - `成績 60, 不及格`
  - `成績 70, 及格`
  - `成績 80, 及格`
  - `成績 90, 及格`

每個元素的執行操作:

```
score => console.log(
  `成績 ${score}, ${score > 70 ? '及格' : '不及格'}`)`)
```

其它情境

- 將元素寫到檔案中
- 將元素傳到伺服器
- 將元素印出來

forEach() 方法

- 使用 `forEach()` 方法對陣列中的每個元素執行操作
  - 不會回傳任何值

```
let scores = [60, 70, 80, 90];
scores.forEach(score => console.log(
  `成績 ${score}, ${score > 70 ? '及格' : '不及格'}`));
```

Quick Practice

銷售金額 [1200, 1500, 800, 2000, 500]。
超過 1000 的為 VIP 客戶，其它的為 一般客戶。

將金額資料改成 [1200(VIP), 1500(VIP), 800(一般), 2000(VIP), 500(一般)].
接著印出這些資料。

▶ 參考答案

15 對所有元素逐一進行歸納

情境

成績資料 [60, 70, 80, 90]

- 算出總和
  - 逐一累加，得到總合
- 找出最大/最小值
  - 逐一比大小，得到最大或最小值
- 找出高於 70 分的人數
  - 逐一判斷，得到符合條件的元素

典型歸納函數

傳入一個歸納函數給 `reduce()` 方法

典型歸納函數的簽名:

```
function reducer(accumulator, currentValue) {
    // ...
    // 回傳歸納的結果
    return newAccumulator;
}
```

- `accumulator`: 累加器，累加的結果
- `currentValue`: 當前元素的值
- 回傳: 新的累加器的值

補充: 完整的簽名

```
function reducer(accumulator, currentValue, currentIndex, array) {
    // ...
    // 回傳歸納的結果
    return newAccumulator;
}
```

- `accumulator` 和 `currentValue` 是必須的參數
- `currentIndex`: 當前元素的索引值
- `array`: 原始陣列的參考

歸納器範例

歸納總合的

```
function sum(accumulator, currentValue) {
    return accumulator + currentValue;
}
// lambda function
(accumulator, currentValue) => accumulator + currentValue
```

歸納最大值的

```
function max(accumulator, currentValue) {
    return Math.max(accumulator, currentValue);
}
// lambda function
(accumulator, currentValue) => Math.max(accumulator, currentValue)
```

歸納符合某個條件的元素的數量

```
function count(accumulator, currentValue) {
    return currentValue > 70 ? accumulator + 1 : accumulator;
}
// lambda function
(accumulator, currentValue) => currentValue > 70 ? accumulator + 1 : accumulator
```

reduce() 方法

- `reduce()` 方法用來對陣列中的每個元素執行歸納操作
- 回傳一個新的值
- 不會改變原始陣列的內容
- 可以指定累加器的初始值
  - 如果沒有指定，則使用陣列中的第一個元素作為初始值

Syntax:

```
reduce(callbackFn)
reduce(callbackFn, initialValue)
```

Ex. 算出成績的總和

```
let scores = [60, 70, 80, 90];
let total = scores.reduce(
  (accumulator, currentValue) => accumulator + currentValue);
console.log(total); // 300
```

Ex. 找出成績大於 70 的學生人數

```
let scores = [60, 70, 80, 90];
let count = scores.reduce(
  (accumulator, currentValue) => currentValue > 70 ? accumulator + 1 : accumulator, 0);
console.log(count); // 3
```

Quick Practice

銷售金額 [1200, 1500, 800, 2000, 500]。

找出最低的金額，並印出來。

▶ 參考答案

16 自行迭代

- 如果迭代方法不符合需求，可以取得 Array 的迭代器，然後自行迭代
- 使用 `for...of` 對迭代器自動迭代，遍歷每個元素

Syntax:

```
for (const element of iterable) {
    // 對每個元素執行操作
}
```

- iterable: 可迭代的對象

Array 提供的 Iterators

- `entries()`: 取得陣列 key/value 的迭代器
  - 假設陣列為 `['A', 'B', 'C']`
  - entries() 的 iterator 逐一回傳的資料 [0, 'A'], [1, 'B'], [2, 'C']
- `keys()`: 取得陣列的索引值的迭代器
  - 逐一回傳的資料 [0, 1, 2]
- `values()`: 取得陣列的值的迭代器
  - 逐一回傳的資料 ['A', 'B', 'C']

entries() 迭代器

情境: 迭代時要取得元素的索引及值

Ex. 印出學生成績的索引及元素值，輸出格式為: `學生 index: xx, 成績: xx`

Array 的 forEach() 沒有提供索引值給回呼函數, 所以我們要使用 entries()

```
let scores = [60, 70, 80, 90];
for (const [index, score] of scores.entries()) {
    console.log(`學生 index: ${index}, 成績: ${score}`);
}
```

- `for...of` 迴圈會自動迭代 entries() 的迭代器
- `[index, score]` 是解構賦值，將迭代器的回傳值解構為 index 和 score

補充: 解構賦值(自行閱讀)

- 情境: 如果要將一個陣列的值指派給多個變數, 要如何寫會比較簡潔?
- Examples:
  - [1, 2, 3] 的值要指派給 a, b, c 三個變數
  - [1, 2, 3] 中 1 和 3 的值要指派給 a 和 b 兩個變數, 2 捨棄
- ES 6 提供了解構賦值(Destructuring Assignment)的語法
  - 在指派符號的左右兩側, 皆可使用陣列或物件
  - 陣列物件會以位置對應的方式進行指派
  - 一般物件會以鍵值對應的方式進行指派

Ex. [1, 2, 3] 的值要指派給 a, b, c 三個變數

```
let [a, b, c] = [1, 2, 3];
```

Ex. [1, 2, 3] 中 1 和 3 的值要指派給 a 和 b 兩個變數, 2 捨棄

```
let [a, , b] = [1, 2, 3];
```

- 也可使用 `...` 展開運算子(Spread operator) 儲存剩餘的值到另個陣列中

Ex. 將 [1, 2, 3] 中 的 1 指派到 a, [2, 3] 放到另個陣列中 subArr 中

```
let [a, ...subArr] = [1, 2, 3];
console.log(a); // 1
console.log(subArr); // [ 2, 3 ]
```

keys() 迭代器

- 取得陣列的索引值的迭代器
- 如果用於 Map 物件，則會取得 Map 中的鍵值的迭代器

Ex. 取得 Map 物件的鍵值

```
let map = new Map([
    ['name', 'John'],
    ['age', 30],
    ['city', 'New York']
]);

for (const key of map.keys()) {
    console.log(key); // name, age, city
}
```

values() 迭代器

- 取得陣列的值的迭代器
- 如果用於 Map 物件，則會取得 Map 中的值的迭代器

Ex. 取得 Map 物件的值

```
let map = new Map([
    ['name', 'John'],
    ['age', 30],
    ['city', 'New York']
]);
for (const value of map.values()) {
    console.log(value); // John, 30, New York
}
```

Quick Review

- Array 物件提供了那些類型的迭代方法？舉例使用說明情境
- 想要把陣列中的每個元素的值除以 100 轉換成百分比, 應該使用那個迭代方法？為什麼?
- Array 物件提供了哪些迭代器？

17 本章內容回顧

- 建立 Array 的不同方法
  - Array Literal, Array.of(), Array.from(), Array constructor
- 如何訪問陣列中的元素
  - arr[x]
- 陣列內容的修改與新增
  - arr[x] = value, arr.push(value), arr.pop(), arr.shift(), arr.unshift(value)
- 遍歷陣列元素的方法
  - for/of loop, forEach()
- 陣列進階操作方法
  - splice(), slice(), concat(), join(), sort(), reverse()
- 陣列的迭代方法
  - filter(), find(), findIndex(), some(), every(), map(), forEach(), reduce()