控制流程

由於程式碼的執行順序，是由最上方的程式碼開始，往下逐行執行。有些時候，我們需要在其中特定的位置，判斷在當時執行期間的情況值，來決定之後程式執行的走向，這稱之為控制流程(Control Flow/康錯 佛肉/)的語法結構。

舉例來說，電腦中的文字冒險類遊戲(例如: 美少女遊戲Galgame)中，主人翁在關鍵時刻必須對攻略對象，在故事進行的對話選項間作出選擇，依照一連串不同的選擇，有可以導向最後是好的結局(GoodEnding)或是壞的結局(BadEnding)。這種依選項(判斷情況)不同，而導致不同的執行結果的程式碼語法結構，就是控制流程的結構。

Expression(表達式)

Expression/依斯伯累遜/ (表達式)是以字面文字(literal/立的羅/)、變數/常數名稱、值(運算元)、 運算符，或其他Expression(表達式)的組合體，最終能運算而計算求出(evaluates)一個值。

> Expression(表達式)代表任何合法的可計算產出值的程式碼單位

簡單的表達式範例如下，這看起有點像某行程式碼的一部份，或只是單純的字面文字(固定值的意思，例如 數字 或 字串):

```
'Hello'
3.1415
x = 7
3 + 5
```

Side Effect(副作用)

你如果有去醫院看過醫生拿過藥，在藥包上都會註明吃了這個藥物後會產生的副作用，Side Effect/塞的 依費特/(副作用)就是這個意思。Side Effect(副作用)這個詞，經常會出現在很多進階使用的JavaScript框架或函式庫之中，它算是一種概念，在表達式中有使用這個概念來進行分類。Expression(表達式)就上面所說的，是用來求出值的，但常見的一些Expression(表達式)的作用在指定值，而當在指定原本的變數/常數的Expression(表達式)值時，如果會變動原本的變數/常數內的值，就稱為是"**具有Side Effect(副作用)**"的Expression(表達式)，例如像下面這些具有副作用的表達式範例:

```
counter++
x += 3
y = "Hello " + name
```

沒有副作用的表達式的範例，它們大概都只是字面文字(literal)或變數/常數名稱，或是產生一個新的運算結果值:

```
3 + 5
true
1.9
x
x > y
'Hello World'
```

Side Effect(副作用)並不是指好或不好的意思，而是它有可能會影響到其他環境的使用情況，在使用時要特別注意這樣。它除了在表達式中有這個概念，我們在函式中也會再見到它。這裡就先大概了解表達式是如何區分有無副作用的。

Statements(語句)

Statements/斯疊門/(語句、陳述)是在程式語言中，一小段功能性的程式碼，語句中包含了關鍵字與合法的語法(Syntax/新泰斯/)。在JavaScript語言中，傳統上是以半形分號(;)作為代表結束與分隔其他的語句，但也可以不使用半形分號(;)，而改以斷行來區分。撰寫一支程式，就如同在寫一篇文章時，其中會包含了各種描述語句。

> Statements(語句)可視為在JavaScript的最小獨立執行程式碼組合

```
//註解也是一種語句
```

```
//指定值也是一種語句
const x = 10

//block statement(區塊語句)
{
  statement_1
  statement_2
  ...
  statement_n
}

//function statement(函式語句)
function name() {
   [statements]
}
```

在JavaScript語言中，Expression(表達式)主要用來產生"值"，因為它的功用很特殊，通常會獨立出來說明稱之為Expression Statements(表達式語句)。

一般的Statements(語句)主要功能是執行動作或定義某種行為，例如之前說過的註解(Comment)就是一種語句。

Statements(語句)還可以依不同情況的使用進行分類，以下列出:

- 控制流程(Control flow)
- 定義(Declarations)
- 函式與類別(Functions and classes)
- 迭代/迴圈(Iterations/依特瑞遜/)
- 其他(Others)

> 註: 此分類參考MDN的分類方式，ECMAScript標準分類並不是這樣。

> 註: Iterations/依特瑞遜/這個字詞，中文一般是翻譯為"迭代"這個字，這個中文字詞太過專業，對初學者來說是有看沒有懂。比較好的理解是它有"重覆作某件事"的意思，也就是和"迴圈"、"重覆"的意思相近。

控制流程

控制流程(flow control)語句是一種特殊的語句，它與程式的執行流程有關，會因為其中的判斷結果不同，導致不同的執行結果。

區塊(block)語句

區塊(block)語句可以組合多行語句，或用於分隔不同語句，區塊(block)語句經常使用於控制流程的語句之中。使用花括號(curly brackets)標記({})，將多行語句包含在其中:

```
{
  statement_1
  statement_2
  ...
  statement_n
}
```

if...else語句

`if/伊芙/...else/埃額斯/` 是常見的控制流程的語句，在中文的意思就是"如果...要不然"。所以整體的結構就像是中文的意思"如果aaa命題為真情況下，就xxx，要不然就ooo"。這個aaa就是一個判斷情況(condition)，判斷情況使用的都是比較運算符(Comparison operators)與邏輯運算符(Logical Operators)，而判斷情況結果則是會用布林的 `true` /觸/ 與 `false` /佛斯/ 值來判斷，但要特別注意之前說明的"falsy/佛西/"情況，這些在判斷情況中會自動轉成布林的 `false` 。

一個簡單的 `if...else` 語句的範例如下:

```
const x = 10

if (x > 100) {
    console.log('x > 100')
```

```
} else {
    console.log('x < 100')
}
```

> 註: `if...else` 並不是寫了 `if` 就一定要有 `else` ，也可以單獨只用 `if` 語句，但 `else` 不能單獨使用。

`if...else` 語句有幾個常見的延申結構，例如多個判斷情況時可以再其中加入 `else if` 。不過當你在寫這些判斷情況時，最好是讓它有真的會出現的情況，以免寫出根本不會有的判斷情況，成為程式碼的多餘部份:

```
const x = 10

if (x > 100) {
    console.log('x > 100')
} else if ( x < 50){
    console.log('x < 100 but x > 50')
} else {
    console.log('x < 50')
}
```

另一種是巢狀的語法結構，也就是進入某個判斷情況下，可以在裡面再寫出另一個 `if...else` 語句，例如:

```
const x = 10

if (x > 100) {
    if (x > 500){
        console.log('x > 500')
    }else{
        console.log('x > 100 but x < 500')
    }
} else if ( x < 50){
    console.log('x < 100 but x > 50')
} else {
    console.log('x < 50')
}
```

在判斷後的執行語句，如果是只有一行的情況下，就可以不需要使用區塊語句({})。也可以用空一格然後把判斷情況與結果語句寫在同一行，不過要注意這裡不要寫過長的語句，以免造成閱讀上的困難。例如:

```
const x = 10

//去除block statement
if (x > 100)
    console.log('x > 100')
else
    console.log('x < 50')

//寫成一行
if (x === 10) console.log('x is 10')
```

而在使用 `if` 加上 `else` 時，為了簡化語句，使用三元運算符(Ternary Operator)(?:)，來讓程式碼更簡潔，這也只能用於簡單的判斷情況與執行語句時:

```
const x = 10

(x > 100) ? console.log('x > 100') : console.log('x < 50')
```

> 注意: 建議只使用三元運算符在單一行的語句上，雖然它可以使用在多行與巢狀結構。

由於三元運算符的語法很簡單，它也常被配合用在指定值的語句中，例如:

```
const foo = value1 > value2 ? 'baz' : null
```

這個像我們之前說過的，用邏輯或運算符(||)來指定變數/常數預設值的語句。不過如果是單純的判斷某個值是否存在，然後設定它為預設值，用邏輯或運算符(||)是比較好的作法，例如:

```
//相當於 const foo = a ? a : b
const foo = a || b
```

> 心得提示: 因為人生還很長，而我們需要寫的程式碼還有很多，所以你會本書上看到有很多感覺上像是偷懶或密技的簡短語法，或許它並不是出現在標準中，也不是課堂上會教的"正規"寫法，但的確是廣泛被使用的一些語法。

比較運算符

比較運算符我們在前面的內容中已經有介紹一部份，還有一些相等比較的概念。以下將它大略分成2個分類來說明:

- 相等比較運算: 相等(Equality)(==)，完全一致(Identity)(===)。不相等(Inequality)(!=)，不完全一致(Non-identity)(!==)。
- 關係比較運算: 大於(>)，小於(<)，大於等於(>=)，小於等於(<=)

相等比較運算時，需要遵守標準中所定義的"抽象相等比較演算法"((==)與(!=)符號的比較)與"嚴格相等比較演算法"((===)與(!==)符號的比較)的規定。抽象相等比較演算法((==)與(!=)符號的比較)的規則如下(假設x是左邊的運算子，而y是右邊的運算子):

- Type(x)與Type(y)的類型相同時，則進行嚴格相等比較
- 數字與字串比較時，字串會自動轉成數字再進行比較
- 其中有布林值時， `true` 轉為1， `false` 轉為+0
- 當物件與數字或字串比較時，物件會先轉為預設值以及轉成原始資料類型的值，再作比較

至於嚴格相等比較演算法(也就是(===)與(!==)符號的比較)，由於一定要比較原始資料類型的類型，只要類型不同就一定是回傳為 `false` 。除了類型相同，值也要相等，才會有回傳 `true` 的情況，什麼值轉換成什麼值再比較就根本不需要。

> 註: 物件的情況會比較特別，我們會在說明物件類型的章節再詳細說明。

你如果之前已經有"falsy"與"truthy"的概念，就會理解到很多比較運算的最終結果，都會以這個為基礎轉變為布林的 `false` 或 `true` 值。

在 `if` 中的判斷情況表達式中可以直接運算出布林值，這就是依照"falsy"與"truthy"的概念。或是再搭配 邏輯反相Logical NOT符號(!)來回傳反轉的布林值，例如:

```
if (undefined) console.log('true') //false

if (null) console.log('true') //false

if (+0) console.log('true') //false

if (!'') console.log('!\'\' true') //true

if (123) console.log('123 true') //true

//下面的還沒教到，是空物件與空陣列
const a = {}
const b = []

if (a) console.log(a, 'true')
if (b) console.log(b, 'true')
```

邏輯運算符

邏輯運算符實際上在前面的課程都已經介紹過了，只有三個而已:

- 邏輯與Logical AND(&&)
- 邏輯或Logical OR(||)
- 邏輯反相Logical NOT(!)

邏輯與Logical AND(&&) 與 邏輯或Logical OR(||) 兩個符號可以組合多個不同的比較運算，然後以邏輯運算的"與"與"或"來作最後的布林值的運算。不過要注意的是，它們的運算回傳值，在JavaScript中並非布林值，而是**最終的值**，轉換為布林值是判斷情況中的作用。

> 註: 邏輯與Logical AND(&&)的結果只有同時為真時才能為真。也就是"真&&真 為 真"，其他都為"假"。 註: 邏輯或Logical OR(||)的結果只要其一為真，結果就為真。

這兩個運算符通常會搭配 群組運算符(())，也就是括號標記(()) ，這在數學運算上是用來作為優先運算的區分，或是用來區隔不同的比較運算。以下為範例:

```
const x = 50
const y = 60
const z = 100

if ((x > 10) && (x < 100)) console.log(true)
if (( ((x + y) > 10) || (x === 50) ) && (z == 100)) console.log(true)
```

> 註: (&)符號英文為And或Amphersand。(|)符號的英文在電腦上通常稱為Pipe，管道的意思。(!)在英文中稱為Exclamation mark，驚嘆號的意思。

風格指引

- (Airbnb 15.1) 優先使用(===)與(!==)而非(==)與(!=)
- (Airbnb 15.2) 像if的條件語句內會使用ToBoolean的抽象方法強制轉為布林類型，遵循以下規範：
  - 物件 轉換為 true
  - Undefined 轉換為 false
  - Null 轉換為 false
  - 布林 轉換為 該布林值
  - 數字 如果是 +0, -0, 或 NaN 則轉換為 false，其他的皆為 true
  - 字串 如果是空字串 '' 則轉換為 false，其他的皆為 true
- (Airbnb 15.3) 使用簡短寫法（註: 範例中這兩種判斷情況的簡短寫法很常見)

```
// 壞寫法
if (name !== '') {
  // ...stuff...
}

// 好的寫法
if (name) {
  // ...stuff...
}

// 壞寫法
if (collection.length > 0) {
  // ...stuff...
}

// 好的寫法
if (collection.length) {
  // ...stuff...
}
```

- (Airbnb 15.6) 不應該使用巢狀的三元運算語句，一般都只會用單行的表達式。
- (Airbnb 15.7) 避免不必要的三元運算語句。

```
// 壞寫法
var isYes = answer === 1 ? true : false

// 好寫法
var isYes = answer === 1

// 壞寫法
var isNo = answer === 1 ? false : true

// 好寫法
var isNo = answer !== 1

// 壞寫法
var foo = bar ? bar : 1

// 好寫法
var foo = bar || 1
```

- (Airbnb 16.1) 具有多行語句的區塊，需要使用花括號(braces)({})框起來

```
// 壞寫法
if (test)
  return false

// 好寫法
if (test) return false

// 好寫法
if (test) {
  return false
}

// 壞寫法
function foo() { return false }

// 好寫法
function bar() {
  return false
}
```

- (Airbnb 16.2) 如果你在多行區塊時使用if與else，將else放在if區塊的結尾花括號(})後的同一行。

```
// 壞寫法
if (test) {
  thing1()
  thing2()
}
else {
  thing3()
}

// 好寫法
if (test) {
  thing1()
  thing2()
} else {
  thing3()
}
```

switch語句

switch/斯餵取/語句有一種講法與用法，是相當於多個 `if...else` 的組合語句，也就是用於判斷情況有很多種不同的回傳值的組合情況，不過這個理解是有些問題。實際上，它最常被使用的是在完全一致相等值(===)比較的情況下，很少用在相關比較運算的情況下。基本上它的語法結構如下:

```
switch (expression) {
  case value1:
    //符合運算得到value1的執行語句
    break
  case value2:
    //符合運算得到value2的執行語句
    break
  ...
  case valueN:
    //符合運算得到valueN的執行語句
    break
  default:
    //符合運算得到其他值的執行語句
    break
}
```

舉一個簡單的範例來說，像下面這樣的 `if...else` 語句:

```
const x = 10

if (x > 100){
  console.log('x > 100')
```

```
} else if (x < 100 && x >50) {
  console.log('x < 100 && x >50')
} else {
  console.log('x < 50')
}
```

轉換為switch語句會變成像下面這樣。 `switch(true)` 代表當 `case` /K斯/中的比較運算需要為布林值的 `true` 時，才能滿足而執行其中包含的語句:

```
const x = 10

switch (true) {
    case (x >100):
        console.log('x > 100')
        break
    case (x < 100 && x >50):
        console.log('x < 100 && x >50')
        break
    default:
        console.log('x < 50')
        break
}
```

這相當於下面這個if...else的語句:

```
const x = 10

if ((x > 100) === true) {
    console.log('x > 100')
}else if ((x < 100 && x >50) === true) {
    console.log('x < 100 && x >50')
} else {
    console.log('x < 50')
}
```

而 `switch` 語句最常被使用的情況，是用於判斷相等值的情況下，在這時候 `case` 語句裡的比較結果的相等於完全一致(Identity)(===)的比較結果，在滿足的情況下才會執行包含在 `case` 其中的執行語句，例如:

```
const x = 10

switch (x) {
    case 100 :
        console.log('x is 100')
        break
    case 50:
        console.log('x is 50')
        break
    case 10:
        console.log('x is 10')
        break
    default:
        console.log(x)
        break
}
```

> 注意: 因為case語句中的值會以一致相等運算(===)來比較，所以資料類型也要完全相等才行，上面的範例如果把 `case 10:` 改為 `case '10':` ，就會輸出 `10` ，而不是 `x is 10`

`break` /博累克/關鍵字雖然在語法上是可選的，但 `case` 語句沒有搭配 `break` 會一直執行到出現 `break` 或是 `switch` 整個語句的結尾，這會出現不正確的結果，像下面這個範例:

```
const x = 50

switch (x) {
    case 100 :
        console.log('x is 100')
```

```
        break
    case 50:
        console.log('x is 50')
        //這邊少一個break
    case 10:
        console.log('x is 10')
        break
    default:
        console.log(x)
        break
}
```

最後的結果會是像下面這樣，通常這個結果並不是我們想要的:

```
x is 50
x is 10
```

> 註: 這算是一個有陷阱的設計，在eslint的no-fallthrough規則頁面上可以看到更多的說明

> 註: 所以不管如何，只要case語句中有包含其它需要執行的語句，一定需要以break作為結尾。但最後一個case或default語句可以不需要break。

判斷時有多個 `case` 情況而執行同一個語句時，會使用像下面這個範例的語法，這個語法結構也是很常見的用法，例如:

```
const fruit = '芒果'

switch (fruit)
{
   case '芭樂':
   case '香蕉':
       console.log(fruit, '是四季都出產的水果')
       break
   case '西瓜':
   case '荔枝':
   case '芒果':
   default:
       console.log(fruit, '是只有夏季出產的水果')
       break
}
```

風格指引

- (Airbnb 15.5)當case與default區塊中包含了字面宣告時（例如：let、const、function 及 class）時使用花括號({})來建立區塊語句。(註: 這是為了要區隔出每個case中的各自字面宣告區塊，以免造成重覆宣告的錯誤)

```
// 壞寫法
  switch (foo) {
    case 1:
      let x = 1
      break
    case 2:
      const y = 2
      break
    case 3:
      function f() {}
      break
    default:
      class C {}
  }

  // 好寫法
  switch (foo) {
    case 1: {
      let x = 1
      break
    }
    case 2: {
      const y = 2
```

```
      break
    }
    case 3: {
      function f() {}
      break
    }
    case 4:
      bar()
      break
    default: {
      class C {}
    }
  }
```

- (Airbnb 18.3) 在控制語句(if、while等等)的開頭花括號符號({)前面多空一格空白，函式的呼叫或定義則不需要。
- `defalt` 語句習慣固定是放在 `switch` 語句中的最後一段的位置，雖然它也不是一定要放在那裡。
- `switch` 語句的 `case/default` 語句，假使是位於最後一段(通常是 `default` 語句)，它的 `break` 在功能上並非必要，習慣加上只是為了讓程式碼更具一致性。

英文解說

Expression名詞，有"表情"、"表達"的意思，表情通常指的是人臉部的表情，這個字詞常常用在專業技術性的呈現或表達。至於像常會用的"表情符號"並不是這個字詞，英文字詞是使用emoticon或emoji。emoticon是emotion加上icon的新字詞，emotion則有"情感"、"情緒"的意思，近似於feeling(感受)的意思。Expression Emoticon這個組合字詞，通常是指"和人臉部表情有關的表情符號"，哭哭笑笑一類。不過表情符號的範圍比較廣，而且有很多是和動作、物品或標識有關。

Statement名詞，有"聲明"、"語句"、"陳述"等意思，它也可以當作財務或商業上的"報表"、"結算表"來使用，看起來這個字詞是用在正經八百的文件內容或訊息上。另外，它的字根是state，這個字的動詞有"宣佈"、"聲明"、"規定"的意思，名詞還有"州"或"國家"的意思，在電腦中最常拿來作"狀態"的名詞使用。

Literal名詞，有"文字"、"字面"的意思，也就是"按照字面上的意思就是這樣"。在電腦專業領域通常用這個字詞來作為"固定值"(fixed value)的記號，例如String literal稱為字串字面量，Numeric literal稱為數字字面量，其他還有陣列、物件、函式的字面量。而相對於Literal(字面)的就是 變數/常數 記號。

結語

本章一開始說明了兩個重要的名詞定義，表達式(Expression)與語句(Statement)。前面所學的變數/常數，以及資料類型的部份，將組合成為完整的程式碼語句的一部份。本章的重點是在於控制流程語句 `if...else` ，以及 `switch` 語句的應用，以及判斷情況的概念。

另外，在ES6後還有另一個可以用在控制流程的結構 - Promise(承諾)，由於它需要更多基礎的知識，我們將會在"特性"單元中，用獨立的一個章節來說明它。

家庭作業

作業一

公司交給你一個案子，要撰寫一支對網站上線上填寫表單進行檢查欄位的程式，以下是這個表單的欄位與要檢查的規則:

- 姓名(fullname): 最多4個字元，必填
- 手機號碼(mobile): 手機號碼必需是10個數字
- 出生年月日(birthday): 年1970-2000，月份1-12，日期1-31
- 住址(address): 最少8個字元，最多50個字元，必填
- Email(email): 最少10個字元，最多50個字元，必填

請問要如何寫出每個欄位的判斷檢查的程式碼。