

目次

Module 1. 開發環境：Anaconda、Jupyter 及爬蟲專案實務	1
Module 2. 正規表達式 (Regular Expression) 說明	14
Module 3. Chrome Developer Tool	20
Module 4. 請求 (Request)	28
Module 5. 套件 requests	30
Module 6. 淺談 HTML 與 CSS Selector (選擇器)	34
Module 7. 套件 BeautifulSoup 4	38
Module 8. cookie 用於 requests	41
Module 9. 案例：PTT_NBA_看板主頁與內頁	42
Module 10. 套件 Selenium (一)	46
Module 11. 套件 Selenium (二)	49
Module 12. 套件 Selenium (三)	56
Module 13. ActionChains	60

- GitHub 專案連結

https://github.com/telunyang/python_web_scraping

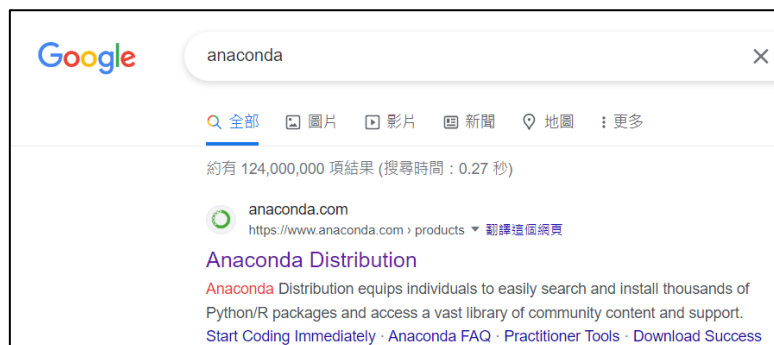
Module 1. 開發環境：Anaconda、Jupyter 及爬蟲專案實務

安裝 Anaconda

參考連結

[1] Anaconda - Individual Edition

<https://www.anaconda.com/products/individual>



圖：可以先至 google 檢索「anaconda」，找到 Anaconda Distribution 的連結

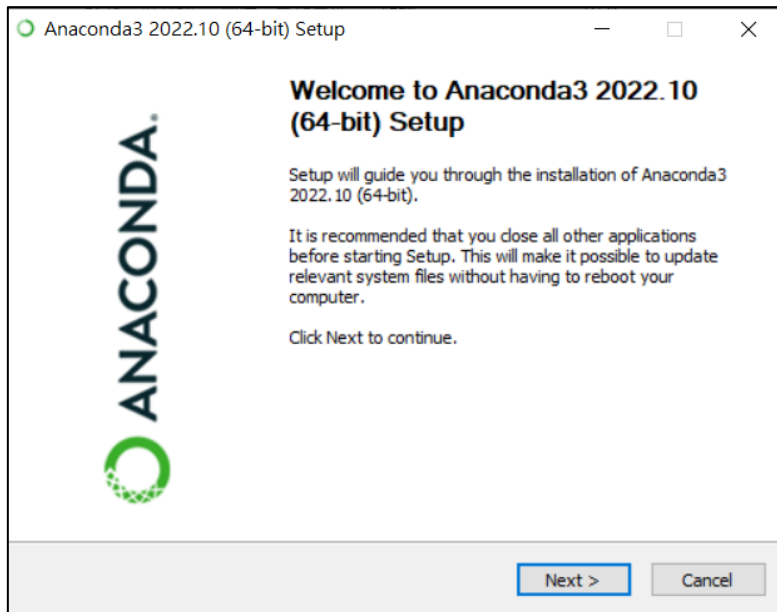


圖：在 Windows 選項下，選擇 64-Bit Graphical Installer，並下載下來

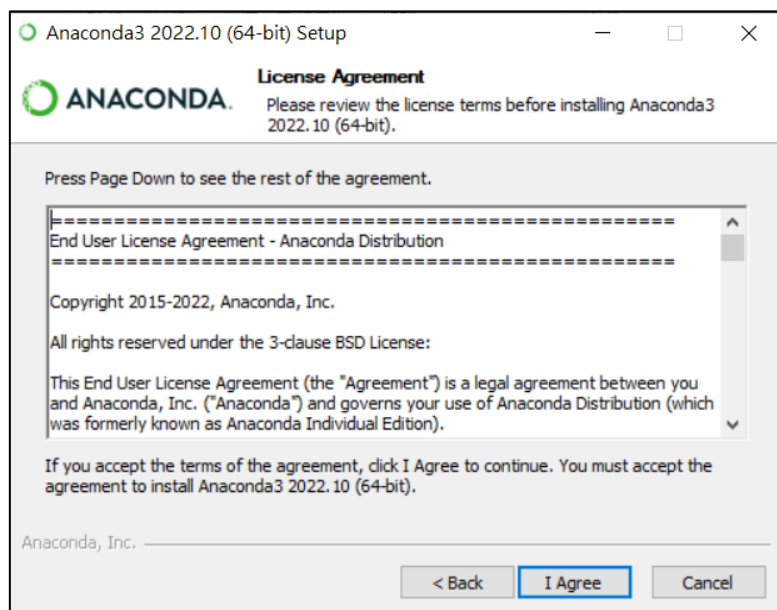


Anaconda3-202
2.10-Windows-x
86_64.exe

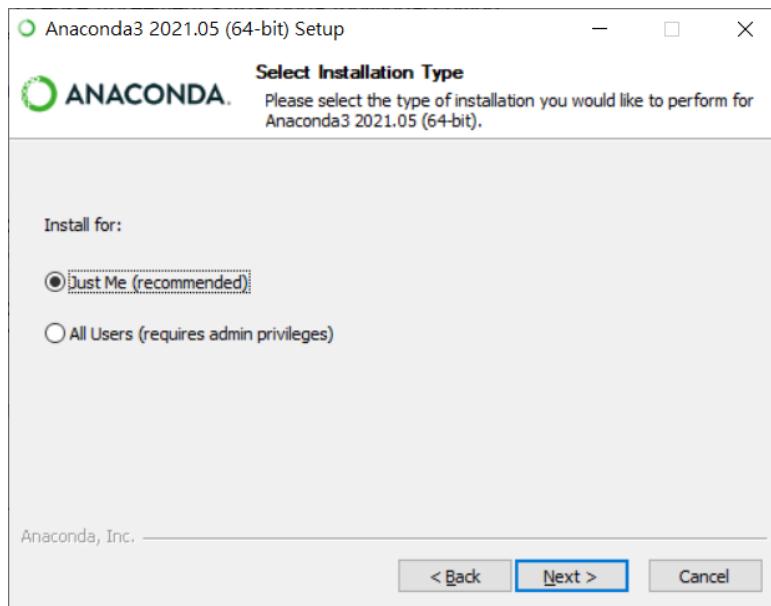
圖：下載後，快速點兩下進行安裝
有安全性警告，可按下執行或同意



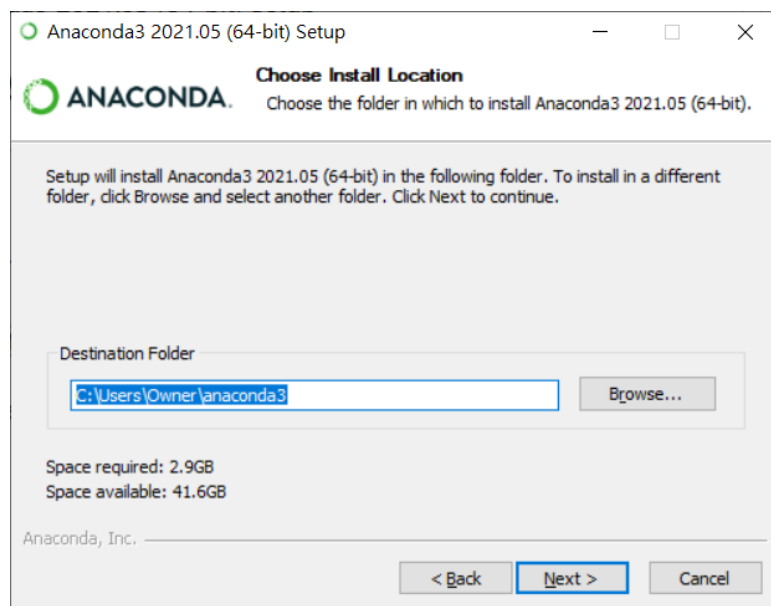
圖：按下「Next」



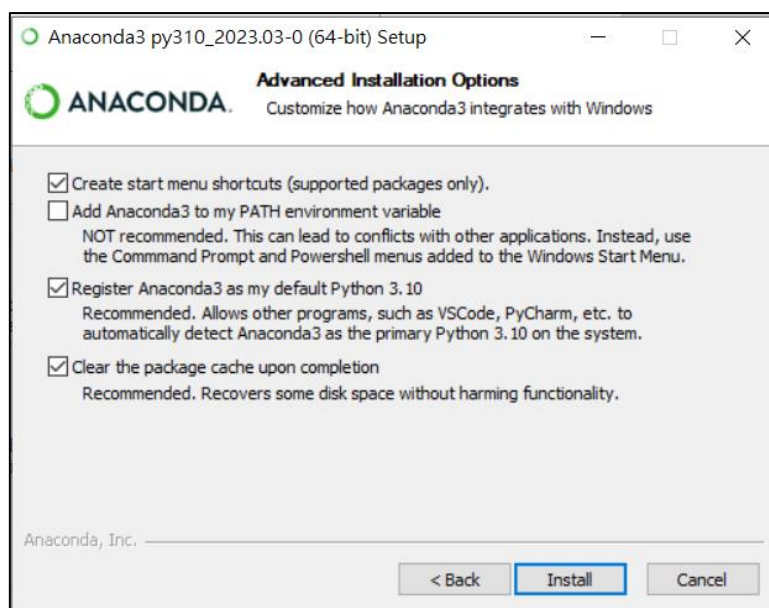
圖：按下「I Agree」



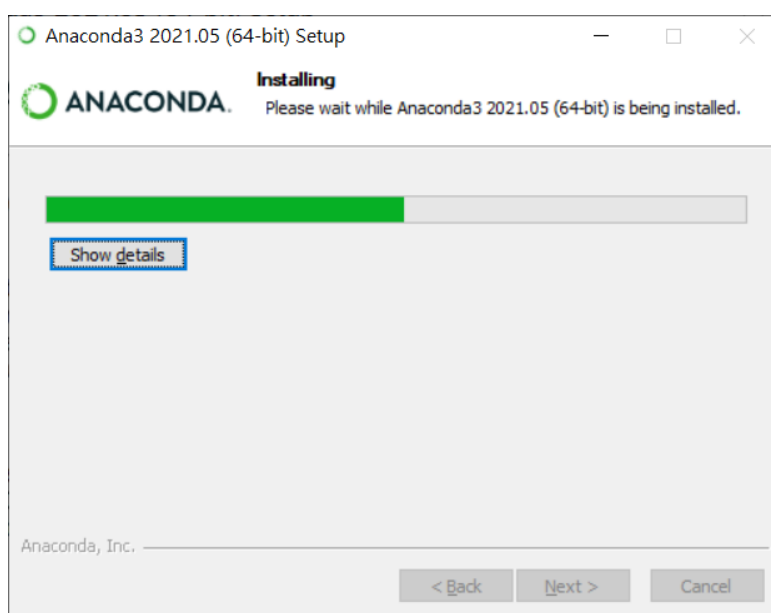
圖：依需求選擇後，按下「Next」



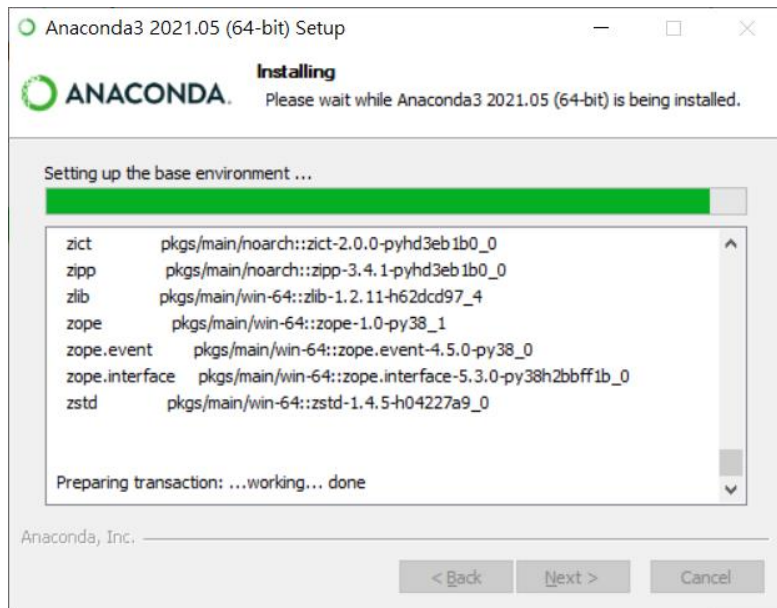
圖：安裝位置會在使用者資料夾中的 anaconda3，依需求設定，按下「Next」



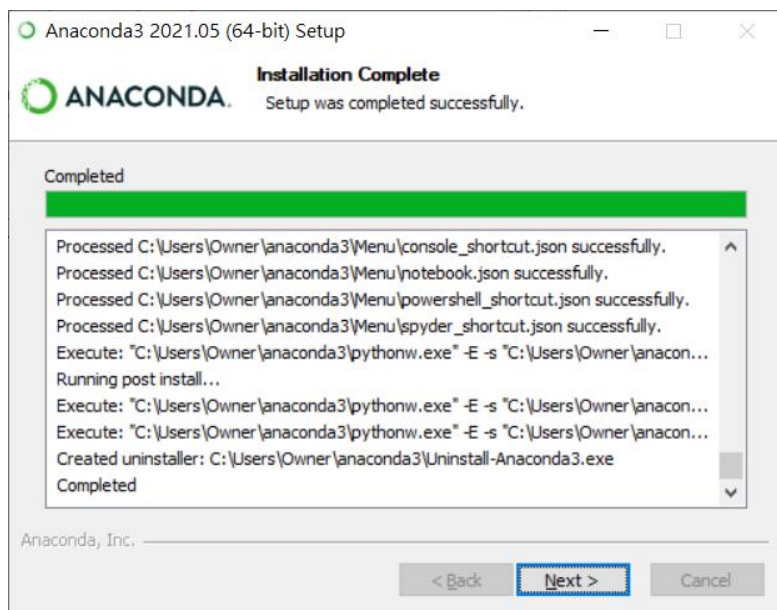
圖：依需求選擇（最後一個建議打勾，省空間），按下「Install」，進行安裝



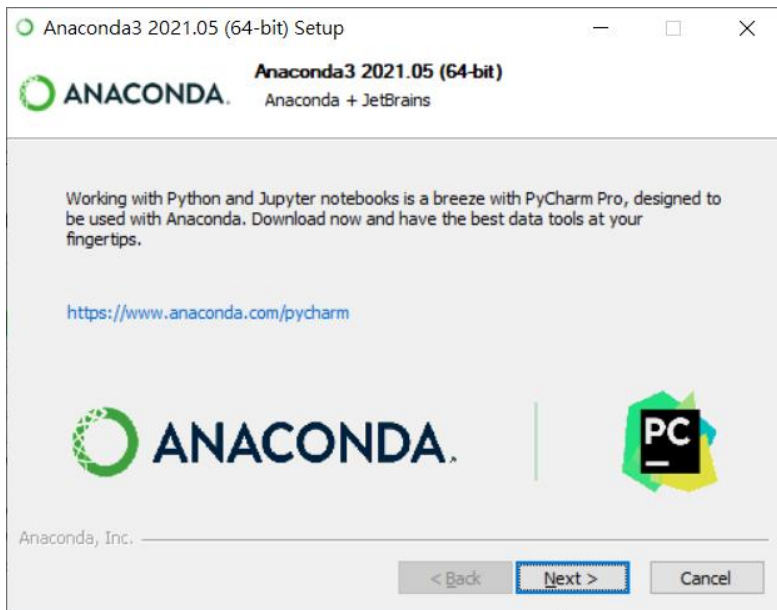
圖：安裝過程，需要一段時間



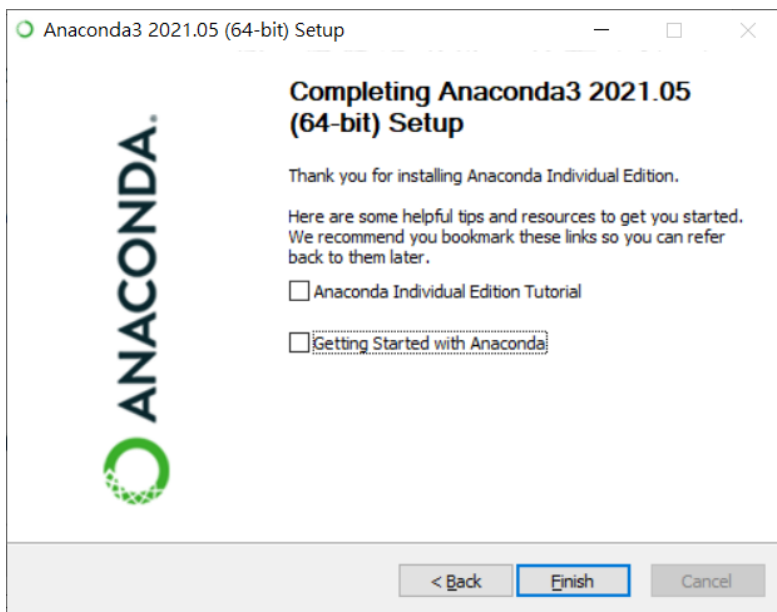
圖：按下「Show details」，會看到安裝過程



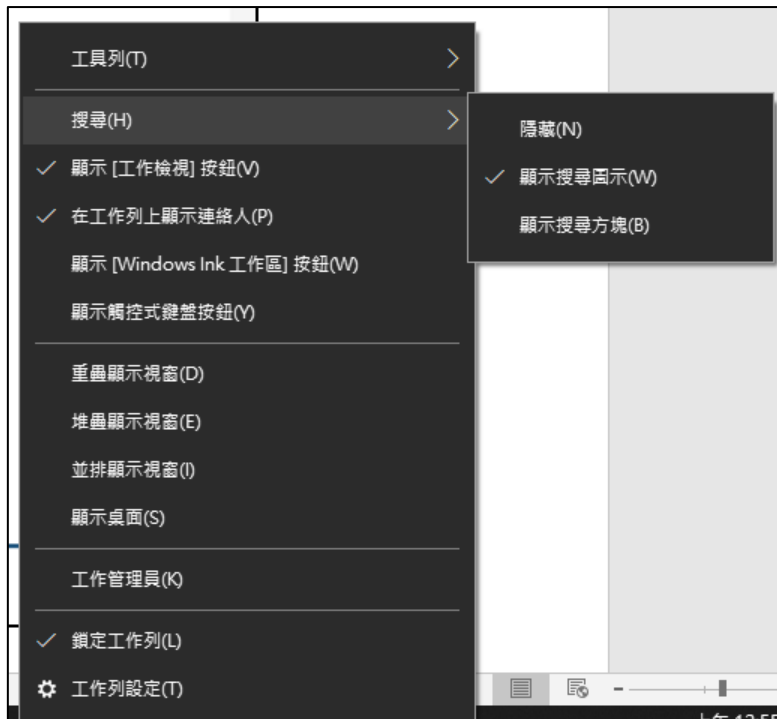
圖：安裝完成後，按下「Next」



圖：按下「Next」



圖：取消勾選圖片中的兩個選項後，按下「Finish」



圖：顯示搜尋圖示



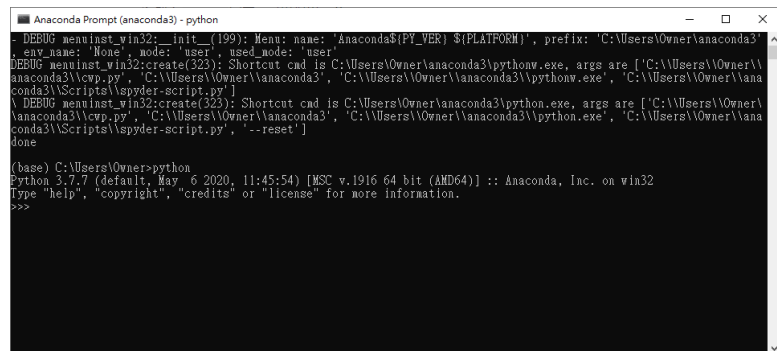
圖：搜尋圖示類似放大鏡，按下搜尋圖示



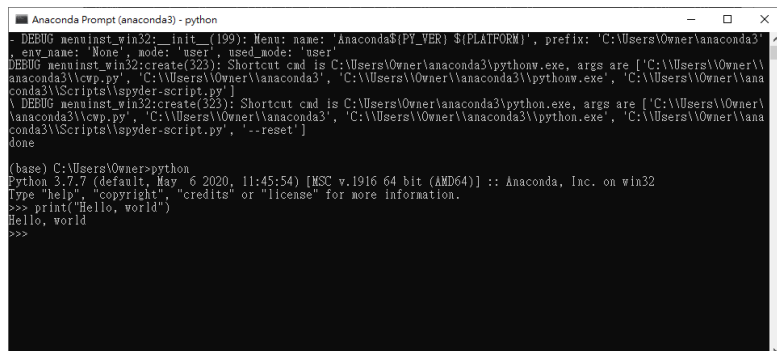
圖：搜尋「anaconda prompt」，按下「Anaconda Prompt (anaconda3)」



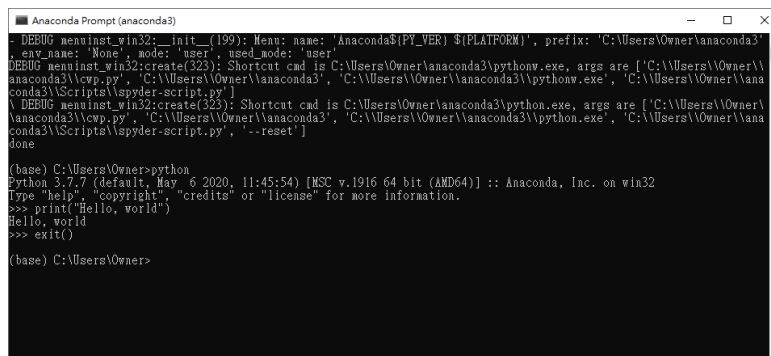
圖：出現 Anaconda Prompt，類似 Windows 的命令提示字元



圖：輸入「python」，進入 python 執行環境



圖：輸入「print("Hello, world")」，輸出「Hello, world」，python 安裝成功



```
DEBUG menulist_win32: __init__(199): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\Owner\anaconda3\...
env_name: 'None', mode: 'user', used_mode: 'user'
DEBUG menulist_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\pythonw.exe, args are ['C:\Users\Owner\anaconda3\Scripts\ipykernel', 'C:\Users\Owner\anaconda3\pythonw.exe', 'C:\Users\Owner\anaconda3\Scripts\ipykernel-script.py']
DEBUG menulist_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\pythonw.exe, args are ['C:\Users\Owner\anaconda3\Scripts\ipykernel', 'C:\Users\Owner\anaconda3\pythonw.exe', 'C:\Users\Owner\anaconda3\Scripts\ipykernel-script.py', '--reset']
done
(base) C:\Users\Owner>python
Python 3.7.7 (default, May 6 2020, 11:45:54) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, world")
Hello, world
>>> exit()
(base) C:\Users\Owner>
```

圖：按下「exit()」回到指令輸入的環境

安裝、切換與刪除 Conda 環境 (Environment)

預設是 (base)，如果有切換環境的需求，例如手上處理著不同 Python 版本的專案，需要不時切換版本來開發，此時可以建立一到多個 Conda 環境，需要的時候可以切換，不需要的時候可以刪除。

在執行以下的指令前，需要確認目前是否在 Anaconda Prompt 當中，或是可以直接使用 Conda 的 Terminal 環境 (例如 Mac OS)。終端機顯示預設路徑時，最前面會有 (base)，代表目前正在預設的 Conda 環境當中。

注意：如果不希望再安裝新的 conda env，可以使用既有的 env 或是 kernel，直接安裝課程所需套件即可。

安裝 conda 環境

```
conda create --name web_scraping python=3.10 notebook
ipykernel
```

進入 conda 環境

```
conda activate web_scraping
```

新增 Kernel

```
python -m ipykernel install --user --name ws --display-name
"Python3@ws"
```

註：

1. python -m 指的是直接使用模組 (module) 的預設功能。
2. --user 代表將 kernel 安裝在個人使用者目錄 (或是個人的家目錄) 當中，而非預設的系統環境。
3. --name ws 與 conda env 的 web_scraping 沒有直接關係，僅是 kernel 在系統當中的別名。

4. `--display-name "Python3@ws"` 是新增 Jupyter Notebook 時的選項。

檢視 Jupyter Notebook kernel

<code>jupyter kernelspec list</code>

刪除 Kernel

<code>jupyter kernelspec uninstall ws</code>
--

注意：

- Jupyter Notebook 建立的 kernel，跟 conda 的 env 概念是不一樣的。
 - Kernel 會啟動網頁版本的線上編輯器，並且使用目前 conda env 的 Python 版本；而 conda 的 env 單純是環境和套件的管理器。
 - 例如在 python=3.10 的 conda env 裡面安裝 kernel，此時該 kernel 用的 python 版本就會是 3.10。
 - 無論在任何 conda env 下，只要啟動 Jupyter Notebook，就會以當下目錄作為主目錄/工作路徑。

執行 Jupyter Notebook (在虛擬環境 web_scraping 下)
--

<code>jupyter notebook</code>

刪除 Conda 環境

<code>conda remove -n web_scraping --all</code>

安裝課程所需套件

可以考慮先切換到自訂的 Conda 環境，例如 web_scraping，再透過 pip 安裝套件。

說明

本課程需要安裝的套件指令


<code>pip install requests beautifulsoup4 lxml selenium wget</code>

下載 Chrome Web Driver

1. 請先下載 ChromeDriver

下載連結：<https://googlechromelabs.github.io/chrome-for-testing/>

網站首頁：<https://chromedriver.chromium.org/>

Chrome for Testing availability 

This page lists the latest available cross-platform Chrome for Testing versions and assets per Chrome release channel.

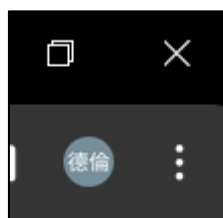
Consult [our JSON API endpoints](#) if you're looking to build automated scripts based on Chrome for Testing release data.

Last updated @ 2023-12-12T03:09:11.714Z

Channel	Version	Revision	Status
Stable	120.0.6099.71	r1217362	✓
Beta	121.0.6167.8	r1233107	✓
Dev	122.0.6170.5	r1234130	✓
Canary	122.0.6179.2	r1235708	✓

圖：下載 (Web)Driver 的頁面

2. 請確認目前你電腦裡面的 chrome 瀏覽器版本



圖：按下瀏覽器右上方的「⋮」



圖：說明→關於 Google Chrome



圖：請下載相同版本號碼的 ChromeDriver

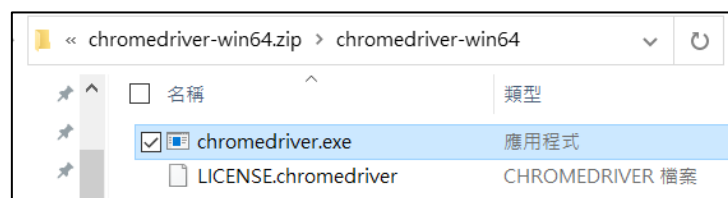
3. 下載 ChromeDriver 檔案，並放到專案資料夾當中

Stable			
Version: 120.0.6099.71 (r1217362)			
Binary	Platform	URL	HTTP status
chrome	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/linux64/chrome-linux64.zip	200
chrome	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/mac-arm64/chrome-mac-arm64.zip	200
chrome	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/mac-x64/chrome-mac-x64.zip	200
chrome	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/win32/chrome-win32.zip	200
chrome	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/win64/chrome-win64.zip	200
chromedriver	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/linux64/chromedriver-linux64.zip	200
chromedriver	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/mac-arm64/chromedriver-mac-arm64.zip	200
chromedriver	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/mac-x64/chromedriver-mac-x64.zip	200
chromedriver	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/win32/chromedriver-win32.zip	200
chromedriver	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/win64/chromedriver-win64.zip	200
chrome-headless-shell	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/linux64/chrome-headless-shell-linux64.zip	200
chrome-headless-shell	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/mac-arm64/chrome-headless-shell-mac-arm64.zip	200
chrome-headless-shell	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/mac-x64/chrome-headless-shell-mac-x64.zip	200

圖：手動複製連結，下載合適的 Chrome 版本

chromedriver	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/linux64/chromedriver-linux64.zip
chromedriver	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/mac-arm64/chromedriver-mac-arm64.zip
chromedriver	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/mac-x64/chromedriver-mac-x64.zip
chromedriver	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/win32/chromedriver-win32.zip
chromedriver	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/win64/chromedriver-win64.zip

圖：Windows 選擇 win64；MacOS 選擇對應版本；Linux 選擇 linux64



圖：以 win64 為例，請將 chromedriver.exe 複製到教材專案目錄底下即可

爬蟲專案開發實務分享

1. 個人 YouTube 頻道（不定時更新）

<https://www.youtube.com/@darreninfo-boatman>

2. 「觀察」是爬蟲工作者的重要技能，HTML & CSS selector 的概念要非常熟練；有些人習慣使用 XPATH，由於課程時間安排的關係，不會提到。

3. 被對方的伺服器擋住，無法繼續爬取，相換 IP，可以透過

- **Amazon Web Service 的 EC2（虛擬主機）**，透過 Running、Stopped 等過程，提供自動換新的 IP 給我們。
- **連接 VPN**，例如 Surfshark，透過切換其它網路環境（俗稱翻牆）來進

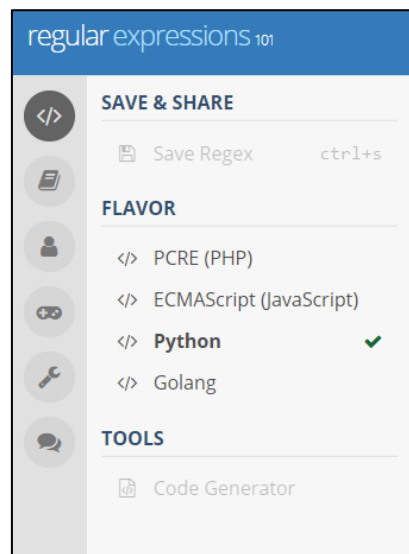
行；

- 若是人在咖啡廳，臨時被擋，需要換 IP，可以切換到**手機網路（手機當作無線基地台）**，**開啟飛航模式**，過個幾秒（例如 10 秒後）再閉關飛航模式，此時網路服務供應商便會提供新的 IP 給我們，便可繼續爬取資料，有時候可以搭配 PC 平台的手機螢幕控制工具，結合課堂上提到的 PyAutoGUI，透過網路請求的品質，來決定是否切換；
 - 可以整合免費的 **Proxy Pool** 來取得臨時可用的 IP，持續對網頁進行請求，如果更付費的話，品質更好。
4. 建議：爬取資料時，每經過一個階段（可能是網站換頁前後、網頁動態生成資料之間），各給一個「隨機」的 **sleep 時間**，例如 1 到 3 秒、0.1 到 0.5 秒，或是以自身經驗，在攻防之中，取得平衡，設定一組比較不會被擋的隨機數。

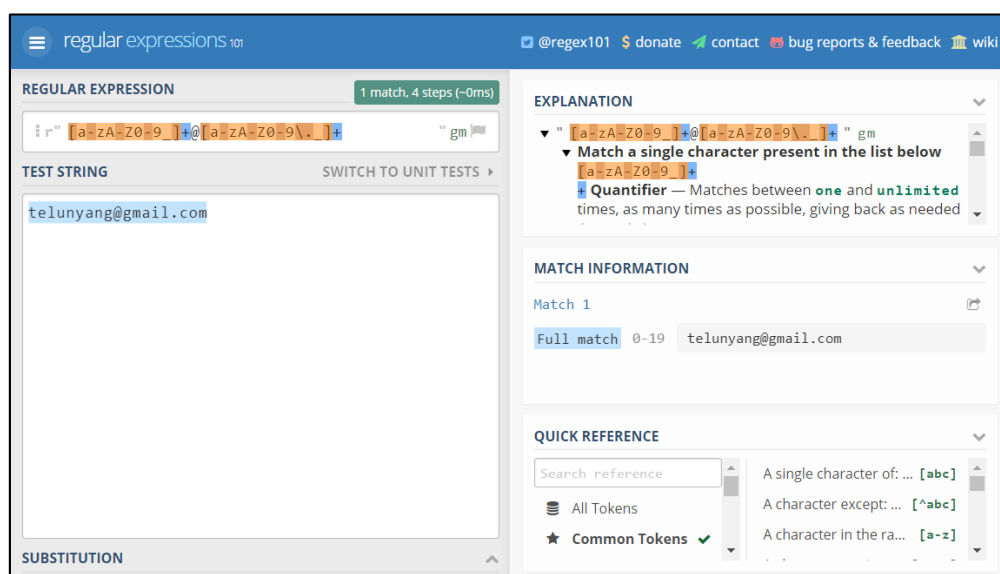
Module 2. 正規表達式 (Regular Expression)

說明

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



圖：選擇 FLAVOR 為 Python



圖：使用正規表達式，來判斷字串是否符合文字格式或條件

下面表格為快速參考的範例：

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

說明	正規表達式	範例
(這位置右邊要出現什麼)		
正向環視否定 (這位置右邊不能出現什麼)	Johnnie (?!Cena)	Johnnie Walker
反向環視 (這位置左邊要出現什麼)	(?<=Johnnie) Walker	Johnnie Walker
反向環視否定 (這位置左邊不能出現什麼)	(?<!=John) Walker	Johnnie Walker

常用方法
<pre># 匯入 regex 套件 import re</pre>
<pre># match ...</pre> <p>說明</p> <p><code>re.match</code> 會從字串的「開頭」開始比對， 比對不到，則回傳 <code>None</code></p> <pre>...</pre> <pre>regex01 = r'2[0-9]{3}\V[0-1]?[0-9]{1}\V([0-3]?[0-9])'</pre> <pre>string01 = "2024/09/18"</pre> <pre>match01 = re.match(regex01, string01)</pre> <pre>print(match01)</pre> <pre>print(match01[0])</pre> <pre>print(match01[1])</pre> <pre>...</pre> <p>補充：</p> <p><code>match.group()</code> 或 <code>match.group(0)</code> 是 regex 所代表的整個完整比對的字串， <code>match.group(1)</code> 是第一組()中的內容， <code>match.group(2)</code> 是第二組()中的內容...</p> <pre>...</pre> <pre>print(match01.group(0))</pre> <pre>print(match01.group(1))</pre>
<pre># findall ...</pre> <p>說明</p> <p><code>re.findall</code> 會將所有配對到的字串</p>

```

回傳成一個 list
'''

regex02 = r'[0-9]+'
string02 = "0911111111, 0922222222, 0933333333"
listMatch02 = re.findall(regex02, string02)
print(listMatch02)
print(listMatch02[0])
print(listMatch02[2])

```

```

# finditer
'''

說明
re.finditer 會將所有配對到的字串
以迭代的方式呈現，若沒有配對到，則回傳 None
'''

regex03 = r'[0-9]+'
string03 = "0911111111, 0922222222, 0933333333"
iterableMatch03 = re.finditer(regex03, string03)
if iterableMatch03 != None:
    for match in iterableMatch03:
        print(match[0])

```

```

# search
'''

說明
re.search 會將整個字串進行搜尋，
但只會比對到第一組，
比對不到，則回傳 None
'''

regex04 = r'[a-zA-Z]([12])\d{8}'
string04 = "A123456789, S299888777"
match04 = re.search(regex04, string04)
print(match04)
print(match04[0])
print(match04[1])

```

```

# split
'''

說明
re.split 類似 string.split('separator')，

```

只是用正規表達式來作為 separator，

並回傳 list

...

```
regex05 = r'\d'
```

```
string05 = "One1Two2Three3Four4"
```

```
listMatch05 = re.split(regex05, string05)
```

```
print(listMatch05)
```

```
# sub
```

...

說明

```
re.sub(regex, replace_string, test_string)
```

將 regex 所代表的文字，改成 replace_string，文字來源是 test_string

...

```
regex06 = r"\D"
```

```
string06 = "5-20 #1314"
```

```
strResult = re.sub(regex06, "", string06)
```

```
print(strResult)
```

環視

名稱	語法	說明
正向環視	(?=)	這位置右邊要出現什麼
正向環視否定	(?!)	這位置右邊不能出現什麼
反向環視	(?<=)	這位置左邊要出現什麼
反向環視否定	(?<!)	這位置左邊不能出現什麼

```
# 環視 (例如去除中文字旁邊的空白)
```

```
regex07 = r"\s(?![a-zA-Z])" # 也可以寫成 r"(?![a-zA-Z])\s"
```

```
string07 = "一天一蘋果醫生遠離我。An apple a day keeps the  
doctor away."
```

```
strResult = re.sub(regex07, '', string07)
```

```
print(strResult)
```

```
# 環視 (加入千分位)
```

```
regex08 = r'(?<=\d)(?=(\d{3})+\b)'
```

```
string08 = '1234567890'
```

```
strResult = re.sub(regex08, ',', string08)
print(strResult)
```

具名群組

```
'''
補充：
除了 .group(n) 以外，
還可以用 key 來代替 n。
'''

# 身分證字號
regex09 = r'[A-Z](?P<gender>[12])\d{8}'
string09 = "A100000001"
match09 = re.match(regex09, string09)

# 完整配對的文字
print(match09[0])
print(match09.group(0))
print(match09.group())

# 具名(類似 key)所代表的值，也可以用索引代號來取得
print(match09.group('gender'))
print(match09['gender'])
print(match09[1])
```

參考資料

1. Python3 正则表达式
<https://www.runoob.com/python3/python3-reg-expressions.html>
2. 正则表达式-全型英數中文字、常用符號 unicode 對照表
<https://blog.typeart.cc/正则表达式-全型英數中文字、常用符號 unicode 對照表/>
3. 匹配中文字符的正則表達式： [/u4e00-/u9fa5]
<https://www.itread01.com/content/1513168876.html>
4. 【Regular Expression】正向環視、反向環視
<https://toyo0103.blogspot.com/2017/01/regular-expression.html>

Module 3. Chrome Developer Tool

各頁籤常用功能簡介 (Elements / Console / Network / ...)

Chrome 開發者工具是內建於 Google Chrome 中的 Web 開發和測試工具

網址：<https://developers.google.com/web/tools/chrome-devtools?hl=zh-tw>



圖：Chrome 開發者工具的說明網頁

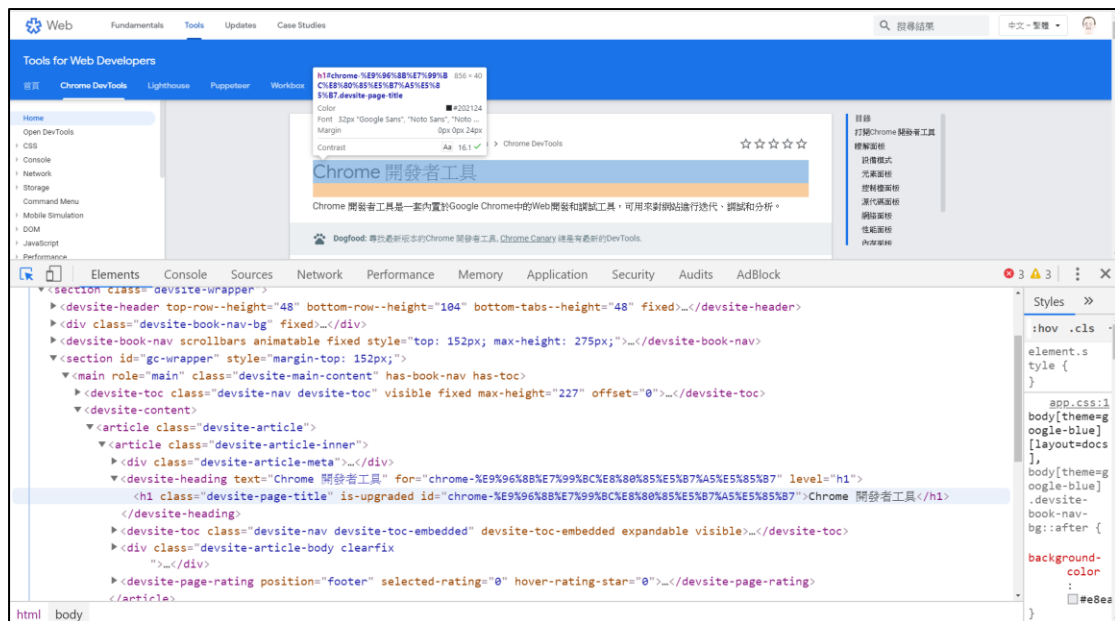
開啟開發工具(dock)

- F12

Elements 面板

檢查 HTML 元素

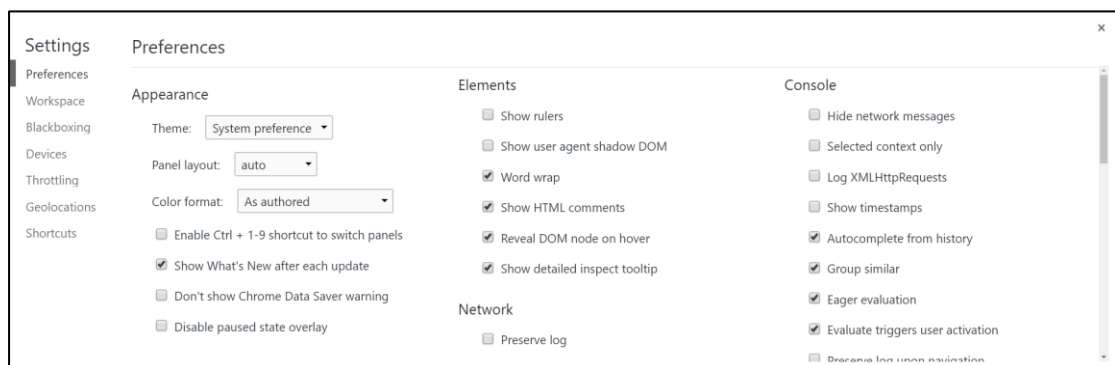
- Ctrl + Shift + C (追蹤滑鼠移過網頁元素所在位置的狀態)
- 網頁內容任意處按滑鼠右鍵→檢查



圖：檢查元素

補充說明

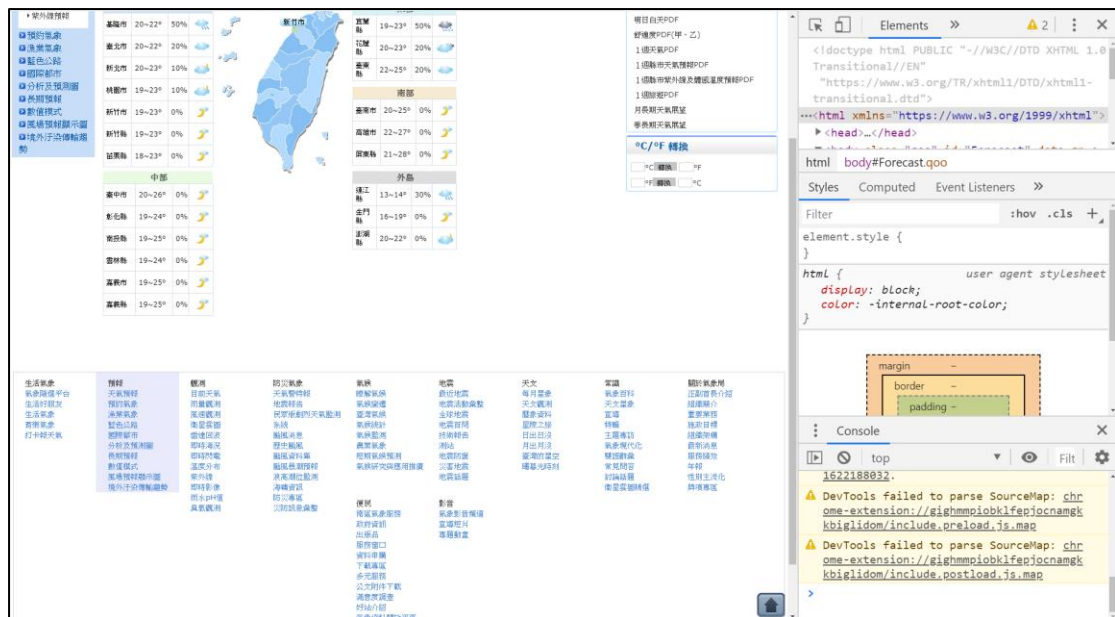
開啟 Chrome 開發者工具以後，按下 F1，可以看到一些偏好設定，方便我們設定開發工具，例如顯示外觀、模擬裝置、自訂地理位置、快捷鍵等。



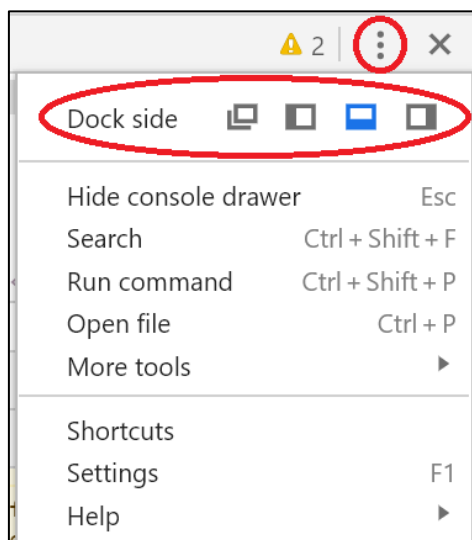
圖：Chrome 開發者工具偏好設定

開啟開發工具後，常用快捷鍵：

- Ctrl + Shift + D 切換檢查元素的 dock side

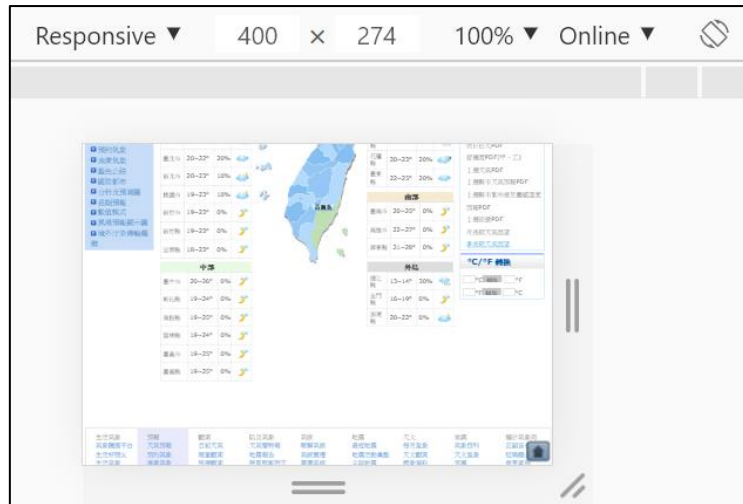


圖：切換 dock side，從下方到右側

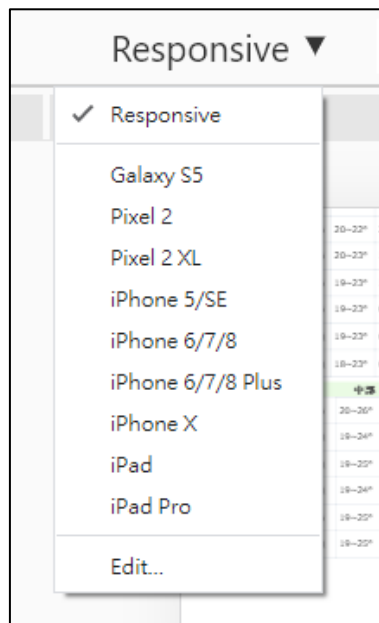


圖：按下三個點的圖示，也可以選擇 dock side

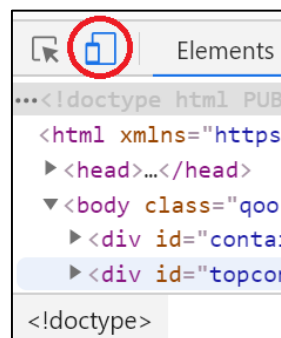
- Ctrl + Shift + M 開啟模擬裝置模式(切換裝置工具欄)



圖：可選擇不用的行動裝置，或自訂寬高，來顯示網頁



圖：選擇裝置來觀看網頁

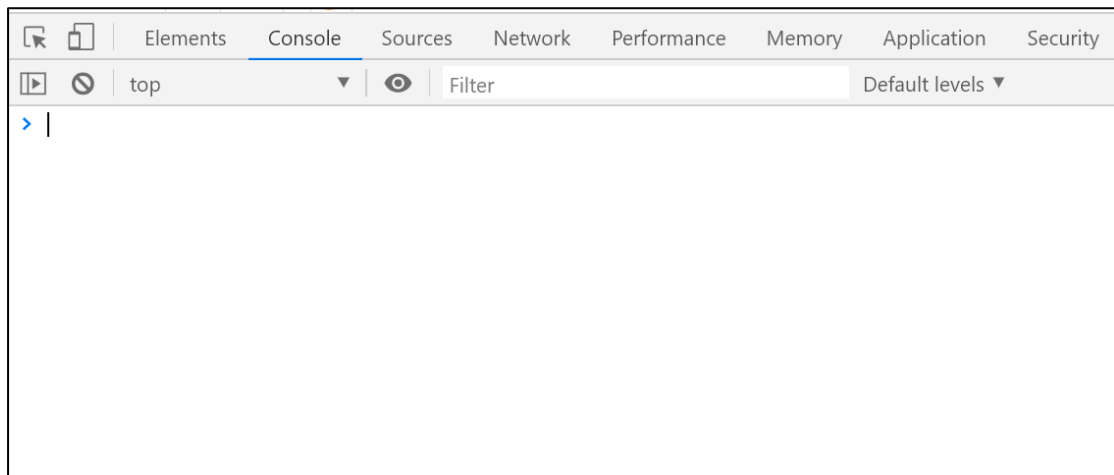


圖：等同按下切換裝置工具欄

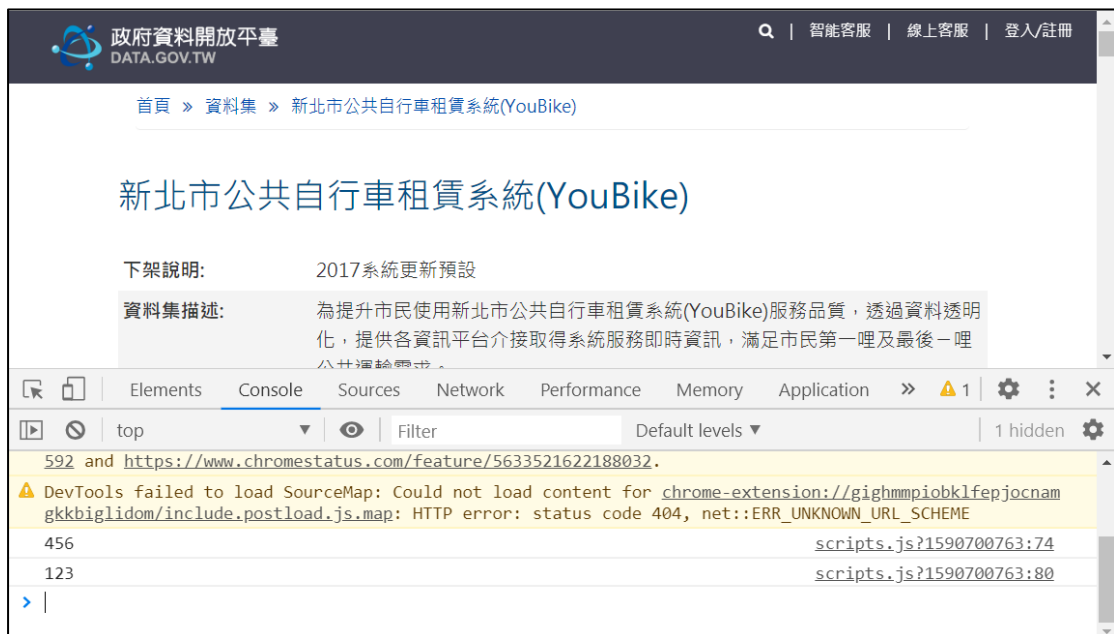
- Ctrl + O 尋找 HTML 當中的檔名
- Ctrl + R 或 F5 刷新頁面
- Ctrl + F5 清除快取後，刷新頁面(重新從伺服器端請求下載 HTML)
- Ctrl + L 清除 Console
- Shift + Enter 在 Console 中斷行(或多行)

Console 面板

我們可以使用 Console 面板，了解目前網頁執行的狀況。



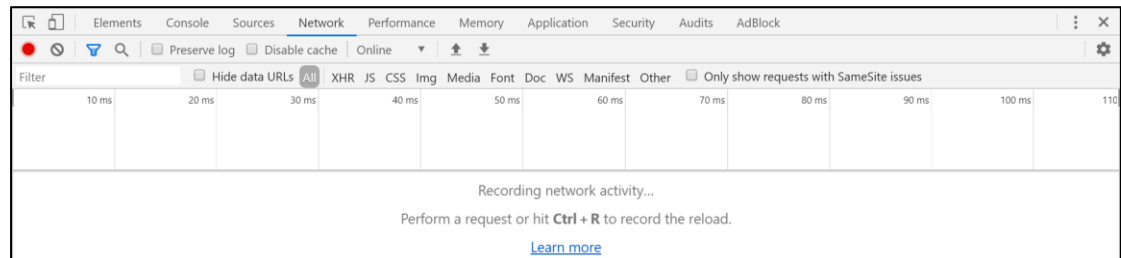
圖：Console 面板



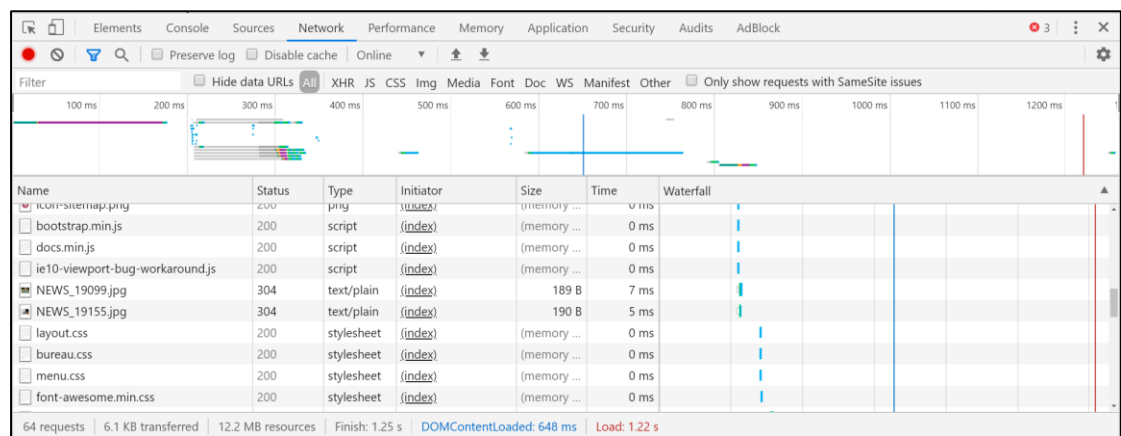
圖：可以看到目前網站的情況。

Network 面板

Network 面板會顯示出所有網路請求的詳細訊息記錄，包括狀態、資源類型、大小、所需時間、HTTP request header 和 response header 等等，明確找出哪些請求比預期還要耗時，並加以調整，是優化網頁的重要工具。



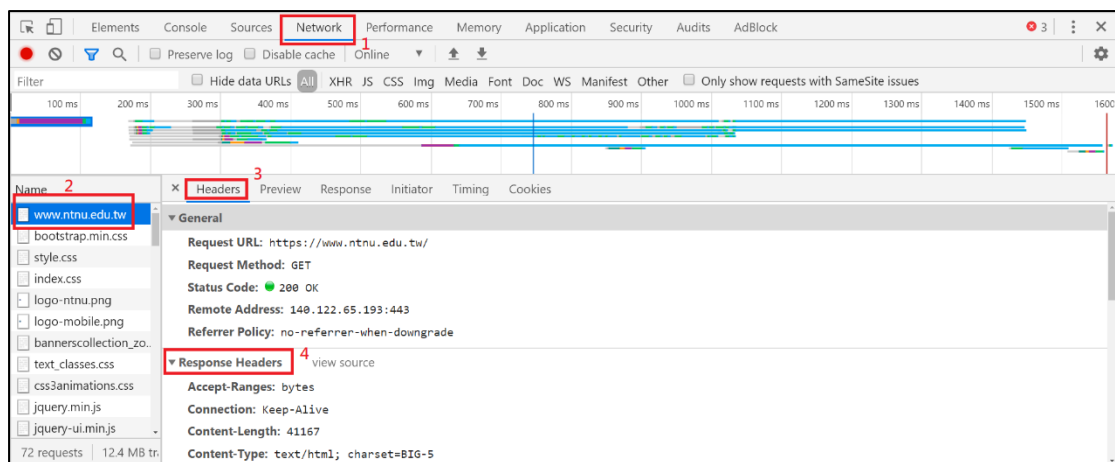
(圖) Network 面板會記錄任何的網路活動



(圖) 記錄網頁讀取的資訊與下載順序

我們可以透過 Headers，來了解網頁請求的狀況。開啟 Headers 的流程為：

1. 開啟 Network 面板
2. Ctrl + R 或是 F5 刷新頁面
3. 點選左側的檔案名稱
4. 點選 Headers

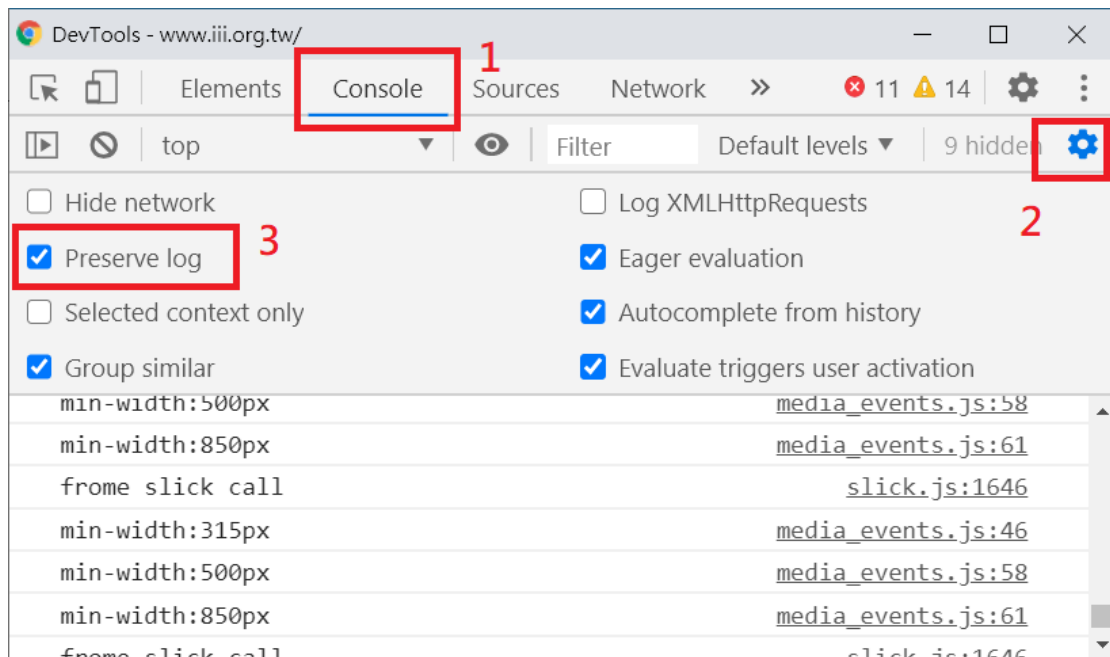


圖：觀看檔案的 Headers 內容

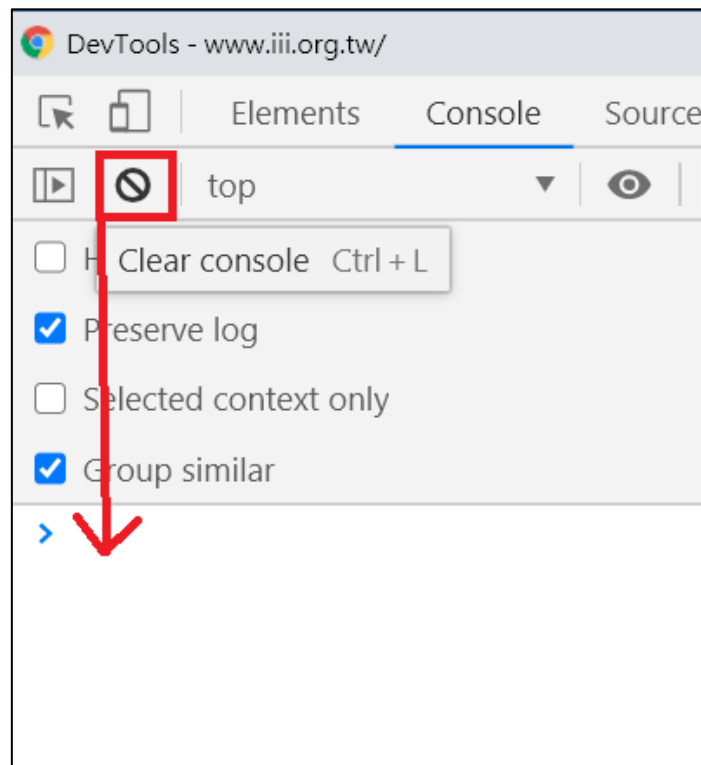
Request Headers (請求標頭，參考[維基百科](#))

標頭欄位	說明	範例
Cookie	之前由伺服器通過 Set-Cookie 傳送的一個 超文字傳輸 協定 Cookie	Cookie: _ga=GA1.3.1322956465.1572335045; locale=zh_TW; _gid=GA1.3.1110994946.1584940974; _gat_gtag_UA_141775379_1=1
User-Agent	瀏覽器的瀏覽器身分標識字串	User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/21.0

常用操作流程介紹 (Preserve Log / Clear / ...)



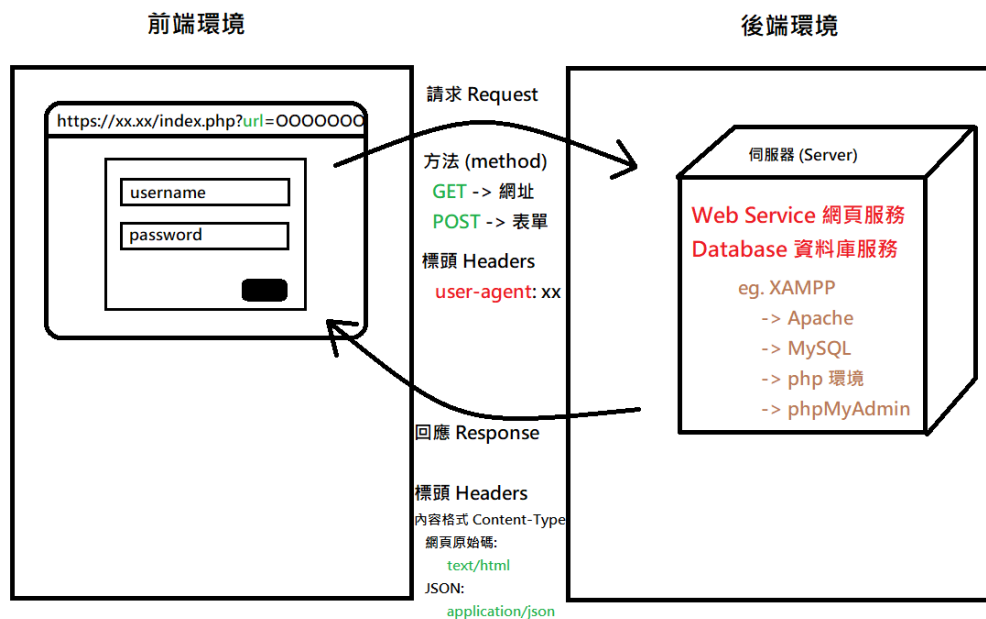
圖：勾選 Preserve log，縱然頁面刷新，過去的 log 依然會保存起來



圖：按下 Clear Console，或是 Ctrl+L，即可清除 log

Module 4. 請求 (Request)

請求 (Request)



圖：請求(request)與回應(response)

GET 方法的 Query String

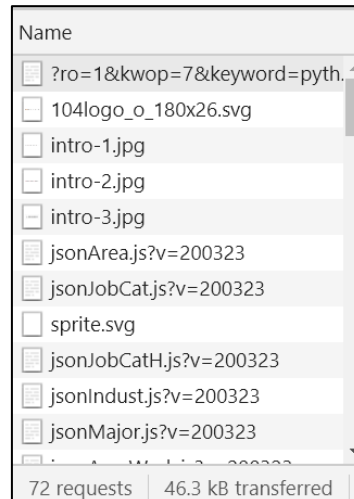
表單資料將以字串方式附加在網址 (URI) 的後面傳送，在網址尾端，會以「？」符號，開啟跟著表單中的資料，每個欄位間的值，以「&」連接起來。

一般來說，GET 參數 (Query String) 的格式如下：

`https://www.104.com.tw/jobs/search/?ro=1&kwop=7&keyword=python&order=13&asc=0&page=1&mode=s&jobsource=2018indexpoc`

key (鍵)	value (值)
ro	1
kwop	7
keyword	python
order	13
asc	0
page	1
mode	s
jobsource	2018indexpoc

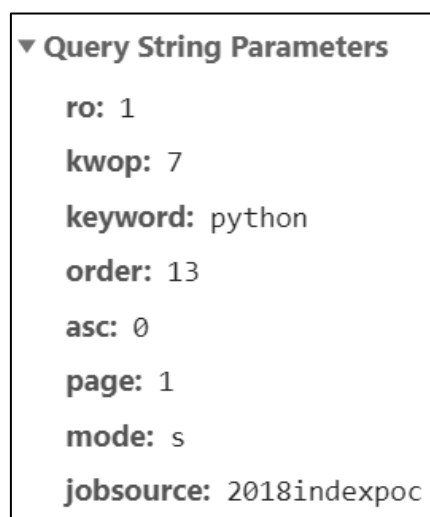
我們將前面的網址放到 chrome 瀏覽器的網址列中，讀取結束後，按下 Ctrl + Shift + i，再按下 Network，然後 Ctrl + R，重新讀取網址，再從左側的 Name 欄位裡，選擇最上面（通常最先被讀取的那個）的項目，再選 Headers，會看到以下的資訊：



圖：選擇第一個項目



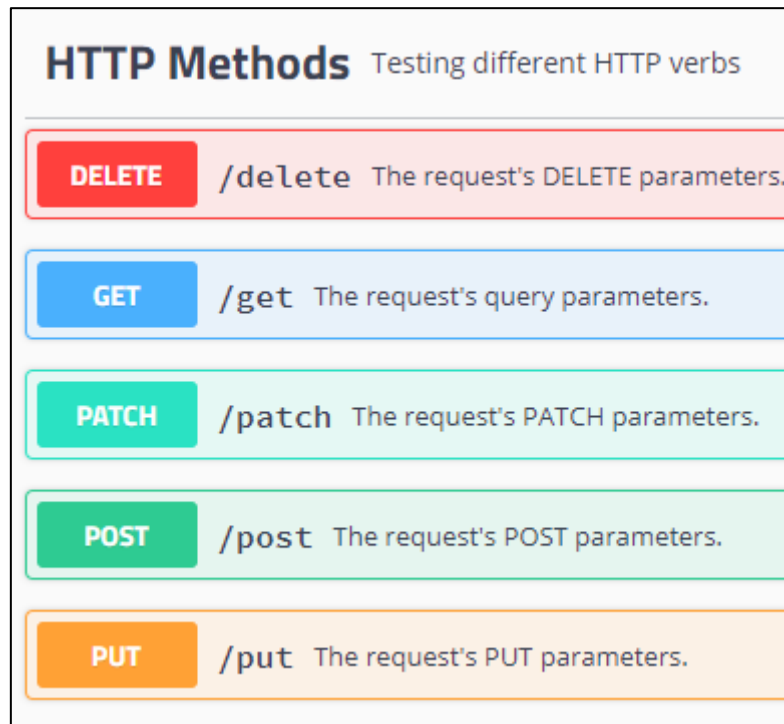
圖：看到 Headers 裡面的 General，明確指出 Request Method 是 GET



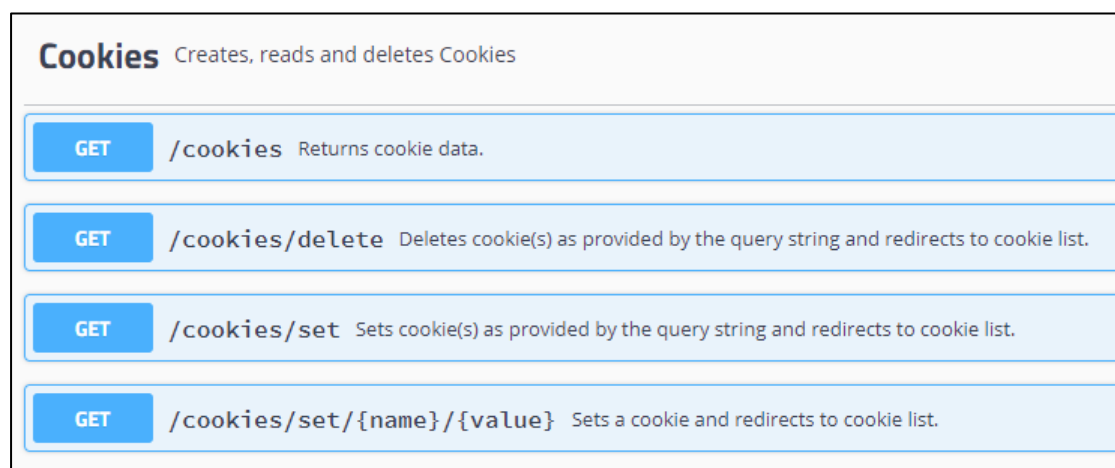
圖：移到最下面，可以看到 Query String 的參數

Module 5. 套件 requests

<https://httpbin.org/> 是一個專門拿來測試 HTTP Request 的網路服務，只要依照文件發動 HTTP Request 到指定的路徑，就會將它收到的內容以 JSON 格式回傳，在測試 API 行為時非常好用。



圖：支援的 HTTP 方法



圖：增刪修 cookies 的操作方法

requests 套件

使用 requests 工具

import requests

```
# 使用 json 工具
```

```
import json
```

```
# 使用 GET 方式下載普通網頁
```

```
res = requests.get('https://httpbin.org/get')
```

```
# 伺服器回應的狀態碼
```

```
# 參考網頁: https://reurl.cc/2DRpan
```

```
print(res.status_code)
```

```
# 回傳資料的編碼
```

```
print(res.encoding)
```

```
# 指定回傳資料的編碼
```

```
# response.encoding = 'utf-8'
```

```
# 輸出網頁 HTML 原始碼
```

```
print(res.text)
```

```
# GET 方法的 query string
```

```
my_params = {  
    'key1': 'value1',  
    'key2': 'value2'  
}
```

```
# 將 query string 加入 GET 請求中
```

```
res = requests.get('https://httpbin.org/get', params = my_params)
```

```
# 觀察 URL
```

```
print(res.url)
```

```
# 輸出網頁 HTML 原始碼
```

```
print(res.text)
```

```
# POST 方法的 form data
```

```
my_data = {  
    'key1': 'value1',  
    'key2': 'value2'  
}
```



```
# 將 form data 加入 POST 請求中
res = requests.post('https://httpbin.org/post', data = my_data)

# 輸出網頁 HTML 原始碼
print(res.text)

# 要上傳的檔案 (變數名稱為 my_filename)
my_files = {
    'my_filename': open('turingcerts.jpg', 'rb')
}

# 將檔案加入 POST 請求中
res = requests.post('https://httpbin.org/post', files = my_files)

# 輸出網頁 HTML 原始碼
print(res.text)

# 自訂標頭
my_headers = {
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.82
Safari/537.36'
}

# 將自訂標頭加入 GET 請求中
res = requests.get('https://httpbin.org/get', headers = my_headers)

# 輸出網頁 HTML 原始碼
print(res.text)

# 自訂 cookie 格式
my_cookies = {
    "first_cookie": "hello",
    "second_cookie": "world"
}

# 將 cookie 加入 GET 請求
res = requests.get('https://httpbin.org/get', cookies = my_cookies)
```

```

# 輸出網頁 HTML 原始碼
print(res.text)

'''
資訊平台: https://greenlifestyle.moenv.gov.tw/
綠色餐廳: https://greenlifestyle.moenv.gov.tw/categories/restaurant
預設連結 https://greenliving.moenv.gov.tw/newPublic/APIs/Restaurant4
分頁連結 https://greenliving.moenv.gov.tw/newPublic/APIs/Restaurant4/1
'''

# 請求網址
url = 'https://greenliving.moenv.gov.tw/newPublic/APIs/Restaurant4/1'
res = requests.get(url)

# 將 json 轉成物件
obj = json.loads(res.text) # 或使用 obj = res.json()

# 輸出對應節點的文字
print(obj['Result'])
print(obj['RowCount'])
print(obj['PageIndex'])

print("=" * 50)

# 輸出部分節點的文字
for o in obj['Detail']:
    # 輸出資料
    print(f"Id: {o['Id']}")
    print(f"Name: {o['Name']}")
    print(f"Address: {o['Address']}")
    print(f"Latitude: {o['Latitude']}")
    print(f"Longitude: {o['Longitude']}")
    print(f"ImgByte: {o['ImgByte']}")
    print("=" * 50)

```

Module 6. 淺談 HTML 與 CSS Selector (選擇器)

HTML

HTML (超文字標示語言, HyperText Markup Language) 是一種標記語言 (Markup Language), 而不是一般熟知的程式語言。一般被稱為 HTML Tags (標籤), 在一些程式語言 (例如 JavaScript) 而言, 它們是 HTML Elements (元素)。每一個 Tag 由元素 (Element) 和屬性 (Attribute) 組合, 用來決定網頁元件的呈現樣貌, 例如:

一般文字
我每天都被自己帥醒, 壓力好大

如果希望讓文字在網頁上變成一個「段落」(Paragraph), 就在文字前後各自加上「<p>」和「</p>」:

HTML
<p>我每天都被自己帥醒, 壓力好大</p>

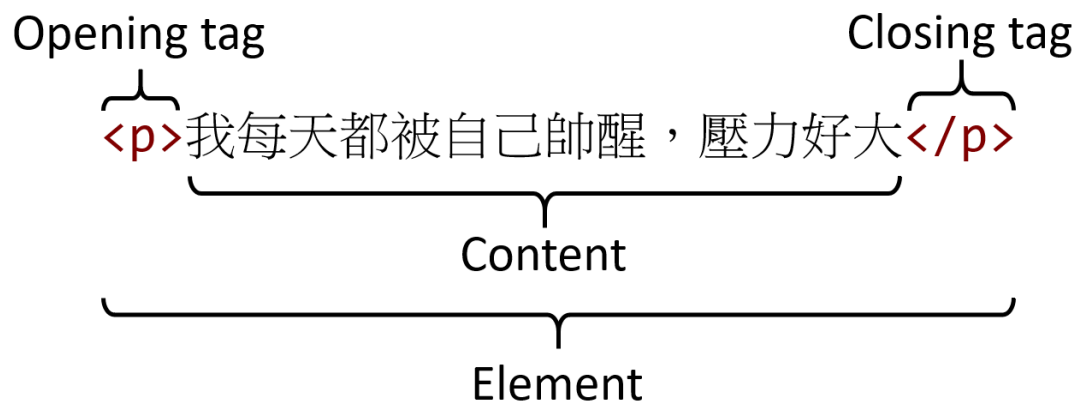
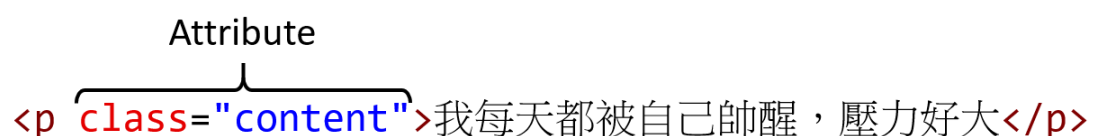


圖: HTML 基本元素的組成

每一個元素都可以設定自己的屬性 (Attribute), 屬性設定可以 0 到多組:

HTML
<p class="content">我每天都被自己帥醒, 壓力好大</p>



圖：屬性可以為元素提供更多的資訊

有些元素沒有內容 (Content)，會稱之為空元素 (Empty Element)，例如

``:

HTML

```

```

註 1: src 圖片連結來自 <https://www.dora-world.com.tw/character.php>

註 2: `` 和其他元素都在一行上，相臨的行內元素會排列在同一行，直到一行排不下，才會換行，其寬度隨元素的內容而變化，這種元素稱為行內 (inline) 元素。

基本的網頁架構：

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>個人首頁</title>
  </head>
  <body>
    <p class="content">我每天都被自己帥醒，壓力好大</p>
    
  </body>
</html>
```

註：元素裡面可以有其它元素，我們稱之為「巢狀元素 (Nesting Element)」

我每天都被自己帥醒，壓力好大



圖：前述 HTML tags 在瀏覽器上顯示的結果

CSS Selector

基本的選擇器類型如下：

選擇器	說明	使用元素	選擇器
型別/元素選擇器 Type selectors	選擇所有符合指定標籤 (Tags) 的元素	<code><a>xxx</code> , <code><p>xxx</p></code> , <code><input /></code> , <code><div></div></code> <code><select>xxx</select></code> 等等	a p input div select
ID 選擇器 ID selectors	選擇指定 id 屬性值的元素。 (一個文件中，每個 ID 屬性都是唯一的。)	<code><input type="submit" id="btn" value="按我送出" /></code> , <code>按我看更多</code> 等等	#btn #more
類別選擇器 Class selectors	選擇所有符合指定 class 屬性值的元素	<code><div class="container mb-3"></code> <code><div class="row">主要內容</code> <code></div></code> <code></div></code>	.container .mb03 .container.mb-3 .container.mb-3 .row .row
屬性選擇器 Attribute selectors	選擇所有符合指定屬性的元素。	<code>下一頁</code>	[class="next-page"] [class=next-page] [href]

註 1. 選擇器語法可以混合使用，例如 `a[class=next-page]`、`div.row`

註 2. 巢狀元素的概念，選擇器也有，例如

「`div.container.mb-3 > div.row`」：只選子節點

「`div.container.mb-3 div.row`」：子子孫孫節點我全都要

參考資料

[1] HTML 基礎

https://developer.mozilla.org/zh-TW/docs/Learn/Getting_started_with_the_web/HTML_basics

[2] HTML Element Reference - By Category

https://www.w3schools.com/TAGS/ref_byfunc.asp

[3] HTML 元素参考

<https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element>

[4] CSS Selectors

https://developer.mozilla.org/zh-TW/docs/Web/CSS/CSS_selectors

[5] CSS Selector Reference

https://www.w3schools.com/cssref/css_selectors.php

[6] 區塊元素 行內元素 空元素特點？分別有哪些？

<https://medium.com/@small2883/區塊元素-行內元素-空元素特點分別有哪些-19f8c05f16f6>

Module 7. 套件 BeautifulSoup 4

套件介紹及常用功能

Beautiful Soup 是一個 HTML parser，將 Document 轉換成一個樹狀結構，提供簡單的函式來走訪、搜尋、修改分析此樹狀結構，支援 CSS 選擇器。

常用功能

我們主要用 BeautifulSoup 套件來作為網站解析的工具。

- find() 方法 (取得單一元素)
- find_all() 方法 (取得元素集合)
- select_one() 方法 (取得單一元素)
- select() 方法 (取得元素集合)

BeautifulSoup 基本用法

soup.select() :

回傳的結果是元素集合 (list 型態, BeautifulSoup ResultSet)

soup.select_one() :

回傳的結果是單一元素 (BeautifulSoup Result)

Beautifulsoup 套件

...

參考網頁

[1] Python 使用 BeautifulSoup 抓取與解析網頁資料，開發網路爬蟲教學
<https://blog.gtwang.org/programming/python-beautiful-soup-module-scrape-web-pages-tutorial/2/>

...

```
import requests as req
from bs4 import BeautifulSoup as bs

# PTT NBA 板
url = "https://www.ptt.cc/bbs/NBA/index.html"

# 用 requests 的 get 方法把網頁抓下來
res = req.get(url)
```

```

# 指定 lxml 作為解析器
soup = bs(res.text, "lxml")

# 第一個 <a></a>
print(soup.find("a"))

# 全部 <a></a>，此時回傳 list
print(soup.find_all("a"))

# 指定 list 某個元素的 html
print(soup.find_all("a")[2])

# 取得 id 為 logo 的元素
logo = soup.find(id = "logo")
print(logo)

# 取得所有 div，類別名稱為 r-ent，回傳為 list
posts = soup.find_all("div", class_ = "r-ent")
print(posts)

'''
以下透過 CSS selector 取得元素，
回傳格式為 list
'''

# 輸出 title
print(soup.select_one('title'))

# 輸出 a
print(soup.select('a'))

# 透過 class 名稱取得元素
print(soup.select("a.board"))

# 透過 id 名稱取得元素
print(soup.select_one("#logo"))

# 透過 attribute 取得元素
print(soup.select('a[class="board"]'))

# 取得單一節點的文字內容 (select_one 會回傳單一 bs element 物件，select 會回傳 list)
print(soup.select_one('title').get_text())

```



```
print(soup.select('a')[0].get_text())  
# 透過迭代取得所有 a 的文字內容  
for a in soup.select('a'):  
    print(a.get_text())  
# 透過迭代取得所有 a 的屬性 href  
for a in soup.select('a'):  
    if a.has_attr('href'):  
        print(a['href']) # a.get("href")  
    else:  
        print("=" * 50)  
        print(f"連結[{a.get_text()}] 沒有 href 屬性")  
        print("=" * 50)
```

Module 8. cookie 用於 requests

以 PTT Gossiiping (八卦版) 為例

```
import requests as req
from bs4 import BeautifulSoup as bs

# PTT Gossiiping (八卦版)
url = "https://www.ptt.cc/bbs/Gossiping/index.html"

# 首頁網址
prefix = 'https://www.ptt.cc'

# 設定 cookie
my_cookies = {
    "over18": "1"
}

# 用 requests 的 get 方法把網頁抓下來
res = req.get(url, cookies = my_cookies)

# 指定 lxml 作為解析器
soup = bs(res.text, "lxml")

# 顯示連結列表
for a in soup.select('div.r-ent > div.title > a'):
    print(a.get_text())
    print(prefix + a['href'])
```

Module 9. 案例：PTT_NBA_看板主頁與內頁

取得 PTT NBA 列表

匯入套件

```
from bs4 import BeautifulSoup as bs
import requests as req
from pprint import pprint
```

取得新聞列表

```
url = "https://www.ptt.cc/bbs/NBA/index.html"
```

用 requests 的 get 方法把網頁抓下來

```
res = req.get(url)
```

指定 lxml 作為解析器

```
soup = bs(res.text, "lxml")
```

建立 list 來放置列表資訊

```
list_posts = []
```

清空放置列表資訊的變數

```
list_posts.clear()
```

取得 列表 的文字與超連結

```
for a in soup.select('div.r-ent div.title a[href]'):
    print(a.get_text())
    print(a['href']) # 或是 a.get('href')
```

加入列表資訊

```
list_posts.append({
    'title': a.get_text(),
    'link': 'https://www.ptt.cc' + a['href']
})
```

走訪每一個 a link，整合網頁內文

```
for index, obj in enumerate(list_posts):
    res_ = req.get(obj['link'])
    soup_ = bs(res_.text, "lxml")
```

去掉 div.article-metaline (作者、標題、時間...等)

```

for div in soup_.select('div[class^="article-metaling"]'):
    div.decompose()

# 去掉 div.push (推文: 推、→、噓) (判斷元素是否存在)
if len( soup_.select('div.push') ) > 0:
    for div in soup_.select('div.push'):
        div.decompose()

# 取得實際需要的內容 (類似 JavaScript 的 innerHTML)
html = soup_.select_one('div#main-content').decode_contents()
# html = str(soup_.select_one('div#main-content')) # 類似
JavaScript outerHTML

# 預覽列表和內文
print(obj['title'])
print(obj['link'])
print(html)
print("=" * 50)

# 整合到列表資訊的變數當中
list_posts[index]['html'] = html

# 預覽所有結果
pprint(list_posts)

```

思考

- 如何取得多個分頁的內容?
 - 觀察分頁數字在網址的呈現方式
 - 將觀察到的分頁數字嵌入對應的網址當中

清空放置列表資訊的變數

```
list_posts.clear()
```

起始頁數

```
init_page = 6503
```

最新頁數

```

latest_page = 6504

# 在已經知道分頁數的情況下
for page in range(init_page, latest_page + 1):

    # 取得新聞列表
    url = f"https://www.ptt.cc/bbs/NBA/index{page}.html"

    # 用 requests 的 get 方法把網頁抓下來
    res = req.get(url)

    # 指定 lxml 作為解析器
    soup = bs(res.text, "lxml")

    # 取得 列表 的文字與超連結
    for a in soup.select('div.r-ent div.title a[href]'):
        # 加入列表資訊
        list_posts.append({
            'title': a.get_text(),
            'link': 'https://www.ptt.cc' + a['href']
        })

# 走訪每一個 a link，整合網頁內文
for index, obj in enumerate(list_posts):
    res_ = req.get(obj['link'])
    soup_ = bs(res_.text, "lxml")

    # 去掉 div.article-metaline (作者、標題、時間...等)
    for div in soup_.select('div[class^="article-metaline"]'):
        div.decompose()

    # 去掉 div.push (推文：推、→、噓) (判斷去掉元素是否存在)
    if len(soup_.select('div.push')) > 0:
        for div in soup_.select('div.push'):
            div.decompose()

    # 取得實際需要的內容 (類似 JavaScript 的 innerHTML)

```

```
html = soup_.select_one('div#main-content').decode_contents()

# 整合到列表資訊的變數當中
list_posts[index]['html'] = html

# 預覽所有結果
pprint(list_posts)
```

Module 10. 套件 Selenium (一)

解析 Selenium、WebDriver 與 Browser 連動關係

1. Selenium：自動化工具框架

- 作用：Selenium 是一個框架，用於測試網頁應用程式或自動化執行瀏覽器操作。
- 功能：
 - 提供 API，用於控制瀏覽器行為。
 - 支援多種程式語言（如 Python、Java、C# 等）。
 - 可進行瀏覽器操作，如點擊按鈕、填寫表單、截圖等。
- 運作原理：
 - Selenium 本身並不直接操作瀏覽器，而是依賴 WebDriver。

2. WebDriver：瀏覽器的控制介面

- 作用：WebDriver 是一種標準化的接口，用於與瀏覽器進行通訊。
- 功能：
 - 負責將 Selenium 的指令傳遞給瀏覽器，並回傳執行結果。
 - 每種瀏覽器都有專屬的 WebDriver（如 ChromeDriver、FirefoxDriver）。
- 特點：
 - 遵循 W3C WebDriver 標準，確保跨瀏覽器的相容性。
 - 作為 Selenium 與瀏覽器之間的「橋樑」。

3. Browser（瀏覽器）：執行目標

- 作用：Browser 是 WebDriver 操控的目標，也是最終執行 Selenium 指令的環境。
 - 功能：
 - 真正執行網頁上的動作（如載入頁面、點擊按鈕）。
- 支援多種瀏覽器（如 Google Chrome、Firefox、Safari 等）。

方法	說明
<code>get_window_position()</code>	取得瀏覽器視窗左上角位置
<code>set_window_position(x, y)</code>	設定瀏覽器視窗左上角位置
<code>get_window_size()</code>	取得瀏覽器視窗大小
<code>set_window_size(x, y)</code>	設定瀏覽器視窗大小
<code>maximize_window()</code>	將瀏覽器視窗最大化
<code>minimize_window()</code>	將瀏覽器視窗最小化

圖：還操控瀏覽器的位置與大小

屬性	說明
name	瀏覽器名稱
title	目前開啟網頁之標題
current_url	目前開啟網頁之 URL
page_source	目前開啟網頁之原始碼
session_id	網頁連線 id
capabilities	瀏覽器功能設定

圖：driver.page_source 外，還有其它可以使用

selenium 套件

```
'''
參考網頁：
[1] 下載 Chrome Web Driver
https://chromedriver.chromium.org/downloads
'''

# 操作 browser 的 API
# from selenium.webdriver.chrome.service import Service
from selenium import webdriver

# 匯入套件
from bs4 import BeautifulSoup as bs

# 強制等待（執行期間休息一下）
from time import sleep

# 使用 Chrome 的 WebDriver
'''
my_service = Service(executable_path="./chromedriver.exe")
driver = webdriver.Chrome(service=my_service)
'''

# 補充：若沒有特別設定，只要電腦有安裝 Chrome，就可以直接使用
driver = webdriver.Chrome()
```



```
# 開啟 104 人力行銀 首頁
driver.get("https://www.104.com.tw/jobs/main/")

# 取得檢視原始碼的內容 (page_source 取得的 html，是動態的、使用者操作過後的
結果)
html = driver.page_source

# 印出 html (也可以跟 BeautifulSoup 整合)
# print(html)

# 指定 lxml 作為解析器
soup = bs(html, "lxml")

# 取得元素
div = soup.select('div.header__container')[0]

# 顯示內文
print(div.get_text())

# 關閉瀏覽器
driver.quit()
```

Module 11. 套件 Selenium (二)

如何查找頁面元素 (ID / Class / Tag / CSS

Selector / ...)

註：新的 webdriver 版本已不支援 `find_element(s)_by_xxx()` 系列的語法

方法	說明
<code>find_element(by, value)</code>	使用 by 指定之方法取得第一個符合 value 的元素
<code>find_elements(by, value)</code>	使用 by 指定之方法取得所有符合 value 的元素
<code>find_element_by_class_name(name)</code>	傳回符合指定 class 名稱之元素
<code>find_elements_by_class_name(name)</code>	傳回符合指定 class 名稱之元素串列
<code>find_element_by_css_selector(selector)</code>	傳回符合指定 CSS 選擇器名稱之元素
<code>find_elements_by_css_selector(selector)</code>	傳回符合指定 CSS 選擇器名稱之元素串列
<code>find_element_by_id(id)</code>	傳回符合指定 id 之元素
<code>find_elements_by_id(id)</code>	傳回符合指定 id 之元素串列
<code>find_element_by_link_text(text)</code>	傳回符合指定超連結文字之元素
<code>find_elements_by_link_text(text)</code>	傳回符合指定超連結文字之元素串列
<code>find_element_by_partial_link_text(text)</code>	傳回符合部分指定超連結文字之元素
<code>find_elements_by_partial_link_text(text)</code>	傳回符合部分指定超連結文字之元素串列
<code>find_element_by_name(name)</code>	傳回符合指定元素名稱之元素
<code>find_elements_by_name(name)</code>	傳回符合指定元素名稱之元素串列
<code>find_element_by_tag_name(tag)</code>	傳回符合指定標籤名稱之元素
<code>find_elements_by_tag_name(tag)</code>	傳回符合指定標籤名稱之元素串列

圖：取得網頁元素方法

期待狀況/條件 (Expected Condition)

通常與 `WebDriverWait` 配合使用，動態等待頁面上元素出現或者消失。

- `title_is`
 - 判斷當前頁面的 title 是否精確等於預期
- `title_contains`
 - 判斷當前頁面的 title 是否包含預期字符串
- `presence_of_element_located`
 - 判斷某個元素是否被加到了 dom 樹裡，並不代表該元素一定可見
- `visibility_of_element_located`
 - 判斷元素是否可見。可見代表元素非隱藏，並且元素的寬和高都不等於 0

- **presence_of_all_elements_located**
 - 判斷是否至少有 1 個元素存在於 DOM tree 中。舉個例子，如果頁面上有 n 個元素的 class 都是 'col-md-3'，那麼只要有 1 個元素存在，這個方法就返回 True
- **text_to_be_present_in_element**
 - 判斷某個元素中的 text 是否包含了預期的字串
- **text_to_be_present_in_element_value**
 - 判斷某個元素中的 value 屬性是否包含了預期的字串
- **frame_to_be_available_and_switch_to_it**
 - 判斷該 frame 是否可以 switch 進去，如果可以的話，返回 True 並且 switch 進去，否則返回 False
- **invisibility_of_element_located**
 - 判斷某個元素中是否不存在於 DOM tree 或不可見
- **element_to_be_clickable**
 - 判斷某個元素中是否可見並且是 enable 的，這樣的話才叫 clickable
- **staleness_of**
 - 等某個元素從 dom 樹中移除，注意，這個方法也是返回 True 或 False
- **element_to_be_selected**
 - 判斷某個元素是否被選中了，一般用在下拉列表
- **element_selection_state_to_be**
 - 判斷某個元素的選中狀態是否符合預期
- **element_located_selection_state_to_be**
 - 跟上面的方法作用一樣，只是上面的方法傳入定位到的 element，而這個方法傳入 locator
- **alert_is_present**
 - 判斷頁面上是否存在 alert，這是個老問題，很多同學會問到

元素定位策略/方式 (By)

- By.ID = "id"
- By.CSS_SELECTOR = "css selector"
- By.XPATH = "xpath"
- By.LINK_TEXT = "link text"
- By.PARTIAL_LINK_TEXT = "partial link text"

- `By.NAME = "name"`
- `By.TAG_NAME = "tag name"`
- `By.CLASS_NAME = "class name"`

匯入自動測試工具相關套件

```
'''
匯入套件
'''

# 操作 browser 的 API
from selenium import webdriver

# 處理逾時例外的工具
from selenium.common.exceptions import TimeoutException

# 面對動態網頁，等待某個元素出現的工具，通常與 expected_conditions 搭配
from selenium.webdriver.support.ui import WebDriverWait

# 搭配 WebDriverWait 使用，對元素狀態的一種期待條件，若條件發生，則等待結束，往下一行執行
from selenium.webdriver.support import expected_conditions as EC

# 期待元素出現要透過什麼方式指定，通常與 EC、WebDriverWait 一起使用
from selenium.webdriver.common.by import By

# 強制等待（執行期間休息一下）
from time import sleep

'''

selenium 啟動 Chrome 的進階配置參數
參考網址：https://stackoverflow.max-everyday.com/2019/12/selenium-chrome-options/
'''

# 啟動瀏覽器工具的選項
my_options = webdriver.ChromeOptions()
# my_options.add_argument("--headless") #不開啟實體瀏覽器
背景執行
```

```

my_options.add_argument("--start-maximized")      #最大化視窗
my_options.add_argument("--incognito")            #開啟無痕模式
my_options.add_argument("--disable-popup-blocking") #禁用彈出攔截
my_options.add_argument("--disable-notifications") #取消 chrome 推播通知
my_options.add_argument("--lang=zh-TW")          #設定為正體中文

# 使用 Chrome 的 WebDriver
driver = webdriver.Chrome(
    options = my_options
)

```

在瀏覽器中執行自訂 JavaScript 程式

```

# 開啟網頁
driver.get("https://crptransfer.moe.gov.tw/")

# 跳出 alert 視窗 (在 chrome 裡面執行 javascript 語法)
driver.execute_script("window.alert('這是我們自訂的彈跳視窗');")

# 等個幾秒
sleep(3)

# 點選彈出裡面的確定按鈕
driver.switch_to.alert.accept()

```

輸入文字，送出表單

```

# 開啟網頁
driver.get("https://crptransfer.moe.gov.tw/")

# 尋找網頁中的搜尋框
inputElement = driver.find_element(
    By.CSS_SELECTOR, 'input#SN'
)

# 在搜尋框中輸入文字
inputElement.send_keys("人帥真好")

```

```

# 睡個幾秒
sleep(2)

# 送出搜尋
inputElement.submit()

# 搜尋結果的 CSS Selector
cssSelector = "body > table > tbody > tr:nth-child(1) > td > main >
article > div > table > tbody > tr:nth-child(2) > td"

try:
    # 等待網頁搜尋結果
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located(
            (By.CSS_SELECTOR, cssSelector)
        )
    )

    # 取得第一頁搜尋結果
    element = driver.find_element(
        By.CSS_SELECTOR, cssSelector
    )

    # 輸出想要爬取的文字
    print(element.text)
    print(element.get_attribute('innerText')) # 另一種寫法
except TimeoutException:
    print('等待逾時！')

```

輸入文字，按下送出鈕

```

# 開啟網頁
driver.get("https://www.104.com.tw/jobs/main/")

# 尋找網頁中的搜尋框
inputElement = driver.find_element(
    By.CSS_SELECTOR, 'input[data-gtm-index^="搜尋欄位"]'
)

```

```

)

# 在搜尋框中輸入文字
inputElement.send_keys("python")

# 睡個幾秒
sleep(3)

# 按鈕選擇器
cssSelectorBtn = 'button.btn[type="submit"][data-gtm-index^="搜尋欄位"]'

try:
    # 等待元素
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located(
            (By.CSS_SELECTOR, cssSelectorBtn)
        )
    )

    # 取得按鈕元素
    btn = driver.find_element(
        By.CSS_SELECTOR, cssSelectorBtn
    )

    # 按下按鈕
    btn.click()
except TimeoutException:
    print('等待逾時！')

```

刷新頁面（類似 F5 或 Ctrl + R）

```

# 開啟網頁
driver.get("https://reurl.cc/jR725D")

# 睡個幾秒
sleep(3)

# 刷新頁面

```

```
driver.refresh()
```

```
# 睡個幾秒
```

```
sleep(3)
```

```
# 刷新頁面
```

```
driver.refresh()
```

```
# 睡個幾秒
```

```
sleep(3)
```

```
# 關閉瀏覽器
```

```
driver.quit()
```


Module 12. 套件 Selenium (三)

等待 (WebDriverWait)

等待有分幾種：

- 強制等待
 - 通常泛指 `sleep()` 函式
- 隱性等待 (`implicitly_wait`)
 - 設置了一個最長等待時間，如果在規定時間內網頁加載完成，或是能夠取得指定的元素（透過 `find_element*`），則執行下一步，否則一直等到時間截止，然後拋出例外。
- 顯性等待 (`WebDriverWait`)
 - 配合 `until()` 和 `until_not()` 方法，就能夠根據判斷條件而進行靈活地等待了。它主要的意思就是：如果條件成立了，則執行下一步，否則繼續等待，直到超過設置的最長時間，直到拋出 `TimeoutException`。

匯入自動測試工具相關套件

'''

參考網址：

[1] Webdriver Manager for Python

<https://pypi.org/project/webdriver-manager/>

'''

匯入套件

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from webdriver_manager.chrome import ChromeDriverManager

from selenium.common.exceptions import TimeoutException,
NoSuchElementException

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

from selenium.webdriver.common.by import By

from time import sleep

自動取得 Chrome WebDriver

(Optional) 開啟用於自動控制的瀏覽器 (自動取得 Chrome 的 WebDriver)

```
driver = webdriver.Chrome(  
    service = Service(ChromeDriverManager().install())  
)
```

強制等待

```
'''  
強制等待  
'''  
  
# 開啟用於自動控制的瀏覽器（自動取得 Chrome 的 WebDriver）  
driver = webdriver.Chrome(  
    service = Service(ChromeDriverManager().install())  
)  
  
try:  
    # 走訪網址  
    driver.get('https://tw.yahoo.com/')  
  
    # 強制等待 3 秒  
    sleep(3)  
  
    # 印出網址  
    print(driver.current_url)  
except:  
    print("程式出錯!")  
finally:  
    # 關閉瀏覽器  
    driver.quit()
```

隱性等待

```
'''  
隱性等待  
'''  
  
# 開啟用於自動控制的瀏覽器  
driver = webdriver.Chrome()  
  
try:  
    # 最多等 15 秒
```

```

driver.implicitly_wait(15)

# 走訪網址
driver.get('https://tw.yahoo.com/')

# 取得元素
element = driver.find_element(By.CSS_SELECTOR, 'a#header-logo')

# 印出超連結 ( 透過 .get_attribute('屬性') 來取得屬性的值 )
print(element.get_attribute('href'))
except NoSuchElementException:
    print("找不到元素!")
finally:
    # 關閉瀏覽器
    driver.quit()

```

顯性等待

```

'''
顯性等待
'''

# 開啟用於自動控制的瀏覽器
driver = webdriver.Chrome()

try:
    # 走訪網址
    driver.get('https://www.youtube.com/?gl=TW')

    # 滿足條件 (10 秒內找到元素)，則往下一步
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located(
            (By.LINK_TEXT, '首頁')
        )
    )

    # 印出首頁連結
    print(driver.find_element(By.LINK_TEXT, '首頁')
        .get_attribute('href'))

```

```
except TimeoutException:  
    print('等待逾時!')  
finally:  
    # 關閉瀏覽器  
    driver.quit()
```

Module 13. ActionChains

匯入工具

加入行為鍊 ActionChains (在 WebDriver 中模擬滑鼠移動、點擊、拖曳、按右鍵出現選單，以及鍵盤輸入文字、按下鍵盤上的按鈕等)

```
from selenium.webdriver.common.action_chains import ActionChains
```

基本用法

方法	說明
click(on_element=None)	單擊滑鼠左鍵
click_and_hold(on_element=None)	點選滑鼠左鍵，不鬆開
context_click(on_element=None)	點選滑鼠右鍵
double_click(on_element=None)	雙擊滑鼠左鍵
drag_and_drop(source, target)	拖拽到某個「元素」然後鬆開
drag_and_drop_by_offset(source, xoffset, yoffset)	拖拽到某個「座標」然後鬆開
key_down(value, element=None)	按下某個鍵盤上的鍵
key_up(value, element=None)	鬆開某個鍵
move_by_offset(xoffset, yoffset)	滑鼠從當前位置移動到某個座標
move_to_element(to_element)	滑鼠移動到某個元素
move_to_element_with_offset(to_element, xoffset, yoffset)	移動到距某個元素 (左上角座標) 多少距離的位置
pause(seconds)	暫停動作一段時間
perform()	執行鏈中的所有動作
release(on_element=None)	在某個元素位置鬆開滑鼠左鍵
send_keys(keys_to_send)	傳送某個鍵到當前焦點的元素
send_keys_to_element(element, keys_to_send)	傳送某個鍵到指定元素

寫法

- 鍊式

```
ActionChains(driver).move_to_element( web_element ).click( web_element ).perform()
```

或是

```
action_chains = ActionChains(driver)
```

```
action_chains.move_to_element( web_element ).click( web_element ).perform()
```

- 分步

```
action_chains = ActionChains(driver)
```

```
action_chains.move_to_element( web_element )
```

```
action_chains.click( web_element )
```

```
action_chains.perform()
```

補充：切換目前到 iframe 當中

- 取得網頁上的 iframe

```
iframe = driver.find_element(By.CSS_SELECTOR, "iframe#game-iframe")
```

- 切換到 iframe 當中

```
driver.switch_to.frame(iframe)
```

- 回到主框架

```
driver.switch_to.default_content()
```

參考連結

1. Selenium 的 ActionChains Api 介面詳解

<https://www.796t.com/article.php?id=325399>

2. python selenium 滑鼠鍵盤操作 (ActionChains)

<https://www.796t.com/article.php?id=94198>

3. 行為鏈

https://python-selenium-zh.readthedocs.io/zh_CN/latest/7.2%E8%A1%8C%E4%B8%BA%E9%93%BE/

4. ActionChains In Selenium

<https://medium.com/@kavidhanda/actionchains-in-selenium-cde43dee0111>

5. Selenium 4.1.0 documentation -
selenium.webdriver.common.action_chains

https://www.selenium.dev/selenium/docs/api/py/webdriver/selenium.webdriver.common.action_chains.html

匯入工具

'''

匯入工具

備註：每次執行以下任一範例前，都要執行一次"匯入工具"的 cell

'''

操作 browser 的 API

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from webdriver_manager.chrome import ChromeDriverManager

期待元素出現要透過什麼方式指定，通常與 EC、WebDriverWait 一起使用

from selenium.webdriver.common.by import By

加入行為鍊 ActionChain (在 WebDriver 中模擬滑鼠移動、點擊、拖曳、按右鍵出現選單，以及鍵盤輸入文字、按下鍵盤上的按鈕等)

from selenium.webdriver.common.action_chains import ActionChains

加入鍵盤功能 (例如 Ctrl、Alt 等)

from selenium.webdriver.common.keys import Keys

強制等待 (執行期間休息一下)

from time import sleep

啟動瀏覽器工具的選項

my_options = webdriver.ChromeOptions()

my_options.add_argument("--headless")

#不開啟實體瀏覽器

背景執行

```
my_options.add_argument("--start-maximized")    #最大化視窗
my_options.add_argument("--incognito")          #開啟無痕模式
my_options.add_argument("--disable-popup-blocking") #禁用彈出攔截
my_options.add_argument("--disable-notifications") #取消通知
```

範例 1：對特定座標連續點擊

'''

範例 1：對特定座標連續點擊

來源連結

<https://cpstest.org/>

補充：

先在 console 面板中，輸入下列程式碼，會在滑鼠一次移動後，顯示座標。

需要先在 console 面板中輸入 allow pasting。

```
document.onmousemove = function(e){
    var x = e.pageX;
    var y = e.pageY;
    e.target.title = "X is " + x + " and Y is " + y;
};
```

參考連結：

[1] Is there a way to tell Chrome web debugger to show the current mouse position in page coordinates?

<https://stackoverflow.com/questions/12888584/is-there-a-way-to-tell-chrome-web-debugger-to-show-the-current-mouse-position-in>

'''

使用 Chrome 的 WebDriver

```
driver = webdriver.Chrome(
    options = my_options
)
```

前往頁面

```
driver.get('https://cpstest.org/10-seconds.php')
```



```

# (Optional) 刪除不必要的網頁元素
driver.execute_script("""
let element = document.querySelector('div.stdadd-class');
if (element) {
    element.parentNode.removeChild(element);
}
""")

# 建立行為鍊
ac = ActionChains(driver)

# 移到指定座標 (視解析度與瀏覽器視窗大小而定)
'''
從 0,0 開始，若先前已移動，則進行相對位移，數值要用負號
例如 ac.move_by_offset(-50, -80)
'''
ac.move_by_offset(652, 471)

# 暫停一下
ac.pause(3)

# 點擊一下
for i in range(10000):
    ac.click()

# 執行
ac.perform()

# 睡一下
sleep(5)

# 關閉 web driver
driver.quit()

```

範例 2: 拖曳網頁元素 (使用 drag_and_drop)

```
'''
```

範例 2: 拖曳網頁元素

參考連結:

[1] Mootools Drag and Drop example

<http://sahitest.com/demo/dragDropMooTools.htm>

...

使用 Chrome 的 WebDriver

```
driver = webdriver.Chrome(  
    options = my_options  
)
```

前往頁面

```
driver.get('http://sahitest.com/demo/dragDropMooTools.htm')
```

取得被拖曳的來源元素

```
dragger = driver.find_element(By.CSS_SELECTOR, "div#dragger")
```

目標元素 (放置的區域, 共 4 個)

```
items = driver.find_elements(By.CSS_SELECTOR, "div.item")
```

建立行為鍊

```
ac = ActionChains(driver)
```

暫停一下

```
ac.pause(1)
```

放置第一個

```
ac.drag_and_drop(dragger, items[0])
```

暫停一下

```
ac.pause(1)
```

放置第二個

```
ac.click_and_hold(dragger).release(items[1])
```

暫停一下

```
ac.pause(1)
```

```
# 放置第三個
ac.click_and_hold(dragger).move_to_element(items[2]).release()

# 暫停一下
ac.pause(1)

# 放置第四個
ac.click_and_hold(dragger).move_by_offset(400, 150).release()

# 執行
ac.perform()

# 睡一下
sleep(3)

# 關閉 web driver
driver.quit()
```

範例 3：組合熱鍵（全選 + 複製 + 貼上）

```
'''
範例 3：組合熱鍵（全選 + 複製 + 貼上）

參考連結：
[1] Display Text Input Fields
https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5\_input\_type\_text
'''

# 使用 Chrome 的 WebDriver
driver = webdriver.Chrome(
    options = my_options
)

# 前往頁面
driver.get('https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5_input_type_text')
```

```
# 取得網頁上的 iframe
iframe = driver.find_element(By.CSS_SELECTOR, "iframe#iframeResult")

# 切換到 iframe 當中
driver.switch_to.frame(iframe)

# # 回到主框頁
# driver.switch_to.default_content()

# 取得第一個文字欄位
inputText01 = driver.find_element(By.CSS_SELECTOR, "input#fname")

# 取得第二個文字欄位
inputText02 = driver.find_element(By.CSS_SELECTOR, "input#lname")

# 建立行為鍊
ac = ActionChains(driver)

# 在第一個文字欄位當中輸入萬用字元
ac.key_down(Keys.SHIFT,
inputText01).send_keys('12345').key_up(Keys.SHIFT).send_keys('67890')

# 暫停一下
ac.pause(1)

# 全選與複製第一個文字欄位當中的所有字元
ac.key_down(Keys.CONTROL,
inputText01).send_keys('ac').key_up(Keys.CONTROL)

# 暫停一下
ac.pause(1)

# 在第二個文字欄位當中貼上文字
ac.key_down(Keys.CONTROL,
inputText02).send_keys('v').key_up(Keys.CONTROL)

# 執行
```

```
ac.perform()

# 睡一下
sleep(3)

# 關閉 web driver
driver.quit()
```

範例 4: 移動 Slider (by a handle)

```
'''
範例 4: 移動 Slider (by a handle)

參考連結:
[1] jQuery UI - Slider
https://jqueryui.com/slider/
'''

# 前往頁面
driver.get('https://jqueryui.com/slider/')

# 取得網頁上的 iframe
iframe = driver.find_element(By.CSS_SELECTOR, "iframe.demo-frame")

# 切換到 iframe 當中
driver.switch_to.frame(iframe)

# 拿到把手元素
span = driver.find_element(By.CSS_SELECTOR, 'span.ui-slider-
handle.ui-corner-all.ui-state-default')

# 建立行為鍊
ac = ActionChains(driver)

# 暫停
ac.pause(1)
```

```
# 點住不放，並向右移動，移到指定的座標（註：目前在 iframe 當中，使用的座標是
iframe 內部座標）
# 註：點住移動，只能在 iframe 範圍內，超出的話，span 會放不掉
ac.click_and_hold(span).move_by_offset(577, 16).release()

# 執行
ac.perform()

# 睡一下
sleep(5)

# 關閉 web driver
driver.quit()
```