# 迴圈

迴圈(loop)是"重覆作某個事"的語句結構，用比較詳細的解釋，則是"在一個滿足的條件下，重覆作某件事幾次"這樣的語句。

迴圈語句經常會搭配陣列與物件之類的集合性資料結構使用，重覆地或循環地取出某些滿足條件的值，或是重覆性的更動其中某些值。

## for語句

for語句需要三個表達式作為參數，它們各自有其功用，分別以分號(;)作為區分。for迴圈的基本語法如下:

```
for ([initialExpression]; [condition]; [incrementExpression])
  statement
```

- initialExpression: 初始(開始時)情況的表達式
- condition: 判斷情況或測試條件，代表可執行for中包含執行動作的滿足條件
- incrementExpression: 更新表達式，每次for迴圈中的執行動作語句完成即會執行一次這個表達式

一個簡單的範例如下:

```
for (let count = 0 ; count < 10 ; count++){
    console.log(count)
}
```

整個程式碼的解說是:

- 在迴圈一開始時，將 `count` 變數指定為數字 `0`
- 定義當 `count < 10` 時，才能滿足執行 `for` 中區塊語句的條件。也就是只有在 `count < 10` 時才能執行for中區塊語句。
- 每次當執行 `for` 中區塊的語句執行後，即會執行 `count++`

### 可自訂的表達式們

`for` 語句中共有三個表達式，分別代表迴圈進行時的設定與功用，這三個表達式的運用方式就可以視不同的應用情況而調整。所有的三個表達式都是可自訂的，也是可有可無。下面這個看起來怪異的 `for` 語句是一個無窮迴圈，執行這個語句會讓你的瀏覽器當掉:

```
//錯誤示範
for (;;){
    console.log('bahbahbah')
}
```

> 註: 用 `for(;;)` 這樣的語法，基本上失去 `for` 語句原本的意義，完全不建議使用。

> 注意: 迴圈是一個破壞性相當高的語句，如果不小心很容易造成整個程式錯誤或當掉。

第一個表達式是用作初始值的定義。它是可以設定多個定義值的，每個定義值之間使用逗號(,)作為分隔，定義值可使用的範圍只在迴圈內部的語句中:

```
for (let count = 0, total = 10 ; count < 10 ; count++){
    console.log(count, total)
}
```

> 註: 初始值同樣也可以定義在迴圈之外，但它的作用範圍會擴大到迴圈之外。

第二個表達式是用於判斷情況(condition)，也就是每次執行迴圈中區塊的語句前，都會來測試(檢查)一下，是不是滿足其中的條件，如果滿足的話(布林 `true` 值)，就執行迴圈中區塊的語句，不滿足的話(布林 `false` 值)，就跳出迴圈語句或略過它。這個表達式就是會造成無止盡的或重覆執行次數不正確的原兇，所以它是這三個表達式中要特別注意的一個。

由於它是一個標準的判斷情況(condition)表達式，如果有多個判斷情況的時候，要使用邏輯與(&&)和邏輯或(||)來連接，邏輯或(||)因為結果會是 `true` 的情況較為容易，所以要更加小心:

```
for (let count = 0, total = 10 ; count < 10 && total < 20 ; count++){
    console.log(count, total)
}
```

第三個表達式是用於更新的，常被稱為遞增表達式(incrementExpression)。不過它也不只用在遞增(increment)，應該說用在"更動(update)"或許會比較恰當，每次一執行完迴圈內部語句就會執行一次的表達式。它的作用相當於在迴圈內的區塊中的最後面加上這個語句，例如下面這個for迴圈和最上面一開始的範例的結果是一樣的:

```
for (let count = 0 ; count < 10 ;){
    console.log(count)
    count++
}
```

多個表達式也是可以的，同樣也是用逗點(,)分隔不同的表達式:

```
for (let count = 0, total = 30 ; count < 10 && total > 20 ; count++, total--){
    console.log(count, total)
}
```

這裡有個小小問題，是有關於遞增運算符(++)與遞減運算符(--)，它們運算元的位置差異，放在運算元的前面和後面差異是什麼？例如以下的範例:

```
let x = 1
let y = 1

console.log(x++) //1
console.log(x) //2

console.log(++y) //2
console.log(y) //2
```

所以放在運算元(值)前面(先)的遞增(++)或遞減(--)符號，就會先把值作加1或減1，也就會直接變動值。放後面的就是值的變動要在下一次再看得到。

> 口訣: 遞增減運算符(++/--) "錢仙"=(放)前(面)先(變動)

### 小結

實際上這三個表達式並沒有限定只能用哪些表達式，只要是合法的表達式都可以，只不過第二個表達式會作判斷情況，也就是轉換為布林值，這一點要注意的。有些程式設計師喜歡用這個特性寫出感覺很高超的語法，只是讓別人更難看懂在寫什麼而已，我完全不認同這種寫法，如何提供高閱讀性的程式碼才是最好的寫法。

至於每個表達式執行的情況，第一個表達式(初始化值)只會在讀取到 `for` 語句時執行一次，之後不會再執行。第二個表達式(判斷情況)會在讀取到 `for` 語句時執行一次，之後每次重覆開始時都會執行一次。第三個表達式會在有滿足條件下，執行到 `for` 語句區塊內部的語句的最後才執行。

## 多重迴圈

所謂的多重迴圈是巢狀的迴圈語句結構，也就是在迴圈的區塊語句中，再加上另一個內部的迴圈語句。典型的範例就是九九乘法表:

```
for (let i = 1 ; i < 10 ; i++){
  for (let j = 1 ; j < 10 ; j++){
    console.log(i + ' x ' + j + ' = ', i * j )
  }
}
```

> 註: 下面的 `while` 語句與 `do...while` 語句也可以寫為多重迴圈(巢狀迴圈)，類似上面的程式碼，就不再多重覆說明了。

## while語句

相較於需要很確定重覆次數(重覆次數為可被計算的)的 `for` 迴圈， `while` 迴圈語句可以說它是 `for` 迴圈語句的簡單版本，你可以根據不同的使用情況選擇要使用哪一種。它只需要一個判斷情況(condition)的表達式即可，它的基本語法結構如下:

```
while (condition)
  statement
```

一個簡單的範例如下，這個範例與之前的for迴圈的範例功能是相等的:

```
let count = 0

while (count < 10) {
    console.log(count)
    count++
}
```

while語句沒什麼特別要說明的，它把更新用表達式的部份交給程式設計師自行處理，在while語句中的區塊語句中再加上。使用時仍要避免出現無窮迴圈或重覆次數不正確的情況。

# do...while語句

`do...while` 語句是 `while` 語句的另一種變形的版本，差異只在於"它會先執行一次區塊中的語句，再進行判斷情況"，也就是會"保証至少執行一次其中的語句"，它的基本語法結構如下:

```
do
  statement
while (condition)
```

正常的使用情況與while語句沒什麼差異，只是位置上下顛倒再加個 `do` 而已:

```
let count = 0

do {
  console.log(count)
  count++
}
while (count < 10)
```

實際上 `while` 語句如果要相等於 `do...while` 的語句，應該是如下面這樣，也就是while語句前要先執行一次 `do...while` 中的語句才能算相等:

```
do_statment
while (condition)
  do_statment
```

因為 `do...while` 語句具有先執行語句再檢查，也就是保証會執行一次的特性，要視應用情況來使用，以簡單的實際例子來說明 `while` 與 `do...while` 語句的應用情況:

- `while`：當有一群人要進入電影院，需要對每個入場者收取門票與檢查，然後才能入場
- `do...while`：當有幾組參賽者參加歌唱比賽，需要先表演後再對每個參賽者評分

# for...in語句

`for...in` 語句主要是搭配物件類型使用的，不要使用在陣列資料類型。它是用於迭代物件的可列舉的屬性值(enumerable properties)，而且是任意順序的。因為陣列資料在進行迭代時，順序是很重要的，所以陣列資料類型不會用這個語句，而且陣列資料類型本身就有太多好用的迭代方法。

另外， `for` 迴圈語句完全可以取代 `for...in` 語句，而且 `for` 迴圈語句可以保証順序，所以這個語句算是不一定要使用的，在這裡說明只是單純的比較其他的語句而已。

> 註: `for...in` 語句是任意順序的迴圈語句，使用時通常是用來檢測是的語句

> 註: 對於複雜的物件資料類型，單純使用JavaScript中的語言特性是常常有所不足，建議是使用其他的函式庫中的API，例如jQuery、Lodash或underscore.js。

# for...of語句

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

```
//用於陣列
let aArray = [1, 2, 3]

for (let value of aArray) {
  console.log(value)
}

//用於字串
let aString = "abcd";

for (let value of aString) {
  console.log(value);
}
```

## break與continue

`break` (中斷)與 `continue` (繼續)是用於迴圈區塊中語句的控制，在 `switch` 語句中也有看過 `break` 這個關鍵字的用法，是一個跳出 `switch` 語句的語法。

`break` 是"中斷整個迴圈語句"的意思，可以中斷整個迴圈，也就是"跳出"迴圈區塊的語句，如果是在巢狀迴圈時是跳出最近的一層。 `break` 通常會用在迴圈語句中的 `if` 判斷語句裡，例如下面這個例子:

```
let count = 0

while (count < 10) {
    console.log(count)
    //count的值為6時將會跳出迴圈
    if(count === 6) break
    count++
}
```

`continue` 是"繼續下一次迴圈語句"的意思，它會忽略在 `continue` 之下的語句，直接跳到下一次的重覆執行語句的最開頭。因為 `continue` 有"隱含的 `goto` 到迴圈的最開頭程式碼"，它算是一個在JavaScript語言中的壞特性，它會讓你的判斷情況(condition)整個無效化，也可以破壞整體的迴圈語句執行結構，容易被濫用出現不被預期的結果，結論是能不用就不要使用。一個簡單的範例如下:

```
let count = 0
let a = 0

while (count < 10) {
    console.log('count = ', count)
    console.log('a = ', a)

    count++
    //count的值大於為6時將會不再遞增a變數的值
    if(count > 6) continue
    a++
}
```

> 註: 雖然JavaScript語言中並沒有 `goto` 語法，但迴圈語句中的 `continue` 搭配 `labeled語句` ，相當於其他程式語言中的 `goto` 語法。 `goto` 語法幾乎在所有的程式語言中，都是惡名昭彰的一種語法結構。

# 風格指引

- (Airbnb 18.3) 在控制語句的圓括號開頭(()前放一個空格(if, while等等)。在函式呼叫與定義的函式名稱與傳入參數之間就不需要空格。eslint: keyword-spacing jscs: requireSpaceAfterKeywords
- (Airbnb 7.3) 絕對不要在非函式區塊中宣告一個函式(if, while等等)。改用指定一個函式給一個變數。瀏覽器可以允許你可以這樣作，但是壞消息是，不同瀏覽器可能會轉譯為不同的結果。eslint: no-loop-func
- (Google) for-in迴圈經常會錯誤地被使用在陣列上。當使用陣列時，總是使用一般的for迴圈。
- 在迴圈中區塊語句裡，定義變數/常數是個壞習慣，應該是要定義在迴圈外面或for迴圈的第一個表達式中。

```
//壞的寫法
for (let i=0; i<10; i++){
  let j = 0
  console.log(j)
  j++
}

//好的寫法
for (let i=0, let j = 0; i<10; i++){
  console.log(j)
  j++
}
```

- 用於判斷情況的運算，應該要儘可能避免，先作完運算在迴圈外部或或for迴圈的第一個表達式中，效率可以比較好些。

```
//較差的寫法
for (let i=0 ; i < aArray.length ; i++) {
  //statment
}

//較好的寫法
for (let i=0, len = aArray.length ; i < len; i++) {
  //statment
}

//另一種寫法，這種寫法腦袋要很清楚才看得懂
for (let i = aArray.length; i--;){
  //statment
}

//較差的寫法
let i = 0
while (i < aArray.length) {
  //statment
  i++
}

//較好的寫法
let i = 0
let len = aArray.length

while ( i < len) {
  //statment
  i++
}
```

> 註: 上面有一說法是在陣列資料類型的迴圈中的第三表達式(更新用表達式)中，使用i--會比i++效率更佳。實際上在現在的瀏覽器上這兩種的執行速度是根本一樣，沒什麼差異。

## 結語

## 家庭作業