web scraping with python

Type	Note
Status	In Progress

covers

- inspecting html of target site
- requests and BeautifulSoup for scraping and parsing

the different libraries

- bs
- scrapy
- selenium
- pandas

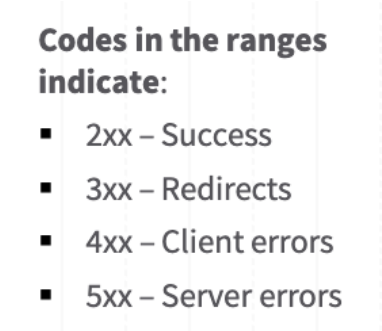
http requests

- GET / POST (le common)
- others exist like PUT / DELETE
- URL → uniform resource locator / basically the address of what youre trying to get
- HEADers
  - extra pieces of info, like:
  - user agent - tells more about the device you use, browser etc

    - cookie - a special value sent back and forth, just to remember info about you
- could have a Body containing stuff youre sending like `user=admin&pass=123`

http response

- status codes
    - 200 OK
    - 404 not found
    - 500 Internal Server error
    - and plenty more...



- Responses also have HEADers
- Body - the actual content you requested for like a html page

i love json

- javascript object notation
- 3 key concepts
    - easy for us to read / write
    - easy for programs to process and generate
    - written in plaintext
- has 2 structures

dictionaries

- data structure that has key-value pairs
- surrounded by curly bracks

```
{
      "key1":"value1",
      "key2":"value2"
      ...
}
```

lists

- collection of items
- sq bracks

```
["item1","item2"...]
```

application prog interface

- specifies how software components should interact
- you are given a KEY and ID that must be in every API req
- like a a real-time weather api

roadblocks when scraping

identification

- some sites need id → solu to set your user agent to more common browsers

cookies

- sites may require users to set cookies → use the session class of the req's library

login

- data locked behind login → you can simulate logins or set your login cookie

excessive req

- cant ask a gaziilion things at once, so make sure limit is set and have timeouts

soup

useful methods

1. extracting first para tag with class = “footer-text”

```
.find("p", class="footer-text")
```

2. extract all links from page

```
.find_all("a")
```

3. returns a dictionary of all attributes of this tag

```
tag.attrs
```

4. returns all text in this tag

```
tag.text
```

5. return list of all children elements of this tag

```
tag.contents
```

parser libs

- supports python’s html parser and alot of 3rd party ones
  - lxml parser
  - html5lib-parser - parses html the way a browser does

Parser	Typical usage	Advantages	Disadvantages
Python's html.parser	BeautifulSoup(markup, "html.parser")	• Batteries included • Decent speed • Lenient (As of Python 3.2)	• Not as fast as lxml, less lenient than html5lib.
lxml's HTML parser	BeautifulSoup(markup, "lxml")	• Very fast • Lenient	• External C dependency
lxml's XML parser	BeautifulSoup(markup, "lxml-xml") BeautifulSoup(markup, "xml")	• Very fast • The only currently supported XML parser	• External C dependency
html5lib	BeautifulSoup(markup, "html5lib")	• Extremely lenient • Parses pages the same way a web browser does • Creates valid HTML5	• Very slow • External Python dependency

scraping part

requests lib

- the req lib will make a GET req to a server which will download the html contents of the given webpage

```
import requests

page = requests.get("https://kr1s7on.github.io/KRYJ-Hotel/facilities.html")

print(page.text)
```

```
=OUTPUT=

<!DOCTYPE html>
<html lang="en">

<head>
  <!-- === Author Information ==== -->
  <!-- Code by Khng Yu En, 244705P -->
  <!-- ============================ -->
  <meta charset="UTF-8" />
  <!-- To set the Favicon of the website. -->
...
```

using soup

- parse then extract

- and ++prettify

```
from bs4 import BeautifulSoup
import requests

page = requests.get("https://kr1s7on.github.io/KRYJ-Hotel/facilities.html")

# Parse the HTML content
soup = BeautifulSoup(page.content, 'html.parser')
# Find all the links in the page
links = soup.find_all('a')
# Print the links
for link in links:
    print(link.get('href')) # Print the href attribute of the link
    print(link.text) # Print the text of the link
    print(link) # Print the link itself
```