

Chapter 2

Python 與 JavaScript之間的銜接橋樑

A Language-Learning Bridge Between Python and JavaScript

~~※本教材不是以簡報為唯一目的，視為講義會更好！

為了方便初學者把握住重點，快速學習。

所以會補充較多字體較小的相關說明，請細細品嘗~~

Instructor: Jiun-Ting Jiang

Email: rjrj0510@gmail.com



上課，儘量往前座哦！

*Dept. of Computer Science and Information Engineering,
Tamkang University*

Outline

- 2-1 : 相似處與相異處
(Similarities and Differences)
- 2-2 : 如何與語言的程式碼互動?
(Interacting with the Code)
- 2-3: 在語言基本工作上的差異
(Basic Bridge Work)
- 2-4 : 從數據可視化的實務角度來看差異
(Differences in Practice)
- 2-5 : 重點快速查閱表 (A Cheat Sheet)

2-1 相似處與相異處(Similarities and Differences)

■ 相似處：

- 都不需經過編譯，就可執行。
(※它們都是直譯式語言(**interpreter**))
- 都有互動式直譯器，所以在逐行輸入程式碼行時，可立即查看執行結果。
- 兩者都有垃圾回收(**garbage collection**)機制。
- 兩種語言都沒有標頭檔(**header file**)。
- 兩者都只要文字編輯器(**text editor**)就可進行開發工作，而非一定要有**IDE**才能開發。
- 這兩種語言，「函數都是一等公民(**first-class citizens**)」，所以函數可當作參數，在函數之間傳遞。

主要不同

- 差異最大的地方是JavaScript是使用單一執行緒執行環境、非阻斷式函數呼叫(**single-threaded and non-blocking**)，非同步式I/O (**asynchronous I/O**)。例如Http請求、讀資料庫...。意味著，如檔案存取的動作，都會有回呼函式(**callback function**)。
- JS原本是只出現在Web開發的，直到近幾年(有了node.js之後)才逐漸脫離瀏覽器的牢籠。而Python則在各個領域皆可見到其身影。
- JS是網路瀏覽器上的唯一語言，反之，Python的情況則是就算有瀏覽器的翻譯器，也還會有很多問題。
- Python具有完善的標準庫，而JS則僅有限(極為受限)的輔助物件程式（例如JSON , Math）。
- Python具有相當典型的物件導向的類別(**object-oriented classes**)，而JS則是沒有類別(**classless**)的語言，是以原型為基礎(**Prototype-based programming**)去定義物件的鏈結關係。
- JS相比於python，只有較少的通用數據處理庫(**general-purpose data-processing**)。
- 由此處的差異，所以本課程強調必須是雙語！
 - 要在瀏覽器上呈現互動視覺化數據，就需要JS
 - 要有通用功能的數據處理與分析程式庫，就需要Python。

※ JavaScript的規範 ECMAScript (ES5, ES6 ...)

在1996年，一個名為ECMA（歐洲計算機製造商協會）定了名為ECMAScript(ES)的標準規範讓瀏覽器供應商遵守，就可以在確保網頁在不同Web瀏覽器之間的互操作性。

關於這個手稿語言規範：

- 在2009年12月：發表ES5：所以是長時期大量被使用的規範！
- 在2015年 6月 發表ES6

- (ES2015, ECMAScript 2015)，此版本引入的語法功能很多，也不斷提交新功能，此後，就讓標準的升級成為常規流程。
- 例如：開始添加類別和模組的語法，但是類別(class)其實是語法糖，本質仍是prototype-based. (※增箭頭函數Arrow function =>,效果約當python的lambda)

□ 2016~2020每年都公佈新加概念和語言特性的規範ES7, ES8, ES9, ES10, ES11

■ ※註：JS容易發生的class誤解：

- 在 JavaScript 的物件導向是Prototype-Based，不區分 Class 和 Object 所有的東西都可以是 Object。不像其他Class-based的物件導向語言，會先class才可以new出object!
- 因此，就算 ES6增加一個 Class保留字用來當 Constructor 創建 Instance，也不代表它物件導向的方式會變成 Class-Based，換言之，是被 Class 包裝的 Prototype-Based 而已。所以千萬不要搞混囉！
- ES6的Class，只是簡化了JavaScript中操作 Constructor 的語法糖。

■ ※註：ES7的新指令 Async 和 Await，可以把層層callback才能完成的流程，輕鬆的進行扁平化處理！

■ ES2021：新增 String.prototype.replaceAll() 和 Promise.any()。

■ ES2022：新增 Array.prototype.at()、Object.hasOwn() 等。

■ ES2023：新增 Array.prototype.toReversed() 等不可變陣列方法

※Python 增強規範(PEP, Python Enhancement Proposal) (或稱為：Python改進建議書)

- 第一個PEP誕生於2000年，由python官網，可查詢：到2018年為止，已擁有478個“兄弟姐妹”，到2020年(編號到PEP8101)已擁有528個兄弟姐妹。
- 其中的**PEP8 -- Style Guide for Python Code** (Python 程式碼風格指引) 是編碼必讀的規範，以下只列舉較重要的幾點：
 - 關於程式碼編排：
 - 縮進規範：PEP 8 規範告訴我們，請選擇四個空格的縮進，不要使用Tab，更不要Tab和空格混用。第二個要注意的是，每行最大長度請限制在 79 個字符。
 - 空行規範：PEP 8 規定，全局的類和函數的上方需要空兩個空行，而類的函數之間需要空一個空行。
 - 空格規範：
 - 函數的參數列表中，調用函數的參數列表中會出現逗號，請注意逗號後要跟一個空格，這是英語的使用習慣，也能讓每個參數獨立閱讀，更清晰。
 - 冒號後面也要跟一個空格。
 - 在#後、注釋前加一個空格。
 - 操作符，例如+，-，*，/，&，|，=，==，!=，請在兩邊都保留空格。不過與此對應，括號內的兩端並不需要空格。
 - 檔案編碼：Python3 發佈版的核心程式碼中應該始終使用UTF-8.

JavaScript最新趨勢

- 類型安全與 **TypeScript** 雖然不是 ECMAScript 的標準，但 **TypeScript** 已經成為前端開發的熱門選擇。它是一種由微軟開發的語言，是 JavaScript 的超集，為其增加了靜態型別。這意味著在開發階段就能檢查出許多潛在的錯誤，特別是在大型專案中，能顯著提升程式碼的可靠性和可維護性。
- **WebAssembly (Wasm)** WebAssembly 是一種低階語言，可以在瀏覽器中執行，其執行速度接近原生程式碼。雖然它不是 JavaScript，但它和 JavaScript 可以無縫協作

視覺化時，好用到函式庫與框架：

- **Chart.js, Plotly.js, ECharts** 等：這些函式庫提供了現成的圖表類型，讓開發者能快速建立常見的長條圖、圓餅圖、散佈圖等，大幅降低了視覺化門檻。它們也持續與新的 JavaScript 語法和模組系統整合，讓使用體驗更流暢。
- 前端框架：現代前端框架如 **React**、**Vue** 和 **Angular** 都與 **ECMAScript** 的模組化、類別等新特性緊密結合。
- 全端(**full-stack**)框架**next.js**：安裝**node.js**後可生成 **next.js** 全框架，不僅繼承 **React**所有特性，還額外提供了許多功能，例如採用基於檔案系統的路由 (**Route**). 此框架推薦的首頁是 **page.tsx**.
(※註：**tsx**是 **TypeScript + JSX**的意思)

全端框架 Next.js 的現況

- **Next.js**是重要的全端框架，但其開發模式已發生重大變化。
 - 更新路由系統：**Next.js** 在 2023 年引入了新的**App Router**（app 目錄），這是目前官方推薦的開發方式。例如：`page.tsx`。
 - **伺服器元件（React Server Components）**：是**App Router**的核心概念，它允許開發者在伺服器端渲染網頁內容，可以大幅提升效能、減少**JavaScript** 程式碼的下載量，並改善 SEO。

※補充：Next.js快速入門：

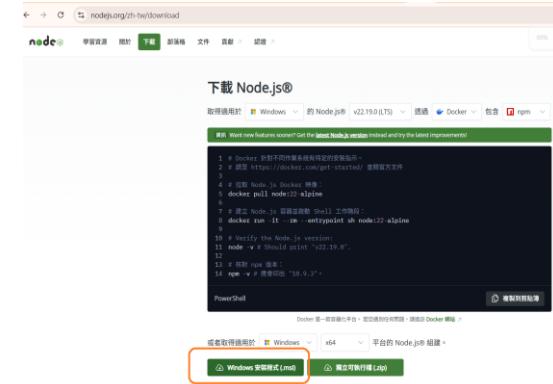
■ Step 1. 安裝 Node.js

- Next.js 需要 Node.js 環境。

■ Step 2. 建立 Next.js 專案

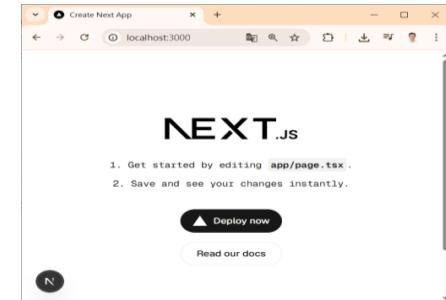
- 打開終端機，輸入：

- `npx create-next-app my-app`
(`my-app` 是資料夾名稱，可改成想要的名字，例如 `hello-vercel`)
- 過程中會問一些問題（新手可以直接一路按 **Enter** 採用預設值）。
- 完成後，進入資料夾: `cd my-app`



■ Step 3. 啟動本地開發伺服器

- 在專案資料夾裡輸入：`npm run dev`
- 打開瀏覽器輸入 `http://localhost:3000`，你會看到 Next.js 的首頁。
- 完成next.js可佈置到git-hub並同步到vercel，成為永久網站
- 例如：<https://my-app2-topaz.vercel.app/>



Exercise 2-1

- Ex2-1Add1: Python與JavaScript有那些相似處？有那些相異處，至少各舉出三點以上。
- Ex2-1Add2: JavaScript的主要規範是什麼？簡介之。
- Ex2-1Add3: 有關Python 增強規範(或稱為Python改進建議書)是什麼？簡介之。
- Ex2-1Add4: JavaScript在ES6加入新符號「=>」作用是什麼？Python在3.0加入「->」作用與JS不同是什麼？簡單說明之。
- Ex2-1Add5: 2015的年底Python3.5增加兩個關鍵字async,await。有趣的情況，在2016的ES7, JavaScript 也有加入新關鍵字async和await。請簡單說明其作用。

2-2: 如何與語言程式碼互動? (Interacting with the Code)

- Python和JavaScript的一大優勢在於，由於它們是即時進行解譯(interpret)的，因此您可以在開發程式時，與執行中的程式碼進行互動。
- Python的解釋器可以從命令行運行，而JavaScript的解釋器通常可以通過從網路瀏覽器的內置開發工具。
- 在本部分中，我們將介紹如何啟動與解釋器的會話並開始嘗試您的代碼。

Python的互動方式

- 安裝官方版python，就會有簡單易學的IDLE可以使用，包含Python shell:
 - IPython是一種基於Python的互動式直譯器。相較於原生的Python Shell，IPython提供了更為強大的編輯和「互動」功能。
 - IPython筆記本被引入Jupyter. 可以使用以下命令從命令行啟動Jupyter筆記本：
 - \$ jupyter notebook
 - (在PC上，強烈建議是安裝完Anaconda，就會連python, jupyter notebook等等一大堆常用數據分析工具一齊安裝好)
 - 線上直接寫程式，例如：<http://repl.it>

The screenshot shows the Python 3.8.2 Shell interface. The title bar reads "Python 3.8.2 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python version information: "Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32". Below this, it says "Type "help", "copyright", "credits" or "license()" for more information". A script window titled "test.py - C:/Users/rr/AppData/Local/Pr..." is open, showing the following Python code:

```
sum=0
for i in range(10):
    sum+=i
print(sum)
```

The code is highlighted in different colors: sum, i, and print are in blue, while the numbers and the range function are in black. The script window has its own menu bar with File, Edit, Format, Run, Options, Window, and Help. The status bar at the bottom right of the script window shows "Ln: 5 Col: 0".

A screenshot of a Windows desktop environment. In the center, a web browser window is open to 'localhost:8890/tree', displaying a Jupyter notebook interface with several code cells and their outputs. The browser's address bar shows the URL. Below the browser, a taskbar lists several open applications: 車用程式 (Vehicle Application), gMeet_議端 (gMeet_Session), Jcase_liu_華ML (Jcase_liu_HuaML), pyDataViz (pyDataViz), FLA(2020au) (FLA(2020au)), sphero (sphero), and micro (micro). On the far left of the taskbar, the Jupyter logo is visible. A system tray icon for a network connection is also present.

The screenshot shows a browser-based Python code editor and interpreter. The URL is `repl.it/repls/LoathsomeUselessShockwave#main.py`. The code in `main.py` is:

```
name = input('What is your name?\n')
print('Hi, %s.' % name)
sum=0
for i in range(10):
    sum+=i
print(sum)
```

The output window on the right shows the following interaction:

```
What is your name?
江俊廷
Hi, 江俊廷.
45
> 2+3
5
> 1+2+3+4+5+6+7+8+9
45
> |
```

JavaScript 的互動方式

- 有很多情況，可以在不啟動服務器下試用JavaScript代碼，儘管啟動服務器並不難。
- 因為JavaScript解釋器嵌入在所有現代Web瀏覽器中，所以有許多網站可以讓您在網路上，直接測試JavaScript以及HTML和CSS並查看結果。
- 可到線上程式撰寫系統測試程式，像JSBin: <https://jsbin.com/> , <https://jsfiddle.net/>, <https://repl.it/> ...等。
 - 這些網站非常適合共享代碼和試用代碼段，並且通常允許您直接添加D3.js之類的庫。
- 如果您想測試單行代碼或測試即時程式碼的狀態，基於瀏覽器的控制台是您的最佳選擇。可打開Chrome，用 [...] / [更多工具] / [開發人員工具] (也可用組合鍵Ctrl-Shift-I 或 J)，來訪問控制台。
 - 除了嘗試一些JS片段外，該控制台還允許您深入研究範圍內的任何物件對象，從而揭示它們的方法和屬性。
 - 這也是查詢活動對象的狀態並蒐索錯誤的好方法。
 - 甚至可進行JavaScript的除錯(要先設好行號或事件中斷點)。
- 使用在線JavaScript編輯器的一個缺點，是失去了您最喜歡的編輯環境的功能，包括某些程式分析工具或熟悉的鍵盤快捷鍵等。

在線編輯 vs 本地服務器

- 可以這麼說：
 - 在線編輯是解決初級或初入門的問題
 - 如果將進行廣泛的**JavaScript**互動並希望使用自己喜歡的編輯器，那麼最好的選擇是運行本地服務器。
- 方法1：首先，創建一個項目目錄（例如，稱為**ch02_js**入門），並添加一個包含**JS**腳本的最小**HTML**文件：



- 可先執行anaconda prompt 再切換到剛創的目錄打下列指令 \$**python -m http.server 8000**

● 【check1說明】Web之(html+javascript)實驗的三步驟 (※採用方法：在本地架一個服務器:8000)

- (1)先在 桌面，建立一個 新資料夾，例如 webtest20250926
 - 並且建立 index.html 與 script.js 兩個檔案。(※可由文字檔複製，不必自己打字！)
- (2)執行anaconda prompt 再切換到剛建立的目錄，並打下列指令
 - \$python -m http.server 8000
- (3)接下來：到chrome連到 <http://localhost:8000/>

●【check1】 ●ch02實作練習1：Lab#ch02-1a_js入門localhost(請在div area01後面的01, 與var data裡 ,0] 的0, 改為自己的學號末四碼)_要在本地架一個服務器。

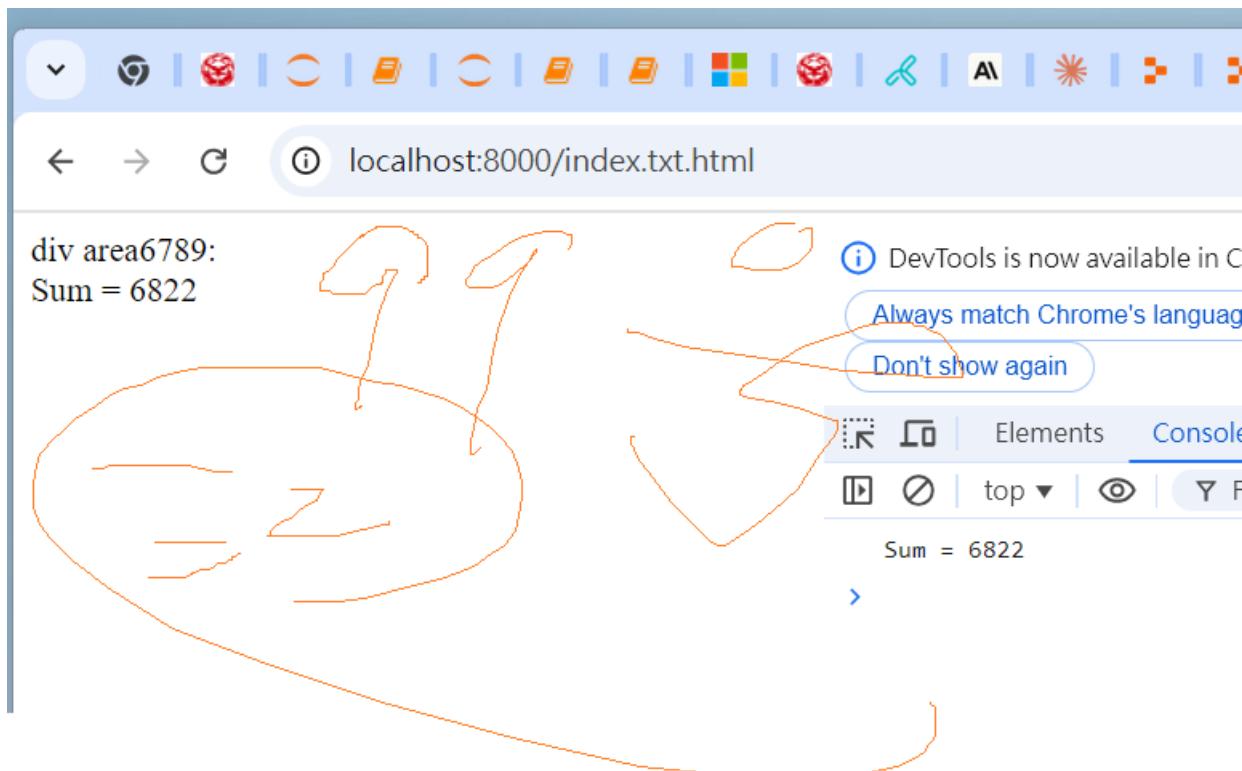
The screenshot shows a Windows desktop environment with several open windows:

- File Explorer:** Shows a folder structure under "lib" named "ch02_js入門". Inside are "index.txt.html" and "script.js". The "index.txt.html" file is selected.
- Terminal:** A Anaconda Prompt window titled "系統管理員: Anaconda Prompt - python -m http.server 8000". It displays log output from a local server at port 8000, showing requests for index.txt.html and script.js.
- Browser Developer Tools:** An "index.txt.html - 記事本" window containing the HTML code for "First JavaScript". It includes a "div area01" section with a red box around it.
- Code Editor:** An "index.txt.html - 記事本" window showing the same HTML code as the browser.
- Code Editor:** A "script.js - 記事本" window containing the following JavaScript code:

```
// script.js
var data = [3, 7, 2, 9, 1, 11, ,0];
var sum = 0;
data.forEach(function(d){
    sum += d;
});

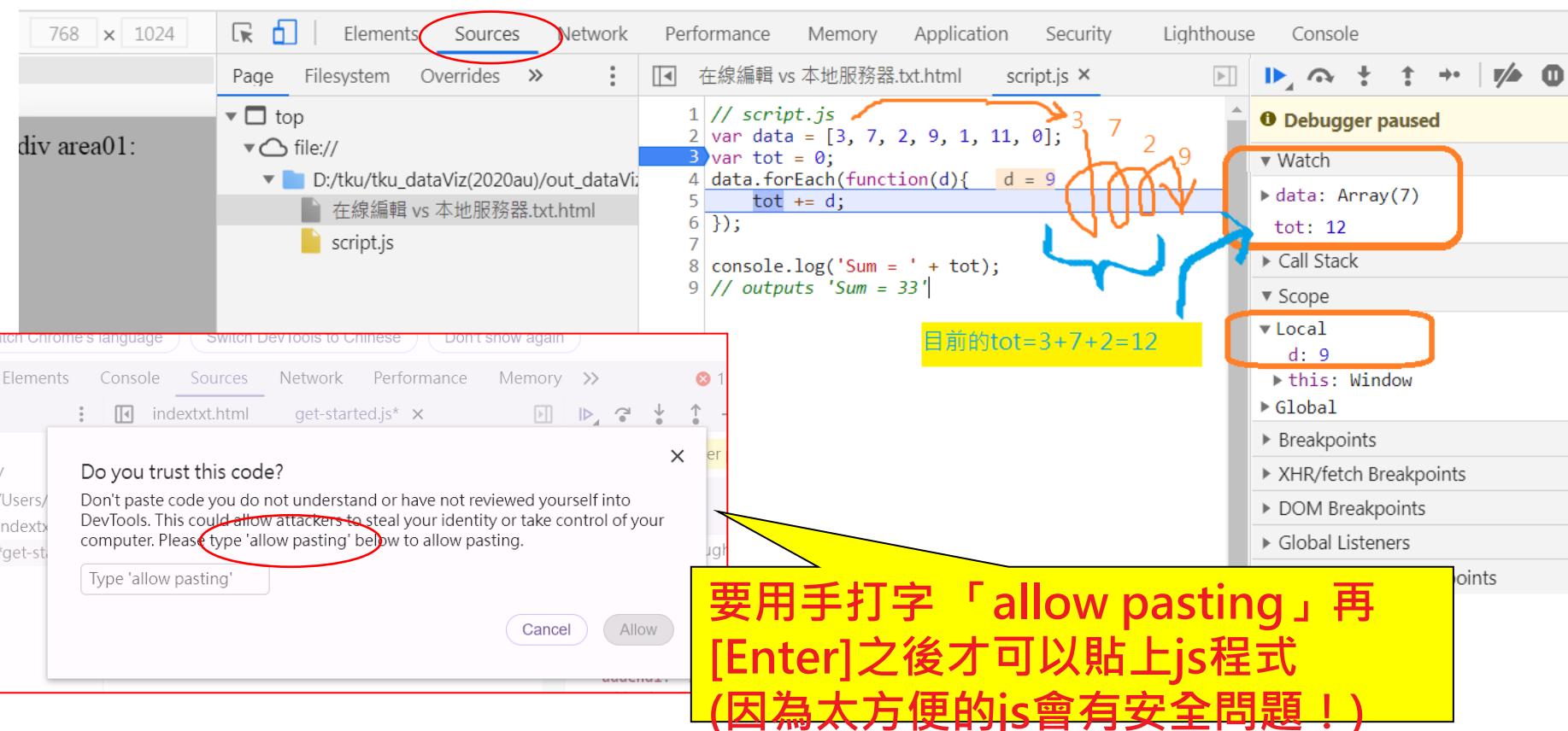
console.log('Sum = ' + sum);
// outputs 'Sum = 33'
```
- Browser Developer Tools:** A "First JavaScript" tab in the browser showing the URL "localhost:8000/index.txt.html". The "Console" tab displays "Sum = 33". The "Sources" tab shows the original HTML and JS code.

※註：● ch02實作練習1的參考答案：若老師的學號末四碼為”6789”應該是以下的結果：



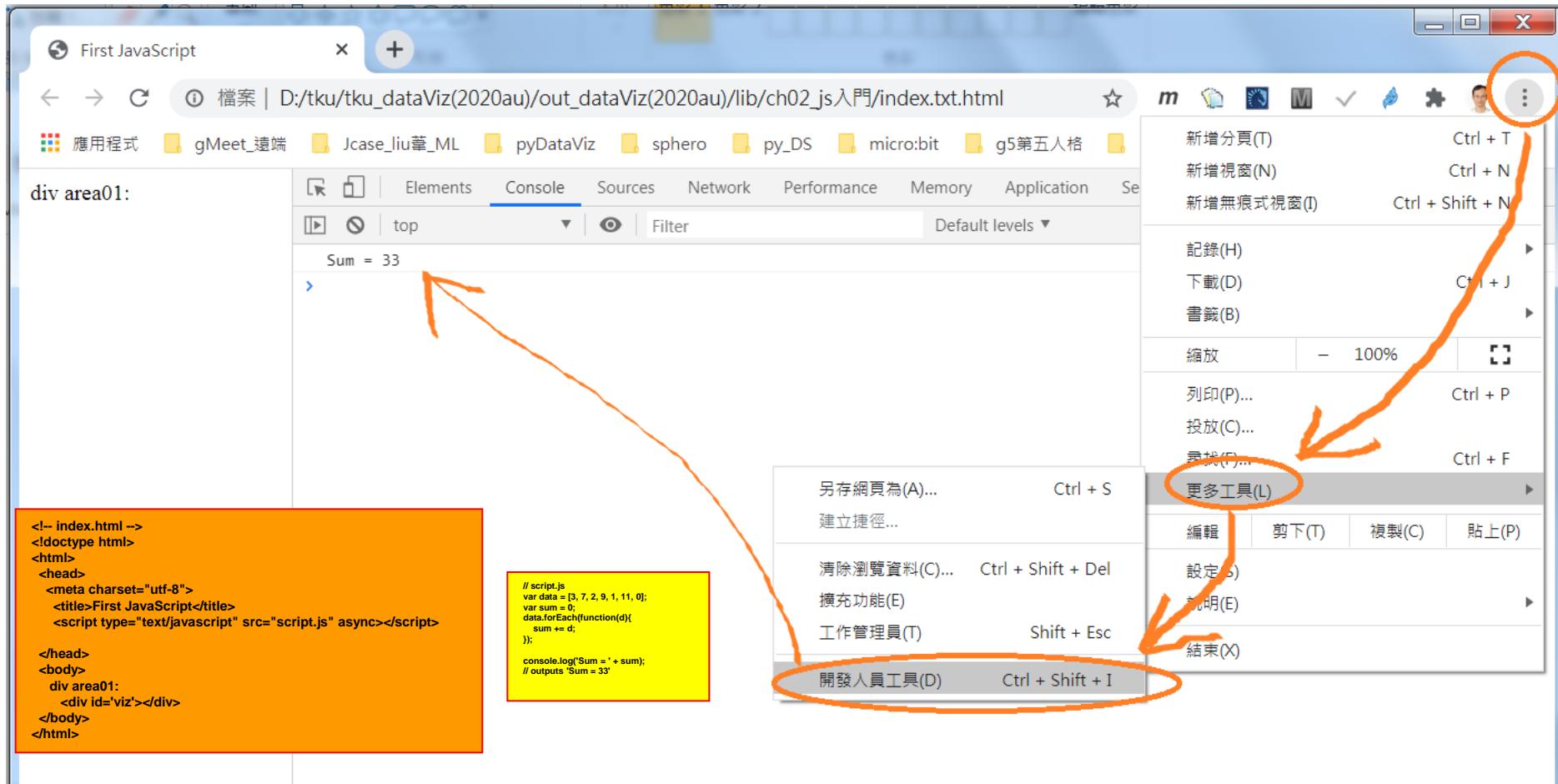
Lab#ch02-1a2_js入門localhost改用chrome的開發者工具 如此就可以進行Debug與Watch。方便程式的開發！

- ※注意：前面實驗完成，則可在發現有bug時，進行debug。要切換到Source就可見行號，並且可逐行執行。



Lab#ch02-1b_js入門_簡易版_(請在div area01後面的01, 與var data裡 ,0] 的0, 改為自己的學號末四碼)

■ 方法2:簡易版試驗(直接打開chrome也可試驗)



Lab#ch02-1c_js入門_簡易版_(請在div area01後面的01,與var data裡 ,0] 的0,改為自己的學號末四碼) , 例如9001

- 方法3:在<https://jsfiddle.net> 的簡單實驗。

The screenshot shows the jsFiddle interface with the following sections:

- HTML**:

```
1 <!-- index.html -->
2 <!doctype html>
3 <html>
4 <head>
5   <meta charset="utf-8">
6   <title>First JavaScript</title>
7   <script type="text/javascript" src="script.js" async></script>
8 </head>
9 <body>
10  div area9001:
11    <div id='viz'></div>
12 </body>
13 </html>
```
- JavaScript + D3 3.x**:

```
1 // script.js
2 var data = [3, 7, 2, 9, 1, 11, 9001];
3 var sum = 0;
4 data.forEach(function(d){
5   sum += d;
6 });
7
8 console.log('Sum = ' + sum);
9 // outputs 'Sum = 934'
```
- Output Area**:

div area9001:
- Console (beta)**:

"Running fiddle"
"Sum = 9034"

※注意：jsfiddle.net第一次實驗時...

- 在<https://jsfiddle.net> 實驗下面程式，會看到輸出是空白，因為要到console才會見到 console.log()
- 綠色是output, 而橘子色則是console區。

The screenshot shows the jsfiddle.net interface. In the top navigation bar, there is a search bar with 'jsfiddle.net' and a user icon. Below the bar, the title 'Untitled fiddle' is displayed. On the left, there's a sidebar with a 'JavaScript' tab selected. The main area contains the following code:

```
1 var t = 0
2 for (var i = 1; i <= 10; i++) {
3     t += 1
4 }
5 console.log(t)
6
```

To the right of the code editor is a large white space, which is highlighted with a green rectangle. At the bottom right, there is a 'Console' panel. This panel has an orange border and contains the output of the code: '10'. Above the output, the 'Console' tab is selected, and there are some status icons. The entire 'Console' panel is also highlighted with an orange rectangle.

Lab#ch02-1d_js入門_簡易版_(請在div area01後面的01,與var data裡 ,0] 的0,改為自己的學號末四碼) , 例如9001

■ 方法4:在https://repl.it 的簡單實驗。

(1) replit有調整免費制度，只可3個repl,
(2) 「replit Agent」是很不錯AI工具！(要付費)

The screenshot shows the repl.it web interface. At the top, there's a navigation bar with a back arrow, forward arrow, and a lock icon. The URL is repl.it/repls/GeneralComfortableScale#script.js. Below the URL is a file browser with several projects listed: 應用程式, gMeet_遠端, Jcase_liu董_ML, pyDataViz, sphero, py_DS, micro:bit, g5第五人格. On the left, there's a sidebar with icons for Files, index.html, script.js (which is selected), and style.css. The main workspace shows the code for 'script.js':

```
1 // script.js
2 var data = [3, 7, 2, 9, 1, 11, 9001];
3 var sum = 0;
4 data.forEach(function(d){
5   sum += d;
6 });
7
8
9 console.log('Sum = ' + sum);
10 // outputs 'Sum = 33'
11
```

To the right of the code, there's a 'Run' button and a preview window. The preview window shows the output of the script: 'div area9001:' followed by two lines of text: 'Sum = 9034'. The entire screenshot is framed by a yellow border.

Lab#ch02-1e_js入門_簡易版_(請在div area01後面的01, 與var data裡 ,0] 的0, 改為自己的學號末四碼) , 例如9001

■ 方法5:在JSBin (<https://jsbin.com/>)的簡單實驗。

The screenshot shows the JSBin interface with the following components:

- Address Bar:** jsbin.com/gifuliriga/edit?html,js,console,output
- File Menu:** File, Add library, Share
- Toolbars:** HTML, CSS, JavaScript, Console, Output, Clear, Run, Loc
- HTML pane:** Contains the HTML code for index.html, which includes a div element with id="viz" and class="area6789".
- JavaScript pane:** Contains the script.js code:

```
// script.js
var data = [3, 7, 2, 9, 1, 11, 6789];
var sum = 0;
data.forEach(function(d){
    sum += d;
});
console.log('Sum = ' + sum);
// outputs 'Sum = 33'

document.getElementById('viz').innerHTML =
```
- Console pane:** Displays the output of the console.log statement: "Sum = 6822"
- Output pane:** Displays the final rendered HTML output: "--> div area6789 Sum = 6822"

Exercise 2-2

- **Ex2-2Add1:** 網路上有很多在線JavaScript編輯器，可以在不啟動服務器測試JavaScript代碼，這種撰寫程式的方式是否有何缺點？
- **Ex2-2Add2:** 如果下載python的官方版，安裝完之後，是否有可以與程式碼互動的功能？簡介之。
- **Ex2-2Add3:** 網路上有很多在線程式編輯器，以能夠編輯JavaScript者，至少舉出三個。

2-3:在語言基本工作上的差異(Basic Bridge Work)

■ Python專門提供Python增強建議(例PEP8風格指南)

- 建議要熟悉PEP-8，但不必完全屈服於其建議。因為多數是對的，但是少數還有一些個人選擇的餘地。
- 有方便的檢查器，可檢查任何違反PEP-8的情況。(官方有推出檢查程式碼風格是否符合PEP8的工具)名字pycodestyle，裝完anaconda也會裝這個檢查程式。
- 用法：`$pycodestyle xxx.py`
- 在python, 應該使用四個空格來縮排程式區塊。

■ JavaScript樣式指南是免費的

- 通常會默認使用jQuery大型庫所用的指南
- 在JavaScript雖然沒有嚴格要求縮幾格，常見縮格是兩個空格。
- JavaScript (ECMAScript 5) 有最新添加是“use strict”指令，該指令施加了嚴格檢查模式。

- 此模式強制執行一些良好的JavaScript寫法，其中包括捕獲意外的全局聲明，強烈建議可以使用它！
- 要用它只需將字符串放在函數或模塊的頂部，如圖所示：

```
(function(foo){  
  'use strict';  
  // ...  
  }(window.foo = window.foo || {}));
```

Lab#2-3 查看: 打開google頁面的js,

“use strict” 就是嚴格檢查模式

The screenshot shows a browser window with the Google homepage. A green arrow points from the 'C' icon in the address bar to the search bar. Another green arrow points from the search bar down to the Google logo. A red box highlights a portion of the JavaScript code in the Network tab's response body, which includes the 'use strict' directive.

```
(function(foo){  
  'use strict';  
  // ...  
} (window.foo = window.foo || {}));
```

The Network tab in DevTools is selected. A green arrow points from the 'Network' tab to the Headers section of the Network tab. A red box highlights the 'm=_b_tp' entry in the Headers list. A red arrow points from this entry to the detailed view of the response body, which contains the same 'use strict' code as the highlighted box.

```
"use strict";  
this.default_OneGoogleWidgetUi = this.default_OneGoogleWidgetUi || {};  
(function(_) {  
  var window = this;  
  try {  
    _._F_toggles_initialize = function(a) {  
      (typeof globalThis !== "undefined" ? globalThis : typeof self === "undefined"  
    }  
  };  
  (0,  
  _._F_toggles_initialize)([0x20304020, 0x1f1, ]);
```

由語言的基本用法上，區別Python與JS兩個語言。

- 先簡單區別下列各方向的異同。
 - 變數名稱的習慣、載入模組(或載入腳本程式)、命名空間、如何輸出“Hello World！”，
- 再用兩個**Example**程式，區分下列常使用到的語法：
 - 資料處理、字串建立的方法(**String Construction**)、
python有重大意義的空白與JavaScript的大括號、註釋
 - 其他各方面的比較：
 - Declaring Variables, var
 - Strings and Numbers
 - Booleans
 - Data Containers: Dicts, Objects, Lists, Arrays
 - Functions
 - Iterating: for Loops and Functional Alternatives
 - Conditionals: if, else, elif, switch
 - File Input and Output
 - Classes and Prototypes

變數名稱

駝峰式大小寫(CamelCase)與下劃線分隔單字

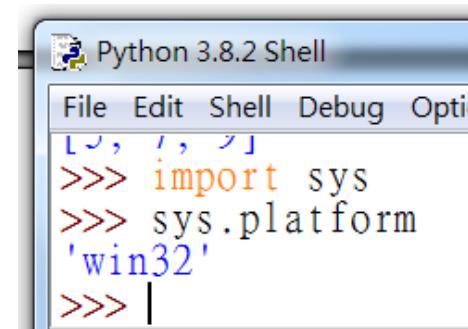
- JS通常採用**CamelCase**的方法來訂變量名稱。
 - 例如processStudentData
- Python則根據**PEP-8**，在其變量名稱中使用下劃線
 - 例如process_student_data
 - Python的約定（約定在Python生態系統中比JS更重要）：
 - 大寫的**CamelCase**進行類聲明（請參見以下示例）
 - 全部大寫的形式用於常量。
 - 所有其他內容中使用下劃線。
 - FOO_CONST = 10
 - class FooBar(object): # ...
 - def foo_bar():
 - baz_bar = 'some string'

載入模組或載入腳本程式

(Importing Modules, Including Scripts)

- Python有一個簡單的導入系統，總體上可以很好地運行。可說帶了“電池”！因為已有全面的庫，涵蓋從擴展數據容器（集合）到使用CSV文件系列（csv）的所有內容。如果要使用其中之一，只需使用import關鍵字將其導入：

- In [1]: import sys
 - In [2]: sys.platform
 - Out[2]: 'win32'



A screenshot of the Python 3.8.2 Shell interface. The menu bar includes File, Edit, Shell, Debug, and Options. The code input area shows three lines of code: 'In [1]: import sys', 'In [2]: sys.platform', and 'Out[2]: 'win32''. The 'sys' variable is highlighted in orange, and 'platform' and 'win32' are highlighted in blue.

- JavaScript尚未有完整的這點機制，建議使用公認的HTML<script>標記來包含腳本。因此，要包括D3可視化庫，您可以將此標籤添加到您的主HTML文件（通常為索引）中。

- <!DOCTYPE html>
 - <meta charset="utf-8">
 - ...
 - <script src="<https://d3js.org/d3.v3.min.js>" charset="utf-8"></script>

Python模塊的命名空間很整潔

- Python模塊中定義的變量是「封裝隔離」的，這意味著除非您明確地導入它們（例如，`from foo import baa`，就可以直接用`baa`），否則您就要使用具有「模塊名稱+點符號（例如`foo.baa`）」的方式，從導入的模塊的命名空間中訪問它們。
--->這是很好的模塊命名管理機制哦！
 - 正確地將「全域命名空間」形成模塊化的機制，是一件好事，並且符合Python的一項主要宗旨：「明確語句優於隱晦語句」。
 - 分析某人的Python代碼時，您應該能夠準確看到類，函數或變量的來源。
 - 同樣重要的是，保留名稱空間會限制衝突或掩蓋變量的機會-隨著代碼庫變大，這是一個潛在的大問題。

JavaScript的名稱空間是鬆散的.

- 對JavaScript的主要批評之一（也是很公平的批評）是它與名稱空間約定起著快速而鬆散的作用。
- 最令人震驚的例子是，在「**函數外部聲明的變量或缺少var關鍵字的變量是『全局變量』**」，而不是局限於聲明它們的腳本中。
- 所以，會用到的變數，儘量都是通過**var**聲明的變數。如此在腳本/函數中是局部的，從而防止它們污染全局名稱空間。

符合ES5以上的模塊樣版(module pattern)：

- 模塊模式示範，在專用導入系統(import system)成為標準之前的樣板寫法：

多script引入多js時的共用樣板.

Example 2-2. A module pattern for JavaScript

```
(function(nbviz) { ①
  'use strict';
  // ...
  nbviz.updateTimeChart = function(data) { ②
    // ...
  }
  (window.nbviz = window.nbviz || {})); ③
```

(3)，會在變量nbviz已存在，使用它，否則創建一個空物件。※最先載入的創建物件，後續的是擴展該物件

- 樣板的頭和尾（標記1 & 3）有效地創建了一個封裝模塊。
- 標記2是，多個js，可分別設計，例如此處：把updateTimeChart的方法附加到全域物件nbviz裡，等同於把它匯出(exporting)給外界
- 這種模式，並非模塊化(modular) JavaScript的完美解決方案，這只是一種折衷方案。
- 此寫法的缺點是模塊(module)是全局名稱空間(window)的一部分！不像Python很清楚，並不需要顯式導入(import)它。

輸出 “Hello World！” 的方法

- 到目前為止，任何編程語言最流行的初始演示是使其能夠打印或傳達“Hello World！”。以某種形式，所以我們首先從Python和JavaScript獲取輸出。

- Python的輸出非常簡單

- Print (“hello”)

```
# In Python 3
print('Hello World!')
```

- JavaScript沒有打印功能，但是您可以將輸出記錄到瀏覽器控制台

- Console.log("hello")

- 或alert()

- 或document.write()



簡單小數據處理—py與js

- 了解差異的一個好方法，就是用兩種語言來編寫相同作用的功能。
- Example2-3和2-4分別顯示了一個用Python和JavaScript處理數據的小例子。
- 我們將使用它們來比較Python和JS語法。
 - A段：準備好的小數據
 - B段：處理函數的宣告，並且參數會有預設值
 - C段：判斷「高分通過 merit」「通過」「失敗」
 - D段：輸出字串
 - E段：呼叫處理函數

Lab#2-3 Example 2-3. Simple data 小數據處理 with

File Edit Format Run Options Window Help

```
#Example 2-3. Simple data munging with Python
from __future__ import print_function

# A
student_data = [
    {'name': 'Bob', 'id':0, 'scores':[68, 75, 56, 81]},
    {'name': 'Alice', 'id':1, 'scores':[75, 90, 64, 88]},
    {'name': 'Carol', 'id':2, 'scores':[59, 74, 71, 68]},
    {'name': 'Dan', 'id':3, 'scores':[64, 58, 53, 62]},
]

# B
def process_student_data(data, pass_threshold=60,
                         merit_threshold=75):
    """ Perform some basic stats on some student data. """

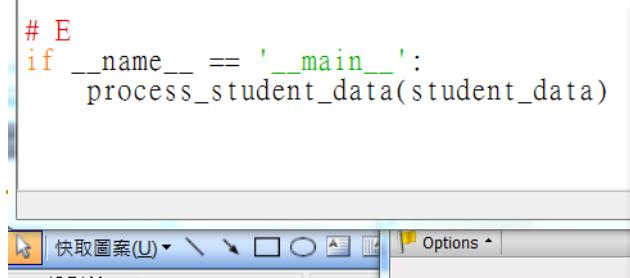
# C
for sdata in data:
    av = sum(sdata['scores'])/float(len(sdata['scores']))
    sdata['average'] = av

    if av > merit_threshold:
        sdata['assessment'] = 'passed with merit'
    elif av > pass_threshold:
        sdata['assessment'] = 'passed'
    else:
        sdata['assessment'] = 'failed'

# D
print("%s's (id: %d) final assessment is: %s"%
      (sdata['name'], sdata['id'], sdata['assessment'].upper()))

# E
if __name__ == '__main__':
    process_student_data(student_data)
```

通過的分數**60**與優秀的
分數**75**的預設值設定



```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
=====
RESTART: C:/Users/rr/AppData/Local/Programs/Python
Bob's (id: 0) final assessment is: PASSED
Alice's (id: 1) final assessment is: PASSED WITH MERIT
Carol's (id: 2) final assessment is: PASSED
Dan's (id: 3) final assessment is: FAILED
>>>
```

Lab#2-3 Example 2-4. Simple data 小數據處理 with JS

File Add library Share HTML CSS JavaScript Console Output

JavaScript

```
//Example 2-4. Simple data munging with JavaScript
//※注意：在key中的name有一些有加引號，有些沒有，都是JS合法寫法！
var studentData = [
    {name: 'Bob', id:0, 'scores':[68, 75, 76, 81]}, // name has quotes
    {name: 'Alice', id:1, 'scores':[75, 90, 64, 88]}, // name has quotes
    {'name': 'Carol', id:2, 'scores':[59, 74, 71, 68]}, // name has no quotes
    {'name': 'Dan', id:3, 'scores':[64, 58, 53, 62]}, // name has no quotes
];
// B
function processStudentData(data, passThreshold, meritThreshold){
    passThreshold = typeof passThreshold !== 'undefined'?
        passThreshold: 60;
    meritThreshold = typeof meritThreshold !== 'undefined'?
        meritThreshold: 75;

    // C
    data.forEach(function(sdata){
        var av = sdata.scores.reduce(function(prev, current){
            return prev+current;
        },0) / sdata.scores.length;
        sdata.average = av;

        if(av > meritThreshold){
            sdata.assessment = 'passed with merit';
        } else if(av > passThreshold){
            sdata.assessment = 'passed';
        } else{
            sdata.assessment = 'failed';
        }
        // D
        console.log(sdata.name + "'s (id: " + sdata.id +
            ") final assessment is: " +
            sdata.assessment.toUpperCase());
    });
}
// E
processStudentData(studentData);
```

Console

```
"Bob's (id: 0) final assessment is: PASSED"
"Alice's (id: 1) final assessment is: PASSED WITH MERIT"
"Carol's (id: 2) final assessment is: PASSED"
"Dan's (id: 3) final assessment is: FAILED"
```

通過的分數60與優秀的
分數75的預設值設定

File Edit Format Run Options Window Help

```
#Example 2-3. Simple data munging
from __future__ import print_function

# A
student_data = [
    {'name': 'Bob', 'id':0, 'scores':[68, 75, 76, 81]}, // name has quotes
    {'name': 'Alice', 'id':1, 'scores':[75, 90, 64, 88]}, // name has quotes
    {'name': 'Carol', 'id':2, 'scores':[59, 74, 71, 68]}, // name has no quotes
    {'name': 'Dan', 'id':3, 'scores':[64, 58, 53, 62]}, // name has no quotes
]

# B
def process_student_data(data, pass_threshold=60, merit_threshold=75):
    """ Perform some basic stats """

    # C
    for sdata in data:
        av = sum(sdata['scores']) / len(sdata['scores'])
        sdata['average'] = av

        if av > merit_threshold:
            sdata['assessment'] = 'passed with merit'
        elif av > pass_threshold:
            sdata['assessment'] = 'passed'
        else:
            sdata['assessment'] = 'failed'

    # D
    print("%s's (id: %d) final assessment is: %s" %
          (sdata['name'], sdata['id'], sdata['assessment']))

# E
if __name__ == '__main__':
    process_student_data(student_data)
```

Lab#2-3 Example 2-4. ※必須本地檔案，再(按右鍵)選chrome開啟本地檔案，再打開開發人員工具，即可設中斷點偵測！方便除

設中斷點！

```
//Lab#Example 2-4. Simple data munging with JavaScript
//Bob Alice Carol Dan
var studentData = [
  {name: 'Bob', id:0, 'scores':[68, 75, 76, 81]},
  {name: 'Alice', id:1, 'scores':[75, 90, 64, 88]},
  {name: 'Carol', id:2, 'scores':[59, 74, 71, 68]},
  {name: 'Dan', id:3, 'scores':[64, 58, 53, 62]},
];
// B
function processStudentData(data, passThreshold, meritThreshold){  data = (4) [...]
passThreshold = typeof passThreshold !== 'undefined'?  
  passThreshold: 60;
meritThreshold = typeof meritThreshold !== 'undefined'?  
  meritThreshold: 75;
// C
data.forEach(function(sdata){
  var av = sdata.scores.reduce(function(prev, current){
    return prev+current;
  },0) / sdata.scores.length,
  sdata.average = av;
  if(av > meritThreshold){
    sdata.length: 4
    sdata.  
  } else if(av < passThreshold){
    sdata.length: 0
  }
// D
  console.log("final", sdata);
});
// E
processStudentData(studentData);
}
```

F9單步Step執行

Paused on breakpoint

Call Stack

Scope

Local

Breakpoints

Global

XHR/fetch Breakpoints

DOM Breakpoints

Global Listeners

Event Listener Breakpoints

Pause on caught exceptions

Watch

Call Stack

processStudentData

(anonymous)

Scope

Local

sdata: Array(4)

0: {name: "Bob", id: 0, scores: Array(4)}

1: {name: "Alice", id: 1, scores: Array(4)}

2: {name: "Carol", id: 2, scores: Array(4)}

3: {name: "Dan", id: 3, scores: Array(4)}

length: 4

__proto__: Array(0)

meritThreshold: undefined

passThreshold: 60

this: Window

Global

Breakpoints

example2-4.js:14
meritThreshold = typeof meritThreshold !== 'und...

example2-4.js:18
data.forEach(function(sdata){

example2-4.js:24
if(av > meritThreshold){

example2-4.js:34
console.log(sdata.name + "'s (id: " + sdata.id +

Coverage: n/a

Line 39, Column 2

Console What's New X

Debug: (直接chrome寫片段程式除錯法)

- ※ 若要直接用**chromeDevTools**「片段程式」，可參考後面的Lab#2-3_ChromeDevTools的第一個小程式

```
8  ];
9  // B
10 function processStudentData(data, passThreshold, meritThreshold){    data =
11   passThreshold = typeof passThreshold !== 'undefined'?
12     passThreshold: 60;
13   meritThreshold = typeof meritThreshold !== 'undefined'?
14     meritThreshold: 75;
15 } // C
```

```
//Lab#Example 2-4. Simple data munging with JavaScript
//※注意：在key中的name有一些有加引號，有些沒有，都是JS合法寫法！
var studentData = [
  {name: 'Bob', id:0, 'scores':[68, 75, 76, 81]},
  {name: 'Alice', id:1, 'scores':[75, 90, 64, 88]},
  {'name': 'Carol', id:2, 'scores':[59, 74, 71, 68]},
  {'name': 'Dan', id:3, 'scores':[64, 58, 53, 62]},
];
// B
function processStudentData(data, passThreshold, meritThreshold){
  passThreshold = typeof passThreshold !== 'undefined'?
    passThreshold: 60;
  meritThreshold = typeof meritThreshold !== 'undefined'?
    meritThreshold: 75;
} // C
data.forEach(function(sdata){
  var av = sdata.scores.reduce(function(prev, current){
    return prev+current;
},0) / sdata.scores.length;
  sdata.average = av;
  if(av > meritThreshold){
```

作用域
本機
this: Window
data: (4) [...], {...}, {...}, {...}
meritThreshold: undefined
passThreshold: undefined
全域
Window

第 1 個指令碼片段 x

新的程式碼片段

第 1 個指令

var av = sdata.scor... 17

作用域
本機
this: Window
av: undefined
sdata:
id: 0
name: "Bob"
scores: Array(4)
0: 68
1: 75
2: 76
3: 81
length: 4
[[Prototype]]: Array(0)
[[Prototype]]: Object
閉包 (processStudentData)
全域
Window

呼叫堆疊
(匿名) 第 1 個指令碼片段:17

【check2】 ch02 實作練習2：簡單小數據

要求：py 與 js 都加入3筆與自己相關的新資料，例如：

- {'name': '江俊廷', 'id':99, 'scores':[80, 81, 82, 83]}, (平均超過75才是完美通過！)
- {'name': '江俊廷2', 'id':98, 'scores':[75, 75, 75, 75]},
- {'name': '江俊廷3', 'id':97, 'scores':[59, 59, 59, 59]},

要學會jupyter notebook複製多
格到新檔案的方法！

jupyter 簡單小數據處理_py與js##### end ##### 製作者：江俊廷99號 Last Change: 2023-09-18 (autosaved) Logout Trusted Python 3 (ipykernel) 0

The screenshot shows a Jupyter Notebook interface with two code cells. The left cell contains Python code for processing student data, and the right cell contains equivalent JavaScript code. Both cells output the same results, demonstrating the equivalence of the two languages for this task.

Python Code (Cell 1):

```
File Edit View Insert Cell Kernel Widgets Help  
+ < > Run C Markdown  
29     else:  
30         sdata['assessment'] = 'failed'  
31     # D  
32     print("%s's (id: %d) final assessment is: %s"%(sdata['name'], sdata['id'], sdata['assessment'].upper()))  
33 #E  
34 if __name__ == '__main__':  
35     process_student_data(student_data)  
  
Bob's (id: 0) final assessment is: PASSED  
Alice's (id: 1) final assessment is: PASSED WITH MERIT  
Carol's (id: 2) final assessment is: PASSED  
Dan's (id: 3) final assessment is: FAILED  
江俊廷's (id: 99) final assessment is: PASSED WITH MERIT  
江俊廷2's (id: 98) final assessment is: PASSED  
江俊廷3's (id: 97) final assessment is: FAILED
```

JavaScript Code (Cell 2):

```
56 // E  
57 // E  
58 processStudentData(studentData);  
59  
Bob's (id: 0) final assessment is: PASSED  
Alice's (id: 1) final assessment is: PASSED WITH MERIT  
Carol's (id: 2) final assessment is: PASSED  
Dan's (id: 3) final assessment is: FAILED  
江俊廷's (id: 99) final assessment is: PASSED WITH MERIT  
江俊廷2's (id: 98) final assessment is: PASSED  
江俊廷3's (id: 97) final assessment is: FAILED
```

In [2]:

```
1 # 顯示小數據的內容：  
2 print(student_data)
```

[{"name": "Bob", "id": 0, "scores": [68, 75, 56, 81], "average": 70.0, "assessment": "passed"}, {"name": "Alice", "id": 1, "scores": [75, 90, 64, 88], "average": 79.25, "assessment": "passed with merit"}, {"name": "Carol", "id": 2, "scores": [59, 74, 71, 68], "average": 68.0, "assessment": "passed"}, {"name": "Dan", "id": 3, "scores": [64, 58, 53, 62], "average": 59.25, "assessment": "failed"}, {"name": "江俊廷", "id": 99, "scores": [80, 81, 82, 83], "average": 81.5, "assessment": "passed with merit"}, {"name": "江俊廷2", "id": 98, "scores": [75, 75, 75, 75], "average": 75.0, "assessment": "passed"}, {"name": "江俊廷3", "id": 97, "scores": [59, 59, 59, 59], "average": 59.0, "assessment": "failed"}]

Lab#2-3_Example 2-4 Simple data小數據處理with 【javascript】

2-3之2:字符串建立的方法(String Construction)

- Example2-3 和 Example 2-4 中的 D 部分
 - 就有應用字符串組合的實例。
 - ### 在 D 的那行程式的寫法就是字符串組合。
- JavaScript 沒有打印語句，但會通過控制台對象登錄到瀏覽器的控制台。

```
console.log(sdata.name + "'s (id: " + sdata.id +
") final assessment is: " + sdata.assessment.toUpperCase());
```

```
print("%s's (id: %d) final assessment is: %s"
%(sdata['name'], sdata['id'], sdata['assessment'].upper()))
```

python有重大意義的空白與Javascript的大括號

- Python的「前導縮格空白」與C/c++/JavaScript的「花括號(大括號)」的是最相關的語法！
 - JS語言使用空格只是為了提高可讀性，並且可以壓縮為一行，
 - 但是，Python的前導空格，則用於指示代碼區塊的關係，若刪除會改變代碼的含義！
 - (※注意：python的前導空格是有意義的！不可以弄錯！)一圖下左。
- 保持正確的「代碼對齊」所需的額外努力可以通過增加可讀性來彌補，因為閱讀所花的時間遠遠超過編寫代碼所花費的時間，而輕鬆閱讀Python可能是Python生態系統如此健康的主要原因。
- 四個空格幾乎是必須遵守的（請參閱PEP 8）。
- JS則不是很在乎空幾格(一般建議是兩格)：一圖下右，兩個程式同效果。

```
def doubler(x):
    return x * 2
# |<-this spacing is important
```

```
var doubler = function(x) {
    return x * 2;
}
```

```
var doubler=function(x){return x*2;}
```

註釋和多行文字的字串(Comments and docstrings)

- 要在代碼中添加註釋，
- Python使用
 - #
 - 或 '' 或 """ 連續3個 代表多行註譯。(同時這也代表多行的字串稱為Docstrings)
- JavaScript使用與C語言相同的約定
 - 反斜線 (//)
 - 或/*...*/用於多行註釋：

```
# ex.py, a single informative comment
data = {} # Our main data-ball
```

```
// script.js, a single informative comment
/* A multiline comment block for
function descriptions, library script
headers, and the like */
var data = {}; // Our main data-ball
```

聲明變量:var強制宣告的重要性

- 在Example2-3和2-4的A節中，對學生數據的聲明需要JavaScript的**var**關鍵字。
 - 我們可以省去**var**，腳本可以很好地運行，但是我們有被JS意外干擾的危險！
 - 因為任何不帶**var**聲明的變量都將附加到全局名稱空間或**window**對象，這意味著它們可以輕鬆地屏蔽或被任何其他共享相同名稱的變量屏蔽。
 - 這種命名空間污染的可能性對於JS來說是一個大問題，也是您應該獲得良好的警惕來警告缺少**var**的原因。
 - 您還應該使用Ecmascript的**“use strict”指令來強制使用var**聲明所有變量（請參見“樣式準則，PEP 8，並使用strict”）。
- 嚴格來說，**JS語句應以分號終止，而不像Python，是用換行符號**。
- 有時會看到省去了分號，現代瀏覽器通常會在這裡做正確的事，但存在風險！（例如，它可能會使代碼壓縮器發生錯誤，連空白一齊消失）。

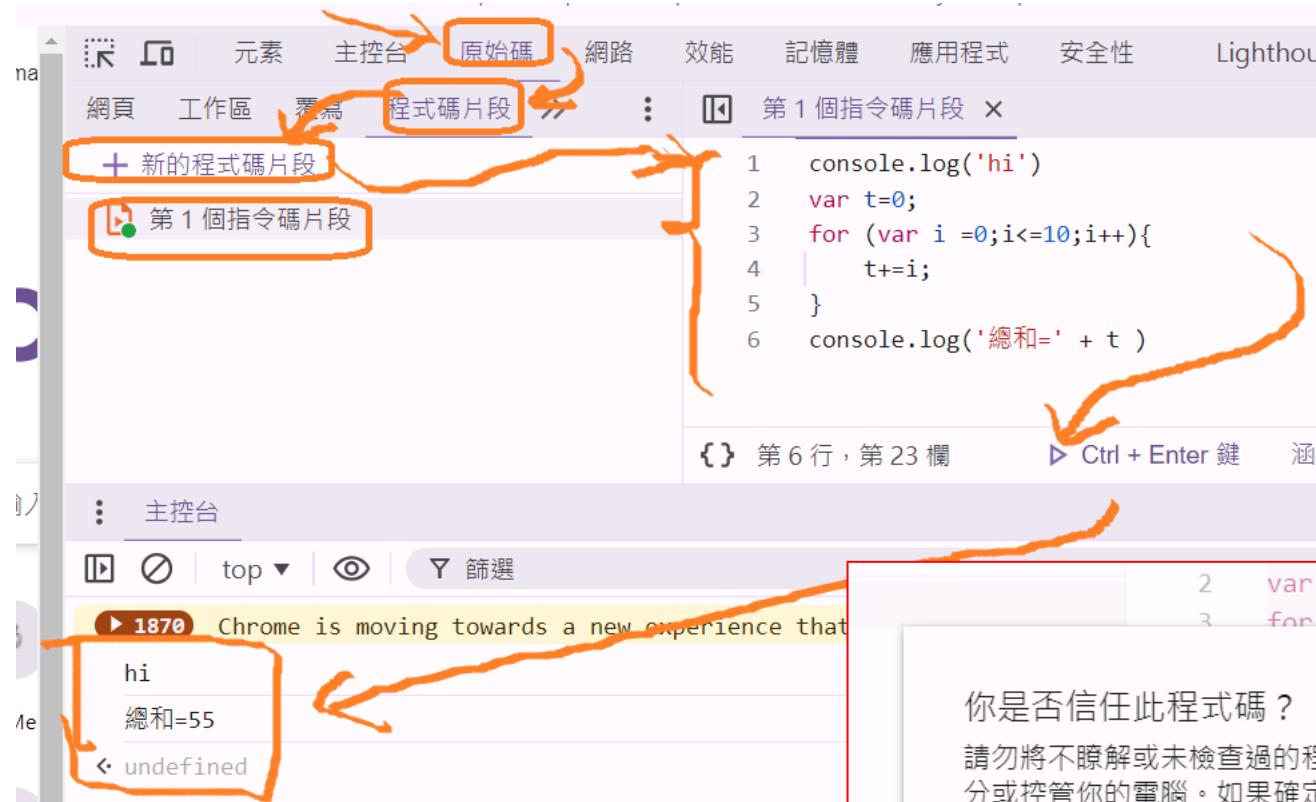
宣告變數：用var、let與未用的差別

- ●先補充ChromeDevTools的快速教學。
- ※可打開DevTools方法主要是：
 - 1. 快捷鍵：
 - 按下 Ctrl + Shift + I 或 + J 或 F12
 - 2. 網頁內，右鍵選擇「檢查」
 - 3. Chrome菜單右上角的三點選單 (:) /更多工具 (More Tools)
/開發者工具 (Developer Tools)。

用 **let** 當預設，因為它有清楚的區塊作用域，適合迴圈與一般變數。
儘量加 “**use strict**” 避免未宣告，而且也幾乎不要用 **var**，因為可能被意外覆蓋。

● ch02 實作練習3：Lab#2-3Add_ChromeDevTools的第一個 小程式

要求：其中的10 改為自己的學號末四碼。



- 程式可用手打，如果程式用貼的：
：要「允許貼上」--中文版
或「allow pasting」英文版

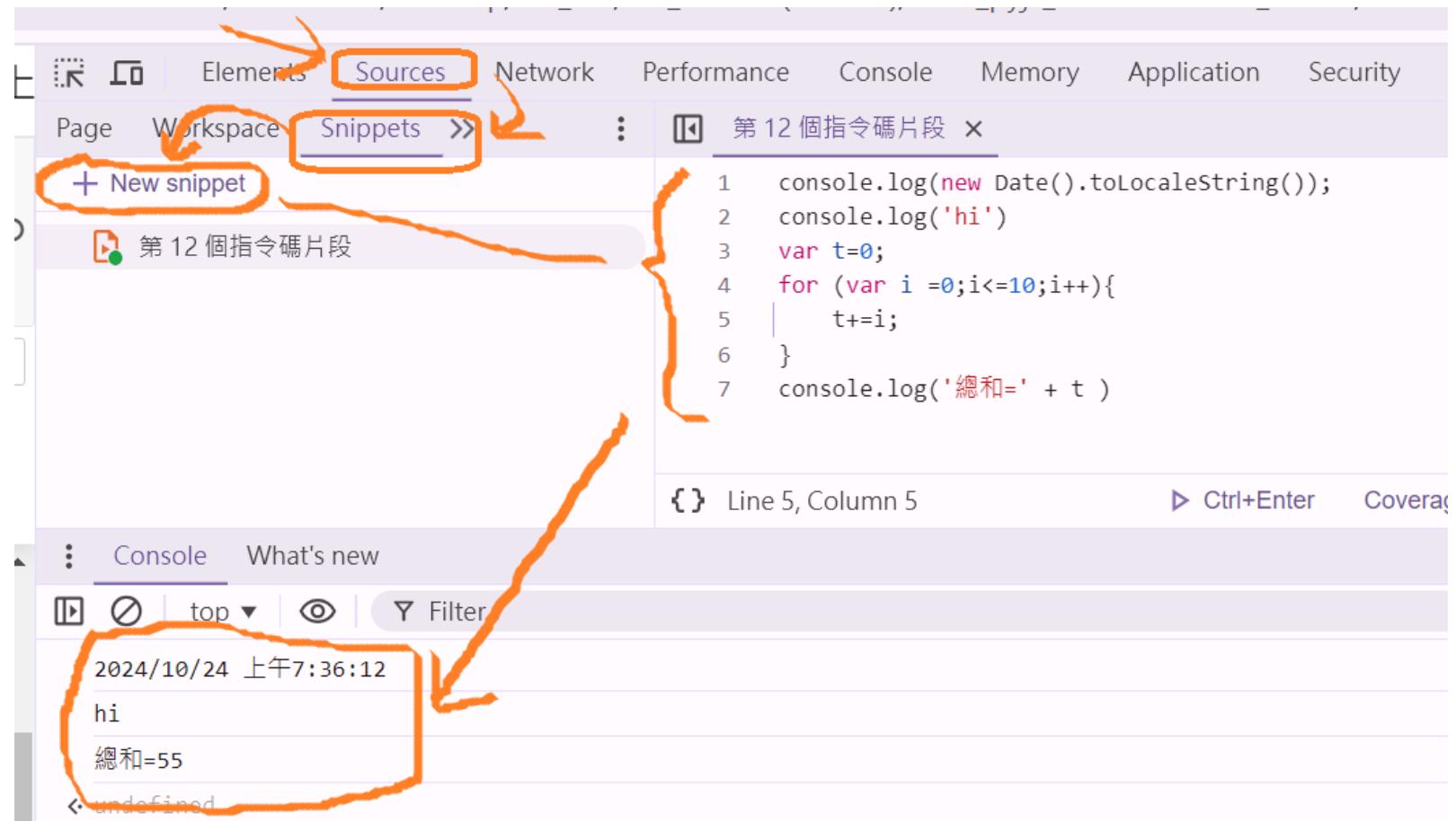
你是否信任此程式碼？

請勿將不瞭解或未檢查過的程式碼貼到開發人員工具，攻擊者可能會藉此
分或控管你的電腦。如果確定要貼上，請在下方輸入「允許貼上」。

輸入「允許貼上」

取消

英文版



Lab#2-3Add__ debug_js在Chrome DevTools的除錯實驗_step1

- 預備，兩個檔案，然後，用Chrome打開，並且`ctrl+shift+l`，並且重載入。就可執行除錯。

```
<html>
<head>
<script src="py_test_Debug.js"></script>
</head>
<body>
這是用 crhome_dev設計師，debug js的實驗。
By 江俊廷 No:99
</body>
</html>
```

//%%js

```
a01=88
a02=99
var abcd=0;
var abcd2=2222;
var abcd3=3333;
for (let i=0;i<=10;i++){
    console.log('hi');
    var abcd4=4444;
    if (true){
        let ii=100;
        console.log('i=' + i);
        console.log('ii=' + ii);
    }
    abcd+=i;
    console.log('i=' + i);
    //console.log('ii=' + ii); //ii not del
}
console.log('abcd=' + abcd);
console.log('i=' + i);
console.log('ii=' + ii);
console.log('abcd=' + abcd);
element.append('t=' + t + '<br>' + 'ok');
```



Lab#2-3 debug_js在Chrome DevTools的除錯實驗 step2_區塊(Scope)域變數let

- 中文「封鎖(Block)」應該翻譯為「區塊」才正確」
- 此時用繼續執行，監看i的變化，跑到i=5時，可發現的abcd是10

Let是區塊域變數 (Block, / Scope),
var是函數域 (function)放在 (global)

```
//%%js  
a01=88  
a02=99  
var abcd=0;  
var abcd2=2222;  
var abcd3=3333;  
for (let i=0;i<10;i++){  
    console.log('hi');  
    var abcd4=4444;  
    if (true){  
        let ii=100;  
        console.log('i=' + i);  
        console.log('ii=' + ii);  
    }  
    abcd+=i;  
    console.log('i=' + i);  
    //console.log('ii=' + ii); //ii not  
}  
console.log('abcd=' + abcd);  
console.log('i=' + i);  
//console.log('ii=' + ii); //ii not  
console.log('abcd=' + abcd);  
//element.append('t=' + t + '<br>')
```

主控台

Lab#2-3 debug_js在Chrome DevTools的除錯實驗 step3 _函數(Function)域變數var

- 注意function內：有var是函數域變數，未加var是全域變數

The screenshot shows the Chrome DevTools developer panel. On the left is the code editor with the following content:

```
//%%js
a01=88
a02=99
var abcd=0;
var abcd2=2222;
var abcd3=3333;
var ab4fun=444001;
function test(){
    var ab4fun=444999; //局部的函數變數
    a00global=99;
    console.log('ab4fun=' + ab4fun);
}
test();
console.log('ab4fun=' + ab4fun);
var x=0;
```

The variable `ab4fun` at line 7 is circled in orange. Inside the `test()` function, a local variable `ab4fun` is declared at line 9, also circled in orange. The value `444999` is highlighted in yellow. The variable `a00global` at line 10 is circled in orange. The `console.log` statement at line 11 is highlighted in yellow. The variable `x` at line 15 is circled in orange.

The variable scope tree on the right shows:

- 作用域 (Scope):
 - 本機 (This): `this: Window`, `ab4fun: 444999`
 - 全域 (Global):
 - `0: Window {window: Windo`
 - `JSCCompiler_renamePropert`
 - `a00global: 99`
 - `a01: 88`
 - `a02: 99`
 - `ab4fun: 444001`
 - `abcd: 0`
 - `abcd2: 2222`

的變數不同含義！

The screenshot shows the Chrome DevTools console tab with the following output:

```
ab4fun=444999
ab4fun=444001
↳ undefined
```

字符串和數字

- 學生數據中使用的名稱(字串)（請參見示例2-3和2-4的A節）
 - 在JavaScript中將字串解釋為UCS-2（※後來UCS-2有改良後的新版UTF-16）
 - 在Python3中則是將字串解釋為 Unicode（預設為UTF-8）
- 兩種語言都允許對字串使用單引號和雙引號。
 - 如果要在字串中，使用單引號，就可以用雙引號將其括起來，如下所示：

```
pub_name = "The Brewer's Tap"
```

2-3之3:數字的規範

- Python的數字:浮點數(float)是符合IEEE754規範 (※特別：Python3會自動調整整數(int)，可處理無窮精確度整數！)

```
foo = 3.4 # type(foo) -> float  
bar = int(3.4) # type(bar) -> int
```

The screenshot shows the Python 3.8.2 Shell window. It displays the following code and its output:

```
>>> print(int(1.1), int(1.9), int(-2.9))  
1 1 -2  
>>> print(0.1, 0.2, 0.3, 0.1+0.2)  
0.1 0.2 0.3 0.30000000000000004  
>>> 82234342434234235339 + 23338435345455243948  
105572777779689479287  
>>> |
```

The shell indicates it's at line 99, column 4.

- JS的數字: 浮點數(float)是符合IEEE754, 整數只有(52+1)位元，最大整數是($2^{53}-1$)。

```
var x = parseInt(3.45); // 'cast' x  
typeof(x); // "number"
```

JavaScript ▾

```
x=parseInt("1.2")  
y=parseInt("1.9")  
z=parseInt("2.001")  
console.log(x,y,z)
```

Console

1

1

2

Lab#2-3_測試JS的數值有效範圍

- `parseInt`是直接去尾取整數。
- 數值是採用64bit的IEEE745(就是所謂的double.)
- 最大整數是 $2^{53} - 1$
- ※注意：因為太大整數， $822..+233...$ 那個計算是錯的！

```
JavaScript ▾
x=parseInt("1.1")
y=parseInt("1.999")
z=parseInt("2.001")
console.log(x +"+" y +"+" z)
uu=2 ** 53 //uu=Math.pow(2, 53)
console.log("uu=" + uu)
console.log( 82234342434234235339 + 23338435345455243948)
console.log(Number.MAX_VALUE )
console.log(Number.MAX_SAFE_INTEGER)
if (0.1+0.2 == 0.3){
  console.log("eq")
} else {
  console.log("not eq")_
}
```

Console

"1,1,2"

"uu=9007199254740992"

105572777779689490000

1.7976931348623157e+308

9007199254740991

"not eq"

邏輯運算的布爾值

- Python與JavaScript和C類語言的區別在於使用命名布爾運算符。除此之外，它們的工作幾乎與預期的一樣。
由表格可進行比較：

Python	bool	True False	not	and	or
JavaScript	boolean	true false	!	&&	

Data Containers: JS的Object, Array 與 py的Dict, List

- 兩種語言裡最重要的容器物件：
 - , JS 的object(物件)可以像Python的dict(字典)一樣使用
 - 都是用「大括號 {}」識別.
 - ※註JSON就是JavaScript的物件
 - , JS的array(陣列) 可以像Python的list(列表) 一樣使用。
 - 都是用「中括號[]」識別.

```
# Python
d = {'name': 'Groucho', 'occupation': 'Ruler of Freedonia'}
l = ['Harpo', 'Groucho', 99]
t = ('an', 'immutable', 'container')

// JavaScript
d = {'name': 'Groucho', 'occupation': 'Ruler of Freedonia'}
l = ['Harpo', 'Groucho', 99]
```

- ※Python還有一種容器物件，稱為元組(tuuple)，功能就是不可變的列表(list)，所以速度會更快。(※註：某些二維陣列的應用，就可用list+tuple來分辨二維的差別！)

轉換JavaScript的 array 轉為 Python 的 list

- array (a) -> list (l)

Table 2-2. Lists and arrays

JavaScript array (a)	Python list (l)
a.length	len(l)
a.push(item)	l.append(item)
a.pop()	l.pop()
a.shift()	l.pop(0)
a.unshift(item)	l.insert(0, item)
a.slice(start, end)	l[start:end]
a.splice(start, howMany, i1, ...)	l[start:end] = [i1, ...]

slice(切片)

splice(拼接，縫合)

Lab#2-3 操作對應 JavaScript 的 array 與 Python 的 list

Changes and other potential problems.

```
In [75]: 1 ##python 的 list的操作示範  
          2 l=[1,2,3,4]  
          3 item=99  
          4 l.append(item)  
          5 l
```

```
Out[75]: [1, 2, 3, 4, 99]
```

```
In [76]: 1 l.pop()  
          2 l
```

```
Out[76]: [1, 2, 3, 4]
```

```
In [77]: 1 l.pop(0)  
          2 l
```

```
Out[77]: [2, 3, 4]
```

```
In [78]: 1 l.insert(0,item)  
          2 l
```

```
Out[78]: [99, 2, 3, 4]
```

```
In [81]: 1 l[2:4] # l[start:end]  
          2 l
```

```
Out[81]: [3, 4]
```

```
In [82]: 1 #l[start:end]=[l1,...]  
          2 l[2:4]=['a','b']  
          3 l
```

```
Out[82]: [99, 2, 'a', 'b']
```

```
In [ ]: 1
```

```
> a=[1,2,3,4]  
item=99  
a.push(item)  
console.log(a)
```

```
► (5) [1, 2, 3, 4, 99]
```

```
< undefined
```

```
> a.pop(); console.log(a)
```

```
► (4) [1, 2, 3, 4]
```

```
< undefined
```

```
> a.shift() ; console.log(a)
```

```
► (3) [2, 3, 4]
```

```
< undefined
```

```
> a.unshift(item); console.log(a)
```

```
► (4) [99, 2, 3, 4]
```

```
< undefined
```

```
> console.log( a.slice(2,4) )
```

```
► (2) [3, 4]
```

```
< undefined
```

```
> a.splice(2,4,'a','b'); console.log(a)
```

```
► (4) [99, 2, "a", "b"]
```

```
< undefined
```

```
⋮ Console What's New ×
```

Functions定義方法

- Python:用「保留字 **def** 定義」，用冒號「：」與「縮格」來標示區塊。

```
def process_student_data(data, pass_threshold=60,  
                         merit_threshold=75):  
    """ Perform some basic stats on some student data. """  
    ...
```

- JS定義函數方法一、用「保留字**function**定義」的**函數宣告**（Function Declaration），用大括號{}來標示區塊。

```
function processStudentData(data, passThreshold, meritThreshold) {  
    passThreshold = typeof passThreshold !== 'undefined' ?  
        passThreshold: 60;  
    meritThreshold = typeof meritThreshold !== 'undefined' ?  
        meritThreshold: 75;  
    ...  
}
```

- ※ ●JS定義函數方法二、函式運算式(function expression):(本課程幾乎都是採用此定義法！)，主要是因為把函數賦值給變數、可方便傳遞給其他函數。(例如回呼函數)

新版JS建議
用let取代var

```
var processStudentData = function( . . . ) {
```

Lab#2-3 JS定義函數兩種方法，簡單比較

The screenshot shows a JS Bin interface with the following details:

- Header:** Exporting/importing gist, Upgrade to pro now, Help, Versions: Processors & core libraries, How to add more libraries to the library list.
- Toolbar:** File ▾, Add library, Share, HTML, CSS, JavaScript, Console, Output.
- HTML Tab Content:** HTML code including meta tags and a title "JS Bin". A note says "實驗者：江俊廷".
- JavaScript Tab Content:** JavaScript code demonstrating two function definition methods:

```
//execute(); //error!! undefined, 因為沒賦值、無法執行
// 函數表達式的形式
var execute = function() {
    console.log("方法一的函數宣告 (Declaration)");
};
execute(); //呼叫函數

xx(); // 呼叫函數_可在定義之前
function xx() {
    console.log("方法二的函式運算式(Expression)");
};
```
- Console Output:** Displays the results of the console.log statements from both snippets.
- Output Tab:** Shows the output "實驗者：江俊廷" (Experimentator: Jiang Jun-Ting).

重覆結構的問題 (for Loops and Functional Alternatives)

■ Python:

- “for” loop可以在具有正確疊代器管道的任何事物上工作，所以您可以執行很多很酷的事情，例如印出Dict，或文件檔內容。

```
foo = {'a':3, 'b':2}
for x in foo:
    print(x)
```

```
for x in foo.items():
    print(x)
# outputs key-value tuples ('a',
```

- 印出檔案內容：

The screenshot shows three separate code snippets and their execution results in a Python 3.8.2 Shell window.

- Top Snippet:** Prints the keys of a dictionary. The output shows 'a' and 'b'.
- Middle Snippet:** Prints the items of a dictionary. The output shows key-value tuples: ('a', 3) and ('b', 2).
- Bottom Snippet:** Prints the contents of a file named 'in.txt'. The output shows the lines of the file: 'hi 202002021', 'hi 202002021', 'aaa,aaa2,aaa3', 'bbb bbb2 bbb3', 'ccc ccc2 ccc3', and 'ddd ddd2,ddd3'.

■ JS:

- 如果想直接把python的for..in 的想法拿到JS是會出錯的！因為傳回的是索引(index)，而不是項目。

```
JavaScript ▾
for (var i in ['a','b','c']) {
    console.log(i);
}
```

Online:JSBin測試

Console
"0"
"1"
"2"

structor: Jiun-Ting Jiang

The screenshot shows a for loop in the Chrome Developer Tools Console tab. The loop iterates over an array of three elements: 'a', 'b', and 'c'. The variable 'i' is used as the index, resulting in the output: 0, 1, 2.

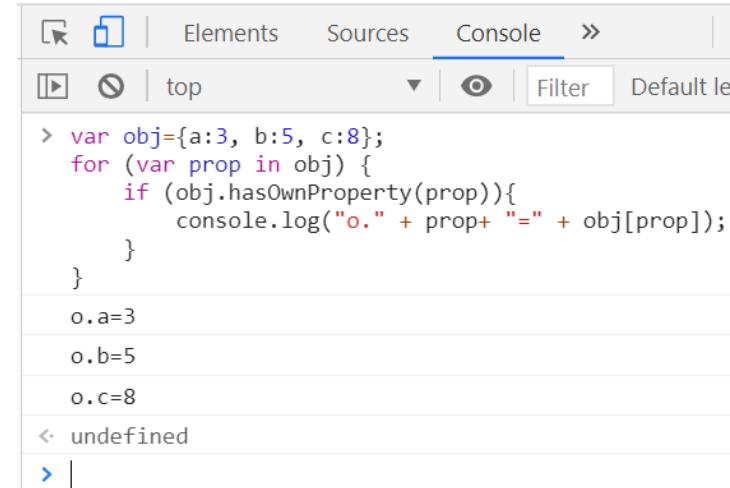
```
> for (var i in ['a','b','c']){
    console.log(i);
}
0
1
2
```

Chrome開發者--測試

JS要遍歷某容器內項目的問題

- 除了，`for .. in` 傳回的是索引(index)，而不是項目的問題，另外是不能保證疊代的順序。而且與Python的指令不同，因為對象可能具有「原型鏈的繼承屬性」，所以還要再使用`hasOwnProperty`保護器將其過濾掉，如下所示：

```
var obj = {a:3, b:2, c:4};  
for (var prop in obj) {  
    if( obj.hasOwnProperty( prop ) ) {  
        console.log("o." + prop + " = " + obj[prop]);  
    }  
}  
// out: o.a = 3, o.b = 2, o.c = 4
```



```
var obj={a:3, b:5, c:8};  
for (var prop in obj) {  
    if (obj.hasOwnProperty(prop)){  
        console.log("o." + prop+ "=" + obj[prop]);  
    }  
}  
o.a=3  
o.b=5  
o.c=8
```

- 好消息是：一般應用也很少需要將JS用於循環，而且配合高效函數庫，表達能力更強，所以熟悉後，不但沒困擾，反而會上癮！
- Example2-4中的C部分，就演示了`forEach()`，這是現代JavaScript陣列可用的功能方法之一。`forEach()`遍歷陣列的項，然後將它們依次發送到第一個參數中定義的匿名回調函數，就可以處理了。

Conditionals: if, else, elif, switch

- Python 沒有switch，但是，一定可以用多幾次的 if... 來表達。
- 而JS則可以如下寫法：(但是要注意每個case 結束最末行，要加上break.)

```
switch(expression) {  
    case value1:  
        // execute if expression === value1  
        break; // optional end expression  
    case value2:  
        //...  
    default:  
        // if other matches fail
```

Lab#2-3 JS的switch(只有JavaScript才有)

The screenshot shows a browser's developer tools with the 'Console' tab selected. The code in the console is:

```
> expression=10
value1=10
value2=200
switch(expression){
  case value1:
    // execute if expression === value1
    console.log(value1);
    break; // optional end expression
  case value2:
    //...
    console.log(value2);
  default:
    // if other matches fail
    console.log("default");
}

10
undefined
```

The output in the console is:

```
10
undefined
```

```
> expression=200
value1=10
value2=200
switch(expression){
  case value1:
    // execute if expression === value1
    console.log(value1);
    break; // optional end expression
  case value2:
    //...
    console.log(value2);
  default:
    // if other matches fail
    console.log("default");
}
```

```
200
default
```

※注意：
因為**case value2**:結束時
，漏了**break**所以會連
default一起執行！

檔案的輸入和輸出

- JavaScript在瀏覽器執行時，沒有文件輸入和輸出（I/O）的真正功能。
(因為安全問題！一般會透過AJAX的相關技術來讀寫檔案！)
- 而Python的功能非常簡單：

```
# READING A FILE
f = open("data.txt") # open file for reading

for line in f: # iterate over file lines
    print(line)

lines = f.readlines() # grab all lines in file into array
data = f.read() # read all of file as single string

# WRITING TO A FILE
f = open("data.txt", 'w')
# use 'w' to write, 'a' to append to file
f.write("this will be written as a line to the file")
f.close() # explicitly close the file
```

- 更好的建議是用with as，可以幫忙處理例外。

```
with open("data.txt") as f:
    lines = f.readlines()
    ...
```

Lab#2-3__ JavaScript 讀檔案的一種簡易方案。

- 讀取本地檔案有安全限制，但我們可以透過以下方式來實現：
 - 使用 `<input type="file">` 元素
 - 使用者點擊 input 元素，選擇本地檔案。
 - JavaScript 獲取選取的檔案，並使用 FileReader 物件將檔案內容讀取為文字。
 - 將讀取到的文字顯示在 console 或網頁上。

The screenshot shows a web-based code editor interface with the following sections:

- File** ▾ Add library Share
- HTML CSS JavaScript Console Output
- Login or Register

HTML tab (left):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="wid</head>
<body>
 實驗者：江俊廷
<input type="file" id="fileInput">
<button onclick="readFile()">
    讀取檔案
</button>
</body>
</html>
```

JavaScript tab (middle):

```
//JavaScript
function readFile() {
    const fileInput = document.getElementById('fileInput');
    const file = fileInput.files[0];

    if (file) {
        const reader = new FileReader();
        reader.onload = (event) => {
            const content = event.target.result;
            console.log(content); // 在瀏覽器 console 印出檔案內容
        };
        reader.readAsText(file);
    } else {
        alert('請選擇檔案');
    }
}
```

Console tab (right):

"實驗者：江俊廷
文字檔read成功！"

Output tab (far right):

Run with JS Auto-Run
實驗者：江俊廷
選擇檔案 未選擇任何檔案
讀取檔案

物件類別和原型(Classes and Prototype)

- 在JS有可能引起混亂的原因，是要注意JS是以原型為基礎(Prototypes-based)的OO，而不是傳統的物件類別為基礎(Classes-based)的OO。
 -
- 使用python的Class與JS的prototype來設計OO時，在想法上的差異是什麼？
 - *Raganwald Braithwaite*曾在 raganwald.com 指出這兩者的差別：「The difference between a prototype and a class is similar to the difference between a **model home** and a **blueprint** for a home.」
 - 也就是說：用Python的類別來實例化時要先有一個「藍圖(blueprint)」，創建一個對象並在繼承樹中調用其各種構造函數。
 - ※ 換句話說，要從頭開始設計，並構建原始的新類別實例。
- 反之，用JavaScript原型，您是從現有的樣品屋模型開始。
 - 一開始就有屋子的模型！如果您只想要一個新客廳，則可以用油漆換成喜歡的顏色代替舊客廳。
 - 如果您想要一個新大音樂學院，則需進行擴展。
- ※ 換句話說，一開始就有一些房間物件，已經可住人。仍可繼續修改和擴展現有物件。不需要從頭開始全新設計。

建立Citizen物件的相同OOP問題,在Python與JS簡單實例

rrr.py - C:/Users/rr/AppData/Local/Programs/Python/Python38-32/rrr.py (3.8.2)

```
File Edit Format Run Options Window Help
class Citizen(object):
    def __init__(self, name, country):
        self.name = name
        self.country = country
    def print_details(self):
        print('Citizen %s from %s' % (self.name, self.country))
c = Citizen('Groucho M.', 'Freedonia')
c.print_details()
```

Python 3.8.2 Shell

```
File Edit Shell Debug Options Window Help
>>>
===== RESTART: C:/Users/rr/AppData/Local/Programs/Python/Pyth
Citizen Groucho M. from Freedonia
>>> |
```

用jsbin線上測試//Example2-5

File Add library Share

HTML CSS JavaScript

```
JavaScript
var Citizen = function(name, country){
    this.name = name;
    this.country = country;
};

Citizen.prototype = {
    printDetails: function(){
        console.log('Citizen ' + this.name + ' from ' + this.country);
    }
};

var c = new Citizen('Groucho M.', 'Freedonia');

c.printDetails();
//Out:
```

Console

```
"Citizen Groucho M. from Freedonia"
```

用chrome的開發者工具測試//Example2-5

The screenshot shows the Google Chrome Developer Tools Console tab. The code block contains the same JavaScript as the jsbin example. The output pane shows the result of running the code: "Citizen Groucho M. from Freedonia". The status bar at the bottom right indicates the VM577:9 frame.

```
> //Example2-5
var Citizen = function(name, country){
    this.name = name;
    this.country = country;
};

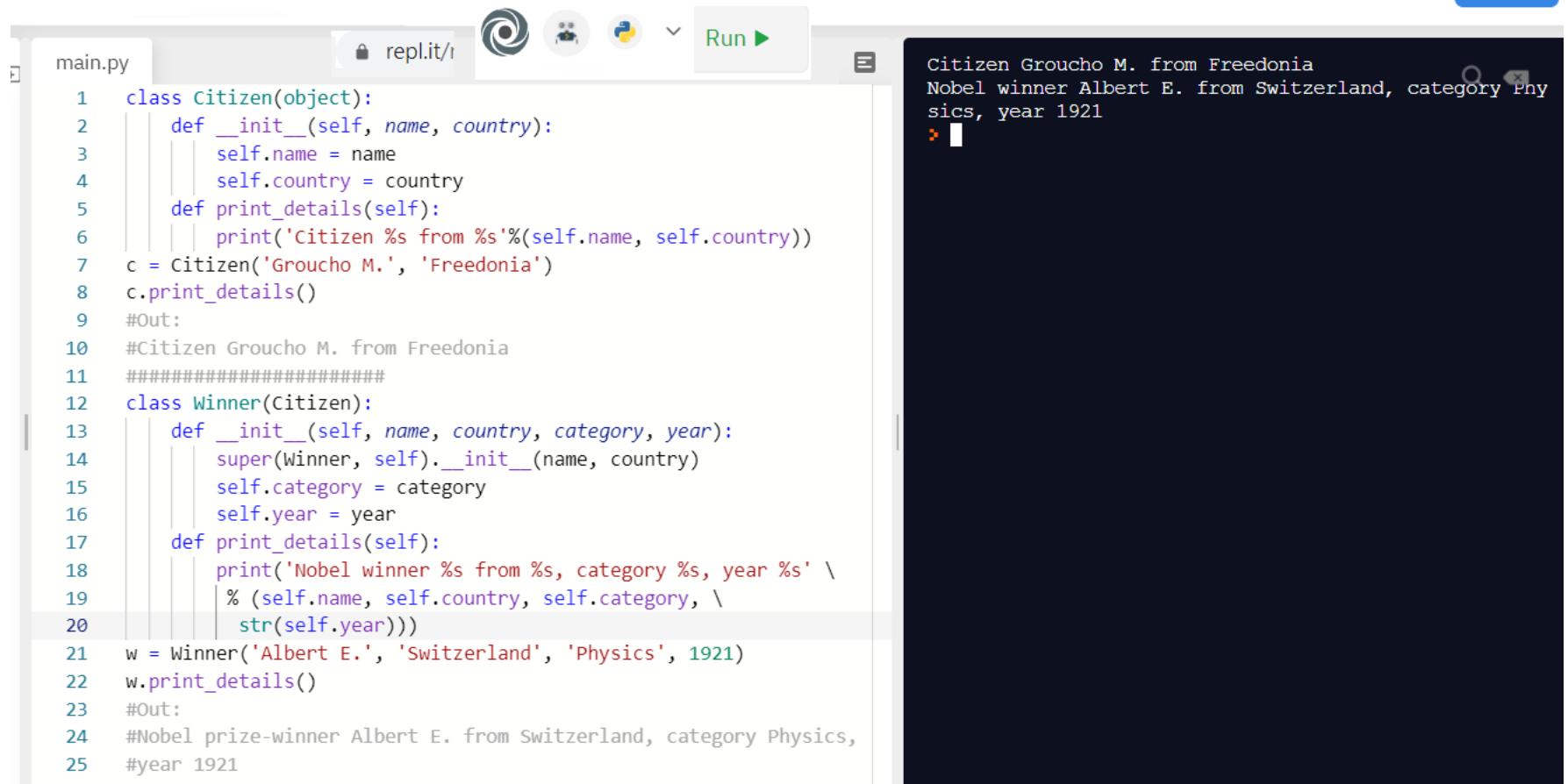
Citizen.prototype = {
    printDetails: function(){
        console.log('Citizen ' + this.name + ' from ' + this.country);
    }
};

var c = new Citizen('Groucho M.', 'Freedonia');

c.printDetails();
//Out:
//citizen Groucho M. from Freedonia
//typeof(c) # object

Citizen Groucho M. from Freedonia
VM577:9
<
>
```

Python版本: 有了Citizen之後，要讓Winner來繼承



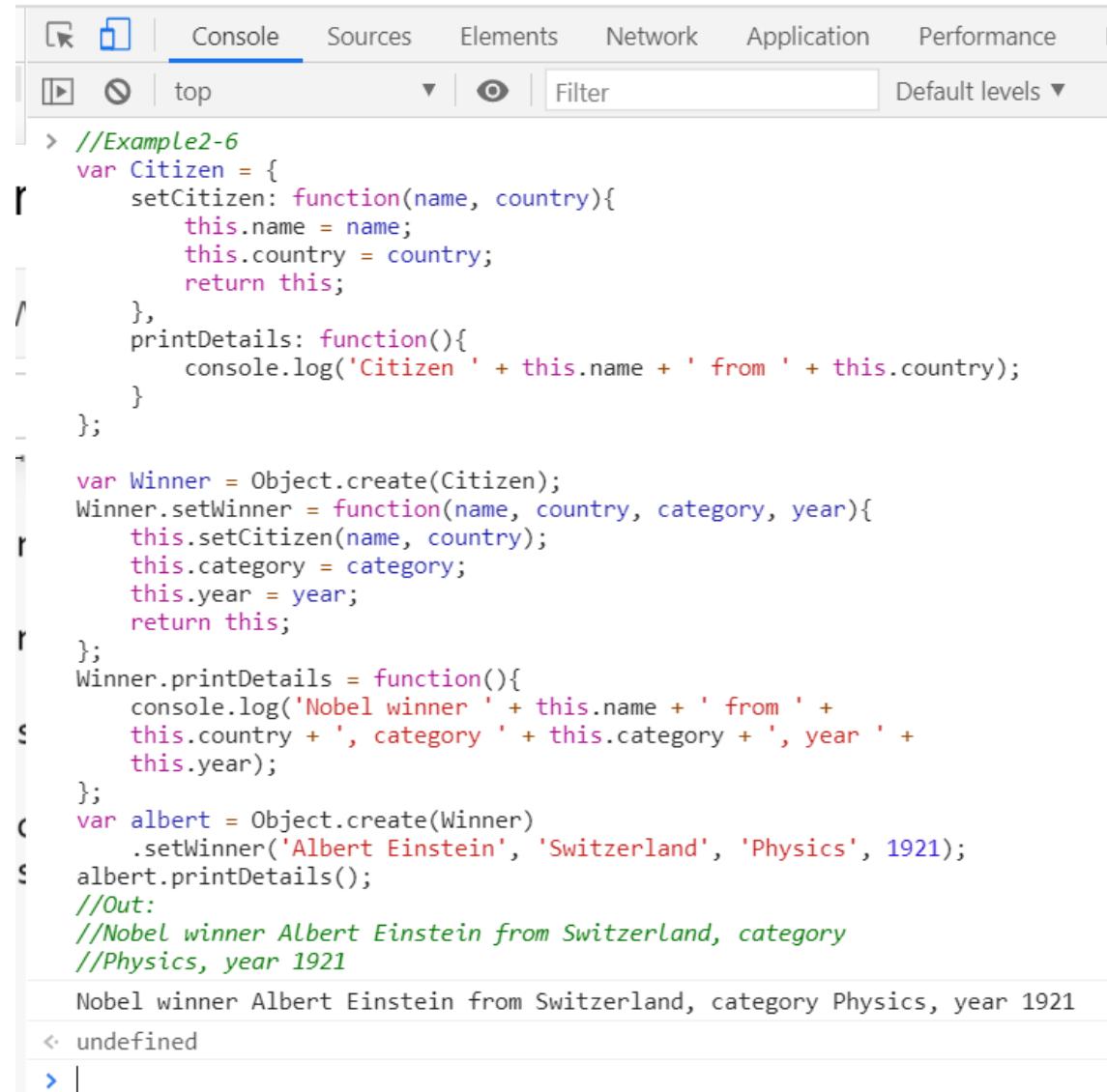
The screenshot shows a Python code editor on repl.it with the file 'main.py' open. The code defines two classes: 'Citizen' and 'Winner'. The 'Citizen' class has an __init__ method that takes name and country, and a print_details method that prints a formatted string. An instance 'c' of 'Citizen' is created for Groucho M. from Freedonia. The 'Winner' class inherits from 'Citizen' and adds a category and year parameter to its __init__ method. It also overrides the print_details method to include category and year in the output. An instance 'w' of 'Winner' is created for Albert E. from Switzerland in the Physics category in 1921. The output window shows the details for both objects.

```
main.py
 1  class Citizen(object):
 2      def __init__(self, name, country):
 3          self.name = name
 4          self.country = country
 5      def print_details(self):
 6          print('Citizen %s from %s' % (self.name, self.country))
 7  c = Citizen('Groucho M.', 'Freedonia')
 8  c.print_details()
 9
#Out:
10 #Citizen Groucho M. from Freedonia
11 #####
12 class Winner(Citizen):
13     def __init__(self, name, country, category, year):
14         super(Winner, self).__init__(name, country)
15         self.category = category
16         self.year = year
17     def print_details(self):
18         print('Nobel winner %s from %s, category %s, year %s' \
19             % (self.name, self.country, self.category, \
20                 str(self.year)))
21 w = Winner('Albert E.', 'Switzerland', 'Physics', 1921)
22 w.print_details()
23
#Out:
24 #Nobel prize-winner Albert E. from Switzerland, category Physics,
25 #year 1921
```

```
Citizen Groucho M. from Freedonia
Nobel winner Albert E. from Switzerland, category Physics, year 1921
```

JS版本1: 有Citizen後,要讓Winner用Object.create來繼承

到Chrome的開發人員工具的測試



The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The code area contains the following JavaScript:

```
//Example2-6
var Citizen = {
    setCitizen: function(name, country){
        this.name = name;
        this.country = country;
        return this;
    },
    printDetails: function(){
        console.log('Citizen ' + this.name + ' from ' + this.country);
    }
};

var Winner = Object.create(Citizen);
Winner.setWinner = function(name, country, category, year){
    this.setCitizen(name, country);
    this.category = category;
    this.year = year;
    return this;
};
Winner.printDetails = function(){
    console.log('Nobel winner ' + this.name + ' from ' +
    this.country + ', category ' + this.category + ', year ' +
    this.year);
};

var albert = Object.create(Winner)
    .setWinner('Albert Einstein', 'Switzerland', 'Physics', 1921);
albert.printDetails();
//Out:
//Nobel winner Albert Einstein from Switzerland, category
//Physics, year 1921
```

The output section at the bottom shows the result of running the last command: "Nobel winner Albert Einstein from Switzerland, category Physics, year 1921".

JS版本2版本3: 有Citizen後,要讓Winner用Object.create來繼承

JavaScript + No-Library (pure JS) ▾

```
1 var Citizen = {
2   setCitizen: function(name, country) {
3     this.name = name;
4     this.country = country;
5     return this;
6   },
7   printDetails: function() {
8     console.log('Citizen ' + this.name + ' from ' + this.country);
9   }
10 };
11 var Winner = Object.create(Citizen);
12 Winner.setWinner = function(name, country, category, year) {
13   this.setCitizen(name, country);
14   this.category = category;
15   this.year = year;
16   return this;
17 };
18 Winner.printDetails = function() {
19   console.log('Nobel winner ' + this.name + ' from ' +
20   this.country + ', category ' + this.category + ', year ' +
21   this.year);
22 };
23
24 var albert = Object.create(Winner)
25   .setWinner('Albert Einstein', 'Switzerland', 'Physics', 1921);
26 albert.printDetails();
```

>_ Console (beta) ⏪ 1 ⏪ 0 ⏳ 0 ⏪ 0

"Nobel winner Albert Einstein from Switzerland, category Physics, year 1921"

到jsfiddle.net測試

下圖是到jsbin.com測試

JavaScript ▾

```
var Citizen = {
  setCitizen: function(name, country){
    this.name = name;
    this.country = country;
    return this;
  },
  printDetails: function(){
    console.log('Citizen ' + this.name + ' from ' + this.country);
  }
};

var Winner = Object.create(Citizen);

Winner.setWinner = function(name, country, category, year){
  this.setCitizen(name, country);
  this.category = category;
  this.year = year;
  return this;
};

Winner.printDetails = function(){
  console.log('Nobel winner ' + this.name + ' from ' +
  this.country + ', category ' + this.category + ', year ' +
  this.year);
};

var albert = Object.create(Winner)
  .setWinner('Albert Einstein', 'Switzerland', 'Physics', 1921);

albert.printDetails();
```

Console

"Nobel winner Albert Einstein from Switzerland, category Physics, year 1921"

OO裡的self vs this

- 乍一看，很容易以為Python的**self**和JavaScript的**this**，似乎是相同的，後者可以是前者的隱式版本，提供給所有類實例方法。
- 實際上，**this** 和 **self** 有很大的不同！
- 比較如下：
 - Python的**self**是提供給每個類方法的變量：可以隨意命名，但不建議取其他名稱，代表該類實例。
 - 但是JS的**this**是一個關鍵字，它引用調用該方法的物件對象。此調用對象可以與方法的對象實例不同，並且JavaScript提供了調用和應用函數的方法。
 - ※JavaScript提供了**call**, **bind**, and **apply**等函數，來綁定**this**指到的物件！

相同作用的兩種語言的寫法

- 因此，JavaScript的this, 比Python的self 更具延展性，它提供了更大的自由度，還承擔了跟蹤調用上下文的責任，並且，如果您使用它，請確保在創建對象時始終使用new。

The screenshot shows two code editors side-by-side. The left editor is for Python 3.8.2 Shell, displaying the following code:

```
File Edit Shell Debug Options Window Help
=====
RESTART: C:/Users/rr/AppData/Local/Programs/Python/Python38-32/rrr.py ====
A title
123457
>>>
```

The right editor is for a browser-based JavaScript environment, displaying the following code:

```
File Edit Format Run Options Window Help
JavaScript
//Constructor
function Course(title,instructor,level,published,views){
    this.title = title;
    this.instructor = instructor;
    this.level = level;
    this.published = published;
    this.views = views;
    this.updateViews = function() {
        return ++this.views;
    }
}

//Create Objects
var a = new Course("A title", "A instructor", 1, true, 0);
var b = new Course("B title", "B instructor", 1, true, 123456);

//Log out objects properties and methods
console.log(a.title); // "A Title"
console.log(b.updateViews()); // "123457"
```

The browser's console output shows the results of running the JavaScript code.

Exercise 2-3

- Ex2-3Add1: 在python與JavaScript裡，放資料的容器，請比較那些相似之處。
- Ex2-3Add2:陣列，在程式語言中是非常重要的，請問在python的list與JavaScript的array，在使用上有那些相似的地方？請列表說明。
- Ex2-3Add3: Python與JavaScript在邏輯運算上有何異同？特別要注意那個大小寫問題？
- Ex2-3Add4: 在OOP設計時，Python的self，JavaScript的this有何異同？
- Ex2-3Add5:在OOP設計時，python與JS各是以什麼為基礎？請簡單比較其差異？
- Ex2-3Add6:關於浮點數，python與JS都是遵守那個規範？取整數的函數，在JS的parseInt()與python的int()，都是把浮點數作怎樣的處理？關於超大整數JS有最大範圍是多少？而python有特別處理，所以範圍會是多少？
- Ex2-3Add7: Python與JavaScript在function的定義上，分別是使用那個保留字？又是如何指明函數區塊的範圍？
- Ex2-3Add8: 兩種語言都允許對字串使用單引號和雙引號。如果要在字串印出單引號，就可以用兩個雙引號來指明。反之要印出雙引號，就用兩個單引號來指明。那麼，如果同時要印出單引與雙引或者要印出反斜線字元，是否會有問題？

2-4從實務上數據可視化的角度來看差異 (Differences in Practice)

- 認識JS和Python之間的語法差異非常重要。幸運的是，它們語法很相似。
- 命令式編寫程式、循環(loop)，條件(conditionals)，數據聲明(data declaration)和操作(manipulation)的實質是相同的。
- 在數據處理和數據可視化的專業領域中，情況尤其如此，這些語言的一流功能允許使用常見的習慣用法。
- 下面是從數據可視化機器的實務角度來看：
 - Python和JavaScript中的重要模式和慣用法的不全面列表。
 - 在可能的情況下，給出兩種語言之間的翻譯。

方法鏈(Method Chaining)

- JavaScript，常見的習慣用法是**方法鏈(method chaining)**：
 - 方法鏈接是隨著jQuery庫的受歡迎而普及，同時，也在D3中有很高的使用率。
 - 方法鏈接，涉及使用「點」表示法，從其自己的方法返回一個對象，以便在結果上調用另一個方法：

```
var sel = d3.select('#viz')
    .attr('width', '600px') ①
    .attr('height', '400px')
    .style('background', 'lightgray');
```

- 此程式，會先用 D3.select回傳當初呼叫的實體，然後呼叫attr方法，再呼叫attr方法。
- 在Python，關於方法鏈接則很少見！因為提倡每行一條語句，以保持簡單性和可讀性。這點是與JavaScript的設計想法，很不同！

Lab#2-4_方法鏈(method_chaining)的鏈呼叫在JavaScript的測試

■ 建議用JSBin方便直接實驗。且可調整js.

The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. On the left, the DOM tree displays a table with two rows and two columns. The first row contains th cells 'AA0000' and 'BB1111'. The second row contains td cells 'c33333' and 'd333'. To the right of the DOM tree, the script.js file is shown with its code:

```
<!-- index.html -->
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>First JavaScript</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/6.1.1/d3.min.js" type="text/javascript" src="script.js" async></script>
  </head>
  <body>
    div area01:
    <div id='viz'>
      <table id=viz2>
        <tr>
          <th>AA0000</th>
          <th>BB1111</th>
        </tr>
        <tr>
          <td>c33333</td>
          <td>d333</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

Below the main code editor, a smaller preview window titled 'script.js' shows the executed JavaScript code:

```
// script.js
var sel = d3.select('#viz2')
  .attr('width', '250px')
  .attr('height', '150px')
  .style('background', 'red');
```

列舉一個列表內容(Enumerating a List)

- 通常，在追蹤列表(list)的同時，追蹤項目的索引(item's index)，常常會很有用。因此，Python具有非常方便的內置列舉函數：(左圖程式)
- JavaScript的列表方法（例如forEach和函數的map, reduce, filter），並將疊代的項目及其索引提供給回調函數(callback function)：(右圖程式)

```
File Edit Format Run Options Window Help
names = ['Alice', 'Bob', 'Carol']
for i, n in enumerate(names):
    print('%d: %s' % (i, n))

Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
=====
===== RESTART: C:/Users/rr/AppData
0: Alice
1: Bob
2: Carol
>>>
```

```
File Edit Format Run Options Window Help
Console Sources Elements
top Filter
> //=====
var names = ['Alice', 'Bob', 'Carol'];
names.forEach(function(n, i){
    console.log(i + ': ' + n);
});
0: Alice
1: Bob
2: Carol
< undefined
> |
```

元組的反打包(Tuple Unpacking)

- 只有Python才可用的酷技巧之一，是使用元組拆包來切換變量
 - 例如兩變數交換： $(a, b) = (b, a)$
 - ※請注意，這裡的括號是可省略，仍代表是tuple。
- 可用於減少臨時變量的一種更實際的用途，例如在斐波那契函數中：

```
def fibonacci(n):
    x, y = 0, 1
    for i in range(n):
        print(x)
        x, y = y, x + y
```

- 如果要忽略其中的某變量，就可使用底線：

```
winner = 'Albert Einstein', 'Physics', 1921, 'Swiss'

name, _, _, nationality = winner
```

- 元組的打包有很多使用實例，是python的基本功能，但是，在JavaScript中不可用！

Collections

- Python非常好用的模塊：「collections模塊」
 -
- 在此僅提出其中的**4**種特殊容器資料結構來擴充Python的標準集。
 - **deque**：「雙端佇列」，該雙端佇列提供了一個類似列表的容器，在兩端都可快速的追加和彈出。
 - **OrderedDict** 「有序字典」，用於記住已添加的訂單項目；
 - **defaultdict**，它提供工廠功能來設置字典的默認值；
 - **Counter容器**，用於計算可哈希(hash)的對象。
- 這幾個都有很多實用場合，例如：

Lab#2-4 ch02-1c_python的好用集合物件

```
from collections import Counter, defaultdict, OrderedDict
items = ['F', 'C', 'C', 'A', 'B', 'A', 'C', 'E', 'F']
cntr = Counter(items)
print(cntr)
cntr['C'] -= 1 #C的數量減1個，就由3變為2
print(cntr)
"""

Out:
Counter({'C': 3, 'A': 2, 'F': 2, 'B': 1, 'E': 1})
Counter({'A': 2, 'C': 2, 'F': 2, 'B': 1, 'E': 1})
"""

d = defaultdict(int) #Sets dictionary default to an integer,with value 0
for item in items:
    d[item] += 1 #因為item-key若不存在，則default值已指定為0,所以+=1就是1
print(d)
"""

Out:
defaultdict(<class 'int'>, {'F': 2, 'C': 3, 'A': 2, 'B': 1, 'E': 1})
"""

e=OrderedDict(sorted(d.items(), key=lambda i: i[1]))
print(e)
"""

Out:
OrderedDict([('B', 1), ('E', 1), ('F', 2), ('A', 2), ('C', 3)])
"""
```

The screenshot shows the Python 3.8.2 Shell window with the following code and its execution results:

```
ch02-1c.py - C:/Users/rr/AppData/Local/Programs/Python/Python38-32/ch02-1c.py (3.8.2)
```

```
from collections import Counter, defaultdict, OrderedDict
items = ['F', 'C', 'C', 'A', 'B', 'A', 'C', 'E', 'F']
cntr = Counter(items)
print(cntr)
cntr['C'] -= 1 #C的數量減1個，就由3變為2
print(cntr)
"""

Out:
Counter({'C': 3, 'A': 2, 'F': 2, 'B': 1, 'E': 1})
Counter({'A': 2, 'C': 2, 'F': 2, 'B': 1, 'E': 1})
"""

d = defaultdict(int) #Sets dictionary default to an integer,with value 0
for item in items:
    d[item] += 1 #因為item-key若不存在，則default值已指定為0,所以+=1就是1
print(d)
"""

Out:
defaultdict(<class 'int'>, {'F': 2, 'C': 3, 'A': 2, 'B': 1, 'E': 1})
"""

e=OrderedDict(sorted(d.items(), key=lambda i: i[1]))
print(e)
"""

Out:
OrderedDict([('B', 1), ('E', 1), ('F', 2), ('A', 2), ('C', 3)])
"""

Python 3.8.2 Shell
```

```
>>>
==== RESTART: C:/Users/rr/AppData/Local/Programs/Python/Python38-32/ch02-1c.py =
Counter({'C': 3, 'F': 2, 'A': 2, 'B': 1, 'E': 1})
Counter({'F': 2, 'C': 2, 'A': 2, 'B': 1, 'E': 1})
defaultdict(<class 'int'>, {'F': 2, 'C': 3, 'A': 2, 'B': 1, 'E': 1})
OrderedDict([('B', 1), ('E', 1), ('F', 2), ('A', 2), ('C', 3)])
>>>
```

此處可看出，用count就可快速計算單字出現的個數！

JS的好用集合物件 _ 最新版名稱為lodash_

舊名稱是直接稱為 (Underscore下劃線函數庫)

- 下劃線函數庫可能是繼無所不在的jQuery之後最受歡迎的JavaScript庫，並且為JavaScript 數據視覺化的程序員提供了一系列功能強大的編程實用程序。
- 使用下劃線函數的最簡單方法是使用「內容分發網路(CDN)遠程加載」它（這些加載將由您的瀏覽器緩存，這使得通用庫非常高效），如下所示：
- <script src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.13.1/underscore-min.js"></script>

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/
underscore.js/1.8.3/underscore-min.js"></script>
```

- 下劃線函數具有許多有用的功能
 - 例如，有一個countBy方法，其作用與剛剛討論的Python的collections計數器相同：

cdnjs是個很常
用的cdn網站

```
var items = ['F', 'C', 'C', 'A', 'B', 'A', 'C', 'E', 'F'];
_.countBy(items) ①
Out:
Object {F: 2, C: 3, A: 2, B: 1, E: 1}
```

- 現在您知道為什麼將該庫稱為下劃線函數庫了吧！

Lab#2-4 JS的好用集合物件 lodash或舊名稱 Underscore下劃線函數庫 (※py有1秒變字典 Counter)

File 3/4 Add library Share

HTML CSS JavaScript Console Output

Login or Register Blog 1

Processor 3/4

```
//<script src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.13.1/underscore-min.js"></script>
// <!--
// <script src="https://cdn.jsdelivr.net/npm/lodash@latest/lodash.min.js"></script>
// -->

var items = ['F', 'C', 'C', 'A', 'B', 'A', 'C', 'E', 'F'];
// 使用 _.countBy 統計每個元素出現的次數
var result = _.countBy(items);
console.log(result); // 输出: { F: 2, C: 3, A: 2, B: 1, E: 1 }

const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = _.map(numbers, function(num) {
    return num * 2;
});
console.log(doubledNumbers); // 输出: [2, 4, 6, 8, 10]
```

Console

```
[object Object] {
  A: 2,
  B: 1,
  C: 3,
  E: 1,
  F: 2
}
```

```
[2, 4, 6, 8, 10]
```

方法鏈(method chaining) : JS的函數式陣列方法(Functional Array Methods)與python的串列推導(List Comprehensions)

- 前面說：規劃不錯的下劃線函數，其實已變得比較不需要了，因為自 Ecmascript 5加入JavaScript陣列函數方法之後有更直接的寫法了！
 - 由於JS 的for循環較難看(ugliness)，了解「函數式陣列」會更簡明。
 - 只要再結合JS的匿名函數(anonymous functions)，它可以實現流暢，富有表現力的編程。
 - 用方法鏈(method chaining)，也非常自然~ (※若用新寫法 => 更簡明)
 - 讓我們看一個例子：

The screenshot shows a browser window with the URL `jsbin.com/?html,js,console`. The tabs at the top include HTML, CSS, JavaScript, Console, and Output. The JavaScript tab is active, displaying the following code:

```
JavaScript
var nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var sum = nums.filter(function(o){ return o%2==1 })
  .map(function(o){ return o * o })
  .reduce(function(a, b){return a+b});
console.log('Sum of the odd squares is ' + sum);
```

The Console tab shows the output: "Sum of the odd squares is 165". A yellow callout box highlights the code structure with the following annotations:

**filter(過濾)出奇數項，
map(對應函數)是每項都平方
Reduce(減少)是兩項加成一項。**

相同問題，在python的寫法

- Python's powerful list comprehensions can emulate the previous example easily enough:
 - `nums = range(1,10+1)`
 - `odd_squares = [x * x for x in nums if x%2]`
 - `sum(odd_squares)`
- Out:
 - 165

●【check3】● ch02實作練習4_Lab#2-4_js方法鏈(method chaining)與python的串列推導(List Comprehensions)

■ 要求:其中的10 改為自己的學號末四碼。

```
12  
14 //var nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];  
15 var nums = Array.from({ length: 10 }, (v, i) => i+1);  
16 //my_log(nums);  
17 var sum = nums.filter(function(o){ return o%2 == 0 })  
18 .map(function(o){ return o * o})  
19 .reduce(function(a, b){return a+b});  
20 console.log('所有奇數平方和= ' + sum);  
21 my_log('所有奇數平方和= ' + sum);
```

所有奇數平方和= 165

File Add library Share

HTML JavaScript

所有奇數總和= 165

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Underscore.js</title>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.9.1/underscore-min.js"></script>  
<script src="https://rawgit.com/jiangjiunting/Ch02/master/lab2-4.js"></script>  
</head>  
<body>  
</body>  
</html>
```

Console

"所有奇數總和= 165"

相同問題：「所有奇數平方和」，上格是JavaScript的寫法，下一格換python的寫法

[40] :

```
1 # 程式目的：篩選出奇數，並將其平方，然後求和  
2 # python方法1: 用最佳的List comprehension(列表理解)來實現。  
3 nums = range(1, 10 + 1)  
4 odd_squares = [x * x for x in nums if x % 2 != 0]  
5 print("所有奇數平方和", sum(odd_squares))
```

所有奇數平方和 165

[41] :

```
1 # python方法2: 使用 filter(), map(), reduce() 聚合函數來實現  
2 from functools import reduce  
3 nums = list(range(1, 10 + 1))  
4 result = reduce(  
5     lambda x, y: x + y,  
6     map(lambda x: x**2,  
7          filter(lambda x: x % 2 != 0, nums)))  
8 print("所有奇數平方和", result)
```

所有奇數平方和 165

1
2
3
4
5

■ 前面Python的list comprehensions 裡面就相當在list執行多個函數.

- def is_odd(x):
 return x%2
-
- def sq(x):
 return x * x
-
- sum([sq(x) for x in l if is_odd(x)])

The screenshot shows a Python code editor window titled 'tt.py' and a shell window. The code defines two functions: 'is_odd' which returns the remainder of x divided by 2, and 'sq' which returns the square of x. It then creates a list 'l' with values 1, 2, 3, 4, 5, and calculates the sum of the squares of the odd numbers in 'l'. The shell window shows the output '35'.

```
def is_odd(x):
    return x%2

def sq(x):
    return x * x

l=[1,2,3,4,5]
z=sum([sq(x) for x in l if is_odd(x)])
print(z)
```

```
>>>
===== RESTART: C:/Users/rr/AppData/Lo
35
>>> |
```

■ 使用JavaScript執行多個函數，與python類似的設計也可以提高可讀性並簡化代碼

- var isOdd = function(x){ return x%2; };
- sum = l.filter(isOdd)
- ...

The screenshot shows a browser developer tools console. It has a 'JavaScript' tab where the user has typed a script block containing variable declarations and assignments. The 'Console' tab on the right shows the output '35'.

```
var l = [1, 2, 3, 4, 5];

var isOdd = function(x){ return x%2; };
var sq = function(x){ return x*x; };
var total = function(a,b){return a+b; };
z = l.filter(isOdd).map(sq).reduce(total);
console.log(z);
```

35

聚合函數(aggregation)的說明：Map, Reduce, and Filter with Python's Lambdas

- 儘管Python缺少匿名函數，但有**lambda**，是有參數的無名函數。是Python函數庫的有力補充，特別與其他函數結合使用時。
- Python的**lambda**接受「許多參數」並使用「冒號」分隔符定義功能塊，並對其返回操作。
 - 這與標準Python函數原理相同。只是沒有正式的寫出**def**，而且隱式返回結果。
- 以下示例顯示了函數編程中使用的一些**lambda**：

The screenshot shows a Python 3.8.2 Shell window. The code area contains the following Python script:

```
File Edit Format Run Options Window Help
from functools import reduce # if using Python 3+
nums = [0, 1, 2, 3, 4, 5]
odds = filter(lambda x: x % 2, nums)
odds_sq = map(lambda x: x * x, odds)
z=reduce(lambda x, y: x + y, odds_sq)
print(z)
```

The shell output area shows the following:

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/rr/AppData/Local/Programs
35
>>> |
```

閉包（Closure）

- 什麼是閉包？
- 閉包是指函式及其所處的詞彙環境（lexical environment）的組合。簡單來說，就是一個函式能夠「記住」它被創建時所處的環境，即使函式已經執行完畢，它仍然可以訪問並操作這些環境中的變量。

JavaScript Closures and the Module Pattern

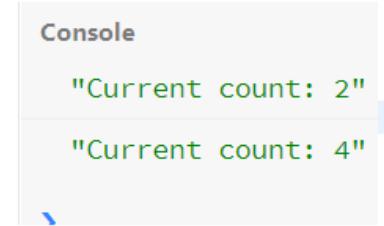
JS的閉包（Closure） & 模組模式Module Pattern

- JavaScript中的關鍵概念之一是閉包的概念，閉包本質上是一個嵌套的函數聲明，它使用在該函數範圍外（但不是全局（global）聲明的變量），這些變量在函數返回後仍然有效。
- 閉包允許使用許多非常有用的編程模式，並且是該語言的共同特徵。
- 在模塊模式常會見到閉包用法。
中我們已經發現的閉包用法
：
在訪問本質上為私有成員
變量的同時公開有限的API。
- 一個簡單的關閉示例是這個
小計數器：

```
JavaScript ▾

function Counter(inc) {
  var count = 0;
  var add = function() {
    count += inc;
    console.log('Current count: ' + count);
  };
  return add;
}

var inc2 = Counter(2);
inc2();
//Out:
//Current count: 2
inc2();
//Out:
//Current count: 4
```



Lab#02_JS的閉包(Closure)例子,此狀況最佳解法是let

※注意：(let是區塊作用域，var是函數作用域)

- 左圖是錯誤示範！因為這樣會是5個6，而不是每秒加1的1到5！

```
JavaScript ▾  
for (var i = 1; i <= 5; i++) {  
    setTimeout(function timer() {  
        console.log(i);  
    }, i * 1000);  
}
```

是錯誤示範，會是5
個6，而不是每秒加
1的1到5！

Console
6
6
6
6
6
6

```
JavaScript ▾  
for (var i = 1; i <= 5; i++) {  
    (function(j) {  
        setTimeout(function timer() {  
            console.log(j);  
        }, j * 1000);  
    })(i);  
}
```

›

Console
1
2
3
4
5

```
JavaScript ▾  
for (let i = 1; i <= 5; i++) {  
    setTimeout( function timer() {  
        console.log(i);  
    }, i * 1000);  
}
```

›

Console
1
2
3
4
5

- 上左圖：這的確是錯誤的。由於 `console.log(i)` 中的 `i` 會存取的範疇是 `for` 所在的範疇（目前看起來是全域範疇，因為 `var` 宣告的變數不具區塊範疇的特性），因此當 1 秒、2 秒...5 秒後執行 `console.log(i)` 時，就會去取 `i` 的值，而此時 `for` 迴圈已跑完，`i` 變成 6，因此就會每隔一秒印出一個「6」
- 上中圖：為每次的疊代，都建立一個新的函式域，然後把(`i`)傳入這個「閉包(函數作用區域)」。（其實，本題，只想要在`for loop`的區塊建立一個區塊變數！所以用方法三會更簡明！）
 - `for (var i = 1; i <= 5; i++) { (function(j) { setTimeout(function timer() { console.log(j); }, j * 1000); }) (i); }`
- 上右圖：為了建立區塊內變數，可以較新的指令寫法：`let`來解決。`let` 會在每次疊代，在 ~~同一區塊域~~，新建變數 `i`，並將上一次迭代的結果作為這一次的初始值。
 - `for (let i = 1; i <= 5; i++) { setTimeout(function timer() { console.log(i); }, i * 1000); }`

Lab#02_JS的顯示模塊模式（Revealing module pattern）的例子—具有正式命名的模組模式。

顯示模組模式（Revealing Module Pattern）：

- 具有正式命名，經由建立一個模組實體（module instance，如下範例的 `foo`），來調用內層函式 `doSomething` 和 `doAnother`。
- 而內層函式由於具有閉包的特性，因此可存取外層的變數和函式（`something` 與 `another`）。

The screenshot shows a browser's developer tools console. On the left, there is a code editor window containing JavaScript code. On the right, the console output is displayed.

```
JavaScript ▾
function CoolModule() {
  var something = 'cool';
  var another = [1, 2, 3];

  function doSomething() {
    console.log(something);
  }

  function doAnother() {
    console.log(another.join(' ! '));
  }

  return {
    doSomething: doSomething,
    doAnother: doAnother,
  };
}

var foo = CoolModule();
```

Console

```
"cool"
"1 ! 2 ! 3"
```

透過模組模式，可選擇有對外公開的屬性， 本例是稱為API

- 透過模組模式，可隱藏私密資訊，並選擇一個，可對外公開的 API。
- 例如可以擴展 Counter，加入對外公開的 API。
- 該技術是 JavaScript 模塊和許多簡單庫的基礎。
 - 本質上講，就是「在隱藏私有方法和變量的同時，選擇性地公開了公共方法」，這在編程領域通常被視為良好實踐。

The screenshot shows a browser's developer tools console window. At the top, there are tabs for 'File', 'Add library', and 'Share'. Below that, a 'JavaScript' dropdown is selected. The main area contains the following code:

```
function Counter(inc) {  
    var count = 0;  
    var api = {};  
    api.add = function() {  
        count += inc;  
        console.log('Current count: ' + count);  
    }  
    api.sub = function() {  
        count -= inc;  
        console.log('Current count: ' + count);  
    }  
    api.reset = function() {  
        count = 0;  
        console.log('Count reset to 0');  
    }  
  
    return api;  
}  
  
cntr = Counter(3);  
cntr.add(); // Current  
cntr.add(); // Current  
cntr.sub(); // Current  
cntr.reset(); // Cour
```

To the right of the code, the 'Console' tab is active, showing the following log entries:

- "Current count: 3"
- "Current count: 6"
- "Current count: 3"
- "Count reset to 0"

Python閉包的實例_使用率不及JS：

- 閉包在JavaScript中有各樣的用途
- Python也有閉包，但是使用率遠不及JS。為了演示用python設計與之前JavaScript計數器，類似效果的閉包程式。如下左圖的程式：※但是會得到錯誤訊息「UnboundLocalError」，最簡單的解決方案是加入nonlocal的宣告。

A screenshot of a Python IDE showing a script named 'y.py'. The code defines a function `get_counter` which returns a closure. When the closure's `add` method is called, it attempts to modify the local variable `count`, which causes a `UnboundLocalError`. A red box highlights the error message in the shell window.

```
def get_counter(inc):
    count = 0
    def add():
        count += inc
        print('Current count: ' + str(count))
    return add
v=get_counter(3)
v()
v()
v()

File Edit Format Run Options Window Help
```

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
ine 4, in add
    count += inc
UnboundLocalError: local variable 'count' referenced before assignment
>>> Ln: 51 Col: 4
```

A screenshot of a Python IDE showing the same script after adding the `nonlocal` keyword to the `count` assignment in the closure's `add` method. This prevents the local `count` from being modified and instead uses the outer `count` variable. The shell window shows the correct sequence of counts: 3, 6, 9.

```
def get_counter(inc):
    count = 0
    def add():
        nonlocal count
        count += inc
        print('Current count: ' + str(count))
    return add
v=get_counter(3)
v()
v()
v()

File Edit Format Run Options Window Help
```

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window
1/Programs/Python/Python38-
Current count: 3
Current count: 6
Current count: 9
>>>
```

This Is That

- 您會發現很多JavaScript是由於閉包(closure)和“this”關鍵字而發生問題！
- 如果您希望在子函數中引用外部範圍的“this”，則必須使用代理，因為子函數的“this”，將根據上下文進行綁定。**慣例是使用“that”來參考“this”。**
- 這樣的代碼比具解釋性，也較少混亂：

```
function outer(bar) {  
    this.bar = bar;  
    var that = this;  
    function inner(baz) {  
        this.baz = baz * that.bar; ❶  
        // ...  
    }  
}
```

- 以上是在dataviz工作中，將會大量使用的模式

Exercise 2-4

- Ex2-4Add1: 請從數據可視化機器的實務角度來看，python很常用，但是javascript卻很少使用的技術？反之是否有javascript比較常用，而python卻很少用的技術呢？
- Ex2-4Add2:若有一個像陣列的容器，請問要如何印出全部的項？請就pyhton與JavaScript分別寫出這個簡單又重要的程式。
- Ex2-4Add3: Filter , Map, Reduce,是在『函數式程式設計』中常用的指令。請分別用pyhton與javascript，解決相同的問題：問題是先有一個容器物件內容是1~5，然後filter(過濾)出奇數項，再使用map(對應函數)將每項值都平方，最後再Reduce(減少)成只有一項的總和。

2-5：重點快速查閱表 (A Cheat Sheet)

- 為了作為方便參考的指南，圖2-2至圖2-7包括一組備忘單，用於轉換Python和JavaScript之間的基本操作。
- 重點快速查閱表_1: 基本語法

JavaScript

```
<script src="lib/ vizUtils.js" >  
</script>  
  
(function(foolib){  
... // module pattern  
}(window.foolib = window.foolib || {}));  
  
var foo; // undefined variables  
var bar=20;  
  
var foo = function(a, b){  
// clunky defaults, fixed in ES6!  
b = typeof b !== 'undefined' ? b : 10;  
var x = a%b;  
...  
return result;  
};
```

Python

```
import vizutils as viz  
from vizutils import gblur  
  
bar = 20  
  
def foo(a, b=10):  
    x = a%b  
    ...  
    return result
```

significant whitespace!

Figure 2-2. Some basic syntax

重點快速查閱表_2:基本邏輯布林

JavaScript

```
var x = false;  
var y = true;  
var l = []  
  
if(!x && y === x){...}  
  
if(l.length === 0){...}
```

Python

```
x = False  
y = True  
l = []  
  
if not x and y == x:  
  
if l: ...
```

Figure 2-3. Booleans

Loop(重覆)與疊代的聚和運算+直接的小數據與輸出

JavaScript

```
camelCase vs  
underscored  
  
var studentData = [  
    {'name': 'Bob',  
     'scores':[68, 75, 56, 81]},  
    {'name': 'Alice',  
     'scores':[75, 90, 64, 88]},  
    ...];  
  
anonymous functions  
  
studentData.forEach(function(sdata){  
    var av = sdata.scores  
        .reduce(function(prev, current){  
            return prev+current;  
        },0) / sdata.scores.length;  
    sdata.average = av;  
}  
  
first-class functional methods  
  
console.log(sdata.name + " scored " +  
    sdata.average);
```

Python

```
student_data = [  
    {'name': 'Bob',  
     'scores':[68, 75, 56, 81]},  
    {'name': 'Alice',  
     'scores':[75, 90, 64, 88]},  
    ...]  
  
line-break  
  
s_data = student_data  
for data in s_data.items():  
    av = sum(data['scores'])\n        /float(len(data['scores']))  
    sdata['average'] = av  
  
print("%s scored %d%\n      (sdata.name, sdata.average));
```

```
while(i < 10){  
    ...  
}  
do {  
    ...  
}  
while(i < 10);
```

```
while i < 10:  
    ...  
    while True:  
        if i >= 10:  
            break
```

Figure 2-4. Loops and iterations

Lab#2-5-1_重要實例：自訂小數據，計算平均並印出。

JavaScript ▾

```
studentData=[  
    {'name':'Bob',  
     'scores':[90,80,70,60]},  
    {'name':'Alice',  
     'scores':[45,75,45,96]},  
]  
  
console.log("before-----")  
studentData.forEach(function(sdata){  
    console.log(sdata.name+ "--scored-->" + sdata.average);  
})  
  
studentData.forEach(function(sdata){  
    var av=sdata.scores  
    .reduce(function(prev,current){  
        return prev+current;  
    },0) / sdata.scores.length;  
  
    sdata.average=av;  
})  
  
console.log("after-----")  
studentData.forEach(function(sdata){  
    console.log(sdata.name+ "--scored-->" + sdata.av)  
})
```

Console

```
"before-----"  
"Bob--scored-->undefined"  
"Alice--scored-->undefined"  
"after-----"  
"Bob--scored-->75"  
"Alice--scored-->65.25"
```

疊代運算(For each)

```
File Edit Format Run Options Window Help  
student_data=[  
    {'name':'Bob',  
     'scores':[90,80,70,60]},  
    {'name':'Alice',  
     'scores':[45,75,45,96]},  
]  
  
s_data=student_data  
print("before:",s_data)  
for data in s_data:  
    av=sum(data['scores'])/ float(len(data['scores']))  
    data['average']=av  
print("after:",s_data)  
  
Python 3.8.2 Shell  
File Edit Shell Debug Options Window Help  
before: [{name: 'Bob', scores: [90, 80, 70, 60]}, {name: 'Alice', scores: [45, 75, 45, 96]}]  
after: [{name: 'Bob', scores: [90, 80, 70, 60], average: 75.0}, {name: 'Alice', scores: [45, 75, 45, 96], average: 65.25}]  
>>> |
```

重點快速查閱表_3: if 結構

JavaScript

```
if(x === 'foo'){
    ...
} else if(x === 'bar'){
    ...
} else{
    ...
}

if(x === foo && y !== bar){...}

if(['foo', 'bar', 'baz']
    .indexOf(s) != -1){...

switch(foo){
    case bar:
        ...
        break;
    case baz: ...
    default:
        return false;
}
```

Python

```
if x == 'foo':
    ...
elif x == 'bar':
    ...
else:
    ...

if x == foo and y != bar:
    ...
if s in ['foo', 'bar', 'baz']:
    ...
```

Figure 2-5. Conditionals

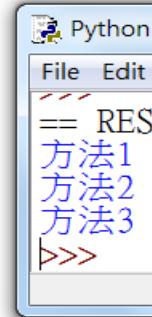
重點快速查閱表_4: Loop(重覆)與疊代的聚和運算+直接的小數據與輸出(方法鏈method chain)

JavaScript	Python
<i>camelCase vs underscore</i>	
<pre>var studentData = [{'name': 'Bob', 'scores':[68, 75, 56, 81]}, {'name': 'Alice', 'scores':[75, 90, 64, 88]}, ...];</pre>	<pre>student_data = [{'name': 'Bob', 'scores':[68, 75, 56, 81]}, {'name': 'Alice', 'scores':[75, 90, 64, 88]}, ...]</pre>
<i>anonymous functions</i>	
<pre>studentData.forEach(function(sdata){ var av = sdata.scores .reduce(function(prev, current){ return prev+current; },0) / sdata.scores.length; sdata.average = av;</pre>	<pre>s_data = student_data for data in s_data.items(): av = sum(data['scores'])\n /float(len(data['scores'])) sdata['average'] = av</pre>
<i>first-class functional methods</i>	
<pre>console.log(sdata.name + " scored " + sdata.average);</pre>	<pre>print("%s scored %d%\n" (sdata.name, sdata.average));</pre>
<pre>while(i < 10){ ... } do { ... } while(i < 10);</pre>	<pre>while i < 10: ... while True: if i >= 10: break</pre>

Figure 2-4. Loops and iterations

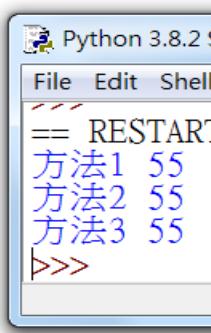
Lab#2-5-2_JS的重覆結構，計算 $1+2+\dots+10$ 的總和的實驗(for, while，與無窮迴圈)，實驗時，請把10改為「自己的座號加100」。

```
tt.py - C:/Users/rr/AppData/Local/Programs/Python  
File Edit Format Run Options Window Help  
  
#方法1  
t=0  
for i in range(11):  
    t+=i  
print("方法1", t)  
  
#方法2  
t=0  
i=1  
while i<=10:  
    t+=i  
    i+=1  
print("方法2", t)  
  
#方法3  
t=0  
i=1  
while True:  
    if i>10:  
        break  
    t+=i  
    i+=1  
print("方法3", t)
```



The screenshot shows the Python 3.8.2 Shell window. The title bar says "Python 3.8.2 Shell". The menu bar includes "File", "Edit", "Shell". The command line starts with "==== RESTART: <...>". The output area displays the results of the three methods:

```
方法1 55  
方法2 55  
方法3 55
```



```
JavaScript + No-Library (pure JS) ▾

1
2   t=0
3   ▶ for (const i of Array(11).keys()) {
4     //console.log(i+1);
5     t+=i;
6   }
7   console.log(t)
8
9   t=0; i=1;
10  ▶ while (i<=10){
11    t+=i;
12    i+=1;
13  }
14  console.log(t)
15
16  t=0; i=1;
17  ▶ while (true){
18    t+=i;
19    if (i>=10) break;
20    i+=1;
21  }
22  console.log(t)
```

>_ Console (beta) ⏱ 3
▶ "Running fiddle"
55
55
55
>_

● ch02實作練習5_ Lab#2-5-2_JS的重覆結構，計算 $1+2+..+10$ 的總和的實驗(for, while，與無窮迴圈)，實驗時，請把10改為「自己的座號加100」。

```
28     while (i<=10){
29         t+=i;
30         i+=1;
31     }
32     my_log("方法2:"+t)
33
34 //方法3
35 t=0; i=1;
36 while (true){
37     t+=i;
38     if (i>=10) break;
39     i+=1;
40 }
41 my_log("方法3:"+t)
```

方法1:55
方法2:55
方法3:55

相同問題：「 $1+2+..+10$ 的總和」，上格是JavaScript的寫法，下一格換python的寫法

```
In [2]: 1 #方法1
2 t=0
3 for i in range(11):
4     t+=i
5 print("方法1", t)
6
7 #方法2
8 t=0
9 i=1
10 while i<=10:
11     t+=i
12     i+=1
13 print("方法2", t)
14
15 #方法3
16 t=0
17 i=1
18 while True:
19     if i>10:
20         break
21     t+=i
22     i+=1
23 print("方法3", t)
24
```

方法1 55
方法2 55
方法3 55

```
In [3]: 1 # ※注意：選擇之前，請「先點成藍色(非編輯模式)」，再按住 Shift，再移動鍵盤的 上、下鍵，再 ctrl+c 一定要用 鍵盤，才會複製成功！
```

重點快速查閱表_5:物件容器的實驗

JavaScript

```
[var l = [1, 2, 3, 4];
l.push('foo'); // [...4, 'foo']
l.pop(); // 'foo', l=[..., 4]
l.slice(1,3) // [2, 3]
l.slice(-3, -1) // [2, 3]

l.map(function(o){ return o*o;})
// [1, 4, 9, 16]
```

```
[d = {a:1, b:2, c:3};
d.a === d['a'] // 1
d.z // undefined

// OLD BROWSERS
for(key in d){
  if(d.hasOwnProperty(key)){
    var item = d[key];

// NEW AND BETTER
Object.keys(d).forEach(key, i){
  var item = d[key];
```

Python

```
[l = [1, 2, 3, 4]
l.append('foo') # [...4, 'foo']
l.pop() # 'foo', l=[..., 4]
l[1:3] # [2, 3]
l[-3:-1] # [2, 3]
l[0:4:2] # [1, 3] (stride of 2)

[o*o for o in l]
// [1, 4, 9, 16]
```

```
[d = {'a':1, 'b':2, 'c':3}
d['a'] # 1
d.get('z') # NoneType
d['z'] # KeyError!

for key, value in d.items(): ...
for key in d:
  for value in d.values():...
```

列表(List)

字典(Dict)

Figure 2-6. Containers

Lab#2-5-3a_python的容器list放入移走印出的實驗

The screenshot shows a Python IDE interface with two main windows. On the left is a code editor window titled "tt.py - C:/Users/rr/AppData/Local/Programs/Python/Python38-32" containing Python code. On the right is a "Python 3.8.2 Shell" window showing the output of the code execution.

```
l=[1,2,3,4]; print(l)
l.append('foo'); print(l)
l.pop(); print(l)
print(l[1:3])
print(l[-3:-1])
print(l[0:4:2])

z=[o**o for o in l]
print('z=>', z)

d={'a': 1, 'b':2, 'c':3}
print('d=>', d)

print(d['a'])
print(d.get('a'))

print(d.get('z')) #None
#print(d['z']) #KeyError:

for key,value in d.items():
    print(key,value)
for key in d:
    print(key)

for value in d.values():
    print(value)
```

The shell output shows the results of each print statement:

```
[1, 2, 3, 4]
[1, 2, 3, 4, 'foo']
[1, 2, 3]
[2, 3]
[2, 3]
[1, 3]
z=> [1, 4, 9, 16]
d=> {'a': 1, 'b': 2, 'c': 3}
1
1
None
a 1
b 2
c 3
a
b
c
1
2
3
>>>
```

Lab#2-5-3b_JS的容器list放入移走印出的實驗

方法一:用jsfiddle.net/測試

JavaScript + Processing.js 1.4.7▼

```
1 var l=[1,2,3,4] ; console.log(l);
2 l.push('foo') ; console.log(l);
3 l.pop() ; console.log(l);
4 console.log(l.slice(1,3) )
5 console.log(l.slice(-3,-1) )
6 console.log(l.slice(0,4,2) )
7
8 z=l.map(function(o){ return o*o;})
9 console.log('z=>', z)
10
11 d={'a': 2, 'b':5,'c':9}
12 console.log('d=>', d)
13
14 console.log(d['a'])
15 console.log(d['a'] === d.a) //兩種寫法 true
16 console.log(d.z) //undefined
17 console.log(d['z']) //undefined
18
19 //較舊的寫法，會取出key的index字串的寫法
20 console.log("print dictionary...")
21 for ( var key in d ) {
22 if (d.hasOwnProperty(key)){
23 var item=d[key];
24 console.log(key, ":", item);
25 }
26 }
27 //較新的寫法
28 console.log("obj=>", Object.keys(d));
29 console.log("values =>", Object.values(d));
30
31 Object.keys(d).forEach(function (item,i) {
32 console.log(i,":")
33 console.log(item); // key
34 console.log(d[item]); // value
35 });
36
```

```
>_ Console (beta) ① 27 ② △
  "Running fiddle"
[1, 2, 3, 4]
[1, 2, 3, 4, "foo"]
[1, 2, 3, 4]
[2, 3]
[2, 3]
[1, 2, 3, 4]
"z=>", [1, 4, 9, 16]
"d=>", {
  a: 2,
  b: 5,
  c: 9
}
2
true
undefined
undefined
"print dictionary..."
"a", ":", 2
"b", ":", 5
"c", ":", 9
"obj=>", ["a", "b", "c"]
"values =>", [2, 5, 9]
0, ":"
```

Instructor: Jiun-Ting Jiang

方法二:用JSbin.com測試

JavaScript ▾

```
var l=[1,2,3,4] ; console.log(l);
l.push('foo') ; console.log(l);
l.pop() ; console.log(l);
console.log(l.slice(1,3) )
console.log(l.slice(-3,-1) )
console.log(l.slice(0,4,2) )

z=l.map(function(o){ return o*o;})
console.log('z=>', z)

d={'a': 2, 'b':5,'c':9}
console.log('d=>', d)

console.log(d['a'])
console.log(d['a'] === d.a) //兩種寫法 true
console.log(d.z) //undefined
console.log(d['z']) //undefined

//較舊的寫法，會取出key的index字串的寫法
console.log("print dictionary...")
for ( var key in d ) {
  if (d.hasOwnProperty(key)){
    var item=d[key];
    console.log(key, ":", item);
  }
}
//較新的寫法
console.log("obj=>", Object.keys(d));
console.log("values =>", Object.values(d));

Object.keys(d).forEach(function (item,i) {
  console.log(i,":")
  console.log(item); // key
  console.log(d[item]); // value
});
```

```
Console
[1, 2, 3, 4]
[1, 2, 3, 4, "foo"]
[1, 2, 3, 4]
[2, 3]
[2, 3]
[1, 2, 3, 4]
"z=>"
[1, 4, 9, 16]
"obj=>"
[{"a": 2, "b": 5, "c": 9}]
2
true
undefined
undefined
"print dictionary..."
"a"
":"
2
"b"
":"
5
"c"
":"
9
"obj=>"
["a", "b", "c"]
"values =>"
[2, 5, 9]
0
":"
"a"
2
1
":"
"b"
5
2
":"
"c"
9
```

重點快速查閱表_6: OO的實驗_基本題+有繼承 (Inheritance)

JavaScript

```
var Foo = {  
    initFoo: function(bar){  
        this.bar = bar;  
        return this;  
    }  
};  
  
var Baz = Object.create(Foo);  
  
Baz.initBaz = function(bar, qux){  
    this.initFoo(bar);  
    this.qux = qux;  
    return this;  
};  
  
var baz = Object.create(Baz)  
    .initBaz('answer', 42);
```

Figure 2-7. Classes and prototypes

Python

```
1  class Foo(): # Foo(object), Object是最基  
2      def __init__(self,bar):  
3          self.bar=bar  
4  class Baz(Foo): #繼承 Foo  
5      def __init__(self , bar,qux):  
6          super().__init__(bar) #呼叫父層()  
7          self.qux=qux  
8  baz=Baz('aaanswer',60)  
9  baz.bar #印出 aaanswer  
10
```

單純以OO的程式碼來看
, python真的很簡潔 !

Exercise 2-5

- **Ex2-5Add1:**一個簡單問題同時寫出py與js的程式：重覆結構的實驗，計算 $1+2+..+10$ 的總和的實驗(**for**, **while**，與無窮迴圈)，實驗時，請把**10**改為「自己的座號加**100**」。
- **Ex2-5Add2:**一個簡單問題同時寫出py與js的程式：先自訂小型數據假設有兩個學生的名字與一系列成績，請計算平均並印出。
- **Ex2-5Add3:**一個簡單問題同時寫出py與js的程式：宣告一個容器，假定原來的數據是**1,2,3,4**，後來加入字串'**foo**'到容器又拿出來。然後印出這個容器的全部內容。

結論

- 在本章已經說明**JavaScript**和**Python**的許多通用語法。
- 在兩種語言中，寫程式，疊代，條件和基本數據的操作都很簡單，而函數的轉換也很簡單。
- 如果已經會其中一種語言，那麼學另一語言就很快。

Exercise 2

- **Ex2-1(章末):** 關於py與js，請就他們相同點、相異點至少各舉出3點。
- **Ex2-2(章末):**結構化程式設計有三種結構，同時 py和js有許多通用語法都很像。請同時用這兩種語言，寫出可以利用三種結構來解決相同問題的程式。
- **Ex2-3(章末):**關於py與js都把函數當作一等公民，而且也都有一級物件(first-class object)，都可以把函數直接當作參數。同時這兩種語言也都可以用省略明確定義的匿名函數(Anonymous Function)來設計程式。請同時用這兩種語言，寫出可以用匿名函數(Anonymous Function)示範來解決相同問題的程式。



- Thank for your attention.



Lab#2-5_JS簡單的OO，有屬性與方法

The screenshot shows a browser's developer tools with the 'Console' tab selected. The console output displays the execution of a script that defines a constructor function 'Person', creates two instances 'person1' and 'person2', and logs their 'firstName' properties to the console.

```
> var Person = function (firstName) {
  this.firstName = firstName;
  console.log('Person產生了實例');
};

var person1 = new Person('AAA'); //建立物件person1
var person2 = new Person('BBB'); // 建立物件person2

// 顯示物件的 firstName 屬性
console.log('person1 is ' + person1.firstName); // 會記錄 "person1 is AAA"
console.log('person2 is ' + person2.firstName); // 會記錄 "person2 is BBB"

Person產生了實例
Person產生了實例
person1 is AAA
person2 is BBB
< undefined
> |
```

Lab#2-5_匿名函數(Anonymous Function) 的簡易實驗

- 兩種語言的函數，都是一級物件(first-class object)，所以可以當作參數。
- 在JS是把'function'直接寫在參數區，
- Python則有lambda的保留字。

```
JavaScript ▾
//有函數名稱的寫法
/*function say(word) {
  console.log(word);
}

function execute(someFunction, value) {
  someFunction(value);
}
execute(say, "Hello");*/

//匿名函數(Anonymous Function )的寫法
function execute(someFunction, value) {
  someFunction(value);
}
execute(function(word){console.log(word);}, "Hello");
```

The screenshot shows a Python IDE interface with two code snippets side-by-side.

Left Snippet (JavaScript):

```
//有函數名稱的寫法
/*function say(word) {
  console.log(word);
}

function execute(someFunction, value) {
  someFunction(value);
}
execute(say, "Hello");*/

//匿名函數(Anonymous Function )的寫法
function execute(someFunction, value) {
  someFunction(value);
}
execute(function(word){console.log(word);}, "Hello");
```

Right Snippet (Python):

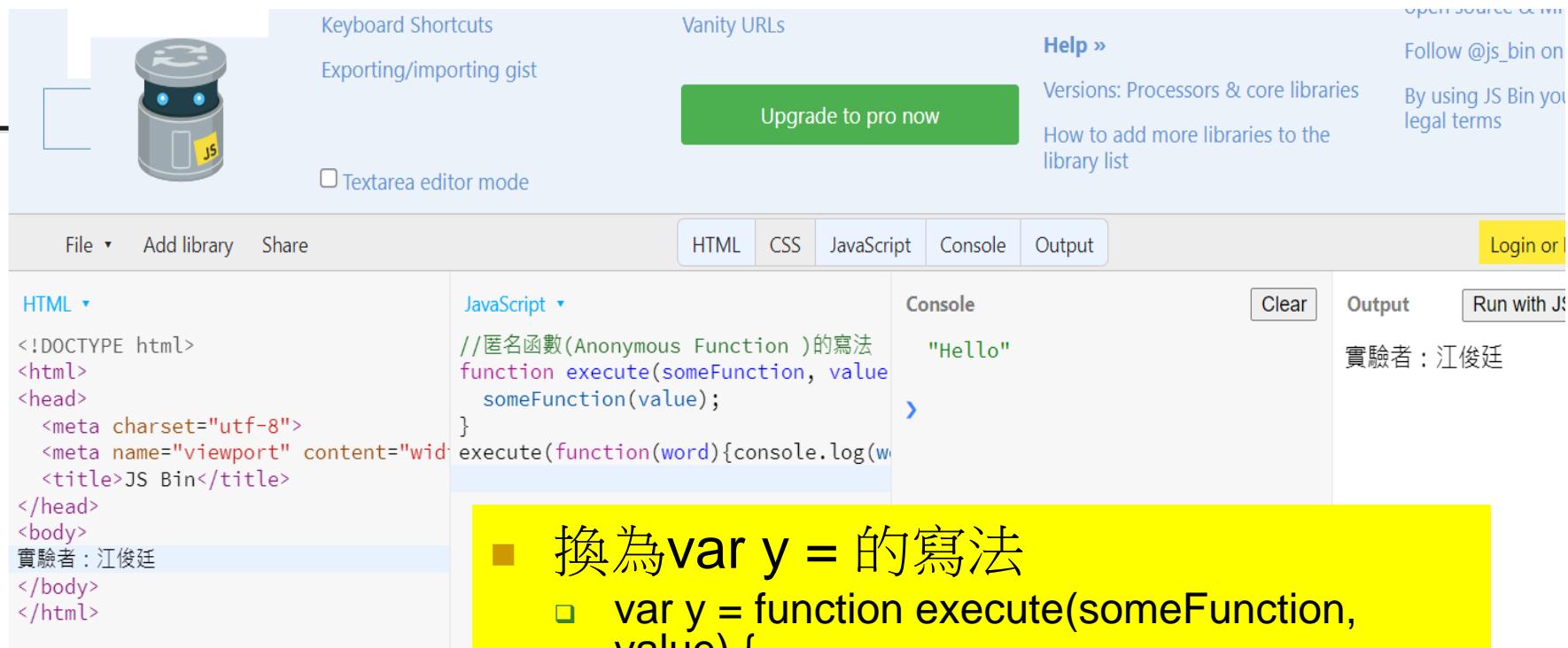
```
tttt.py - C:/Users/rr/AppData/Local/Programs/Python/Python38-32/tttt.py
File Edit Format Run Options Window Help
...
## //有函數名稱的寫法
def say(word):
    print(word)

def execute(someFunction, value):
    someFunction(value)
execute(say, "Hello");

## //匿名函數(Anonymous Function )的寫法
def execute(someFunction, value):
    someFunction(value)
execute(lambda word: print(word), "Hello")
```

The Python code demonstrates two ways to define a function: with a name (def) and anonymously (lambda). The output window shows the word "Hello" being printed twice.

Lab#2-5b_匿名函數(Anonymous Function)的簡易實驗_用jsbin



The screenshot shows the jsbin interface. The top navigation bar includes links for Keyboard Shortcuts, Vanity URLs, Help, Versions, and legal terms. It also features a 'Upgrade to pro now' button and a 'Textarea editor mode' checkbox. The main workspace has tabs for HTML, CSS, JavaScript, Console, and Output. The HTML tab contains the following code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JS Bin</title>
</head>
<body>
  實驗者：江俊廷
</body>
</html>
```

The JavaScript tab contains the following code:

```
//匿名函數(Anonymous Function )的寫法
function execute(someFunction, value) {
  someFunction(value);
}
execute(function(word){console.log(word);}, "Hello")
```

The Console output shows the result: "Hello". The Output tab shows the text "實驗者：江俊廷".

■ 換為var y = 的寫法

- var y = function execute(someFunction, value) {
- someFunction(value);
- }
- y (function(word){console.log(word);}, "Hello");

Chrome DevTools 開發者工具，改中文方法

■ 更改 Chrome DevTools 語言的步驟：

- 打開 Chrome 瀏覽器。
- 開發者工具 (Developer Tools) (按下 Ctrl + Shift + I, 或 F12)
- 點擊右上角的 齒輪圖示 (設置) 打開偏好設置。
- 在偏好設置面板中，找到 **Languages** (語言) 選項，選擇「中文」。
- 啟開發者工具，界面會切換為中文。

■ ※可打開DevTools方法主要是：

- 1. 快捷鍵：
 - 按下 Ctrl + Shift + I 或 + J 或 F12
- 2. 網頁內，右鍵選擇「檢查」
- 3. Chrome菜單右上角的三點選單 (:) /更多工具 (More Tools) /開發者工具 (Developer Tools)。

