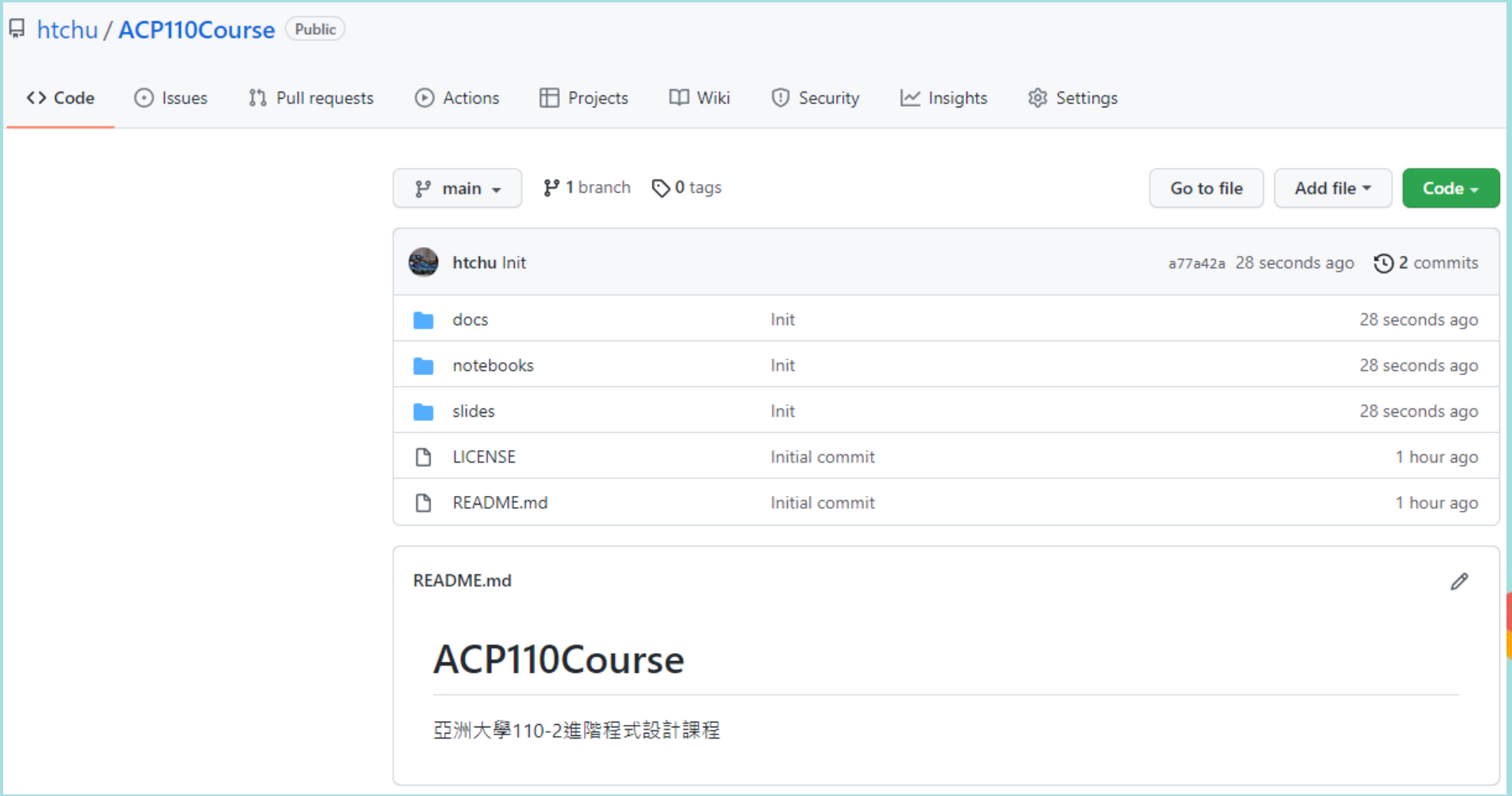
111-2進階程式設計課程(6)
Advanced Computer Programming

亞大資工系

課程大綱

- W1-課程介紹/Introduction
- W2-Python libraries
- W3-BeautifulSoup(1)
- W4-BeautifulSoup(2)
- W5-
- W6-Scrapy(1)
- W7-Scrapy(2)
- W8-Storing Data
- W9-Midterm project
- W10-Web & HTTP
- W11-Flask
- W12-Flask Routes
- W13-Jinja template
- W14-Flask-form
- W15-Flask-mail
- W16-REST API
- W17-Project development(2)
- W18-Final presentation

教材Github



https://github.com/htchu/ACP110Course

大綱

- Review-Web Crawlers
- Review-robots.txt
  - https://developers.google.com/search/docs/advanced/robots/robots_txt
- Review-Regular Expressions

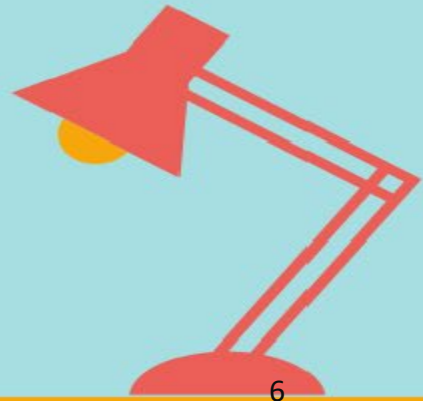
- 作業 2(Assignment 2):

作業1/Assignment 1

- 概述：
  - 在本次作業中，我們將使用基本的 Python 網頁抓取工具 urllib 和 BeautifulSoup 從 cna 網站抓取數據。 請列出抓取的新聞內容並將其提交到 Tronclass 的作業條目。
- 目標：
  - 了解如何使用網頁抓取獲取網頁內容。
  - 探索真正的 html 文件。
  - 反思網絡抓取功能在數據科學中的可能用途。
- 指示：
  - 使用任何瀏覽器訪問 www.cna.com.tw 網站。
  - 檢查 html 內容中的標籤。

作業2/Assignment 2

- 概述：
  - 在本次作業中，我們將使用BeautifulSoup從 cna 網站抓取所有的新聞內容並將其存到一個txtd檔提交到 Tronclass 的作業條目。
- 目標：
  - 了解如何使用網頁抓取獲取網頁link。
  - 探索crawling an entire site。
  - 反思網絡抓取功能在數據科學中的可能用途。
- 指示：
  - 使用任何瀏覽器訪問 focustaiwan 網站。 www.cna.com.tw
  - 檢查 html 內容中的標籤。



Review-Building Scrapers

Adobe Digital Editions - Web Scraping with Python

檔案(F) 編輯(E) 閱讀(R) 說明(H)

◀ 圖書館

目錄

Chapter 1. Your First Web Scraper

Once you start web scraping, you start to appreciate all the little things that browsers do for you. The web, without a layer of HTML formatting, CSS styling, JavaScript execution, and image rendering, can lo first, but in this chapter, as well as the next one, we'll cover how to format and interpret data without the help of a browser.

This chapter starts with the basics of sending a GET request (a request to fetch, or "get," the content of a web page) to a web server for a specific page, reading the HTML output from that page, and doing som in order to isolate the content that you are looking for.

Connecting

If you haven't spent much time in networking or network security, the mechanics of the internet might seem a little mysterious. You don't want to think about what, exactly, the network is doing every time yo go to *http://google.com*, and, these days, you don't have to. In fact, I would argue that it's fantastic that computer interfaces have advanced to the point where most people who use the internet don't have the f works.

However, web scraping requires stripping away some of this shroud of interface—not just at the browser level (how it interprets all of this HTML, CSS, and JavaScript), but occasionally at the level of the net

To give you an idea of the infrastructure required to get information to your browser, let's use the following example. Alice owns a web server. Bob uses a desktop computer, which is trying to connect to Alic machine wants to talk to another machine, something like the following exchange takes place:

1. Bob's computer sends along a stream of 1 and 0 bits, indicated by high and low voltages on a wire. These bits form some information, containing a header and body. The header contains an immediat router's MAC address, with a final destination of Alice's IP address. The body contains his request for Alice's server application.
2. Bob's local router receives all these 1s and 0s and interprets them as a packet, from Bob's own MAC address, destined for Alice's IP address. His router stamps its own IP address on the packet as the sends it off across the internet.
3. Bob's packet traverses several intermediary servers, which direct his packet toward the correct physical/wired path, on to Alice's server.
4. Alice's server receives the packet at her IP address.
5. Alice's server reads the packet port destination in the header, and passes it off to the appropriate application—the web server application. (The packet port destination is almost always port 80 for web thought of as an apartment number for packet data, whereas the IP address is like the street address.)
6. The web server application receives a stream of data from the server processor. This data says something like the following:

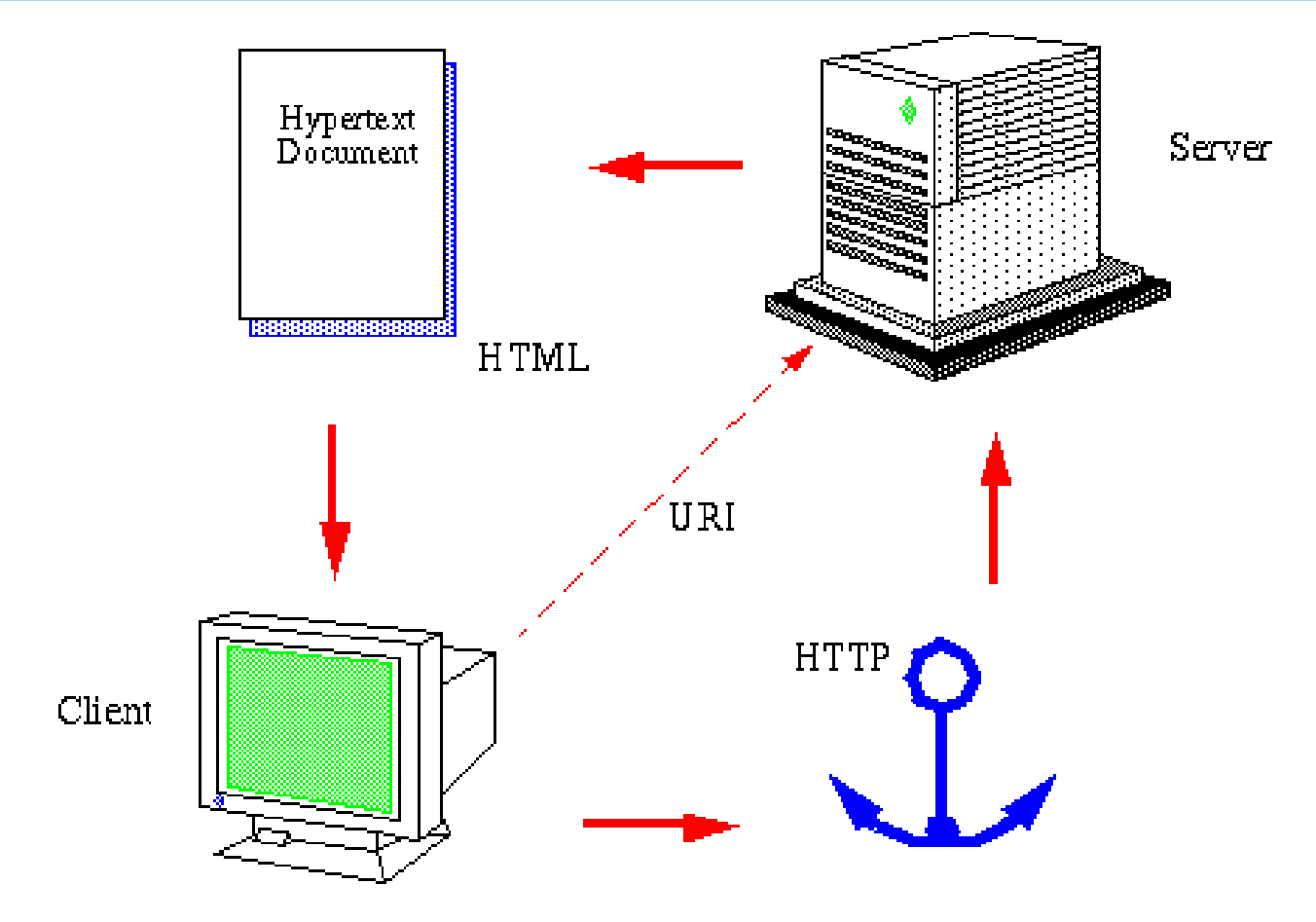
   - This is a GET request.

   - The following file is requested: *index.html*.
7. The web server locates the correct HTML file, bundles it up into a new packet to send to Bob, and sends it through to its local router, for transport back to Bob's machine, through the same process.

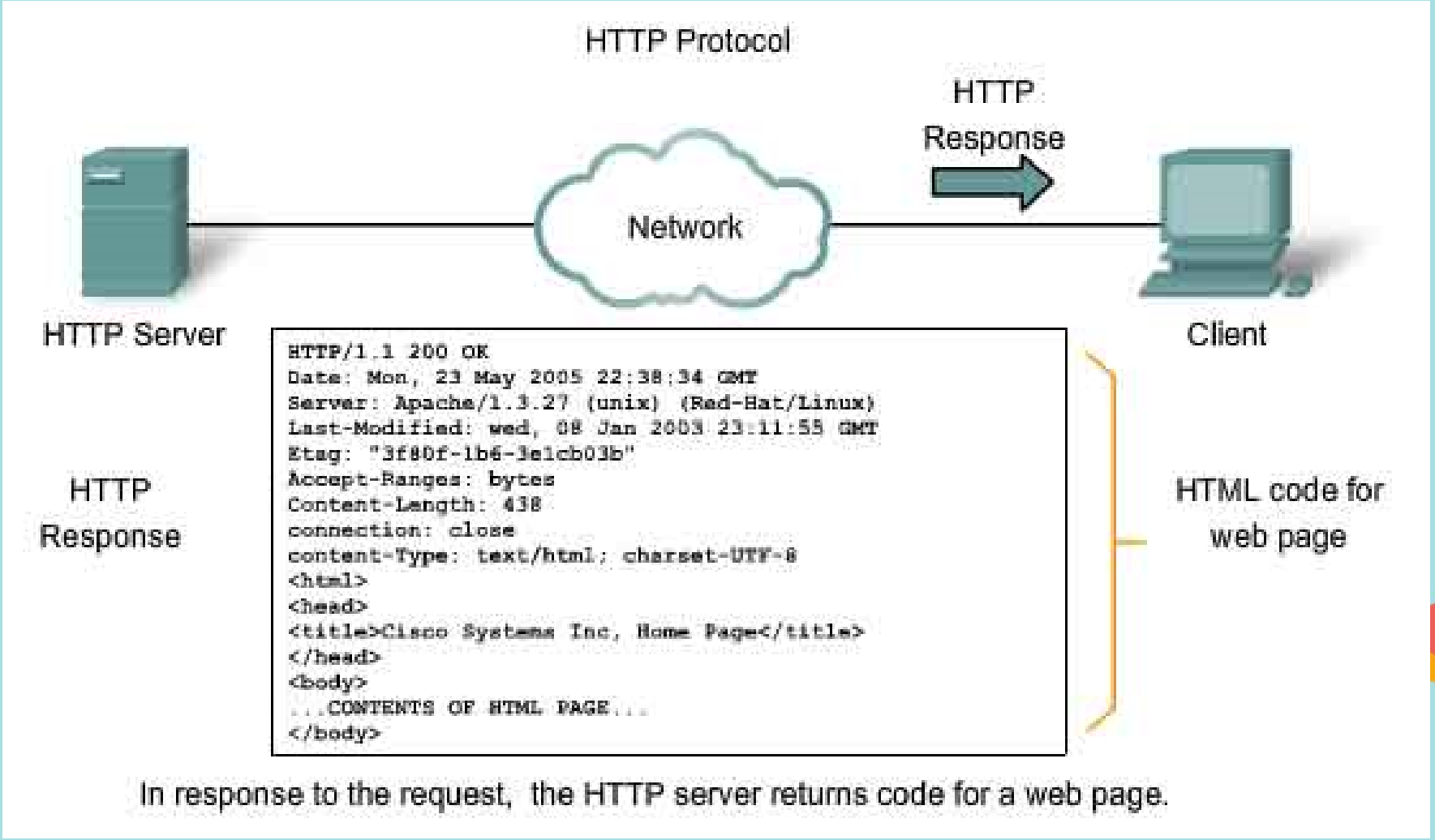
HTTP, URI, HTML

```
<!DOCTYPE html>
<html>
<!-- created 2010-01-01 -->
 <head>
  <title>sample</title>
 </head>
 <body>
  <p>Voluptatem accusantium
   totam rem aperiam.</p>
 </body>
</html>
```

HTML



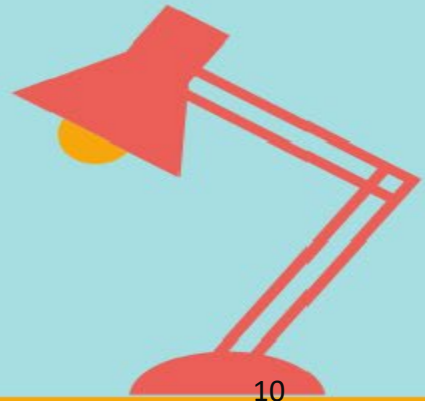
HTTP


HTTP Protocol
HTTP
Response
Network
HTTP Server
Client
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (unix) (Red-Hat/Linux)
Last-Modified: wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
connection: close
content-Type: text/html; charset-UTF-8
<html>
<head>
<title>Cisco Systems Inc, Home Page</title>
</head>
<body>
...CONTENTS OF HTML PAGE...
</body>
HTTP
Response
HTML code for
web page
In response to the request, the HTTP server returns code for a web page.

robots.txt

- A robots.txt file is a simple text file containing rules about which crawlers may access which parts of a site. For example, the robots.txt file for example.com may look like this:

```
# This robots.txt file controls crawling of URLs under https://example.com.
# All crawlers are disallowed to crawl files in the "includes" directory, such
# as .css, .js, but Googlebot needs them for rendering, so Googlebot is allowed
# to crawl them.
User-agent: *
Disallow: /includes/

User-agent: Googlebot
Allow: /includes/

Sitemap: https://example.com/sitemap.xml
```



https://developers.google.com/search/docs/advanced/robots/robots_txt

Regular Expressions in Python

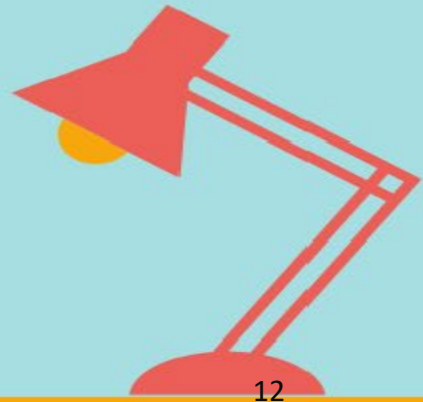
正規表示法 (Regex) 是用來處理字串的方法，Regex 用自己一套特殊的符號表示法，讓我們可以很方便的搜尋字串、取代字串、刪除字串或測試字串是否符合樣式規則。
Regex 它不是一個程式語言，他只是一種「字串樣式規則」的「表示法」，用來表達字元符號在字串中出現的規則，大部分的程式語言都有支援 Regex 的用法，而任何工具只要支援這種表示法，你就可以在這工具上用 Regex 來處理字串。

Regex 表示式	說明
\d{4}-\d{2}-\d{2}	從文本裡找出 YYYY-MM-DD 格式的日期字串
cat|dog	從文本裡找出 cat 或 dog 字串
[A-Z]\w+	從文本裡找出所有字首是大寫的英文字
^[A-Za-z]\d{9}$	驗證字串是否是台灣身份證字號

https://www.dataquest.io/blog/regex-cheatsheet/

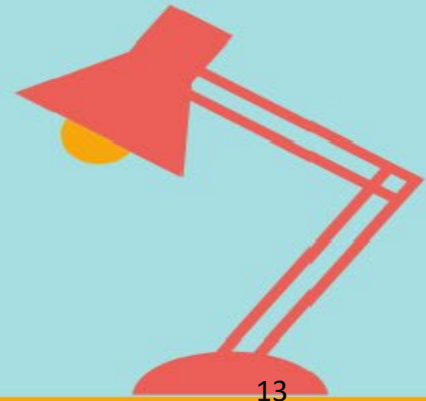
Regex 正規表示法
基本語法

- 或 (or)：| 管線符號，用來將所有可能的選擇條件分隔開。
- 群組 (grouping)：() 小括號，用來表示作用範圍或優先順序。。
- 量詞 (quantifier)：quantifier 用來接在字符串或群組後面，表示某個條件應該出現「幾次」。常見的量詞有：
  - ?：表示連續出現 0 次或 1 次。例如 colou?r 可以用來匹配 color 或 colour。
  - *：表示連續出現 0 次或多次。例如 ab*c 可以用來匹配 ac, abc, abbc, abbbc 或 abbbbbbc。
  - +：表示連續出現 1 次或多次。例如 ab+c 可以用來匹配 abc, abbc, abbbc, abbbbbbc，但 ac 不符合。
  - {min,max}：表示至少連續出現 min 次，但最多連續出現 max 次。
- **字元**
  - \d 用來匹配所有阿拉伯數字 0-9
  - \s 用來匹配所有的空白字元
  - \w所有非空白字元,意思同等於 [A-Za-z0-9_]
  - .(dot or period) 用來匹配除了換行符號 (line breaks) \n \r 之外的任何一個字元



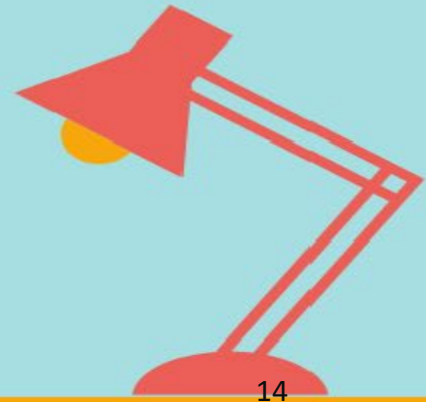
Anchors — ^ and $

- ^The matches any string that starts with The -> Try it!
- end$ matches a string that ends with end
- ^The end$ exact string match (starts and ends with The end)



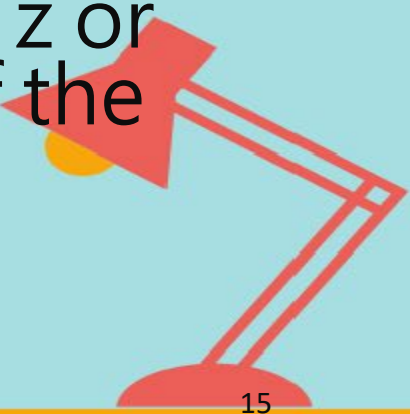
Grouping and capturing

- a(bc) parentheses create a capturing group with value bc
- a(?:bc)* using ?: we disable the capturing group
- a(?<foo>bc) using ?<foo> we put a name to the group



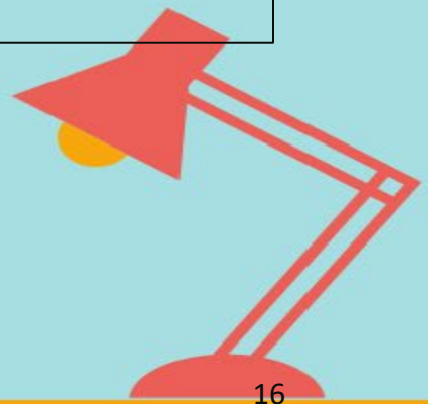
Bracket expressions — []

- [abc] matches a string that has either an a or a b or a c -> is the same as a|b|c
- [a-c] same as previous
- [a-fA-F0-9] a string that represents a single hexadecimal digit, case insensitively
- [0-9]% a string that has a character from 0 to 9 before a % sign
- [^a-zA-Z] a string that has not a letter from a to z or from A to Z. In this case the ^ is used as negation of the expression



Regular Expressions and BeautifulSoup

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

html = urlopen('http://www.pythonscraping.com/pages/page3.html')
bs = BeautifulSoup(html, 'html.parser')
images = bs.find_all('img', {'src':re.compile('\.\.\/img\/gifts/img.*\.jpg')})
for image in images:
    print(image['src'])
```



Lambda Expressions

function definition

```
def max(m, n):
    return m if m > n else n

print(max(10, 3)) # 顯示 10
```

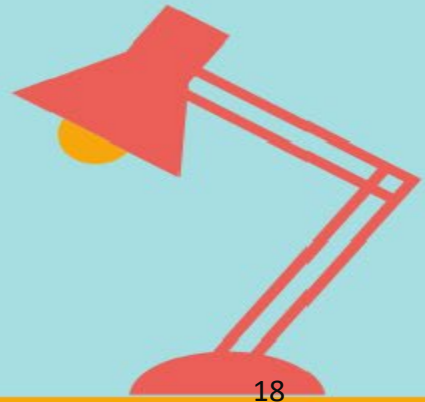
Lambda function

```
max = lambda m, n: m if m > n else n
print(max(10, 3)) # 顯示 10
```

```
bs.find_all(lambda tag: len(tag.attrs) == 2)

bs.find_all(lambda tag: tag.get_text() == 'Or maybe he\'s only resting?')
```

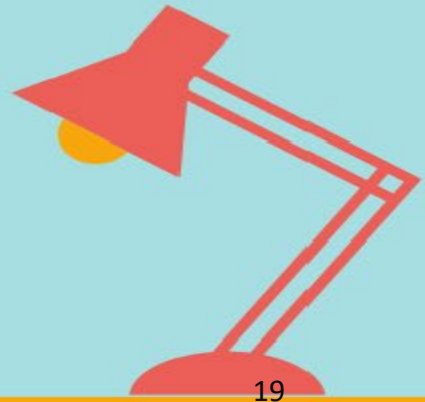
Writing Web Crawlers

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://en.wikipedia.org/wiki/Kevin_Bacon')
bs = BeautifulSoup(html, 'html.parser')
for link in bs.find_all('a'):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```



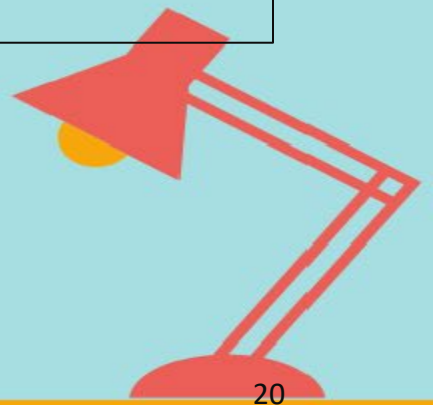
Recursively crawling an entire site

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

pages = set()
def getLinks(pageUrl):
    global pages
    html = urlopen('http://en.wikipedia.org{}'.format(pageUrl))
    bs = BeautifulSoup(html, 'html.parser')
    for link in bs.find_all('a', href=re.compile('^(/wiki/)')):
        if 'href' in link.attrs:
            if link.attrs['href'] not in pages:
                #We have encountered a new page
                newPage = link.attrs['href']
                print(newPage)
                pages.add(newPage)
                getLinks(newPage)
getLinks('')
```



Regular Expressions and BeautifulSoup

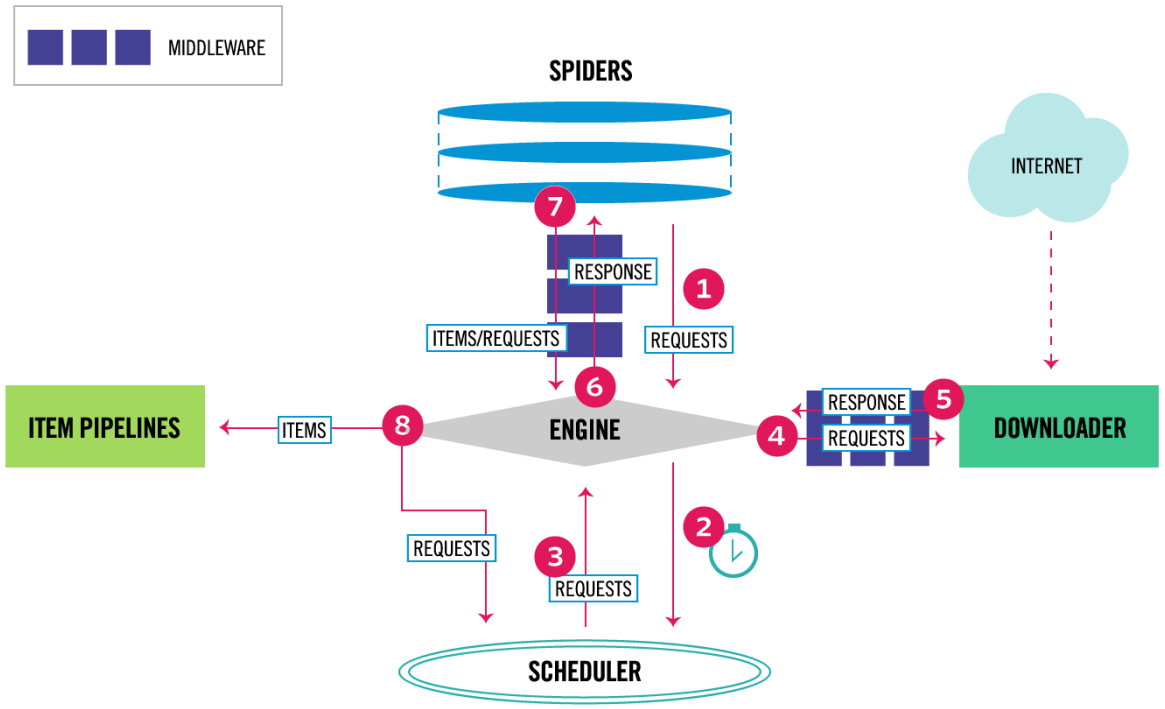
```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

html = urlopen('http://www.pythonscraping.com/pages/page3.html')
bs = BeautifulSoup(html, 'html.parser')
images = bs.find_all('img', {'src':re.compile('\.\.\/img\/gifts/img.*\.jpg')})
for image in images:
    print(image['src'])
```



Scrapy爬蟲框架

- **Scrapy Engine(引擎)**: 負責Spider、ItemPipeline、Downloader、Scheduler中間的通訊，信號、數據傳遞等。
- **Scheduler(調度器)**: 它負責接受引擎發送過來的Request請求，並按照一定的方式進行整理排列，入隊，當引擎需要時，交還給引擎。
- **Downloader（下載器）**：負責下載Scrapy Engine(引擎)發送的所有Requests請求，並將其獲取到的Responses交還給Scrapy Engine(引擎)，由引擎交給Spider來處理，
- **Spider（爬蟲）**：它負責處理所有Responses,從中分析提取數據，獲取Item欄位需要的數據，並將需要跟進的URL提交給引擎，再次進入Scheduler(調度器).
- **Item Pipeline(管道)**：它負責處理Spider中獲取到的Item，並進行進行後期處理（詳細分析、過濾、存儲等）的地方。
- **Downloader Middlewares（下載中間件）**：你可以當作是一個可以自定義擴展下載功能的組件。
- **Spider Middlewares（Spider中間件）**：你可以理解爲是一個可以自定擴展和操作引擎和Spider中間通信的功能組件（比如進入Spider的Responses;和從Spider出去的Requests）

Scrapy爬蟲流程



Scrapy Engine(引擎):
Scheduler(調度器):
Downloader（下載器）
Spider（爬蟲）
Item Pipeline(管道)
Downloader Middlewares（下載中間件）
Spider Middlewares（Spider中間件））

1. Spider發送最初的請求(Requests)給Engine。
2. Engine在Scheduler調度一個請求(Requests)，並要求下一次Requests做爬取。
3. Scheduler回傳下一個Requests給Engine。
4. Engine透過Downloader Middlewares發送請求給Downloader。
5. 只要頁面結束下載，Downloader產生一個Response透過Downloader Middlewares傳送給Engine。
6. Engine收到來自Downloader的Response並透過Spider Middlewares發送給Spider處理。
7. Spider處理Response並爬取的項目(item)和新的請求(Requests)，透過Spider Middlewares回傳給Engine。
8. Engine發送處理的項目(item)給Item Pipelines接著發送處理的請求(Requests)到Scheduler要求下一個可能的爬蟲請求。

Thanks!

Q&A