

JavaScript

100%新手快速入門

JavaScript for Novice

ES6

Eddy Chang

前言

孔子說：「學而不思則罔，思而不學則怠」。如果你已經學了很多關於程式語言的知識，但卻無法正確地、有效率地在適合的地方使用它們，最後只會覺得學這些知識沒什麼用處，所以心中會感到困惑，不知道是要繼續學更多的東西，還是先停止不要再繼續學習。另一種情況則是相反的，你雖然會運用已經學到的知識，但沒有再繼續充實自己的知識，擴充你可以利用的工具等等，就會變得停滯不前，因為資訊科技不斷地進步，而你仍然只會使用舊的思維、方法來作這些事情。資訊軟體科技是一個日新月異的產業，我們每一天都要面臨許多新的資訊，或許有時候心中會感到焦慮，或是無所適從，這是很正常的情況。不論如何，如何找出最適合自己學習與進步的方式，是一個應對市場變化快速的重要課題。我會認為這世界上沒有一種教學，可以完全保証你一定學得會，只有比較接近正確的，或是較有效率的學習方式可以提供給你參考。我所能提供給各位初學者的，是一些建議與一些經驗，雖不敢說是百分之百的正確，但這些都是我從一開始初學者跌跌撞撞一路走來的經驗，希望你可以從中得到一些啟發，或是找到更適合你自己的一條學習之路。

你可能不知道怎麼開發 JavaScript 語言，卻每天都在使用它，但每一天當你上網瀏覽 Facebook，用 Google 地圖查詢，用 Gmail 收發信件等等，這些程式背後的功能都是用 JavaScript 所寫成，它與你我的生活中各式各樣的網路應用緊密而相關。JavaScript 與網站瀏覽器的關係很密切，它仍然是目前唯一一個能夠在各種品牌的、各版本的網站瀏覽器，執行各種應用的程式語言。

JavaScript 語言，同時是一個進步很快的程式語言，它的功能每一年都在增加，執行速度和效率每一年也在提升。它同時也是一個很普遍被使用，也很容易學習的程式語言。JavaScript 的技術生態團相當龐大，網路上的資源也很豐富，各種應用領域的發展都可以找到對應的新技術，以程式語言界來說，JavaScript 的發展是相當特別的，它誕生於網路發展的萌芽時代，也因為網路的快速發展，變得無所不在與幾乎無所不能。

JavaScript 語言不是一個完美的程式語言，相反的它充滿了缺點，也有很多問題。它雖然是一個容易學習的程式語言，但同時也很難精通其核心與運作原理，它的本質設計與現今流行的其它程式語言，有非常多不同之處，不過也一直從其它程式語言中學習各種新功能。對於較為資深的開發者來說，我們仍然需要不斷地的學習與精進，新的語法與開發思維逐漸在普及，舊的東西也逐漸要被淘汰。

本書大部份是取材自我之前寫的一本免費的電子書籍 - 從 ES6 開始的 JavaScript 學習生活，這本書可以很容易在網路上找到，它的內容相當豐富，也詳盡地說明了 ES6 的重要特性，以及 JavaScript 語言的重要內容。因為本書的篇幅有限，所以在閱讀過本書後，你可以再看這本電子書裡面相關的章節，作為一個補充與進階的學習。

在本書中希望用較為白話的方式，介紹該如何來學習這個 JavaScript 程式語言，每一個建議與意見，背後都有它的意涵，也有很多是實際中得到的寶貴經驗。這本書的適合的閱讀對象，是完全對 JavaScript 語言沒有任何開發經驗，或甚至完全沒有程式開發經驗的學習者。

~ Eddy Chang (eddy@joomla.com.tw) 2018.9(第 1 版)

第一章：JavaScript 的 5W1H

JavaScript 程式語言有其誕生的歷史背景，本章中藉由 5W1H 的問答方式，帶各位初學者先了解這個程式語言的各種面向，你可以先理解這個程式語言的例如說應用的層面是什麼，為什麼要學習這個語言，或是它目前的發展情況等等。

JavaScript 是什麼(What)

JavaScript 是一個程式語言的名稱，雖然名稱上有"Java"的字詞，但實際上它與另一個知名的 Java 程式語言相差甚遠，除了語言的語法、結構上都不同，在針對的應用領域也是不相同的。Javascript 語言本身是一個已經很有歷史的程式語言，在 1995 年就已經實作發佈，迄今已超過 20 年。

JavaScript 算是一個通用的語言名稱，它的正式的語言標準稱為 ECMAScript，但目前通行的 JavaScript 語言內建的 API(應用程式介面) 不光只有 ECMAScript 標準的內容，而是包含了以下三個主要來源：

- ECMAScript 標準
- HTML/CSS 標準(W3C 組織制定)
- 各大瀏覽器品牌廠商自訂

ECMAScript 的標準制定成員，有來自開發界知名的學者和工程師，以及各家瀏覽器大廠的代表。ECMAScript 的各版本的發行日期如下，目前是每年就會討論出一些新的特性然後加入，最近較為重大的改版應該算是 ES6 這個版本。不過，標準定案發行並不代表在各個瀏覽器品牌中，即可使用其中的功能規格，還需等待各瀏覽器品牌廠商進行實作。但有時瀏覽器廠商會搶先一步，實作出某些尚在制定中的語言功能，所以有些太新的功能，會在不同的瀏覽器品牌中有相容性的問題。

- ECMAScript 9 (ES9) 發行於 2018 年中
- ECMAScript 8 (ES8) 發行於 2017 年中
- ECMAScript 7 (ES7) 發行於 2016 年中
- ECMAScript 6 (ES6) 發行於 2015 年中
- ECMAScript 5 (ES5) 發行於 2009 年底
- ECMAScript 4 (ES4) 舍棄
- ECMAScript 3 (ES3) 發行於 1999 年底

JavaScript 五個重要的特色

直譯式程式語言(腳本程式語言)

JavaScript 我們通常會把它稱為"直譯式程式語言"(Interpreted language)，它的名字中也有"Script"(腳本)這個字詞，代表它是一個"腳本程式語言"。

那麼，什麼是"直譯式程式語言"？

在傳統的程式語言的開發過程，例如 C 或 C++ 程式語言，程式設計師寫好了程式碼後，要在作業系統平台上執行，需要經過一些過程，把程式碼編譯(Compile)成為機器碼，這樣才能夠執行，所以通常整個的開發過程是「撰寫 - 編譯 - 連結 - 執行」(edit-compile-link-run) 這樣的流程，當然這種方式不是固定的，現今有很多新式的程式語言，例如 Java 或 C# 程式語言採用混合方式，編譯時是先產生位元組碼，執行時再進行直譯，這種作法有一個很大目的，就是可以更方便移植到不同作業平台上執行。

直譯式程式語言，就是上面所說的流程更佳簡化的一種方式。當程式設計師寫好程式碼後，就可以直接到平台上執行，不需要經過編譯、連結的過程，在作業系統平台中會有執行引擎，動態的將程式碼進行直譯執行。這種執行方式大幅度縮短了傳統的應用程式的開發到執行的過程，只剩下「撰寫 - 執行」(edit-run) 流程。程式設計師只需要專心撰寫程式碼，至於要如何編譯成機器碼與執行，就交給直譯器來處理。直譯式程式語言在現今，因為電腦的運算能力和資源都很充足的情況下，執行效率也愈來愈好，目前除了 JavaScript 外，也有許多熱門的、常見的直譯式程式語言，例如 PHP、Python、R 語言等等。

那麼，"腳本程式語言"又是什麼？

腳本程式語言與直譯式程式語言是相關的，腳本程式語言通常是一種設計得較為易學、易用的程式語言，而且都是直譯的執行方式。在撰寫程式碼時，會像是在對電腦下達指令一樣，最早有所謂的"指令碼程式語言"與後來的"巨集語言"，它們都有類似的特性，一開始都是為了方便進行電腦自動化工作所發展出來的。

由於腳本程式語言的指令(或 API)名稱，通常都會用較接近口語的英文字詞來命名，方便開發者使用與容易閱讀，所以又稱這種程式語言為"高階程式語言"。不過，"高階"一詞只是對應機器碼的"低階"(更接近硬體)而言的相對概念，並不是比較高級的或進階(困難)的意思，千萬不要誤解它的意思了。

JavaScript 語言與一般的巨集或指令碼語言相比，巨集或指令碼語言的內建指令或 API 通常不多，而且可用的語法也有限制，但 JavaScript 的內建 API 相對非常多，語法方面也多樣化，這一點相當不同。主要是 JavaScript 已經發展得很長一段時間(20 年以上)，除了它的內建 API 的來源有好幾個之外，而且 JavaScript 也經常加入新的功能。除此之外，為了要和舊的瀏覽器版本相容，執行引擎都會保留舊的 API 來達到向下相容，才會造成有些新舊的同功能 API 會混在一起的情況。

另一個很常聽到的專業講法是"動態類型程式語言"，或是"弱資料類型程式語言"，這也是很常在腳本語言或直譯式語言中的設計，目的也是讓開發者更易學、易用，在本書之後資料類型的章節中可以更明白的理解這個設計。

JavaScript 也是物件導向程式語言

物件導向程式語言算是現今相當熱門的一種，常見的程式語言，例如 Java、Python、C#、Swift 等等，都具有物件導向語言設計的特性。JavaScript 語言也是屬於物件導向程式語言，但它的內部設計與上述的其它語言都不相同。JavaScript 是使用原型(prototype)為基礎的物件導向設計，而其它常見的語言都是以類別(class)為基礎的物件導向設計。這兩種的用法和設計思考方式不一樣，所以如果你已經有學過其它程式語言的基礎，反而會覺得學習上無法互相參照對應。

慶幸的是，在 ES6(ECMAScript 2015)新標準中，新增了"類別(class)"的章節。新一代的 JavaScript 中已經可以使用類別的物件導向設計方式，來撰寫程式碼。ES6 在類別中的可用語法目前還相當簡單，不過已經有許多函式庫開始採用這種語法。

不過，因為長期以來 JavaScript 的開發設計方式的發展，並不會以物件導向設計開發方式為主流。一般來說在早期，腳本程式語言不建議用物件導向設計方式來撰寫應用，主要的原因是執行效率並不好，也沒有那麼多類別基礎的物件導向的特性可用，因此造成物件導向設計方式在 JavaScript 中並不流行。在技術圈有再發展出其它更合適的開發設計方式，例如熱門的函數式程式開發(functional programming)，或者是其它的開發設計方式。

JavaScript 的執行環境主要是瀏覽器

Javascript 程式語言通常直接透過瀏覽器的直譯器執行，它並沒有經過編譯為機器碼的過程，所以它的執行環境與資源還更受到限制，並沒有辦法像一般在作業系統中的應用程式，能存取到這麼多的資源與功能。而它的

程式原始碼，在瀏覽器中就可以被看得見，如果要加快執行速度以及作簡單的保護措施，常見的作法是最小化(minify)與醜化(uglify)原始碼。

Javascript 現今已經可以作為伺服器中來執行程式碼，主要是要透過 Node.js 伺服器的執行環境，Node.js 採用的是 Google 所開發的 V8 執行引擎，這是一套開放原始碼、高效率的 JavaScript 直譯器，同樣也用於知名的 Google Chrome 瀏覽器之中。

JavaScript 程式碼也可以直接在手機 App 中執行，這是另一個熱門的新技術，一開始是透過手機中在開發時的內嵌可瀏覽網頁的一種視圖使用者操作介面(通常稱為 WebViews)，然後把 JavaScript、HTML、CSS 所製作好的網頁包裝到 App 之中，包裝的工具通常是知名的 Phonegap 或 Cordova。新一代的技術則是採用內嵌 JavaScript 程式碼到 App 之中，使用的使用者操作介面是手機開發時原本的操作介面(通常稱為原生的操作介面)，可以這樣作的原因是蘋果(Apple)公司開放了 JavaScriptCore(JSC) 這個框架，可以在 App 中直接運行 JavaScript 程式碼，而 Android 手機的部份也是由此在延伸。

JavaScript 的執行是單執行緒、非阻塞的，以及有異步執行的程序

單執行緒的執行就像是在操場上只有一條跑道的設計，如果在跑道上有多個跑步的人，有個人停了下來，就會造成後面的人被迫停下腳步，也就是被"阻塞(block)"了，所以在開發 JavaScript 的應用程式時，要注意不能阻塞這條執行專用的跑道。

不過，你可能會有個疑問，現在電腦不是都是多 CPU 多個執行緒，JavaScript 難道不能用多執行緒的設計或執行方式嗎？

這是因為 JavaScript 原本的一開始的設計，只是在個人電腦中瀏覽器執行使用的程式語言，它的執行環境比起一般的個人電腦中的應用程式更受到限制，也不需要什麼多工同步，所以就被這樣設計一直用到現在。當然，現在也有一些新技術發展，例如多執行緒，或是輔助執行緒等等的設計。

為了達成多工同步與非阻塞，JavaScript 採用了一種特別的開發方式，就是把這些會造成阻塞的執行程序，例如與伺服器相連接、計時器、事件等等，利用異步執行的程序去執行。簡單的比喻來說，就是現在跑道上有個人想要喝水，先叫他站到旁邊等候區去等人送水過來，讓跑道上的其它跑者先繼續跑完，等他到喝完水，而且跑道都清空了，這個原本要喝水的人再繼續回到跑道上來跑完。

所以說 JavaScript 雖然表面上自己是單執行緒的執行方式，但實際上至少還有一個協助的執行緒或低層的執行輔助，在瀏覽器中當然就是透過瀏覽器程式來協助執行。

JavaScript 具有高階函式的特性

有許多程式語言的專有名詞，例如高階函式(higher-order function)、頭等函式(first-class function)等等，這些都算是 JavaScript 語言中有的特性之一，不過太專門的術語就不在這裡深入解釋，讀者如果有興趣可以到維基百科查看。

這些特性用最簡單的一句話來說，就是像下面這個講法：

在 JavaScript 中的"函式"這東西，可以當作另一個函式的傳入參數輸入，也可以作為另一個函式的回傳值輸出

所以說 JavaScript 把"函式"看作很重要的一種物件，在它裡面函式基本上是一種物件延伸的設計，所以函式的地位很高，是頭等公民。

JavaScript 是什麼時候被創造的(When)

JavaScript 語言的誕生與網頁瀏覽器關係密切，一開始是在 1995 年，當時的瀏覽器大廠網景(Netscape)的工程師布蘭登·艾克所創造出來的，據說當時只花了十天就寫出語言的原型。

後來微軟在自家的 Internet Explorer 瀏覽器中也制作一套仿造 JavaScript 的 JScript 語言，網景(Netscape)在 1996 年向 ECMA (歐洲電腦製造商協會) 提交語言標準，讓 JavaScript 成為公開的標準。現今的 ECMAScript 幾乎是為了 JavaScript 量身打造的一套標準。

JavaScript 有誰在使用(Who)

隨著網路的速度與普及，使用電腦或手機上的瀏覽器來觀看網站，已經是每個人的日常生活的一部份，在大大小小的網站中，我們每天都會使用的 Gmail、Google 地圖、Facebook、Instagram 網站中，其中都有 JavaScript 程式的運行之處，甚至在手機的 App 裡也有。由網路發展起來的網路應用龍頭 Google，所有的網路上的服務，背後都是 JavaScript 程式，而 Google 所開發的 V8 引擎，讓 JavaScript 程式的運行速度高速的提升，這是一個非常重要的發展關鍵。

根據國外知名的技術討論區 StackOverflow 在 2017 的調查結果(總計 64,000 名受訪者)，有高達 6 成的工程師有在使用 JavaScript 程式語言，而知名的專案原始碼儲存管理平台 Github 在 2017 的調查結果，在所有專案中使用數量最高的也是 JavaScript 程式語言，相比於第二名的 Python 語言超出兩倍。在許多的求職平台調查報告中，JavaScript 程式語言的專業人才需求數量，通常也是位於第一名或第二名。

JavaScript 程式幾乎無所不在，也快接近無所不能，你雖然可能不知道怎麼開發它，卻每天都在使用它，這個程式語言是一個從誕生到現在，充滿缺點但又不得不使用的一個程式語言，它並不是一個完美的程式語言，甚至較為準確地來說，它是一個問題相當多的程式語言，但因為它是目前唯一一個可以在各瀏覽器品牌上運行的程式語言，所以不論有多少大企業想要用別的程式語言來取代它，最終仍然動搖不了 JavaScript 程式語言的地位。

為什麼要學 JavaScript(Why)

JavaScript 是一個很熱門的程式語言，有很多大廠支持

隨著網路服務的興起，JavaScript 已經變為一個很重要的、相當流行的程式語言，它也是唯一一個可以在幾乎所有瀏覽器品牌中運行的程式語言，更重要的是它是公開的標準，而不是由單一私人公司所制定。有許多知名的網路服務為主的大企業，都極力支持 JavaScript 語言的發展，例如 Google、Apple、Facebook 等等，能發展得如此快速，而且應用延伸到不同領域(手機 app、桌面應用等等)的程式語言，目前也只有 JavaScript 能作得到而已。

JavaScript 是一個易學、易用的程式語言

其次，JavaScript 相當地容易學、容易用。正確來說，它是一個"易學難通"的程式語言，初學者要學習它，一開始很簡單就可以學會開始使用，但要理解其中的一些特別的特性與設計，就得花時間多多研究才行。有許多教授小學生的程式設計課程，一開始也是採用 JavaScript 作為教學的程式語言。另外，JavaScript 不論在網路上的、實體的學習資源，數量都是相當多的。

最知名的例子之一，就屬美國前總統歐巴馬為了推行資訊教育，他也學習和寫了一段 JavaScript 程式碼

投資在 JavaScript 的學習，對於求職與工作很有幫助

JavaScript 對於求職找工作相當有幫助，因為應用的層面很廣泛多元，相對的在人才的需求市場上，數量也是最多的。如果你能擁有 JavaScript 開發的專業知識和經驗，相信對於找工作會相當有幫助，所以投資在

JavaScript 上的學習，不論是在工作上，或是想要開發自己的應用，都是相當值得的。

JavaScript 用在什麼地方(Where)

網站瀏覽器

瀏覽器是 JavaScript 一開始被設計要運行的環境，所以 JavaScript 程式普遍被使用在各個網站之中，不論是網站中的常見的動態廣告看板、表單檢查、下拉選單等等，有可能背後都有一些 JavaScript 程式在運行。至於像大型網站 Facebook 或 Twitter 等等，更多的動態效果都需要靠 JavaScript 程式運作得起來，一些專屬的網站應用，像是 Gmail 或 Google 地圖，可說是完全要依賴 JavaScript 程式。

伺服器

Node.js 的核心是使用高速的 V8 JavaScript 引擎，可以運行於一般常見的各種作業系統平台之上。Node.js 有效的利用 JavaScript 語言中非阻塞 I/O 的優勢，在相同的伺服器資源下，可以提供承載高並行的運算處理。從 2009 年發展至今，已經逐漸成熟，近年來也發展在多核或多執行緒執行環境下的解決方案。目前已有許多大型的網站採用這個執行環境，例如 IBM、Microsoft、Yahoo!、Walmart、SAP、LinkedIn、PayPal 等等公司。

行動裝置

要談到 JavaScript 語言用於行動裝置開發的發展，就不得不先說目前的兩大生態圈，一是以 Facebook 為首的 ReactJS 陣營，ReactJS 只是一套視圖用的函式庫，但加入相當創新的想法，使用類似的語法可以使用 React Native 專案開發手機 App，是可以跨 iOS 與 Android 平台，React Native 開發出來的 App 是 Native(原生)的，它目前是新式的一種開發手機 App 方式。

另一個是以 Google 為主的 AngularJS 陣營，它是一個完整的應用程式框架，另有一個專案是 ionic，也是用來開發跨平台手機 App，它還基於另一個專案 Apache Cordova，這個專案原本就是使用 HTML、CSS 與 JS 來開發跨平台手機 App 的工具專案，這種開發出來的 App 稱為 Hybrid(混合) Apps。

使用 Hybrid(混合) Apps 的開發工具或專案還有很多，例如 jQuery Mobile、Framework 7、Sencha Touch 等等，各自有各自使用的框架或函式庫。因為 React Native 的出現，有許多函式庫開始往原生的路線發展，目前原生的 App 大多使用 JavaScriptCore 作為執行的核心技術。

桌面應用程式

Electron 是由 Github 公司主導的，桌面應用程式開發框架的開放原始碼專案，基於 Chromium 與 Node.js。它原本是用來要開發 Atom 程式碼編輯工具所使用的專案，原名稱為 Atom Shell，後來成為獨立的專案，歷經兩年左右的開發，現在已有很多廠商使用它來開發桌面軟體。

Electron 除了也是使用 JavaScript 語言與 HTML 作為基礎程式語言，其優點主要是開發出來的桌面應用可以跨作業系統，能在 macOS、Windows、Linux 等作業系統上運行，開發的門檻會低很多。不過，缺點是應用程式的檔案大小會比原生的應用大很多，而且執行效能比原生的應用程式差，對電腦資源如 CPU 與記憶體的需求會較高，目前這些缺點仍然是需要改進的重點。

其它各種專屬應用平台

當然，JavaScript 語言可以作為許多硬體平台的腳本語言，例如整合 Arduino、樹莓派(Raspberry Pi)硬體控制板的應用。在目前最新的 VR/AR 的應用中，也有見到 JavaScript 語言的身影。

JavaScript 如何學習(How)

本書的主旨就是在於如何學習的部份，我先列出一些基本的、簡單的建議方向在下面。

先學習基本的語言特性

這是一開始你可以著手的第一步，許多免費的學習資源在網路上都可以找得到，你可以試著先閱讀幾個章節，然後動手練習看看。不過，雖然在網路上有很多的 Javascript 入門教學與文章，但有很多中文的教學，內容已經很過時了，這要特別注意。我最推薦的是以下：

- Mozilla MDN (中文) 教學與範例中的權威，內容針對新式的標準語法都有說明到。
- w3schools JavaScript Tutorial (英文) 雖然是英文，但用字都很淺顯易懂，程式範例多。

學習任何一種程式語言，除了要了解這個程式語言的應用領域，與一些基本背景外，最一開始一定是要理解這個程式語言的基本特性，學習程式語言有點像是在學一種語言一樣，都是從點(單字=關鍵字)到線(基本句型=表達式)到面(整個句子=函式、程式碼片段)的過程，例如以下幾個項目，這幾乎在學習所有的程式語言都是通用的：

- 資料類型
- 運算
- 流程控制
- 迴圈
- 函式&作用域
- 物件、陣列、其它集合的資料類型
- 錯誤(例外)處理

上面這些只是一些基礎，學習這些東西有可能是個滿無聊的過程，就像你一開始學英語(或其它外語文)時，都要歷經相當無聊的背單字的過程，英語畢竟並不是我們的從小使用的母語，要自然地就學會是不太可能的，所以包裝得再精美的學習教材，仍然回到學習的本質，你還是得要老老實實的蹲馬步，打好基礎才能繼續學習之路。

當然每種程式語言都有它特殊的設計與特性，我把一些重要的、必學的 JavaScript 特性寫在下面，如果你沒把這些特性學好，最後只會落得一知半解的程度，無法發揮這個語言真正的特性：

- callback(回調)、CPS 風格
- closure(閉包)
- 事件迴圈、異步程序
- this
- 事件處理

以上這五個特性都是 JavaScript 中相當重要的特性，而且注意它們與"函式"都有密切的關係。當然如果你有多的時間的話，可以再深入理解 JavaScript 的原型繼承內容，這可以會讓你對 JavaScript 語言的本質設計更加理解。

這些特性中有部份會涉及到程式語言較底層的設計，或許學習起來會覺得有些部份看得不是很了解，這是正常的情況，我們要掌握住這些特性會在何時使用到，或是在什麼地方會這樣使用，也就是說我們要清楚地理解下面這兩大重點：

- 這個地方就是要這樣寫才可以
- 如果這樣寫，就會產生這樣的結果

至於"為什麼"會造成這樣的結果，或是"為什麼"要造成這樣的結果，這會涉及到一些太底層的實作，或語言的結構設計，這些一開始先不需要了解得那麼清楚的。

學習的進度要按照自己的程度來安排，不需要一下子學得太快或太多，死記是沒有用的，撰寫程式碼並不是考英文筆試，多練習思考反而更重要，要充份的理解每一段範例程式碼和教學的內容。坊間雖然有很多所謂的知名的書籍，但它們有可能是給進階的開發者參考或閱讀用的，並不一定適合初學者，你可以選擇適合自己目前程度的教材，不一定要跟著其它人的講法硬著頭皮去讀這些，並不適合自己的教材。

一開始就要養成程式碼撰寫的好習慣(風格指引)

第二點要提出來的，是撰寫程式碼的"習慣"問題，一開始養成良好的習慣在 JavaScript 語言中非常重要，當然在其它的程式語言的學習也是同樣的。

JavaScript 語言有個很特別的情況，因為它有些歷史了，再加上新的瀏覽器還是可以向下兼容舊的程式，很早以前的程式碼現在還是在網路上可以找得到。這會變成好像這樣寫也可以，那樣寫也可以，反正都可以執行出同樣的結果，有些開發者就會開始亂寫養成壞習慣，這也是因為實際上沒有一個真正標準的寫法，程式本來就是可以達成要執行的目的就可以，但是，壞的習慣容易造成潛在的問題，因為語言本身設計的問題，它裡面有一些隱藏的副作用語法，也就是你用了這些語法，你其實不是很了解它裡面會是怎麼執行的，這些潛在的、有可能的問題，因為壞習慣的寫法，在某個時間點就有可能會造成不好的影響或結果。

舉一個很簡單的例子來說，對於變數或函式的命名應該怎麼取名字會比較好，很早以前為了執行的效率，都會建議取得愈短愈好，有些程式設計師(尤其是非英語為母語的)經常會取一些 a、b、x、y 這種不知道作什麼用的名字，這算一個壞習慣，如果它只是個暫時用的變數還好，如果有用途的變數或函式，這種名稱除了寫的人自己看得懂之外，別的開發者根本看不懂，搞不好原本撰寫的開發者，過一段時間回頭來看，自己也搞不清楚是作什麼用的。

那麼，應該怎麼取名字才是比較理想的？

下面這個是來自 react-redux 專案的一段原始碼：

```
export function whenMapDispatchToPropsIsMissing(mapStateToProps) {
  return !mapStateToProps
    ? wrapMapToPropsConstant(dispatch => ({ dispatch }))
    : undefined
}
```

我們先不管它是作什麼用的，你有看到它的函式的取名"whenMapDispatchToPropsIsMissing"這麼長，意思是"When Map Dispatch To Props Is Missing"，中文翻譯是"當映射 Dispatch 到 Props 沒對應到時"，這實在相當白話的一個取名。這種取名方式的好處是大概連註解都不需用寫了，看函式的名稱就知道它要作什麼用了，這種寫法有個專用名詞叫"程式碼自己帶文件說明(self documenting)"，這種撰寫的方式可以提高程式碼的可閱讀性。當然你可能會有疑問，這麼長的命名，到最後不是會增加整個程式碼執行時的文字數量，對於執行可能會有一點點影響？當然因為現在開發 JavaScript 程式，最後還有個"編譯"的過程(實際上它並不太像是傳統程式語言的編譯，實際上的工作是"轉化為 ES5 標準、壓縮、醜化、分析整理等等")，這過程會把程式碼再加工處理，所以這些變數或函式名稱，最後在執行時一樣是很簡單的名稱，所以你不用太關心執行的效率問題。

為什麼要注重提高程式碼的可閱讀性？

因為程式碼經常是要寫給別人看的，而不光只是開發者自己而已。一個較為大型的應用程式，通常會有好幾個開發者共同開發，或是一個應用程式完成功能後，後續還有未來維護的問題，例如之後的要升級功能，或是修

改功能等等，這可能都不是原來撰寫這個應用程式的開發者。為了這些與其它開發者容易溝通，讓其它開發者可以更容易的、正確的理解程式的原始碼中的作用，提高程式碼的可閱讀性是一個非常重要的，最好一開始就能建立的一些好習慣。不過，這一點在台灣的程式語言教育中，也是常常被忽視的一環。

提升程式碼的可閱讀性的作法很多，在書本或課堂上一定會學到的是在程式原始碼中的"註解"(comment)這部份，大部份的程式語言的註解方式都差不多，有單行註解與多行註解兩種，像下面這樣的範例程式碼：

```
// 我是單行註解  
  
/*  
 *我是多行的註解  
 */
```

那麼另一個新的問題是，註解裡面要寫什麼呢？或是註解是該在什麼時候寫呢？

註解裡面要寫什麼，有著很嚴謹的像教科書的方式，但也有很隨興的寫法，這並沒有一個統一的標準。有一種簡便的方式，是可以從註解中自動產生 API 文件，不過這要要求註解中，需要撰寫固定的一些標記才可以，在一些大型的專案中是會被要求要這樣作的，至於是用哪一些標記則要視專案在規劃時，看選擇了什麼工具來決定。

對於初學者來說，寫註解這個習慣是一定要一開始就養成的，你可以先加一些輔助的說明就可以，一般我們會在註解中加上對函式或變數，或是這一行語句是在作什麼用的之類的，例如像下面這樣的範例程式碼：

```
// 處理文字框的輸入  
handleChange = event => {  
  // 先得到輸入值  
  const inputValue = event.target.value  
  
  // 如果輸入字串長度大於1以上，更動原本的state值  
  if (inputValue.length > 1) this.setState({ inputValue })  
}
```

註解的撰寫沒有一定的標準，總之能讓人看得懂就好。不過我這裡要建議每個初學者，進一步學習一個很重要的好習慣，就是：

用英語來寫註解，而不要用中文

為什麼要用英語而不要用中文呢？因為我們在撰寫程式碼功能時，其實都是使用英語來寫，所有的符號都是半形的符號，但因為我們對中文語言較為熟悉，所以一般都會用中文來寫註解說明。但是不管你用什麼中文輸入法，在任何的作業系統中，中文和英語的輸入都是需要切換的。那麼仔細想想註解這東西，註解是什麼時候才會被寫上的？通常是在寫某一小段的程式碼就該寫上去了，而不是整個程式碼都寫得差不多了，才回過頭到最前面開始一段一段寫註解，邊寫程式碼邊寫註解的情況，也是很經常的事情，有時候註解先寫在某個地方，作為一個提醒這部份功能似乎還有點問題，或是還沒有全部完成，後面再回頭來繼續撰寫的情況，也是很經常使用的情況。

如果在使用中文註解的時候，你要一邊寫了一小段程式碼，然後接著要寫一些註解文字，然後接著又要寫程式碼...寫註解...這樣的重覆動作中，你有發覺有一件動作也不斷重覆嗎？就是你會一直切換中英的輸入法，認真的說來，這是一種干擾撰寫程式碼的動作，有時候在註解裡還要打上英語字詞，例如變數或函數的名稱，切換

輸入法的頻率又更高了，這樣來來回回雖然表面上看起來只會花一點點時間，但長期累積起來也是很驚人的。不過我個人覺得這種在中英文間切換輸入的行為，另外還會干擾到你在撰寫程式碼的思考。

所以，用英語來寫註解，是我提出來的一個針對非英語母語程式開發者的習慣，而且我建議一開始就要慢慢學習養成。不管你的英語有多爛，在學校時拿過幾次不及格，你還是得這樣作。當然一開始用這方式會覺得不能適應，但過一陣子你就能體會這樣作的好處，經常看看別人寫的程式碼範例，學習別的國外資深程式開發者是如何寫註解的，然後就會習慣這樣用了。上面的範例是方便教學這樣寫的，實際上如果直接用英語來寫註解，大概會寫成下面這樣：

```
// handle text field input
handleChange = event => {
  // first get input string value
  const inputValue = event.target.value

  // if input string length > 1, set it to state
  if (inputValue.length > 1) this.setState({ inputValue })
}
```

你也不用太注重英語的語法正確與否，因為我們今天不是要去參加英語寫作比賽，只是要寫出這一段程式碼語句在作什麼用而已。至於要不要這樣作就由你決定，這與許多程式碼風格樣式中的一些建議一樣，都是因為長期撰寫程式碼後，累積下來的寶貴(慘痛)經驗，希望後面新加入的開發者不要再犯這樣的問題，所以才會有這些建議。

JavaScript 程式碼撰寫的風格指引，以下幾個是最常被提及的，而且大部份都有中文翻譯，在網路上都可以找得到：

- Airbnb JavaScript Style Guide 中文翻譯 這篇主要針對 ES6 語法
- Principles of Writing Consistent, Idiomatic JavaScript 中文翻譯 這篇主要是針對 ES5 語法，有少部份對新式語法來說不一定準確。
- Google JavaScript Style Guide 中文翻譯 這篇主要是針對 ES5 語法，有少部份對新式語法來說不一定準確。
- Code Conventions for the JavaScript Programming Language 簡體中文翻譯 大師的說明的一些規則，算是通則。

學習看懂英語文件

因為我們並非是以英語為母語的程式開發者，但程式碼撰寫時用的是英語，另外最新的、最完整的學習文件或資訊，也都是以英語寫成。因此，如果你決心要學好程式語言，不論是哪一種語言，在看懂文件的這方面的英語能力絕對是必備的。不要盼望會有好心的台灣或中國網友，都會正確的、立即的幫你翻譯好這些重要的英語文件，通常在網路上的中文翻譯文件，經常是翻譯得七零八落，要不然就是翻到後面連翻譯的人都不知道在翻什麼內容了，要不然就是已經過時的，早就改版或不一樣的一些文章，與其浪費時間在尋找或學習這些可能有問題的中文翻譯文件，還不如直接花點時間直接看原版或官方的文件，它們的正確性和時效性是百分之百的。

那麼，要如何看懂英語的文件呢？這才是我們要學習的部份。

首先準備一本中英字典是重要的，我指的並不是那種厚重的實體字典，現在網路上有很多免費的電子字典，手機上也有很多這類的 App，你可以上網找找，這些字典裡面的內容很豐富，也可以幫你從很多不同的來源查詢，甚至是維基百科找你要查詢的字詞。

所有的程式開發用的英語字句敘述並不會太難，大概只需要高中程度的英語水準就可以看得懂，難的部份會落在"專用名詞或術語"的地方，有些單字詞並不是在國高中學的那個意思，這部份長久養成習慣後，經常看到就會很容易理解，例如"class"這個字詞，在學校學的是"班級或課程"的意思，但是在程式語言中指的是"類別"的意思，由此可以了解英語中的單字詞，常常都是一個字詞有好幾種意思，要視使用的情況來決定。

另一個很常見的是程式語言中，會縮寫或簡寫的字詞，尤其是原來的字詞太長，或是這個專用術語是多個單字詞組成的，例如像"int"是"integer(整數)"的簡寫字詞，或是"char"是"character(字元)"的簡寫字語等等。多個單字詞組成的，常見的像是 API、SDK 等等，這種多字詞組成的縮寫，在中譯文件中有時候會被翻譯得不倫不類，翻譯出來雖然是這幾個中文字湊起來，但是在講什麼東西還是經常看得一頭霧水，太過專業的術語反而會這樣，此外很多也沒有統一的對照字詞，加上簡體中文與繁體中文的翻譯，經常也是不相同。

說完了字詞的學習部份，再來是關於句子的部份，下面以一個簡單的句子來說明。

以下的字句是摘自 Mozilla 中關於 [JavaScript 基礎教學](#)的一小段說明：

Variables are containers that you can store values in. You start by declaring a variable with the var keyword, followed by any name you want to call it:

我們的目的只是要看懂這個句子在說什麼而已。首先，你可以把看不太懂的單字詞，用字典先查過一遍後，把中文的意思標註在那個單字詞後面，變成下面這樣，這裡只是一個示範，實際上視你自己的英語單詞的水準而定：

Variables(變數) are containers(包含) that you can store(儲存) values(值) in. You start by declaring(宣告、聲明) a variable(變數) with the var keyword(關鍵字), followed by(接著是) any name you want to call(呼叫) it:

然後仔細再看過一遍，能夠看懂這句話的意思就可以，不需要硬是要整句都翻成中文才行。英語的句子要完整的翻成中文的，是需要專業的，主要是因為英語的句子的結構，和中文的相差很多，英語句子經常會很長一大串的，或是有倒裝句的語法，或是有一些加強語氣的修飾部份，但在中文裡這類用法不一定有。其次，你根本也不需要這樣作，你的目的主要只是讀懂這些句子裡面說什麼，而不是要學習如何專業的翻譯的英語句子。

選擇適合的文件與學習文章也是必要的，如果你願意看英語的教學文章，你會發現另一片天空，英語的程式語言教學文章，比中文多得太多，而且通常都是最新的。不過，有些文章是很進階的或是講解底層的設計，裡面會有很多專用術語，沒有基礎是看不懂的，所以一開始你可以先大概粗略瀏覽一下，如果文章裡出現非常多你沒見過的專用術語(例如一大堆英文縮寫)，或是有一些複雜的流程圖、結構圖等等，這篇文章可能不太適合你的程度。

通常在官方網站上的入門教學，或是針對初學者的教學文章，裡面的英語字詞和句子通常也很簡單，很適合初學者觀看，經常的多看幾篇，之後你大概很少會再去查字典，這就代表你的英語程度提升了。至於要給開發者查詢的線上 API 手冊，或是一些技術的、新版本的消息，裡面該講什麼就講什麼，不會有太多廢話，但是有些技術是進階的，或是針對特定情況才會用到的，這種的你也不需要花太多時間一定要讀得很詳細，有需要用到時再查閱就可以了，資訊科技進步很快，太深究一些現在還用不到的、或是還學不了的內容是不必要的，也會浪費寶貴的時間。

值得要提出來的是，有些書籍或文章的作者，會很習慣用日常生活中的、或是時下流行的一些東西或事物來作說明或比喻，例如有些教學者喜歡用電影中的台詞或場景來比喻一則新技術消息，你如果不是生活在美國或當地的人，就算看得懂上面的意思，也無法有感同身受的體會，這類的文章就相當傷腦筋，有很多英文名詞會與程式語言一點關係都沒有，花時間在查這些字詞也會浪費時間。另一種就是用的範例都是以美國當地為主，和我們的一些文化或商業習慣不同，像網路商店程式這種例子，在台灣用不到小數點這種價格數字，在美國很常使用像 \$99.99 這種價格，這些例子就會和真正我們要用在實務上有一些差異，這點要非常注意。(有些翻譯成中文的書籍中也有類似的問題)

最後，有一些不錯的針對 JavaScript 初學者教學，在網路上都可以免費觀看，如果你有興趣不妨花點時間用上面說的方式來閱讀看看：

- [你所不知道的 JS](#)

練習英文打字

撰寫程式語言的基礎是英語，英文打字是一個必備的技能，打字速度愈快、愈正確你在撰寫程式時會愈有效率。如果你是一般職業學校的資料處理科系，通常都有中打和英打這兩門課程，學校課程會要求一定的速度和正確程度。但一般的大專院校卻不注意"打字"這種基礎的教育課程，我曾看過很多大專院校的資訊科系學生，甚至是研究所的學生，撰寫程式碼時還在看鍵盤找英文字或符號的鍵在哪裡，這樣子撰寫程式怎麼會有效率呢？

如果你還在用一指或是兩指神功輸入英文，你應該先學習如何正確地使用你的雙手 10 支手指頭來打英文字，然後多加練習，讓你的英文打字至少有一定的水準。網路上有很多免費的課程和軟體可以使用，我們並沒有要說達到多快才可以，至少你在撰寫程式碼時，不要經常地還在看鍵盤，這會養成壞習慣，會浪費過多的時間還在打字上。

當然現在也有一些專門針對撰寫程式碼的打字訓練網站服務，我是覺得你能多動手寫(打)一些程式碼，當作練習就可以了，不用太專門為程式碼作打字練習。下面是一些免費的線上英打課程，按照這些課程一步步就可以學好英打，雖然它們都是英文的網站，你應該很容易可以找到課程如何開始：

- <https://www.typing.com/student/start>
- <https://www.typingclub.com/>

英文打字的自我訓練方式，我會建議最好要固定的、持續的進行一段時間，例如每天固定抽出半個小時或是一個小時來訓練，不需要太長的時間，然後按照課程安排的進度，進行反覆機械式的訓練就可以了。如果能這樣堅持一段時間例如半個月或一個月，就會發現有效果了，一下子訓練太長時間反而不會很好的效果。一直到你可以不需要看鍵盤，能正確地打出 26 個英文字母、數字、符號，其實就差不多有一定的水準了。

學習如何使用開發環境與程式碼撰寫工具

最一開始的 JavaScript 程式碼，其實是直接寫在 HTML 檔案中的，這種寫法到現今還是被廣泛的使用，我有時候也會有這樣寫的情況，主要是一些網頁中的小功能，不過這種寫法問題很多。首先，你不能寫太複雜的程式碼，愈複雜愈容易出錯，這種寫法很容易會與這個網頁中的其它程式碼互相干擾，所以很難管理與除錯。再來，你也不能使用太新的標準的 JavaScript 語法，太新的語法在舊的瀏覽器版本不能執行，這也是個很麻煩的事情，所以你只能用舊語法(ES5)來寫，對初學者來說，這是件不容易的事情。

ES5 和 ES6(ECMAScript 第 5 版和第 6 版)是一個明確的分水嶺，ES6 中加入了大量的新語法，大概是近期內最大一次的改變，如果你想要學習使用新的語法，但又想要和目前最多數量的瀏覽器取得相容性，你該在什麼地方寫這些程式碼？又該怎麼執行這些程式碼？這是一些需要了解的問題。

對於初學者來說，應該學習的是目前新的語法和 API，而不是還回頭學那些舊的東西，我們都知道資訊科技汰舊換新非常快，這些舊的東西，市場也會很快淘汰掉它們，所以還學習舊東西會變成浪費時間。如果現在要開始學習，就得從最新的開始學習才是正確的方式。

一開始當然有很多種方式，可以開始來練習我們所寫的範例，有些書或教學文章的教學者，會建議用瀏覽器的主控台，或是用 Node 的直譯工具，有些則會用一些網路上的撰寫程式碼的服務，像是 jsbin、jsfiddle、codepen 這些等等，這些都可以用瀏覽器來使用，線上有個編輯程式，可以簡單的、很快的來編寫程式碼，然後就可以看到結果，感覺馬上就可以來練習學習。不過，用這些方式是對的嗎？答案是沒有什麼對或錯，每個

教學者的偏好而已，有些教學者則是不想一開始就因為開發環境或工具的複雜部份，先嚇跑初學者的學習意願。

我個人的觀點是，你應該要思考另一個問題，你如果之後學到一個程度，要開始真正認真的開發一個應用程式，或是要到一家公司去實習或當正式的程式開發者，請問他們是用剛說的那幾種方式在開發 JavaScript 應用程式的嗎？

答案都不是，"完全沒有一家公司是這樣開發 JavaScript 應用程式的"

所以你學這些方式是要作什麼用的，只是體驗性質的？還是學來殺時間的？千萬不要認為因為我們一開始只是寫一些小範例而已，所以用不著這麼麻煩，建置一個開發環境或是安裝什麼開發工具，你如果之後學到一定的程度後，認真去比較這些線上服務的，與實務上用開發環境，它們少了很多東西，實際上差異很大。而且我覺得最重要的，它們無法妥善地、有組織地管理程式碼專案。你不可能永遠都只在一個程式碼檔案中寫程式，你的程式碼與檔案都會拆分、然後再整合，你的專案一樣會因為規劃而拆成幾個部份，另外還有每個程式碼檔案一定需要的版本管理，這些線上服務有可能有一些這樣的功能，但都過於簡單而且不好用。至於瀏覽器的主控台，或是 Node 的直譯工具，你就把它當作只是執行一小段程式碼的遊樂園就好，它們不是設計來讓你撰寫程式碼用的。

有很多學習的方向，你當然可以選擇，我的建議就是要學就直接學實務上是怎麼使用的。雖然，你可能會多花一點時間要安裝一些軟體，或是調整一些設定，這些花的時間也是一種學習，因為實務上就是這樣，你今天不先學，將來到職場上或是參加到開發團隊中，一樣還是得學著用。

JavaScript 的開發工具和環境，對比其它的程式語言來說，已經夠簡單、夠容易學了，你可能還沒有撰寫其它應用程式的概念，如果你今天要開發的手機上的最流行的 App，光要建置起整個撰寫程式碼、除錯、模擬器、實機測試...一堆的有的沒有的環境，可能就會讓你花上好幾天的時間。

測試伺服器環境、套件管理工具

現在都會建置一個開發用的伺服器環境，用來模擬程式執行，因為 Node 與 NPM 工具用於管理 JavaScript 的各模組相當方便，所以這是在實務上都會這樣使用的方式。一開始我們要先在電腦上安裝 Node，它是一套可以在伺服器環境執行 JavaScript 程式的系統，裡面就有包含 NPM 工具，NPM 工具則是專門用於管理 JavaScript 套件相依性的。下面是它的官方下載連結：

- [Node 官方網站](#)

在官方網站上，你可以看到有兩個下載的按鍵區塊，一個是像是「8.12.0 LTS」後面有 LTS 字詞的，但版本號碼可能不一樣，另一個是「10.10.0 Current」後面有 Current(目前最新)字詞的。LTS 是長期維護版本的意思，如果你並沒有要開發或學習 Node，也就是伺服器端的 JavaScript 程式，下載這個 LTS 版本就可以。

Node 的安裝就像是一般你在作業系統上安裝軟體一樣，會有一個安裝的精靈程式，引導你一步步設定然後安裝，設定的部份使用預設就可以，這部份不會有什麼困難的地方。

麻煩的地方可能會出現在日後的維護和升級部份，對於 Windows 作業系統的開發者來說，如果要升級就直接下載新的版本，再重新安裝一遍就可以。macOS 作業系統的開發者來說，也可以用重新安裝的方式，或是為了日後方便起見，改用 Homebrew 來安裝 Node。不過，如果你也是要學習 Node(伺服器端的 JavaScript)要如何開發，那麼安裝多個 Node 版本的方式，是一定要學習的，至少你的作業系統裡要安裝 LTS 和 Current 兩種版本，這樣才能方便測試和使用新的功能，這個時候就需要用 n 或 nvm 這種套件(它們都是 NPM 的套件)，來管理和切換到不同的版本。

程式碼撰寫工具、相關套件

現在通常的程式碼撰寫工具，都使用 Visual Studio Code(VSC) 這套由微軟出品的軟體，它是開放原始碼的專案，而且可以在各種作業系統上使用，最棒的是它還提供了中文的操作介面。不過 Visual Studio Code 與微軟的另一個知名的開發整合式工具 Visual Studio 是完全不同的軟體，這個千萬不要搞混了。下面是它的官方下載連結：

- [Visual Studio Code](#)

在官方網站上有一個「Download for Mac」或「Download for Windows」字詞的下載區塊，按下後就可以下載這個開發工具。然後依照一般的軟體安裝方式來安裝到你的電腦中。

Visual Studio Code 只有幾個區域，所以很簡單就可以理解每個區域是作什麼用的。

Visual Studio Code 一開始安裝好之後，只有內建的功能和套件，所以我們還需要利用它的擴充套件功能，來安裝一些搭配的套件，以下是幾個常用的 JavaScript 套件，你從擴充套件中可以用名字來搜尋它們，然後安裝它們：

- Babel JavaScript - 支援最新的 JavaScript 語法和 API，可以高亮度顯示
- ESLint - 搭配 ESLint 套件使用
- vscode-icons - 不同檔案以特定圖示顯示
- Prettier - 程式碼格式化排版用

當然套件還有很多，這裡只是一些最常用的而已，你可以視需求再加裝其它的套件，例如搭配除錯器，或是特定 JavaScript 函式庫或框架使用的。有些套件是安裝後就會自動套用，有一些則是需要再進行設定才能正確運作，這部份就要看每個套件的說明文件。

Visual Studio Code 從下方的狀態列點按後，可以叫出主控台的區域，裡面就有可以輸入命令的終端機功能。或是從上方的選單中「檢視」->「整合式終端機」也可以達成。在專案中我們需要加入新的套件模組，或是啟動測試伺服器，都需要輸入指令，所以知道如何打開終端機的輸入命令的介面也是很重要的。

上一章節我們裝好了 Node 與 NPM 工具，你可以試著在終端機輸入命令的介面，輸入以下的指令，看看 Node 與 NPM 工具是不是已經安裝好了，這兩行指令是要取得目前已安裝的版本編號：

```
node -v
```

```
npm -v
```

啟動專案工具(Starter Kits)或專案樣版文件(boilerplate)

現今在專案開發時，如果是較為大型的專案，通常都會由開發小組自己維護一個專案的樣版，裡面會先設定好所需的套件、相關的測試設定等等。然後其它有參與的開發者，會直接從這個專案樣版，來建立自己的開發用的專案。不過，這種專案樣版文件，通常都會針對某些特定的函式庫或框架來設計，在網路上也有很多這類的專案樣版文件，針對不同的需求而加入各種套件。

當然你也可以自己建立一套，不過因為專案樣版文件通常會包含以下幾個套件或工具，所以你也需要多花時間一一了解：

- babel
- webpack
- eslint
- hmr
- test

- css module

而且專案樣版文件麻煩的地方除了設定外，還有日後的更新與升級，所以我會建議初學者用現成的專案樣版文件即可，因為我對 React 開發較為熟悉，所以我會推薦直接使用下面這套專用於 React 學習與開發的專案樣版工具：

- [create-react-app](#)

要注意的是，這工具並不是直接從上面的網站中下載來安裝，它是用你剛剛安裝好的 NPM 工具，來安裝到你的電腦中。我們在終端機中輸入以下的指令(下面install可以只簡寫為 i):

```
npm install -g create-react-app
```

這行指令會安裝create-react-app工具程式到你的電腦中，-g的參數代表它是要安裝到電腦的全域，也就是你的電腦各地方都可以直接用這個工具。

安裝完成後，我們可以用這個指令create-react-app來建立新的專案，在終端機中輸入以下的指令：

```
create-react-app my-app
```

這行指令會在你目前終端機所在的目錄位置(通常是使用者各別的目錄中)，建立一個新的名稱為my-app資料夾，然後把相關的專案套件安裝在裡面。這需要一點時間來安裝整個模組套件，當然因為我們現在只是初學 JavaScript 而已，這裡面有一半以上的東西還不會用到，用這專案樣版工具來建立新專案只是圖個方便而已。

再來開啟一個新的 Visual Studio Code 的新視窗，然後點按"開啟資料夾"，指向你剛刷新建立的專案my-app 資料夾，在左邊的檔案總管區域應該會看到目前專案的所有檔案列表。接著再開啟在終端機中輸入以下的指令：

```
npm start
```

這個指令可以啟動測試伺服器，並且會自動開啟你的預設瀏覽器，連到`http://localhost:3000`這個網路位置，localhost 代表的是你自己這台電腦，3000 指的是埠號 3000，這是它的預設伺服器用的埠號。如果你能正確地看到執行的預設頁面，那麼就是這個專案樣版已經安裝成功了。

在專案的檔案中，我們在一開始學習時，只會用到兩個檔案，一個是 `src` 目錄裡的 `index.js`，另一個是 `public` 目錄中的 `index.html` 檔案，其它的檔案你可以先不用管它們。測試伺服器可以先啟動，然後把 `index.js` 先清除裡面的內容，開始練習撰寫你的程式碼，因為有熱模組抽換的功能，所以你在修改程式碼的同時，瀏覽器裡的測試網頁也會一起跟著重新執行出結果。

最後，如果你需要把寫好的程式，傳到正式的網站上執行，你可以在終端機中輸入以下的指令：

```
npm run build
```

上面這行指令，會將目前的程式碼進行編譯和壓縮，你會看到有一個 `build` 資料夾出現在你的專案目錄中，這裡面就是編譯過的 `index.html` 和 `js` 相關檔案，傳到伺服器上就可以執行。當然，有一些在 `src` 或 `public` 目錄中沒用到的檔案在編譯前最好先移出專案之外，以免造成編譯的檔案過大。

第二章：變數與常數

變數(Variable)在 JavaScript 語言中扮演了重要的資料(值)存放角色。你可以把變數想成是一個容器或是盒子的概念，你可以把不同的數值放在其中。

首先要先校正我們長期以來在數學教育中學到的知識，如果套用到程式語言教育時，講法和意義上是有所不同的，像下面這個的一段程式碼語句：

```
a = 1
```

要怎麼解說這一段程式碼的意思？如果用我們學校中學過的數學知識來說明，我們會說“a 等於 1”。不過在程式語言教育中，這說法是不正確的，如果你遇到一個教程式語言的老師或文章，還在用“a 等於 1”的講法，你可以認為教學者沒真正了解這最基本的程式語言知識。

那麼，正確的說法應該是什麼？

用英語的講法是“assign the value 1 to a”，在英語中“assign”與“give”的意義相近，所以中文翻譯應該是“給定 a 一個值 1”，“給定”或是“指定、指派”意思都是相近的。所以這並不是一種“等於”的運算，因為“等於”會有“相等”的意義，英語的對應單詞是“equal”，在程式語言裡面有“相等比較”的另一種運算情況，而不是這裡的“指定”運算。

註：單一個“equality”字詞在程式語言中，通常是指“值相等”的運算，是值相等符號(==)這種比較運算。另一種“identity”或“strict equality”(嚴格相等)，指的是更嚴格的除了“值相等”之外，“資料類型”也要相等的運算，是嚴格相等符號(===)的這種比較運算。

因為變數有“容器”的概念，“指定”運算則是有將數值放入到容器之中的意思。你可以想成把你的左手打開，放入 1 塊餅乾，然後把右手打開，放入 2 塊餅乾，你的左右手就是變數，但經過運算過後(放入餅乾後)，你的左右手並不會“等於”餅乾，你的左右手只是單純地被放入餅乾而已。

接下來的問題是，為什麼要用“變數”這東西？

在程式語言中，變數的角色像是一個資料的代名詞。資料通常就是單純的資料，像是數字 1234、或是一大長串的文字等等，你從資料上是看不出它是作什麼用的，例如說我們想要了解學校有多少學生，現在知道學校有 20 個班級，然後每個班級有 40 個學生，所以會像下面這樣定義：

```
let numberOfClassesInSchool = 20
let numberOfStudentsEachClass = 40
let totalNumberOfStudentsInSchool =
    numberOfClassesInSchool * numberOfStudentsEachClass
```

這只是個簡單的例子，實際上可能每個班級的學生並不一定剛好 40 個，甚至每個年級的班級數量也不一樣。這裡只是方便讓你理解變數的作用是什麼，基本上就是作一個裝載資料的代稱，讓這個裡面的資料產生類似有標籤的意義，方便開發者在需要的時候用代稱來取用、修改這些裡面包含的資料。

變數使用“let”來作為定義(或宣告)的關鍵字，而另一個所謂的“常數”則是使用“const”。變數與常數兩者的差異，對初學者來說，一定要很清楚的理解。變數中儲放的資料(值)，想成是暫時存放的，而在常數中的資料(值)

則是要長期(永久)存放的，一旦給定值後就不會再改變的資料。範例如下：

```
// 這個a是不可變的(常數)
const a = 10

// 這行程式碼會發生錯誤： "a" is read-only(只能讀不能寫)
a = 11

// 這個b是可變的(變數)
let b = 5

// b可以再改變其中的值
b = 6
```

為了要厘清這個概念這裡需要再解說一下，我們在中文裡認為的"常數"的說法，有可能你會誤認為它是變數的相反意義。實際上，在 ECMAScript 的標準規範中，用來敘述"常數"的字詞是"constant variable"，中文翻譯會是"固定的變數"，所以它在設計上是一種特殊的變數，"常數"較為正確的中文解釋如下：

"常數"是一種"一旦給定值後就不能再更動值的變數"

以之前的實例來比喻，你的左手打開放入 1 塊餅乾後，就把手握起來，之後要再放入其它的餅乾就作不到，這樣就是常數的設計。

在實務上 const 宣告比 let 使用的頻率高出很多，大部份情況都是用 const 宣告，只有用 const 宣告不合適時，才會用 let 宣告。但不論你用 let 或 const 宣告，養成下面這個好習慣是一定要的：

不論是變數或常數，在宣告(定義)時就要給定一個值

常數當然不用說，如果在宣告時不給定值，就會出現錯誤。但變數也要養成這個習慣，因為變數在宣告時，原則上是可以不用給定值的。一開始宣告時就給定值，有一些好處，我們會在下個章節說明真正的用意。如果你一開始在定義變數時，不知道會是給定什麼初始值才好，那麼就先給定 null。

註：在 ECMAScript 的標準規範中，變數與常數很多是相似的設計，其中如果是關於"variable"的相關說明段落，其實是指變數與常數兩者，有特別要說明常數才會用"constant variable"。如果你想要看看標準裡面寫了什麼，這一點特別要注意，當然這不太適合初學者觀看，標準裡面都是一些內部定義和設計的說明。

動態資料類型(鬆散資料類型)

有一些專用術語，會在腳本語言中看到，這是一般腳本語言共通的特性，例如鬆散資料類型、動態資料類型或弱資料類型等等。這些說法的意義是相類似的，意思是說當變數(常數)在宣告與指定值時，變數(常數)會依照給定的值，而自動改變自己的資料類型。例如下面的程式碼範例：

```
// a現在是數字的資料類型
let a = 5

// a現在是字串的資料類型
a = 'a string'
```

這看起來不是很理所當然的？難道這樣作是有問題的嗎？

完全沒問題，只是程式語言實際上有很多種類，有些設計得很嚴謹的程式語言(例如 Java)是不能這樣作的，在這些程式語言中，變數是一開始就要規定是只能用什麼樣的資料類型，例如只能存放整數的變數，你指定一個字串給它就會發生錯誤，反之亦然。嚴謹有嚴謹的好處，當然鬆散也有鬆散的好處。

嚴謹是因為電腦實際在底層運算時，不同的資料類型的運算，是天差地遠的設計，這樣作會容易作程式的最佳化，另外程式碼寫得愈嚴謹，當然出錯的機會也會愈少，缺點就是不容易學習和使用上會比較麻煩些。

鬆散是方便讓開發者使用和學習，不用去管這麼多複雜的資料類型宣告，至於程式的最佳化就交給執行的引擎去處理。相對的它的缺點也很明顯，就是很容易在資料類型上運算時有出錯的情況，而且執行的引擎相對要處理的工作多就會複雜的多。

JavaScript 既然是鬆散資料類型的程式語言，為了讓你的程式碼減少出錯的機會，你可以養成的兩個習慣是：

- 優先使用 const，因為 const 的資料類型一宣告就會決定了，而且之後無法改變
- 使用 let 時，資料類型最好也要固定，在宣告變數時就要給定一個初始值。如果初始值不明確，可以使用 null。

其它的還有不要在陣列裡面用不同的資料類型成員等等，這些都是與資料類型有密切關係的撰寫風格，養成這些撰寫程式碼的好習慣，可以大幅度的減少，在撰寫程式碼時遇到鬆散資料類型造成的問題。

另外，有些輔助工具可以協助開發者在撰寫程式碼的期間，就可以定義得更為清楚，例如另一套由 Facebook 出品的 Flow 工具，就是作這用途的。也有另一種方式，是改用類似 Java 語言的強資料類型的語法，最後再輸出編譯為 JavaScript 程式碼來執行，例如微軟出品的 TypeScript 語言，實際上真的就是把 JavaScript 當成 Java 在寫就是，這已經算是具有嚴格資料類型的另一種程式語言，不過學習門檻會變得比較高，有些開發者認為這樣會比較好。

複合型資料類型(參照資料類型)

雖然我們在下一章節才會說到各種資料類型，但這裡談到變數和常數，也得了解一下在集合型資料類型的變數與常數的宣告。

所謂的複合型的資料類型，在 JavaScript 中最常見的就是陣列(Array)和物件(Object)兩種，實際上陣列是經過特殊設計過的物件資料類型。我們也常把這種複合資料類型，稱為"參照資料類型"，"參照"的意義你可以把它想作是"地址"的意思，這種資料類型因為它是集合很多資料在裡面，所以變數(常數)裡面記載的是一段"地址"，而不是真正的原始資料，原始資料是像上面說的數字、字串這種資料。

"地址"的概念就像我們日常生活中，一棟大樓裡可能共用同一個地址，然後再區分樓層這樣。所以當某個樓層住進了新房客，或是搬出新房客，地址並不會有所變動。因此，這個概念就可以套用到我們在變數(常數)的宣告上，像下面這個的範例：

```
let a = [1, 2, 3]
let b = a
a[1] = 10
console.log(b) // b陣列中會是什麼？
```

因為記錄到的是地址，所以實際上範例中的 b 陣列它會和 a 陣列的地址相同，所以當 a 陣列中的成員有變動時，b 陣列中一樣會看到變動，所以最後的 b 陣列是 [1, 10, 3]，和 a 陣列一模一樣。

當常數用於定義陣列時，如果進行更動其中包含的成員時，就像地址相同但樓層的房客有所變動，所以是可以作得到的，像下面的範例：

```
// a陣列是一個常數
const a = [1, 2, 3]
a[1] = 3

// b陣列是一個變數
let b = [1, 2, 3]
// b陣列要整理換成a陣列
b = a

//下面這段程式碼會有錯誤
a = b
```

陣列可以用常數宣告，如果你只是要改變裡面包含的資料而已(同一地址大樓的不同樓層房客)，就像這裡宣告的 a 陣列。但如果你要把整個大樓不要了，要換一個新的地址，這時候就是要用變數的宣告才可以，用常數宣告就會有錯誤。

在實務上，我們很少會去作像這種直接更換地址的事情，而且也建議少這樣用。假設你換了一間新大樓，那舊的那間大樓到哪去了？交待給執行的底層去作資源回收了，而資源回收都是有代價的。

因此，我們對於複合型的、參照的資料類型，主要是陣列與物件，它們的在定義時通常是像下面這條使用建議原則：

■ 對於陣列與物件的資料類型，大部份情況都是使用 const 來宣告

最後，還有另一種資料類型，實際上它也是一種特殊的物件類型，也就是 JavaScript 中相當重要的"函式"，它也有一種宣告定義的語法，類似於變數(常數)的宣告方式，像下面這個範例：

```
const foo = function(a) {
  return a + 1
}
```

你可以先記住下面的建議原則：

■ 對於函式表達式的宣告，一律用 const 來宣告

所以這樣大概可以知道，為什麼 const 比 let 還常用，一般情況下也都是優先使用 const，只有真正那些需要之後改變其中的值時，才會用 let 來宣告成為變數，不過有一些特別的情況是只能用 let 的，例如 for 迴圈定義中的第一個語句。

變數與常數的命名

我們需要先為這些數值的代表名稱命名，稱為常數名稱(constant name)或變數名稱(variable name)。以下說明的命名規則，不只包含常數與變數，還包含了函式、類別名稱的命名。

命名規則

變數或常數名稱基本上要遵守以下的規則:

- 開頭字元需要是 ASCII 字元(英文大小寫字元)，或是下底線(_)、錢號(\$)。注意，數字"不可用"於開頭字元。
- 接下來的字元可以使用英文字元、數字或下底線(_)。
- 大小寫是有差異的。
- 名稱不可使用 JavaScript 語言保留字詞。
- 名稱被稱為 識別符 (identifier，簡稱 ID)，它在程式碼上下文中具有唯一性。

注意: 以下底線(_)為開頭的命名通常是有特別用途，它是用在類別中的私有變數、常數或函式(方法)。錢號(\$)通常也會用於特殊的情況。

好的變數(函式、類別)命名

變數、函式、類別命名

變數與函式，都用小駝峰式(camelCase)的命名。只有"類別"會用大駝峰式命名法(CamelCase)命名，例如以下的命名範例:

```
let number0fStudents  
const number0fLegs  
function setBackgroundColor() {}  
class Student {}
```

常數命名

在許多舊式的風格樣式指引中，因為沒有常數專門使用的宣告方式，所以會建議使用全大寫英文命名，字詞間使用下底線(_)連接:

```
const NAMES_LIKE_THIS = 'Hello'
```

不過，因為 ES6 中加入了const用於指示為常數後，這個規則不再需要，程式檢查工具或執行時都會用常數的方式來檢查。所以你可以用一般對變數的命名規則就可以了:

```
const helloText = 'Hello'
```

不好的命名：用簡寫或自己發明的縮寫

命名時的英文字詞，應該寫得愈清楚愈好，最好不要自己發明縮寫

- 不好的命名: setBgColor / 好的命合: setBackgroundColor
- 不好的命名: userAdr / 好的命名: userAddress
- 不好的命名: fName, lName / 好的命名: firstName, lastName

注意：使用長一點的命名可以提供更佳的閱讀性，而且與效能一點關係都沒有，JavaScript 的程式碼最後都會再經過醜化與壓縮，變數名稱會用很短的名稱來取代，這點與程式開發中使用的名稱無關。

不好的命名：語意不明或對象不明

動作或名詞最好加上對象是誰，或是限制一個範圍

- 不好的命名: insert() 這是是要插入什麼？ / 好的命名: insertDiv()
- 不好的命名: name 這是什麼名稱？ / 好的命名: memberName
- 不好的命名: isOk() 什麼東西 ok 不 ok？ / 好的命名: isConnected 連上線了嗎？

不好的命名：拼錯英文單字

拼錯英文單字非常常見，專有名詞不了解時請多查字典

- 錯誤的拼字: memulitem / 正確的拼字: menuItem
- 錯誤的拼字: pueryString / 正確的拼字: queryString

英文單複數

陣列之類的集合結構，有數量很多的意思，大部份都用"複數"型態的字詞，或者資料的類型來分別，例如：

```
studentArray  
students  
todoList
```

執行的動作（函式或方法），如果針對單一個變數的行為，後面接的名詞會用單數：

```
addItem()
```

如果針對多個數的行為，後面接的名詞會用複數：

```
addItems()
```

針對全體的行為，會用「All」：

```
removeAll()
```

常見的英文計量字詞：

count numberOf amountOf price cost length width height speed

常見的布林值開頭字詞：

isEmpty hasBasket

常見的字串值開頭字詞：

string name description label text

常用的動作詞（函數用）開頭

- make take 作某...事
- move 移動
- add 加上、相加
- delete/remove 移除
- insert 單體 splice 複合體
- extend append 展開
- set 設定
- get 獲得
- print 印出
- list 列出
- reset 重置
- link 連至
- repeat 重覆
- replace 取代
- find search 尋找
- xxxxTo 到 xxx

具時間意義或指示的字詞

- will 通常指即將發生但未發生
- did 已發生
- should 應該發生

其它命名相關的風格指引

- 避免使用單個英文字元的命名，像 q, a, b, x, y, z
- 避免在命名的前後使用下底線(_)，例如__firstName__、_firstName、firstName_都是很糟的命名

第三章：資料類型

資料類型是剛初學程式語言時，首先開始學習的部份。像 JavaScript 這種腳本語言，它們的資料類型都會設計得較為簡單，以此更容易學習，而在這個章節中我們也將了解到所謂的"動態類型語言"或"弱資料類型語言"的說法是什麼意義。

資料類型

資料類型是一些對於電腦中即將被處理的資料分類，每一種資料(data)在程式語言中會區分有不同的處理方式，以及與其相關的各種運算與 API，例如數字資料，就會有加減乘除等運算，字串資料就會需要計算長度、組合或分割不同字串等等的運算。資料是所有電腦運算的基礎，對於電腦來說，所有的資料都只是 0 與 1 的訊號。但對於人類來說，資料類型的區分就複雜得多了，需要因應各種不同的情況來使用。

在 JavaScript 中的基礎(原始)資料類型只有 7 種，大致上可以像下面這樣再以不用用途來區分：

- 基礎: 數字(Number)、字串(String)、布林(Boolean)
- 空值或未定義: null、undefined
- 複合(集合)或參照：物件(Object)、陣列(Array)
- 特殊: 符號(Symbol)

註: 陣列(Array)實際上也是一種物件(Object)的資料類型，因為很常使用所以會獨立出來討論。函式(Function)也是一種物件的延伸的資料類型。

基礎類型

數字(Number)、字串(String)、布林(Boolean)是最基本的三種資料類型，也是最常被使用的一些資料的類型。這三種資料除了我們需要學習它們的一些基本的特性外，另外進一步學習它們怎麼作運算，最後當然是這三種是怎麼相互的轉換。

這三種資料類型，JavaScript 分別為它們設計了各自的包裝物件互相對應，也就是名稱為 Number、String、Boolean 這三個內建的物件，相關的 API 或一些屬性，都在這三個物件上面，可以對來對原始的資料來進行各種操作。

註：除了包裝物件外，JavaScript 還有內建三個 Number()、String()、Boolean()方法，這三個主要是用來轉換類型用的，不過也很少會用到這三個方法。

數字

數字(Number)，像是 1234 這種數字的這種資料，JavaScript 語言中只有數字一種類型，沒有像一些其它的程式語言，對於數字會區分不同的類型，例如整數或浮點數(有小數點的數字)，所以對 JavaScript 來說，1.0 與 1 是同樣的。

數字(Number)在所有的程式語言中的設計是有其極限的，電腦不可能幫你無極限的計算出任何的數字，所謂的極限是有最大值和最小值，以及一些例如"1/0"這種得不到值的運算，或是小數點除不盡的情況。另外，浮點數(有小數點的數字)的運算也有所謂"精度"的問題，例如像下面這個在討論區很常見的問題：

```
0.1 + 0.2 //0.30000000000000004
```

這答案對我們來說很理所當然的是 0.3，不過但對程式語言最後給出的答案並不是，這是因為浮點數的設計，這也與 JavaScript 語言無關，幾乎常見的其它語言都是一樣的設計，JavaScript 語言雖然表面上看起來用簡單的一種數字類型，可以通用於浮點數或整數的定義，但實際上在執行運行前，執行引擎一樣會對這些數字區分，所以看起來很容易使用，並不代表它執行引擎的內部設計也是很簡單，相對來說反而更複雜。

與數字有關的還有幾個很特別的數字定義值，下面這兩個也是數字資料類型，但它們代表的某些特殊的情況：

- Infinity - 無窮大(不帶正負號時為正無窮大)
- NaN - 不是數字(Not a Number)。通常是運算或轉換後得到的值不是數字時

數字經常需要從其它的類型轉換過來，而其中最常見的是字串轉為數字，最常從數字轉換出去的，也是數字轉為字串。

數字要轉為字串的操作，通常在轉出到 HTML 時直接給定一個數字，讓瀏覽器的引擎直接輸出即可，不需要多作什麼。如果真要轉為字串的話，最簡單的方式是和一個空字串用加號(+)運算，這裡的加號(+)實際上是"字串連接運算符"，原因在下面的字串章節有說明，例如下面的範例程式碼：

```
const a = 123 + '' //得到 '123'字串資料
```

字串要轉為數字的操作，通常是一種被稱為"數字字串"的東西，有被單引號或雙引號括住的數字的，實際上是一種字串。字串轉為數字的最簡單的語法也是使用加號(+)，這裡的加號(+)稱為"一元正號運算符"，這是符號共同的情況，例如下面的範例程式碼：

```
const b = +'101' //得到 101 數字資料
```

加號(+)的符號共用情況，是初學者一定要學好的，在不同的情況有不同的運算，它有三個地方會用到同樣的符號：

- 數字相加的運算
- 字串連接的運算
- 數字的正號(稱為一元正號)

在 JavaScript 語言中不是只有加號(+)會這樣，之後你會再看到其它的，這些都是很容易造成程式錯誤的地方，所以一定要真正的理解這些。初學者只要先了解上面這三種情況，分別在什麼時候會用到，還有它們是誰優先誰居後就可以了。

因為在 ECMAScript 標準中的加號(+)運算規則，會比你想像中還複雜很多，我曾在網路上有發表過一篇"JS 中的{} + {}與{} + []的結果是什麼"的文章，內容都是對照標準中的步驟與規則，但這篇文章可能比較不適合初學者閱讀，如果你有興趣可以再找來看看參考。

字串

同樣的 JavaScript 中也沒有區分所謂的字元和字串，它只有字串這種用於描述文字的資料類型，不像有的其它程式語言中對於文字相關的資料類型區分得那麼細。除此之外，字串有些 API 與陣列中的 API 非常相似。

字串的定義可以用單引號('')或雙引號("")，但對 JavaScript 語言來說，也是完全一樣沒差別，這部份看每種撰寫習慣風格的不同，有些函式庫建議都用單引號('')，有些則建議都用雙引號("")。但是單引號('')或雙引號("")只能選擇其中一種，你不能在同一字串中混用兩種符號。

註：我個人是在 JavaScript 中使用單引號(')的字串風格，雙引號("")則會保留給 HTML 的屬性字串使用

另一種最近加入的字串定義用的符號是重音符號(`)，這稱為樣版字串(Template strings)，如果你的字串中還要插有變數可以轉換的話，用這種新式的定義是比較合適的。樣版字串會用到一些特定的情況，你可以再看我寫的電子書的內容了解詳細的語法說明。

值得注意的一點是，因為 JavaScript 最常搭配的操作介面是網頁，也就是以 HTML 語法所編寫而成的一種文字格式檔案，HTML 的全名是 HyperText Markup Language，所以它本身也是一種語言，只是語法都是在標記文字之用的。在傳統的 JavaScript 與 HTML 搭配的應用程式，這兩者之間不論是從 HTML 上取得資料，到 JavaScript 程式中，或是要把 JavaScript 程式中計算完的結果，顯示在 HTML 中，對 HTML 來說都是"字串值"。

也就是說，你今天在 JavaScript 程式裡，計算了一個 $1+2$ 的數字運算，然後最後把它呈現在網頁上，看起來是 3 沒錯，但實際上這個數字在 HTML 上呈現時已經是個字串，瀏覽器裡的引擎幫你作了自動轉換的事情。另一個常見的例子是，我們如果寫了一個網頁上的表單，讓使用者可以輸入數字，然後按下按鈕幫你作加總之類的數字運算，要特別注意你的 JavaScript 程式裡，一開始從網頁上的表單得到的數字，其實是字串類型，如果使用者輸入了 123，實際上你的 JavaScript 程式一開始獲取到的是一個字串"123"，你如果接著要作數字的運算，就需要先把這字串轉成數字類型，才能接著作數字的運算，這一點是初學者很容易搞錯的地方。

字串有一個最重要的、也是最常見的運算，就是"連接(concatenate)"運算這種運算，意思就是把兩個字串連接在一起，變成一個新的字串，使用的符號是加號(+)，像下面這個範例：

```
const a = '易學、易用的'  
const b = 'JavaScript'  
  
const c = a + b
```

字串連接運算用加號(+)不是什麼太大的問題，問題在於這個加號(+)它是一個符號共用的情況，基本上它有三個地方會用到同樣的符號：

- 數字相加的運算
- 字串連接的運算
- 數字的正號(稱為一元正號)

那麼，請問一下如果一個數字和字串間，用了一個加號(+)，像下面這樣子的範例，那是該算是字串連接，還是數字相加？(這問題據說是 JavaScript 初學者必考題)

```
const a = '10'  
const b = 1  
  
const c = a + b
```

答案是"字串連接"，因為字串連接的運算，會比數字的相加運算優先，所以這是一個需要記住的重點：

字串與數字間使用加號(+)運算時，是字串連接優先，而不是數字相加。其它非字串類型會被自動轉換為字串類型。

至於一元正號(+)這運算上面在數字章節有說過了，它是把其它資料類型轉為數字類型的運算符，像下面的範例是改自上面的範例：

```
const a = '10'  
const b = 1  
  
const c = +a + b
```

這裡的`+a`是先把 `a` 這個字串先強制轉變為數字，然後再和 `b` 變數相加，這裡的加號(+)就會變成數字的相加運算，而不是字串的連接運算。這個範例這樣寫是扣分的，你不需要經常考驗自己的眼力，或其它要看這程式碼開發者的程度，用`c = (+a) + b`來寫會比較容易看得懂。圓括號()在這個地方稱為"群組運算符"。當然，圓括號()也是一個有很多符號共用的運算符。

在這個字串的章節中，你一定要記住的是，在加號(+)的字串連接運算，它的優先運算權是高於數字的相加運算的，除了不同類型的轉換之外，優先次序也是一個很重要的議題，比較進階的優先次序部份，是不同類型的相等、大小於的比較，這部份你可以之後再研究。

布林

布林(Boolean)或簡寫為 Bool，它是由發明的科學家 George Boole 命名。是一種使用絕對兩分法的值(黑白/陰陽/真假)，在 JavaScript 中以關鍵字 `true` 與 `false` 來作為布林的兩種可用的值，這種值通常都是用來作判斷或比較用的。例如以下的範例:

```
const a = true  
const b = false  
  
1 == '1' //true  
b != a //true
```

注意: 所有的 JavaScript 關鍵字(保留字)都是小寫的英文，像 `true` 的話，如果寫成 `True`、`TRUE`、`TrUe`，都是不對的寫法。

布林只會用在是程式流程要判斷的地方，像是今天有沒有吃早餐？有或沒有，或是昨天有沒有睡超過 8 小時之類的，就是一個絕對是或否的判斷，如果是就運行這一段程式碼，如果不是就運行另一段程式碼，幾乎所有的程式語言都有布林值的設計。

JavaScript 中的布林資料類型，設計得與其它語言有一些不同的地方，首先是它有"真家族(truthy)"和"假家族(falsy)"隱含的設計，這兩個英文字詞就不用去查字典了，這兩個名詞是自創的、字典裡沒有的字詞，一開始是由 Douglas Crockford 大師所提出的。意義如下：

"假家族(falsy)"包含了 `0`, `-0`, `null`, `NaN`, `undefined`, 空白字串('')，當然也一定包含了 `false` 值

當然反過來說，不是"假家族"就是"真家族"了。但是，這設計有什麼應用上的意義呢？

我們看一個實際的例子，你想要寫一個從網頁上填寫表單的簡單應用，HTML 上有好幾個欄位，有姓名、Email、電話等等，然後你要在 JavaScript 程式裡，判斷使用者有沒有填寫必填的欄位，例如說姓名是必填的，所以如果使用者沒填這個欄位，就不會送出表單到伺服器中，那該怎麼寫這個判斷？

```
// 假設nameInputValue是使用者輸入的姓名字符串
if (nameInputValue.length > 0) {
    // 姓名有填，送出表單
} else {
    // 跳出必填的警告，不送出表單
}
```

字串本來就有一個 length 屬性，可以得到字串的長度，也就是字元數量(註：實際上是位在包裝物件 String 上的屬性)。這樣寫很合理，如果長度不是 0 的情況下，也就是一個有字在裡面的字串時，這個判斷就會成立了。但你仔細對照一下上面的"真假家族"的定義，有字的字串本身就是一個真家族的成員，所以像下面這樣寫就可以了：

```
// 假設nameInputValue是使用者輸入的姓名字符串
if (nameInputValue) {
    // 姓名有填，送出表單
} else {
    // 跳出必填的警告，不送出表單
}
```

也就是說nameInputValue如果裡面有字串值，必然就是"真(true)"，反過來說，如果nameInputValue裡面沒值，就一定是"假(false)"。這樣寫和上面的寫法，並沒有什麼結果上的不同，只是運用了真假家族的概念而已。這裡主要是說明 JavaScript 裡面有這種隱含的設計，雖然你不一定要用這種方式來寫你自己的程式碼，但有可能以後會經常看到別人的程式碼這樣寫，所以還是得學起來。而且最好是先背起來有哪一些是假家族的成員。

"假家族"與"真家族"的概念，除了會應用在判斷中之外，另外還有一個很重要的 JavaScript 特性 - "短路求值(Short-circuit)"的運算，不過因為篇幅的關係，在本書中不會講到這個部份，你可以上網找找相關內容，或是參考我寫的電子書。

註：Douglas Crockford 大師是知名的 JSON 格式的發明人，它有寫過一本"JavaScript: The Good Parts(JavaScript: 優良部份)"，適合已經有一些基礎的學習者觀看，但書的內容因為年代較久遠，有些概念與現今的 JavaScript 有些不同。

陣列

陣列是一種有順序的複合式的資料結構，用於定義、存放複數的資料類型，在 JavaScript 的陣列中，並沒有規定它能放什麼資料進去，可以是原始的資料類型、其他陣列、函式等等。

陣列是用於存放大量資料的結構，要如何有效地處理資料，需要更加注意。它的搭配方法與語法很多，也有很多作相同事情的不同方式，並不是每樣都要學，有些只需要用到再查詢相關用法即可。

註：雖然陣列資料類型也是屬於物件，但 Array 這個包裝物件的 typeof Array 也是回傳'function'。但是typeof用在的陣列資料值上，例如typeof [1, 2]則是回傳'object'。因此typeof並不適合檢測資料類型是否為陣列，這要改用 isArray()方法來檢測。

陣列定義

陣列定義有兩種方式，最常見的使用陣列字面文字的定義語法，以下說明定義的方式。

陣列的索引值(index)是從 0 開始的順序整數值，陣列可以用方括號([])來取得成員的指定值，用這個方式也可以改變成員包含的值：

```
const aArray = []
const bArray = [1, 2, 3]

aArray[0] = 1
aArray[1] = 2
aArray[2] = 3
aArray[3] = 5

console.log(typeof aArray) // object
console.log(aArray) // [1,2,5]
console.log(aArray.length) // 3
console.log(aArray[3]) // undefined
```

註：方括號([])在這裡是一個符號共用的情況，陣列定義是一種運算，存取陣列中的成員是另一種運算。

註：第二種陣列的定義方式是用new Array(10)這種長得很像其它物件導向語言的語法，這種方式不建議使用，它有陷阱而且容易出錯，一般也很少看到有開發者用這種語法。在本章中就不詳細說明了。

儲存多種資料類型

雖然 JavaScript 並沒有規定說，你只能在同一個陣列中使用單一種資料類型。但是，在陣列中儲存多種不同的資料類型，絕對是個壞主意。在包含有大量資料的陣列中會嚴重影響處理效能，例如像下面這樣的例子。所以一定是要存放多種資料類型，還不如先直接都用字串類型，需要取值時再作轉換。

```
var arr = [1, '1', undefined, true, 'true']
```

另外你需要注意的是，雖然數字類型在 JavaScript 中並沒有分浮點數或整數，但實際上在瀏覽器的 JavaScript 引擎(例如 Google Chrome 的 V8 引擎)中，整數的陣列的處理效率高於浮點數的陣列，可見其實引擎可以分辨各種不同的資料類型，然後會作最有效的儲存與運算，比你想像中聰明得很。

在 ES6 後加入了一種新式的進階資料結構，稱為型別陣列(Typed Arrays)，它是類似陣列的物件，但並非一般的陣列，也沒有大部份的陣列方法。這種資料結構是儲存特定的資料時使用的，主要是為了更有效率的處理二進位的資料(raw binary data)，例如檔案、圖片、聲音與影像等等。不過，就像上面一段說明的，聰明的 JavaScript 引擎在執行時會認得在一般陣列中儲存的資料類型，然後作最有效率的運算處理，在某些情況型別陣列(Typed Arrays)在運算上仍然不見得會比一般陣列還有效率。

陣列的其它部份

陣列還有許多屬性、方法、判斷等等運算，都需要一一學習了解。因為篇幅的關係，如果你想了解更多可以參考我寫的電子書的陣列這一章節。不過你並不需要整個背起來，只是常用的幾種語法和運算你要大概知道，等之後有用到可以再回頭來查詢。

- 陣列 · 從 ES6 開始的 JavaScript 學習生活

物件

物件(Object)類型是電腦程式的一種資料類型，用抽象化概念來比喻為人類現實世界中的物體。

在 JavaScript 中，除了原始的資料類型例如數字、字串、布林等等之外，所有的資料類型都是物件。不過，JavaScript 的物件與其他目前流行的物件導向程式語言的設計有明顯的不同，它一開始是使用原型基礎(prototype-based)的設計，而其他的物件導向程式語言，大部份都是使用類別基礎(class-based)的設計。

在 ES6 之後加入了類別為基礎的語法(是原型基礎的語法糖)，JavaScript 仍然是原型基礎，但可以用類別語法建立物件與繼承之用，雖然目前來說，仍然是很基本的類別語法，但讓開發者多了另一種選擇。

物件在 JavaScript 語言中可分為兩種應用層面來看：

- 主要用於資料的描述，它扮演類似關連陣列的資料結構，儲存"鍵-值"的成對資料。很常見到用陣列中包含物件資料來代表複數的資料集合。
- 主要用於物件導向的程式設計，可以設計出各種的物件，其中包含各種方法，就像已經介紹過的各種包裝物件，例如字串、陣列等等的包裝物件。

物件類型使用以屬性與方法為主要組成部份，這兩種合稱為物件成員(member)：

- 屬性：物件的基本可被描述的量化資料。例如水果這個物件，有顏色、產地、大小、重量、甜度等等屬性。
- 方法：物件的可被反應的動作或行為。例如車子這個物件，它的行為有加速、煞車、轉彎、打方向燈等等的行為或可作的動作。

物件定義方式

物件字面(Object Literals)用於資料描述的物件定義，使用花括號(curly braces)({})作為區塊宣告，其中加入無關順序的"鍵-值"成對值，屬性的值可以是任何合法的值，可以包含陣列、函式或其他物件。

而在物件定義中的"鍵-值"，如果是一般的值的情況，稱為"屬性(property, prop)"，如果是一個函式，稱之為"方法(method)"。屬性與方法我們通常合稱為物件中的成員(member)。

註：屬性名稱(鍵)中也不要使用保留字，請使用合法的變數名稱

```
const emptyObject = {}

const player = {
  fullName: 'Inori',
  age: 16,
  gender: 'girl',
  hairColor: 'pink',
}
```

註：在物件字面定義區塊中的最後一個屬性或方法，後面加上逗號(,)是一種撰寫的風格，目的是為了要統一每一行的寫法而已，這個逗號(,)可寫可不寫。

以如果你已經對陣列有一些理解的基礎下，物件的情況相當類似，首先在定義與獲取值上：

```
const aArray = []
const aObject = {}
```

```

const bArray = ['foo', 'bar']
const bObject = {
  firstKey: 'foo',
  secondKey: 'bar',
}

bArray[2] = 'yes'
bObject.thirdKey = 'yes'

console.log(bArray[2]) //yes
console.log(bObject.thirdKey) //yes

```

不過，對於陣列的有順序索引值，而且只有索引值的情況，我們會更加關心物件中"鍵"的存在，物件中的成員(屬性與方法)，都是使用物件加上點(.)符號來存取。上面的程式碼雖然在 thirdKey 不存在時，會自動進行擴充，但通常物件的定義是在使用前就會定義好的，總是要處於可預測情況下是比較好的作法。物件的擴充是經常使用在對現有的 JavaScript 語言內建物件，或是函式庫的擴充之用。

心得口訣: 對於初學者要記憶是用花括號({})來定義物件，而方括號([])來定義陣列，可以用口訣來快速記憶: 物(霧)裡看花。方陣快炮。

註: 存取物件中的成員(屬性或方法)，使用的是句點(.)符號，這已經在書中的很多內建方法的使用時都有用到，相信你應該不陌生。

註: 相較於陣列中不建議使用的new Array()語法，也有 new Object()的語法，也是不需要使用它。

註：花括號({})也是有符號共同的情況，它除了用於物件的字面定義外，也是"區塊敘述"的符號，例如函式、if...else、for 等等都會用到這個符號。這個符號共用也很容易造成陷阱，要特別注意。

物件的其它部份

物件還有許多屬性、方法、判斷等等運算，都需要一一學習了解。因為篇幅的關係，如果你想了解更多可以參考我寫的電子書的物件這一章節:

- [物件 · 從 ES6 開始的 JavaScript 學習生活](#)

空值與未定義

"空值"在所有的程式語言中都有這種設計，主要是針對尚未確定的、目前還沒有但未來可能有的一種值。有一些使用的場合是在於某一段程式碼時，這個值需要在程式執行過程中從外部資源得到，所以預先定義為空值或是未定義，可以在之後用於判斷是不是有從外部來源得到值。

JavaScript 語言中的設計相當特別，有兩種意思相近的"空值"或"未定義值"，一個是 null 另一個是 undefined。而其它的程式語言通常只會有一種空值的設計。null 與 undefined 有值相等的情況，但它們在 JavaScript 代表的意義不太一樣，這也是初學者很容易搞混的地方。簡單的分述這兩種不同的類型說明如下：

空(null)是空值，代表的是目前這個變數沒有值的意思。

未定義(undefined)是尚未被定義，有可能是根本不存在的意思。

以下用範例來說明:

```
let name //name 未被定義完成，不知道其類型與值，所以是 undefined
```

```
let name = null //name 沒有值，name有存在，但目前不知道裡面的值是什麼
```

特別的是 null 在 ECMAScript 標準中認定它經過 typeof 運算回傳後，會得到'object'也就是物件，這部份有引起一些開發者的爭議，認為原始資料類型應該要回傳自己的類型名稱。實際上根據 JavaScript 的原型物件導向設計，最上層的物件它有一個原型就是 null，也就是"有生於無"的一種概念。目前 ECMAScript 標準並沒有要更動這個部份，當然有一部份原因是一開始就這樣設計了，所以如果現在修改有可能會影響到之前的舊程式。

另一個常見的 null 與 undefined 相互關係，它們兩者的值是相等的，但資料類型不同，由下面的範例可以看到：

```
null == undefined //true
```

```
null === undefined //false
```

經常的用途是，null 會用來當作一種運算後的特殊情況回傳值。而未定義(undefined)則常使用的是加上 typeof 運算符的判斷方式。雖然 JavaScript 並沒有明確規範這兩個資料類型的用途，我們以下面這個原則來使用即可：

對開發者而言，應儘量使用 null 來進行運算或指定值，而 undefined 留給 JavaScript 系統或是函式庫開發者使用。

初學者可以不用太煩惱該如何使用這兩個特殊的資料類型，一開始我們在開發時都只會用到 null 而已，undefined 這東西你就當 JavaScript 語言裡 API 專用的就好。然後你只要記得這兩者要如何判斷就可以了，它們的值相等運算是 true 的。

註：typeof 運算是一種用檢測某個變數或值的資料類型的 API。typeof 運算只能判斷出基礎的資料類型，進階或延伸的資料類型用這運算就沒辦法。而且由物件延伸的資料類型例如 Array、Date 都會判斷為'function'，而不是'object'。不過據說 typeof 的回傳值是 JavaScript 工程師面試時經常出現的考題之一。

第四章：一個簡單的應用程式

我們可以開始來寫一個簡單的應用程式，這個應用程式的目的是為了理解程式撰寫是如何進行的，以及如何開始培養撰寫程式的思考。

猜拳遊戲

從幼稚園時你就已經會這個遊戲，兩個人猜拳，只能出剪刀、石頭、布三種，然後規則是 石頭 勝 剪刀、布 勝 石頭、剪刀 勝 布。這麼簡單的遊戲，我們如何要寫一個應用程式，讓電腦與人來猜拳？這是這個章節所要解說的範例。

註：我們所謂的"剪刀、石頭、布"遊戲，在歐美的它的名字是"Rock, Paper, Scissors"，中譯應該是"石頭、紙、剪刀"。歐美這個遊戲據說是從日本在十九世紀傳入的，認為它是個"日本遊戲"，日本的說法是"石、鋏、紙"這個順序。一開始的發源有兩種說法，一種是中國開始有的，另一是在日本開始的。

第一步：操作介面

一般程式開發的流程，使用者的操作介面通常是第一個先作的部份，從操作介面思考使用者該如何操作這個應用程式，進而規劃應用程式的整個流程。JavaScript 通常的操作介面是 HTML，也就是我們常說的網頁，要與網頁作溝通來顯示，或是放上使用者可以互動的操作介面，其中的存取與互動，透過的是 JavaScript 的一些內建的介面，例如你要存取到網頁上的文字是像下面這樣的程式碼範例：

```
<!-- 這是HTML檔案中的一個片段-->
<!-- 下面元素需要加上id，讓js程式碼中可以存取得到-->
<h1 id="result">比賽結果</h1>

// 這是JavaScript程式碼中的一個片段

// 取得id為result中的文字
const result = document.getElementById('result').innerHTML

// 改變id為result中的文字
document.getElementById('result').innerHTML = '比賽結果：使用者勝利'
```

這裡要記住的是，不論你是從 JavaScript 輸出到 HTML 裡，或是要從 HTML 中抓取資料，HTML 上面的資料一定都是字串的資料類型，如果你從 HTML 抓到的是一個要用來作數字或布林值運算的值，到 JavaScript 程式碼中一開始一定要先作轉換資料類型的動作，才能接著進行運算。

接著在與使用者互動操作的部份，JavaScript 也有與這些操作介面互動的 API，一樣是用 id 的方式來進行，這裡會使用"事件"監聽的方式來作，像下面的範例程式碼：

```
<!-- 這是HTML檔案中的一個片段-->
<!-- 下面元素需要加上id，讓js程式碼中可以存取得到-->
<button id="scissors">剪刀</button>
```

```
// 這是JavaScript程式碼中的一個片段

// 取得id為scissors中的元素，加上事件監聽的函式
document.getElementById('scissors').addEventListener(
  'click',
  function() {
    alert('使用者按下了剪刀')
  },
  false
)
```

程式碼中一開始也是用`document.getElementById()`這個方法，先取得元素的參照後，然後加上`addEventListener`這個 API，可以加入事件的監聽者(函式)。這個 `addEventListener` 一共有三個傳入參數，第一個是要監聽哪一類的事件，以這裡的按鈕而言，我們用的是"點按"這個事件，也就是"click"這種。第二個傳入參數則會是一個函式，這個函式中的程式碼代表著，如果使用者點按了這個按鈕後，要作什麼事情，這種函式通常稱之為回調函式(callback)。第三個傳入參數與事件傳播擴散的方式有關，一般情況固定用 `false` 就可以。

當然，如果你在回調函式中要執行的程式碼很多，整個程式碼會看起來太長，有些地方會擠成一團的感覺，你可以試著把程式碼分離一下再套用，讓程式碼看起來比較清楚，像下面這樣的程式碼範例：

```
// 這是JavaScript程式碼中的一個片段
// 要給事件用的回調函式
const callback = function() {
  alert('使用者按下了剪刀')
}

// 取得id為scissors中的元素，加上事件監聽的函式
const scissors = document.getElementById('scissors')
scissors.addEventListener('click', callback, false)
```

把事件監聽要用的回調函式，先獨立寫成一個函式是很常見的作法，通常有很多事件會共用某一個特定的處理函式，要被使用的函式，儘量寫在要使用它的程式碼之前，就像你要使用某個變數前，記得先宣告它一樣。程式碼編寫的基本原則是要看起來清楚、一目瞭然，多寫個幾行，完全不會對你的程式執行造成什麼效能上的影響，最好每行語句都不要太長，符號與文字之間，該空格就要空格，該用什麼符號就用什麼符號，習慣的養成比什麼都重要。另外要提的是，千萬不要一開始就去學那種硬要把一堆程式碼擠在一行的寫法，這種擠在一行的寫法並不"優雅"也不是"簡潔"，要記住我們在寫的是準備要一板一眼執行的程式碼，不是在創作藝術品，過分追求感覺似乎高明的寫法，只是增加更多出錯的機會。

思考點之一：隨機數(亂數)

首先要考慮的是隨機數，很多情況都會用到隨機，像這裡的三種情況，電腦與人猜拳，需要隨機出現三種的其中一個情況，在抽獎、抽卡牌、牌類遊戲、動態的特效等等，都是由電腦產生隨機數，然後再進行接下來的處理。隨機數字是如何產生的，或是要怎麼樣才夠"隨機"這是很專業的一門學問，我們主要只需要知道要怎麼樣取得一個隨機的數字，而且在需要的範圍(例如這裡的三種情況)就可以了。

隨機數(亂數)的產生，通常在程式語言中都會內建這樣的 API，在 JavaScript 語言中是透過一個 Math(數學)物件來產生，Math.random() 會隨機產生一個 0 到 1 的浮點數，它的基本用法如下：

```
Math.floor(Math.random() * 6) + 1
```

上面的語句代表要產生 1 到 6 的整數亂數，所以它的意義如下：

- 1 代表開始數字
- 6 代表可能情況有 6 種

所以如果是以最大值 max，最小值 min 來寫這個語句，這也是產生整數的亂數，會像下面這樣：

```
Math.floor(Math.random() * (max - min + 1)) + min
```

這是一個簡單的"公式"語法，所謂的公式就都這樣用而已。說明白些，你會去深入研究每年要繳交多少所得稅的公式為什麼要這樣計算，或是會去研究肥胖程度的 BMI(身體質量指數) 公式為什麼要這樣寫嗎？一般情況是不需要的，只是需要套用進去這樣得出你要結果而已。我並不是不建議每個程式開發者，都應該發揮追根究底的精神，對於每個 API 或每個公式都要仔細地研究它的來龍去脈，而是這樣作會很浪費時間，而且你也不需要這樣作，以 JavaScript 語言來說，內建的 API 有數百上千個，加上各種常用的工具函式庫，API 可能有幾千幾萬個，這些不可能都研究得透徹或學得完。除非是對基礎的演算法或結構興趣，才會去研究這些公式或 API 怎麼寫，而且也只會針對某些特別的領域而已，對於初學者而言，你可以把這些東西當成現成的工具，拿來用就可以了。

對於初學者而言，你唯一能學習到哪些 API 會常用到，哪些 API 是必要的，就是要依靠撰寫程式的經驗。光抄書本上或文章上的程式碼是學不熟這些 API 的用法的，在還沒開始上戰場前，新兵要先練習瞄靶、打靶，不然等到真正上戰場時，可能連槍的保險該怎麼打開都有問題。所以要多練習撰寫各種常用的應用程式，從簡單到較為複雜的，從小功能到完整的整個應用，自然就會漸漸熟悉各種常用的 API 或是公式語法。

思考點之二：現實事物的抽象化

從隨機數的出發點，第二個要考量的問題，就是"抽象化"，抽象化是一種把現實的事物，用程式語言的數字、字串之類的資料類型來代表。我們有了隨機數，因為猜拳遊戲只有三種情況，電腦要隨機出一種，所以用上一段的公式要寫成像下面這樣：

```
Math.floor(Math.random() * 3) + 1
```

也就是說，電腦隨機產生 1~3，這三個整數中的其中一個。現在問題來了，"123"要分別代表什麼？"剪刀、石頭、布"對吧，所以我們要定義 1 是什麼 2 又是什麼，這就是一個抽象化的過程，講明白些，程式根本不知道什麼"剪刀、石頭、布"，它只認得它所知道的資料類型。寫一些簡單的程式碼，然後讓它可以在瀏覽器的主控台，或是 Node 的直譯器中執行看看，像下面這樣的範例：

```
// 產生隨機數，剪刀=1、石頭=2、布=3
const randomNumber = Math.floor(Math.random() * 3) + 1

if (randomNumber === 1) {
```

```

        console.log('剪刀')
    }

if (randomNumber === 2) {
    console.log('石頭')
}

if (randomNumber === 3) {
    console.log('布')
}

```

再來，還有一個地方也需要抽象化的，就是電腦因為也認不得你出了什麼，所以它也判斷不出來是誰勝利誰輸了，這個部份你也需要把判斷的規則寫出來。在程式語言中能夠用來判斷誰大誰小的，會用數字類型來判斷比較，我們這個簡單應用中的判斷還算容易，因為目前"剪刀=1、石頭=2、布=3"，只要後面的大於前面就算後面的勝利，例如"布是3，石頭是2，3大於2"這樣，兩者相同則是平手。唯一會出現問題的是，"剪刀要贏過布"這個規則，這有很多種寫法，你也可以想一下怎麼寫會比較好。下面是程式碼範例：

```

// 這個是用來測試用的，使用者出的拳
// 剪刀=1、石頭=2、布=3
const userNumber = 1

// 比賽情況，0=平手 1=使用者贏 -1=電腦贏
let state = 0

// 產生隨機數，剪刀=1、石頭=2、布=3
const randomNumber = Math.floor(Math.random() * 3) + 1

// 使用者勝利的情況 3>2，2>1
if (userNumber > randomNumber) {
    state = 1
}

// 電腦勝利的情況 2<3，1<2
if (userNumber < randomNumber) {
    state = -1
}

// 電腦勝利的 1>3
if (userNumber === 3 && randomNumber === 1) {
    state = -1
}

// 使用者勝利的情況 3<1
if (userNumber === 1 && randomNumber === 3) {
    state = 1
}

if (randomNumber === 1) {
    console.log('剪刀')
}

if (randomNumber === 2) {

```

```

    console.log('石頭')
}

if (randomNumber === 3) {
    console.log('布')
}

console.log(state)

```

從這個程式碼中，你可以手動來改變userNumber的值，模擬使用者選擇了"剪刀=1、石頭=2、布=3"三種情況，然後電腦隨機產生一種情況，看是不是能夠正確地判斷出誰最後贏了這次的猜拳。這種小型的功能測試方式，尤其是在某些判斷的控制流程時，會經常需要先作測試，以免功能如果套用到整個應用中，會出現判斷錯誤的問題。

當然，你也可以把判斷的部份合併一下，讓程式碼更為簡短，這次我們用使用者的代表數字和電腦的代表數字來作相減，讓程式碼簡短些，像下面的範例這樣：

```

// 這個是用來測試用的，使用者出的拳
// 剪刀=1、石頭=2、布=3
const userNumber = 1

// 比賽情況，0=平手 1=使用者贏 -1=電腦贏
let state = 0

// 產生隨機數，剪刀=1、石頭=2、布=3
const randomNumber = Math.floor(Math.random() * 3) + 1

const subNumber = userNumber - randomNumber

// 使用者勝利的情況 3-2, 2-1, 1-3
if (subNumber === 1 || subNumber === -2) {
    state = 1
}

// 電腦勝利的情況 2-3, 1-2, 3-1
if (subNumber === -1 || subNumber === 2) {
    state = -1
}

// 下面輸出電腦出拳的情況
if (randomNumber === 1) {
    console.log('剪刀')
}

if (randomNumber === 2) {
    console.log('石頭')
}

if (randomNumber === 3) {
    console.log('布')
}

```

```
// 輸出最後的結果情況  
console.log(state)
```

註：我有一些撰寫程式碼的習慣，所以範例看起來和一般的教學文章或書籍不同。首先我不會使用分號();作語句的結尾，所以每個語句會寫得短而簡單。其次，我不會用太長的判斷敘述，像 if...elseif...else...這種語句，或是 if 裡面還有 if 的巢狀寫法，我會建議少用，如果能夠只用 if 是最好的，判斷語句是撰寫程式時很容易出錯的地方之一，寫得愈清楚過好。再者，優先使用 const 來宣告你的變數(常數)，除非這個變數你確定之後會因為程式碼執行有可能會變動，才會使用 let 來宣告，但記得一定也要給一個初始化的值。最後，我仍然要鼓勵初學者學習，用英語來寫程式碼註解，理由在最前面的文章中有提到了，因為這裡是教學文章的範例，所以我會用中文來寫每一行程式碼語句的註解，真正在實務中我會用英語來寫註解。

思考點之三：印出結果，從抽象到現實

因為我們最後要顯示結果出來，當然程式中的代表數字，使用者不清楚這些數字的意義是如何，所以需要一個轉換的過程，最後的結果需要以使用者的角度來考量，所以是一個"從抽象到現實"的過程，這種集合式的資料，使用陣列的資料類型是最理想的，例如下面的改寫範例：

```
// 這個是用來測試用的，使用者出的拳  
// 剪刀=1、石頭=2、布=3  
const userNumber = 1

// 比賽情況，0=平手 1=使用者贏 -1=電腦贏
let state = 0

// 數字代表的意義
const textArray = ['剪刀', '石頭', '布']

// 結果數字代表的意義
const resultArray = ['電腦贏', '平手', '使用者贏']

// 產生隨機數，剪刀=1、石頭=2、布=3
const randomNumber = Math.floor(Math.random() * 3) + 1

const subNumber = userNumber - randomNumber

// 使用者勝利的情況 3-2, 2-1, 1-3
if (subNumber === 1 || subNumber === -2) {
    state = 1
}

// 電腦勝利的情況 2-3, 1-2, 3-1
if (subNumber === -1 || subNumber === 2) {
    state = -1
}

// 輸出最後的結果情況
console.log('使用者出了：', textArray[userNumber - 1])
console.log('電腦出了：', textArray[randomNumber - 1])
console.log('最後結果：', resultArray[state + 1])
```

思考點之四：與操作介面整合

最後，我們把整個應用程式的操作介面完成，首先我們在網頁上放上三個按鈕，與幾行文字的地方，分別給上 id 值，讓 JavaScript 程式碼可以存取或改變它們，像下面這一段程式碼：

```
<!-- 這是HTML檔案中的一個片段-->
<!-- 下面元素需要加上id，讓js程式碼中可以存取得到-->

<button id="scissors">剪刀</button>
<button id="rock">石頭</button>
<button id="paper">布</button>

<h1 id="user">使用者玩家的出拳</h1>
<h1 id="computer">電腦玩家的出拳</h1>
<h1 id="result">勝負的結果</h1>

// // 這是JavaScript程式碼
function playGame(value) {
    // 剪刀=1、石頭=2、布=3
    const userNumber = value

    // 比賽情況，0=平手 1=使用者贏 -1=電腦贏
    let state = 0

    // 數字代表的意義
    const textArray = ['剪刀', '石頭', '布']

    // 結果數字代表的意義
    const resultArray = ['電腦贏', '平手', '使用者贏']

    // 產生隨機數，剪刀=1、石頭=2、布=3
    const randomNumber = Math.floor(Math.random() * 3) + 1

    const subNumber = userNumber - randomNumber

    // 使用者勝利的情況 3-2, 2-1, 1-3
    if (subNumber === 1 || subNumber === -2) {
        state = 1
    }

    // 電腦勝利的情況 2-3, 1-2, 3-1
    if (subNumber === -1 || subNumber === 2) {
        state = -1
    }

    // 輸出最後的結果情況
    let user = document.getElementById('user')
    user.innerHTML = '使用者出了：' + textArray[userNumber - 1]

    let computer = document.getElementById('computer')
    computer.innerHTML = '電腦出了：' + textArray[randomNumber - 1]
```

```

let result = document.getElementById('result')
result.innerHTML = '最後結果：' + resultArray[state + 1]
}

// 取得id為scissors中的元素，加上事件監聽的函式
const scissors = document.getElementById('scissors')
scissors.addEventListener(
  'click',
  function() {
    playGame(1)
  },
  false
)

const rock = document.getElementById('rock')
rock.addEventListener(
  'click',
  function() {
    playGame(2)
  },
  false
)

const paper = document.getElementById('paper')
paper.addEventListener(
  'click',
  function() {
    playGame(3)
  },
  false
)

```

程式碼是組合從之前第一步到現在的完整程式碼，原先的測試用的比賽判斷結果，我們把它組合變成一個叫 playGame 的函式之中，每次只要使用者按下按鈕後，就會被重覆地執行一次。playGame 函式有一個傳入值，就是使用者出的拳的情況，分別依照"剪刀=1、石頭=2、布=3"的方式來傳入，在這裡面電腦會產生一個隨機數字，然後判斷是誰勝誰負，把結果輸出在網頁上。

按下按鈕時的事件處理，這裡因為每次按下要處理的 playGame 傳入的參數值都不一樣，所以把它先用另一個函式包起來，當作準備要這樣處理的"延後運行"用的函式。你一定要記得，事件的回調函式，是使用者要先按下按鈕後，才要執行的函式，所以要給它一些準備要執行的程式碼，而不是直接給它函式執行的結果。當然現在這個部份看起來有些重複使用到的程式碼，它可以再用一些簡化的語法讓它看起來更清楚、更簡短，不過現在我們只需要先這樣寫就可以了。

實際運行這個程式

你可以使用第 1 章中所建立的開發測試環境來測試你所寫的程式，如果需要到真正的伺服器上測試，可以先打包(build)後，將 build 目錄裡的檔案上傳到伺服器中。

作業

"黑白配男生女生配"，這個遊戲是猜拳遊戲的進階版了，它需要先猜過拳之後，由勝利的一方指定一個方向，看能不能猜中輸的一方頭轉的方向。我想大概大家都有玩過這個遊戲，遊戲的規則如下：

兩人猜石頭剪刀布，同時嘴裡喊著"黑白猜"，然後猜贏的一個用手指指向上、下、左、右任何一個方向，嘴裡喊著"男生女生配"，猜拳輸的一方則可以選擇抬頭、低頭、頭轉左或右，只要頭的方向跟手指的方向一致，那猜拳輸的一方就輸了，不一致的話那就重新開始猜拳。

這個應用程式就是要讓使用者可以和電腦玩這個遊戲，至於要怎麼寫就靠你來思考了。並沒有什麼正確解答，寫得出來能運作正常就可以了。