darreninfo.cc

# 網路爬蟲

使用 Node.js

授課講師：楊德倫

授課時間：30 小時

# 目錄

【基礎篇】 1
Node.js 介紹與環境安裝 1
Visual Studio Code 介紹與安裝 1
Node Version Manager 介紹與安裝 7
XAMPP 介紹與安裝 13
youtube-dl 介紹與安裝 23
ffmpeg 介紹與安裝 25
前置作業 27
課程成果展示 30
Node.js 介紹 32
資料類別與變數 34
數字與運算子 34
變數 36
字串 43
布林 47
運算子先後順序 49
undefined & null 52
案例討論 53
條件與迴圈 54
if 陳述句、if ... else 陳述句、if...else 陳述句鏈 54
switch 判斷 56
while 迴圈、for 迴圈 57
案例討論 63
函式 64
基本結構 64
建立函式 66
呼叫函式 66
參數傳遞 66
回傳值 67
回呼函式 67
遞迴函式 69

案例討論 ..... 71
陣列與物件 ..... 71
陣列 ..... 71
物件 ..... 80
【進階篇】 ..... 94
HTML & CSS 基礎知識 ..... 94
HTML 簡介 ..... 94
CSS 簡介 ..... 102
案例討論 ..... 104
非同步特性與套件管理介紹 ..... 105
同步 & 非同步 ..... 105
非同步的概念 ..... 106
套件管理工具介紹與使用 ..... 117
案例討論 ..... 120
網頁元素走訪、分析與擷取 ..... 121
靜態網頁元素走訪展示 ..... 121
正規表達式介紹與用法 ..... 130
自動測試工具（網頁瀏覽器自動化工具）介紹 ..... 136
Nightmare 套件基礎教學 ..... 137
動態網頁元素走訪展示 – 使用 Nightmare 套件 ..... 140
關聯式資料庫基礎應用 ..... 152
資料庫種類介紹 ..... 152
XAMPP 安裝 ..... 154
快速導覽 ..... 172
資料庫設計議題 ..... 179
建立資料庫、資料表 ..... 185
新增、修改、刪除資料 ..... 195
資料庫基本查詢 ..... 198
進階查詢 ..... 214
使用 MySQL 內建函式 ..... 221
CRUD 基礎操作 ..... 224
使用 mysql 套件來存取資料庫 ..... 225

# 【基礎篇】

# Node.js 介紹與環境安裝

## Visual Studio Code 介紹與安裝

Visual Studio Code（vs code）是一款程式碼編輯器，由 Microsoft 開發，提供許多好用的擴充功能，還能跨平台使用。

網址：[https://code.visualstudio.com/](https://code.visualstudio.com/)

（圖）Visual Studio Code 的官方網站

（圖）可以跨平台使用

（圖）等待下載時的畫面

（圖）下載後的檔案圖示，本例為 1.35.0 版

（圖）按下一步

（圖）同意協議後，按下一步

（圖）確認空間足夠後，按下一步

（圖）使用預設設定，按下一步

（圖）勾選建立桌面圖示，按下一步

（圖）確認設定後，開始安裝

（圖）安裝中

（圖）按下完成，準備啟動編輯器

| 補充說明 |
| --- |

一開始 vs code 需要 Open Folder，設定放置專案的位置，若是沒有特別要求

Windows 可以設定成以下路徑（若沒有路徑上的資料夾，可自行新增）：
C:\Users\{你的使用者帳號}\source\repos\{專案資料夾}
或是
D:\{專案資料夾}

Linux 或 Mac 使用者，可以設定成：
/home/{你的使用者帳號}/{專案資料夾}

原則上，上述路徑通常都是使用者帳號權限最大的地方，操作起來比較不需要額外考量權限問題；若是沒有資料夾，可以自行手動建立。

教學上，我個人是設定成下面的路徑：
C:\Users\telun\source\repos\Nodejs-Html-Parser

（圖）設定完專案資料夾，將會出現上方的畫面

## Node Version Manager 介紹與安裝

Node Version Manager（NVM）一款管理 Node.js 版本的工具，可以安裝不同版本，依需求切換。安裝時附帶安裝 Node Package Manager（NPM），用來管理 Node.js 所使用的套件等。安裝流程，以 Windows 版本為例。

Windows 版：https://github.com/coreybutler/nvm-windows

Linux / Mac 版：https://github.com/nvm-sh/nvm

（圖）按下 Download Now

（圖）本課程使用 1.1.7 版，下載 nvm-setup.zip

（圖）壓縮檔解開後的執行檔

（圖）同意協議後，按下一步

（圖）沒有特別設定的話，按下一步

（圖）建立類似軟連結的路徑，沒有特別設定，按下一步

（圖）確認設定後，開始安裝

（圖）安裝結束後，按下完成

（圖）讓我們在終端機下，輸入 nvm，若出現上方畫面，代表環境建立成功

```
PS C:\Users\telun\source\repos\Nodejs-Html-Parser> nvm install 10.16.0
Downloading node.js version 10.16.0 (64-bit)...
Complete
Creating C:\Users\telun\AppData\Roaming\nvm\temp

Downloading npm version 6.9.0... Complete
Installing npm v6.9.0...

Installation complete. If you want to use this version, type

nvm use 10.16.0
PS C:\Users\telun\source\repos\Nodejs-Html-Parser>
```

（圖）安裝 Node.js，請參考 LTS 版本，本例使用 10.16.0，會連同 npm 一起安裝

```
PS C:\Users\telun\source\repos\Nodejs-Html-Parser> nvm use 10.16.0
Now using node v10.16.0 (64-bit)
PS C:\Users\telun\source\repos\Nodejs-Html-Parser> node -v
v10.16.0
PS C:\Users\telun\source\repos\Nodejs-Html-Parser>
```

（圖）輸入 nvm use 10.16.0，切換到該版本，再輸入 node -v 確認環境是否設定完成

| nvm 安裝流程 |
| --- |
| 1. 安裝 nvm（Windows 版或其它作業系統版本皆可；請勿透過系統管理員權限安裝）<br>2. 打開 Visual Studio Code（若已開啟，請先關閉後再打開，讓其存取 nvm 環境變數） |

下列流程，每一個輸入完成後，請按下 Enter 執行：

1. 輸入 nvm （確認是否有 Usage 相關說明）
2. 輸入 nvm install 10.16.3
3. 輸入 nvm use 10.16.3
4. 輸入 node -v （確認 node.js 版本，理應輸出 v10.16.3）
5. 輸入 npm -v（確認套件管理工具版本，理應輸出 6.9.0）

| 補充說明 |
| --- |
| 若是專案壓縮檔有更新的情況，請解壓縮後，透過 Visual Studio Code 開啟專案資料夾，再使用 Ctrl + Shift + ` 來開啟終端機，輸入「npm i --save」來安裝套件。 |

## XAMPP 介紹與安裝

XAMPP 是 Apache、MariaDB、PHP、Perl 等整合在一起的工具，提供 Web、FTP、phpMyAdmin 等服務，Windows、Linux、Mac 環境都能使用。

網址：https://www.apachefriends.org

（圖）可下載不同作業系統版本，也可以切換語系

（圖）等待下載畫面

（圖）下載後的檔案

（圖）不會安裝到 Program Files 路徑下，可以直接按下 OK

Setup
Setup - XAMPP
Welcome to the XAMPP Setup Wizard.
bitnami
< Back
Next >
Cancel

（圖）按下一步

Setup
Select Components
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Server
Apache
MySQL
FileZilla FTP Server
Mercury Mail Server
Tomcat
Program Languages
PHP
Perl
Program Languages
phpMyAdmin
Webalizer
Fake Sendmail
XAMPP Installer
< Back
Next >
Cancel

（圖）除了預設的 Apache、PHP，僅留下 MySQL、phpMyAdmin 後，按下一步

（圖）使用預設路徑，按下一步

（圖）課程中不會用到架站工具，所以取消上方勾選，按下一步

（圖）按下一步，開始安裝

（圖）安裝過程

（圖）勾選上方選項，開啟控制台，按下完成

（圖）選擇美國（英文）後，按儲存

（圖）控制台畫面，請按下 Apache、MySQL 右側的 Start 鍵，啟動服務

（圖）按下 MySQL 右側的 Admin，開啟 phpMyAdmin 網頁

（圖）看到 phpMyAdmin 的畫面，代表安裝成功

（圖）選擇上面的使用者帳號，並按下圖片左下角的連結，新增使用者帳號

| 範例用資料庫帳號、密碼 |
| --- |
| 課堂中所使用的範例，大部分會需要寫入資料庫，我們需要建立一組資料庫的帳號、密碼。 |

帳號：test
密碼：T1st@localhost

（圖）登入資訊

（圖）全域權限，請勾選「全選」；參閱補充說明。

（圖）最後按下網頁右下角的執行，建立新帳號

| 補充說明 |
| --- |
| 資料庫使用者帳號的增修，有很大的學問。課程中，為了教學方便，可以按照前述的流程建立使用者；若是公司有使用到資料庫，帳號管理不能草率，一定要請教有經驗的資料庫管理人員，協助帳號新增與權限設定。 |

## youtube-dl 介紹與安裝

youtube-dl 是一套透過命令列下指令，來下載 YouTube 網站的影片的程式，它不僅能下載 YouTube 網站裡面的影片，還能從其它提供影片播放的網站，進行影音下載，最重要的是，它還能跨平台。（網址連結：https://ytdl-org.github.io/youtube-dl/index.html）

（圖）youtube-dl 首頁

**youtube-dl** **Download Page**

Remember *youtube-dl* requires Python version 2.6, 2.7, or 3.2+ to work except for Windows exe.

Windows exe requires Microsoft Visual C++ 2010 Redistributable Package (x86) and does not require Python that is already embedded into the binary.

**2019.07.27 (sig)**

**SHA256**: a1074aa3b33aad816c5d8b15b414f1ce1e286291f9af9c54716575ba99a89f4b

Windows exe (sig - SHA256 4209abd7e284aea928a1f58be0b58c781b542d576e2481398ba4a28bc0f06a36)
Full source + docs + binary tarball (sig - SHA256 cdfc691ca9dcc63addb52dbbadb1b9e86325dd2d53d38ac844d504f482468bcc)

To install it right away for all UNIX users (Linux, OS X, etc.), type:

```
sudo curl -L https://yt-dl.org/downloads/latest/youtube-dl -o /usr/local/bin/youtube-dl
sudo chmod a+rx /usr/local/bin/youtube-dl
```

If you do not have curl, you can alternatively use a recent wget:

```
sudo wget https://yt-dl.org/downloads/latest/youtube-dl -O /usr/local/bin/youtube-dl
sudo chmod a+rx /usr/local/bin/youtube-dl
```

You can also use pip:

```
sudo pip install --upgrade youtube_dl
```

This command will update youtube-dl if you have already installed it. See the pypi page for more information.

You can use Homebrew if you have it:

```
brew install youtube-dl
```

To check the signature, type:

```
sudo wget https://yt-dl.org/downloads/latest/youtube-dl.sig -O youtube-dl.sig
gpg --verify youtube-dl.sig /usr/local/bin/youtube-dl
rm youtube-dl.sig
```

（圖）下載頁面

**youtube-dl** **Download Page**

Remember *youtube-dl* requires Python version 2.6, 2.7, or 3.2+ to work except for Windows exe.

Windows exe requires Microsoft Visual C++ 2010 Redistributable Package (x86) and does not require Python that is already embedded into the binary.

**2019.07.27 (sig)**

（圖）以 Windows 為例，按下連結後，自動下載

- 按下 [Windows exe] 連結，可以下載到 C:\Users\users\Documents\tools\video\bin，若無路徑上的資料夾，請個別手動建立。
- 然後到電腦→內容→進階系統設定→環境變數→系統變數下方的列表中，有個「Path」，按下編輯，把「C:\Users\users\Documents\tools\video\bin」新增上去，之後一直按確定就可以了。
- 若是 Windows 7 的話，需要按下編輯，然後在文字的最後面，加上「;」，之後一直按確定即可。
- 打開命令列（command-line），輸入「youtube-dl.exe --version」來確認版本，輸出日期文字，代表環境安裝成功。

（圖）使用命令列，輸入「youtube-dl.exe --version」

## ffmpeg 介紹與安裝

ffmpeg 是一款用來錄音、轉檔與提供影音串流的跨平台解決方案。它是一個多媒體框架，能夠解碼、編碼、轉碼、多工、分離、串流、過濾、播放「人類與機器已經創造出來的任何東西」。它支援鮮為人知、直到最新的檔案格式。（網站連結：https://ffmpeg.org）

（圖）ffmpeg 首頁

（圖）下載頁面

（圖）以 Windows 為例，選擇中間的微軟圖示後，按下左下「Builds」連結

（圖）Windows 版的下載頁面，按下右下角的「Download Build」來進行下載

- 按下 [Download Build] 連結，可以下載到 C:\Users\users\Documents\tools，若無資料夾，請手動建立，最後將資料夾命名為「ffmpeg」。（此時路徑應該是：C:\Users\users\Documents\tools**ffmpeg**）
- 然後到電腦→內容→進階系統設定→環境變數→系統變數下方的列表中，有個「Path」，按下編輯，把「C:\Users\users\Documents\tools\ffmpeg\bin」新增上去，之後一直按確定就可以了。
- 若是 Windows 7 的話，需要按下編輯，然後在文字的最後面，加上「;」，之後一直按確定即可。
- 打開命令列（command-line），輸入「ffmpeg」來確認版本，輸出相關資訊，代表環境安裝成功。

（圖）輸入 ffmpeg，會出現使用的函式庫以及啟動元件參數

## 前置作業

請先下載專案資料夾或是壓縮檔（GitHub 連結：https://bit.ly/2mgVhPo），解壓縮後，資料夾修改名稱為「Nodejs-Html-Parser」，並放置登入帳號擁有增刪修等權限的地方，作為專案資料夾。以下以 Windows 環境為例，路徑為「**C:\Users\{你的使用者名稱}\source\repos\Nodejs-Html-Parser**」（若是 Linux 環境，可以放在 /home/{你的使用者帳號}/Nodejs-Html-Parser）；若有 D 槽，也可以直接放在該目錄下，例如 **D:\Nodejs-Html-Parser**。

（圖）資料夾建立範例

（圖）開啟 vs code，資源管理器（Explorer）→ 中間的開啟資料夾（Open Folder）

（圖）開啟先前下載好的資料夾，會看到圖中的畫面

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\telun\source\repos\Nodejs-Html-Parser> npm i --save

> electron@2.0.18 postinstall C:\Users\telun\source\repos\Nodejs-Html-Parser\node_modules\electron
> node install.js

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN nodejs-html-parser@1.0.0 No repository field.

added 220 packages from 236 contributors and audited 425 packages in 11.096s
found 0 vulnerabilities

PS C:\Users\telun\source\repos\Nodejs-Html-Parser>
```

（圖）輸入「npm i --save」，會自動依 package.json 的 dependency 來安裝套件

（圖）專案資料夾會新增 package-lock.json 和 node_modules 資料夾

# 課程成果展示

原則上，我們會在課堂上，帶領同學完成下面的成果，然而一切還是要依課堂進度、上課情形而定：

| 名稱 | 預覽圖片與連結 | 說明 |
|---|---|---|
| Wine Searcher HTML parser | <br>https://youtu.be/**SHmqWvO_ziw** | 使用 cURL 指令來取得特定酒的歷史價格，例如 2017-06-01 到 2019-05-01，每個月的紅酒價格。未付費版本，可以看到 2 年內的資料，專業版則可以看到 5 年內的資料。 |
| LINE stickers HTML parser | <br>https://youtu.be/**SMwI2i5xkck** | 使用 cURL 指令來取得 LINE 官網的「靜態」與「動畫語音」貼圖的 HTML 元素，同時解析元素內容。 |
| YouTube HTML parser | <br>https://youtu.be/**885WDQjq7Vw** | 使用網頁瀏覽器自動化工具來取得 YouTube 的搜尋結果。搜尋後，我們讓瀏覽器自行滾動，滾動同時擷取動態產生的 HTML 元素。 |
| Instagram HTML parser | <br>https://youtu.be/**wILsmJz25d4** | 使用網頁瀏覽器自動化工具來解析 Instagram 的 HTML 元素，無論是單張圖片、滑動的多張圖片、單支影片、滑動的多支影片。 |
| 動態網頁元素擷取 - 以 104 人力銀行為例 | <br>https://youtu.be/**Qafruw53e8M** | 使用網頁瀏覽器自動化工具，來解析 104 人力銀行的 HTML，透過自動輸入關鍵字、點選地點、設定全職或兼職等自動化操作行為，取得職缺列表。 |

| | | |
|---|---|---|
| Google translation 小姐幫我代言啦！（誤） | <br>https://youtu.be/**KkACYsrbNt4** | 不串 Google Translation API，直接用正規表達式擷取字幕檔案，並將文字結合語音原址，取得 google translation 小姐的聲音檔，最後再與剪輯影片、上字幕的軟體結合。 |
| 歡迎收看臺大計中狂新聞（誤） | <br>https://youtu.be/**Qafruw53e8M** | 原理同上。我們將 google translation 小姐的聲音加速 1.5 倍，同時將影像與字幕結合，建立一個示範小品。 |
| Web browser automation for crawling CLAD data with pagination | <br>https://youtu.be/**ay81kZ-phMY** | 使用 selenium 去爬取「Corpus-derived Chinese Lexical Association Dictionary (CLAD)」的表格資料，預設有 84,700 筆衍生詞資訊。 |
| Automatic downloading WoS records | <br>https://youtu.be/**jQJagK8Fr28** | （僅限臺大網路使用）使用 selenium 來進行「網路瀏覽器自動化」，模仿人類操作網頁的方式，進行 Journal Citation Reports 資料儲存，減少大量人力手動操作。 |

# Node.js 介紹

Node.js 是一個能夠在伺服端（Server）執行 Javascript 的環境，讓以往使用 Javascript 撰寫前端程式的程式開發人員，也能建立自己的後端服務。Node.js 擁有 Javascript 的特性，使用事件驅動程式設計，以及擁有非阻塞、非同步輸入輸出、使用單執行緒等特性。

| 名詞 | 說明 |
| --- | --- |
| 事件驅動（Event-driven） | 有別於「批次性」程式設計流程（程式碼逐行執行，沒有互動），事件驅動的流程是由使用者行為（例如按下送出鈕、鍵盤輸入、檔案上傳等）或其它互動功能所決定。 |
| 非阻塞輸入輸出（Non-blocking I/O） | 阻塞式的程式運作，需要等待程式 I/O 執行完畢，才會執行另一個程式 I/O；非阻塞式的程式運作，會透過輪詢（polling）方式，不斷確認 I/O 過程是否滿足結束條件，滿足以後，才會結束程式。<br><br>通常在 Network Programming（尤其是 Linux socket 進行文字串接，是透過 Blocking I/O 來運作）的議題裡，討論較多；在 Node.js 裡，為非阻塞式的輸入輸出，通常會在串流（streams）的概念中加以討論。 |
| 非同步輸入輸出（Asynchronous I/O） | 同步 I/O 的程式設計與相關程式語言，通常是逐行執行，在上一行相關程式碼尚未執行完畢前，原則上不會執行下一行的程式碼，例如 C、PHP 等。<br><br>Node.js 具有非同步 I/O 性質，執行過程中，不會等待上一行的程式 I/O 結束，直接往下執行到程式結束，中間執行到非同步的函式（例如 setTimeout），會先放置於事件佇列（Event Queue）中，然後繼續執行下一行，程式結束前，只要非同步函式滿足條件（例如滿足 |

| | |
|---|---|
| | setTimeout 指定的秒數），便開始執行，而後離開佇列。 |
| 單執行緒（Single Thread） | 執行緒多寡，由 CPU 核心數所決定，而 Node.js 在執行時，僅會使用 1 個執行緒。 |

# 資料類別與變數

## 數字與運算子

Javascript 可以進行加（+）、減（-）、乘（*）、除（/）的基本運算，傳統的數學計算概念，也可以在程式當中運作，例如先乘除、後加減等等的概念；關於上述的基本運算，我們稱為「算數運算子」（Arithmetic operators）。以下我們透過簡單的例子，來介紹數字與運算子在。

範例

```
console.log(5566 + 3312);
// 輸出 8878

console.log(11 + 22 + 33 + 44);
// 輸出 110

console.log(7 - 3 + 11);
// 輸出 15

console.log(8 * 3 + 12);
// 輸出 36

console.log(4 * 12 + 12 / 6);
// 輸出 50

console.log(7 - 8 / 2 + 23 * 3);
// 輸出 72

console.log(7 - 8 / (2 + 23) * 3);
// 加上括號，輸出 6.04
```

我們也可以透過變數之間的運算，來建立新的變數

範例

```
let secondsInMinute = 60; //一分鐘 60 秒
let minutesInAnHour = 60; //每小時 60 分鐘
let secondsInAnHour = secondsInMinute * minutesInAnHour;
console.log(secondsInAnHour);
// 輸出 3600 (秒)

let hoursInADay = 24;
let secondsInADay = secondsInAnHour * hoursInADay;
console.log(secondsInADay);
// 輸出 86400
```

從事程式開發與設計，需要常常對數值變數加 1 或減 1，例如累加、累減的計算，我們稱為「遞增」和「遞減」。以下我們透過幾個範例，說明遞增、遞減的操作方式。

範例

```
let count = 0;
count++;
console.log(count);
// 輸出 1

count++;
count++;
console.log(count);
// 輸出 3

count--;
console.log(count);
// 輸出 2

有時候，我們也會將遞增或遞減，寫成
count = count + 5; //加後賦值，count 值為 7
count = count - 3; //減後賦值，count 值為 4
```

| 補充說明 |
| --- |
| let count = 0;<br>console.log(count++);<br>// 輸出 0<br><br>上面的例子，輸出的結果為 0，因為 count++ 在輸出的時候，會先顯示原先的值，再進行遞增，所以輸出為 0，實際的值已經變成 1。<br><br>讓我們看看下面的例子：<br><br>let count = 0;<br>console.log(++count);<br>// 輸出 1<br><br>上面的例子，輸出結果為 1，是因為 ++count 先進行了遞增，而後才將 count 值輸出。<br><br>count++ → 先輸出再加<br>++count → 先加再輸出 |

## 變數

我們可以透過變數名稱的宣告，來為「值」（文字、數值、運算結果、暫存資料等）取個名字。我們使用關鍵字 var 來宣告變數，例如 var myName，而所謂「關鍵字」，是指在程式語言中具有意義的單詞，通常不適合拿來作為變數命名使用。原則上，變數的賦值，通常**由右往左**，我們稱之為「賦值運算子」（Assignment operators）。

| 範例 |
| --- |
| //宣告變數 myName<br>var myName;<br><br>//宣告變數 myName，同時賦值'Darren'<br>var myName = 'Darren'; |

```
//宣告變數 numberOfDays，同時賦值 30
var numberOfDays = 30;
```

變數命名的規則，有很多種方式，例如駱駝命名法、匈牙利命名法等。駱駝命名法是一種慣例，看起來像駱駝的駱峰，變數以小寫字母開頭，除了第一個單詞外，其它單詞的首個字母都大寫，例如 numberOfCandies，或是我們上方的 myName，或是 numberOfDays。

課程中，原則上使用駱駝命名法，但沒有特別強制規定，你也可以自由選擇自己慣用的方式。

| 補充說明 |
| --- |
| Javascript 透過 var 關鍵字宣告的變數，具有抬升（Hoisting）的性質，不易掌握變數的生命週期。近年的 Javascript 版本，增加了 let 與 const 關鍵字，解決了 var 關鍵字的抬升問題（Hoisting）。<br><br>let 與 const，都在程式區塊內有效（{...}，block-scoped），例如在某個函式或某個判斷式裡面宣告常數，若是在區塊外嚐試輸出，則會出現錯誤訊息（因為解決了抬升問題）。在實務案例當中，會頻繁用到 let 與 const 關鍵字。<br><br>const 關鍵字用於「常數」的宣告，有別於我們先前看到的「變數」，原則上不宜重覆賦值，也常用大寫的方法與變數加以區隔，例如 const MAX_LENGTH、DB_HOST、DB_PASSWORD 等。 |

| 補充說明 |
| --- |
| 我們來看看 var 在區域變數中的特性<br><br>一般來說，我們會這樣宣告變數、初始化變數（給變數初始值）、輸出變數內容<br>var name = 'Darren';<br>console.log(name);// 輸出 Darren<br><br>換個方式<br><br>console.log(name);<br>var name = 'Darren'; |

// 這裡會輸出 undefined

若是這樣子呢

```
console.log(name);
let name = 'Darren';
//應該輸出 undefined，卻拋出 ReferenceError: name is not defined 的訊息
```

這是因為 Javascript 在使用 var 宣告變數時，會將變數抬升（hoisting）到作用域（程式區塊）的前面，作用域內外的每一行都有機會使用到。

再看一個例子

```
function getValue(condition){
  if(condition) {
    var value = "Yellow";
    return value;
  } else {
    return null;
  }
}

console.log( getValue(true) );
// 輸出 "Yellow’ "
```

上面的範例裡，var 會因為編譯器的關係，被提升（Hoisting）放到 if (...) 的上一行來進行宣告，讓 else {...} 區塊內也可以使用 value 這個變數。

```
function getValue(){
  var value; //被編譯器放到這裡
  if(condition) {
    value = "Yellow";
    return value;
  } else {
    //原先的概念上不會使用到 value 這個變數，
    //卻因為 hoisting 的關係，讓 else {...} 區塊內，也能使用到 value 變數
```

```
    return null;
  }
}
```

在這個情況下，程式設計人員無法精確地掌控變數的生命週期（宣告、使用、消滅等），於是近年的 Javascript 版本增加了 let 與 const 關鍵字，來強化對變數生命週期的控制。

大家可以試試，以下四個 IIFE（Immediately Invoked Function Expression，是一個定義完馬上就執行的 JavaScript function），各自會出現什麼訊息？

| 使用 var 來宣告 value | 使用 let 來宣告 value |
|---|---|
| (function getValue(condition){<br>if(condition) {<br>var value = "Yellow";<br>console.log(value);<br>} else {<br>console.log(value);<br>}<br>}<br>)(true);<br><br>//輸出 Yellow | (function getValue(condition){<br>if(condition) {<br>let value = "Yellow";<br>console.log(value);<br>} else {<br>console.log(value);<br>}<br>}<br>)(true);<br><br>//輸出 Yellow |
| (function getValue(condition){<br>if(condition) {<br>var value = "Yellow";<br>console.log(value);<br>} else {<br>console.log(value);<br>}<br>}<br>)(false);<br><br>//輸出 Undefined<br>//因為 else {...} 也能存取 value | (function getValue(condition){<br>if(condition) {<br>let value = "Yellow";<br>console.log(value);<br>} else {<br>console.log(value);<br>}<br>}<br>)(false);<br><br>//拋出錯誤訊息，因為 value 在 else {...}<br>//裡面尚未被宣告，可以從這裡控制變數的生命週期 |

範例：

```
var count = 30;
let count = 40; //這裡會拋出錯誤，因為重覆宣告
```

範例：

```
var count = 30;

if(condition) {
```

```
    let count = 40; // 這裡不會拋出錯誤
    // if 區塊內部的 count，會遮住外部的 count，
    // 外部的 count 只有 if 區塊「外面」才能存取得到

    //其它程式碼...
}
```

以上討論完 let 關鍵字，下面讓我們聊聊 const 關鍵字。

const 關鍵字是常數，其值一旦被設定後，**原則上**不可任何更改，每一個常數宣告時，都要初始化（就是一宣告就賦予初始值）：

範例：

```
const maxItems = 30;
const name; //拋出錯誤，常數尚未初始化
```

範例：

```
let condition = true;
if(condition) {
    const maxItems = 5;
}
console.log(maxItems); // 拋出錯誤，因為 if 區塊以外，無法存取 maxItems
```

範例：

```
var message = "Hello Word!";
let age = 25;

// 下面兩行都會拋出錯誤，因為重複宣告
const message = "Good Job!";
const age = 30;
```

雖然常數原則上重新賦值，但是當 const 宣告的變數，是「陣列」或是「物件」，便會產生例外。

範例：

```
const arr = ["Hello", "World"];
```

```
arr.push("!");
console.log(arr);
// 輸出 [ 'Hello', 'World', '!' ]
```

範例：

```
const obj  = {
   name: "Darren Yang",
};
obj.lineId = "telunyang";
obj.website = "https://darreninfo.cc";
console.log(obj);
// 輸出
// { name: 'Darren Yang', lineId: 'telunyang', website: 'https://darreninfo.cc' }
```

簡而言之，上面宣告為常數的陣列、物件變數，在 Javascript 的概念裡，之所以能夠增修，是因為陣列、物件變數擁有「感覺像是」 call by reference 的特性，在新增內部元素或屬性時，都在同一個記憶體位置作業，不會額外佔用記憶體位置，但**實際上，Javascript 用的是 call by sharing，當參數物件修改的是「屬性」，則類似 call by reference；當參數物件修改的是本身的「值」，則類似 call by value。**

| call by value | call by ~~reference~~ → call by sharing |
|---|---|
| let a = 10;<br>let b = 20;<br><br>console.log(a, b);<br>// 輸出 10, 20<br><br>function swap(x, y){<br>  let tmp = x;<br>  x = y;<br>  y = tmp;<br>}<br><br>swap(a, b);<br><br>console.log(a, b); | let objA = {name: 'Darren'};<br>let objB;<br><br>objB = objA;<br><br>console.log(objA, objB);<br>// 輸出 { name: 'Darren' } { name: 'Darren' }<br><br>objA.name = 'Bill';<br><br>console.log(objA, objB);<br>// 輸出 { name: 'Bill' } { name: 'Bill' }<br><br>function setName(objX){ |

| | |
|---|---|
| // 輸出 10, 20 | objX.name = 'Alex'; //類似傳址呼叫<br>// objX = {name: 'Doris'} //類似傳值呼叫<br>}<br><br>setName(objA);<br><br>console.log(objA, objB);<br>// 輸出 { name: 'Alex' } { name: 'Alex' }<br><br>若是把 setName 當中的兩行程式註解互換， objA 跟 objB 還各是什麼？ |
| let a = 30;<br>let b;<br><br>b = a;<br>a = 20;<br><br>console.log(a); // 輸出 20<br>console.log(b); // 輸出 30 | let a = {greeting: "Hi"};<br>let b;<br><br>b = a;<br>b.greeting = "Hello World!";<br><br>console.log(a); // 輸出{ greeting: 'Hello World!' }<br>console.log(b); // 輸出{ greeting: 'Hello World!' }<br><br>b = {greeting: 'Hello!'};<br><br>console.log(a); // 輸出 { greeting: 'Hello World!' }<br>console.log(b); // 輸出 { greeting: 'Hello!' } |

| call by value | call by reference |
|---|---|
| 布林值、字串、數值、空值（null）、未定義（undefined） | 物件、陣列、函式 |

| | |
|---|---|

關於賦值運算子的種類，常見的有以下幾種，提供給大家參考：

| 名稱 | 簡化後運算子 | 說明 |
|---|---|---|
| 賦值 | x = y | x = y |
| 加法賦值 | x += y | x = x + y |
| 減法賦值 | x -= y | x = x - y |
| 乘法賦值 | x *= y | x = x * y |
| 除法賦值 | x /= y | x = x / y |
| 餘數賦值 | x %= y | x = x % y |
| 指數賦值 | x **= y | x = x ** y |
| 左移賦值 | x <<= y | x = x << y |
| 右移賦值 | x >>= y | x = x >> y |
| 無號右移賦值 | x >>>= y | x = x >>> y |
| 位元 AND 賦值 | x &= y | x = x & y |
| 位元 XOR 賦值 | x ^= y | x = x ^ y |
| 位元 OR 賦值 | x \|= y | x = x \| y |

## 字串

Javascript 中的字串有序列的概念，可以包括文字、數字、標點與空格等，例如「Hello World!」。字串與相關處理，非常重要，我們下面使用一些具體的範例，供大家參考。

範例

```
let myStr = "Hello World!";
console.log(myStr);
// 輸出 Hello World!

let myStr01 = "Hello";
let myStr02 = " "; //裡面是空格
let myStr03 = "World";
```

```
let myStr04 = "!";
let myStr05 = myStr01 + myStr02 + myStr03 + myStr04;
console.log(myStr05);
// 輸出 Hello World!

let fname = 'Darren';
let lname = 'Yang';
console.log(fname + ' ' + lname);
// 輸出 Darren Yang

let numberNine = 9;
let stringNine = "9";
console.log(numberNine + numberNine); // 輸出 18
console.log(stringNine + stringNine); // 輸出 99
console.log(numberNine + stringNine); //  輸出 99，數值將會自動轉型，與 "9" 前後串接
```

補充說明

雙引號、單引號所含括的字串，可以連同跳脫字元的效果與特性，一同輸出。以字串「When you say nothing at all.」為例。

```
console.log("When\tyou\nsay\nnothing\bat all.");
```

輸出結果為

```
When    you
say
nothinat all.
```

要注意的是，字串前後的引號，使用要一致，不能一個單、一個雙，只能單單、雙雙。

```
console.log('Hello World!'); // 正確輸出 Hello World!
console.log("Hello World!'); // 拋出 SyntaxError 語法錯誤的訊息
```

補充說明

檢查字串的長度

```
console.log("Hello World!".length);
// 輸出 12
```

從字串中取得單個字元（字串索引從 0 開始起算）

```
let myName = "Darren";
console.log(myName[0]); // 輸出 D
console.log(myName[2]); // 輸出 r
console.log(myName[5]); // 輸出 n
console.log(myName[6]); // 輸出 undefined
```

截取字串

```
"Darren Yang".slice(0,6);
// 0 是起始位置，6 是結束位置；位置 6 之前的字串才會被取得，故截取 0 到 5 位置的字串
```

| D | a | r | r | e | n | | Y | a | n | g |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

轉換所有字串為大寫

```
console.log( "Darren Yang".toUpperCase() );
// 輸出 DARREN YANG
```

轉換所有字串為小寫

```
console.log( "Darren Yang".toLowerCase() );
// 輸出 darren yang
```

你也可以透過變數來輸出大（小）寫結果

```
let id = 'a123456789';
console.log( id.toUpperCase() );
// 輸出 A123456789
```

補充說明

有關字串合併的方法，除了單/雙引號前面加號（+）來串接，還有提供樣版字面值（Template literals），讓字串合併與變數內嵌字串等問題，變得簡單許多。

使用單/雙引號來合併兩個句子，同時實現句子斷行

```
console.log('你好！我是' + '\n' + 'Darren Yang');
```

有的人會這麼做

```
console.log('你好！我是' +
'\n' +
'Darren Yang');
```

在上述的例子中，可以使用樣版字面值，即是使用「`」符號（一般鍵盤 ESC 下面、數字 1 波浪符號的按鍵），放在字串前後，取代單/雙引號的位子。

```
console.log(`你好！我是\nDarrenYang`);
```

你也可以這麼使用

```
console.log(`你好！我是
Darren Yang`);
```

輸出的結果與上面幾個例子一樣！

還有一個很重要的用法，就是把變數放在樣版字面值當中來顯示

```
let fname = 'Darren';
let lname = 'Yang';
console.log(`你好! 我是 ${fname} ${lname}`);
//輸出 你好! 我是 Darren Yang
```

若是嵌入變數計算，可以這麼做

```
let a = 5;
let b = 10;
console.log(`Fifteen is ${a + b},
not ${2 * a + b}.`);
```

有關字串的進階運用，我們之後將在「正規表達式」的章節中詳細討論。

## 布林

布林提供了 true (真) 與 false (假) 的判斷值，透過邏輯運算子（&&、||、!）來進行操作。

| 範例 |
| --- |
| &&（and 符號）：<br><br>幼兒園學生出門前，檢查是否有鞋子跟背包，兩個東西都很重要；全部為真，結果才為真。<br>let hadShoes = true;<br>let hadBackpack = false;<br>console.log(hadShoes && hadBackpack);<br>//輸出 false，代表還沒準備好出門<br><br>若是把 hadBackpack 的值，改成 true<br>let hadShoes = true;<br>let hadBackpack = true;<br>console.log(hadShoes && hadBackpack);<br>//輸出 true，代表準備好了，可以出門囉！<br><br><br>\|\|（or 符號）：<br><br>幼兒園學生出門前，選擇水果作為飯後甜點，至少帶一樣；一個為真，結果為真。<br>let hasApple = true;<br>let hasBanana = false;<br>console.log(hasApple \|\| hasBanana);<br>// 輸出 true<br><br><br>!（not 符號）：<br><br>假設現在是週末，那麼，我們不需要工作整天；真變假、假變真。<br>let isWeekend = true;<br>let workAllDay = !isWeekend;<br>console.log(workAllDay);<br>// 輸出 false，代表不需要工作整天 |

另一種以相互比較的方式，來呈現布林邏輯運算結果，稱之為「比較運算子」，例如常常提到的「大於 >、大於等於 >=、等於 ==（或 ===）、小於 <、小於等於 <=、不等於 !=」等概念。

範例

判斷身高與身高限制的比較

```
let height = 171;
let heightRestriction = 140;
console.log(height > heightRestriction); // 輸出 true
console.log(height >= heightRestriction); // 輸出 true
console.log(height == heightRestriction); // 輸出 false
console.log(height < heightRestriction); // 輸出 false
console.log(height <= heightRestriction); // 輸出 false
console.log(height != heightRestriction); // 輸出 true
```

補充說明

「==」vs.「===」

有時候閱讀程式設計教課書，在討論是否等價這件事，有著不同的寫法，如同上面的「==」與「===」，兩個的差別在於：

「x == y」：若是型態不相等，變數會先強制轉換成相同的型態，再進行嚴格比對。

```
console.log(1 == 1); // 輸出 true
console.log("1" == "1"); // 輸出 true
console.log(1 == "1"); // 輸出 true
console.log("1" == 1); // 輸出 true
```

註：若是比對的是「物件」，會比對變數佔用的記憶體位置是否相同。

```
var object1 = {'key': 'value'}, object2 = {'key': 'value'};
console.log(object1 == object2);
```

```
//輸出 false，變數宣告時，會各自佔用不同的記憶體位置。

var object1 = {'key': 'value'};
var object2 = object1;
console.log(object1); // 輸出 { key: 'value' }
console.log(object2); // 輸出 { key: 'value' }
object2.key = 'vvv'; //隨意改值
console.log(object1); //輸出 { key: 'vvv' }
console.log(object2); //輸出 { key: 'vvv' }
console.log(object1 == object2);  //輸出 true


「===」：兩個型態不相等，不轉換型態，直接嚴格比對。最常用的等價邏輯判斷方式。

console.log(1 === 1); // 輸出 true
console.log("1" === "1"); // 輸出 true
console.log(1 === "1"); // 輸出 false
console.log("1" === 1); // 輸出 false
```

## 運算子先後順序

這裡提供給大家參考，無須特別記憶，常用的幾個有運算子先後順利的概念（例如先乘除、後加減）有印象，其它透過未來實作或是工作上用到，再去了解即可。以下表格參考「mozilla MDN web docs」（網址：https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence），表格愈上面的優先權愈高，愈下面優先權愈低：

| 運算子的型態 | 相依性 | 運算子用法 |
|---|---|---|
| 群組 | n/a | ( … ) |
| 成員存取 | left-to-right | … . … |
| 計算的成員存取 | left-to-right | … [ … ] |

| | | |
|---|---|---|
| new (帶有參數) | n/a | new ... ( ... ) |
| 函式呼叫 | left-to-right | ... ( ... ) |
| new (不帶參數) | right-to-left | new ... |
| 後綴增加 | n/a | ... ++ |
| 後綴減少 | | ... -- |
| 邏輯 NOT | right-to-left | ! ... |
| 位元 NOT | | ~ ... |
| 一元運算的加號 | | + ... |
| 一元運算的負號 | | - ... |
| 前綴增加 | | ++ ... |
| 前綴減少 | | -- ... |
| typeof | | typeof ... |
| void | | void ... |
| delete | | delete ... |
| await | | await ... |
| 指數運算 | right-to-left | ... ** ... |
| 乘法運算 | left-to-right | ... * ... |
| 減法運算 | | ... / ... |
| 餘數運算 | | ... % ... |
| 加法運算 | left-to-right | ... + ... |
| 減法運算 | | ... - ... |
| 位元左移 | left-to-right | ... << ... |
| 位元右移 | | ... >> ... |
| 位元無號右移 | | ... >>> ... |

| | | |
|---|---|---|
| 小於 | left-to-right | … < … |
| 小於等於 | | … <= … |
| 大於 | | … > … |
| 大於等於 | | … >= … |
| in | | … in … |
| instanceof | | … instanceof … |
| 等於 | left-to-right | … == … |
| 不等於 | | … != … |
| 嚴格等於 | | … === … |
| 嚴格不等於 | | … !== … |
| 位元 AND | left-to-right | … & … |
| 位元 XOR | left-to-right | … ^ … |
| 位元 OR | left-to-right | … \| … |
| 邏輯 AND | left-to-right | … && … |
| 邏輯 OR | left-to-right | … \|\| … |
| 條件式 | right-to-left | … ? … : … |
| 賦值 | right-to-left | … = … |
| | | … += … |
| | | … -= … |
| | | … **= … |
| | | … *= … |
| | | … /= … |
| | | … %= … |
| | | … <<= … |

| | | |
|---|---|---|
| | | ... >>= ... |
| | | ... >>>= ... |
| | | ... &= ... |
| | | ... ^= ... |
| | | ... \|= ... |
| yield | right-to-left | yield ... |
| yield* | | yield* ... |
| 逗號 / 序列 | left-to-right | ... , ... |

## undefined & null

經常用在討論變數賦值的狀況，undefined 通常會出現在變數已宣告，但未賦值的狀況；null 會出現在變數已宣告，卻還沒決定賦什麼值，需要先給個初始值，或是代表沒有值的概念。

範例

```
let name; //已宣告變數，未賦值
console.log(name); //輸出 undefined

// 下面的物件，有些屬性還不確定，就可以先用 null 來賦予初始值
let obj = {
  name: '南宮恨',
  alias: '黑白郎君',
  age: null,
  skills: [
    '五絕神功', '陰陽一氣', '怒馬凌關', '一氣化九百', '一氣化三千',
    '收化運發', '離合並流', '封靈斬', '陰陽合一碎日月', 'more maneuvers'
  ]
};
```

註：有些 NoSQL 資料庫，在資料屬性為空值時，該資料的屬性不會一同被儲存。

## 案例討論

1. 有一字串「Hello World」，找出 Hello 的部分，再另外加上「空格」與「Kitty」，最後全部轉為大寫後輸出。
2. 3 的 14 次方是多少？9487 除以 7 的餘數是多少？請分別用運算子進行計算後輸出。
3. console.log(false && true && false && true); 的輸出？
   console.log(false || true || false && true); 的輸出？
   console.log( (false || true) && (false && true) ); 的輸出？

# 條件與迴圈

我們把條件與迴圈，稱之為「控制結構」，對於任何程式，都扮演著重要的角色。它們讓你定義的特定條件，控制在何時、何種頻率來執行哪些部分的程式碼。

## if 陳述句、if ... else 陳述句、if...else 陳述句鏈

範例

```
let condition = true;
if(condition) {
   console.log('Hello World!');
}
//輸出 Hello World!
```

範例

```
let condition = false;
if(condition) {
   //若條件為真，則執行這個區塊的程式碼
   console.log('Hello World!');
} else {
   //若條件為假，則執行這個區塊的程式碼
   console.log('May you have a nice day!');
}

//輸出 May you have a nice day!
```

範例

```
let number = 7;
if(number > 10) {
   console.log(1);
} else if(number < 3){
   console.log(2);
} else if(number > 5) {
   console.log(3);
} else if(number === 6) {
   console.log(4);
```

```
}
```

下面的情境，請問輸出是多少？

```
let number = 7;
if(number > 10) {
   console.log(1);
} else if(number === 7){
   console.log(2);
} else if(number > 5) {
   console.log(3);
} else if(number === 6) {
   console.log(4);
}
// 輸出 2
```

if … else 陳述句鏈會依序進行判斷，縱然有多個判斷符合條件，依然會以第一個符合條件的區塊來執行。

| 補充說明 |
| --- |
| 當 if 判斷只需要單獨執行一行，可以不用加「{…}」<br><br>let num = 10;<br>if(num === 8) console.log("It's 8.");<br>if(num === 9) console.log("It's 9.");<br>if(num === 10) console.log("It's 10.");<br><br>//輸出 It’ s 10. |

| 補充說明 |
| --- |
| 三元條件運算子（?:）<br>(condition) ? 條件為真才執行 : 條件為假才執行;<br><br>(3 > 2) ? console.log('true') : console.log('false');<br>// 輸出 true |

```
(1 > 2) ? console.log('true') : console.log('false');
// 輸出 false

可以將判斷結果帶到變數中

let bool = (1 > 2) ? true : false;
console.log(bool);
// 輸出 false

let x = 3;
let y = 4;
let answer = (x > y) ? 'x is greater than y' : 'x is less than y';
console.log(answer);
// 輸出 x is less than y
```

## switch 判斷

範例

```
let number = 7;
switch(number){
   case '7':
      console.log('string 7');
   break;

   case 7:
      console.log('number 7');
   break;

   case 'number':
      console.log('string number');
   break;

   default:
```

```
    console.log('string default');
}
// 輸出 number 7


let number = 7;
switch(number){
  case '7':
  case 7:
  case 'number':
    console.log('number');
  break;

  default:
    console.log('default');
}
// 輸出 number
```

## while 迴圈、for 迴圈

while 迴圈結合了條件判斷的概念，符合條件，則繼續執行區塊內的程式，直到條件不成立，才跳出程式區塊。以下提供範例來說明：

範例

```
印出 1 到 7
let count = 1;
while(count <= 7){
  console.log(count);
  count++;
}
// 輸出 1 2 3 4 5 6 7
// count 遞增到 8 的時候，count <= 7 的條件不成立，則跳出 while() {...}

另一種 while 迴圈使用形式 do{...}while()
```

```
let count = 1;
do{
   console.log(count);
   count++;
}
while(count <= 7)
// 輸出 1 2 3 4 5 6 7，與 while(){...} 差異在於 do 區塊會先執行完，才進行 while 判斷
```

補充說明

無限迴圈：

```
while(1){
   console.log('running');
}
console.log('Done');
//會不斷輸出 running，沒有輸出 Done 的時候，我們需要使用 Ctrl + C 來終止程式運作。
```

通常 while(1){...} 的程式設計方式，會建立起類似 Listener 的程式，隨時監聽資料的接受狀態，例如 Socket 網路程式設計。

for 的無限迴圈形式：

```
for(;;){
   console.log('Hello World!');
}
```

範例

for 迴圈起手式

```
for(setup; condition; increment) {
   //statements
}
```

setup 是變數初始宣告的地方

condition 是變數狀態與判斷條件

increment 是變數賦值的方式

```
for(let i = 0; i < 10; i++) {
   console.log('The value is ' + i);
}
// 輸出 The value is 0 .... The value is 9
```

迴圈內部也可以使用迴圈

```
for(let i = 1; i <= 9; i++) {
   for(let j = 1; j <= 9; j++){
      process.stdout.write(i + ' * ' + j + ' = ' + (i*j) + '\t');
   }
   console.log();
}
```

由於 console.log() 有尾端斷行的特性，所以我們使用「process.stdout.write()」，來達到輸出卻不斷行的效果。

補充說明

迴圈也有 block-scoped

在 for 迴圈以 var 關鍵字宣告 i

```
for(var i = 0; i < 10; i++){
   console.log(i);
}
console.log(i);
//輸出 0, 1, ... , 8, 9, 10
```

從上面的例子可以發現，i 被抬升（Hoisting）到 for(){...} 外的上一行了，縱然迴圈執行完，變數 i 理應在 {...} 結束時被消滅，卻因為變數抬升的關係，導致 for(){...} 以外的作用城，可以存取到變數 i。

在 for 迴圈以 let 關鍵字宣告 i

```
for(let i = 0; i < 10; i++){
```

```
    console.log(i);
}
console.log(i);
// 拋出錯誤訊息 ReferenceError: i is not defined
```

因為在這裡用 let 關鍵字宣告的 i，不會被抬升到 for(){...} 區塊的外部，所以變數 i 不會被外部存取，可以確實掌握變數的生命週期。

補充說明

for 迴圈的形式，常見還有其它幾種。

若是要取得物件/陣列當中的「索引（index）/鍵（key）」
可以使用「for(property/key/index in dataSet) {...}」：

```
let obj = {
    fname: 'Darren',
    lname: 'Yang',
    age: null,
    lineId: 'telunyang'
};
for(let attr in obj) {
    console.log(`The property in object variable is ${attr}`);
}
//輸出 The property in object variable is fname
//輸出 The property in object variable is lname
//輸出 The property in object variable is age
//輸出 The property in object variable is lineId

let arr = [
    'a',
    'b',
    'c'
];
for(let key in arr){
    console.log(`The index number in array variable is ${key}`);
```

```
}
//輸出 The index number in array variable is 0
//輸出 The index number in array variable is 1
//輸出 The index number in array variable is 2
```

若是要取得陣列當中的「值」，可以使用「for(value of dataSet) {...}」：

```
let arr = [
   'a',
   'b',
   'c'
];
for(let value of arr){
   console.log(`The value in array variable is ${value}`);
}
```

但是取得物件當中的值，則無法使用「for(value of dataSet) {...}」，因為 dataSet 是需要可以迭代的（Iterable），是指變數是可以透過 [0], [1], [2], ... , [n] 方式來取得相對應的值，物件內部成員以屬性（Property）的方式存在，沒有「有序的鍵（Key）」，故無法迭代：

```
let obj = {
   fname: 'Darren',
   lname: 'Yang',
   age: null,
   lineId: 'telunyang'
};
for(let value of obj){
   console.log(`The value in object variable is ${value}`);
}
// 拋出錯誤 TypeError: obj is not iterable
```

| 補充說明 |
| --- |
| 「迭代器（Iterator）」<br><br>一般走訪 for 迴圈，是透過來實現迴圈： |

```
let items = [1,2,3,4,5];
for(let i = 0; i < items.length; i++){
   console.log(`The item is ${items[i]}`);
}
//輸出 The item is 1
//輸出 The item is 2
//輸出 The item is 3
//輸出 The item is 4
//輸出 The item is 5
```

然而 Iterator 簡化了這個過程。

我們在下面模擬一個自訂的 Iterator

```
function createIterator(items){
   let i = 0;
   return {
      next: function(){
         let done = (i >= items.length);
         let value = done ? undefined : items[i++];
         return {
            done: done,
            value: value
         }
      }
   }
}

let iterator = createIterator([1, 2, 3]);

console.log(iterator.next()); //輸出 { done: false, value: 1 }
console.log(iterator.next()); //輸出 { done: false, value: 2 }
console.log(iterator.next()); //輸出 { done: false, value: 3 }
console.log(iterator.next()); //輸出 { done: true, value: undefined }
console.log(iterator.next()); //輸出 { done: true, value: undefined }
```

| |
|---|
| 迭代器透過類似指標的方式，在每一次 next() 後，將 key 自動往下遞增計算，直到 done 為 true，迴圈即執行完畢。 |

| 補充說明 |
|---|
| Iterable Proctol<br>必須有一個 [@@iterator] 屬性，並且回傳一個 iterator<br><br>Iterator Proctol<br>必須有一個 next 屬性，呼叫該屬性每次必須回傳一個 { value: any, done: boolean } 的物件 |

## 案例討論

1. 使用 for 迴圈，判斷 1 到 100 當中，有哪些奇數？
2. 使用 for 迴圈，判斷 10000 裡面，有多少 3 的乘方？（3, 6, 9, 12, ....）
3. 使用 while 迴圈，完成 1、2 的問題。

# 函式

函式（Function，又稱函數），是把程式碼集合在一起，以便能夠重複使用它們的一種方法。

## 基本結構

```
function 函式名稱() {
   //程式執行區域
}
```

| 補充說明 |
| --- |
| 有時候會寫成<br><br>var 函式名稱 = function() {<br>  //程式執行區域<br>}<br><br>var 函式名稱 = () => {<br>  //程式執行區域<br>} |

| 補充說明 |
| --- |
| 箭頭函數：<br><br>let reflect = value => value;<br>函式參數 回傳值<br><br>相當於<br><br>let reflect = function(value) {<br>  return value;<br>}<br>console.log( reflect(20) );<br>// 輸出 20 |

```
let sum = (num1, num2) => num1 + num2;
```

相當於

```
let sum = function(num1, num2){
   return num1 + num2;
}
console.log( sum(20,50) );
// 輸出 70
```

```
let getName = () => "Darren Yang";
```

相當於

```
let getName = function() {
   return "Darren Yang";
}
console.log( getName() );
// 輸出 Darren Yang
```

下面這一種比較常見：

```
let sum = (num1, num2) => {
   return num1 + num2;
}
```

相當於

```
let sum = function(num1, num2){
   return num1 + num2;
}
console.log( sum(8,9) );
// 輸出 17
```

## 建立函式

```
function say() {
   console.log('Hello World!');
}
```

或是

```
let say = function() {
   console.log('Hello World!');
}
```

## 呼叫函式

接續建立函式的內容，我們使用在函式名稱後面加上「()」，便能執行函式內容。

```
say();  //輸出 Hello World!
```

## 參數傳遞

將參數設定在「函式的括號內」，同時在外部使用「賦予參數特定值」的函式，便能將「參數的值」傳遞給區塊當中的程式碼使用。

```
function say(title){
```

```
  console.log(`Hello ${title}!`);
}

say('Darren Yang'); //輸出 Hello Darren Yang!
```

| 補充說明 |
| --- |
| 多個參數傳遞時的作法<br><br>function say (greeting, title) {<br>  console.log('You said: ' + greeting + ' ' + title);<br>}<br><br>say('Hello', 'World'); //輸出 You said: Hello World |

## 回傳值

```
function say(greeting, title) {
  return 'You said: ' + greeting + ' ' + title;
}

console.log( say('Hello', 'World') ); // 輸出 You said: Hello World
```

## 回呼函式

常稱為回呼函數、回調函式，為一種「延續傳遞風格」（Continuation-passing style）的函式程式寫法，它的對比的是前面我們所提供的基本函式範例（直接風格，Direct style）。回呼函數可以將特定函式作為傳遞參數，在該函式中呼叫執行，將原本應該在該函式中回傳的值，交給下一個函式來執行。

| 範例 |
| --- |
| 建立一個 say 函式，其中的第二個參數也是函式： |

```
//主程式
say('Darren Yang', function(result){
   console.log(result);
});

//建立 say 函式
function say(name, callback_function){
   //say 函式會處理特定程式碼後，透過 callback_function 把結果或訊息回傳到主程式
   callback_function(`Hi, [${name}] ... how have you been ?`);
}
```

1. 主程式呼叫 say 函式的時候，除了第 1 個參數 name，在第 2 個參數放置一個 callback_function 函式，一起傳遞到 say 函式。
2. say 程式區塊中，使用主程式傳遞到 say 函式當中的 name 跟 callback_function 參數。
3. 經過處理，將結果作為 callback_function 函式的參數，再透過 callback_function 送回主程式，變成主程式的第二個參數。
4. 此時 callback_function 展開變成一般的函式「function(result) { ... }」，此時該函式的「result」就是在 say 函式中處理的結果，被 callback_function 作為參數，帶回主程式。

流程

1. 想像主程式原先的樣子：

```
say('Darren Yang', callback_function);
```

2. 想像建立 say 函式的樣子：

```
function say(name, callback_function){
   callback_function(`Hi, [${name}] ... how have you been ?`);
}
```

3. 經過一連串的傳遞與處理：

```
say('Darren Yang', callback_function);

function say(name, callback_function){
    callback_function(`Hi, [${name}] … how have you been ?`);
}
```

4. 主程式變成這個樣子，result 是「Hi, xxx … how have you been ?」的文字處理結果，可以在「{ … }」當中使用：

```
say('Darren Yang', function(result){ … });
```

5. 我們可以自訂「function(result) { … }」的內容：

```
say('Darren Yang', function(result){
    console.log(`=====>  ${result}`);
});
//輸出 =====>  Hi, [Darren Yang] … how have you been ?
```

使用回呼函數的目的，在於確保程式運作流程的明確性（另一種說法是指移交程式執行的控制權），例如一個回呼函式用於讀取檔案內容，主程式為了確保檔案內容被完整讀取出來，使用回呼函數，待回呼函數執行完畢後，再透過主程式帶入的函式參數，將檔案內容回傳到主程式當中。

## 遞迴函式

舉個例子，知名漫畫/卡通《哆啦 A 夢》裡面的大雄，在房間裡面，用時光電視看著未來的情況。電視畫面中的那個時候，他正在用時光電視，看著未來的情況。電視畫面中的那個時候，他正在用時光電視，看著未來的情況。電視畫面中的那個時候，他正在用時光電視，看著未來的情況。電視畫面中的那個時候，他正在用時光電視，看著未來的情況。電視畫面中的那個時

候，他正在用時光電視，看著未來的情況。電視畫面中的那個時候，他正在用時光電視，看著未來的情況...

上述的例子說明了，遞迴是一種類似鏡中巢狀的圖像，電視中有電視，該電視中又有電視，如此重複著。以程式的角度來看，是指在函式當中，使用函式自身的方法。

| 範例 |
| --- |
| 自己呼叫自己：<br>function go(){<br>  go();<br>}<br>go();<br><br>遞減數字相加（10 + 9 + 8 + ... + 2 + 1）：<br>function add(num){<br>  if(num === 0) return 0;<br>  if(num === 1) return 1;<br>  return num + add(num-1);<br>}<br>console.log( add(10) );<br>// 輸出 55<br><br>計算 n 階乘：<br>function factorial(num){<br>  if(num === 0) return 1;<br>  return num * factorial(num-1);<br>}<br>console.log( factorial(5) );<br>// 輸出 120<br><br>Fibonacci 數列：<br>function fibonacci(num){ |

```
    if(num === 0) return 0;
    if(num === 1) return 1;
    return fibonacci(num-1) + fibonacci(num-2);
}
console.log( fibonacci(10) );
//數列從第 0 項開始，所以 fibonacci(10) 是指第 11 項，輸出 55
```

使用遞迴時，需要有結束程式的時候，例如加上條件判斷，否則會不斷執行，造成資源浪費。

## 案例討論

1. 撰寫一個函式 print()，印出字串圖像，第一個參數是輸出次數，第二個參數是字串 "=^.^="，輸出以下結果：

   0 =^.^=

   1 =^.^=

   2 =^.^=

   3 =^.^=

   4 =^.^=
2. 將 1. 案例的函式，改成箭頭函式來輸出，並將字串圖像改為 "^_^"。
3. 撰寫一個函式 double()，帶入一數值參數，回傳結果為參數 * 2 的結果，並加以輸出。
4. 承 4.，double(double(15)) 結果是什麼？

# 陣列與物件

## 陣列

### 簡介

在沒有使用陣列的情況下，我們需要這樣記錄資料：

```
let name1 = "Alex";
let name2 = "Bill";
let name3 = "Cook";
let name4 = "Darren";
.
.
.
let name9999 = 'Somebody';
```

上面這種列表會變得很不好用，假設每一個人的名字都需要一張紙來記錄，這要浪費多少紙張？於是我們可以使用類似「清單」概念的陣列，將所有名字都記錄在同一張紙上，這樣就簡單多了。

建立陣列

| 範例 |
| --- |
| 陣列的宣告：<br>let arr = [];<br><br>也有人這麼寫<br>let arr = new Array();<br><br><br><br>宣告陣列時，建立初始值：<br>let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];<br><br>有時候為了排版好看，會將陣列中的元素，透過鍵盤的 Enter，讓每個元素對齊：<br>let listOfName = [<br>  'Alex',<br>  'Bill',<br>  'Cook',<br>  'Darren'<br>]; |

存取陣列

一般來說，陣列的索引，從 [0] 開始，到 [n – 1] 結束。

範例

```
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
console.log( listOfName[0] ); // 輸出 Alex
console.log( listOfName[3] ); // 輸出 Darren
console.log( listOfName[4] ); // 輸出 Undefined
```

設定、修改陣列元素

範例

```
設定（新增）元素：
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
listOfName[4] = 'Ellen'; // 指定索引位置來新增元素
console.log(listOfName); //輸出 [ 'Alex', 'Bill', 'Cook', 'Darren', 'Ellen' ]


使用 push()，將資料加到陣列尾端：
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
listOfName.push('Ellen');
listOfName.push('Fox');
console.log(listOfName);


修改元素：
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
listOfName[0] = 'Allen';
listOfName[2] = 'Carl';
console.log(listOfName[0]); // 輸出 Allen
console.log(listOfName[2]); // 輸出 Carl
```

```
console.log(listOfName); // 輸出 [ 'Allen', 'Bill', 'Carl', 'Darren' ]
```

刪除元素，使用 pop()，將會刪除陣列尾端的資料：

```
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
listOfName.pop();
console.log(listOfName); // 輸出 [ 'Alex', 'Bill', 'Cook' ]
listOfName.pop();
console.log(listOfName); // 輸出 [ 'Alex', 'Bill' ]
```

可以透過變數賦值的方式，取得被 pop() 刪除的資料：

```
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
let name = listOfName.pop();
console.log(name); // 輸出 Darren
```

另一種刪除元素的方式，使用 delete，但會保持原先索引的位置：

```
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
delete listOfName[3];
console.log(listOfName); // 輸出   [ 'Alex', 'Bill', 'Cook', <1 empty item> ]

for(let i = 0; i < listOfName.length; i++){
   console.log(`index: ${i}, value: ${listOfName[i]}`);
}
// 輸出 index: 0, value: Alex
// 輸出 index: 1, value: Bill
// 輸出 index: 2, value: Cook
// 輸出 index: 3, value: undefined
```

刪除陣列第一個元素，使用 shift()：

```
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
listOfName.shift();
console.log(listOfName); // 輸出 [ 'Bill', 'Cook', 'Darren' ]
```

有時候從尾端刪除的陣列資料，需要放在陣列前端，使用 unshift()：

```
let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];
let name = listOfName.pop(); // 從陣列尾端刪除 Darren
listOfName.unshift(name); // 將刪除的陣列尾端資料放到陣列前端
console.log(listOfName); // 輸出 [ 'Darren', 'Alex', 'Bill', 'Cook' ]
```

補充說明

二維陣列：

```
let arr = [
   ['a0', 'a1', 'a2', 'a3'],
   ['b0', 'b1', 'b2'],
   ['c0', 'c1', 'c2', 'c3', 'c4'],
];

console.log( arr[0][1] ); // 輸出 a1
console.log( arr[2][4] ); // 輸出 c4
```

它的概念如下表格：

| 二維陣列 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | a0 | a1 | a2 | a3 | |
| 1 | b0 | b1 | b2 | | |
| 2 | c0 | c1 | c2 | c3 | c4 |

左側索引代表每一列的陣列資料，上方索引代表每一列當中特定欄位的位置。

建立二維陣列：

```
let arr1d = [];
for(let i = 1; i <= 9; i++){
   arr1d.push(i); //先建立一維陣列，該陣列之後被新陣列新增，變成二維陣列
}
```

```
let arr2d = [];
for(let j = 1; j <= 9; j++){
   arr2d.push(arr1d); //連續新增先前建立的一維陣列，便可成為二維陣列
}

console.log(arr2d);

// 輸出:
// [ [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
//   [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
//   [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
//   [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
//   [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
//   [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
//   [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
//   [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ],
//   [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ] ]
```

陣列混合其它資料類別

```
let arr = [
   3,
   'Darren',
   ['aaa', 'bbb', 3.14],
   10
];
console.log( arr[0] ); // 輸出 3
console.log( arr[2][2] ); // 輸出 3.14
console.log( arr[3] ); // 輸出 10
```

陣列合併

使用陣列的 concat() ，則可將陣列合併。

| 範例 |
| --- |
| 兩個陣列合併：<br>let arr1 = ['a0', 'a1', 'a2'];<br>let arr2 = ['b0', 'b1', 'b2'];<br>let newArr = arr1.concat(arr2);<br>console.log(newArr); // 輸出 [ 'a0', 'a1', 'a2', 'b0', 'b1', 'b2' ]<br><br>三個陣列合併：<br>let arr1 = ['a0', 'a1', 'a2'];<br>let arr2 = ['b0', 'b1', 'b2'];<br>let arr3 = ['c0', 'c1', 'c2'];<br>let newArr = arr1.concat(arr2, arr3);<br>console.log(newArr); // 輸出 [ 'a0', 'a1', 'a2', 'b0', 'b1', 'b2', 'c0', 'c1', 'c2' ] |

陣列元素索引查詢

當我們需要查詢/尋找某個資料的索引，可以使用 indexOf()。

| 範例 |
| --- |
| let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];<br>console.log( listOfName.indexOf('Bill') ); // 輸出 1<br>console.log( listOfName.indexOf('Darren') ); // 輸出 3<br><br>若是找不到該元素的索引（代表陣列中沒有該元素的資料），則會回傳 -1：<br>let listOfName = ['Alex', 'Bill', 'Cook', 'Darren'];<br>console.log( listOfName.indexOf('Grace') ); // 輸出 -1 |

| 補充說明 |
| --- |
| 當我們需要判斷某個值是否存在陣列當中，可以透過 if 條件來進行： |

（註：indexOf() 為 -1，代表 scale 值不存在於 arrScale 檔中）

```
let scale = 9;
let arrScale = [1,2,3,4,5];
if( arrScale.indexOf(scale) === -1 ) {
   //若是 scale 不存在於 arrScale 當中，則執行這個區塊的語法
   console.log('Not exists');
}
// 輸出 Not exists
```

你也可以把陣列直接放在 if 判斷中，不用額外宣告變數

```
let scale = 9;
if( [1,2,3,4,5].indexOf(scale) === -1 ) {
   //若是 scale 不存在於陣列 [1,2,3,4,5] 當中，則執行這個區塊的語法
   console.log('Not exists');
}
```

補充說明

若是希望能隨機選出陣列元素，例如抽獎、餐廳選擇等，我們可以透過 Math 物件來處理。以下簡單介紹 Math 物件常用的功能。

產生隨機數（0 到 1 之間的隨機浮點數）：

```
console.log( Math.random() ); // 輸出 0.02529304315381542
console.log( Math.random() ); // 輸出 0.8048744968811596
console.log( Math.random() ); // 輸出 0.3708412087199269
console.log( Math.random() ); // 輸出 0.9822622936344187
```

取整數：

```
console.log( Math.floor(3.1415926) ); // 輸出 3
```

建立隨機索引：

```
console.log( Math.floor( Math.random() * 4 ) );
```

註：因為 Math.random() 產生 0 - 1 之間的浮點數，0.01 乘上 4 的整數為 0，0.9 乘上 4 的整數為 3，所以可以確定建立的索引範例，會在 0 – 3 之間（[0],[1],[2],[3]，共四個索引）。

取得隨機索引：

```
let index = Math.floor( Math.random() * 4 );
```

隨機選擇餐廳

```
let arrMeal = ['麥當勞', '肯德基', '摩斯漢堡', '頂呱呱', '漢堡王'];
let idxRandom = Math.floor( Math.random() * 5 );

console.log(`今天中餐，我選擇 ${arrMeal[idxRandom]}`);
// 輸出 今天中餐，我選擇 麥當勞
```

案例討論

1. 使用迴圈來新增一維陣列 [9,8,7, … , 2,1]。
2. 建立二維陣列，輸出結果如下：

   [ [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ],
   [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ],
   [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ],
   [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ],
   [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ],
   [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ],
   [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ],
   [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ],
   [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ] ]
3. 透過陣列建立餐點清單，有五個自訂選項（例如水餃、三明治、麵包等自訂文字），並以隨機方式選擇餐點。

# 物件

## 簡介

物件和陣列很類似，但是物件使用字串來存取不同元素，而非數字。這個字串，叫作「鍵」（Key）或是屬性（property），它所指向的元素叫作「值」（Value）。通常把這兩個合在一起，稱為「鍵值對」（Key-Value pair），最主要的目的，在於儲存更多特定對象的資訊

## 建立物件

| 範例 |
| --- |
| 物件的宣告：<br>let obj = {};<br><br>也有人這麼寫<br>let obj = new Object();<br><br><br>自訂物件屬性<br>let cat = {<br>  'legs': 4,<br>  'name': 'Darren',<br>  'color': 'Tangerine',<br>  'ability': {<br>  'jump': function(){<br>  console.log('Jump!');<br>  },<br>  'sound': function(){<br>  console.log('Meow~');<br>  },<br>  'sleep': function() {<br>  console.log('ZZZzzz...');<br>  }<br>  }<br>}; |

```
console.log( cat.legs ); // 輸出 4
console.log( cat.color ); // 輸出 Tangerine
cat.ability.sound(); // 輸出 Meow~
cat.ability.sleep(); // 輸出 ZZZzzz...
```

也有人物件屬性不帶引號

```
let cat = {
  legs: 4,
  name: 'Darren',
  color: 'Tangerine',
  ability: {
    jump: function(){
      console.log('Jump!');
    },
    sound: function(){
      console.log('Meow~');
    },
    sleep: function() {
      console.log('ZZZzzz...');
    }
  }
};
```

存取物件

| 範例 |
| --- |

```
let cat = {
  legs: 4,
  name: 'Darren',
  color: 'Tangerine',
  ability: {
    jump: function(){
      console.log('Jump!');
```

```
      },
      sound: function(){
         console.log('Meow~');
      },
      sleep: function() {
         console.log('ZZZzzz...');
      }
   }
};
```

使用「[ ]」來存取屬性或函式：

```
console.log( cat['legs'] ); // 輸出 4
console.log( cat['name'] ); // 輸出 Darren
cat['ability'].sleep(); // 輸出 ZZZzzz...
cat['ability'].sound(); // 輸出 Meow~
```

使用「.」來存取屬性或函式：

```
console.log( cat.legs ); // 輸出 4
console.log( cat.color ); // 輸出 Tangerine
cat.ability.sound(); // 輸出 Meow~
cat.ability.sleep(); // 輸出 ZZZzzz…
```

增加物件屬性：

```
cat['eye_color'] = 'black';
cat.habbit = 'play with slave';

console.dir( cat , {depth: null} ); // console.dir 可以列出物件（或陣列）的內容
//輸出：
// { legs: 4,
//   name: 'Darren',
```

```
//   color: 'Tangerine',
//   ability:
//    { jump: [Function: jump],
//      sound: [Function: sound],
//      sleep: [Function: sleep] },
//   eye_color: 'black',
//   habbit: 'play with slave' }
```

物件與陣列的整合運用

範例

建立三個記錄學生資料的物件：

```
let studentA = {
   id: '10801001',
   name: 'Alex',
   age: 18,
   phone_number: '0911111111'
};

let studentB = {
   id: '10801002',
   name: 'Bill',
   age: 19,
   phone_number: '0922222222'
};

let studentC = {
   id: '10801003',
   name: 'Cook',
   age: 18,
   phone_number: '0933333333'
};
```

建立空陣列，儲存所有學生的資料：

```
let arrStudents = [];
```

```
arrStudents.push(studentA);
arrStudents.push(studentB);
arrStudents.push(studentC);

console.dir(arrStudents, {depth: null});
// 輸出：
// [ { id: '10801001',
//     name: 'Alex',
//     age: 18,
//     phone_number: '0911111111' },
//   { id: '10801002',
//     name: 'Bill',
//     age: 19,
//     phone_number: '0922222222' },
//   { id: '10801003',
//     name: 'Cook',
//     age: 18,
//     phone_number: '0933333333' } ]
```

取得第二個學生的資料：

```
console.log( arrStudents[1] );
//輸出
// { id: '10801002',
//   name: 'Bill',
//   age: 19,
//   phone_number: '0922222222' }

console.log(`Bill is ${arrStudents[1]['age']} years old.`);
// 輸出 Bill is 19 years old.

console.log(`Bill's student ID is ${arrStudents[1].id} .`);
// 輸出 Bill's student ID is 10801002.
```

透過迴圈顯示學生資料，並以自定格式輸出：

```
for(let i = 0; i < arrStudents.length; i++){
   console.log(`${arrStudents[i].name}'s student ID is ${arrStudents[i].id},
he is ${arrStudents[i].age} years old,
his phone number is ${arrStudents[i].phone_number}.`);
}

//輸出：
// Alex's student ID is 10801001,
// he is 18 years old,
// his phone number is 0911111111.
// Bill's student ID is 10801002,
// he is 19 years old,
// his phone number is 0922222222.
// Cook's student ID is 10801003,
// he is 18 years old,
// his phone number is 0933333333.
```

補充說明

在前面的案例中，屬性會使用有「單/雙引號」括住的字串，或是以「點」作為屬性存取的方式，若是選擇使用「點」來存取屬性，若是屬性的文字有「-」、「.」等特殊用途的字，會產生錯誤。

最好的方式，就是在屬性文字當中，有「-」、「.」等字，一定要用「[ ]」以及「單/雙引號」的格式，來存取屬性的值。

```
let studentC = {
   id: '10801003',
   name: 'Cook',
   age: 18,
   phone_number: '0933333333',
   'test-test': 'test'
};

console.log( studentC['test-test'] ); // 輸出 test
```

## Date 物件

在 Javascript 的互動應用中，經常使用「Date 物件」當中的類時間函式，來判斷、比較、計算時間的差距，或是使用特定的時間格式，例如 timestamp、ISO 8601 等。以下提供幾個範例：

| 範例 |
| --- |
| 我們可以直接使用「new Date()」來使用時間相關函式，並取得相對應的結果；以下請用 console.log() 輸出。<br><br>宣告 Date 物件：<br>let date = new Date();<br><br>今天星期幾：<br>date.getDay(); // 0，指星期天；數值範例是 0 - 6<br><br>你也可以直接使用時間物件，例如看看今天星期幾：<br>new Date().getDay(); // 0，指星期天；數值範例是 0 – 6<br><br>其它範例：<br>date.getFullYear() // 2019，西元年<br>date.getMonth() // 月份，6 月時執行會出現 5，比正常值少 1，記得加 1<br>date.getDate() // 26，當月 26 號<br>date.getHours(); // 幾時，例如 14 時<br>date.getMinutes(); // 幾分，例如 58 分<br>date.getSeconds(); //幾秒，例如 53 秒<br><br>還有許多應用，可查閱 MDN web docs。<br>https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date |

| 補充說明 |
| --- |

判斷今天星期幾（0 為星期日，1 為星期一，2 為星期二，...，6 為星期六）

```
let day;
switch (new Date().getDay()) {
   case 0:
      day = "Sunday";
   break;

   case 1:
      day = "Monday";
   break;

   case 2:
      day = "Tuesday";
   break;

   case 3:
      day = "Wednesday";
   break;

   case 4:
      day = "Thursday";
   break;

   case 5:
      day = "Friday";
   break;

   case 6:
      day = "Saturday";
   break;

   default:
      day = "What?!";
}
console.log(day);
```

| |
|---|
| |

| 補充說明 |
|---|
| 若是時間物件的函式（方法）回傳，只有一位數的結果（例如六月只回傳 6，而非 06），我們可以用一些小技巧來輸出自訂的格式。<br><br>let month = (new Date().getMonth() + 1); // getMonth() 範圍 0 到 11，所以要加 1<br><br>if( month < 10 ){ // 數值小於 10，則左側加個 '0'<br>  month = '0' + month;<br>}<br><br>console.log(month); // 輸出 06 |

案例討論

1. 請在課堂上的三名學生物件資料中，新增 scores 屬性，屬性的值為一物件，分別擁有「english」、「math」兩個屬性，各自隨機產生分數（0 到 100，取整數）。其中一位的範例如下：

```
let studentA = {
  id: '10801001',
  name: 'Alex',
  age: 18,
  phone_number: '0911111111',
  scores: {
    english: 94,
    math: 87
  }
};
```

| 參考答案 |
|---|
| let studentA = {<br>  id: '10801001',<br>  name: 'Alex', |

```
        age: 18,
        phone_number: '0911111111'
};

let studentB = {
        id: '10801002',
        name: 'Bill',
        age: 19,
        phone_number: '0922222222'
};

let studentC = {
        id: '10801003',
        name: 'Cook',
        age: 18,
        phone_number: '0933333333'
};


studentA.scores = {
   english: Math.floor(Math.random() * 100 + 1),
   math: Math.floor(Math.random() * 100 + 1)
}

studentB.scores = {
   english: Math.floor(Math.random() * 100 + 1),
   math: Math.floor(Math.random() * 100 + 1)
}

studentC.scores = {
   english: Math.floor(Math.random() * 100 + 1),
   math: Math.floor(Math.random() * 100 + 1)
}


console.dir(studentA);
```

```
console.dir(studentB);
console.dir(studentC);


// 輸出
{ id: '10801001',
name: 'Alex',
age: 18,
phone_number: '0911111111',
scores: { english: 20, math: 72 } }

{ id: '10801002',
name: 'Bill',
age: 19,
phone_number: '0922222222',
scores: { english: 98, math: 78 } }

{ id: '10801003',
name: 'Cook',
age: 18,
phone_number: '0933333333',
scores: { english: 100, math: 49 } }
```

2. 承上題，這三名學生的「english」平均以及「math」的平均是多少？請用自訂的文字輸出格式來顯示結果。（提示：先將三名學生的個人物件資料加入陣列，再用迴圈走訪迴圈來取得三名學生的資料，各自將 english 或 math 的分數加總後，在迴圈外分別除以 3，取整數）

參考答案

（接續第 1 題的參考答案程式碼）

```
let arrStudents = [];
arrStudents.push(studentA);
arrStudents.push(studentB);
arrStudents.push(studentC);
```

```
let numScoresEnglish = 0;
let numScoresMath = 0;

for(let i = 0; i < arrStudents.length; i++) {
        numScoresEnglish += arrStudents[i].scores.english;
        numScoresMath += arrStudents[i].scores.math;
}

console.log(`三名學生英文平均分數: ${Math.floor(numScoresEnglish/3)}`);
console.log(`三名學生數學平均分數: ${Math.floor(numScoresMath/3)}`);
```

3. 顯示當前的「西元年-月-日 時:分:秒」。（例如：2019-06-26 21:15:30）

參考答案（?:三元條件運算子版本）

```
let date = new Date();

let year, month, day, hour, minute, second;

// 2019，西元年
year = date.getFullYear()

// 月份，10 月時執行會出現 9，比正常值少 1，記得加 1
month = ( ( date.getMonth() + 1 ) < 10 ) ? '0' + (date.getMonth() + 1) : (date.getMonth() + 1);

// 25，當月 25 號
day = ( date.getDate() < 10 ) ? '0' + date.getDate() : date.getDate();

// 幾時，例如 14 時
hour = ( date.getHours() < 10 ) ? '0' + date.getHours() : date.getHours();

// 幾分，例如 58 分
minute = ( date.getMinutes() < 10 ) ? '0' + date.getMinutes() : date.getMinutes();

//幾秒，例如 53 秒
second = ( date.getSeconds() < 10 ) ? '0' + date.getSeconds() : date.getSeconds();
```

```
console.log(`${year}-${month}-${day} ${hour}:${minute}:${second}`);
```

參考答案（if else 版本）

```
let date = new Date();

let year, month, day, hour, minute, second;

// 2019，西元年
year = date.getFullYear()

// 月份，10 月時執行會出現 9，比正常值少 1，記得加 1
if( (date.getMonth() + 1) < 10 ) {
   month = '0' + (date.getMonth() + 1);
} else {
   month = (date.getMonth() + 1);
};

// 25，當月 25 號
if( date.getDate() < 10 ) {
   day = '0' + date.getDate();
} else {
   day = date.getDate();
};



// 幾時，例如 14 時
if( date.getHours() < 10) {
   hour = '0' + date.getHours();
} else {
   hour = date.getHours();
};

// 幾分，例如 58 分
if( date.getMinutes() < 10) {
```

```
   minute = '0' + date.getMinutes();
} else {
   minute = date.getMinutes();
};

//幾秒，例如 53 秒
if( date.getSeconds() < 10) {
   second = '0' + date.getSeconds();
} else {
   second = date.getSeconds();
};
console.log(`${year}-${month}-${day} ${hour}:${minute}:${second}`);
```

# 【進階篇】

# HTML & CSS 基礎知識

## HTML 簡介

HTML（Hyper Text Markup Language，超文字標記語言）是用來產生 Web 網頁的語言。HTML 裡面的標籤（tags，例如 ul、li、a、p、div 等，以及 HTML5 增加的 section、article、aside、nav、footer 等），這些標籤會告訴瀏覽器何時顯示、顯示什麼、如何顯示。

### 建立 HTML 檔案

範例

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no" />
    <title>歡迎來到我的家</title>
  </head>
  <body>
    <div id="wrapper">
      <!-- nav 在這裡是網頁上緣導覽列 -->
      <header class="head-info">
        <nav class="nav nav-info">
          <ul class="nav-body">
            <li class="nav-list">
              <a class="center link-custom">首頁</a>
            </li>
            <li class="nav-list">
              <a class="center link-custom">連結 1</a>
            </li>
            <li class="nav-list">
              <a class="center link-custom">連結 2</a>
            </li>
            <li class="nav-list">
              <a class="center link-custom">連結 3</a>
            </li>
          </ul>
        </nav>
      </header>
```

```
<!-- aside 在這裡是左側的選單列表 -->
<aside class="menu">
  <ul class="menu-body">
    <li class="menu-list">
      <a class="center">側欄連結 1</a>
    </li>
    <li class="menu-list">
      <a class="center">側欄連結 2</a>
    </li>
    <li class="menu-list">
      <a class="center">側欄連結 3</a>
    </li>
  </ul>
</aside>

<!-- main 是主要顯示內容的區域 -->
<main class="content-container">
  <article class="article-paragraph">
    <section class="episode">
      <h3 class="title">HTML parser 開發不求人 - 第三節</h3>
      <div class="content">
        <p>動態網頁的元素走訪，都需要透過不斷地實作、練習（自己手動輸入程式碼），同時...</p>
      </div>
      <div class="content-more">
        <a class="more-link">More</a>
      </div>
    </section>
    <section class="episode">
      <h3 class="title">HTML parser 開發不求人 - 第二節</h3>
      <div class="content">
        <p>動態網頁的元素走訪，都需要透過不斷地實作、練習（自己手動輸入程式碼），同時...</p>
      </div>
      <div class="content-more">
        <a class="more-link">More</a>
      </div>
    </section>
    <section class="episode">
      <h3 class="title">HTML parser 開發不求人 - 第一節</h3>
      <div class="content">
        <p>動態網頁的元素走訪，都需要透過不斷地實作、練習（自己手動輸入程式碼），同時...</p>
      </div>
      <div class="content-more">
        <a class="more-link">More</a>
      </div>
    </section>
  </article>
</main>

<!-- footer 在這裡是簡單提供網站的基礎資訊 -->
```

```
        <footer class="footer">
          <p>歡迎光臨！本站由 Darren Yang 本人親自虛構...</p>
        </footer>
      </div>
    </body>
</html>
```

補充說明

基本 HTML 階層，大致上長這個樣子：

若是網頁排版上的設定，應該長這個樣子：

html
head
title
body
div#wrapper
header.head-info
nav.nav.nav-info
ul.nav-body
li.nav-list
a.center.link-custom
aside.menu
ul.menu-body
li.menu-list
a.center
main.content-container
article.article-paragraph
section.episode
h3.title
div.content
p
div.content-more
a.more-link
footer.footer
p

標籤介紹

介紹範例裡頭的標籤。以下的文件（document），是指操作 html 的一種物件：

| 標籤 | 範例 | 說明 |
|---|---|---|
| <!DOCTYPE> | <!DOCTYPE html><html>…</html> | 定義文件的格式。 |
| <head> | <head><title>我的網站</title></head> | 定義有關這個文件的資訊，至少會與 <title></title>一起使用。 |
| <title> | <title>我的網站</title> | 定義文件的標題。 |
| <header> | <header>頁首資訊</header> | 文件的頁首資訊。 |
| <nav> | <nav>導覽列</nav> | 導覽列，定義導覽連結。 |
| <ul> | <ul><li>清單項目 1</li><li>清單項目 2</li></ul> | 定義尚未排序的列表。 |
| <li> | <ul><li>清單項目 1</li><li>清單項目 2</li></ul> | 定義一個項目列表。 |
| <!-- … --> | <!-- 這裡是註解 --> | HTML 文件中的註冊。 |
| <a> | <a>友站連結</a> | 定義超連結。 |
| <main> | <main>放置主要內容</main> | 具體說明文件的主要內容。 |
| <section> | <section>放置需要區隔的資訊</section> | 定義文件的部分內容。 |
| <article> | <article>文章內容</article> | 定義一篇文章。 |
| <aside> | <aside>側欄資料</aside> | 定義在網頁側邊的內容。 |
|  | 放置需要區隔的資訊 | 類似<section>，定義文件的部分內容。 |
| <h1> 到 <h6> | <h3>某個主題或是需要明顯標註的資訊</h3> | 定義 HTML 的標題/標頭。 |
|  | 文章當中的段落 | 定義文字段落。 |
| <footer> | <footer>頁尾資訊</footer> | 文件的頁尾資訊。 |

| 補充說明 |
|---|
| 網頁當中，也有表格元素，叫作「table」，它的格式通常如下：<br><br><table><br>  <thead><br>  <tr><br>  <th>標題 1</th><br>  <th>標題 2</th><br>  <th>標題 3</th><br>  </tr><br>  </thead><br>  <tbody><br>  <tr> |

```
        <td>第 1 行的第 1 個表格</td>
        <td>第 1 行的第 2 個表格</td>
        <td>第 1 行的第 3 個表格</td>
      </tr>
      <tr>
        <td>第 2 行的第 1 個表格</td>
        <td>第 2 行的第 2 個表格</td>
        <td>第 2 行的第 3 個表格</td>
      </tr>
      <tr>
        <td>第 3 行的第 1 個表格</td>
        <td>第 3 行的第 2 個表格</td>
        <td>第 3 行的第 3 個表格</td>
      </tr>
    </tbody>
    <tfoot>
      <tr>
        <td>表格尾端第 1 個表格</td>
        <td>表格尾端第 2 個表格</td>
        <td>表格尾端第 3 個表格</td>
      </tr>
    </tfoot>
  </table>
```

它跟其它元素一樣，可能會有自己的 css 設定，例如 id、class 等。

其它標籤的使用方式，可以參閱 w3school 的介紹：

https://www.w3schools.com/tags/

加入超連結

| 範例 |
| --- |
| `<a href="https://www.ntu.edu.tw" target="_blank">國立臺灣大學</a>` |

a 是一種 html tag/element，也可視為一種物件，href 與 target 是 a 的屬性。

自訂屬性

在 HTML 5 的規範下，可以用「data-*」的格式，來自訂屬性，以便 jQuery 可以透過自訂屬性來取得自訂的資料，例如「data-item-id='11'」、「data-user-name='Darren Yang'」、「data-height='1024'」、「data-width='768'」、「data-what-you-may-call-it='Iron man'」

範例

```
<a href="https://www.ntu.edu.tw" target="_blank" data-item-id="5566" data-user-name="Darren Yang" data-tmp-path="/tmp">國立臺灣大學</a>
```

確定屬性後，便能透過元素擷取的套件，來取得屬性資訊。

補充說明

```
let html = `<a id="alink" href="https://www.ntu.edu.tw" target="_blank" data-item-id="5566" data-user-name="Darren Yang" data-tmp-path="/tmp">國立臺灣大學</a>`;
```

使用 **jQuery**：

```
const jsdom = require("jsdom");
const { JSDOM } = jsdom;
const { window } = new JSDOM(html);
const $ = require('jquery')(window);

console.log( $('a').attr('href') );  //輸出 https://www.ntu.edu.tw
console.log( $('a').attr('data-item-id') );  //輸出 5566
console.log( $('a').attr('data-user-name') );  //輸出 Darren Yang
console.log( $('a').text() ); //輸出 國立臺灣大學
```

補充說明

```
let html = `<a id="alink" href="https://www.ntu.edu.tw" target="_blank" data-item-id="5566" data-user-name="Darren Yang" data-tmp-path="/tmp">國立臺灣大學</a>`;
```

另一種 jQuery 使用方法，將 <a> 直接變成 jQuery 物件，直接存取屬性或內文，無須向下尋找或走訪。

```
const jsdom = require("jsdom");
const { JSDOM } = jsdom;
const { window } = new JSDOM();
const $ = require('jquery')(window);

console.log( $(html).attr('href') ); //輸出 https://www.ntu.edu.tw
console.log( $(html).attr('data-item-id') ); //輸出 5566
console.log( $(html).attr('data-user-name') ); //輸出 Darren Yang
console.log( $(html).text() ); //輸出 國立臺灣大學
```

補充說明

```
let html = `<div><a id="alink" href="https://www.ntu.edu.tw" target="_blank" data-item-id="5566" data-user-name="Darren Yang" data-tmp-path="/tmp">國立臺灣大學</a></div>`;
```

我們將原先 <a> 外面，用  來包覆，此時，「$(html)」是以  元素作為 jQuery 物件的開頭，而後使用「.find()」再往下尋找子孫元素（例如 <a>）。

```
const jsdom = require("jsdom");
const { JSDOM } = jsdom;
const { window } = new JSDOM(); //沒有帶入 html
const $ = require('jquery')(window);
console.log( $(html).find('a').attr('href') );  //輸出 https://www.ntu.edu.tw
console.log( $(html).find('a').attr('data-item-id') );  //輸出 5566
console.log( $(html).find('a').attr('data-user-name') );  //輸出 Darren Yang
console.log( $(html).find('a').text() );  //輸出 國立臺灣大學
```

# CSS 簡介

CSS（Cascading Style Sheets）為一種描述 HTML 元素如何呈現在螢幕、紙本或其它媒體（手機、平板等）顯示的技術，它可以一次控制、整合多個網頁的規格，也可以透過外部嵌入的方法來使用。在課程中，由於將以後端資料擷取的角度來操作，所以前端頁面渲染、產生的方式，我們並不會深入討論，僅以透過選擇器來進行元素的擷取與使用。

語法

| 基本結構 |
| --- |
| 選擇器 {<br>　屬性: 設定值;<br>　…<br>}<br><br>選擇器[屬性=值] {<br>　屬性: 設定值;<br>　…<br>}<br><br>父選擇器 子選擇器 {<br>　屬性: 設定值;<br>　…<br>}<br><br>父選擇器 子選擇器 孫選擇器 … {<br>　屬性: 設定值;<br>　…<br>}<br><br>父選擇器 > 子選擇器 > 孫選擇器 > … {<br>　屬性: 設定值;<br>　…<br>}<br><br>選擇器 1, 選擇器 2, 選擇器 3, 選擇器 4, …… , 選擇器 n { |

```
  屬性: 設定值;
  ...
}
```

選擇器

範例

```
p {
  color: red;
  text-align: center;
}

#para1 {
  text-align: center;
  color: red;
}

.center {
  text-align: center;
  color: red;
}

p#para2 {
  text-align: center;
  color: red;
}

p.center {
  text-align: center;
  color: red;
}

nav ul {
  list-style-type: circle;
}
```

```
nav ul li {
  padding-left: 20px;
}

h1, h2, p {
  text-align: center;
  color: red;
}

div[data-item-id="11"] {
  width: 1280px;
  height: 960px;
  background-color: white;
}
```

常見選擇器的格式：

| 符號 | 說明 | 範例 |
|---|---|---|
| # | 元素 id | <article id="A001">國立臺灣大學</article> |
| . | 元素類別 | <br>  <img src="https://bit.ly/2X1JAZs" /><br> |
| 元素標籤 | 直接使用元素的標籤 | <nav><br>  <ul><br>  <li>Item01</li><br>  <li>Item02</li><br>  <li>Item03</li><br>  </ul><br></nav> |

網頁元素的屬性、內文擷取，都會以「選擇器」的設定，作為指定元素對象的方式。

## 案例討論

1. 打開 examples/cases/vistiHTML/localHTML.js，分別將不同的註解內容執行看看。
   （取消註解 “//” ：反白有 “//” 的程式碼，然後按下 Ctrl + /）
2. 承上，試著走訪別的元素，取得屬性與元素內部文字。

# 非同步特性與套件管理介紹

## 同步 & 非同步

同步（synchronous）的處理方式：
想像我們在銀行、郵局櫃台申辦業務，在你申辦的項目尚未完成前，你會繼續站在櫃台，承辦人員持續幫你處理業務，直到項目完成，你便會離開，櫃台會叫號，換下一位。

非同步（asynchronous）的處理方式：
想像我們在正在寫考卷，倘若你先寫完，你可以先繳卷，然後先離開，至於閱卷老師何時改完，我們並不知道，只知道老師改完，會立刻讓你知道成績。

### ECMAScript 的沿革

Javascript 核心的語言特性，由 ECMA-262 標準來定義，而該標準中定義的語言，又被稱為 ECMAScript，可以視為 Javascript 的一種子集合（意思指 Javascript 可以做得更多，例如在某些瀏覽器裡，可以實現多於 ECMAScript 定義的功能）。

參考維基百科的資料

| 版本 | 發表日期 | 與前版本差異 |
|---|---|---|
| 1 | 1997 年 6 月 | 首版。 |
| 2 | 1998 年 6 月 | 格式修正，以使得其形式與 ISO/IEC16262 國際標準一致。 |
| 3 | 1999 年 12 月 | 強大的正規表示式，更好的詞法作用域鏈處理，新的控制指令，例外處理，錯誤定義更加明確，資料輸出的格式化及其它改變。 |
| 4 | 放棄 | 由於關於語言的複雜性出現分歧，第 4 版本被放棄，其中的部分成為了第 5 版本及 Harmony 的基礎。 |
| 5 | 2009 年 12 月 | 新增「嚴格模式（strict mode）」，一個子集用作提供更徹底的錯誤檢查,以避免結構出錯。澄清了許多第 3 版本的模糊規範，並適應了與規範不一致的真實世界實現的行為。增加了部分新功能，如 getters 及 setters，支援 JSON 以及在物件屬性上更完整的反射。 |

| | | |
|---|---|---|
| 5.1 | 2011 年 6 月 | ECMAScript 標 5.1 版形式上完全一致於國際標準 ISO/IEC 16262:2011。 |
| 6 | 2015 年 6 月 | ECMAScript 2015（ES2015），第 6 版，最早被稱作是 ECMAScript 6（ES6），添加了類和模組的語法，其他特性包括疊代器，Python 風格的生成器和生成器表達式，箭頭函式，二進位資料，靜態型別陣列，集合（maps，sets 和 weak maps），promise，reflection 和 proxies。作為最早的 ECMAScript Harmony 版本，也被叫做 ES6 Harmony。 |
| 7 | 2016 年 6 月 | ECMAScript 2016（ES2016），第 7 版，多個新的概念和語言特性。 |
| 8 | 2017 年 6 月 | ECMAScript 2017（ES2017），第 8 版，多個新的概念和語言特性。 |
| 9 | 2018 年 6 月 | ECMAScript 2018（ES2018），第 9 版，包含了非同步迴圈，生成器，新的正規表示式特性和 rest/spread 語法。 |
| 10 | 2019 年 6 月 | ECMAScript 2019（ES2019），第 10 版 |
| 11 | 2020 年 6 月 | ECMAScript 2020（ES2020），第 11 版 |

參考網址：https://zh.wikipedia.org/wiki/ECMAScript

## 非同步的概念

`setTimeout(function(){ 程式碼 }, 毫秒)`：幾毫秒後，執行程式碼。

範例

```
setTimeout(function(){
    console.log("Hello World");
}, 3000);
```

範例

```
setTimeout(function(){
    console.log("001");
}, 3000);
```

```
setTimeout(function(){
    console.log("002");
}, 5000);
```

上面的範例，分別會在第 3 秒、第 5 秒後，印出 001 以及 002。若是以下的情形呢？

範例

```
setTimeout(function(){
    console.log("001");
}, 5000);

setTimeout(function(){
    console.log("002");
}, 3000);
```

會先印出 002，再印出 001，這個結果可能不是我們要的；此時我們可以用以下方法，讓 setTimeout() 依序執行：

範例

```
//先執行外部 setTimeout()
setTimeout(function(){
    console.log("001");

    //再執行內部 setTimeout()
    setTimeout(function(){
        console.log("002");
    }, 3000);

}, 5000);
```

如果我們要印出 001 ~ 010 呢？可能會變成下面這樣：

範例

```
setTimeout(function(){
    console.log("001");
    setTimeout(function(){
        console.log("002");
        setTimeout(function(){
            console.log("003");
            setTimeout(function(){
                console.log("004");
                setTimeout(function(){
```

```
                    console.log("005");
                    setTimeout(function(){
                        console.log("006");
                        setTimeout(function(){
                            console.log("007");
                            setTimeout(function(){
                                console.log("008");
                                setTimeout(function(){
                                    console.log("009");
                                    setTimeout(function(){
                                        console.log("010");
                                    }, 2000);
                                }, 2000);
                            }, 2000);
                        }, 2000);
                    }, 2000);
                }, 2000);
            }, 2000);
        }, 2000);
    }, 2000);
}, 2000);
```

此時程式架構可能會這樣下面這張圖：

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

此時程式會開始變得難以維護與持續開發，我們就要使用 JavaScript ES6 的 Promise() 物件。

範例

```
new Promise(function(resolve, reject){
    setTimeout(function(){
        console.log("001");
        resolve()
    }, 2000);
}).then(function(data){
    setTimeout(function(){
        console.log("002");
    }, 2000);
}).catch(function(error){
    console.log(error);
});
```

當我需要輸出 001、002、003、004 呢？可以用下面的結構來進行：

說明

```
new Promise(function(resolve, reject){
    //你的程式碼 + resolve();
}).then(() => {
    let p2 = new Promise(function(resolve, reject){
        //你的程式碼 + resolve();
    });
    return p2;
}).then(() => {
    let p3 = new Promise(function(resolve, reject){
        //你的程式碼 + resolve();
    });
    return p3;
}).then(() => {
    let p4 = new Promise(function(resolve, reject){
       //你的程式碼 + resolve();
    });
    return p4;
}).then((data) => {
    console.log(`執行完畢`);
})
.catch((err) => {
    console.log(err); //例外處理
});
```

範例

```
new Promise(function(resolve, reject){
    setTimeout(function(){
        console.log("001");
```

```
        resolve();
    }, 2000);
}).then(() => {
    let p2 = new Promise(function(resolve, reject){
        setTimeout(function(){
            console.log("002");
            resolve();
        }, 2000);
    });
    return p2;
}).then(() => {
    let p3 = new Promise(function(resolve, reject){
        setTimeout(function(){
            console.log("003");
            resolve();
        }, 2000);
    });
    return p3;
}).then(() => {
    let p4 = new Promise(function(resolve, reject){
        setTimeout(function(){
            console.log("004");
            resolve();
        }, 2000);
    });
    return p4;
}).then((data) => {
    console.log(`執行完畢`);
})
.catch((err) => {
    console.log(err); //例外處理
});
```

此時使用一堆 .then()，雖然結構上有改善，但還是稍嫌複雜，所以可以另外用函式（function）加以定義，並在 function 前面加上 async，之後再到主程式裡面，在執行函式的名稱前面加上 await，便可依序執行。

範例

```
//定義函式
async function get001(){
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            console.log("001");
            resolve();
        }, 2000);
```

```
    });
}

async function get002(){
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            console.log("002");
            resolve();
        }, 2000);
    });
}

async function get003(){
    return new Promise(function(resolve, reject){
        setTimeout(function(){
            console.log("003");
            resolve();
        }, 2000);
    });
}

//IIFE
(
    async function (){
        await get001();
        await get002();
        await get003();
    }
)();
```

透過 Promise 進行非同步開發

在以前，為了確保程式進行的流程，我們會使用回呼函式。

範例（檔案位置：examples\cases\promise\callback01.js）

```
const fs = require('fs');

//讀取 source.txt 的檔案內容
fs.readFile("source.txt", function(err, contents){
    if(err){
        throw err;
    }
    console.log("讀取 source.txt 完成!");

    //若是讀取沒問題，則寫入 destination.txt 檔案
    fs.writeFile("destination.txt", contents + ' Hello NTU!',  function(error){
        if(error){
```

```
            throw error;
        }
        console.log("寫入 destination.txt 成功!");
    });
});
```

若是讀寫的次數很多呢？

範例（檔案位置：examples\cases\promise\callback02.js）

```
const fs = require('fs');

//讀取 tmp.log 的檔案內容
fs.readFile("tmp.log", function(err, contents){
    if(err){
        throw err;
    }

    console.log(`讀取 tmp.log 成功`);

    //若是讀取沒問題，則加上新增的文字，然後寫入 cb1.txt 檔案
    fs.writeFile("cb1.txt", contents + ' cb1', function(err){
        if(err){
            throw err;
        }

        console.log(`寫入 cb1.txt 成功`);

        //讀取 cb1.txt 的檔案內容
        fs.readFile("cb1.txt", function(err, contents){
            if(err) throw err;

            console.log(`讀取 cb1.txt 成功`);

            //若是讀取沒問題，則加上新增的文字，然後寫入 cb2.txt 檔案
            fs.writeFile("cb2.txt", contents + ' cb2', function(err){
                if(err) throw err;

                console.log(`寫入 cb2.txt 成功`);

                //讀取 cb2.txt 的檔案內容
                fs.readFile("cb2.txt", function(err, contents){
                    if(err) throw err;

                    console.log(`讀取 cb2.txt 成功`);

                    //若是讀取沒問題，，則加上新增的文字，然後寫入 cb3.txt 檔案
                    fs.writeFile("cb3.txt", contents + ' cb3', function(err){
                        if(err) throw err;
```

```
                        console.log(`寫入 cb3.txt 成功`);

                        //讀取 cb3.txt 的檔案內容
                        fs.readFile("cb3.txt", function(err, contents){
                            if(err) throw err;

                            console.log(`讀取 cb3.txt 成功`);

                            //若是讀取沒問題，則加上新增的文字，然後寫入 cb4.txt 檔案
                            fs.writeFile("cb4.txt", contents + ' cb4', function(err){
                                if(err) throw err;

                                console.log(`寫入 cb4.txt 成功`);

                            });
                        });
                    });
                });
            });
        });
    });
});
```

此時便會進入回呼地獄（callback hell），程式碼會變得複雜且難以維護。

在 ES6（ECMAScript 2015）當中，支援 Promise（還有先前討論到的 let、const、arrow function 等），將原先回呼函式的格式，以鏈結的方式，來確保程式執行的流程，以及生命週期。

範例（檔案位置：examples\cases\promise\promise01.js）

```
const fs = require('fs');

let p = new Promise(function(resolve, reject){
    fs.readFile("tmp.log", function(err, contents){
        if(err) {
            //讀取錯誤就回傳錯誤的物件到 .catch() 中
            reject(err);
            return;
        };

        //成功就往 .then() 傳遞
        resolve(contents);
    });
});
```

```
p.then(function(contents){
    fs.writeFile("promise01.log", `${contents} I'm ironman!!`, function(err)
{
        if(err){
            throw err;
        }
        console.log("寫入完成");
    });
}).catch(function(err){
    console.error(err);
});
```

另一種 Promise 使用方式↓

範例（檔案位置：examples\cases\promise\promise01.js）

```
const fs = require('fs');

new Promise(function(resolve, reject){
    fs.readFile("tmp.log", function(err, contents){
        if(err) {
            //讀取錯誤就回傳錯誤的物件到 .catch() 中
            reject(err);
            return;
        };

        fs.writeFile("tmp1.log", `${contents} I'm ironman!!`, function(err){
            if(err){
                throw err;
            }
            resolve('tmp1'); //成功就往 .then() 傳遞
        });
    });
}).then((tmpNum) => {
    console.log(`Write to ${tmpNum} ok`);
    let p2 = new Promise(function(resolve, reject){
        fs.readFile("tmp1.log", function(err, contents){
            if(err) {
                //讀取錯誤就回傳錯誤的物件到 .catch() 中
                reject(err);
                return;
            };

            //成功就往 .then() 傳遞
            fs.writeFile("tmp2.log", `${contents} I'm spiderman!!`, function(err){
                if(err){
                    throw err;
```

```
                }
                resolve('tmp2'); //成功就往 .then() 傳遞
            });
        });
    });
    return p2;
}).then((tmpNum) => {
    console.log(`Write to ${tmpNum} ok`);
    let p3 = new Promise(function(resolve, reject){
        fs.readFile("tmp2.log", function(err, contents){
            if(err) {
                //讀取錯誤就回傳錯誤的物件到 .catch() 中
                reject(err);
                return;
            };

            //成功就往 .then() 傳遞
            fs.writeFile("tmp3.log", `${contents} I'm superman!!`, function(err){
                if(err){
                    throw err;
                }
                resolve('tmp3'); //成功就往 .then() 傳遞
            });
        });
    });
    return p3;
}).then((tmpNum) => {
    console.log(`Write to ${tmpNum} ok`);
    let p4 = new Promise(function(resolve, reject){
        fs.readFile("tmp3.log", function(err, contents){
            if(err) {
                //讀取錯誤就回傳錯誤的物件到 .catch() 中
                reject(err);
                return;
            };

            //成功就往 .then() 傳遞
            fs.writeFile("tmp4.log", `${contents} I'm batman!!`, function(err){
                if(err){
                    throw err;
                }
                resolve('tmp4'); //成功就往 .then() 傳遞
            });
        });
    });
    return p4;
}).then((tmpNum) => {
    console.log(`Write to ${tmpNum} ok`);
```

```
})
.catch((err) => {
    console.error(err); //例外處理
});
```

但一直 .then() 也不是辦法，流程一長，程式碼依然不好維護，於是在 ES7（ECMAScript 2016）完整加入了 async、await，讓程式排版看起來更簡潔。

範例（檔案位置：examples\cases\promise\await.js）

```
const fs = require('fs');
const util = require('util');

// 讓 readFile 可以使用 await 關鍵字
const readFile = util.promisify(fs.readFile);
// 讓 writeFile 可以使用 await 關鍵字
const writeFile = util.promisify(fs.writeFile);

//IIFE, Immediately-Invoked Function Expressions
(async function(){
    try {
        //讀取 tmp.log
        let contents = await readFile("tmp.log", "utf8");
        //寫入 await01.log
        await writeFile("await01.log", `${contents} I'm ironman...`);
        //讀取 await01.log
        contents = await readFile("await01.log", "utf8");
        //寫入 await02.log
        await writeFile("await02.log", `${contents} I'm spiderman...`);
        //讀取 await02.log
        contents = await readFile("await02.log", "utf8");
        //寫入 await03.log
        await writeFile("await03.log", `${contents} I'm superman...`);
        //讀取 await03.log
        contents = await readFile("await03.log", "utf8");
        //寫入 await04.log
        await writeFile("await04.log", `${contents} I'm batman...`);
        console.log(`await.js 執行完成`);
    } catch(err) {
        throw err;
    }
})();
```

註：

使用 await，區塊外必須使用 async 函式，否則會拋出錯誤。

| |
|---|
| **SyntaxError: await is only valid in async function** |

建立非同步的函式

| 範例 |
|---|
| 與一般函式相同，在函式前面加上 async 即可：<br>async function **函式名稱**() {<br>  ....<br>}<br><br>let func = async function() {<br>  ....<br>}<br><br>let func = async () => {<br>  ....<br>} |

使用非同步函式

| 範例 |
|---|
| async function 新函式(){<br>  await 既有函式(); // 使用非同步方式開發的函式<br>}<br><br>註：既有函式，也需要是非同步函式 |

# 套件管理工具介紹與使用

在課程中，我們會經常使用到以下的套件：

| 套件名稱 | 簡介 |
|---|---|

| | |
|---|---|
| nightmare | 用來運作瀏覽器行為的工具，例如按下超連結、網頁換頁、滾動捲軸等。 |
| jsdom | 在 Node.js 裡頭操作 DOM 元件的工具，因為 jQuery 需要視窗元件才能在後端環境作業，所以借用 jsdom 當中的 window，讓 jQuery 能夠產生作用。 |
| jQuery | 前端環境操作 HTML 元素的工具，借用 jsdom 的視窗元件，達到在後端環境作業的目的。 |
| cheerio | 類似 jQuery 的用法，但是更輕量化。 |
| moment | 用來顯示、分析、驗證、操作、計算日期時間的工具。 |
| util | Node.js 的輔助工具。 |
| exec | Node.js 執行指令的工具。 |
| mysql | 驅動 MySQL 或 MariaDB 的套件，用來存取資料庫。 |
| crypto | 雜湊、加密工具，在本案例中，使用 md5 技術，協助建立 associated key。 |
| fs | 檔案系統，用來讀寫檔案的工具。 |

套件的安裝，可以執行指令：npm i <套件名稱> --save

例如 npm i nightmare --save

也有人使用 npm install nightmare –save

更多好用套件的資訊，可以到 npm（Node.js Package Manager）網站查詢：

https://www.npmjs.com/

（圖）npm 首頁

（圖）輸入套件名稱，或關鍵字，可以透過自動完成列表來檢索

（圖）可以透過左側的排序功能，選擇合適的套件

（圖）搜尋結束，套件資訊會完整顯示在頁面上

## 案例討論

1. 打開 examples/cases/promise/visitHTML 路徑中的 curl.js，執行看看；複製程式中的網址，檢視網頁原始碼，再試著走訪其它資料。
2. 在專案資料夾的根目錄位置，新增 tmpA.txt 檔案，檔案內容為「A」；建立 async 函式，以連續 await readFile 和 writeFile 來建立 tmpB.txt、tmpC.txt，tmpB.txt 檔案內容為「A B」, tmpC.txt 檔案內容為「A B C」，以 IIFE 方式進行。

# 網頁元素走訪、分析與擷取

這裡開始，是本課程的重頭戲，我們將透過 cURL 或 nightmare 來取得 html tags 的文字資料，並傳遞到特定變數中，再將變數傳遞給 jQuery 套件來轉成 jQuery 物件，並使用 jQuery 選擇器（selector）來走訪不同階層的 html 元素，並取得元素的屬性值或內文，同時將屬性值與內文，透過正規表達式來過濾出我們要的資訊，再將相關資訊儲存在物件或陣列當中，轉成 JSON 檔案，供後續使用（寫入資料庫，或是與他人資料交換等）。

## 靜態網頁元素走訪展示

「靜態」指的是瀏覽器前往指定的網址之後，除了使用者跳頁（按其它連結或表單送出）外，不會因為後續的互動而產生新的 html 元素。以下我們使用 Chrome 瀏覽器，以「維基百科」的「三國演義角色列表」頁面，來進行展示。

（圖）維基百科：三國演義角色列表

多人姓氏角色 [編輯]

張 [編輯]

| 姓名 | 字 | 籍貫 | 列傳 | 首回 | 末回 | 史構 |
|---|---|---|---|---|---|---|
| 張讓 | | 豫州潁川郡 | 侍奉靈帝的宦官十常侍之首，暗殺了何進，但被袁紹追殺時投河自殺。 | 001 | | 史 |
| 張角 | | 冀州鉅鹿郡 | 鉅鹿郡的在野人士。為太平道的教祖，向民眾廣傳教義。趁社會動亂，得到廣大民眾支持。結成黃巾黨對抗漢王朝，發動黃巾之亂。 | 001 | | 史 |
| 張寶 | | 冀州鉅鹿郡 | 張角之弟。和張角一起舉兵欲推翻漢王朝，而發動黃巾之亂。自稱地公將軍，用兄長傳授的妖術指揮叛亂，後被屬下嚴政刺殺。 | 001 | | 史 |
| 張梁 | | 冀州鉅鹿郡 | 張角、張寶之弟。和兄長們一起舉兵欲推翻漢王朝，而發動黃巾之亂。自稱人公將軍，張角死後繼續指揮叛亂，敗給了官軍，在曲陽之戰戰死。 | 001 | | 史 |
| 張飛 | 益德，演義字翼德 | 幽州涿郡涿縣 | 演義中為蜀漢五虎大將之一。與劉備、關羽結為異姓兄弟。在長坂坡之戰中，單槍匹馬在長坂橋上，喝退曹操萬大軍的追擊。後因關羽死於東吳之手，急於為兄報仇，要在三天以內做成大量白色的鎧甲。但遷怒部將范疆、張達，在睡覺時被范疆、張達所殺。 | 001 | | 史 |
| 張鈞 | | 冀州中山國 | 東漢郎中，向靈帝上奏討伐黃巾的功臣，並陳述十常侍的惡政，卻遭靈帝逐出。<br>史上張鈞被十常侍誣陷修習黃巾道，遭下獄拷打致死。 | 002 | 002 | 史 |
| | | | 漁陽郡的土豪。與張純聯合發動叛 | | | |

（圖）我們要取得表格的資料

（圖）在網頁空白處，按滑鼠右鍵，選擇檢視網頁原始碼

```
211 <table class="wikitable sortable" width="100%">
212
213 <tbody><tr>
214 <th align="center" width="60px">姓名
215 </th>
216 <th align="center" width="50px">字
217 </th>
218 <th align="center" width="130px">籍貫
219 </th>
220 <th align="center" width="">列傳
221 </th>
222 <th align="center" width="40px">首回
223 </th>
224 <th align="center" width="40px">末回
225 </th>
226 <th align="center" width="40px">史構
227 </th></tr>
228 <tr>
229 <td><a href="/wiki/%E5%BC%B5%E8%AE%93" class="mw-disambig" title="張讓">張讓</a>
230 </td>
231 <td>
232 </td>
233 <td>豫州潁川郡
234 </td>
235 <td>侍奉靈帝的宦官十常侍之首，暗殺了何進，但被袁紹追殺時投河自殺。
236 </td>
237 <td>001
238 </td>
239 <td>
240 </td>
241 <td>史
242 </td></tr>
243 <tr>
244 <td><a href="/wiki/%E5%BC%B5%E8%A7%92" title="張角">張角</a>
245 </td>
246 <td>
247 </td>
248 <td>冀州鉅鹿郡
249 </td>
250 <td>鉅鹿郡的在野人士。為太平道的教祖，向民眾廣傳教義。趁社會動亂，得到廣大民眾支持。結成黃巾黨對抗漢王朝，發動黃巾之亂。
251 </td>
252 <td>001
253 </td>
254 <td>
255 </td>
256 <td>史
257 </td></tr>
258 <tr>
259 <td><a href="/wiki/%E5%BC%B5%E5%AF%B6" class="mw-redirect" title="張寶">張寶</a>
260 </td>
261 <td>
262 </td>
263 <td>冀州鉅鹿郡
264 </td>
265 <td>張角之弟。和張角一起舉兵欲推翻漢王朝，而發動黃巾之亂。自稱地公將軍，用兄長傳授的妖術指揮叛亂，後被屬下嚴政刺殺。
266 </td>
267 <td>001
```

（圖）HTML 原始碼。我們要取得 <table> 的資料

（圖）對目標 <table> 附近任一元素（例如「張」這個姓）按滑鼠右鍵，再按「檢查」

（圖）出現解析的元素資訊，我們移到 <table> 元素，可以看到網頁顯示 css 選擇器

（圖）table → tbody → 第 1 個 tr → 第 1 個 td

讓我們來看看程式碼；請參考 examples/Wiki/ThreeKingdoms.js

初始化設定

```
//透過 node.js 來操作檔案系統，例如建立資料夾、新增檔案、寫入檔案等
const fs = require('fs');

//node.js 的工具，目前用於將 exec 異步化（非同步化）
const util = require('util');

//將 exec 非同步化 (可以使用 await，以及 .then, .catch)
const exec = util.promisify(require('child_process').exec);

//將 fs 功能非同步化
```

```
const mkdir = util.promisify(fs.mkdir);
const writeFile = util.promisify(fs.writeFile);

//引入 jQuery 機制
const jsdom = require("jsdom");
const { JSDOM } = jsdom;
const { window } = new JSDOM();
const $ = require('jquery')(window);

//放置主要資訊變數 (陣列)
let arrLink = [];

//瀏覽器標頭，讓對方認為我們是人類，而非機器人 (爬蟲)
const headers = {
  'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36',
 'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3',
 'Accept-Language': 'zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7',
};
```

主要函式

```
//建立一個名為 func 的 async fucntion，參數是 url
async function func(url) {
  let { stdout, stderr } = await exec(`curl -X GET ${url} -L
  -H "User-Agent: ${headers['User-Agent']}"
  -H "Accept-Language: ${headers['Accept-Language']}"
  -H "Accept: ${headers.Accept}"`, {encoding: 'utf8', maxBuffer: 500 * 1024});

  //取得 html (then 回傳回來的是物件，物件的屬性 stdout 才是 html 字串)
  let html = stdout;

  //額外寫入未經網頁 js 程式變更過的 html 原始碼，到自訂檔案中
  await writeFile(`wiki.html`, html);

  //姓名, 人物的維基百科連結, 字, 籍貫, 列傳, 首回, 末回, 史構
  let wikiName = '', wikiLink = '', wikiAlias = '',
    wikiBirthplace = '', wikiDescription = '', wikiBeginEpisode = '',
    wikiEndEpisode = '', wikiIdentity = '';
```

```
//物件變數，用來放置人物相關資訊
let obj = {};

//取得人物姓名的表格
$(html).find('table.wikitable.sortable').each(function(index, element) {
  //走訪取得每一個人物的表格資料（使用 jQuery 的功能來取得 html 元素裡面的資料）
  $(element).find('tbody tr').each(function(idx, elm) {
    //姓名
    wikiName = $(elm).find('td:eq(0)').find('a').text();
    //維基百科連結
    wikiLink = $(elm).find('td:eq(0)').find('a').attr('href');
    //字
    wikiAlias = $(elm).find('td:eq(1)').text();
    //籍貫
    wikiBirthplace = $(elm).find('td:eq(2)').text();
    //列傳
    wikiDescription = $(elm).find('td:eq(3)').text();
    //首回
    wikiBeginEpisode = $(elm).find('td:eq(4)').text();
    //末回
    wikiEndEpisode = $(elm).find('td:eq(5)').text();
    //史構
    wikiIdentity = $(elm).find('td:eq(6)').text();

    //若是姓名變數沒有文字，則跳到下一個元素去執行。
    //註：.each 是 return，while 跟 for 迴圈是 continue 或 break
    if (wikiName == '') return;

    //整理人物資訊在物件裡
    obj = {
      name: wikiName, //姓名
      link: 'https://zh.wikipedia.org' + wikiLink, //維基百科連結
      alias: wikiAlias, //字
      birthplace: wikiBirthplace, //籍貫
      description: wikiDescription, //列傳
      beginEpisode: wikiBeginEpisode, //首回
      endEpisode: wikiEndEpisode, //末回
      identity: wikiIdentity, //史構
    };
```

```
        //過濾掉不必要的字元
        for(let key in obj){
          let str = String(obj[key]);
          obj[key] = str.replace(/\n/g, '');
        }

        //加入陣列變數
        arrLink.push(obj);

        //物件變數初始化，讓下一個人物能夠用
        obj = {};
      });
    });
}
```

主程式區域

```
//主程式區域
try {
  //範例: 維基百科「三國演義角色列表」
  let url = 'https://zh.wikipedia.org/wiki/%E4%B8%89%E5%9B%BD%E6%BC%94%E4%B9%89%E8%A7%92%E8%89%B2%E5%88%97%E8%A1%A8';

  //執行函式
  func(url).then(async () => {
    //建立資料夾
    await mkdir('output', {recursive: true});

    //將物件轉成 json 格存，儲存檔案
    await writeFile('output/wiki.json', JSON.stringify(arrLink, null, 2));
  });
} catch(err) {
  console.error(err);
}
```

補充說明

透過 curl 指令取得的 html 原始碼，會發現 <table> 的 class='wikitable sortable jquery-tablesorter' 少了「jquery-tablesorter」，是因為維基百科有使用前端 js 函式庫，將原先的 <table> 從單純的表格，變成擁有可以排序的功能與圖示（前端動態加入 class）。

因此，若是使用選擇器「table.wikitable.sortable.jquery-tablesorter」則一定找不到，要修正成「table.wikitable.sortable」（就是拿掉 jquery-tablesorter 即可）。

```
<li>異：《三國演義》人物的姓名與歷史上不同。</li></ul></li
<h2><span id=".E5.A4.9A.E4.BA.BA.E5.A7.93.E6.B0.8F.E8.
<h3><span id=".E5.BC.B5"></span><span class="mw-headli
<table class="wikitable sortable" width="100%">

<tbody><tr>
<th align="center" width="60px">姓名
</th>
```

（圖）實際的 html 原始碼，沒有 jquery-tablesorter 類別


```
Nodejs-Html-Parser ▸ output ▸ {} wiki.json ▸ {} 1
[
  {
    "name": "張讓",
    "link": "https://zh.wikipedia.org/wiki/%E5%BC%B5%E8%AE%93",
    "alias": "",
    "birthplace": "",
    "description": "侍奉靈帝的宦官十常侍之首，暗殺了何進，但被袁紹追殺時投河自殺。",
    "beginEpisode": "001",
    "endEpisode": "",
    "identity": "史"
  },
  {
    "name": "張角",
    "link": "https://zh.wikipedia.org/wiki/%E5%BC%B5%E8%A7%92",
    "alias": "",
    "birthplace": "",
    "description": "鉅鹿郡的在野人士。為太平道的教祖，向民眾廣傳教義。趁社會動亂，得到廣大民眾支持。結成黃巾黨對抗漢王朝，發動黃巾之亂。",
    "beginEpisode": "001",
    "endEpisode": "",
    "identity": "史"
  },
  {
    "name": "張寶",
    "link": "https://zh.wikipedia.org/wiki/%E5%BC%B5%E5%AF%B6",
    "alias": "",
    "birthplace": "",
    "description": "張角之弟。和張角一起舉兵欲推翻漢王朝，而發動黃巾之亂。自稱地公將軍，用兄長傳授的妖術指揮叛亂，後被屬下嚴政刺殺。",
    "beginEpisode": "001",
    "endEpisode": "",
    "identity": "史"
  },
  {
    "name": "張梁",
    "link": "https://zh.wikipedia.org/wiki/%E5%BC%B5%E6%A2%81",
    "alias": "",
    "birthplace": "",
    "description": "張角、張寶之弟。和兄長們一起舉兵欲推翻漢王朝，而發動黃巾之亂。自稱人公將軍，張角死後繼續指揮叛亂，敗給了官軍，在曲陽之戰戰死。",
    "beginEpisode": "001",
    "endEpisode": "",
    "identity": "史"
```


（圖）將擷取結果整理成 JSON 格式的檔案

# 正規表達式介紹與用法

正規表達式（Regular Expression）是用來配對、過濾、替換文字的一種表示法。請先進入「https://regex101.com/」頁面，我們之後測試正規表達式，都會透過這個網頁的功能。大家，正規表達式是需要大量練習才能了解的知識，希望大家都能透過頻繁地練習，慢慢感受到正規表達式在文字處理上的便捷。

（圖）網頁的樣式，請記得選擇 FLAVOR 為 ECMAScript（JavaScript）

（圖）使用正規表達式，來判斷字串是否符合文字格式或條件

下面表格為快速參考的範例：

| 說明 | 正規表達式 | 範例 |
|---|---|---|
| 一個字元: a, b or c | [abc] | abcdef |
| 一個字元，除了: a, b or c | [^abc] | abcdef |
| 一個字元，在某個範圍內: a-z | [a-z] | abcd0123 |
| 一個字元，不在某個範圍內: a-z | [^a-z] | abcd0123 |
| 一個字元，在某個範圍內: a-z or A-Z | [a-zA-Z] | abcdXYZ0123 |
| 避開特殊字元 | \ ex. \? | ? |
| 任何單一字元 | . | 任何字元 |
| 任何空白字元（\f \r \n \t \v） | \s | 空格、換行、換頁等 |
| 任何非空白字元（不是 \f \r \n \t \v） | \S | 非空格、非換行、非換頁等 |
| 任何數字 | \d | 10ab |
| 任何非數字 | \D | 10ab |
| 任何文字字元 | \w | 10ab/*AZ^$ |
| 任何非文字字元 | \W | 10ab/*AZ^$ |
| 以群組的方式配對，同時捕捉被配對的資料 | (...)<br>ex. (1[0-9]{3}\|20[0-9]{2}) | 1992, 2019, 1789, 1776, 1024, 3000, 4096, 8192 |
| 配對 a 或 b | a\|b | addbeeeaaccbaa |
| 0 個或 1 個 a | a? | addbeeeaaccbaa |
| 0 個或更多的 a | a* | addbeeeaaccbaa |
| 1 個或更多的 a | a+ | aaa, aaaaa |
| 完整 3 個 a | a{3} | aaa, aaaaa |
| 3 個以上的 a | a{3,} | aa, aaa, aaaaa |
| 3 個到 6 個之間的 a | a{3,6} | aaa, aaaaaa, aaaa, aaaaaaaa |
| 字串的開始 | ^ ex. ^Darren | ^DarrenYang |
| 字串的結束 | $ ex. Yang$ | DarrenYang$ |
| 位於邊界的字元 | \b ex. \bD | DarrenYang |
| 非位於邊界的字元 | \B ex. \Ba | DarrenYang |
| 配對卻不在群組裡顯示 | John (?:Cena) | John Cena |
| 正向環視 | John (?=Cena) | John Cena |

| | | |
|---|---|---|
| （這位置右邊要出現什麼） | | |
| 正向環視否定<br>（這位置右邊不能出現什麼） | Johnnie (?!Cena) | Johnnie Walker |
| 反向環視<br>（這位置左邊要出現什麼） | (?<=Johnnie) Walker | Johnnie Walker |
| 反向環視否定<br>（這位置左邊不能出現什麼） | (?<!John) Walker | Johnnie Walker |

範例說明：

| 用途 | 正規表達式 | 範例 |
|---|---|---|
| E-mail | [a-zA-Z0-9_]+@[a-zA-Z0-9\._]+ | darren@darreninfo.cc<br>telunyang@gmail.com |
| 英文名字 | [a-zA-Z]+ | Darren<br>Alex |
| 網址 | https:\/\/[a-zA-Z0-9\./?_=]+ | https://darreninfo.cc/?page_id=10 |
| 實數與小數 | [0-9]{1,4}\.[0-9]+ | 9487.94 |

下列表格，我們以課堂上會用到的字串作為範例（實際上不只這些）：

| 範例 |
|---|
| https://store.line.me/stickershop/product/14446/zh-Hant<br>https:\/\/[a-zA-Z0-9./]+\/([0-9]+)\/zh-Hant<br><br>https://stickershop.line-scdn.net/stickershop/v1/sticker/187166590/iPhone/sticker_key@2x.png<br>https:\/\/stickershop\.line-scdn\.net\/stickershop\/v1\/sticker\/([0-9]+)\/(android\|iPhone)\/sticker(?:_key\|)@2x\.png<br><br>https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-1.fna.fbcdn.net 640w<br>(https:\/\/[a-zA-Z0-9_./-]+.jpg\?[a-zA-Z0-9_=.-]+)\s(\d{3,4}w) |

在 Javascript 中的正規表達式用法：

範例

```
let url = 'https://store.line.me/stickershop/product/14446/zh-Hant';

let pattern = /https:\/\/[a-zA-Z0-9./]+\/([0-9]+)\/zh-Hant/g;

let arrMatch = pattern.exec(url);

console.dir(arrMatch);



//輸出下列結果：
[ 'https://store.line.me/stickershop/product/14446/zh-Hant',
  '14446',
  index: 0,
  input: 'https://store.line.me/stickershop/product/14446/zh-Hant',
  groups: undefined ]
```

範例

```
let url = 'https://stickershop.line-
scdn.net/stickershop/v1/sticker/187166590/iPhone/sticker_key@2x.png';

let pattern = /https:\/\/stickershop\.line-scdn\.net\/stickershop\/v1\/sticker\/([0-
9]+)\/(android|iPhone)\/sticker(?:_key|)@2x\.png/g;

let arrMatch = pattern.exec(url);

console.dir(arrMatch);



//輸出下列結果：
[ 'https://stickershop.line-scdn.net/stickershop/v1/sticker/187166590/iPhone/sticker_key@2x.png',
  '187166590',
  'iPhone',
  index: 0,
  input: 'https://stickershop.line-scdn.net/stickershop/v1/sticker/187166590/iPhone/sticker_key@2x.png',
  groups: undefined ]
```

範例

```
let url = 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-1.fna.fbcdn.net 640w';

let pattern = /(https:\/\/[a-zA-Z0-9_./-]+.jpg\?[a-zA-Z0-9_=.-]+)\s(\d{3,4}w)/g;

let arrMatch = pattern.exec(url);

console.dir(arrMatch);


//輸出下列結果：
[ 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-1.fna.fbcdn.net 640w',
  'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-1.fna.fbcdn.net',
  '640w',
  index: 0,
  input: 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-1.fna.fbcdn.net 640w',
  groups: undefined ]
```

補充說明

若是一眼看出來可能在一個字串中，配對出兩個以上的結果，則需要用迴圈來處理：

```
let str = `https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-1.fna.fbcdn.net 640w,
https://instagram.ftpe8-1.fna.fbcdn.net/vp/94ed54d906b8610b258730380d1950bf/5DC4956A/t51.2885-15/sh0.08/e35/p750x750/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-1.fna.fbcdn.net 750w,
```

```
https://instagram.ftpe8-
1.fna.fbcdn.net/vp/91436ad29787dfbd31251cff5caf5622/5DC58B8F/t51.2885-
15/e35/p1080x1080/65114017_173415110333284_3918630239216667279_n.jpg?_nc
_ht=instagram.ftpe8-1.fna.fbcdn.net 1080w`;

let pattern = /(https:\/\/[a-zA-Z0-9_./-]+.jpg\?[a-zA-Z0-9_=.-]+)\s(\d{3,4}w)/g;

while( (arrMatch = pattern.exec(str)) !== null ){
   console.dir(arrMatch);
}



//輸出下列結果：
[ 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-
15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 640w',
  'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-
15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net',
  '640w',
  index: 0,
  input: 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-
15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 640w,\nhttps://instagram.ftpe8-
1.fna.fbcdn.net/vp/94ed54d906b8610b258730380d1950bf/5DC4956A/t51.2885-
15/sh0.08/e35/p750x750/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 750w,\nhttps://instagram.ftpe8-
1.fna.fbcdn.net/vp/91436ad29787dfbd31251cff5caf5622/5DC58B8F/t51.2885-
15/e35/p1080x1080/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 1080w',
  groups: undefined ]

[ 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/94ed54d906b8610b258730380d1950bf/5DC4956A/t51.2885-
15/sh0.08/e35/p750x750/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 750w',
  'https://instagram.ftpe8-1.fna.fbcdn.net/vp/94ed54d906b8610b258730380d1950bf/5DC4956A/t51.2885-
15/sh0.08/e35/p750x750/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net',
  '750w',
  index: 213,
  input: 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-
15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 640w,\nhttps://instagram.ftpe8-
1.fna.fbcdn.net/vp/94ed54d906b8610b258730380d1950bf/5DC4956A/t51.2885-
15/sh0.08/e35/p750x750/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
```

```
1.fna.fbcdn.net 750w,\nhttps://instagram.ftpe8-
1.fna.fbcdn.net/vp/91436ad29787dfbd31251cff5caf5622/5DC58B8F/t51.2885-
15/e35/p1080x1080/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 1080w',
  groups: undefined ]

[ 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/91436ad29787dfbd31251cff5caf5622/5DC58B8F/t51.2885-
15/e35/p1080x1080/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 1080w',
  'https://instagram.ftpe8-1.fna.fbcdn.net/vp/91436ad29787dfbd31251cff5caf5622/5DC58B8F/t51.2885-
15/e35/p1080x1080/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net',
  '1080w',
  index: 426,
  input: 'https://instagram.ftpe8-1.fna.fbcdn.net/vp/dc88f54d1e9366b1ba9b2a63fc9d304e/5DABF66A/t51.2885-
15/sh0.08/e35/p640x640/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 640w,\nhttps://instagram.ftpe8-
1.fna.fbcdn.net/vp/94ed54d906b8610b258730380d1950bf/5DC4956A/t51.2885-
15/sh0.08/e35/p750x750/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 750w,\nhttps://instagram.ftpe8-
1.fna.fbcdn.net/vp/91436ad29787dfbd31251cff5caf5622/5DC58B8F/t51.2885-
15/e35/p1080x1080/65114017_173415110333284_3918630239216667279_n.jpg?_nc_ht=instagram.ftpe8-
1.fna.fbcdn.net 1080w',
  groups: undefined ]
```

## 自動測試工具（網頁瀏覽器自動化工具）介紹

個人比較常用的有：

| 名稱 | 說明 |
|---|---|
| Nightmare | 一個高階的瀏覽器自動化（測試）函式庫，以 Electron（一種可以開發桌面應用程式的工具，可以開發類似 Windows Form，例如音樂播放器、小算盤等應用程式）為基礎。它可以簡單模仿使用者行為，例如點擊按鈕、換頁、在特定欄位輸入文字等。我們藉此來取得瀏覽器操作過後的 html 元素。<br><br>範例程式：<br>`const Nightmare = require('nightmare')`<br>`const nightmare = Nightmare({ show: true })`<br><br>`nightmare`<br>`  .goto('https://duckduckgo.com')`<br>`  .type('#search_form_input_homepage', 'github nightmare')` |

| | |
|---|---|
| | .click('#search_button_homepage')<br>.wait('#r1-0 a.result__a')<br>.evaluate(() => document.querySelector('#r1-0 a.result__a').href)<br>.end()<br>.then(console.log)<br>.catch(error => {<br>console.error('Search failed:', error)<br>})<br><br>網址：http://www.nightmarejs.org/<br>GitHub：https://github.com/segmentio/nightmare |
| Selenium | 為使用者設計的一套瀏覽器自動化的工具集，使用「真正的瀏覽器」來模仿使用者行為，透過 WebDriver 來取得瀏覽器的核心功能，例如執行時引用 chrome、firefox 等，進而模仿使用者行為來操作瀏覽器，取得互動後的 html 元素。<br><br>網址：https://www.seleniumhq.org/ |

手機 APP 也有該領域內的自動化測試工具，例如 Appium。

網址：http://appium.io/

## Nightmare 套件基礎教學

以下我們會介紹本課程教學中常用的功能：

| 引入方式 |
|---|
| //使用 nightmare 套件<br>const Nightmare = require('nightmare')<br><br>//實體化 Nightmare 物件，初始設定為「顯示視窗」、「寬 1280」、「高 800」<br>const nightmare = Nightmare({ show: true, width: 1280, height: 800 }); |

| 標頭設定 |
|---|
| //瀏覽器標頭，讓對方得知我們是人類，而非機器人 (爬蟲)<br>const headers = {<br>'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36', |

```
  'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3',
  'Accept-Language': 'zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7',
};
```

標頭設定資訊來源

| 方法（物件為 nightmare） | 說明與範例 |
|---|---|
| .goto(網址, headers) | 前往指定網址<br><br>await nightmare<br>.goto('https://www.104.com.tw', headers); |
| .type(選擇器, 輸入文字) | 輸入文字<br><br>await nightmare<br>.type('input#ikeyword', '人帥真好'); |
| .wait(毫秒等待時間) | 讓腳本等待一下，單位為毫秒<br><br>await nightmare.wait(2000); |

| | |
|---|---|
| .click(選擇器) | 按下哪個 html 元素，例如按鈕、超連結等<br><br>`await nightmare.click('span#icity');` |
| .catch((err) =>{ 回呼後的程式碼 }) | 補捉到錯誤後的程式區段<br><br>`await nightmare`<br>`.catch(error => {`<br>`  console.error('Search failed: ', error)`<br>`});` |
| .scrollTo( top , left ) | 視窗滾動到指定位置<br><br>`let offset = 0; //總偏移量`<br>`offset += 500;`<br>`await nightmare`<br>`.scrollTo( offset, 0 )`<br>`.wait(500);` |
| .evaluate(() => {<br>執行典型 js，並回傳結果<br>}); | `let html = await nightmare.evaluate(() => {`<br>`  return document.documentElement.innerHTML;`<br>`});` |
| .end((err) =>{<br>回呼後的程式碼<br>}); | 關閉 nightmare<br><br>`await nightmare.end((err) => {`<br>`  if(err) throw err;`<br>`  console.log('Nightmare is close.');`<br>`});` |
| .exists(選擇器) | 確認網頁元素是否存在<br><br>`await nightmare`<br>`.exists('button.b-btn.b-btn--link.js-more-page')` |

## 動態網頁元素走訪展示 – 使用 Nightmare 套件

不同於靜態網頁，動態網頁會隨著使用者操作瀏覽器之際，在不同的時機產生 html 元素，無法在單次透過 cURL 取得相對應的動態元素，於是我們需要瀏覽器自動化的工具，協助我們取得動態產生的資訊，再交給程式擷取、處理與分析。以下我們以 YouTube 為例，進行自動搜尋，而後取得搜尋之後的影音列表。

（圖）連入 YouTube 網站，看到搜尋欄位，我們會用程式協助我們輸入關鍵字

（圖）Nightmare 會將我們的關鍵字輸入在欄位裡，然後按下右側的放大鏡，送出結果

（圖）看到搜尋結果列表

（圖）Nightmare 協助我們按下「篩選器」，再按下類型當中的「視訊」選項

（圖）除了類型中的「視訊」，Nightmare 也會協助我們按下排序依據的「觀看次數」

（圖）排序由高至低

（圖）Nightmare 會協助我們不斷滾動捲軸，往下瀏覽列表，我們也會看到動態產生的新項目

（圖）原先有 20 個搜尋結果

（圖）程式自動滾動之際，會產生新的元素（搜尋結果），滾動到底，將連結整理到陣列中

（圖）陣列中有搜尋結果的影片連結，我們會各別進入，同時取得影片名稱與觀看次數

Nodejs-Html-Parser ▸ output ▸ {} youtube.json ▸ {} 0


```
[
  {
    "img": "https://i.ytimg.com/vi/LpxHlo_Oe0o/hqdefault.jpg",
    "id": "LpxHlo_Oe0o",
    "title": "[歌詞] 張學友 - 只想一生跟你走",
    "link": "https://www.youtube.com/watch?v=LpxHlo_Oe0o",
    "singer": "張學友",
    "pageview": 11530101
  },
  {
    "img": "https://i.ytimg.com/vi/SY2SDAAFvWg/hqdefault.jpg",
    "id": "SY2SDAAFvWg",
    "title": "Jacky Cheung 張學友 [用餘生去愛/The Rest Of Time ]Official 官方 MV",
    "link": "https://www.youtube.com/watch?v=SY2SDAAFvWg",
    "singer": "張學友",
    "pageview": 11328645
  },
  {
    "img": "https://i.ytimg.com/vi/aH_x4i6UdNY/hqdefault.jpg",
    "id": "aH_x4i6UdNY",
    "title": "張學友 · 聽海",
    "link": "https://www.youtube.com/watch?v=aH_x4i6UdNY",
    "singer": "張學友",
    "pageview": 10197142
  },
  {
    "img": "https://i.ytimg.com/vi/gEpjMDlrjcE/hqdefault.jpg",
    "id": "gEpjMDlrjcE",
    "title": "张学友 - 一千个伤心的理由",
    "link": "https://www.youtube.com/watch?v=gEpjMDlrjcE",
    "singer": "張學友",
    "pageview": 7369470
  },
  {
    "img": "https://i.ytimg.com/vi/Ark56_mClLw/hqdefault.jpg",
    "id": "Ark56_mClLw",
    "title": "張學友Jacky《味道+聽海+每次都想呼喊你的名字+眼淚+愛如潮水+原來你什麼都不要+愛很簡單》(Live Version)",
    "link": "https://www.youtube.com/watch?v=Ark56_mClLw",
    "singer": "張學友",
    "pageview": 6747040
```

（圖）連結走訪完後，整理成 JSON 檔案

（圖）同時將整理的資料，存入資料庫

初始化

```
const Nightmare = require('nightmare');
const nightmare = Nightmare({ show: true });
const moment = require('moment');
const util = require('util');
const fs = require('fs');

//將 exec 非同步化 (可以使用 await)
const exec = util.promisify(require('child_process').exec);

//將 MySQL 連線物件的 query 與 end 方法非同步化
const pool = require('../../modules/db_youtube');
pool.query = util.promisify(pool.query);
pool.end = util.promisify(pool.end);

//引入 jQuery 機制
const jsdom = require("jsdom");
const { JSDOM } = jsdom;
const { window } = new JSDOM();
const $ = require('jquery')(window);

//關鍵字
const strSinger = '張學友';

//放置影音重要資訊的全域變數 (陣列)
var arrLink = [];

//瀏覽器標頭，讓對方得知我們是人類，而非機器人 (爬蟲)
```

```
const headers = {
  'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36',

'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3',
  'Accept-Language': 'zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7',
};
```

函式

```
//讓程式休息一下，作為瀏覽器執行下一個動作前的緩衝
async function pause(seconds) {
  console.log('Take a break...');
  return new Promise(function (resolve, reject) {
    setTimeout(function () {
      resolve("Go!");
    }, seconds * 1000 ); //這裡的參數為毫秒，所以需要乘上 1000
  });
}
```

函式

```
//進行歌手檢索
async function search(){
  console.log('Ready to search...');

  //輸入關鍵字
  await nightmare
  .goto('https://www.youtube.com/', headers)
  .type('input[id="search"]', strSinger)
  .click('button#search-icon-legacy')
  .catch(error => {
    console.error('Search failed: ', error)
  });

  //讓程式休息一下
  await pause(3).then((str) => { console.log(str); });
}
```

函式

```
//設定篩選條件
async function filter(){
```

```
//按下篩選器
await nightmare
.wait('yt-formatted-string#text.style-scope.ytd-toggle-button-renderer.style-text') //等待篩選器(Filter)出現在網頁上
.click('yt-formatted-string#text.style-scope.ytd-toggle-button-renderer.style-text') //按下篩選器
.catch(error => {
  console.error('Entering filter failed: ', error)
})

//讓程式休息一下
await pause(3).then((str) => { console.log(str); });

//透過 jQuery 來操作當前的網頁的元素(搜尋結果 - 列表頁)
let html = await nightmare
.evaluate(() => {
  return document.documentElement.innerHTML;
}).catch(error => {
  console.error('Getting result failed: ', error)
});;

//取得篩選器中的視訊(Video)文字連結
let linkOfVideoInFilter = $(html)
.find('div#collapse-content > ytd-search-filter-group-renderer.style-scope.ytd-search-sub-menu-renderer')
.find('a#endpoint:contains("Video"), a#endpoint:contains("視訊")')
.attr('href');
linkOfVideoInFilter = 'https://www.youtube.com' + decodeURIComponent(linkOfVideoInFilter);

//前往篩選器中的視訊(Video)文字連結，並取得完整的 html 資料
html = await nightmare
.goto(linkOfVideoInFilter, headers)
.wait('yt-formatted-string#text.style-scope.ytd-toggle-button-renderer.style-text')
.click('yt-formatted-string#text.style-scope.ytd-toggle-button-renderer.style-text')
.evaluate(() => {
  return document.documentElement.innerHTML;
})
.catch(error => {
  console.error('Going to [Video] link failed: ', error)
});

//讓程式休息一下
await pause(5).then((str) => { console.log(str); });

// //取得篩選器中的觀看次數(View count)連結
let linkOfViewCountInFilter = $(html)
.find('div#collapse-content > ytd-search-filter-group-renderer.style-scope.ytd-search-sub-menu-renderer')
.find('a#endpoint:contains("觀看次數"), a#endpoint:contains("View count")')
.attr('href');
linkOfViewCountInFilter = 'https://www.youtube.com' + decodeURIComponent(linkOfViewCountInFilter);
```

```
  //前往篩選器中的觀看次數(View count)連結，
  await nightmare
  .goto(linkOfViewCountInFilter, headers)
  .catch(error => {
    console.error('Going to [View Count] link failed: ', error)
  });
}
```

函式

```
//滾動頁面，將動態資料逐一顯示出來
async function scroll(){
  console.log('Ready to scroll...');

  await pause(2); //休息數秒

  let previousHeight = 0; //先前偏移的高度
  let currentHeight = 0; //目前的高度
  let offset = 0; //總偏移量

  //不斷地 scroll down，直到沒有辦法再往下捲動
  while(offset <= currentHeight) {
    previousHeight = offset;
    currentHeight = await nightmare.evaluate(() => {
      return document.documentElement.scrollHeight; //回傳瀏覽器當前已滾動的高度
    });

    //每次滾動 500 單位的距離。offset 需要累加，才能對應到合適的距離
    offset += 500;
    await nightmare.scrollTo( offset, 0 ).wait(500);

    //列出當前瀏覽器滾動的程度
    console.log('previousHeight = ' + previousHeight + ', currentHeight = ' + currentHeight + ', offset = ' + offset);

    if(offset > 500) break; //滾動一段高度後，強迫跳出迴圈，視情況使用
  }
}
```

函式

```
//分析、整理、收集重要資訊
async function parse(){
  console.log('Ready to parse metadata/elements of YouTube video pages...');

  //取得滾動後，得到動態產生結果的 html 元素，這裡很重要，因為裡面有需要的元素，例如歌名、播放長度等資訊
  let html = await nightmare.evaluate(() => {
```

```
    return document.documentElement.innerHTML;
  });

  //將重要資訊放到陣列中，以便後續儲存；下面範例，將使用正規表達式來擷取我們想要的資訊
  $(html)
  .find('div#contents.style-scope.ytd-item-section-renderer ytd-video-renderer.style-scope.ytd-item-section-
renderer')
  .each((index, element) => {
    let pattern = null;
    let arrMatch = null;
    let obj = {}; //儲放主要資訊的物件

    //縮圖連結 & 影片 ID
    let linkOfImage = $(element).find('img#img.style-scope.yt-img-shadow').attr('src');
    pattern = /https:\/\/i\.ytimg\.com\/vi\/([a-zA-Z0-9_]{11})\/hqdefault\.jpg/g;
    if( (arrMatch = pattern.exec(linkOfImage)) !== null ) {
      obj.img = arrMatch[0]; //縮圖連結
      obj.id = arrMatch[1]; //從連結擷取出來的 video id (watch?v=xxxxxxxx)

      //影片名稱
      let titleOfVideo = $(element)
      .find('a#video-title.yt-simple-endpoint.style-scope.ytd-video-renderer')
      .text();
      titleOfVideo = titleOfVideo.trim(); //去掉左右側的空白
      obj.title = titleOfVideo; //將影片名稱的值，帶到物件的 title 屬性

      //影片連結
      let linkOfVideo = $(element)
      .find('a#video-title.yt-simple-endpoint.style-scope.ytd-video-renderer')
      .attr('href');
      linkOfVideo = 'https://www.youtube.com' + linkOfVideo;
      obj.link = linkOfVideo; //將影片連結的值，帶到物件的 link 屬性

      //歌手名稱
      obj.singer = strSinger;

      //收集、整理各個擷取到的影音連結元素資訊，到全域的陣列變數中
      arrLink.push(obj);
    }
  });
}
```

函式

```
//走訪詳細頁面，看看有無進階資訊可用
async function visit() {
  console.log('Ready to visit YouTube links...');
```

```
  for(let i = 0; i < arrLink.length;i ++){
    //到各個影音連結的詳細頁面，並取得 html
    let html = await nightmare
    .goto(arrLink[i].link, headers)
    .wait('div#count.style-scope.ytd-video-primary-info-renderer')
    .evaluate(() => {
      return document.documentElement.innerHTML;
    });

    //取得觀看次數的文字描述，再用正規表達式取得我們所需要的資訊
    let strViewCount = $(html)
    .find('div#count.style-scope.ytd-video-primary-info-renderer span.view-count.style-scope.yt-view-count-
renderer')
    .text();
    let pattern = /[0-9,]+/g;
    let arrMatch = null;
    let countPageView = 0;
    if( (arrMatch = pattern.exec(strViewCount)) !== null ) {
      strViewCount = arrMatch[0]; //例如「觀看次數：11,272,456 次」
      strViewCount = strViewCount.replace(/,/g, ''); //去除逗號
      countPageView = parseInt(strViewCount); //轉換文字，成為數值，例如「11272456」
      arrLink[i].pageview = countPageView; //將觀看次數的資訊，新增在 i 索引順序中的物件屬性
    }
  }
}
```

函式

```
//將擷取的資訊儲存在資料庫中
async function save(){
  //在這個地方，你可以將資料寫入自己的資料庫，無論是關聯式資料庫，或是 NoSQL (Not Only SQL)，依照您的需求而定
  for(let i = 0; i < arrLink.length;i ++){
    //陣列元素的放置，要與 raw sql statement 的欄位對齊
    let arr = [
      arrLink[i].id,
      arrLink[i].singer,
      arrLink[i].title,
      arrLink[i].img,
      arrLink[i].link,
      arrLink[i].pageview
    ];

    //執行 SQL 語法
    await pool.query('INSERT INTO `songs` (`id`, `singer`, `title`, `img`, `link`, `pageview`) values (?,?,?,?,?,?)', arr)
    .then((results) => {
```

```
        console.log(`${(new moment().format("YYYY-MM-DD HH:mm:ss"))} YouTube ID [${arrLink[i].id}] 的資料 新增 完成。`);
      }).catch((err) => {
        console.log(`${(new moment().format("YYYY-MM-DD HH:mm:ss"))} YouTube ID [${arrLink[i].id}] 的資料 寫入 失敗 !!`);
        console.log(`失敗原因: ${err.code}`);
        console.log(`失敗訊息: ${err.sqlMessage}`);
        console.log();
      });
    }

    //因為我們已經將相關資訊儲存到資料庫，所以可以在這裡先關閉瀏覽器
    await nightmare.end((err) => {
      if(err) throw err;
      console.log('Nightmare is close.');
    });
}
```

函式

```
//照順序執行各個函式
async function asyncArray(functionsList) {
  for(let func of functionsList){
    await func();
  }
}
```

函式

```
//寫入 json 資料
async function write(){
  //建立資料夾
  await fs.mkdir('output', {recursive: true}, async (err) => {
    if(err) throw err;

    //將物件轉成 json 格存，儲存檔案
    await fs.writeFile('output/youtube.json', JSON.stringify(arrLink, null, 2), function(err) {
      if(err) throw err;
      console.log('Saved.');
    });
  });
}
```

| 函式 |
| --- |

```
//關閉相關的功能
async function close() {
  //關閉資料庫連線
  await pool.end(async (err) => {
    if(err) throw err;
    console.log('MySQL connection is close.');
  });
}
```

| 函式 |
| --- |

```
//主程式區域
try {
  asyncArray([search, filter, scroll, parse, visit, save, write, close]).then(async () => {
    console.log('Done');
  });
} catch (err) {
  console.log('try-catch: ');
  console.dir(err, {depth: null});
}
```

# 關聯式資料庫基礎應用

## 資料庫種類介紹

| 種類 | 說明 |
| --- | --- |
| 關聯式資料庫<br>Relational Database | 常見的有 MySQL, MariaDB, PostgreSQL, Microsoft Access, Microsoft SQL Server, Oracle Database, Sybase, FoxPro 等。<br><br>此種資料庫，透過行（放置屬性 Attributes，有時候稱為「欄」）、列（放置屬性值的集合，又稱為「記錄」）的形式，產生結構性的資料，稱為資料表。每一列將各個欄位連結起來，形式一種關係，各個資料表之間，還能透過相同的欄位值域定義，來建立表與表之間的關係。<br><br>每一筆記錄，原則上都會有一個唯一的編號，或是一組具有代表性多個欄位；可能是數字，可能是字元或字串，例如常見的 |

<table>
<tr>
<td></td>
<td>流水號（1,2,3,...）或是具有代表性的編號（例如身分證、員工編號等）。<br><br>

（圖）資料表內部的關係<br><br>

（圖）來源為維基百科「關聯式資料庫」→資料表之間的關係</td>
</tr>
<tr>
<td>非關聯式資料庫<br>NoSQL（Not Only SQL）</td>
<td>常見的有 Google BigTable、Apache Cassandra、MongoDB、CouchDB 等。<br><br>
通常透過「鍵 key：值 value」的形式來儲存資料，儲存的值還可能是另一種「鍵：值」資料的集合，格式類似我們陣列、物件混合使用的結果，也類似我們範例中所儲存的 JSON 格式。<br><br>
請參考先前建立的 JSON 檔案內容。</td>
</tr>
</table>

本課程裡，我們使用「關聯式資料庫」來儲存我們擷取到的網頁資訊。

# XAMPP 安裝

課程中，我們原則上使用 XAMPP 來進行環境安裝與操作。XAMPP 是 Apache、MariaDB、PHP、Perl 等整合在一起的工具，提供 Web、FTP、phpMyAdmin 等服務，Windows、Linux、Mac 環境都能使用。

網址：https://www.apachefriends.org

（圖）可下載不同作業系統版本，也可以切換語系

（圖）等待下載畫面

（圖）下載後的檔案

（圖）不會安裝到 Program Files 路徑下，可以直接按下 OK

（圖）按下一步

（圖）除了預設的 Apache、PHP，僅留下 MySQL、phpMyAdmin 後，按下一步

（圖）使用預設路徑，按下一步

（圖）課程中不會用到架站工具，所以取消上方勾選，按下一步

Setup
Ready to Install
Setup is now ready to begin installing XAMPP on your computer.
XAMPP Installer
< Back
Next >
Cancel

（圖）按下一步，開始安裝

Setup
Welcome to XAMPP!
XAMPP is an easy to install Apache distribution containing MySQL, PHP and Perl
Installing
Unpacking files
XAMPP Installer
< Back
Next >
Cancel

（圖）安裝過程

（圖）勾選上方選項，開啟控制台，按下完成

（圖）選擇美國（英文）後，按儲存

（圖）控制台畫面，請按下 Apache、MySQL 右側的 Start 鍵，啟動服務

（圖）按下 MySQL 右側的 Admin，開啟 phpMyAdmin 網頁

（圖）看到 phpMyAdmin 的畫面，代表安裝成功

（圖）選擇上面的使用者帳號，並按下圖片左下角的連結，新增使用者帳號

| 範例用資料庫帳號、密碼 |
| --- |
| 課堂中所使用的範例，大部分會需要寫入資料庫，我們需要建立一組資料庫的帳號、密碼。<br><br>帳號：test<br>密碼：T1st@localhost |

（圖）登入資訊

（圖）全域權限，請勾選「SELECT, INSERT,UPDATE,DELETE」

（圖）最後按下網頁右下角的執行，建立新帳號

| 補充說明 |
| --- |

資料庫使用者帳號的增修，有很大的學問。課程中，為了教學方便，可以按照前述的流程建立使用者；若是公司有使用到資料庫，帳號管理不能草率，一定要請教有經驗的資料庫管理人員，協助帳號新增與權限設定。

設定 Root 密碼的方式，有兩種，一個是透過 phpMyAdmin 的 SQL 介面、一個是使用 mysql 執行檔，透過命令視窗來執行 SQL 語法。

## phpMyAdmin

(圖) 先開啟 Apache、MySQL，再按下 MySQL 的 Admin 來開啟

(圖) 點選「使用者帳號」連結

(圖) 使用者帳號一覽

這裡有 3 個 root，我們先修改主機名稱為「localhost」的那一個帳號。

(圖) 按下 SQL 連結

(圖) 輸入 / 執行 SQL 的文字欄位

修改特定帳號的密碼：

| SQL 語法 |
| --- |
| **# 選擇資料庫**<br>use mysql;<br><br>**# 設定 特定帳號@主機名稱 = PASSWORD('你的密碼');**<br>SET PASSWORD FOR 'root'@'localhost' = PASSWORD('T1st@localhost');<br><br>**# 在不重新啟動 MySQL 服務的情況下，直接啟用前面的語法執行結果**<br>flush privileges; |

伺服器: 127.0.0.1

資料庫 | SQL | 狀態 | 使用者帳號 | 匯出 | 匯入

在伺服器「127.0.0.1」執行 SQL 查詢:

```
# 選擇資料庫
use mysql;

# 設定 特定帳號@主機名稱 = PASSWORD('你的密碼');
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('T1st@localhost');

# 在不重新啟動 MySQL 服務的情況下，直接啟用前面的語法執行結果
flush privileges;
```

(圖) 修改 root 密碼的 SQL 語法

伺服器: 127.0.0.1

資料庫 | SQL | 狀態 | 使用者帳號 | 匯出 | 匯入 | 設定 | 備援 | 變數

顯示查詢框

MySQL 傳回空的查詢結果 (即0列資料)。 (查詢用了 0.0004 秒。)

```
# 選擇資料庫 use mysql
```

Error: #1046 No database selected

MySQL 傳回空的查詢結果 (即0列資料)。 (查詢用了 0.0036 秒。)

```
# 設定 特定帳號@主機名稱 = PASSWORD('你的密碼'); SET PASSWORD FOR 'root'@'localhost' = PASSWORD('T1st@localhost')
```

Error: #1046 No database selected

MySQL 傳回空的查詢結果 (即0列資料)。 (查詢用了 0.0006 秒。)

```
# 在不重新啟動 MySQL 服務的情況下，直接啟用前面的語法執行結果 flush privileges
```

Error: #1046 No database selected

(圖) root 密碼修正完畢

改完密碼後，我們重新整理 phpMyAdmin 的頁面，會發現連線失敗的畫面。

(圖) 先前的 root 密已經無法使用

我們需要到 phpMyAdmin 裡面的 config.inc.php 去修正登入密碼。

(圖) 選擇 phpMyAdmin (config.inc.php)

我們接下來要將新密碼放到 config.inc.php 裡面。

```
/* Authentication type and info */
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = '';
$cfg['Servers'][$i]['extension'] = 'mysqli';
$cfg['Servers'][$i]['AllowNoPassword'] = true;
$cfg['Lang'] = '';
```

(圖) 將新密碼放到 $cfg['Servers'][$i]['password'] = '你的新密碼'

```
/* Authentication type and info */
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = 'T1st@localhost';
$cfg['Servers'][$i]['extension'] = 'mysqli';
$cfg['Servers'][$i]['AllowNoPassword'] = true;
$cfg['Lang'] = '';
```

(圖) 放入新密碼

(圖) 再刷新 phpMyAdmin 網頁後，恢復到原先的樣子，一樣是 root 帳號

## mysql 執行檔

確認兩件事：

- 有「C:\xampp\mysql\bin」路徑
- 路徑裡面，有幾個執行檔，其中一個叫作「mysql」

首先，我們先在「C:\xampp\mysql\bin」上頭的路徑列，輸入「cmd」。

(圖) 路徑列上，輸入 cmd

(圖) 出現命令提示字元

接下來，我們要輸入剛才修改的 root 帳號與密碼。

| 說明 |
| --- |
| # 使用 mysql 執行檔，使用者名稱為 root，密碼之後會詢問<br>> mysql.exe -u root -p<br>Enter password: ********** |

(圖) 登入成功的畫面

| 指令流程 |
| --- |
| Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.<br>Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.<br>MariaDB [(none)]> use mysql;<br>MariaDB [**mysql**]> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('use2@localhost');<br>Query OK, 0 rows affected (0.004 sec)<br><br>MariaDB [mysql]> flush privileges;<br>Query OK, 0 rows affected (0.000 sec)<br><br>MariaDB [mysql]> |

此時我們回到 phpMyAdmin，重新刷新頁面，便會出現錯誤訊息。

(圖) 連線錯誤的畫面

我們要再一次將新密碼放到 config.inc.php 裡面。

```
/* Authentication type and info */
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = 'use2@localhost';
$cfg['Servers'][$i]['extension'] = 'mysqli';
$cfg['Servers'][$i]['AllowNoPassword'] = true;
$cfg['Lang'] = '';
```

(圖) 輸入新密碼

(圖) 此時又回到正常畫面

## 快速導覽

以 XAMPP 的 MySQL 資料夾為例，路徑在「C:\xampp\mysql」。

(圖) XAMPP 的 MySQL 資料夾結構

(圖) 直接官方網站下載 MariaDB 的資料夾結構

C:\xampp\mysql/data

MySQL 放置「資料」的資料夾，也是日誌（log）被儲存放置的地方。在 data 中，子資料夾（mysql、performance_schema、phpmyadmin、test）也是放置資料庫（database）的地方，在放置資料庫的資料夾中，也存放著這些資料庫所擁有的資料表（tables）。

(圖) 以資料庫(database)為名的資料夾，裡面是資料表(tables)的檔案

一個新建立的資料庫，可能會有以下的檔案格式（以 InnoDB 為例）：

- frm：描述資料表的格式，或許說定義（table definition）
- ibd：放置資料表中的資料，以及關聯的索引資料
- db.opt：設定預設文字編碼和文字排序規則

(圖) address_book

(圖) .frm 和.ibd 檔案結構

備註：**ibd**ata1 是儲存 MySQL InnoDB 引擎的結構、資料、索引和快取

C:\xampp\mysql/data/mysql

放置主要的 mysql 資料庫資料夾，也是管理 MySQL 用的資料庫。

C:\xampp\mysql/share/charset

放置支援字元集的檔案

```
MariaDB [(none)]> use mysql;
Database changed
MariaDB [mysql]> show character set;
+----------+----------------------------+--------------------+--------+
| Charset  | Description                | Default collation  | Maxlen |
+----------+----------------------------+--------------------+--------+
| big5     | Big5 Traditional Chinese   | big5_chinese_ci    |      2 |
| dec8     | DEC West European          | dec8_swedish_ci    |      1 |
| cp850    | DOS West European          | cp850_general_ci   |      1 |
| hp8      | HP West European           | hp8_english_ci     |      1 |
| koi8r    | KOI8-R Relcom Russian      | koi8r_general_ci   |      1 |
| latin1   | cp1252 West European       | latin1_swedish_ci  |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci  |      1 |
| swe7     | 7bit Swedish               | swe7_swedish_ci    |      1 |
| ascii    | US ASCII                   | ascii_general_ci   |      1 |
| ujis     | EUC-JP Japanese            | ujis_japanese_ci   |      3 |
| sjis     | Shift-JIS Japanese         | sjis_japanese_ci   |      2 |
| hebrew   | ISO 8859-8 Hebrew          | hebrew_general_ci  |      1 |
| tis620   | TIS620 Thai                | tis620_thai_ci     |      1 |
| euckr    | EUC-KR Korean              | euckr_korean_ci    |      2 |
| koi8u    | KOI8-U Ukrainian           | koi8u_general_ci   |      1 |
| gb2312   | GB2312 Simplified Chinese  | gb2312_chinese_ci  |      2 |
| greek    | ISO 8859-7 Greek           | greek_general_ci   |      1 |
| cp1250   | Windows Central European   | cp1250_general_ci  |      1 |
| gbk      | GBK Simplified Chinese     | gbk_chinese_ci     |      2 |
| latin5   | ISO 8859-9 Turkish         | latin5_turkish_ci  |      1 |
| armscii8 | ARMSCII-8 Armenian         | armscii8_general_ci |      1 |
| utf8     | UTF-8 Unicode              | utf8_general_ci    |      3 |
| ucs2     | UCS-2 Unicode              | ucs2_general_ci    |      2 |
| cp866    | DOS Russian                | cp866_general_ci   |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak | keybcs2_general_ci |      1 |
| macce    | Mac Central European       | macce_general_ci   |      1 |
| macroman | Mac West European          | macroman_general_ci |      1 |
| cp852    | DOS Central European       | cp852_general_ci   |      1 |
| latin7   | ISO 8859-13 Baltic         | latin7_general_ci  |      1 |
| utf8mb4  | UTF-8 Unicode              | utf8mb4_general_ci |      4 |
| cp1251   | Windows Cyrillic           | cp1251_general_ci  |      1 |
| utf16    | UTF-16 Unicode             | utf16_general_ci   |      4 |
| utf16le  | UTF-16LE Unicode           | utf16le_general_ci |      4 |
| cp1256   | Windows Arabic             | cp1256_general_ci  |      1 |
| cp1257   | Windows Baltic             | cp1257_general_ci  |      1 |
| utf32    | UTF-32 Unicode             | utf32_general_ci   |      4 |
| binary   | Binary pseudo charset      | binary             |      1 |
| geostd8  | GEOSTD8 Georgian           | geostd8_general_ci |      1 |
| cp932    | SJIS for Windows Japanese  | cp932_japanese_ci  |      2 |
| eucjpms  | UJIS for Windows Japanese  | eucjpms_japanese_ci |      3 |
+----------+----------------------------+--------------------+--------+
40 rows in set (0.007 sec)

MariaDB [mysql]>
```

(圖) 使用 SHOW CHARACTER SET; 來看字元集列表

(圖) 從 phpMyAdmin 中一覽字元集

使用指令查詢特定字元集所支援的排序規則

```
MariaDB [mysql]> SHOW COLLATION LIKE 'utf8%';
+-----------------------------+---------+------+---------+----------+---------+
| Collation                   | Charset | Id   | Default | Compiled | Sortlen |
+-----------------------------+---------+------+---------+----------+---------+
| utf8_general_ci             | utf8    |   33 | Yes     | Yes      |       1 |
| utf8_bin                    | utf8    |   83 |         | Yes      |       1 |
| utf8_unicode_ci             | utf8    |  192 |         | Yes      |       8 |
| utf8_icelandic_ci           | utf8    |  193 |         | Yes      |       8 |
| utf8_latvian_ci             | utf8    |  194 |         | Yes      |       8 |
| utf8_romanian_ci            | utf8    |  195 |         | Yes      |       8 |
| utf8_slovenian_ci           | utf8    |  196 |         | Yes      |       8 |
| utf8_polish_ci              | utf8    |  197 |         | Yes      |       8 |
| utf8_estonian_ci            | utf8    |  198 |         | Yes      |       8 |
| utf8_spanish_ci             | utf8    |  199 |         | Yes      |       8 |
| utf8_swedish_ci             | utf8    |  200 |         | Yes      |       8 |
| utf8_turkish_ci             | utf8    |  201 |         | Yes      |       8 |
| utf8_czech_ci               | utf8    |  202 |         | Yes      |       8 |
| utf8_danish_ci              | utf8    |  203 |         | Yes      |       8 |
| utf8_lithuanian_ci          | utf8    |  204 |         | Yes      |       8 |
| utf8_slovak_ci              | utf8    |  205 |         | Yes      |       8 |
| utf8_spanish2_ci            | utf8    |  206 |         | Yes      |       8 |
| utf8_roman_ci               | utf8    |  207 |         | Yes      |       8 |
| utf8_persian_ci             | utf8    |  208 |         | Yes      |       8 |
```

```
| utf8_esperanto_ci          | utf8    | 209 |       | Yes     |      8 |
| utf8_hungarian_ci          | utf8    | 210 |       | Yes     |      8 |
| utf8_sinhala_ci            | utf8    | 211 |       | Yes     |      8 |
| utf8_german2_ci            | utf8    | 212 |       | Yes     |      8 |
| utf8_croatian_mysql561_ci  | utf8    | 213 |       | Yes     |      8 |
| utf8_unicode_520_ci        | utf8    | 214 |       | Yes     |      8 |
| utf8_vietnamese_ci         | utf8    | 215 |       | Yes     |      8 |
| utf8_general_mysql500_ci   | utf8    | 223 |       | Yes     |      1 |
| utf8_croatian_ci           | utf8    | 576 |       | Yes     |      8 |
| utf8_myanmar_ci            | utf8    | 577 |       | Yes     |      8 |
| utf8_thai_520_w2           | utf8    | 578 |       | Yes     |      4 |
| utf8_general_nopad_ci      | utf8    | 1057 |       | Yes     |      1 |
| utf8_nopad_bin             | utf8    | 1107 |       | Yes     |      1 |
| utf8_unicode_nopad_ci      | utf8    | 1216 |       | Yes     |      8 |
| utf8_unicode_520_nopad_ci  | utf8    | 1238 |       | Yes     |      8 |
| utf8mb4_general_ci         | utf8mb4 |  45 | Yes   | Yes     |      1 |
| utf8mb4_bin                | utf8mb4 |  46 |       | Yes     |      1 |
| utf8mb4_unicode_ci         | utf8mb4 | 224 |       | Yes     |      8 |
| utf8mb4_icelandic_ci       | utf8mb4 | 225 |       | Yes     |      8 |
| utf8mb4_latvian_ci         | utf8mb4 | 226 |       | Yes     |      8 |
| utf8mb4_romanian_ci        | utf8mb4 | 227 |       | Yes     |      8 |
| utf8mb4_slovenian_ci       | utf8mb4 | 228 |       | Yes     |      8 |
| utf8mb4_polish_ci          | utf8mb4 | 229 |       | Yes     |      8 |
| utf8mb4_estonian_ci        | utf8mb4 | 230 |       | Yes     |      8 |
| utf8mb4_spanish_ci         | utf8mb4 | 231 |       | Yes     |      8 |
| utf8mb4_swedish_ci         | utf8mb4 | 232 |       | Yes     |      8 |
| utf8mb4_turkish_ci         | utf8mb4 | 233 |       | Yes     |      8 |
| utf8mb4_czech_ci           | utf8mb4 | 234 |       | Yes     |      8 |
| utf8mb4_danish_ci          | utf8mb4 | 235 |       | Yes     |      8 |
| utf8mb4_lithuanian_ci      | utf8mb4 | 236 |       | Yes     |      8 |
| utf8mb4_slovak_ci          | utf8mb4 | 237 |       | Yes     |      8 |
| utf8mb4_spanish2_ci        | utf8mb4 | 238 |       | Yes     |      8 |
| utf8mb4_roman_ci           | utf8mb4 | 239 |       | Yes     |      8 |
| utf8mb4_persian_ci         | utf8mb4 | 240 |       | Yes     |      8 |
| utf8mb4_esperanto_ci       | utf8mb4 | 241 |       | Yes     |      8 |
| utf8mb4_hungarian_ci       | utf8mb4 | 242 |       | Yes     |      8 |
| utf8mb4_sinhala_ci         | utf8mb4 | 243 |       | Yes     |      8 |
| utf8mb4_german2_ci         | utf8mb4 | 244 |       | Yes     |      8 |
| utf8mb4_croatian_mysql561_ci | utf8mb4 | 245 |       | Yes     |      8 |
| utf8mb4_unicode_520_ci     | utf8mb4 | 246 |       | Yes     |      8 |
| utf8mb4_vietnamese_ci      | utf8mb4 | 247 |       | Yes     |      8 |
| utf8mb4_croatian_ci        | utf8mb4 | 608 |       | Yes     |      8 |
| utf8mb4_myanmar_ci         | utf8mb4 | 609 |       | Yes     |      8 |
| utf8mb4_thai_520_w2        | utf8mb4 | 610 |       | Yes     |      4 |
| utf8mb4_general_nopad_ci   | utf8mb4 | 1069 |       | Yes     |      1 |
| utf8mb4_nopad_bin          | utf8mb4 | 1070 |       | Yes     |      1 |
| utf8mb4_unicode_nopad_ci   | utf8mb4 | 1248 |       | Yes     |      8 |
| utf8mb4_unicode_520_nopad_ci | utf8mb4 | 1270 |       | Yes     |      8 |
+----------------------------+---------+------+---------+----------+---------+
67 rows in set (0.000 sec)
```

C:\xampp\mysql/share/{語系}

資料夾包含了各種語系的錯誤訊息

執行檔放置路徑為「C:\xampp\mysql\bin」我們介紹幾個常用執行檔的簡易使用範例。

**mysqldump**

資料庫匯出/備份

1. 備份單一資料庫:

$ mysqldump -u root -p 你的資料庫名稱 > 你的備份名稱.sql;

2. 僅備份特定資料庫下的資料表:

$ mysqldump -u root -p 你的資料庫名稱 Table1 Table2... > 你的備份名稱.sql;

3. 僅匯出資料表結構:

$ mysqldump -u root -p --no-data 你的資料庫名稱 > 你的備份名稱.sql;

4. 備份全部的資料庫:

$ mysqldump -u root -p -A > 你的備份名稱.sql;

| mysql/mysqldump/mysqladmin 與「**登入**」有關的參數說明: |
| --- |
| -u Username<br>用來存取 mysql 資料表的用戶名稱。此參數可省略，若省略，則表示為目前登入 linux 的用戶名稱。<br><br>-p<br>於執行時詢問密碼。 |

**mysql**

登入

> mysql -u 使用者帳號 -p 要求輸入密碼

> mysql -u root -p

Enter Password:

資料庫匯入/還原

1. 建立資料庫，再從備份檔匯入單一資料庫:

$ mysqladmin -u root -p create 你的資料庫名稱;

$ mysql -u root -p 你的資料庫名稱 < 你的備份名稱..sql

2. 從備份檔匯入全部的資料庫 (不需先建立資料庫):

$ mysql -u root -p < 所有資料庫備份檔.sql;

```
C:\xampp\mysql\bin>mysql -u root -p
Enter password: ***************
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 296
Server version: 10.4.8-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

(圖) 登入 mysql 後，出現 Welcome to the MariaDB monitor

安裝完 mysql 後，系統預設會建立一個不需要密碼的 root 用戶，和一個無用戶名、無密碼的匿名用戶（Anonymous Account）。進行下面的初始化操作以合理授權，增強安全。我們可以直接在 phpMyAdmin 裡面，使用 SQL 語法，來刪除匿名帳號。

| SQL 語法 |
| --- |
| **# 選擇資料庫**<br>use mysql;<br><br>**# 移除匿名使用者**<br>DROP USER ''@'localhost';<br>或是<br>DELETE FROM `User` WHERE `User`='';<br><br>**# 在不重新啟動 MySQL 服務的情況下，直接啟用前面的語法執行結果**<br>flush privileges; |

## 資料庫設計議題

資料庫的設計原則如下：

- 確認綱要（schema）中屬性（attributes）的語意（semantic）清楚無誤。
- 減少值組中的重覆資訊。
- 減少值組中的 NULL 值。
- 不允許產生假值組（spurious tuple）

異常（anomalies）原則上可以分成三種：

- 新增異常（insertion anomaly）
- 刪除異常（delection anomaly）
- 修改異常（modification anomaly）

正規化（normalization）是在結構化分析與設計中，建立資料模式所用的技術，一種資料表分割的法則，目的是為了降低資料的重覆性，同時避免異常（anomalies）情況的發生。

(圖) 正規化的常見形式

| 正規化步驟 | 規則 |
|---|---|
| 1NF | ● 每一個欄位只能有一個基元值。<br>● 沒有任何兩筆以上的資料完全重覆。<br>● 資料表中有主鍵，其它欄位相依與主鍵。 |
| 2NF | ● 符合 1NF。<br>● 每一個非鍵的屬性（例如姓名、性別）必須完全相依於主鍵（例如學號）。<br>● 也就是不可以部分功能相依於主鍵。 |
| 3NF | ● 符合 2NF。 |

| | |
|---|---|
| | ● 各欄位與主鍵之間，沒有遞移相依的關係。<br>● 若要找出資料表中的遞移相依性，簡單的方法就正從左到右掃瞄各個欄位，有沒有與主鍵無關的相依性存在。 |
| BCNF | ● 符合 3NF。<br>● 主鍵中的各個欄位（一張表有多個主鍵），不可以相依於其它非主鍵的欄位。 |

(表) 實務上最多進行到 3NF

(圖) 正規化步驟

尚未正規化的資料表：學生選課資料表

| 學號 | 姓名 | 性別 | 暱稱 | 課程代號 | 課程名稱 | 學分數 | 必選修 | 成績 | 老師編號 | 老師姓名 |
|---|---|---|---|---|---|---|---|---|---|---|
| 087 | 楊○○ | 男 | 好人 | C001<br>C002 | 程式設計<br>網頁設計 | 4<br>3 | 必<br>選 | 74<br>93 | T001<br>T002 | 曾○○<br>林○○ |
| 088 | 陳○○ | 女 | 小白 | C002<br>C003<br>C004 | 網頁設計<br>視覺設計<br>網路教學 | 3<br>2<br>4 | 選<br>必<br>選 | 63<br>82<br>94 | T002<br>T003<br>T005 | 林○○<br>王○○<br>謝○○ |

1NF 後的資料表

| 學號 | 姓名 | 性別 | 暱稱 | 課程代號 | 課程名稱 | 學分數 | 必選修 | 成績 | 老師編號 | 老師姓名 |
|---|---|---|---|---|---|---|---|---|---|---|
| 087 | 楊○○ | 男 | 好人 | C001 | 程式設計 | 4 | 必 | 74 | T001 | 曾○○ |
| 087 | 楊○○ | 男 | 好人 | C002 | 網頁設計 | 3 | 選 | 93 | T002 | 林○○ |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 088 | 陳○○ | 女 | 小白 | C002 | 網頁設計 | 3 | 選 | 63 | T002 | 林○○ |
| 088 | 陳○○ | 女 | 小白 | C003 | 視覺設計 | 2 | 必 | 82 | T003 | 王○○ |
| 088 | 陳○○ | 女 | 小白 | C004 | 網路教學 | 4 | 選 | 94 | T005 | 謝○○ |

新增異常：

| 學號 | 姓名 | 性別 | 暱稱 | 課程代號 | 課程名稱 | 學分數 | 必選修 | 成績 | 老師編號 | 老師姓名 |
|---|---|---|---|---|---|---|---|---|---|---|
| NULL | | | | C005 | 系統分析 | | | | NULL | |

若要新增一門課程，要等學生選課、確定授課老師之後，才能新增。

修改異常：

| 學號 | 姓名 | 性別 | 暱稱 | 課程代號 | 課程名稱 | 學分數 | 必選修 | 成績 | 老師編號 | 老師姓名 |
|---|---|---|---|---|---|---|---|---|---|---|
| 087 | 楊○○ | 男 | 好人 | C001 | 程式設計 | 4 | 必 | 74 | T001 | 曾○○ |
| 087 | 楊○○ | 男 | 好人 | **C002** | **網頁設計** | 3 | 選 | ~~93~~ 98 | T002 | 林○○ |
| 088 | 陳○○ | 女 | 小白 | **C002** | **網頁設計** | 3 | 選 | 63 | T002 | 林○○ |
| 088 | 陳○○ | 女 | 小白 | C003 | 視覺設計 | 2 | 必 | 82 | T003 | 王○○ |
| 088 | 陳○○ | 女 | 小白 | C004 | 網路教學 | 4 | 選 | 94 | T005 | 謝○○ |

若是希望有修「網頁設計」的同學，全班加 5 分，卻可能因為加分行為重覆多次，有些人加到、有些人沒加到，而造成資料不一致的異常現象。

刪除異常：

| 學號 | 姓名 | 性別 | 暱稱 | 課程代號 | 課程名稱 | 學分數 | 必選修 | 成績 | 老師編號 | 老師姓名 |
|---|---|---|---|---|---|---|---|---|---|---|
| 087 | 楊○○ | 男 | 好人 | C001 | 程式設計 | 4 | 必 | 74 | T001 | 曾○○ |
| 087 | 楊○○ | 男 | 好人 | C002 | 網頁設計 | 3 | 選 | 93 | T002 | 林○○ |
| 088 | 陳○○ | 女 | 小白 | C002 | 網頁設計 | 3 | 選 | 63 | T002 | 林○○ |
| ~~088~~ | ~~陳○○~~ | ~~女~~ | ~~小白~~ | ~~C003~~ | ~~視覺設計~~ | ~~2~~ | ~~必~~ | ~~82~~ | ~~T003~~ | ~~王○○~~ |
| 088 | 陳○○ | 女 | 小白 | C004 | 網路教學 | 4 | 選 | 94 | T005 | 謝○○ |

當我們刪除學生記錄時，同時也會刪除課程、老師的資訊，造成資料遺失的現象。

## 2NF 後的資料表

將部分功能相依的欄位分割出去，組成新的資料表。

| 學號 | 姓名 | 性別 | 暱稱 | 課程代號 | 課程名稱 | 學分數 | 必選修 | 成績 | 老師編號 | 老師姓名 |
|---|---|---|---|---|---|---|---|---|---|---|
| 087 | 楊○○ | 男 | 好人 | C001 | 程式設計 | 4 | 必 | 74 | T001 | 曾○○ |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 087 | 楊○○ | 男 | 好人 | C002 | 網頁設計 | 3 | 選 | 93 | T002 | 林○○ |
| 088 | 陳○○ | 女 | 小白 | C002 | 網頁設計 | 3 | 選 | 63 | T002 | 林○○ |
| 088 | 陳○○ | 女 | 小白 | C003 | 視覺設計 | 2 | 必 | 82 | T003 | 王○○ |
| 088 | 陳○○ | 女 | 小白 | C004 | 網路教學 | 4 | 選 | 94 | T005 | 謝○○ |

分割出去的結果：

學生資料表

| 學號 | 姓名 | 性別 | 暱稱 |
|---|---|---|---|
| 087 | 楊○○ | 男 | 好人 |
| 088 | 陳○○ | 女 | 小白 |

成績資料表

| 學號 | 課程代號 | 成績 |
|---|---|---|
| 087 | C001 | 74 |
| 087 | C002 | 93 |
| 088 | C002 | 63 |
| 088 | C003 | 82 |
| 088 | C004 | 94 |

課程資料表

| 課程代號 | 課程名稱 | 學分數 | 必選修 | 老師編號 | 老師姓名 |
|---|---|---|---|---|---|
| C001 | 程式設計 | 4 | 必 | T001 | 曾○○ |
| C002 | 網頁設計 | 3 | 選 | T002 | 林○○ |
| C002 | 網頁設計 | 3 | 選 | T002 | 林○○ |
| C003 | 視覺設計 | 2 | 必 | T003 | 王○○ |
| C004 | 網路教學 | 4 | 選 | T005 | 謝○○ |

## 3NF 後的資料表

從左到右掃瞄各個欄位，有沒有與主鍵無關的相依性存在。

老師編號相依於課程代號

| 課程代號 | 課程名稱 | 學分數 | 必選修 | 老師編號 | 老師姓名 |
|---|---|---|---|---|---|
| C001 | 程式設計 | 4 | 必 | T001 | 曾○○ |
| C002 | 網頁設計 | 3 | 選 | T002 | 林○○ |
| C002 | 網頁設計 | 3 | 選 | T002 | 林○○ |
| C003 | 視覺設計 | 2 | 必 | T003 | 王○○ |
| C004 | 網路教學 | 4 | 選 | T005 | 謝○○ |

老師姓名相依於老師編號

老師姓名遞移相依於課程代號

將「老師編號、老師姓名」獨立成一個資料表

老師資料表

| 老師編號 | 老師姓名 |
|---|---|
| T001 | 曾○○ |
| T002 | 林○○ |
| T003 | 王○○ |
| T005 | 謝○○ |

課程資料表

| 課程代號 | 課程名稱 | 學分數 | 必選修 | 老師編號 |
|---|---|---|---|---|
| C001 | 程式設計 | 4 | 必 | T001 |
| C002 | 網頁設計 | 3 | 選 | T002 |
| C002 | 網頁設計 | 3 | 選 | T002 |
| C003 | 視覺設計 | 2 | 必 | T003 |
| C004 | 網路教學 | 4 | 選 | T005 |

資料表正規化完成結果

學生資料表

| 學號 | 姓名 | 性別 | 暱稱 |
|---|---|---|---|
| 087 | 楊○○ | 男 | 好人 |
| 088 | 陳○○ | 女 | 小白 |

成績資料表

| 學號 | 課程代號 | 成績 |
|---|---|---|
| 087 | C001 | 74 |
| 087 | C002 | 93 |
| 088 | C002 | 63 |
| 088 | C003 | 82 |
| 088 | C004 | 94 |

老師資料表

| 老師編號 | 老師姓名 |
|---|---|
| T001 | 曾○○ |
| T002 | 林○○ |
| T003 | 王○○ |

| | |
|---|---|
| T005 | 謝○○ |

課程資料表

| 課程代號 | 課程名稱 | 學分數 | 必選修 | 老師編號 |
|---|---|---|---|---|
| C001 | 程式設計 | 4 | 必 | T001 |
| C002 | 網頁設計 | 3 | 選 | T002 |
| C003 | 視覺設計 | 2 | 必 | T003 |
| C004 | 網路教學 | 4 | 選 | T005 |

# 建立資料庫、資料表

我們將透過 phpMyAdmin 的管理介面，進行資料庫建置。

(圖) 點選資料庫

(圖) 輸入自訂的資料庫名稱，選擇排序規則

| SQL |
|---|
| DROP DATABASE IF EXISTS `my_db`;<br>CREATE DATABASE IF NOT EXISTS `my_db` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;<br>USE `my_db`; |

(圖) 建立資料表，選擇欄位

依照我們先前的範例，我們可以建立正規化後的那四張表。

- students（學生資料表）
- scores（成績資料表）
- teachers（老師資料表）
- courses（課程資料表）

（圖）學生資料表， 6 個欄位

(圖) 學生資料表設計規格

| # | 名稱 | 類型 | 編碼與排序 | 屬性 | 空值(Null) | 預設值 | 備註 | 額外資訊 |
|---|---|---|---|---|---|---|---|---|
| 1 | sId | varchar(3) | utf8mb4_unicode_ci | | 否 | 無 | 學生編號 | |
| 2 | sName | varchar(20) | utf8mb4_unicode_ci | | 否 | 無 | 學生姓名 | |
| 3 | sGender | varchar(1) | utf8mb4_unicode_ci | | 否 | 無 | 學生性別 | |
| 4 | sNickname | varchar(50) | utf8mb4_unicode_ci | | 否 | 無 | 學生暱稱 | |
| 5 | created_at | datetime | | | 否 | current_timestamp() | 新增時間 | |
| 6 | updated_at | datetime | | | 否 | current_timestamp() | 更新時間 | ON UPDATE CURRENT_TIMESTAMP() |

(圖) 學生資料表結構

SQL 語法

```
CREATE TABLE `my_db`.`students` ( `sId` VARCHAR(3) NOT NULL COMMENT '學生編號' , `sName`
VARCHAR(20) NOT NULL COMMENT '學生姓名' , `sGender` VARCHAR(1) NOT NULL COMMENT '學生性別' ,
`sNickname` VARCHAR(50) NOT NULL COMMENT '學生暱稱' , `created_at` DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP COMMENT '新增時間' , `updated_at` DATETIME on update CURRENT_TIMESTAMP
NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '更新時間' , PRIMARY KEY (`sId`)) ENGINE = InnoDB
COMMENT = '學生資料表';

ALTER TABLE `students`
  ADD PRIMARY KEY (`sId`);
COMMIT;
```

(圖) 成績資料表，5 個欄位

(圖) 成績資料表設計規格

| # | 名稱 | 類型 | 編碼與排序 | 屬性 | 空值(Null) | 預設值 | 備註 | 額外資訊 |
|---|---|---|---|---|---|---|---|---|
| 1 | sld | varchar(3) | utf8mb4_unicode_ci | | 否 | 無 | 學生編號 | |
| 2 | cld | varchar(4) | utf8mb4_unicode_ci | | 否 | 無 | 課程編號 | |
| 3 | score | tinyint(3) | | | 否 | 無 | 成績 | |
| 4 | created_at | datetime | | | 否 | current_timestamp() | 新增時間 | |
| 5 | updated_at | datetime | | | 否 | current_timestamp() | 更新時間 | ON UPDATE CURRENT_TIMESTAMP() |

(圖) 成績資料表結構

SQL

```
CREATE TABLE `scores` (
  `sld` varchar(3) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '學生編號',
  `cld` varchar(4) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '課程編號',
  `score` tinyint(3) NOT NULL COMMENT '成績',
  `created_at` datetime NOT NULL DEFAULT current_timestamp() COMMENT '新增時間',
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
COMMENT '更新時間'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='成績資料表';
```

在這裡，我們尚未在「索引」的下拉式選單選擇主鍵（Primary Key），待執行完成後，額外設定。

(圖) 在這裡會看見沒有定義索引的提示

(圖) 選擇學生編號、課程編號，按下「主鍵」圖示

(圖) 看到 sId 和 cId 右側，各有一把主鍵圖示

| SQL |
| --- |
| ALTER TABLE `scores` ADD PRIMARY KEY( `sId`, `cId`); |

(圖) 老師資料表，4 個欄位

(圖) 老師資料表設計規格

| # | 名稱 | 類型 | 編碼與排序 | 屬性 | 空值(Null) | 預設值 | 備註 |
|---|---|---|---|---|---|---|---|
| 1 | tId | varchar(4) | utf8mb4_unicode_ci | | 否 | 無 | 老師編號 |
| 2 | tName | varchar(20) | utf8mb4_unicode_ci | | 否 | 無 | 老師姓名 |
| 3 | created_at | datetime | | | 否 | 無 | 新增時間 |
| 4 | updated_at | datetime | | | 否 | 無 | 更新時間 |

(圖) 老師資料表結構

SQL

```
CREATE TABLE `teachers` (
  `tId` varchar(4) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '老師編號',
  `tName` int(11) NOT NULL COMMENT '老師姓名',
  `created_at` datetime NOT NULL COMMENT '新增時間',
  `updated_at` datetime NOT NULL COMMENT '更新時間'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='老師資料表';


ALTER TABLE `teachers`
  ADD PRIMARY KEY (`tId`);
COMMIT;
```

建立資料表

名稱: courses　欄位數: 7

(圖) 課程資料表，7 個欄位

結構

| 名稱 | 類型 | 長度／值 | 預設值 | 編碼與排序 | 屬性 | 空值 (Null) | A_I | 備註 |
|---|---|---|---|---|---|---|---|---|
| cId 從中央欄位挑選 | VARCHAR | 4 | 無 | utf8mb4_unicode | | | | 課程編號 |
| cName 從中央欄位挑選 | VARCHAR | 10 | 無 | utf8mb4_unicode | | | | 課程名稱 |
| credit 從中央欄位挑選 | TINYINT | 1 | 無 | | | | | 學分數 |
| isCompulsory 從中央欄位挑選 | TINYINT | 1 | 無 | | | | | 必選修 |
| tId 從中央欄位挑選 | VARCHAR | 4 | 無 | utf8mb4_unicode | | | | 老師編號 |
| created_at 從中央欄位挑選 | DATETIME | | 無 | | | | | 新增時間 |
| updated_at 從中央欄位挑選 | DATETIME | | 無 | | | | | 更新時間 |

(圖) 課程資料表設計規格

| SQL |
| --- |
| CREATE TABLE `my_db`.`courses` (`cId` varchar(4) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '課程編號', `cName` varchar(10) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '課程名稱', `credit` tinyint(1) NOT NULL COMMENT '學分數', `isCompulsory` tinyint(1) NOT NULL COMMENT '必選修', `tId` varchar(4) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '老師編號', `created_at` datetime NOT NULL COMMENT '新增時間', `updated_at` datetime NOT NULL COMMENT '更新時間') ENGINE = InnoDB COMMENT = '課程資料表'; |

(圖) 沒有索引的提示

(圖) 勾選 cId、tId 後，按下「主鍵」圖示

| # | 名稱 | 類型 | 編碼與排序 | 屬性 | 空值(Null) | 預設值 | 備註 |
|---|---|---|---|---|---|---|---|
| 1 | cId | varchar(4) | utf8mb4_unicode_ci | | 否 | 無 | 課程編號 |
| 2 | cName | varchar(10) | utf8mb4_unicode_ci | | 否 | 無 | 課程名稱 |
| 3 | credit | tinyint(1) | | | 否 | 無 | 學分數 |
| 4 | isCompulsory | tinyint(1) | | | 否 | 無 | 必選修 |
| 5 | tId | varchar(4) | utf8mb4_unicode_ci | | 否 | 無 | 老師編號 |
| 6 | created_at | datetime | | | 否 | 無 | 新增時間 |
| 7 | updated_at | datetime | | | 否 | 無 | 更新時間 |

(圖) 課程資料表結構

建立索引的目的，在於提升資料查詢的效率，我們可以針對單獨的欄位建立索引，也可以勾選多欄位，成立索引。讓我們檢視學生資料表的的結構，以及它的索引設定（主鍵也是一種索引）。

(圖) 建立索引後的資料儲存方式

students

資料表備註： 學生資料表

| 欄位 | 類型 | 空值 (Null) | 預設值 | 連結到 | 備註 | MIME |
|---|---|---|---|---|---|---|
| sId *(主鍵)* | varchar(3) | 否 | | | 學生編號 | |
| sName | varchar(20) | 否 | | | 學生姓名 | |
| sGender | varchar(1) | 否 | | | 學生性別 | |
| sNickname | varchar(50) | 否 | | | 學生暱稱 | |
| created_at | datetime | 否 | current_timestamp() | | 新增時間 | |
| updated_at | datetime | 否 | current_timestamp() | | 更新時間 | |

索引

| 鍵名 | 類型 | 獨一 | 緊湊 | 欄位 | 基數 | 編碼與排序 | 空值 (Null) | 備註 |
|---|---|---|---|---|---|---|---|---|
| PRIMARY | BTREE | 是 | 否 | sId | 0 | A | 否 | |

我們可以設定單一欄位為索引：

(圖) 單一欄位建立索引

(圖) 除了主鍵，多了一個索引

| SQL – 新增單一欄位索引 |
| --- |
| ALTER TABLE `students` ADD INDEX( `sNickname`); |

| SQL – 刪除單一欄位索引 |
| --- |
| ALTER TABLE `students` DROP INDEX `sNickname`; |

我們也可以設定複數欄位為索引：

| # | 名稱 | 類型 | 編碼與排序 | 屬性 | 空值(Null) | 預設值 | 備註 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | sId | varchar(3) | utf8mb4_unicode_ci | | 否 | 無 | 學生編號 |
| 2 | sName | varchar(20) | utf8mb4_unicode_ci | | 否 | 無 | 學生姓名 |
| 3 | sGender | varchar(1) | utf8mb4_unicode_ci | | 否 | 無 | 學生性別 |
| 4 | sNickname | varchar(50) | utf8mb4_unicode_ci | | 否 | 無 | 學生暱稱 |
| 5 | created_at | datetime | | | 否 | current_timestamp() | 新增時間 |
| 6 | updated_at | datetime | | | 否 | current_timestamp() | 更新時間 |

全選 已選擇項目： 瀏覽 修改 刪除 主鍵 獨一 索引

(圖) 複數欄位建立索引

索引

| 動作 | 鍵名 | 類型 | 獨一 | 緊湊 | 欄位 | 基數 | 編碼與排序 | 空值(Null) | 備註 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 編輯 刪除 | PRIMARY | BTREE | 是 | 否 | sId | 0 | A | 否 | |
| 編輯 刪除 | sName | BTREE | 否 | 否 | sName | 0 | A | 否 | |
| | | | | | sNickname | 0 | A | 否 | |

(圖) 除了主鍵，多了複數欄位索引

| SQL – 新增複數欄位索引 |
| --- |
| ALTER TABLE `students` ADD INDEX( `sName`, `sNickname`); |

| SQL – 刪除複數欄位索引 |
| --- |
| ALTER TABLE `students` DROP INDEX `sName`; |

## 新增、修改、刪除資料

| 格式 |
| --- |
| INSERT INTO `資料表名稱` (`欄位 1`,`欄位 2`,`欄位 3`, … , `欄位 N`)<br>VALUES (值 1, 值 2, 值 3, … , 值 N),<br>(值 1, 值 2, 值 3, … , 值 N),<br>(值 1, 值 2, 值 3, … , 值 N); |

| sId 學生編號 | sName 學生姓名 | sGender 學生性別 | sNickname 學生暱稱 | created_at 新增時間 | updated_at 更新時間 |
| --- | --- | --- | --- | --- | --- |
| 087 | 楊○○ | 男 | 好人 | 2019-12-02 11:11:37 | 2019-12-02 11:11:37 |
| 088 | 陳○○ | 女 | 小白 | 2019-12-02 11:11:37 | 2019-12-02 11:11:37 |

(圖) 學生資料表 – 新增前

| SQL |
| --- |
| INSERT INTO `students` (`sId`,`sName`,`sGender`,`sNickname`)<br>VALUES<br>('003', '王○○', '男', '小王'),<br>('004', '江○○', '女', '小江'),<br>('005', '周○○', '女', '小周'),<br>('006', '黃○○', '男', '小黃'),<br>('007', '丁○○', '男', '小丁'),<br>('008', '鄭○○', '男', '小鄭'); |

| sId 學生編號 | sName 學生姓名 | sGender 學生性別 | sNickname 學生暱稱 | created_at 新增時間 | updated_at 更新時間 |
| --- | --- | --- | --- | --- | --- |
| 003 | 王○○ | 男 | 小王 | 2019-12-02 18:27:16 | 2019-12-02 18:27:16 |
| 004 | 江○○ | 女 | 小江 | 2019-12-02 18:27:16 | 2019-12-02 18:27:16 |
| 005 | 周○○ | 女 | 小周 | 2019-12-02 18:27:16 | 2019-12-02 18:27:16 |
| 006 | 黃○○ | 男 | 小黃 | 2019-12-02 18:27:16 | 2019-12-02 18:27:16 |
| 007 | 丁○○ | 男 | 小丁 | 2019-12-02 18:27:16 | 2019-12-02 18:27:16 |
| 008 | 鄭○○ | 男 | 小鄭 | 2019-12-02 18:27:16 | 2019-12-02 18:27:16 |
| 087 | 楊○○ | 男 | 好人 | 2019-12-02 11:11:37 | 2019-12-02 11:11:37 |
| 088 | 陳○○ | 女 | 小白 | 2019-12-02 11:11:37 | 2019-12-02 11:11:37 |

(圖) 學生資料表 – 新增後

| 格式 |
| --- |
| UPDATE `資料表名稱`<br>SET<br>`欄位 1` = 值 1,<br>`欄位 2` = 值 2,<br>`欄位 3` = 值 3<br>WHERE `欄位 A` = 值 A<br>AND `欄位 B` = 值 B<br>OR `欄位 C` = 值 C |

| tId<br>老師編號 | tName<br>老師姓名 | created_at<br>新增時間 | updated_at<br>更新時間 |
| --- | --- | --- | --- |
| T001 | 曾○○ | 2019-12-02 00:00:00 | 2019-12-02 00:00:00 |
| T002 | 林○○ | 2019-12-02 00:00:00 | 2019-12-02 00:00:00 |
| T003 | 王○○ | 2019-12-02 00:00:00 | 2019-12-02 00:00:00 |
| T005 | 謝○○ | 2019-12-02 00:00:00 | 2019-12-02 00:00:00 |

(圖) 老師資料表 – 修改前

| SQL |
| --- |
| UPDATE<br>  `teachers`<br>SET<br>  `tName` = '黃○○'<br>WHERE<br>  `tId` = 'T001' |

| tId<br>老師編號 | tName<br>老師姓名 |
| --- | --- |
| T001 | 黃○○ |
| T002 | 林○○ |
| T003 | 王○○ |
| T005 | 謝○○ |

(圖) 老師料表 – 修改後

| 格式 |
| --- |
| DELETE FROM `資料表名稱`<br>WHERE `欄位 1` = 值 1,<br>AND `欄位 B` = 值 B<br>OR `欄位 C` = 值 C; |

| sId<br>學生編號 | cId<br>課程編號 | score<br>成績 |
| --- | --- | --- |
| 087 | C001 | 74 |
| 087 | C002 | 93 |
| 088 | C002 | 63 |
| 088 | C003 | 82 |
| 088 | C004 | 94 |

(圖) 成績資料表 – 刪除前

| SQL |
| --- |
| DELETE FROM `scores`<br>WHERE `sId` = '088'<br>AND `cId` = 'C003'; |

(圖) 跳出警示視窗

| sld 學生編號 | cld 課程編號 | score 成績 |
|---|---|---|
| 087 | C001 | 74 |
| 087 | C002 | 93 |
| 088 | C002 | 63 |
| 088 | C004 | 94 |

(圖) 成績資料表 – 刪除後

## 資料庫基本查詢

| 格式 |
|---|
| SELECT *<br>FROM `資料表名稱`<br>WHERE `欄位 A` = 值 1<br>AND `欄位 B` > 值 B,<br>OR `欄位 C` != 值 C<br>ORDER BY `欄位 D` ASC |

| 格式 |
|---|
| SELECT `欄位 1`,`欄位 2`, … ,`欄位 N`<br>FROM `資料表名稱`<br>WHERE `欄位 A` = 值 1<br>AND `欄位 B` > 值 B,<br>OR `欄位 C` != 值 C<br>ORDER BY `欄位 D` ASC |

| 格式 |
|---|
| SELECT `資料庫`.`資料表名稱`.`欄位 1`,<br>`資料庫`.`資料表名稱`.`欄位 2`, … ,<br>`資料庫`.`資料表名稱`.`欄位 N`<br>FROM `資料庫`.`資料表名稱`<br>WHERE `資料庫`.`資料表名稱`.`欄位 A` = 值 1<br>AND `資料庫`.`資料表名稱`.`欄位 B` > 值 B, |

| |
|---|
| OR `資料庫`.`資料表名稱`.`欄位 C` != 值 C<br>ORDER BY `資料庫`.`資料表名稱`.`欄位 D` ASC |

以上課的「my_db」資料為例：

| M0: 列出所有的學生資料 |
|---|
| SELECT *<br>FROM `students` |

| sId<br>學生編號 | sName<br>學生姓名 | sGender<br>學生性別 | sNickname<br>學生暱稱 |
|---|---|---|---|
| 003 | 王○○ | 男 | 小王 |
| 004 | 江○○ | 女 | 小江 |
| 005 | 周○○ | 女 | 小周 |
| 006 | 黃○○ | 男 | 小黃 |
| 007 | 丁○○ | 男 | 小丁 |
| 008 | 鄭○○ | 男 | 小鄭 |
| 087 | 楊○○ | 男 | 好人 |
| 088 | 陳○○ | 女 | 小白 |

(圖) M0 執行結果

| M1: 列出學生的學號、姓名、暱稱 |
|---|
| SELECT `sId`, `sName`, `sNickname`<br>FROM `students` |

| sId<br>學生編號 | sName<br>學生姓名 | sNickname<br>學生暱稱 |
|---|---|---|
| 003 | 王○○ | 小王 |
| 004 | 江○○ | 小江 |
| 005 | 周○○ | 小周 |
| 006 | 黃○○ | 小黃 |
| 007 | 丁○○ | 小丁 |
| 008 | 鄭○○ | 小鄭 |
| 087 | 楊○○ | 好人 |
| 088 | 陳○○ | 小白 |

(圖) M1 執行結果

| M2: 找出老師編號為 T001 或 T003 的老師姓名 |
| --- |
| SELECT `tName`<br>FROM `teachers`<br>WHERE `tId` = 'T001'<br>OR `tId` = 'T003' |

| tName<br>老師姓名 |
| --- |
| 黃○○ |
| 王○○ |

(圖) M2 執行結果

| M3: 將學生依學生排序，號碼最大的排前面，全部列出來 |
| --- |
| SELECT *<br>FROM `students`<br>ORDER BY `students`.`sId` DESC |

| sId 1<br>學生編號 | sName<br>學生姓名 | sGender<br>學生性別 | sNickname<br>學生暱稱 |
| --- | --- | --- | --- |
| 088 | 陳○○ | 女 | 小白 |
| 087 | 楊○○ | 男 | 好人 |
| 008 | 鄭○○ | 男 | 小鄭 |
| 007 | 丁○○ | 男 | 小丁 |
| 006 | 黃○○ | 男 | 小黃 |
| 005 | 周○○ | 女 | 小周 |
| 004 | 江○○ | 女 | 小江 |
| 003 | 王○○ | 男 | 小王 |

(圖) M3 執行結果

| M4: 找出性別為女、暱稱為小白的學生姓名 |
| --- |
| SELECT `sName`<br>FROM `students`<br>WHERE `sGender` = '女' |

```
AND `sNickname` = '小白';
```

M4A: 找出性別為女、暱稱為小白的學生姓名 (用 AS 來替代資料表名稱)

```
SELECT `s`.`sName`
FROM `students` AS `s`
WHERE `s`.`sGender` = '女'
AND `s`.`sNickname` = '小白';
```

補充說明：

`資料表` AS `資料表代稱`，其中 AS 可以省略。

| sName 學生姓名 |
| --- |
| 陳○○ |

(圖) M4、M4A 執行結果

M5: 找出暱稱是「小」開頭的學生姓名與性別

```
SELECT `sName`, `sGender`
FROM `students`
WHERE `sNickname` like '小%'
```

M5A: 找出暱稱是「人」結尾的學生姓名與性別

```
SELECT `sName`, `sGender`
FROM `students`
WHERE `sNickname` like '%人'
```

| sName 學生姓名 | sGender 學生性別 |
| --- | --- |
| 王○○ | 男 |
| 江○○ | 女 |
| 周○○ | 女 |
| 黃○○ | 男 |
| 丁○○ | 男 |
| 鄭○○ | 男 |
| 陳○○ | 女 |

| sName 學生姓名 | sGender 學生性別 |
| --- | --- |
| 楊○○ | 男 |

(圖) M5A 執行結果

(圖) M5 執行結果

| M6: 找出學號（三位數）第二碼為 8 的學生姓名與暱稱 |
| --- |
| SELECT `sName`, `sNickname`<br>FROM `students`<br>WHERE `sId` like '_8_'; |

| sName<br>學生姓名 | sNickname<br>學生暱稱 |
| --- | --- |
| 楊○○ | 好人 |
| 陳○○ | 小白 |

(圖) M6 執行結果

下面的關聯式資料庫設計有些難度，只要理解即可，或是對可以理解的部分練習即可。

| 練習用資料庫建立語法 |
| --- |
| -- phpMyAdmin SQL Dump<br>-- version 4.9.1<br>-- https://www.phpmyadmin.net/<br>--<br>-- 主機： 127.0.0.1<br>-- 產生時間：<br>-- 伺服器版本： 10.4.8-MariaDB<br>-- PHP 版本： 7.3.11<br><br>SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";<br>SET AUTOCOMMIT = 0;<br>START TRANSACTION;<br>SET time_zone = "+00:00";<br><br>--<br>-- 資料庫： `company`<br>--<br><br>-- --------------------------------------------------------<br><br>--<br>-- 資料表結構 `department`<br>--<br><br>CREATE TABLE `department` (<br>  `Dname` varchar(15) COLLATE utf8mb4_unicode_ci NOT NULL,<br>  `Dnumber` int(11) NOT NULL, |

```
  `Mgr_ssn` char(9) COLLATE utf8mb4_unicode_ci NOT NULL,
  `Mgr_start_date` date DEFAULT NULL,
  `created_at` datetime NOT NULL DEFAULT current_timestamp() COMMENT '新增時間',
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
COMMENT '結束時間'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='department';

--
-- 傾印資料表的資料 `department`
--

INSERT INTO `department` (`Dname`, `Dnumber`, `Mgr_ssn`, `Mgr_start_date`, `created_at`, `updated_at`)
VALUES
('Headquarters', 1, '888665555', '1981-06-19', '2019-12-02 21:23:56', '2019-12-02 21:23:56'),
('Administration', 4, '987654321', '1995-01-01', '2019-12-02 21:23:56', '2019-12-02 21:23:56'),
('Research', 5, '333445555', '1988-05-22', '2019-12-02 21:23:56', '2019-12-02 21:23:56');

-- --------------------------------------------------------


--
-- 資料表結構 `dependent`
--

CREATE TABLE `dependent` (
  `Essn` char(9) COLLATE utf8mb4_unicode_ci NOT NULL,
  `Dependent_name` varchar(15) COLLATE utf8mb4_unicode_ci NOT NULL,
  `Sex` char(1) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `Bdate` date DEFAULT NULL,
  `Relationship` varchar(8) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `created_at` datetime NOT NULL DEFAULT current_timestamp() COMMENT '新增時間',
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
COMMENT '修改時間'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='dependent';

--
-- 傾印資料表的資料 `dependent`
--

INSERT INTO `dependent` (`Essn`, `Dependent_name`, `Sex`, `Bdate`, `Relationship`, `created_at`,
`updated_at`) VALUES
('123456789', 'Alice', 'F', '1988-12-30', 'Daughter', '2019-12-02 21:45:14', '2019-12-02 21:45:14'),
('123456789', 'Elizabeth', 'F', '1967-05-05', 'spouse', '2019-12-02 21:45:14', '2019-12-02 21:45:14'),
('123456789', 'Michael', 'M', '1988-01-04', 'Son', '2019-12-02 21:45:14', '2019-12-02 21:45:14'),
('333445555', 'Alice', 'F', '1986-04-05', 'Daughter', '2019-12-02 21:41:37', '2019-12-02 21:41:37'),
('333445555', 'Joy', 'F', '1958-05-03', 'Spouse', '2019-12-02 21:45:14', '2019-12-02 21:45:14'),
('333445555', 'Theodore', 'M', '1983-10-25', 'Son', '2019-12-02 21:41:37', '2019-12-02 21:41:37'),
('987654321', 'Abner', 'M', '1942-02-28', 'Spouse', '2019-12-02 21:45:14', '2019-12-02 21:45:14');

-- --------------------------------------------------------
```

```
--
-- 資料表結構 `dept_locations`
--

CREATE TABLE `dept_locations` (
  `Dnumber` int(11) NOT NULL,
  `Dlocation` varchar(15) COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` datetime NOT NULL DEFAULT current_timestamp() COMMENT '新增時間',
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
COMMENT '更新時間'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci
COMMENT='dept_locations';

--
-- 傾印資料表的資料 `dept_locations`
--

INSERT INTO `dept_locations` (`Dnumber`, `Dlocation`, `created_at`, `updated_at`) VALUES
(1, 'Houston', '2019-12-02 21:25:12', '2019-12-02 21:25:12'),
(4, 'Stafford', '2019-12-02 21:25:12', '2019-12-02 21:25:12'),
(5, 'Bellaire', '2019-12-02 21:25:12', '2019-12-02 21:25:12'),
(5, 'Houston', '2019-12-02 21:25:12', '2019-12-02 21:25:12'),
(5, 'Sugarland', '2019-12-02 21:25:12', '2019-12-02 21:25:12');

-- --------------------------------------------------------

--
-- 資料表結構 `employee`
--

CREATE TABLE `employee` (
  `Fname` varchar(15) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '名',
  `Minit` char(1) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `Lname` varchar(15) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '姓',
  `Ssn` char(9) COLLATE utf8mb4_unicode_ci NOT NULL,
  `Bdate` date DEFAULT NULL COMMENT '生日',
  `Address` varchar(30) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '地址',
  `Sex` char(1) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '性別',
  `Salary` decimal(10,2) DEFAULT NULL COMMENT '薪資',
  `Super_ssn` char(9) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `Dno` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT current_timestamp() COMMENT '新增時間',
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
COMMENT '更新時間'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='EMPLOYEE';

--
-- 傾印資料表的資料 `employee`
--
```

```
INSERT INTO `employee` (`Fname`, `Minit`, `Lname`, `Ssn`, `Bdate`, `Address`, `Sex`, `Salary`, `Super_ssn`, `Dno`, `created_at`, `updated_at`) VALUES
('John', 'B', 'Smith', '123456789', '1965-01-09', '731 Fondren, Houston, TX', 'M', '30000.00', '333445555', 5, '2019-12-02 21:11:36', '2019-12-02 21:11:36'),
('Franklin', 'T', 'Wong', '333445555', '1955-12-08', '638 Voss, houston, TX', 'M', '40000.00', '888665555', 5, '2019-12-02 21:14:28', '2019-12-02 21:14:28'),
('Joyce', 'A', 'English', '453453453', '1972-07-31', '5631 Rice, Houston, TX', 'F', '25000.00', '333445555', 5, '2019-12-02 21:22:14', '2019-12-02 21:22:14'),
('Ramesh', 'K', 'Narayan', '666884444', '1962-09-15', '975 Fire Oak, Humble, TX', 'M', '38000.00', '333445555', 5, '2019-12-02 21:22:14', '2019-12-02 21:22:14'),
('James', 'E', 'Borg', '888665555', '1937-11-10', '450 Stone, Houston, TX', 'M', '55000.00', NULL, 1, '2019-12-02 21:22:14', '2019-12-02 21:22:14'),
('Jennifer', 'S', 'Wallace', '987654321', '1941-06-20', '291 Berry, Bellaire, TX', 'F', '43000.00', '888665555', 4, '2019-12-02 21:22:14', '2019-12-02 21:22:14'),
('Ahmad', 'V', 'Jabbar', '987987987', '1969-03-29', '980 Dallas, Houston, TX', 'M', '250000.00', '987654321', 4, '2019-12-02 21:22:14', '2019-12-02 21:22:14'),
('Alicia', 'J', 'Zelaya', '999887777', '1968-01-19', '3321 Castle, Spring, TX', 'F', '25000.00', '987654321', 4, '2019-12-02 21:14:28', '2019-12-02 21:14:28');

-- --------------------------------------------------------

--
-- 資料表結構 `project`
--

CREATE TABLE `project` (
  `Pname` varchar(15) COLLATE utf8mb4_unicode_ci NOT NULL,
  `Pnumber` int(11) NOT NULL,
  `Plocation` varchar(15) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `Dnum` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT current_timestamp() COMMENT '新增時間',
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp() COMMENT '更新時間'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='project';

--
-- 傾印資料表的資料 `project`
--

INSERT INTO `project` (`Pname`, `Pnumber`, `Plocation`, `Dnum`, `created_at`, `updated_at`) VALUES
('ProductX', 1, 'Bellaire', 5, '2019-12-02 21:36:29', '2019-12-02 21:36:29'),
('ProductY', 2, 'Sugarland', 5, '2019-12-02 21:36:29', '2019-12-02 21:36:29'),
('ProductZ', 3, 'Houston', 5, '2019-12-02 21:36:29', '2019-12-02 21:36:29'),
('Computerization', 10, 'Stafford', 4, '2019-12-02 21:36:29', '2019-12-02 21:36:29'),
('Reorganization', 20, 'Houston', 1, '2019-12-02 21:36:29', '2019-12-02 21:36:29'),
('Newbenefits', 30, 'Stafford', 4, '2019-12-02 21:36:29', '2019-12-02 21:36:29');

-- --------------------------------------------------------

--
```

```
-- 資料表結構 `works_on`
--

CREATE TABLE `works_on` (
  `Essn` char(9) COLLATE utf8mb4_unicode_ci NOT NULL,
  `Pno` int(11) NOT NULL,
  `Hours` decimal(3,1) DEFAULT NULL,
  `created_at` datetime NOT NULL DEFAULT current_timestamp() COMMENT '新增時間',
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
COMMENT '修改時間'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT='works_on';

--
-- 傾印資料表的資料 `works_on`
--

INSERT INTO `works_on` (`Essn`, `Pno`, `Hours`, `created_at`, `updated_at`) VALUES
('123456789', 1, '32.5', '2019-12-02 21:26:47', '2019-12-02 21:26:47'),
('123456789', 2, '7.5', '2019-12-02 21:26:47', '2019-12-02 21:26:47'),
('333445555', 2, '10.0', '2019-12-02 21:29:38', '2019-12-02 21:29:38'),
('333445555', 3, '10.0', '2019-12-02 21:29:38', '2019-12-02 21:29:38'),
('333445555', 10, '10.0', '2019-12-02 21:29:38', '2019-12-02 21:29:38'),
('333445555', 20, '10.0', '2019-12-02 21:29:38', '2019-12-02 21:29:38'),
('453453453', 1, '20.0', '2019-12-02 21:26:47', '2019-12-02 21:26:47'),
('453453453', 2, '20.0', '2019-12-02 21:26:47', '2019-12-02 21:26:47'),
('666884444', 3, '40.0', '2019-12-02 21:26:47', '2019-12-02 21:26:47'),
('888665555', 20, '0.0', '2019-12-02 21:33:12', '2019-12-02 21:33:12'),
('987654321', 20, '15.0', '2019-12-02 21:33:12', '2019-12-02 21:33:12'),
('987654321', 30, '20.0', '2019-12-02 21:33:12', '2019-12-02 21:33:12'),
('987987987', 10, '35.0', '2019-12-02 21:33:12', '2019-12-02 21:33:12'),
('987987987', 30, '5.0', '2019-12-02 21:33:12', '2019-12-02 21:33:12'),
('99887777', 30, '30.0', '2019-12-02 21:29:38', '2019-12-02 21:29:38'),
('999887777', 10, '10.0', '2019-12-02 21:33:12', '2019-12-02 21:33:12');

--
-- 已傾印資料表的索引
--

--
-- 資料表索引 `department`
--
ALTER TABLE `department`
  ADD PRIMARY KEY (`Dnumber`);

--
-- 資料表索引 `dependent`
--
ALTER TABLE `dependent`
  ADD PRIMARY KEY (`Essn`,`Dependent_name`);
```

```
--
-- 資料表索引 `dept_locations`
--
ALTER TABLE `dept_locations`
  ADD PRIMARY KEY (`Dnumber`,`Dlocation`);

--
-- 資料表索引 `employee`
--
ALTER TABLE `employee`
  ADD PRIMARY KEY (`Ssn`);

--
-- 資料表索引 `project`
--
ALTER TABLE `project`
  ADD PRIMARY KEY (`Pnumber`),
  ADD UNIQUE KEY `Pname` (`Pname`);

--
-- 資料表索引 `works_on`
--
ALTER TABLE `works_on`
  ADD PRIMARY KEY (`Essn`,`Pno`);
COMMIT;
```

**EMPLOYEE**

| Fname | Minit | Lname | Ssn | Bdate | Address | Sex | Salary | Super_ssn | Dno |
|---|---|---|---|---|---|---|---|---|---|
| John | B | Smith | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | M | 30000 | 333445555 | 5 |
| Franklin | T | Wong | 333445555 | 1955-12-08 | 638 Voss, Houston, TX | M | 40000 | 888665555 | 5 |
| Alicia | J | Zelaya | 999887777 | 1968-01-19 | 3321 Castle, Spring, TX | F | 25000 | 987654321 | 4 |
| Jennifer | S | Wallace | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX | F | 43000 | 888665555 | 4 |
| Ramesh | K | Narayan | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX | M | 38000 | 333445555 | 5 |
| Joyce | A | English | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX | F | 25000 | 333445555 | 5 |
| Ahmad | V | Jabbar | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX | M | 25000 | 987654321 | 4 |
| James | E | Borg | 888665555 | 1937-11-10 | 450 Stone, Houston, TX | M | 55000 | NULL | 1 |

**DEPARTMENT**

| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
|---|---|---|---|
| Research | 5 | 333445555 | 1988-05-22 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Headquarters | 1 | 888665555 | 1981-06-19 |

**DEPT_LOCATIONS**

| Dnumber | Dlocation |
|---|---|
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Sugarland |
| 5 | Houston |

**WORKS_ON**

| Essn | Pno | Hours |
|---|---|---|
| 123456789 | 1 | 32.5 |
| 123456789 | 2 | 7.5 |
| 666884444 | 3 | 40.0 |
| 453453453 | 1 | 20.0 |
| 453453453 | 2 | 20.0 |
| 333445555 | 2 | 10.0 |
| 333445555 | 3 | 10.0 |
| 333445555 | 10 | 10.0 |
| 333445555 | 20 | 10.0 |
| 999887777 | 30 | 30.0 |
| 999887777 | 10 | 10.0 |
| 987987987 | 10 | 35.0 |
| 987987987 | 30 | 5.0 |
| 987654321 | 30 | 20.0 |
| 987654321 | 20 | 15.0 |
| 888665555 | 20 | NULL |

**PROJECT**

| Pname | Pnumber | Plocation | Dnum |
|---|---|---|---|
| ProductX | 1 | Bellaire | 5 |
| ProductY | 2 | Sugarland | 5 |
| ProductZ | 3 | Houston | 5 |
| Computerization | 10 | Stafford | 4 |
| Reorganization | 20 | Houston | 1 |
| Newbenefits | 30 | Stafford | 4 |

**DEPENDENT**

| Essn | Dependent_name | Sex | Bdate | Relationship |
|---|---|---|---|---|
| 333445555 | Alice | F | 1986-04-05 | Daughter |
| 333445555 | Theodore | M | 1983-10-25 | Son |
| 333445555 | Joy | F | 1958-05-03 | Spouse |
| 987654321 | Abner | M | 1942-02-28 | Spouse |
| 123456789 | Michael | M | 1988-01-04 | Son |
| 123456789 | Alice | F | 1988-12-30 | Daughter |
| 123456789 | Elizabeth | F | 1967-05-05 | Spouse |

| Q0: 查詢名叫「John B. Smith」的員工，他的生日與住址 |
|---|
| SELECT `Bdate`, `Address`<br>FROM `employee`<br>WHERE `Fname` = 'John' AND `Minit` = 'B' AND `Lname` = 'Smith'; |

| Q0A: 查詢名叫「John B. Smith」的員工，他的生日與住址 (欄位使用 * 號) |
|---|
| SELECT *<br>FROM `employee`<br>WHERE `Fname` = 'John' AND `Minit` = 'B' AND `Lname` = 'Smith'; |

| Bdate 生日 | Address 地址 |
|---|---|
| 1965-01-09 | 731 Fondren, Houston, TX |

(圖) Q0 執行結果

| Fname 名 | Minit | Lname 姓 | Ssn | Bdate 生日 | Address 地址 | Sex 性別 | Salary 薪資 | Super_ssn | Dno |
|---|---|---|---|---|---|---|---|---|---|
| John | B | Smith | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | M | 30000.00 | 333445555 | 5 |

(圖) Q0A 執行結果

| Q1: 擷取在 Research 部門工作的所有員工姓名與住址 |
|---|
| SELECT `Fname`, `Lname`, `Address`<br>FROM `employee`, `department`<br>WHERE `Dname` = 'Research' AND `Dnumber` = `Dno`; |

*造成交叉乘積

| Fname 名 | Lname 姓 | Address 地址 |
|---|---|---|
| John | Smith | 731 Fondren, Houston, TX |
| Franklin | Wong | 638 Voss, houston, TX |
| Joyce | English | 5631 Rice, Houston, TX |
| Ramesh | Narayan | 975 Fire Oak, Humble, TX |

(圖) Q1 執行結果

| Q2: 列出所有位在 Stafford 地點的計畫，其計畫編號、部門編號，以及部門經理的姓氏、住址和生日 |
|---|
| SELECT `Pnumber`, `Dnum`, `Lname`, `Address`, `Bdate`<br>FROM `project`,`department`,`employee`<br>WHERE `Dnum` = `Dnumber`<br>AND `Mgr_ssn` = `Ssn`<br>AND `Plocation` = 'Stafford'; |

*造成交叉乘積

| Pnumber | Dnum | Lname 姓 | Address 地址 | Bdate 生日 |
|---|---|---|---|---|
| 10 | 4 | Wallace | 291 Berry, Bellaire, TX | 1941-06-20 |
| 30 | 4 | Wallace | 291 Berry, Bellaire, TX | 1941-06-20 |

(圖) Q2 執行結果

處理模稜兩可的屬性名稱、別名、重新命名與值組變數：

| Q1A: 擷取在 Research 部門工作的所有員工姓名與住址 |
|---|
| SELECT `employee`.`Fname`, `employee`.`Lname`, `employee`.`Address`<br>FROM `employee`, `department`<br>WHERE `department`.`Dname` = 'Research'<br>AND `department`.`Dnumber` = `employee`.`Dno`; |

*造成交叉乘積

| Fname 名 | Lname 姓 | Address 地址 |
|---|---|---|
| John | Smith | 731 Fondren, Houston, TX |
| Franklin | Wong | 638 Voss, houston, TX |
| Joyce | English | 5631 Rice, Houston, TX |
| Ramesh | Narayan | 975 Fire Oak, Humble, TX |

(圖) Q1A 執行結果

| Q8: 擷取每一位員工的姓名與其直屬上司的姓名 |
|---|
| SELECT `E`.`Fname`, `E`.`Lname`, `S`.`Fname`, `S`.`Lname`<br>FROM `employee` AS `E`, `employee` AS `S`<br>WHERE `E`.`Super_ssn` = `S`.`Ssn` |

*造成交叉乘積

| Q8: 擷取每一位員工的姓名與其直屬上司的姓名 (沒有 AS) |
| --- |
| SELECT `E`.`Fname`, `E`.`Lname`, `S`.`Fname`, `S`.`Lname`<br>FROM `employee` `E`, `employee` `S`<br>WHERE `E`.`Super_ssn` = `S`.`Ssn` |

*造成交叉乘積

| Fname | Lname | Fname | Lname |
| --- | --- | --- | --- |
| John | Smith | Franklin | Wong |
| Franklin | Wong | James | Borg |
| Joyce | English | Franklin | Wong |
| Ramesh | Narayan | Franklin | Wong |
| Jennifer | Wallace | James | Borg |
| Ahmad | Jabbar | Jennifer | Wallace |
| Alicia | Zelaya | Jennifer | Wallace |

(圖) Q8 執行結果

| Q9: 在資料庫中選擇所有的 EMPLOYEE Ssn |
| --- |
| SELECT `Ssn`<br>FROM `employee` |

| Ssn |
| --- |
| 123456789 |
| 333445555 |
| 453453453 |
| 666884444 |
| 888665555 |
| 987654321 |
| 987987987 |
| 999887777 |

(圖) Q9 執行結果

| Q10: 選擇 EMPLOYEE Ssn 與 DEPARTMENT Dname 的所有組合 |
| --- |
| SELECT `Ssn`, `Dname`<br>FROM `employee`, `department` |

*造成交叉乘積

| Ssn | Dname |
|---|---|
| 123456789 | Headquarters |
| 123456789 | Administration |
| 123456789 | Research |
| 333445555 | Headquarters |
| 333445555 | Administration |
| 333445555 | Research |
| 453453453 | Headquarters |
| 453453453 | Administration |
| 453453453 | Research |
| 666884444 | Headquarters |
| 666884444 | Administration |
| 666884444 | Research |
| 888665555 | Headquarters |
| 888665555 | Administration |
| 888665555 | Research |
| 987654321 | Headquarters |
| 987654321 | Administration |
| 987654321 | Research |
| 987987987 | Headquarters |
| 987987987 | Administration |
| 987987987 | Research |
| 999887777 | Headquarters |
| 999887777 | Administration |
| 999887777 | Research |

(圖) Q10 執行結果

| Q11: 擷取每一位員工的薪資 |
|---|
| SELECT ALL `Salary`<br>FROM `employee`; |

| Salary 薪資 |
|---|
| 30000.00 |
| 40000.00 |
| 25000.00 |
| 38000.00 |
| 55000.00 |
| 43000.00 |
| 25000.00 |
| 25000.00 |

(圖) Q11 執行結果

| Q11A: 擷取所有不同的薪資值 |
|---|
| SELECT DISTINCT `Salary`<br>FROM `employee` |

| Salary 薪資 |
| --- |
| 30000.00 |
| 40000.00 |
| 25000.00 |
| 38000.00 |
| 55000.00 |
| 43000.00 |

(圖) Q11A 執行結果

| Q4A: 列出姓氏為 Smith 的員工所參與的所有計畫編號，不論該員工是計畫的工人員，或是管理計畫的部門經理 |
| --- |
| (SELECT DISTINCT `Pnumber`<br>FROM `project`,`department`,`employee`<br>WHERE `Dnum` = `Dnumber`<br>AND `Mgr_ssn` = `Ssn`<br>AND `Lname` = 'Smith')<br><br>UNION<br><br>(SELECT DISTINCT `Pnumber`<br>FROM `project`,`works_on`,`employee`<br>WHERE `Pnumber` = `Pno`<br>AND `Essn` = `Ssn`<br>AND `Lname` = 'Smith') |

*造成交叉乘積

| Pnumber |
| --- |
| 1 |
| 2 |

(圖) Q4A 執行結果

# 進階查詢

(圖) 資料表結合的常見態樣

| 格式 |
| --- |
| SELECT `欄位 1`, `欄位 2`<br>FROM `資料表 1` INNER JOIN `資料表 2`<br>ON `資料表 1`.`欄位 A` = `資料表 2`.`欄位 A`<br><br>說明：<br>INNER JOIN 兩個資料表進行內部結合<br>LEFT JOIN 以左邊資料表為主，進行結合<br>RIGHT JOIN 以右邊資料表為主，進行結合<br>OUTTER JOIN |

| M0: 查詢各個課程中的授課老師姓名 |
| --- |
| SELECT `cName`, `tName`<br>FROM `courses` INNER JOIN `teachers` |

```
ON `courses`.`tId` = `teachers`.`tId`
```

| cName 課程名稱 | tName |
|---|---|
| 程式設計 | 黃○○ |
| 網頁設計 | 林○○ |
| 視覺設計 | 王○○ |
| 網路教學 | 謝○○ |

(圖) M0 執行結果

M1: 列出所有擁有成績的學生姓名與課程名稱

```
SELECT `sName`, `cName`, `score`
FROM `scores`
LEFT JOIN `students`
ON `scores`.`sId` = `students`.`sId`
LEFT JOIN `courses`
ON `scores`.`cId` = `courses`.`cId`
```

| sName | cName | score 成績 |
|---|---|---|
| 楊○○ | 程式設計 | 74 |
| 楊○○ | 網頁設計 | 93 |
| 陳○○ | 網頁設計 | 63 |
| 陳○○ | 網路教學 | 94 |

(圖) M1 執行結果

接下來，讓我們使用 JOIN，來修改先前的範例。

**EMPLOYEE**

| Fname | Minit | Lname | Ssn | Bdate | Address | Sex | Salary | Super_ssn | Dno |
|---|---|---|---|---|---|---|---|---|---|
| John | B | Smith | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | M | 30000 | 333445555 | 5 |
| Franklin | T | Wong | 333445555 | 1955-12-08 | 638 Voss, Houston, TX | M | 40000 | 888665555 | 5 |
| Alicia | J | Zelaya | 999887777 | 1968-01-19 | 3321 Castle, Spring, TX | F | 25000 | 987654321 | 4 |
| Jennifer | S | Wallace | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX | F | 43000 | 888665555 | 4 |
| Ramesh | K | Narayan | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX | M | 38000 | 333445555 | 5 |
| Joyce | A | English | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX | F | 25000 | 333445555 | 5 |
| Ahmad | V | Jabbar | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX | M | 25000 | 987654321 | 4 |
| James | E | Borg | 888665555 | 1937-11-10 | 450 Stone, Houston, TX | M | 55000 | NULL | 1 |

**DEPARTMENT**

| Dname | Dnumber | Mgr_ssn | Mgr_start_date |
|---|---|---|---|
| Research | 5 | 333445555 | 1988-05-22 |
| Administration | 4 | 987654321 | 1995-01-01 |
| Headquarters | 1 | 888665555 | 1981-06-19 |

**DEPT_LOCATIONS**

| Dnumber | Dlocation |
|---|---|
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Sugarland |
| 5 | Houston |

**WORKS_ON**

| Essn | Pno | Hours |
|---|---|---|
| 123456789 | 1 | 32.5 |
| 123456789 | 2 | 7.5 |
| 666884444 | 3 | 40.0 |
| 453453453 | 1 | 20.0 |
| 453453453 | 2 | 20.0 |
| 333445555 | 2 | 10.0 |
| 333445555 | 3 | 10.0 |
| 333445555 | 10 | 10.0 |
| 333445555 | 20 | 10.0 |
| 999887777 | 30 | 30.0 |
| 999887777 | 10 | 10.0 |
| 987987987 | 10 | 35.0 |
| 987987987 | 30 | 5.0 |
| 987654321 | 30 | 20.0 |
| 987654321 | 20 | 15.0 |
| 888665555 | 20 | NULL |

**PROJECT**

| Pname | Pnumber | Plocation | Dnum |
|---|---|---|---|
| ProductX | 1 | Bellaire | 5 |
| ProductY | 2 | Sugarland | 5 |
| ProductZ | 3 | Houston | 5 |
| Computerization | 10 | Stafford | 4 |
| Reorganization | 20 | Houston | 1 |
| Newbenefits | 30 | Stafford | 4 |

**DEPENDENT**

| Essn | Dependent_name | Sex | Bdate | Relationship |
|---|---|---|---|---|
| 333445555 | Alice | F | 1986-04-05 | Daughter |
| 333445555 | Theodore | M | 1983-10-25 | Son |
| 333445555 | Joy | F | 1958-05-03 | Spouse |
| 987654321 | Abner | M | 1942-02-28 | Spouse |
| 123456789 | Michael | M | 1988-01-04 | Son |
| 123456789 | Alice | F | 1988-12-30 | Daughter |
| 123456789 | Elizabeth | F | 1967-05-05 | Spouse |

| Q1: 擷取在 Research 部門工作的所有員工姓名與住址 |
| --- |
| ~~SELECT `Fname`, `Lname`, `Address`~~<br>~~FROM `employee`, `department`~~<br>~~WHERE `Dname` = 'Research' AND `Dnumber` = `Dno`;~~<br><br>SELECT `employee`.`Fname`, `employee`.`Lname`, `employee`.`Address`<br>FROM `employee` INNER JOIN `department`<br>ON `employee`.`Dno` = `department`.`Dnumber`<br>WHERE `department`.`Dname` = 'Research' |

| Fname<br>名 | Lname<br>姓 | Address<br>地址 |
| --- | --- | --- |
| John | Smith | 731 Fondren, Houston, TX |
| Franklin | Wong | 638 Voss, houston, TX |
| Joyce | English | 5631 Rice, Houston, TX |
| Ramesh | Narayan | 975 Fire Oak, Humble, TX |

(圖) Q1 執行結果

| Q2: 列出所有位在 Stafford 地點的計畫，其計畫編號、部門編號，以及部門經理的姓氏、住址和生日 |
| --- |
| ~~SELECT `Pnumber`, `Dnum`, `Lname`, `Address`, `Bdate`~~<br>~~FROM `project`,`department`,`employee`~~<br>~~WHERE `Dnum` = `Dnumber`~~<br>~~AND `Mgr_ssn` = `Ssn`~~<br>~~AND `Plocation` = 'Stafford';~~<br><br>SELECT `project`.`Pnumber`, `project`.`Dnum`, `employee`.`Lname`,<br>`employee`.`Address`, `employee`.`Bdate`<br>FROM `project`<br>INNER JOIN `department`<br>ON `project`.`Dnum` = `department`.`Dnumber`<br>INNER JOIN `employee`<br>ON `department`.`Mgr_ssn` = `employee`.`Ssn`<br>AND `project`.`Plocation` = 'Stafford' |

| Pnumber | Dnum | Lname 姓 | Address 地址 | Bdate 生日 |
|---|---|---|---|---|
| 10 | 4 | Wallace | 291 Berry, Bellaire, TX | 1941-06-20 |
| 30 | 4 | Wallace | 291 Berry, Bellaire, TX | 1941-06-20 |

(圖) Q2 執行結果

處理模稜兩可的屬性名稱、別名、重新命名與值組變數：

Q1A: 擷取在 Research 部門工作的所有員工姓名與住址

~~SELECT `employee`.`Fname`, `employee`.`Lname`, `employee`.`Address`~~
~~FROM `employee`, `department`~~
~~WHERE `department`.`Dname` = 'Research'~~
~~AND `department`.`Dnumber` = `employee`.`Dno`;~~

```
SELECT `employee`.`Fname`, `employee`.`Lname`, `employee`.`Address`
FROM `employee` INNER JOIN `department`
ON `department`.`Dnumber` = `employee`.`Dno`
WHERE `department`.`Dname` = 'Research'
```

| Fname 名 | Lname 姓 | Address 地址 |
|---|---|---|
| John | Smith | 731 Fondren, Houston, TX |
| Franklin | Wong | 638 Voss, houston, TX |
| Joyce | English | 5631 Rice, Houston, TX |
| Ramesh | Narayan | 975 Fire Oak, Humble, TX |

(圖) Q1A 執行結果

Q8: 擷取每一位員工的姓名與其直屬上司的姓名

```
SELECT `E`.`Fname`, `E`.`Lname`, `S`.`Fname`, `S`.`Lname`
FROM `employee` AS `E`, `employee` AS `S`
WHERE `E`.`Super_ssn` = `S`.`Ssn`
```

*造成交叉乘積

Q8: 擷取每一位員工的姓名與其直屬上司的姓名 (沒有 AS)

~~SELECT `E`.`Fname`, `E`.`Lname`, `S`.`Fname`, `S`.`Lname`~~
~~FROM `employee` `E`, `employee` `S`~~

```
~~WHERE `E`.`Super_ssn` = `S`.`Ssn`~~

SELECT `E`.`Fname`, `E`.`Lname`, `S`.`Fname`, `S`.`Lname`
FROM `employee` AS `E` INNER JOIN `employee` AS `S`
ON `E`.`Super_ssn` = `S`.`Ssn`
```

| Fname | Lname | Fname | Lname |
|---|---|---|---|
| John | Smith | Franklin | Wong |
| Franklin | Wong | James | Borg |
| Joyce | English | Franklin | Wong |
| Ramesh | Narayan | Franklin | Wong |
| Jennifer | Wallace | James | Borg |
| Ahmad | Jabbar | Jennifer | Wallace |
| Alicia | Zelaya | Jennifer | Wallace |

(圖) Q8 執行結果

EMPLOYEE

| Fname | Minit | Lname | Ssn | Bdate | Address | Sex | Salary | Super_ssn | Dno |
|---|---|---|---|---|---|---|---|---|---|
| John | B | Smith | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | M | 30000 | 333445555 | 5 |
| Franklin | T | Wong | 333445555 | 1955-12-08 | 638 Voss, Houston, TX | M | 40000 | 888665555 | 5 |
| Alicia | J | Zelaya | 999887777 | 1968-01-19 | 3321 Castle, Spring, TX | F | 25000 | 987654321 | 4 |
| Jennifer | S | Wallace | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX | F | 43000 | 888665555 | 4 |
| Ramesh | K | Narayan | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX | M | 38000 | 333445555 | 5 |
| Joyce | A | English | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX | F | 25000 | 333445555 | 5 |
| Ahmad | V | Jabbar | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX | M | 25000 | 987654321 | 4 |
| James | E | Borg | 888665555 | 1937-11-10 | 450 Stone, Houston, TX | M | 55000 | NULL | 1 |

(圖) EMPLOYEE 資料表的內容

由於該資料表同時有員工的 Ssn 和上級 Superior 的 Ssn（Super_ssn），我們可以試著查詢每一個員工的頂頭上司是誰。

Q1: 找查詢每一個員工的頂頭上司是誰，包括員工本身的姓名、Ssn，以及上級的姓名 SSn

```
SELECT `InterE`.`Fname` AS `InterE_Fname`, `InterE`.`Lname` AS `InterE_Lname`,
`InterE`.`Ssn` AS `InterE_Ssn`, `SuperE`.`Fname` AS `SuperE_Fname`,
`SuperE`.`Lname` AS `SuperE_Lname`, `SuperE`.`Ssn` AS `SuperE_Ssn`
FROM `employee` AS `InterE`
LEFT JOIN `employee` AS `SuperE`
ON `InterE`.`Super_ssn` = `SuperE`.`Ssn`
```

| InterE_Fname | InterE_Lname | InterE_Ssn | SuperE_Fname | SuperE_Lname | SuperE_Ssn |
|---|---|---|---|---|---|
| John | Smith | 123456789 | Franklin | Wong | 333445555 |
| Franklin | Wong | 333445555 | James | Borg | 888665555 |
| Joyce | English | 453453453 | Franklin | Wong | 333445555 |
| Ramesh | Narayan | 666884444 | Franklin | Wong | 333445555 |
| James | Borg | 888665555 | *NULL* | *NULL* | *NULL* |
| Jennifer | Wallace | 987654321 | James | Borg | 888665555 |
| Ahmad | Jabbar | 987987987 | Jennifer | Wallace | 987654321 |
| Alicia | Zelaya | 999887777 | Jennifer | Wallace | 987654321 |

(圖) Q1 執行結果

| 補充說明 |
|---|
| from 和 join 均是用於指定需要從哪些表查詢數據。<br><br>● from 可以是一個資料表或多個資料表，如果是多個資料表則是生成一個笛卡爾集，會涉及到大量數據。所以通常在涉及到多個資料表的查詢時，通常通過 join 來拼接多個資料表。故如果資料表的數據行多時，則數據量很多，造成巨大的磁碟、記憶體開銷，當然查詢速度也會很慢。<br>● join 主要是通過多個資料表之間的外鍵關聯來進行拼接，注意用於拼接的鍵需要加上索引，如果沒有則 MySQL 也會默認加上，不過前提是外鍵和引用的主鍵需要是相同的資料型態如數字類型需要是相同的長度和均是有符號或無符號數，字符串類型長度可以不一樣。 |

以下額外補充聚合（aggregation）的概念：

| 格式 1 |
|---|
| SELECT 函式(`欄位`)<br>FROM `資料表`<br><br>備註：也可以加上 WHERE 條件來過濾資料 |

| M1: 查詢所有課程中，最大的學分數是多少 |
|---|
| SELECT MAX(`credit`)<br>FROM `courses` |

| MAX(`credit`) |
|---|
| 4 |

(圖) M1 執行結果

| 格式 2 |
| --- |
| SELECT `欄位 1`,`欄位 2`, 函式(`欄位 3`)<br>FROM `資料表`<br>WHERE `欄位 A` = `欄位 B`<br>GROUP BY `欄位 1`,`欄位 2`<br>HAVING 函式(`欄位 3`) > 值 |

| M2: 查詢各個學分數，相對應的課程各有幾個 |
| --- |
| SELECT `credit`, COUNT(`cId`)<br>FROM `courses`<br>GROUP BY `credit` |

| credit 學分數 | COUNT(`cId`) |
| --- | --- |
| 2 | 1 |
| 3 | 1 |
| 4 | 2 |

(圖) M2 執行結果

## 使用 MySQL 內建函式

| 運算子 | 作用 |
| --- | --- |
| == | 等於 |
| <=> | 安全的等於 |
| <> (!=) | 不等於 |
| <= | 小於等於 |
| >= | 大於等於 |
| > | 大於 |
| IS NULL | 判斷一個值是否為 NULL |
| IS NOT NULL | 判斷一個值是否不為 NULL |
| LEAST | 在有兩個或多個引數時,返回最小值 |
| GREATEST | 當有 2 或多個引數時，返回最大值 |

| | |
|---|---|
| **BETWEEN AND** | 判斷一個值是否落在兩個值之間 |
| **ISNULL** | 與 IS NULL 作用相同 |
| **IN** | 判斷一個值是否落在兩個值之間 |
| **NOT IN** | 判斷一個值不是 IN 列表中的任意一個值 |
| **LIKE** | 萬用字元匹配 |
| **REGEXP** | 正規表示式匹配 |

| 運算子 | 作用 |
|---|---|
| **NOT 或者 !** | 邏輯非 |
| **AND 或者 &&** | 邏輯與 |
| **OR 或者 \|\|** | 邏輯或 |
| **XOR** | 邏輯互斥或 |

| 控制流函式 | 說明 |
|---|---|
| **CASE()** | SELECT CASE column<br>WHEN a THEN b<br>WHEN c THEN d<br>ELSE e<br>END<br><br>若 column 等於 a，則返回 b<br>若 column 等於 c，則返回 d<br>否則返回 e<br>另外，a 與 c 可為運算式，例如 a<10，只要是 a 小於 10 都返回 b |
| **IF()** | SELECT IF(x, y, z)<br>IF()返回一個數字或字符串值。<br>如果 x 是 TRUE (x 不為 0 且 x 不為 NULL)，那麼 IF() 返回 y，否則它返回 z。<br>類似 PHP 中的 三元運算子 |
| **IFNULL()** | SELECT IFNULL(x, y)<br><br>如果 x 是 NULL，IFNULL() 返回 y，否則它返回 x。 |

| | |
|---|---|
| | IFNULL() 回傳一個數字或字串值。 |
| **NULLIF()** | SELECT NULLIF(x, y)<br><br>如果 x 等於 y，則結果回傳 NULL，否則回傳 x。<br><br>效果等同於<br>CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END |

CASE()

```
SELECT `sId`, `cId`, `score`,
CASE
    WHEN `score` >= 90 THEN '成績在 90 分以上'
    WHEN `score` < 90 AND `score` >= 70 THEN '成績在 70 到 89 分之間'
ELSE '成績低於 70'
END AS `msg`
FROM `scores`
```

| sId 學生編號 | cId 課程編號 | score 成績 | msg |
|---|---|---|---|
| 087 | C001 | 74 | 成績在70到89分之間 |
| 087 | C002 | 93 | 成績在90分以上 |
| 088 | C002 | 63 | 成績低於70 |
| 088 | C004 | 94 | 成績在90分以上 |

(圖) CASE() 執行結果

IF()

```
SELECT `sId`, `cId`, `score`, IF(`score` >= 90, "大於 90 分", "低於 90 分") AS `msg`
FROM `scores`;
```

| sId 學生編號 | cId 課程編號 | score 成績 | msg |
|---|---|---|---|
| 087 | C001 | 74 | 低於 90 分 |
| 087 | C002 | 93 | 大於 90 分 |
| 088 | C002 | 63 | 低於 90 分 |
| 088 | C004 | 94 | 大於 90 分 |

(圖) IF() 執行結果

| IFNULL() |
|---|
| SELECT IFNULL(NULL, "It's not NULL") AS `msg`; |

| msg |
|---|
| It's not NULL |

(圖) IFNULL() 執行結果

| NULLIF() |
|---|
| SELECT NULLIF(10, 10) AS `msg`; |

| msg |
|---|
| *NULL* |

(圖) NULLIF() 執行結果

## CRUD 基礎操作

CRUD（Create、Read、Update、Delete）是資料庫存取資料的基本流程，通常會聽到有人說「新增、修改、刪除、查詢」，意思是一樣的。以下表格將簡單說明：

| 名稱 | SQL 關鍵字 | 說明 |
|---|---|---|
| C：Create | insert | 新增記錄（tuple, record）。<br><br>範例：<br>insert into 資料表名稱 (欄位 1, 欄位 2, ...)<br>values (值 1, 值 2, ...); |
| R：Read | select | 將記錄取出。 |

| | | |
|---|---|---|
| | | 範例：<br>select 欄位 1, 欄位 2<br>from 資料表名稱<br>where 條件 1 = 值 1<br>and 條件 2 = 值 2<br>or 條件 3 = 值 3<br>…<br>order by 需要排序的欄位 asc (或 desc); |
| U：Update | update | 更新資料表欄位值。<br><br>範例：<br>update 資料表名稱<br>set 欄位 1=值 1, 欄位 2=值 2<br>where 記錄的主要編號 = 特定編號<br>and 特定欄位 = 特定值<br>or 特定欄位 = 特定值 |
| D：Delete | delete | 刪除記錄。<br><br>範例：<br>delete from 資料表名稱<br>where 欄位 1 = 值 1<br>and 欄位 2 = 值 2<br>… |

## 使用 mysql 套件來存取資料庫

| 初始化 |
|---|
| //基本連線設定<br>const mysql = require('mysql');<br>const pool = mysql.createPool({<br>  connectionLimit : 10,<br>  host  : 'localhost',<br>  user  : '{帳號}',<br>  password : '{密碼}',<br>  database : '{資料庫名稱}, |

```
  supportBigNumbers: true,
  charset: 'UTF8_GENERAL_CI'
});


const util = require('util');


//將 MySQL 連線物件的 query 與 end 方法非同步化
pool.query = util.promisify(pool.query);
pool.end = util.promisify(pool.end);
```

我們使用 sql/youtube.sql 的資料庫結構來練習：

| 用途 | 程式範例 |
| --- | --- |
| 新增 | 目的：新增一筆記錄<br><br>let arr = [<br>  'xyz01234567',<br>  '胖虎',<br>  '【我是孩子王】',<br>  'https://xxx.cc/images/cover.jpg',<br>  'https://xxx.cc/video/abc.mp4',<br>  9487,<br>  '2019-08-28 21:30:00',<br>  '2019-06-26 18:30:00',<br>  '2019-07-31 18:30:00'<br>];<br>await pool.query('INSERT INTO `songs` (`id`, `singer`, `title`, `img`, `link`, `pageview`, `downloaded_at`, `created_at`, `updated_at`) values (?,?,?,?,?,?,?,?,?)', arr)<br>.then((results) => {<br>  console.dir(results);<br>}).catch((err) => {<br>  console.error(err);<br>  console.log(`失敗原因: ${err.code}`);<br>  console.log(`失敗訊息: ${err.sqlMessage}`);<br>});<br><br>正確輸出結果如下：<br>OkPacket {<br>  fieldCount: 0,<br>  affectedRows: 1, |

```
  insertId: 0,
  serverStatus: 2,
  warningCount: 0,
  message: '',
  protocol41: true,
  changedRows: 0 }
```

| id<br>YouTube ID | singer<br>歌手名稱 | title<br>影片名稱 | img<br>縮圖連結 | link<br>影片連結 | pageview<br>觀看次數 | downloaded_at<br>影音下載時間 | created_at<br>新增時間 | updated_at<br>更新時間 |
|---|---|---|---|---|---|---|---|---|
| xyz01234567 | 胖虎 | 【我是孩子王】 | https://xxx.cc/images/cover.jpg | https://xxx.cc/video/abc.mp4 | 9487 | 2019-08-28 21:30:00 | 2019-06-26 18:30:00 | 2019-07-31 18:30:00 |

## 修改

目的：修正 title 欄位

```
let arr = [
  '【我是胖虎】',
  'xyz01234567',
];
await pool.query('UPDATE `songs` SET `title` = ? WHERE `id` = ?', arr)
.then((results) => {
  console.dir(results);
}).catch((err) => {
  console.error(err);
  console.log(`失敗原因: ${err.code}`);
  console.log(`失敗訊息: ${err.sqlMessage}`);
});
```

正確輸出結果如下：

```
OkPacket {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0,
  serverStatus: 2,
  warningCount: 0,
  message: '(Rows matched: 1  Changed: 1  Warnings: 0',
  protocol41: true,
  changedRows: 1 }
```

| id<br>YouTube ID | singer<br>歌手名稱 | title<br>影片名稱 | img<br>縮圖連結 | link<br>影片連結 | pageview<br>觀看次數 | downloaded_at<br>影音下載時間 | created_at<br>新增時間 | updated_at<br>更新時間 |
|---|---|---|---|---|---|---|---|---|
| xyz01234567 | 胖虎 | 【我是胖虎】 | https://xxx.cc/images/cover.jpg | https://xxx.cc/video/abc.mp4 | 9487 | 2019-08-28 21:30:00 | 2019-06-26 18:30:00 | 2019-06-25 19:31:38 |

## 查詢

目的：取得資料

方式 1：

```
let results = await pool.query('select `id`, `link` from `songs` order by `created_at` asc');
console.dir(results, {depth: null});
```

方式 2：

```
await pool.query('select `id`, `link` from `songs` order by `created_at` asc')
.then((results) => {
  console.dir(results, {depth: null});
```

<table>
<tr>
<td></td>
<td>

```
})
.catch((err) => {
  console.error(err);
  console.log(`失敗原因: ${err.code}`);
  console.log(`失敗訊息: ${err.sqlMessage}`);
});
```

正確輸出結果如下：

[ RowDataPacket { id: 'xyz01234567', link: 'https://xxx.cc/video/abc.mp4' } ]
</td>
</tr>
<tr>
<td>刪除</td>
<td>

目的：刪除特定記錄

```
let arr = ['xyz01234567'];

await pool.query('DELETE FROM `songs` WHERE `id` = ?', arr)
.then((results) => {
  console.dir(results, {depth: null});
})
.catch((err) => {
  console.error(err);
  console.log(`失敗原因: ${err.code}`);
  console.log(`失敗訊息: ${err.sqlMessage}`);
});
```

正確輸出結果如下：

```
OkPacket {
 fieldCount: 0,
 affectedRows: 1,
 insertId: 0,
 serverStatus: 2,
 warningCount: 0,
 message: '',
 protocol41: true,
 changedRows: 0 }
```

</td>
</tr>
</table>