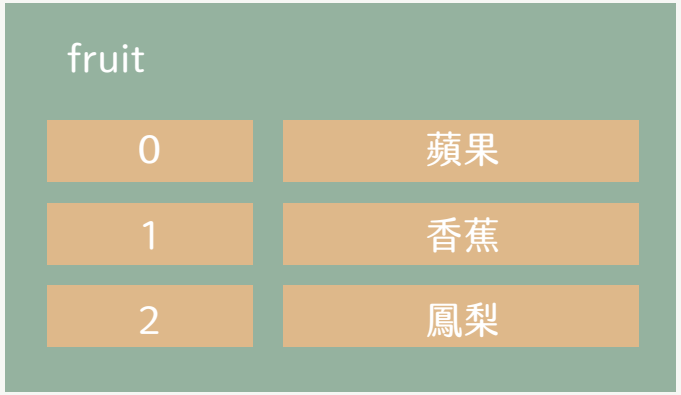
物件及陣列基礎說明

JavaScript 基本資料管理結構

陣列

Array [] 陣列：依編號排列，從 0 開始

```
const fruits = [
  '蘋果', '香蕉', '鳳梨'
]
```

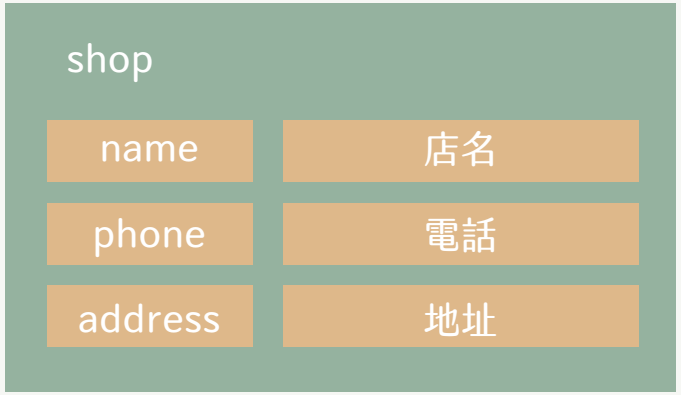



```
console.log(fruits[0]) // 蘋果
console.log(fruits[1]) // 香蕉
console.log(fruits[2]) // 鳳梨
```

物件

Object {} 物件：有名字的資料 => keyname: value

```
const shop = {
  name: '店名',
  phone: '電話',
  address: '地址'
}
```




```
console.log(shop[name]) // 店名
console.log(shop.phone) // 電話
console.log(shop.address) // 地址
```

陣列物件相互關係

陣列裡有物件

```
const fruits = [
  {
    name: 'apple',
    price: 100
  },
  {
    name: 'berry',
    price: 200
  },
  {
    name: 'lemon',
    price: 30
  }
]
```

物件裡有陣列

```
const order = {
  name: 'Jarvis',
  phone: '09123123',
  cart: ['apple', 'berry', 'lemon']
}
```

字串、陣列處理及批次操作

進階 JavaScript 資料管理

字串處理

indexOf( ) // 尋找文字

```
const fruits = 'apple&berry&lemon'
fruits.indexOf('apple') // 0
fruits.indexOf('berry') // 6
fruits.indexOf('banana') // -1
```

slice(開始, 結束之前) // 切割文字

```
const sayHello = 'helloworld'
sayHello.slice(0, 5) // 'hello'
```

substr(開始, 長度) // 切割文字

```
const sayHello = 'helloworld'
sayHello.substr(2, 3) // 'llo'
```

replace(原始, 替換) // 取代文字

```
const sayHello = 'helloworld'
sayHello.replace('world', 'apple') // 'helloapple'
const sayHello2 = 'helloworldworld'
sayHello2.replace('world', 'apple') // 'helloappleworld'（只會取代第一個）
sayHello2.replace(/world/g, 'apple') // 'helloappleapple'（使用正規表達式）
```

split( ) // 分割成陣列

```
const fruits = 'apple/berry/lemon'
fruits.split('/')
// ['apple', 'berry', 'lemon']
```

正規表達式

https://www.runoob.com/regexp/regexp-syntax.html

https://blog.poychang.net/note-regular-expression/

正規表達式測試站

https://regex101.com/

身分證驗證

```
^[A-Z](1|2)\d{8}$
```

陣列處理（產生值或新陣列）

length // 陣列長度

```
const fruits = [
  'apple', 'berry', 'lemon'
]
console.log(fruits.length) // 3
```

indexOf( ) // 尋找資料

```
const fruits = [
  'apple', 'berry', 'lemon'
]
fruits.indexOf('apple') // 0
fruits.indexOf('berry') // 1
fruits.indexOf('banana') // -1
```

concat( ) // 合併陣列

```
const fruits1 = ['apple', 'berry']
const fruits2 = ['lemon', 'banana']
const fruits3 = fruits1.concat(fruits2)
console.log(fruits3)
// ['apple', 'berry', 'lemon', 'banana']
```

slice(開始, 結束之前) // 截取

```
const fruits = [
  'apple', 'berry', 'lemon'
]
const result = fruits.slice(0, 2)
console.log(result)
// ['apple', 'berry']
```

includes( ) // 尋找資料

```
const fruits = [
  'apple', 'berry', 'lemon'
]
fruits.includes('apple') // true
fruits.includes('berry') // true
fruits.includes('banana') // false
```

join( ) // 陣列轉字串

```
const fruits = [
  'apple', 'berry', 'lemon'
]
const stringFruits = fruits.join('&')
console.log(stringFruits)
// 'apple&berry&lemon'
```

陣列處理（將改變原始陣列）

push( ) // 新增元素至末段

```
const fruits = ['apple', 'berry']
fruits.push('lemon')
console.log(fruits)
// ['apple', 'berry', 'lemon']
```

unshift( ) // 新增元素至第一位

```
const fruits = ['berry', 'lemon']
fruits.unshift('apple')
console.log(fruits)
// ['apple', 'berry', 'lemon']
```

reverse( ) // 反轉陣列

```
const fruits = ['apple', 'berry', 'lemon']
fruits.reverse()
console.log(fruits)
// ['lemon', 'berry', 'apple']
```

pop( ) // 移除並回傳最後一個元素

```
const fruits = [
  'apple', 'berry', 'lemon'
]
const popItem = fruits.pop()
console.log(fruits)
// ['apple', 'berry']
console.log(popItem) // 'lemon'
```

shift( ) // 移除並回傳第一個元素

```
const fruits = [
  'apple', 'berry', 'lemon'
]
const shiftItem = fruits.shift()
console.log(fruits)
// ['berry', 'lemon']
console.log(shiftItem) // 'apple'
```

splice(位置, 數量, 取代) // 移除元素

```
const fruits = ['apple', 'berry', 'lemon']
fruits.splice(1, 1)
// ['apple', 'lemon']
fruits.splice(1, 1, 'banana')
// ['apple', 'banana', 'lemon']
fruits.splice(1, 0, 'banana')
// ['apple', 'banana', 'berry', 'lemon']
```

陣列批次操作

Array.forEach(函式)

將陣列元素一個一個抓出來，帶入函式執行（類似for迴圈）
陣列元素如果是物件可直接改變其值，不會產生新陣列

```
const fruits = [
  { name: 'apple', price: 100 },
  { name: 'berry', price: 200 },
  { name: 'lemon', price: 30 }
]
fruits.forEach(function(item, index) {
  if (item.price > 50) item.price += 10
})
console.log(fruits)
// [
//   { name: 'apple', price: 110 },
//   { name: 'berry', price: 210 },
//   { name: 'lemon', price: 30 }
// ]
```

Array.map(函式)

將陣列元素一個一個抓出來，
回傳自訂義的值，並將其建立成新陣列

```
const number = [1, 2, 3, 4, 5]
const newNum = number.map(function(num, index) {
  return num * num
})
console.log(newNum)
// [1, 4, 9, 16, 25]
```

陣列批次操作

Array.filter(函式)

根據函式過濾符合條件的元素，並建立成新陣列

```
const fruits = [
  { name: 'apple', price: 100 },
  { name: 'berry', price: 200 },
  { name: 'lemon', price: 30 }
]
const result = fruits.filter(function(item) {
  console.log(item.price >= 100)
  return item.price >= 100
})
console.log(result)
// [
//   { name: 'apple', price: 100 },
//   { name: 'berry', price: 200 }
// ]
```

Array.find(函式)

根據函式回傳第一個符合條件的元素

```
const fruits = [
  { name: 'apple', price: 100 },
  { name: 'berry', price: 200 },
  { name: 'lemon', price: 30 }
]
const result = fruits.find(function(item) {
  return item.name.indexOf('berry') > -1
})
console.log(result)
// [{ name: 'berry', price: 200 }]
```

陣列批次操作

Array.some(函式)

陣列中只要有符合條件的元素就回傳true，否則false

```
const fruits = [
  { name: 'apple', price: 100 },
  { name: 'berry', price: 200 },
  { name: 'lemon', price: 30 }
]
const result = fruits.some(function(item) {
  return item.price > 100
})
console.log(result)  // true
```

Array.every(函式)

陣列中每一個元素皆符合條件就回傳true，否則false

```
const fruits = [
  { name: 'apple', price: 100 },
  { name: 'berry', price: 200 },
  { name: 'lemon', price: 30 }
]
const result = fruits.every(function(item) {
  return item.price > 100
})
console.log(result) // false
```

陣列批次操作

Array.reduce(函式)

根據元素值與暫存做運算並回傳結果

```
const fruits = [
  { name: 'apple', price: 100 },
  { name: 'berry', price: 200 },
  { name: 'lemon', price: 30 }
]
const result = fruits.reduce(function(cache, item) {
  return cache + item.price
}, 0) // 0 為 cache 初始值
console.log(result) // 330
```

Array.sort(函式)

根據函式回傳值排列陣列（將改變原始陣列）

```
// 升冪排序
fruits.sort(function(a, b) {
  return a.price - b.price
})
// 降冪排序
fruits.sort(function(a, b) {
  return b.price - a.price
})
// 自定義排序
fruits.sort(function(a, b) {
  if (a.price < b.price) return -1
  if (a.price > b.price) return 1
  return 0
})
```

物件拷貝

相同記憶體

只有位置改變，但記憶體相同

```
const objA = { a: 1 }
const objB = objA

objB.a = 2
console.log(objA.a) // 2
```

淺拷貝

可成功拷貝，不過僅限一層，故稱淺拷貝；
所有產生新陣列的批次操作皆為淺拷貝

```
const objA = { a: 1, b: { b1: 2 } }

// 直接賦值
const objB = { a: objA.a, b: objA.b }
// ES6
const objB = Object.assign({}, objA)

objB.a = 2
console.log(objA.a) // 1

objA.b.b1 = 4
console.log(objB.b) // { b1: 4 }
```

深拷貝

完全拷貝，源物件與拷貝物件互相獨立

```
const objA = { a: 1, b: { b1: 2 } }

// 將物件轉成字串再轉回來
const objB = JSON.parse(JSON.stringify(objA))
// jQuery
const objB = $.extend(true, {}, objA)
// Lodash (JavaScript Plugin)
const objB = _.cloneDeep(objA)

objA.b.b1 = 4

console.log(objB.b) // { b1: 2 }
```

專案實作

學程式都要寫過的 TodoList