
Web scraping

PID_00256968

Laia Subirats Maté
Mireia Calvo González

Temps mínim de dedicació recomanat: 5 hores



**Laia Subirats Maté**

Enginyera de Telecomunicacions per la Universitat Pompeu Fabra (2008), màster en Telemàtica per la Universitat Politècnica de Catalunya (2009) i doctora en Informàtica per la Universitat Autònoma de Barcelona (2015). Des de 2009, treballa com a investigadora a Eurecat (Centre Tecnològic de Catalunya) aplicant la ciència de dades a diferents camps com ara la salut, el medi ambient o l'educació. Des de 2016, col·labora amb la UOC com a docent en el màster de Data Science i en el grau d'Informàtica. És especialista en intel·ligència artificial, ciència de dades, salut digital i representació del coneixement.

**Mireia Calvo González**

Enginyera de Telecomunicacions per la Universitat Politècnica de Catalunya (2011), màster en Enginyeria Biomèdica per la Universitat de Barcelona i la Universitat Politècnica de Catalunya (2014) i doctora en Processament de Senyals i Telecomunicacions per la Universitat de Rennes 1 i en Enginyeria Biomèdica per la Universitat Politècnica de Catalunya (2017). Des de 2012, ha treballat com a investigadora en diferents entorns acadèmics, clínics i industrials, aplicant el processament de dades a l'estudi de diferents malalties cardiaques i respiratòries. Des de 2017, col·labora amb la UOC com a docent en el màster de Data Science.

La revisió d'aquest recurs d'aprenentatge UOC ha estat coordinada per la professora: Isabel Guitart Hormigo (2019)

Índex

| | |
|---|----|
| Introducció..... | 5 |
| Objectius..... | 8 |
| 1. Per què i com fer web scraping?..... | 9 |
| 1.1. Per què fem web scraping? | 9 |
| 1.2. Com fem web scraping? | 12 |
| 1.2.1. Avaluació inicial | 12 |
| 1.2.2. Principals reptes del web scraping..... | 18 |
| 2. Primers passos per a fer web scraping..... | 20 |
| 2.1. Funcionament del navegador web | 20 |
| 2.1.1. Enviament de peticions HTTP | 20 |
| 2.1.2. Enviament de respistes HTTP | 21 |
| 2.1.3. Conversió d'HTML en estructura imbricada | 23 |
| 2.2. Descàrrega de la pàgina web | 23 |
| 2.3. Tipus d'objectes | 25 |
| 2.3.1. Tag | 25 |
| 2.3.2. NavigableString | 26 |
| 2.3.3. BeautifulSoup | 26 |
| 2.3.4. Comment | 27 |
| 2.4. Navegar per l'estructura imbricada | 27 |
| 2.4.1. Anàlisi vertical | 29 |
| 2.4.2. Anàlisi horitzontal | 31 |
| 2.5. Funcions principals | 32 |
| 3. Web scraping de contingut gràfic i audiovisual..... | 35 |
| 4. Emmagatzematge i compartició de dades..... | 37 |
| 4.1. Creació d'un arxiu de dades CSV | 37 |
| 4.2. Creació d'un arxiu de dades JSON | 38 |
| 4.3. Creació d'una API | 41 |
| 4.4. Repositoris de dades | 42 |
| 5. Prevenció del web scraping..... | 45 |
| 6. Resolució d'obstacles en web scraping..... | 47 |
| 6.1. Modificació del <i>user agent</i> i altres capçaleres HTTP | 47 |
| 6.2. Gestió d'inici de sessió (<i>logins</i>) i de galetes (<i>cookies</i>) de sessió | 48 |
| 6.3. Respecte a l'arxiu robots.txt | 48 |
| 6.4. Espaiat de peticions HTTP | 49 |

| | |
|--|-----------|
| 6.5. Ús de múltiples adreces IP | 49 |
| 6.6. Configuració de temporitzacions (<i>timeouts</i>) i altres excepcions | 50 |
| 6.7. Evitar els paranyys d'aranya | 50 |
| 7. Aspectes legals..... | 52 |
| 8. Millors pràctiques i consells..... | 55 |
| 9. Exemples de <i>web scraping</i> i casos d'èxit..... | 57 |
| Resum..... | 60 |
| Exercicis d'autoavaluació..... | 61 |
| Solucionari..... | 62 |
| Glossari..... | 63 |
| Bibliografia..... | 66 |

Introducció

Internet és actualment el repositori de dades més gran, accessibles en la seva major part gratuïtament, que mai s'hagi recopilat. Amb l'aparició de la web 2.0, la filosofia de la qual es basa en la interoperabilitat i la col·laboració en xarxa, els usuaris van passar a formar part activa d'aquesta xarxa, no solament utilitzant internet com una eina de cerca d'informació, sinó també com un mitjà per a comunicar-se i generar contingut i coneixement.

Així, gràcies a una gran quantitat d'iniciatives que promouen la compartició de dades de valor, generades tant en entorns públics com en privats, la World Wide Web s'ha convertit en una font inesgotable d'informació. En el context de la recerca, per exemple, les principals revistes científiques d'accés obert sugereixen, i en molts casos exigeixen, la compartició de les dades utilitzades en els estudis que s'hi publiquen. D'aquesta manera, es promou la inclusió de coneixement d'alta qualitat a internet, cosa que estimula la recerca col·laborativa i, per tant, fomenta el progrés cap a la cerca de solucions als problemes del moment.

Encara que en alguns casos és possible recuperar informació de manera estructurada, en múltiples formats com ara Comma Separated Values (CSV), JavaScript Object Notation (JSON), Extensive Markup Language (XML), Resource Description Framework (RDF), Excel Microsoft Office Open XML Format Spreadsheet (XLSX) o Extensible Stylesheet Language (XSL), mitjançant interfícies de programació d'aplicacions o API, la major part del coneixement a internet es troba integrat en l'estructura i estil de les diferents pàgines web. És en aquests casos que l'extracció d'informació pot esdevenir una tasca complexa i tediosa si no es manegen adequadament les eines de programari actualment disponibles que permeten simplificar i automatitzar el procés.

El **web scraping**, que, traduït literalment de l'anglès, seria rasclar la web, permet obtenir la informació útil per a un projecte de dades que es troba disponible a internet.

Per exemple, quan necessitem analitzar la competència per a definir estratègies en el nostre negoci, pot ser interessant recuperar tota la informació relacionada amb productes o preus que es trobi a la seva pàgina web. Així mateix, podem recuperar dades per a fer un estudi que permeti millorar el nostre servei o producte. Per exemple, a la pàgina web d'un hospital on es registri la llista d'espera a urgències en temps real, recuperar aquesta informació periòdicament ens pot ajudar a detectar les hores més concorregudes que necessiten la incorporació de personal sanitari de reforç. És per això que, amb l'increment

constant de la informació disponible a internet, el *web scraping* s'ha convertit en una eina amb un potencial incalculable en el domini de la ciència de dades i, més concretament, en les etapes d'extracció d'informació útil.

No obstant això, es tracta d'una tècnica que presenta diverses complexitats. Les pàgines web que pretenguin evitar el *web scraping* podran aplicar alguns mètodes que dificultin significativament l'extracció d'informació. A més, abans de recopilar dades d'una pàgina web concreta, serà important conèixer les implicacions legals, principalment quan les dades obtingudes es pretenguin publicar posteriorment a internet.

Així mateix, l'heterogeneïtat del contingut que podem trobar en qualsevol pàgina web (text, taules, imatges, vídeos, mapes, etc.) dificultarà els processos d'automatització. I això serà solament el principi, ja que després de la recollida de bases de dades que continguin informació d'interès serà necessari guardar, analitzar i mostrar els resultats obtinguts, de manera que proporcionin nou coneixement i, per tant, valor afegit.

Prèviament a l'aparició del *web scraping*, les dades havien de recollir-se manualment de diferents fonts, de manera poc eficient, no reproduïble i propensa a errors. No obstant això, la ciència de dades ha anat incorporant cada vegada més processos automatitzats per a la recopilació i publicació d'informació en línia, amb l'ús d'eines de programari expandides principalment per a les etapes d'anàlisi, com Python, encara que també es pot fer *web scraping* en R, com detallen Munzert i altres (2014), amb l'objectiu d'estendre aquestes eines a les fases prèvies a la mineria de dades.

Així, per a abordar els aspectes fonamentals del *web scraping*, aquest material didàctic es divideix en vuit seccions principals. L'apartat 1 inclou algunes reflexions sobre per què i com s'ha d'aplicar aquesta tècnica. A continuació, es detallen els primers passos que cal fer quan s'aplica *web scraping*, utilitzant el llençatge de programació Python. A l'apartat 3, s'aborda l'extracció d'informació de contingut audiovisual, i en la següent secció es revisen els formats estandarditzats més comunament utilitzats en l'emmagatzematge de dades generades mitjançant *web scraping*, i també els principals repositoris públics en els quals es poden compartir aquestes dades.

En els apartats 5 i 6, es revisen diferents mesures implementades per a dificultar les tasques de *web scraping*, a més d'alguns mètodes que permeten resoldre aquests obstacles. A continuació, s'enllaren els principals aspectes legals relacionats amb l'extracció de dades procedents d'internet; en el següent apartat es fa referència a una sèrie de millors pràctiques i consells que permeten fer un bon ús del *web scraping*. Així mateix, amb l'objectiu de ressaltar els potencials beneficis de l'ús d'aquesta tècnica, s'esmenten alguns exemples i casos d'èxit en els quals el *web scraping* ha permès extreure informació de gran interès.

Finalment, després d'un breu resum dels continguts més rellevants d'aquest material didàctic, es proposen alguns exercicis d'autoavaluació, i les seves solucions, amb els quals el lector pot revisar l'assimilació dels principals conceptes que aquí es presenten.

Aquest material didàctic s'acompanya d'un repositori Github en què s'inclou el codi d'alguns dels exemples proporcionats. D'aquesta manera, el lector pot descarregar, provar i modificar directament cadascun d'aquests exemples, sense la necessitat de copiar-los manualment.

Enllaç d'interès

Podeu accedir al repositori Github en aquest enllaç:
<https://github.com/datalife-cycleuoc>.

Objectius

En aquest material didàctic es proporcionen les eines fonamentals que permeten assimilar els següents objectius:

- 1.** Conèixer el significat i els potencials beneficis del *web scraping*.
- 2.** Ser capaç d'avaluar la dificultat de fer *web scraping* en un lloc web determinat.
- 3.** Ser capaç de fer *web scraping* simple, utilitzant Python.
- 4.** Ser capaç d'extreure contingut audiovisual d'un lloc web.
- 5.** Ser capaç d'emmagatzemar les dades obtingudes d'internet en un format interoperable.
- 6.** Conèixer els principals repositoris de bases de dades creades a partir d'informació extreta mitjançant *web scraping*.
- 7.** Ser capaç de buscar bases de dades disponibles per a un domini d'aplicació determinat.
- 8.** Ser capaç de solucionar els principals obstacles implementats per a evitar el *web scraping*.
- 9.** Conèixer els principals aspectes legals relacionats amb el *web scraping*.
- 10.** Conèixer diferents casos d'èxit o usos pràctics del *web scraping*.

1. Per què i com fer *web scraping*?

A continuació, abordarem el perquè i el com fem *web scraping*.

1.1. Per què fem *web scraping*?

El primer que ens preguntem quan ens trobem amb el concepte de *web scraping*, és: per què en fem?

Quan naveguem per internet, sovint trobem contingut del nostre interès que ens agradarà recuperar. Així, ens pot interessar recopilar, emmagatzemar i analitzar:

- Una llista de crítiques d'un lloc web sobre llibres, sèries o pel·lícules, amb l'objectiu de crear un motor de recomanació, o construir un model predictiu que detecti les crítiques falses.
- Característiques addicionals que enriqueixin, amb informació disponible en línia, una base de dades determinada. Per exemple, es pot afegir informació meteorològica en un conjunt de dades dissenyat per a predir la venda de refrescos.
- Notícies, de manera periòdica, per a conèixer les últimes tendències sobre un tema d'interès concret.

Algunes pàgines web ofereixen la possibilitat d'accendir i descarregar informació estructuradament, amb les seves interfície de programació d'aplicacions o API (*application programming interface*). Twitter, Facebook, LinkedIn i Google, per exemple, proporcionen aquest tipus d'eines. No obstant això, malgrat que el més convenient és utilitzar aquestes API quan sigui possible, cal dir que no sempre poden descarregar tota la informació disponible a internet.

Quan el propietari d'una pàgina web no posa a la disposició dels seus usuaris eines de programari que possibilitin la descàrrega de dades, ja sigui per voluntat pròpia o per falta de recursos, el *web scraping* apareix com una alternativa per a l'extracció d'informació. Així mateix, quan es descarreguen dades de diferents pàgines web que disposen d'API que no permet la integració cohesiva de les dades, o quan el volum i velocitat requerits per a la descàrrega, o els tipus de dades i formats que proporciona l'API són insuficients per al nostre propòsit, l'extracció de dades mitjançant *web scraping* es converteix en una necessitat.

Per tant, per regla general, buscarem en primer lloc si existeix una API que ens permeti extreure la informació desitjada. Si el nostre objectiu és, per exemple, recuperar una llista de piulades més recents, aquesta informació pot descarregar-se fàcilment mitjançant l'API de Twitter. Però es poden donar diverses situacions en les quals l'extracció de dades mitjançant *web scraping* pot ser interessant, fins i tot quan ja hi ha una API per a fer-ho:

- Quan existeix una API, però aquesta no és gratuïta, mentre que l'accés a la pàgina web sí que ho és.
- Quan l'API limita el nombre d'accisos per segon, per dia, etc.
- Quan l'API no permet recuperar tota la informació d'interès que es mostra a la pàgina web.

Així, en un món on, segons Mikko Hypponen, «les dades són el nou petroli», el *web scraping* té un paper rellevant perquè obre la porta a dades pràcticament il·limitades, que poden ser de gran utilitat en qualsevol disciplina. L'accés a informació disponible a internet ha obert noves fronteres per a la creació de coneixement, cosa que permet fer pronòstics de mercat en qualsevol sector econòmic com ara l'oci, la restauració, l'automoció, etc.; millorar el diagnòstic mèdic a partir d'informació procedent de fòrums de salut; i fins i tot ha revolucionat el mercat de l'art. El 2006, Jonathan Harris i Sep Kamvar van llançar el projecte «We Feel Fine» (ens sentim bé), durant el qual van crear, a partir de la informació compartida en una extensa gamma de blogs, una base de dades formada per frases començant per «I feel/I am feeling» (em sento). Aquest experiment va donar lloc a una visualització de la informació que es va fer molt popular, en descriure com el món se sentia dia rere dia, minut rere minut. Així, independentment del camp d'interès, el *web scraping* té un potencial incalculable que, adequadament aplicat, pot augmentar l'eficiència de qualsevol recerca o negoci.

A més, l'ús del *web scraping* aporta el valor afegit de ser fàcilment automatitzable. Quan detectem informació útil per al nostre projecte que es troba disponible a internet, la recollida automàtica, o semiautomàtica, pot ser altament recomanable si:

- Planegem repetir la tasca en el futur, per exemple, per a mantenir actualitzada la base de dades.
- Ens interessa que altres persones siguin capaces de replicar el nostre procés de recollida de dades.
- Treballem sovint amb dades l'origen de les quals es troba a internet.
- L'extracció de les dades d'interès presenta certa complexitat.

Bibliografia recomanada

CEBIT d!talk, 12.06.2018, Keynote «Data Is The New Oil - The Internet revolution already started years ago and it isn't over yet», Mikko Hyppönen, Chief Research Officer, F-Secure. URL: <https://www.youtube.com/watch?v=he0ruduy9jm>.

Enllaç d'interès

Us pot interessar la pàgina web del projecte «We Feel Fine» de Jonathan Harris i Sep Kamvar <http://www.wefelfine.org>.

Així, el *web scraping* consisteix en la construcció d'un agent que permeti descarregar, analitzar i organitzar dades procedents d'internet, de manera automàtica. Gràcies a l'ús d'aquesta tècnica, podem dissenyar un *script* que desenvolupi una sèrie de tasques repetitives amb les quals emmagatzemar informació d'interès de manera estructurada i molt més eficientment que si ho féssim manualment, accelerant el procés i evitant errors produïts en el procés de copiar i enganxar.

En resum, segons Brody (2017), la informació procedent d'internet pot recuperar-se, principalment, mitjançant els següents mètodes:

- **API.** Idealment, es tractaria del millor mètode per a obtenir informació procedent d'una pàgina web, ja que el proveïdor generalment es compromet a oferir les dades en un format estàndard i ben documentat. No obstant això, no és habitual que els propietaris d'aquestes pàgines dediquin recursos a crear API proporcionant dades a tercets, ja que aquesta tasca no sol trobar-se entre les seves prioritats.
- **Web pública.** Encara que el *web scraping* pot ser costós en alguns casos, permet extreure informació dinàmica emmagatzemada a qualsevol pàgina web.
- **Aplicació mòbil.** Atès que extreure informació d'una aplicació mòbil sol ser més complex que fer-ho d'una pàgina web, en els casos en els quals la informació es trobi en ambdues plataformes, es prioritzarà la recuperació de les dades de la pàgina web.
- **RSS (*rich site summary*) o *atom feeds*.** S'utilitzen, principalment, amb l'objectiu de rebre actualitzacions d'informació. Són formats ben definits i s'utilitzen sovint en blogs, serveis de subscripció, resultats de cerques i altres casos en què un conjunt d'informació es pot actualitzar amb freqüència. Encara que no és la millor manera d'analitzar totes les dades contingudes en un lloc web, permeten analitzar la informació més recent que s'hi ha publicat, per la qual cosa pot ser molt recomanable analitzar aquests *feeds* abans de fer *web scraping*.
- **Exportació de dades en arxius.** Algunes pàgines web permeten descarregar la informació directament en formats estructurats com ara CSV, XLSX o un altre tipus de fulls de càlcul. Encara que aquest mètode ofereix l'avantatge que permet exportar dades directament de manera molt senzilla, el principal desavantatge és que aquests arxius contenen informació extreta en un instant de temps concret, per la qual cosa són menys dinàmics que les pàgines web i, per tant, poden estar desactualitzats.

Bibliografia recomanada

H. Brody (2017). *The ultimate guide to web scraping*. LeanPub.

1.2. Com fem web scraping?

Encara que el *web scraping* es pot fer amb diferents llenguatges de programació, en aquest material didàctic ens centrem en l'ús de les llibreries Python Requests i BeautifulSoup, dissenyades per a l'extracció de contingut web.

Existeixen, però, altres eines àmpliament utilitzades, com Scrapy (Kouzis-Loukas, 2016), que no seran tractades en aquest material didàctic.

1.2.1. Avaluació inicial

Independentment del llenguatge utilitzat, qualsevol tipus de *web scraping* ha d'iniciar una fase prèvia centrada en l'avaluació dels següents aspectes:

- 1) l'arxiu *robots.txt*,
- 2) el mapa del lloc web,
- 3) la seva grandària,
- 4) la tecnologia emprada i
- 5) el propietari del lloc web.

1) Arxiu *robots.txt*. És important analitzar el contingut de *robots.txt*, ja que és en aquest arxiu on la major part de pàgines web indiquen les restriccions que cal tenir en compte quan es pretén rastrejar-les. Encara que aquestes restriccions són només un suggeriment i mai una obligació, és recomanable tenir-les en compte, principalment amb l'objectiu de reduir les possibilitats de ser bloquejats.

A continuació, es mostren alguns exemples de restriccions o permisos definits en un arxiu *robots.txt*.

a) Exclusió de tres directoris

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /tmp/
Disallow: /~joe/
```

b) Exclusió de tots els robots

```
User-agent: *
Disallow: /
```

c) Permís d'accés complet a tots els robots

```
User-agent: *
Disallow:
```

Bibliografia recomanada

D. Kouzis-Loukas (2016). *Learning Scrapy*. Packt Publishing.

Bibliografia recomanada

R. Lawson (2015). *Web Scraping with Python*. Packt Publishing Ltd.

Enllaç d'interès

A l'enllaç següent es pot trobar més informació sobre aquest tipus d'arxius: <http://www.robotstxt.org>.

d) Permís d'accés a un sol robot

```
User-agent: Google
Disallow:
User-agent: *
Disallow: /
```

e) Exclusió de pàgines concretes

```
User-agent: *
Disallow: /~joe/junk.html
Disallow: /~joe/foo.html
Disallow: /~joe/bar.html
```

2) Mapa del lloc web. Examinar el mapa del lloc web (*sitemap*) ens ajudarà a localitzar el contingut actualitzat sense la necessitat de rastrejar cadascuna de les pàgines que el componen. El mapa d'un lloc web es descriu mitjançant el format Simplemaps XML i es compon, principalment, de les següents etiquetes:

- Etiqueta d'obertura `<urlset>`, dins de la qual s'ha d'especificar l'espai de nom (estàndard de protocol).
- Cada URL s'ha d'especificar entre les etiquetes `<url>` i `</url>`, com una etiqueta XML principal.
- Dins de cada etiqueta primària `<url>`, una entrada secundària `<loc>` especifica l'adreça URL.
- Etiqueta de tancament `</ urlset>`.

A continuació es mostra un exemple de *sitemap* en què se'n pot observar l'estructura.

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url>
<loc>http://www.example.com</loc>
<lastmod>2005-01-01</lastmod>
<changefreq>monthly</changefreq>
<priority>0.8</priority>
</url>
</urlset>
```

El principal avantatge dels *sitemaps* és que permeten als motors de cerca, com Google o Bing, rastrejar més fàcilment el lloc web. Encara que aquests cercadors solen indexar correctament qualsevol pàgina web petita o mitjana adequadament dissenyada, rastrejar els llocs més grans comportarà certes com-

Enllaç d'interès

El format Simplemaps XML es descriu detalladament en l'enllaç següent: <https://www.sitemaps.org/protocol.html>.

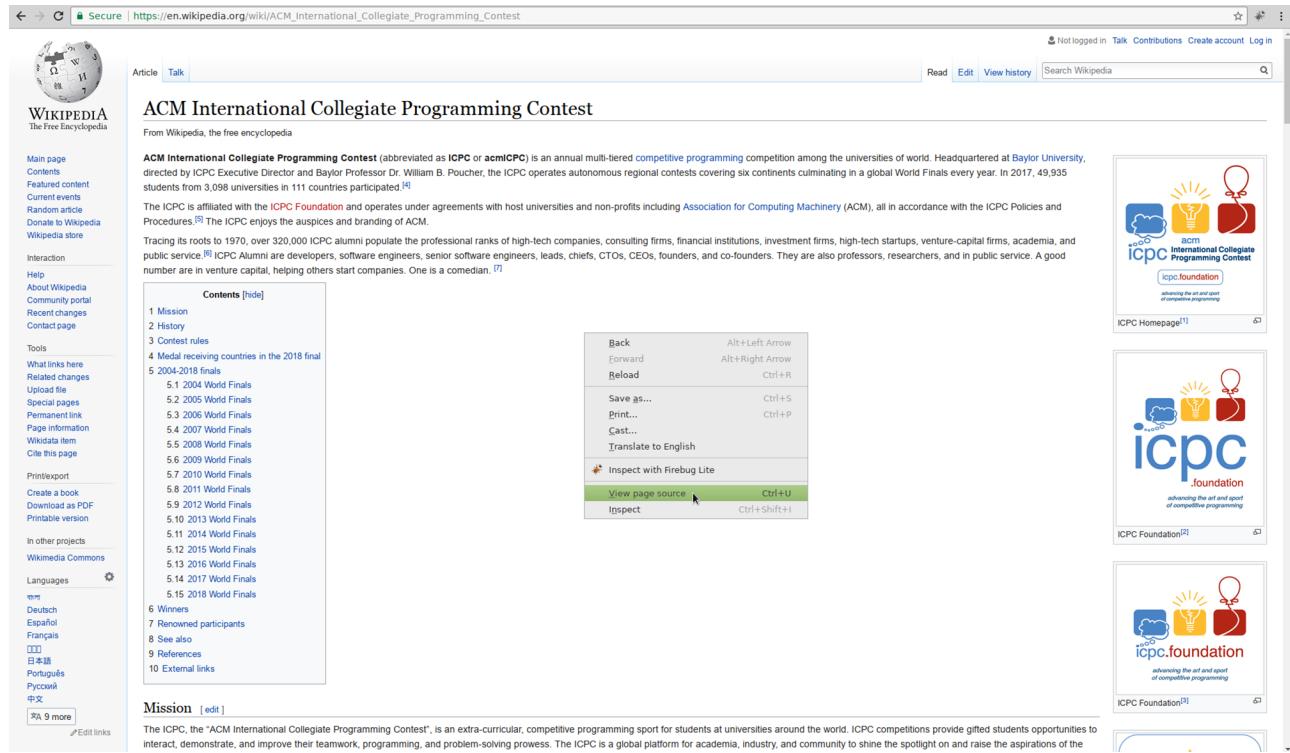
Enllaç d'interès

Per a més informació sobre XML-Sitemaps, consulteu l'enllaç següent: <https://www.xml-sitemaps.com/>.

plexitats, principalment quan s'actualitzin sovint. Així mateix, el *sitemap* permet als usuaris navegar més còmodament pel lloc. Per això, existeixen alguns generadors automàtics de *sitemaps*, com XML-Sitemaps.

D'altra banda, inspeccionar l'estructura d'una pàgina web també pot ser molt útil. Amb aquest objectiu es pot clicar el botó dret del ratolí i posteriorment seleccionar l'opció «View page source», tal com es mostra a la figura 1.

Figura 1. Accés al codi font de la pàgina web



Font: Viquipèdia

Així mateix, per a inspeccionar més fàcilment el contingut d'una pàgina web, es pot afegir l'extensió Firebug Lite al navegador. En la figura 2 es mostra un exemple en què es pot observar el contingut d'una pàgina procedent de la plataforma Viquipèdia.

Segons Acodemy (2015), encara que Viquipèdia és una extensa font d'informació, el seu format no sempre facilita l'extracció de dades, per la qual cosa analitzar la seva estructura abans de fer *web scraping* sol ser molt recomanable.

Bibliografia recomanada

Acodemy (2015). *Learn Web Scraping With Python in a Day: The Ultimate Crash Course to Learning the Basics of Web Scraping With Python in No Time*. CreateSpace Independent Publishing Platform.

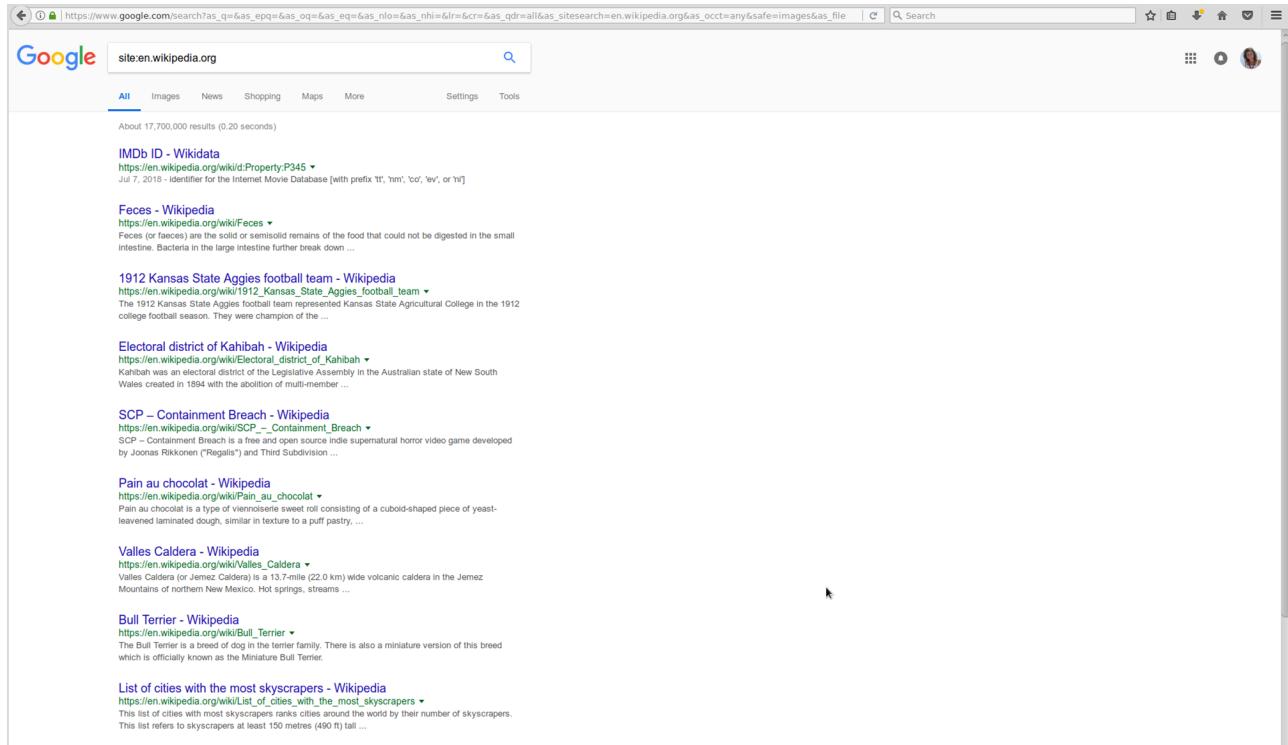
Figura 2. Anàlisi mitjançant Firebug Lite de la pàgina web

The screenshot shows a browser window displaying the ACM International Collegiate Programming Contest page on Wikipedia. The Firebug Lite extension is active, providing a detailed analysis of the page's structure and styling. A tooltip from the extension highlights the 'ICPC' logo, showing its DOM node details.

Font: Viquipèdia

3) Grandària. L'estimació de la grandària d'una pàgina web també afectarà la manera de fer el rastreig. Quan el lloc estiguí format només per un centenar de pàgines, l'eficiència no serà important; en canvi, quan contingui més d'un milió d'adreses, l'ús de descàrregues concurrents, en lloc de seqüencials, serà rellevant. Una manera ràpida de verificar la grandària d'una pàgina web es fa amb la cerca avançada a Google, amb la paraula clau `site`, del lloc web d'interès. En la figura 3 es mostra el resultat obtingut d'analitzar la grandària de Viquipèdia, al voltant de 17.700.000 enllaços.

Figura 3. Cerca avançada del lloc Viquipèdia a Google Advanced Search.



4) Tecnologia. De la mateixa manera, la tecnologia utilitzada en el disseny del lloc web condicionarà el tipus de *web scraping* aplicat. Aquesta es pot analitzar després d'instal·lar l'eina `builtwith`, mitjançant la instrucció `pip3 install builtwith` (o `pip install builtwith` a Python 2), i després executar `builtwith.builtwith`, en què la funció `builtwith` crida el lloc web que es pretén inspeccionar.

A continuació, es mostra el resultat obtingut per a aquest exemple:

```
{'blogs': ['PHP', 'WordPress'],
'cms': ['WordPress'],
'ecommerce': ['WooCommerce'],
'font-scripts': ['Google Font API'],
'programming-languages': ['PHP'],
'web-servers': ['Nginx']}}
```

5) Propietari. Finalment, conèixer el propietari de la pàgina web que pretenem rastrejar pot ser interessant quan, per exemple, aquest sigui conegit per bloquejar els processos de *web scraping*. En aquest cas, amb l'objectiu d'evitar ser bloquejats, ajustarem la descàrrega utilitzant taxes més conservadores.

Així, per a conèixer el propietari de la pàgina web d'interès, farem les següents accions:

```
pip3 install python-whois
import whois
```

```
print(whois.whois('https://www.wordpress.com'))
```

On la funció `whois` crida el lloc web el propietari del qual es pretén conèixer; de manera que, per al mateix exemple de WordPress, el resultat obtingut es mostra a continuació:

```
# {
  # "registrar": "MarkMonitor, Inc.",
  # "city": null,
  # "expiration_date": [
    # "2020-03-03 12:13:23",
    # "2020-03-03 04:13:23-08:00"
  ],
  # "domain_name": [
    # "WORDPRESS.COM",
    # "wordpress.com"
  ],
  # "dnssec": "unsigned",
  # "name": null,
  # "state": "CA",
  # "status": [
    # "clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited",
    # "clientTransferProhibited https://icann.org/epp#clientTransferProhibited",
    # "clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited",
    # "serverDeleteProhibited https://icann.org/epp#serverDeleteProhibited",
    # "serverTransferProhibited https://icann.org/epp#serverTransferProhibited",
    # "serverUpdateProhibited https://icann.org/epp#serverUpdateProhibited",
    # "clientUpdateProhibited (https://www.icann.org/epp#clientUpdateProhibited)",
    # "clientTransferProhibited (https://www.icann.org/epp#clientTransferProhibited)",
    # "clientDeleteProhibited (https://www.icann.org/epp#clientDeleteProhibited)",
    # "serverUpdateProhibited (https://www.icann.org/epp#serverUpdateProhibited)",
    # "serverTransferProhibited (https://www.icann.org/epp#serverTransferProhibited)",
    # "serverDeleteProhibited (https://www.icann.org/epp#serverDeleteProhibited)"
  ],
  # "org": "Automattic, Inc.",
  # "whois_server": "whois.markmonitor.com",
  # "country": "US",
  # "emails": [
    # "abusecomplaints@markmonitor.com",
    # "whoisrelay@markmonitor.com"
  ],
  # "zipcode": null,
  # "creation_date": [
    # "2000-03-03 12:13:23",
    # "2000-03-03 04:13:23-08:00"
  ],
  # "name_servers": [
```

```

    # "NS1.WORDPRESS.COM",
    # "NS2.WORDPRESS.COM",
    # "NS3.WORDPRESS.COM",
    # "NS4.WORDPRESS.COM",
    # "ns2.wordpress.com",
    # "ns3.wordpress.com",
    # "ns4.wordpress.com",
    # "ns1.wordpress.com"
  ],
  # "updated_date": [
    # "2017-01-12 22:53:10",
    # "2017-01-13 14:26:51-08:00"
  ],
  # "referral_url": null,
  # "address": null
}

```

En l'apartat «Resolució d'obstacles en *web scraping*» es pot trobar més informació sobre com resoldre els principals obstacles en *web scraping*.

1.2.2. Principals reptes del *web scraping*

Encara que el *web scraping* es presenta com una eina amb un enorme potencial en permetre l'accés a dades pràcticament il·limitades, com comenta O. Bosch (2017) en un document oficial de la Comissió Europea, comporta importants reptes que hem de tenir presents a l'hora de rastrejar dades de qualsevol pàgina web.

D'una banda, serà rellevant determinar quines dades es volen extreure en fer *web scraping*. Així, encara que diversos llocs web oferiran informació del nostre interès, haurem d'analitzar quins d'aquests llocs són els més adequats segons les nostres necessitats. Així mateix, haurem d'identificar les fonts d'informació que tinguin l'última versió de les dades. També s'ha d'analitzar la qualitat d'aquestes, punt que es tractarà a l'apartat 8, en què s'enumeren les variables que cal tenir en compte a l'hora de determinar la qualitat de la informació. En ocasions, serà necessari escollir entre el propietari de les dades i un agregador de continguts, per la qual cosa serà interessant explorar els fluxos entre llocs web.

D'altra banda, és important recordar que internet és dinàmic. Encara que cada lloc web té una estructura predeterminada, el seu contingut canvia constantment (en fer ús, per exemple, del *scroll* infinit). Per tant, els *web scrapers* que implementem hauran de ser tan robustos com sigui possible, tenint en compte que internet és volàtil, i, de la mateixa manera que algunes pàgines poden deixar d'existir amb el pas del temps, n'apareixeran altres de noves. Així, el més recomanable serà controlar constantment la informació d'interès.

Bibliografia recomanada

O. Bosch (2017). *An introduction to web scraping, IT and legal aspects*. <<https://bit.ly/2pmuykc>>

Un altre punt fonamental que cal tenir en compte és el dels aspectes legals relacionats amb els propietaris de les dades que pretenem rastrejar, tenint en compte que la legislació pot ser específica per a cada país.

Finalment, el fet d'organitzar les dades extretes mitjançant *web scraping* de manera estàndard pot ser un altre punt diferenciador.

2. Primers passos per a fer *web scraping*

Una idea clau a l'hora de fer *web scraping* és que, amb l'objectiu de planificar i implementar una descàrrega òptima de la informació continguda en una pàgina web determinada, és necessari entendre com funciona el navegador mitjançant el qual s'accedeix al contingut d'aquesta pàgina.

Per això, en aquest apartat es tracta en primer lloc el funcionament del navegador web. Posteriorment, es detalla com s'ha de descarregar la pàgina web que es desitja rastrejar, i també l'estructura imbricada que s'obté d'aquest procés. Finalment, amb l'objectiu d'identificar la informació d'interès, es presenten diverses operacions útils a l'hora de navegar per aquesta estructura imbricada, disponibles mitjançant l'ús de la llibreria BeautifulSoup.

2.1. Funcionament del navegador web

Naveguem per la xarxa cada dia, cada vegada que accedim al correu electrònic i a les xarxes socials, consultem les últimes notícies, comprem alguna cosa a internet, busquem informació a la Viquipèdia o tutorials a YouTube sobre un tema d'interès, etc. Però com funciona realment el procés amb el qual s'accedeix a una pàgina web des del nostre navegador? Principalment, aquest procés pot resumir-se en tres passos:

- 1) enviament de peticions HTTP,
- 2) recepció de peticions HTTP,
- 3) conversió de pàgina web objectiu en estructura imbricada.

En els següents apartats, s'expliquen breument cadascun d'aquests passos fonamentals a l'hora de fer *web scraping*.

2.1.1. Enviament de peticions HTTP

Com el seu nom indica, el protocol de transferència d'hipertext (HTTP) és un protocol de comunicació que permet la transferència d'informació mitjançant documents de tipus hipertext, és a dir, per mitjà d'internet. Així, quan es vol accedir a una pàgina web dissenyada en llenguatge HTML (llenguatge d'etiquetatge d'hiperext) per mitjà del navegador, es fa una petició HTTP.

El següent fragment de codi mostra una petició HTTP feta amb l'objectiu d'accendir a la pàgina web de Viquipèdia:

```
GET page www.wikipedia.org HTTP/2.0
```

La figura 4 mostra la informació transmesa a les capçaleres d'una petició HTTP, entre les quals es destaquen les més rellevants:

- **Connection.** Especifica el tipus de connexió amb el servidor HTTP. Normalment, com es mostra en l'exemple, el seu valor és `keep-alive`.
- **Accept.** Fa referència al tipus de continguts o fitxers acceptats com a resposta; generalment, `text/html`.
- **User-agent.** Conté informació sobre la petició, això és, sobre el navegador utilitzat, el sistema operatiu, etc.
- **Accept-encoding.** Especifica el tipus de codificacions (*encodings*) admeses (el servidor pot comprimir la resposta).
- **Accept-language.** El servidor indica els idiomes acceptats.
- **Cookie.** Un altre concepte important són les galetes (*cookies*), ja que permeten establir preferències que persisteixen al llarg de diferents pàgines web.

Figura 4. Capçaleres d'una petició HTTP

```
② Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
② Accept-Encoding: gzip, deflate, br
② Accept-Language: ca,en-US;q=0.7,en;q=0.3
② Cache-Control: max-age=0
② Connection: keep-alive
② Cookie: WMF-Last-Access-Global=26-Jul-...; WMF-Last-Access=26-Jul-2018
② Host: www.wikipedia.org
② If-Modified-Since: Mon, 23 Jul 2018 10:07:54 GMT
② If-None-Match: W/"12742-571a7d1a715ba"
② Upgrade-Insecure-Requests: 1
② User-Agent: Mozilla/5.0 (Windows NT 6.1; W...) Gecko/20100101 Firefox/60.0
```

Enllaç d'interès

La següent entrada a Viquipèdia inclou una llista completa de capçaleres HTTP:
<https://bit.ly/2pempij>

2.1.2. Enviament de respostes HTTP

Una vegada feta la petició HTTP per part del navegador, el servidor envia una resposta de tipus HTTP/2.0 200 OK, incloent capçaleres HTTP de resposta, i també un document HTML. La figura 5 mostra un exemple de capçaleres de resposta.

Figura 5. Capçaleres de resposta HTTP

```

? age: 77899
? backend-timing: D=206 t=1532521483616492
? cache-control: s-maxage=86400, must-revalidate, max-age=3600
? content-encoding: gzip
? content-type: text/html
? date: Thu, 26 Jul 2018 10:03:04 GMT
? etag: W/"12742-571a7d1a715ba"
? last-modified: Mon, 23 Jul 2018 10:07:54 GMT
? server: mw1264.eqiad.wmnet
? strict-transport-security: max-age=106384710; includeSubDomains; preload
? vary: Accept-Encoding
? via: 1.1 varnish (Varnish/5.1), 1.1...1), 1.1 varnish (Varnish/5.1)
x-analytics: WMF-Last-Access=26-Jul-2018;WM...ss-Global=26-Jul-2018;https=1
x-cache: cp1065 hit/10, cp3040 hit/1, cp3040 hit/147455
x-cache-status: hit-front
x-client-ip: 84.88.76.3
X-Firefox-Spdy: h2
x-varnish: 548206840 24383145, 516973151 525052522, 403431753 292805460

```

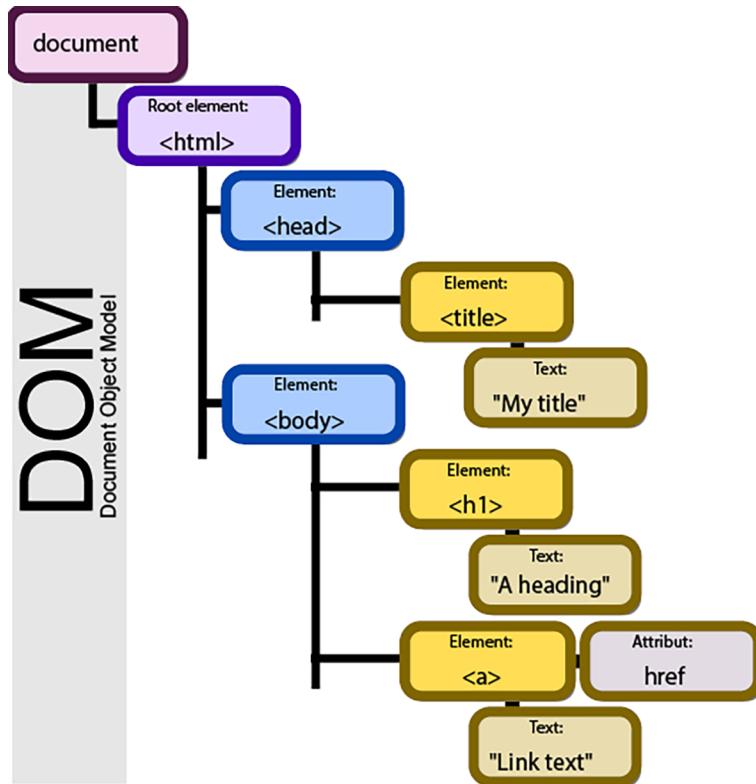
En el cas de la figura 5, el codi d'estat enviat pel servidor va ser un 200 OK, això és, el servidor va enviar una resposta estàndard per a peticions correctes. No obstant això, existeixen molts altres codis d'estat, detallats a Viquipèdia, que poden classificar-se en quatre grans grups:

- **2XX.** Peticions reeixides. Aquesta classe de codi d'estat indica que la petició ha estat rebuda correctament, entesa i acceptada. Un altre exemple d'aquest tipus és el 201 Created, que indica que la petició ha estat completada i ha donat pas a la creació d'un nou recurs.
- **3XX.** Redireccions. En aquest cas, el client ha de fer una acció addicional per a completar la petició. Un exemple d'aquest tipus és el codi 300 Multiple Choices, el qual indica opcions múltiples que el client ha de seleccionar: presentant diferents opcions de format per a la visualització de vídeos, llistant arxius amb diferents extensions, etc.
- **4XX.** Errors del client. La sol·licitud conté una sintaxi incorrecta o no es pot processar. Així, el codi 404 Not Found fa referència a un recurs no oposat i s'utilitza quan el servidor web no és capaç de trobar la pàgina o recurs sol·licitats.
- **5XX.** Errors del servidor, en completar una sol·licitud aparentment vàlida. El codi 500 Internal Server Error és comunament emès per aplicacions incrustades en servidors web que generen contingut dinàmicament; per exemple, aplicacions muntades en Tomcat, quan es troben amb situacions d'error alienes a la naturalesa del servidor web.

2.1.3. Conversió d'HTML en estructura imbricada

Finalment, el navegador analitza la pàgina web objectiu per a construir així una estructura imbricada coneguda com el model d'objecte de document (*document object model*, DOM). Encara que aquest model pot ser molt complex, estableix la jerarquia imbricada de qualsevol lloc web. La figura 6 mostra, esquemàticament, un exemple de DOM.

Figura 6. Model d'objecte de document



Font: Birger Eriksson CC-BY-SA-3.0

2.2. Descàrrega de la pàgina web

En *web scraping*, el primer pas que es fa és la descàrrega del lloc web d'interès. Això es pot fer mitjançant les llibreries Requests i BeautifulSoup.

En primer lloc, s'ha d'instal·lar i importar la llibreria Requests amb les següents instruccions:

```
pip3 install requests
import requests
```

Posteriorment, el mètode `requests.get` permet recuperar la informació corresponent a la resposta de la petició, on `str` fa referència a la pàgina sobre la qual volem fer *web scraping*.

```
page = requests.get(str)
```

Així, `page` serà un objecte amb una sèrie d'atributs, entre els quals destaquen com a més importants aquests:

- `page.status_code`: codi HTTP retornat pel servidor.
- `page.content`: contingut brut de la resposta del servidor.

Per tant, també, després d'instal·lar i importar la llibreria BeautifulSoup, s'ha d'analitzar aquest contingut brut i emmagatzemar el resultat en un nou objecte, en aquest cas anomenat `soup`:

```
pip3 install beautifulsoup4
from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content)
```

A continuació, amb l'objectiu d'obtenir l'estructura imbricada que hem d'analitzar per a identificar la informació d'interès, utilitzarem la funció `prettify`. Així, si mostrem un exemple d'ús d'aquesta funció en un fragment d'*'Alícia al país de les meravelles'*, la funció `print` retorna el següent resultat:

```
print(soup.prettify())
# <html>
# <head>
# <title># The Dormouse's story
# </title>
# </head>
# <body>
# <p class="title">
# <b>
# The Dormouse's story
# </b>
# </p>
# <p class="story">
# Once upon a time there were three little sisters; and their names were
# <a class="sister" href="http://example.com/elsie" id="link1">
# Elsie
# </a>
#
# <a class="sister" href="http://example.com/lacie" id="link2">
# Lacie
# </a>
# and
# <a class="sister" href="http://example.com/tillie" id="link2">
# Tillie
# </a>
# ; and they lived at the bottom of a well.
# </p>
# <p class="story">
```

```
# ...
# </p>
# </body>
# </html>
```

2.3. Tipus d'objectes

En BeautifulSoup existeixen diversos tipus d'objectes. Si tenim en compte que BeautifulSoup transforma qualsevol document HTML en un arbre complex d'objectes, podem dir que existeixen quatre tipus d'objectes fonamentals amb els quals es pot navegar còmodament per l'estruccura imbricada resultants: Tag, NavigableString, BeautifulSoup i Comment.

2.3.1. Tag

Aquest objecte correspon a una etiqueta XML o HTML en el document original. A continuació se'n mostra un exemple:

```
soup = BeautifulSoup('<b class="boldest">Extremely bold</b>')
tag = soup.b
type(tag)
# <class 'bs4.element.Tag'>
```

Cada etiqueta s'associa a un nom, accessible mitjançant .name:

```
tag.name
# u'b'
```

Així mateix, una etiqueta pot contenir un nombre indefinit d'atributs. En l'exemple anterior, l'etiqueta `<b id="boldest">` conté un atribut «id» el valor del qual és «boldest». Es pot accedir a aquests atributs de la següent manera:

```
tag.attrs
# {u'id': 'boldest'}
```

O bé directament:

```
tag['id']
# u'boldest'
```

D'altra banda, els atributs d'una etiqueta es poden afegir, eliminar o modificar:

```
tag['id'] = 'verybold'
tag['another-attribute'] = 1
tag
# <b another-attribute="1" id="verybold"></b>
```

```

del tag['id']
del tag['another-attribute']
tag
# <b></b>

tag['id']
# KeyError: 'id'

print(tag.get('id'))
# None

```

Així mateix, alguns atributs poden contenir més d'un valor, per la qual cosa BeautifulSoup els representa com una llista. És el cas de `class`, encara que altres exemples són `rel`, `rev`, `accept-charset`, `headers`, i `accesskey`.

```

css_soup = BeautifulSoup('<p class="body"></p>')
css_soup.p['class']
# ["body"]

css_soup = BeautifulSoup('<p class="body strikeout"></p>')
css_soup.p['class']
# ["body", "strikeout"]

```

2.3.2. NavigableString

En aquest cas, `NavigableString` correspon a una cadena de caràcters dins d'un `tag`. La següent sentència permet accedir al seu contingut:

```

tag.string
# u'Extremely bold'

type(tag.string)
# <class 'bs4.element.NavigableString'>

```

Encara que aquest tipus d'objectes no es poden modificar, és possible substituir el contingut de la cadena utilitzant la funció `replace_with`:

```

tag.string.replace_with("No longer bold")
tag
# <blockquote>No longer bold</blockquote>

```

2.3.3. BeautifulSoup

Aquest objecte representa el document en el seu conjunt. Generalment es pot tractar com un objecte de tipus `tag`, per la qual cosa suporta l'ús de la majoria d'operacions que permeten navegar per l'estructura imbricada.

```
soup.name
# u'[document]'
```

2.3.4. Comment

Encara que els objectes anteriors cobreixen pràcticament la totalitat de la informació continguda en un document HTML o XML, pot ser necessari accedir a algunes dades d'interès mitjançant l'objecte `Comment`. Es tracta d'un tipus de `NavigableString`, que es mostra amb un format especial.

```
markup = "<b><!--Hey, buddy. Want to buy a used parser?--></b>"
soup = BeautifulSoup(markup)
print(soup.b.prettify())
# <b>
# <!--Hey, buddy. Want to buy a used parser?-->
# </b>

comment = soup.b.string
type(comment)
# <class 'bs4.element.Comment'>
```

2.4. Navegar per l'estructura imbricada

Reprenguem el fragment d'*Alícia al país de les meravelles*, amb l'objectiu d'analitzar algunes instruccions útils a l'hora de navegar pel DOM resultant.

```
print(soup.prettify())
# <html>
# <head>
#   <title># The Dormouse's story
#   </title>
# </head>
# <body>
#   <p class="title">
#     <b>
#       The Dormouse's story
#     </b>
#   </p>
#   <p class="story">
#     Once upon a time there were three little sisters; and their names were
#     <a class="sister" href="http://example.com/elsie" id="link1">
#       Elsie
#     </a>
#     ,
#     <a class="sister" href="http://example.com/lacie" id="link2">
#       Lacie
#     </a>
```

```
# and
# <a class="sister" href="http://example.com/tillie" id="link2">
#   Tillie
# </a>
# ; and they lived at the bottom of a well.
# </p>
# <p class="story">
# ...
# </p>
# </body>
# </html>
```

A continuació, es destaquen les instruccions més àmpliament utilitzades a l'hora de navegar per l'estructura imbricadaa:

1) soup.title

```
# <title>The Dormouse's story</title>
```

2) soup.title.name

```
# u'title'
```

3) soup.title.string

```
# u'The Dormouse's story'
```

4) soup.title.parent.name

```
# u'head'
```

5) soup.p

```
# <p class="title"><b>The Dormouse's story</b></p>
```

6) soup.p['class']

```
# u'title'
```

7) soup.a

```
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
```

8) soup.find_all('a')

```
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#   <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
```

```
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

9) soup.find(id="link3")

```
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

Així mateix, és habitual extreure totes les URL contingudes en un lloc web, associades a les etiquetes de tipus `<a>`, mitjançant la instrucció:

```
for link in soup.find_all('a'):
    print(link.get('href'))
# http://example.com/elsie
# http://example.com/lacie
# http://example.com/tillie
```

Finalment, la següent funció permet extreure la totalitat del text contingut en una pàgina web:

```
print(soup.get_text())
# The Dormouse's story
# Once upon a time there were three little sisters; and their names were
# Elsie,
# Lacie and
# Tillie;
# and they lived at the bottom of a well.
# ...
```

2.4.1. Anàlisi vertical

Les diferents etiquetes poden contenir cadenes de caràcters, i també altres etiquetes. Aquests elements són els que es coneixen com els fills (*children*) de l'etiqueta, i BeautifulSoup proporciona gran quantitat de mètodes que permeten navegar i iterar sobre els fills d'aquesta etiqueta.

El mètode més senzill per a navegar per l'estructura consisteix a utilitzar el nom de l'etiqueta d'interès. Així, per a accedir al `<head>` del document HTML d'interès, utilitzarem la instrucció `.head`:

```
head_tag = soup.head
head_tag
# <head><title>The Dormouse's story</title></head>
```

De la mateixa manera, es pot ampliar la cerca encadenant etiquetes. En el següent exemple es mostra la primera etiqueta `` trobada en el `<body>` del document:

```
soup.body.b
```

```
# <b>The Dormouse's story</b>
```

És important destacar que aquest mètode solament retorna la primera etiqueta amb aquest nom, per la qual cosa si es pretén recuperar totes les etiquetes d'un mateix tipus, és necessari utilitzar la funció `find_all`.

Un altre mètode per a obtenir els fills d'una etiqueta consisteix a utilitzar la funció `.contents`:

```
head_tag.contents
# [<title>The Dormouse's story</title>]

title_tag = head_tag.contents[0]
title_tag
# <title>The Dormouse's story</title>

title_tag.contents
# [u'The Dormouse's story']
```

Així mateix, en lloc d'obtenir els fills com una llista, es poden extreure mitjançant el generador `.children`:

```
for child in title_tag.children:
    print(child)
# The Dormouse's story
```

Tant `.contents` com `.children` només consideren els fills directes d'una etiqueta. Així, en l'exemple anterior, `<head>` només té com a fill directe l'etiqueta `<title>`, però aquesta última té al seu torn un fill, la cadena «`The Dormouse's story`», per la qual cosa aquesta cadena també es pot considerar descendència de l'etiqueta `<head>`. Si es vol recuperar la totalitat de la descendència d'una etiqueta, serà necessari utilitzar el mètode `.descendants`:

```
head_tag.contents
# [<title>The Dormouse's story</title>]

for child in head_tag.descendants:
    print(child)
# <title>The Dormouse's story</title>
# The Dormouse's story
```

De manera anàloga, tota etiqueta i tota cadena, excepte l'objecte `BeautifulSoup`, té un element pare (*parent*). Així, per exemple, es pot accedir a l'etiqueta `<head>` a partir del seu fill `<title>`, mitjançant la instrucció `.parent`:

```
title_tag = soup.title
```

```

title_tag
# <title>The Dormouse's story</title>

title_tag.parent
# <head><title>The Dormouse's story</title></head>

```

Finalment, atès que `.parent` només accedeix a l'element pare directe, la instrucció `.parents` permet iterar sobre tota l'ascendència. El següent exemple mostra com és possible navegar, mitjançant la funció `.parents`, des d'una etiqueta `<a>` continguda en un document fins a la part superior d'aquest document:

```

link = soup.a
link

# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>

for parent in link.parents:
    if parent is None:
        print(parent)
    else:
        print(parent.name)

# p
# body
# html
# [document]
# None

```

2.4.2. Anàlisi horitzontal

Els elements germans (*siblings*) d'un document HTML són els que es troben al mateix nivell, és a dir, que són fills d'una mateixa etiqueta. Així, fixem-nos en el senzill exemple següent:

```

sibling_soup = BeautifulSoup("<a><b>text1</b><c>text2</c></a>")
print(sibling_soup.prettify())

# <html>
# <body>
#   <a>
#     <b>
#       text1
#     </b>
#     <c>
#       text2
#     </c>
#   </a>
# </body>

```

```
# </html>
```

Les etiquetes ` i <c>` es presenten com a germanes, en ser ambdues elements fills d'una etiqueta `<a>`.

En aquest cas, les funcions `.next_sibling` i `.previous_sibling` permeten navegar pels elements d'un lloc web que es troben al mateix nivell:

```
sibling_soup.b.next_sibling
# <c>text2</c>

sibling_soup.c.previous_sibling
# <b>text1</b>
```

De la mateixa manera, `.next_siblings` i `.previous_siblings` permeten iterar sobre els diferents germans d'una etiqueta. Així, representant l'exemple d'*Alícia al país de les meravelles*, a partir d'un element `<a>`, es pot obtenir la resta:

```
for sibling in soup.a.next_siblings:
    print(repr(sibling))
# u',\n'
# <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
# u' and\n'
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
# u'; and they lived at the bottom of a well.'
# None

for sibling in soup.find(id="link3").previous_siblings:
    print(repr(sibling))
# ' and\n'
# <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
# u',\n'
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
# u'Once upon a time there were three little sisters; and their names were\n'
# None
```

2.5. Funcions principals

La següent llista enumera el conjunt de funcions de BeautifulSoup més utilitzades:

- **find_all.** Aquest mètode extreu tots els objectes Tag i NavigableString de l'estructura analitzada que coincideixen amb els criteris donats. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findAll.
- **find.** En aquest cas, la funció només retorna el primer objecte que coincideix amb els criteris donats.
- **find_all_next.** Extreu tots els elements següents a un objecte donat, que compleixen els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findAllNext.
- **find_next.** Identifica el primer element següent a un objecte donat, que compleix els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findNext.
- **find_all_previous.** Extreu tots els elements previs a un objecte donat, que compleixen els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findAllPrevious.
- **find_previous.** Identifica el primer element previ a un objecte donat, que compleix els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findPrevious.
- **find_next_siblings.** Extreu tots els elements germans següents d'un objecte donat, que compleixen els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findNextSiblings.
- **find_next_sibling.** Identifica el primer element germà següent d'un objecte donat, que compleix els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findNextSibling.
- **find_previous_siblings.** Extreu tots els elements germans previs d'un objecte donat, que compleixen els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findPreviousSiblings.
- **find_previous_sibling.** Identifica el primer element germà previ d'un objecte donat, que compleix els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findPreviousSibling.
- **find_parents.** Extreu tots els elements pare d'un objecte donat, que compleixen els criteris específics. Funció equivalent (accessible en versions anteriors de BeautifulSoup): findParents.

Bibliografia recomanada

V. G. Nair (2014). *Getting started with BeautifulSoup*. Packt Publishing Ltd. Open Source Collaborative framework in Python. Disponible a: <https://scrapy.org> (data d'accés: 15 de març del 2018).

Enllaç d'interès

En l'enllaç següent es pot trobar més informació sobre el funcionament de la llibreria BeautifulSoup: <https://bit.ly/2z01cvf>.

- **find_parent.** Identifica el primer element pare d'un objecte donat, que compleix els criteris especificats. Funció equivalent (accessible en versions anteriors de BeautifulSoup): `findParent`.
- **replace_with.** Elimina un element i el substitueix per l'etiqueta o la cadena proporcionades. Funció equivalent (accessible en versions anteriors de BeautifulSoup): `replaceWith`.
- **wrap.** Introdueix un element en l'etiqueta especificada.
- **unwrap.** Reemplaça una etiqueta amb el seu contingut.

3. Web scraping de contingut gràfic i audiovisual

A més del text procedent d'un lloc web, pot ser útil extreure i emmagatzemar certes imatges o un altre contingut audiovisual. Per això, es pot implementar un mètode que guardi aquest contingut predeterminat, donada la seva URL. El següent exemple emmagatzema la imatge amb URL `source_url` a la carpeta Pictures:

```
import requests

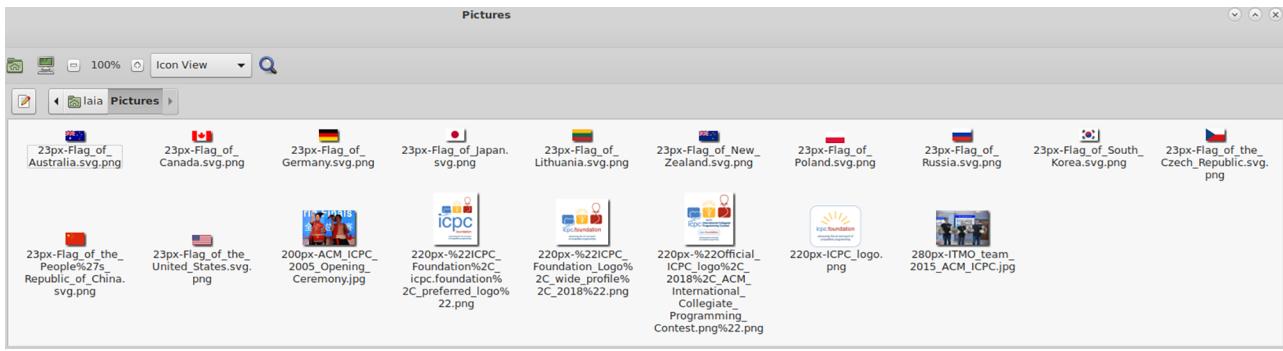
def load_requests(source_url):
    r = requests.get(source_url, stream = True)
    if r.status_code == 200:
        aSplit = source_url.split('/')
        ruta = "/home/user/Pictures/" + aSplit[len(aSplit)-1]
        print(ruta)
        output = open(ruta, "wb")
        for chunk in r:
            output.write(chunk)
        output.close()
```

Per tant, per a fer *web scraping* sobre les imatges d'un lloc web haurem d'obtenir les seves URL per a emmagatzemar-les després en una carpeta, en aquest cas Pictures, mitjançant la funció `load_requests` creada anteriorment.

El següent exemple mostra el resultat d'estreure les imatges contingudes a la pàgina de Viquipèdia destinada a l'ACM International Collegiate Programming, per a emmagatzemar-les posteriorment a la carpeta Pictures mostrada a la figura 7.

```
from bs4 import BeautifulSoup
import requests
url = 'https://en.wikipedia.org/wiki/acm_international_collegiate_programming_contest'
page = requests.get(url)
soup = BeautifulSoup(page.content)
images = []
i = 0
for img in soup.findAll('img'):
    images.append(img.get('src'))
    if ('static' not in images[i]):
        load_requests("https:"+images[i])
    i = i+1
```

Figura 7. Imatges emmagatzemades a la carpeta Pictures



Font: Viquipèdia.

La descàrrega de qualsevol altre tipus de contingut gràfic o audiovisual es faria de manera similar a l'exemple anterior.

Així mateix, existeixen algunes llibreries, com la presentada a Heydt (2018), que permeten obtenir captures de pantalla periòdiques d'una pàgina web. En funció de l'objectiu final del *web scraping* aplicat, aquesta informació emmagatzemada en format gràfic pot ser de gran utilitat per al rastreig de l'evolució d'una pàgina web.

Bibliografia recomanada

M. Heydt (2018). *Python Web Scraping Cookbook: Over 90 proven recipes to get you scraping with Python, microservices, Docker, and AWS*. Packt Publishing.

4. Emmagatzematge i compartició de dades

Les dades obtingudes mitjançant *web scraping* poden emmagatzemar-se i compartir-se amb diferents formats estandarditzats.

En aquest material didàctic, es descriu breument el procés de creació d'arxius en dos dels formats més àmpliament utilitzats: CSV i JSON. No obstant això, existeixen molts altres formats estandarditzats per a l'emmagatzematge de dades, com ara XML o RDF.

Bibliografia recomanada

R. Mitchell (2015). *Web Scraping with Python: Collecting Data from the Modern Web*. O'Reilly.

4.1. Creació d'un arxiu de dades CSV

Després de fer *web scraping*, el format més comunament utilitzat per a emmagatzemar dades de tipus text és el CSV o *comma separated values*.

El següent codi mostra un exemple de creació d'arxiu de tipus CSV:

```
import csv

with open('eggs.csv', 'w', newline='') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=' ',
                           quotechar='|', quoting=csv.QUOTE_MINIMAL)
    spamwriter.writerow(['Spam'] * 5 + ['Baked Beans'])
    spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])
```

Enllaç d'interès

En l'enllaç següent es pot trobar més informació sobre el funcionament de la llibreria CSV: <https://docs.python.org/3.4/library/csv.html>.

Una vegada creat l'arxiu *eggs.csv*, és possible llegir-ne el contingut emprant el següent codi:

```
import csv

with open('eggs.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in spamreader:
        print(', '.join(row))

# Spam, Spam, Spam, Spam, Baked Beans
# Spam, Lovely Spam, Wonderful Spam
```

4.2. Creació d'un arxiu de dades JSON

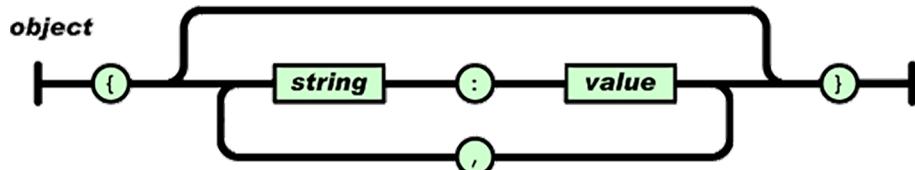
Un altre tipus de dades que pot ser interessant, a l'hora d'emmagatzemar la informació extreta amb *web scraping*, és el format JSON (Javascript Object Notation), ja que es tracta d'un format computacionalment senzill, fàcil de llegir, interpretar i escriure.

El format JSON es basa en dues estructures principals:

- 1) una col·lecció de parells nom/valor (objecte)
- 2) una llista ordenada de valors (*array*)

Així, un objecte, definit com un conjunt de parells nom/valor, inclou entre claus ('{}') cada nom seguit de ':' i el seu valor; separant els diferents parells amb comes. La figura 8 mostra un esquema de la seva sintaxi.

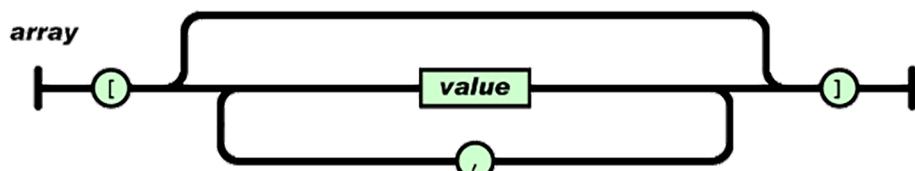
Figura 8. Sintaxi d'un objecte



Font: JSON

D'altra banda, un *array* es defineix entre claudàtors ('[]'); separant els diferents valors mitjançant comes (vegeu la figura 9).

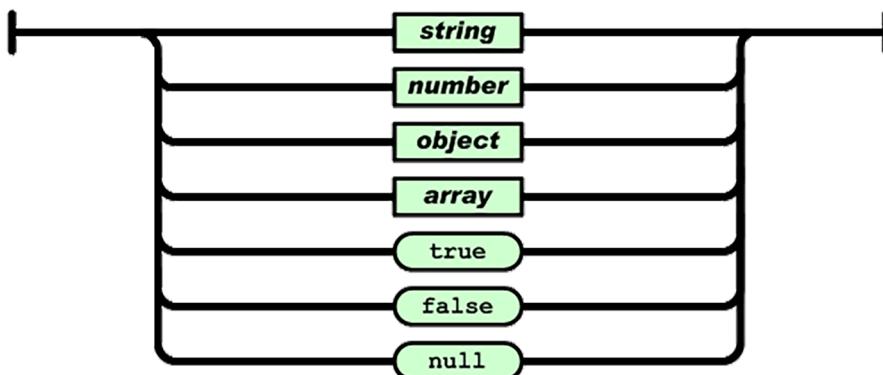
Figura 9. Sintaxi d'un array



Font: JSON

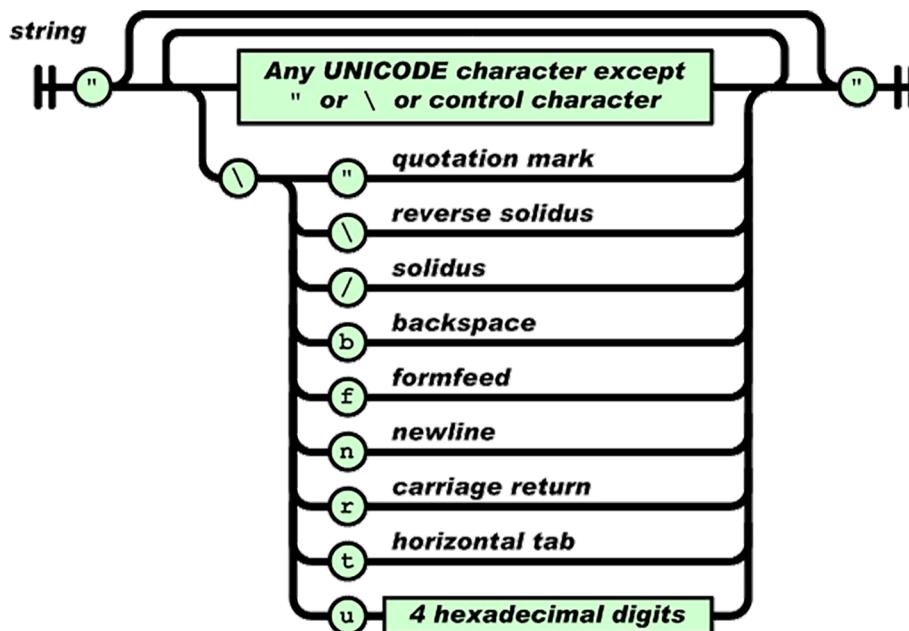
Per la seva banda, cada valor pot ser un *string*, un *number*, un *true*, un *false*, un *null*, un *object* o un *array* (vegeu la figura 10).

Figura 10. Sintaxi d'un valor

value

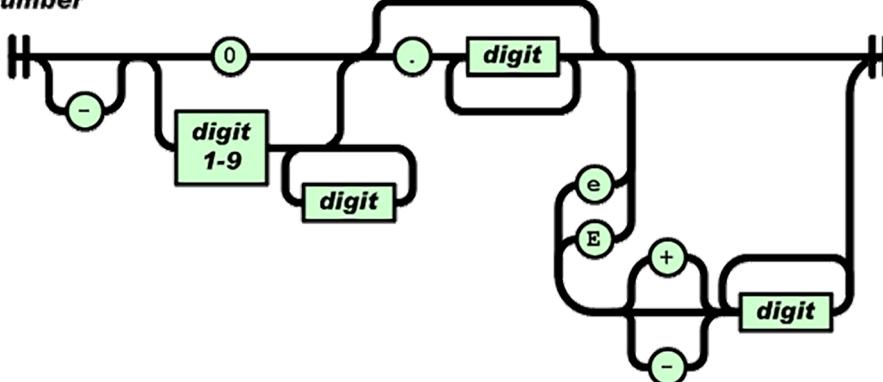
Font: JSON

Finalment, les figures 11 i 12 representen les sintaxis d'un *string* i un nombre, respectivament.

Figura 11. Sintaxi d'una cadena o *string*

Font: JSON

Figura 12. Sintaxi d'un nombre

number

Font: JSON

Per a aplicar aquest tipus de codificació s'utilitza la llibreria JSON de Python.
A continuació, es mostren alguns exemples d'ús:

```
import json

json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)})]
# '['"foo", {"bar": ["baz", null, 1.0, 2]}]'

print(json.dumps({'c': 0, 'b': 0, 'a': 0}, sort_keys=True))
# {"a": 0, "b": 0, "c": 0}

print(json.dumps({'4': 5, '6': 7}, sort_keys=True,
indent=4, separators=(',', ': ')))
# {
#     "4": 5,
#     "6": 7
# }
```

Finalment, el següent exemple mostra el contingut d'un arxiu JSON extret de la pàgina web de l'Ajuntament de Barcelona:

Enllaç d'interès

Podeu consultar l'arxiu en aquest enllaç: <http://opendata-ajuntament.barcelona.cat>.

```
{
  "name" : "Carril_Bici_Construccio_GeoJson",
  "type" : "FeatureCollection",
  "crs" : {
    "type" : "name",
    "properties" : {
      "name" : "EPSG:25831"
    }
  },
  "features" : [
    {
      "type" : "Feature",
      "geometry" : {
        "type" : "LineString",
        "coordinates" : [
          [ 427704.83499489503, 4579666.9013711698 ],
          [ 427709.81394183601, 4579642.00360063 ],
          [ 427716.32587691199, 4579625.1497550504 ],
          [ 427723.98780126096, 4579607.9119127104 ],
          [ 427763.058417933, 4579528.6226368099 ],
          [ 427780.27225003002, 4579497.1649236102 ],
          [ 427781.06223390898, 4579466.1872106604 ],
          [ 427899.71207724401, 4579251.9421635903 ],
          [ 428209.77504851203, 4578673.2964416901 ],
          [ 427704.83499489503, 4579666.9013711698 ]
        ]
      }
    }
  ]
}
```

```
[ 428242.22674152104, 4578647.8856680803 ],
[ 428341.96476299502, 4578447.6954964502 ],
[ 428357.75161059998, 4578424.7347049201 ],
[ 428549.68073765899, 4578075.81088675 ]

],
{

"properties" : {

"CODI_CAPA" : "K028",
"CODI_SUBCAPA" : "K06",
"ID" : "GL241872",
"TOOLTIP" : "Carril Bici Bidireccional pg Zona Franca"
}

},
[...]
```

4.3. Creació d'una API

Una vegada obtingudes les dades d'interès mitjançant *web scraping*, pot ser interessant crear una API que contingui aquestes dades i hi accedeixi amb facilitat.

Una possible solució consisteix a crear una REST API, mitjançant les eines Flask i Flask-Restful. Així, després d'instal·lar la llibreria Flask-restful, el següent codi crearia una API senzilla d'aquest tipus:

```
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, '/')

if __name__ == '__main__':
    app.run(debug=True)
```

Bibliografia recomanada

K. Dale (2016). *Data Visualization with Python and JavaScript*. O'Reilly.

Enllaç d'interès

Per a més informació sobre la generació d'API mitjançant Flask, es pot consultar l'enllaç següent: <https://bit.ly/2swrcfu>.

4.4. Repositoris de dades

Atesa la importància de compartir dades de qualitat a internet amb l'objectiu de contribuir amb nou coneixement explotable, existeixen repositoris públics en els quals és possible compartir la informació obtinguda mitjançant processos *de web scraping*.

Encara que hi ha gran quantitat de repositoris de dades disponibles, els més utilitzats actualment són Kaggle, UCI Machine Learning Repository, Github i Data World. En les figures 13, 14, 15 i 16 es mostren exemples respectius d'aquests repositoris.

Figura 13. Exemple de conjunts de dades disponibles a Kaggle

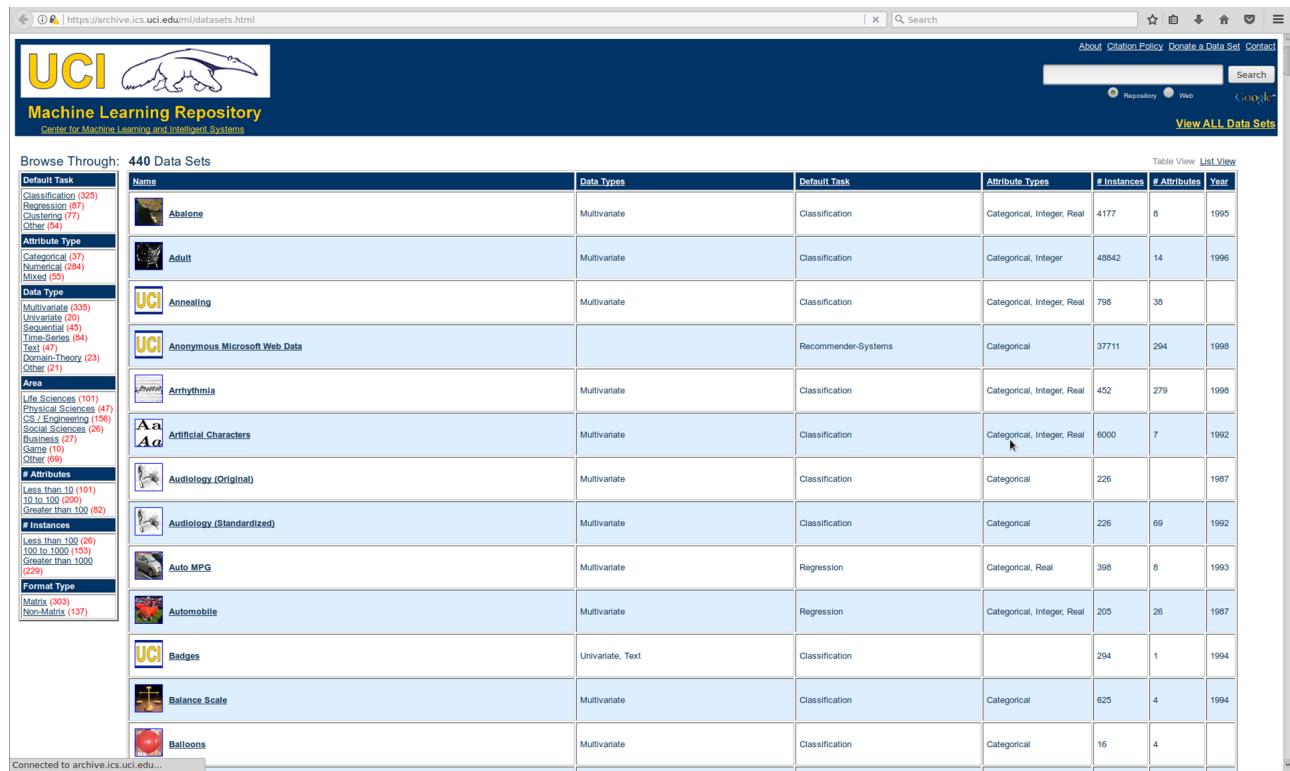
The screenshot shows the Kaggle Datasets page. At the top, there's a navigation bar with links for Competitions, Datasets (which is highlighted), Kernels, Discussion, Jobs, and Sign In. Below the header, a large blue banner displays the word 'Datasets'. A blue arrow points upwards from the bottom of the banner towards the 'Datasets' link in the header. On the left, there's a sidebar with a 'Featured' tab selected and an 'All' tab. The main content area shows three datasets:

- (LoL) League of Legends Ranked Games**: Details from over 50,000 ranked games of LoL. Posted by DataSnaek, updated 17 hours ago. 30 downloads, 0 comments.
- Severely Injured Workers**: ~22k Injury Reports for US Workers, 2015-2017. Posted by Jacob Boysen, updated a day ago. 12 downloads, 0 comments.
- European Soccer Database**: 25k+ matches, players & teams attributes for European Professional Football. Posted by Hugo Mathien, updated 10 months ago. 32,974 downloads, 96 comments.

Below the datasets, there's a search bar labeled 'Search datasets' and a 'Sort by' dropdown set to 'Hotness'.

Font: blog oficial de Kaggle

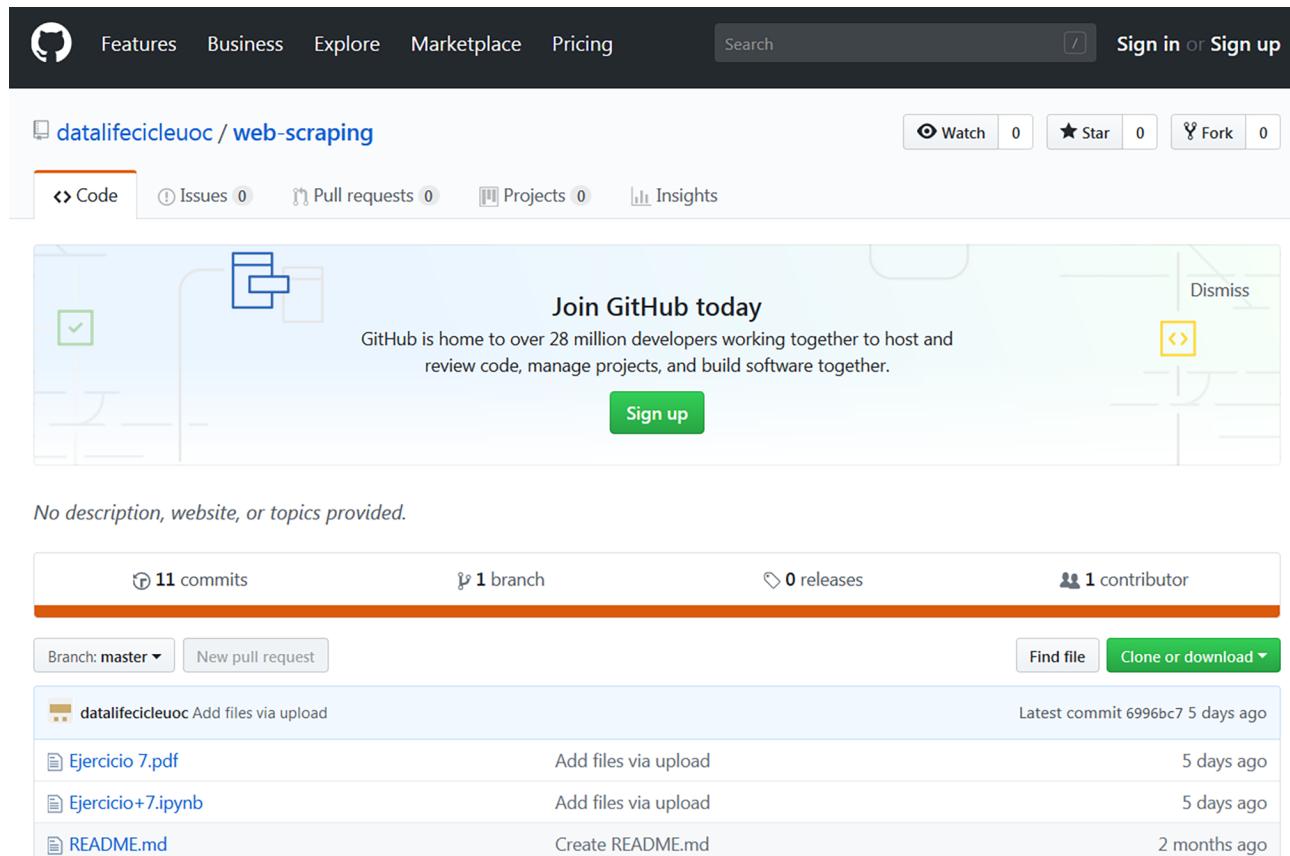
Figura 14. Exemple dels conjunts de dades disponibles a UCI Machine Learning Repository



The screenshot shows the homepage of the UCI Machine Learning Repository. On the left, there is a sidebar with filters for 'Default Task' (Classification, Regression, Clustering, Other), 'Attribute Type' (Categorical, Numerical, Mixed), 'Data Type' (Multivariate, Univariate, Sequences, Time-Series, Text, Domain/Theory, Other), 'Area' (Life Sciences, Physical Sciences, CS / Engineering, Social Sciences, Business, Game, Other), '# Attributes' (Less than 10, 10 to 100, Greater than 100), '# Instances' (Less than 1000, 1000 to 10000, Greater than 10000), 'Format Type' (Matrix, Non-Matrix), and 'Connected to archive.ics.uci.edu...'. The main area displays a table titled 'Browse Through: 440 Data Sets' with columns: Name, Data Types, Default Task, Attribute Types, # Instances, # Attributes, and Year. The first few rows include Abalone, Adult, Annealing, Anonymous Microsoft Web Data, Arhythmia, Artificial Characters, Audiology (Original), Audiology (Standardized), Auto MPG, Automobile, Badges, Balance Scale, and Balloons.

| Browse Through: 440 Data Sets | | | | | | |
|-------------------------------|------------------------------|------------------|---------------------|----------------------------|-------------|------|
| | Name | Data Types | Default Task | Attribute Types | # Instances | Year |
| | Abalone | Multivariate | Classification | Categorical, Integer, Real | 4177 | 8 |
| | Adult | Multivariate | Classification | Categorical, Integer | 48842 | 14 |
| | Annealing | Multivariate | Classification | Categorical, Integer, Real | 798 | 38 |
| | Anonymous Microsoft Web Data | | Recommender-Systems | Categorical | 37711 | 294 |
| | Arhythmia | Multivariate | Classification | Categorical, Integer, Real | 452 | 279 |
| | Artificial Characters | Multivariate | Classification | Categorical, Integer, Real | 6000 | 7 |
| | Audiology (Original) | Multivariate | Classification | Categorical | 226 | |
| | Audiology (Standardized) | Multivariate | Classification | Categorical | 226 | 69 |
| | Auto MPG | Multivariate | Regression | Categorical, Real | 398 | 8 |
| | Automobile | Multivariate | Regression | Categorical, Integer, Real | 205 | 26 |
| | Badges | Univariate, Text | Classification | | 294 | 1 |
| | Balance Scale | Multivariate | Classification | Categorical | 625 | 4 |
| | Balloons | Multivariate | Classification | Categorical | 16 | 4 |

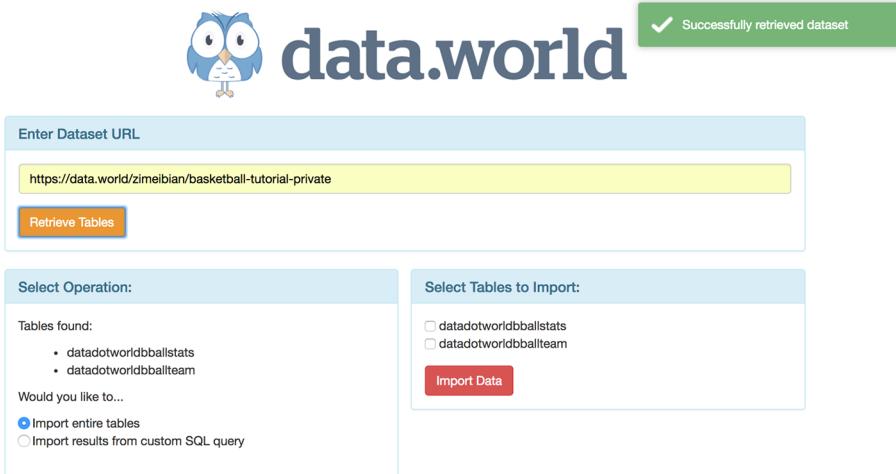
Figura 15. Exemple de pantalla d'un repositori Github



The screenshot shows a GitHub repository page for 'datalifeclieuoc/web-scraping'. At the top, there is a navigation bar with links for Features, Business, Explore, Marketplace, Pricing, and Sign in or Sign up. Below the navigation bar, the repository name 'datalifeclieuoc / web-scraping' is displayed, along with 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', and 'Insights' buttons. A 'Watch' button shows 0, a 'Star' button shows 0, and a 'Fork' button shows 0. A large green 'Sign up' button is prominently displayed. The main content area features a 'Join GitHub today' banner with the text 'GitHub is home to over 28 million developers working together to host and review code, manage projects, and build software together.' Below the banner, it says 'No description, website, or topics provided.' Underneath, there is a summary bar with '11 commits', '1 branch', '0 releases', and '1 contributor'. At the bottom, there is a list of files: 'Add files via upload' (Latest commit 6996bc7 5 days ago), 'Ejercicio 7.pdf' (5 days ago), 'Ejercicio+7.ipynb' (5 days ago), and 'README.md' (2 months ago). There are also buttons for 'Find file' and 'Clone or download'.

Font: Github.

Figura 16. Imatge de la interfície DataWorld



Font: DataWorld.

5. Prevenció del web scraping

Amb l'objectiu d'evitar el *web scraping*, l'administrador d'un lloc web pot aplicar diverses mesures que permetin detenir o alentir l'ús de bots. Com a conseqüència, s'han desenvolupat noves eines basades en visió per ordinador i processament de llenguatge natural que simulen el comportament humà i, per tant, aconsegueixen accedir al contingut web que es pretén rastrejar.

Enllaç d'interès

Per a més informació sobre el concepte de *web scraping*, consulteu l'enllaç següent:
https://en.wikipedia.org/wiki/web_scraping.

Hi ha, però, diversos mètodes dissenyats per a la prevenció del *web scraping*; a continuació, se'n presenten alguns dels més habituals:

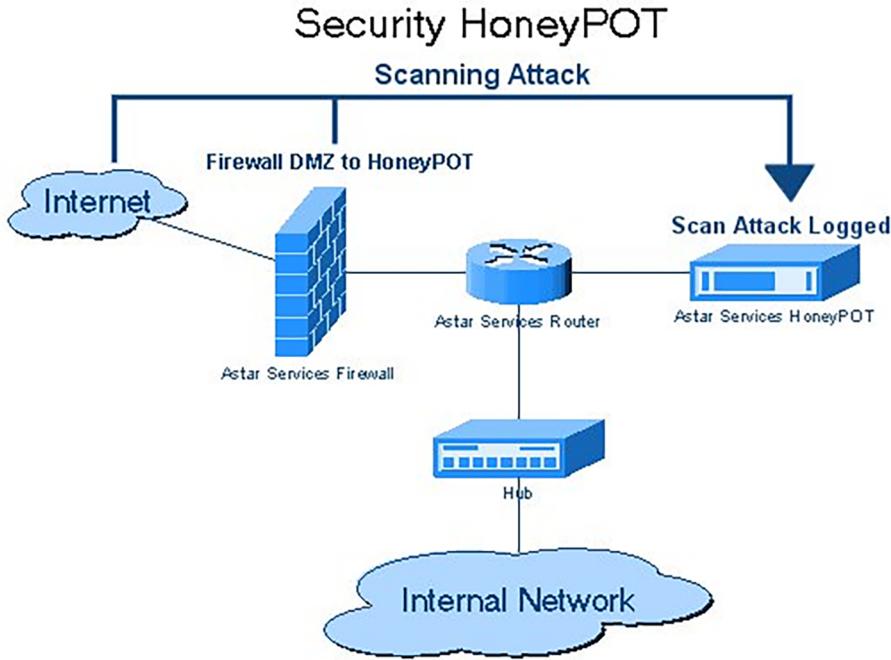
- 1) Bloqueig d'una adreça IP, manualment o basat en criteris com la geolocalització.
- 2) Desabilitació de qualsevol API o servei web associat al lloc.
- 3) Ús de l'arxiu *robots.txt* per a especificar el bloqueig de certs bots, com ara *googlebot*, o cadenes *user agent*.
- 4) Control de l'excés de tràfic.
- 5) Ús de CAPTCHA (*completely automated public Turing test to tell computers and humans apart*) per a verificar que qui accedeix al lloc és una persona real (vegeu la figura 17). No obstant això, alguns bots estan dissenyats per a resoldre certs patrons CAPTCHA. Així mateix, alguns sistemes de rastreig utilitzen mà d'obra humana per a respondre a aquests CAPTCHA en temps real.

Figura 17. Exemple d'un CAPTCHA (smwm)



- 6) Ús de serveis comercials contra el *web scraping*.
- 7) Ús d'esquers (*honeypots*) per a identificar adreces IP de rastrejadors automàtzats (vegeu la figura 18). Un esquer (o sistema trampa) és una eina de seguretat informàtica que es basa a atraure atacs amb la finalitat de neutralitzar-los.

Figura 18. Diagrama d'un esquer (*honeypot*)



Font: Viquipèdia (computing)

8) Ús de CSS *sprites* per a mostrar dades, com ara números de telèfon o adreces de correu electrònic, ja que això dificulta l'extracció de text als rastrejadors automàtics (vegeu la figura 19). No obstant això, aquest mètode comporta una pèrdua d'accessibilitat per als lectors de pantalla i els motors de cerca, a més d'una disminució en el rendiment del lloc.

Figura 19. Exemple de CSS *sprites*



Font: FormGet.

9) Afegir petites variacions entorn de les dades i elements de navegació HTML/CSS requereix més participació humana a l'hora d'inicialitzar un bot, per la qual cosa, si es fa de manera efectiva, pot complicar molt l'automatització del procés de *web scraping*.

6. Resolució d'obstacles en web scraping

A continuació, es resumeixen diferents mètodes que permeten evitar els obstacles o paranyos dissenyats per a prevenir el *web scraping* en una pàgina web.

6.1. Modificació del *user agent* i altres capçaleres HTTP

Com s'esmenta a l'apartat 2.1, quan es fa una petició HTTP, el navegador envia una sèrie de capçaleres al servidor web en les quals s'inclou informació sobre aquesta petició.

Una de les capçaleres per defecte més importants és l'agent d'usuari o *user agent*, ja que conté informació sobre el programari que està enviant la petició. En qualsevol navegador web, aquest s'ajusta automàticament a valors com Mozilla/5.0 (Macintosh, Intel Mac OS X...). De fet, és possible identificar quin agent d'usuari està utilitzant el nostre navegador simplement escrivint «check user agent» a la barra d'adreses, com es mostra a la figura 20.

Figura 20. Cerca de l'agent d'usuari o *user agent* amb Google



No obstant això, de manera predeterminada, les biblioteques utilitzades per a fer peticions HTTP automàticament estableixen el seu propi agent d'usuari basant-se en el nom de la llibreria, l'idioma, etc. Atès que això evidencia el fet que les peticions fetes provinguin d'un *script*, en lloc de ser una persona utilitzant el navegador, resulta molt recomanable reemplaçar aquesta capçalera per a evitar ser bloquejats a l'hora de fer *web scraping*.

Per a solucionar aquest problema, el següent codi Python mostra un exemple de com modificar algunes capçaleres HTTP, inclosa la corresponent a l'agent d'usuari:

```
import requests
headers = {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,",
    "*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate, sdch, br",
```

Bibliografia recomanada

H. Brody (2017). *The ultimate guide to web scraping*. LeanPub.

```

"Accept-Language": "en-US,en;q=0.8",
"Cache-Control": "no-cache",
"dnt": "1",
"Pragma": "no-cache",
"Upgrade-Insecure-Requests": "1",
"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/5\\
37.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
}

r = requests.get("http://www.example.com", headers=headers)

```

6.2. Gestió d'inici de sessió (*logins*) i de galetes (*cookies*) de sessió

De vegades, el lloc web que es pretén rastrejar requereix un inici de sessió (*login*) per a obtenir del servidor una galeta (*cookie*) de sessió que haurà d'acompanyar cadascuna de les peticions fetes posteriorment en el mateix lloc. En aquests casos, si s'intenta navegar anònimament, sense iniciar sessió, el servidor respondrà amb pàgines d'error o redireccions a pàgines d'inici.

Així, per a gestionar el seguiment de galetes configurades pel servidor, la llibreria Requests disposa de l'objecte `session`, que permet agregar aquestes galetes de manera automàtica a les posteriors peticions fetes en el mateix lloc.

El següent codi Python mostra un exemple d'ús:

```

import requests

session = requests.Session()
session.post("http://example.com/login", data=dict(
    email="me@domain.com",
    password="secret_value"
))
<!-- peticions realitzades amb session afegeixen automàticament les cookies--&gt;
r = session.get("http://example.com/protected_page")
</pre>

```

6.3. Respecte a l'arxiu robots.txt

Com s'esmenta en el subapartat 1.2.1, és important verificar si la pàgina d'interès disposa d'un arxiu *robots.txt* on el propietari del lloc hagi indicat les restriccions que cal tenir en compte quan es pretén rastrejar-lo.

Aquestes restriccions són només un suggeriment i mai una obligació, per la qual cosa en molts casos és possible recuperar informació de pàgines en les quals el propietari ha expressat la seva voluntat de no ser rastrejat. No obstant això, el més recomanable és seguir sempre els suggeriments indicats a *robots.txt*, amb l'objectiu de reduir les possibilitats de ser bloquejats i evitar problemes legals futurs.

6.4. Espaiat de peticions HTTP

L'usuari mitjà dedica uns segons a navegar per una pàgina abans d'intentar accedir a la següent, la qual cosa desencadena una altra petició HTTP en el seu navegador. Però quan les peticions procedeixen d'un *script*, aquest pot enviar tantes peticions simultànies que és capaç de saturar el servidor web. Fins i tot si el *web scraping* està permès en una pàgina web determinada, altres usuaris hi poden estar navegant alhora, per la qual cosa és important verificar que les nostres accions no estan col·lapsant el servidor web.

Per això, una pràctica habitual consisteix a introduir retards exponencials entre peticions consecutives quan es detecta algun error a la pàgina. En lloc de reexpedir una mateixa petició de manera pràcticament instantània, afegir un retard exponencialment creixent entre peticions proporciona al servidor web l'oportunitat de recuperar-se.

Un altre mètode àmpliament utilitzat consisteix a calcular el temps que els cal per a completar-se les diferents peticions i, a continuació, afegir un retard proporcional al temps estimat. D'aquesta manera, si el lloc es comença a alentir i les peticions fetes demanen més temps per a rebre una resposta, es pot ajustar el temps d'espera entre peticions automàticament.

El codi següent mostra un exemple d'espaiat automàtic entre peticions:

```
import time

for term in ["web scraping", "web crawling", "scrape this site"]:
    t0 = time.time()
    r = requests.get("http://example.com/search", params=dict(query=term))

    <!-- estimació del temps de resposta en segons--&gt;
    response_delay = time.time() - t0

    <!-- espera de 10x, segons el temps de resposta--&gt;
    time.sleep(10 * response_delay)</pre>

```

6.5. Ús de múltiples adreces IP

Alguns servidors web limiten el nombre de peticions rebudes per cada adreça IP en un cert període de temps. Per això, pot ser interessant utilitzar un conjunt d'adreces IP de manera que no totes les consultes es facin des de la mateixa adreça.

No obstant això, si bé és possible modificar les capçaleres HTTP esmentades en els punts anteriors, petites variacions en l'adreça IP a l'hora d'enviar una petició no permetran rebre la resposta del servidor i contindran la informació d'interès. Així, una solució consisteix a utilitzar un servidor intermediari (*proxy*) que emmascari l'origen de la petició. Quan s'envia una petició mitjan-

çant un servidor intermediari, aquesta es transmet primer a aquest servidor, el qual fa la petició al servidor web, de manera que rep la resposta i la transmet de nou a la màquina d'origen.

No obstant això, cal dir que l'ús d'aquests servidors sol implicar un cost addicional, al voltant de 40 USD per cada 100 adreces IP, segons Brody (2017).

6.6. Configuració de temporitzacions (*timeouts*) i altres excepcions

En un servidor web, les temporitzacions (*timeouts*) es produueixen quan aquest servidor triga molt a retornar una resposta, generalment més de 30 segons. Quan una petició pren tant temps, cal pensar que alguna cosa no funciona correctament. Per això, el més recomanable en aquests casos és anul·lar la petició, esperar una mica i tornar a intentar-ho de nou.

El codi següent mostra un exemple de configuració de temporitzacions mitjançant la llibreria Requests:

```
try:
    <!-- esperar fins a 10 segons--&gt;
    requests.get("http://example.com", timeout=10)
except requests.exceptions.Timeout:
    pass</pre>

```

Un altre problema habitual és el de les connexions caigudes (*broken connections*). Si la connexió o el servidor cauen inesperadament, la petició HTTP feta es trobarà en un estat ambigu que no retornarà una resposta útil. Per a solucionar aquest problema, la llibreria Requests permet afegir una excepció, semblant a la utilitzada en la gestió de temporitzacions, que evitarà que el codi utilitzat deixi de funcionar.

A continuació, se'n mostra un exemple:

```
try:
    requests.get("http://example.com")
except requests.exceptions.RequestException:
    pass
```

6.7. Evitar els paranyys d'aranya

Alguns llocs generen dinàmicament el seu contingut de manera que poden tenir un nombre infinit de pàgines web. Per exemple, si el lloc conté un calendari amb enllaços al mes i any següents, el mes següent també tindrà un enllaç

Bibliografia recomanada

H. Brody (2017). *The ultimate guide to web scraping*. LeanPub.

al mes següent, i així successivament. Atès que el nostre rastrejador seguirà en principi qualsevol enllaç del lloc que no hagi vist abans, es veurà atrapat en una successió infinita d'enllaços, coneguda com a parany d'aranya (*spider trap*).

Una manera senzilla d'evitar els paranys d'aranya consisteix a registrar la profunditat de la pàgina, definida com el nombre d'enllaços que s'han seguit per a arribar-hi. Així, en definir prèviament una profunditat màxima, quan s'aconsegueix aquest llindar, el rastrejador deixa d'agregar enllaços a la cua.

Per a implementar aquesta solució, es pot utilitzar el codi següent:

```
def link_crawler(..., max_depth=2):
    max_depth = 2
    seen = {}
    ...
    depth = seen[url]
    if depth != max_depth:
        for link in links:
            if link not in seen:
                seen[link] = depth + 1
                crawl_queue.append(link)
```

Així mateix, per a desabilitar aquesta opció es pot fixar la profunditat màxima (`max_depth`) a un valor negatiu, de manera que la profunditat actual de la pàgina no assoleixi mai aquest valor.

7. Aspectes legals

El panorama legal entorn del *web scraping* es troba en plena evolució, per la qual cosa moltes de les lleis vigents presenten encara certa complexitat i ambigüïtat quan s'apliquen en nous escenaris sorgits durant l'era digital, com els relacionats amb el *web scraping*.

Per això, aquesta secció, basada en Vanden Broucke i Baeyens (2018), resumeix les principals disposicions en les quals se solen basar els casos judicials associats al *web scraping*, a més d'alguns consells per a evitar problemes legals a l'hora d'embarcar-se en projectes que impliquin l'extracció d'informació de llocs web que no siguin de la nostra propietat.

D'una banda, a Estats Units, la majoria de casos judicials associats al *web scraping* s'han basat en algunes de les següents teories d'infracció o responsabilitat:

1) Incompliment de termes i condicions. La majoria de pàgines web publiquen una sèrie de termes i condicions o acords de llicència d'usuari que, sovint, aborden explícitament l'accés al seu contingut mitjançant rastrejadors (*scrapers*). Amb això, es pretén crear un incompliment de la responsabilitat contractual en establir un contracte entre el propietari del lloc web i l'*scraper*.

No obstant això, la publicació d'aquests termes en un lloc web pot no ser suficient per a mostrar que un rastrejador ha incomplert les condicions, si no hi ha una acceptació activa per part d'aquest.

Per això, l'ús d'una casella de verificació explícita o enllaç del tipus «Accepto» obliga el *web scraper* a acceptar activament els termes. De la mateixa manera, en els llocs en els quals és necessari iniciar sessió, la creació d'un compte sol incloure un acord explícit dels termes i condicions.

2) Infracció de drets d'autor o marca registrada. A Estats Units, la legislació legal de l'ús raonable (*fair use*) permet l'ús limitat de material protegit per drets d'autor sota certes condicions, sense el permís explícit del titular dels drets esmentats. Així, els usos amb aquestes finalitats com la paròdia, la crítica, els comentaris o la recerca acadèmica es consideren ús legítim. No obstant això, la majoria d'usos comercials es consideren una infracció.

3) Llei de frau i abús informàtic. Existeixen diverses lleis federals i estatals que prohibeixen l'accés a l'ordinador d'una altra persona. En resum, aquestes lleis afirmen que «qui accedeix intencionadament a un ordinador sense autorització [...] i com a resultat de tal conducta causa dany» està incomplint la llei.

Bibliografia recomanada

S. Vanden Broucke; B. Baeyens (2018). *Practical Web Scraping for Data Science*. Springer

4) Violació de domicili. Aquest terme es refereix a un delicte civil en el qual una entitat interfereix en la propietat personal d'un individu i hi causa pèrdua de valor o dany. El 1999, aquesta llei es va aplicar en un cas judicial entre Ebay i Bidder's Edge.

5) Protocol d'exclusió de robots. Es tracta d'un estàndard industrial que permet a una pàgina web disposar d'un arxiu *robots.txt* on es proporcionen instruccions sobre qui pot accedir al lloc i a quines pàgines es pot accedir. Encara que aquest arxiu té un valor legal limitat, abans de rastrejar qualsevol pàgina web és aconsellable verificar si el propietari hi està d'acord, per a evitar futurs problemes legals.

6) Llei de drets d'autor del mil·lenni digital i Llei CAN-SPAM. Aquestes lleis també han estat utilitzades en alguns casos judiciais relacionats amb *web scraping*.

La primera tipifica com a delicte la producció i difusió de tecnologia, dispositius o serveis destinats a eludir les mesures que controlen l'accés a material protegit per drets d'autor. Així mateix, penalitza l'acte d'eludir un control d'accés, independentment que existeixi o no una infracció real dels drets d'autor.

D'altra banda, la Llei de control de la invasió de pornografia i publicitat no sol·licitada (CAN-SPAM, per les sigles en anglès de *controlling the assault of non-solicited pornography and marketing*) va establir el 2003 els primers estàndards per a l'enviament de correu electrònic comercial.

Encara que la situació a la Unió Europea (UE) es regeix per diferents legislacions i sistemes jurídics, molts dels principis prèviament esmentats s'apliquen de manera similar, per exemple, en relació amb els termes i condicions o el contingut protegit per drets d'autor. De fet, la majoria de propietaris de pàgines web a la UE tendeixen a confiar en les demandes per infracció de drets d'autor amb l'objectiu d'inriminar els rastrejadors. Altres disposicions clau en aquest tipus de judicis s'enumeren a continuació:

1) Directiva de la UE sobre bases de dades, de 1996. Aquesta directiva proporciona protecció jurídica als creadors de bases de dades que no estiguin coberts per drets de propietat intel·lectual, de manera que protegeix els elements d'una base de dades que no són creació original de l'autor. En particular, proporciona protecció quan «s'ha fet una inversió substancial, tant qualitativa com quantitativa, per a l'obtenció, verificació o presentació dels resultats».

2) Llei d'ús indegit d'ordinadors. A més de la violació dels drets de propietat intel·lectual, teòricament, els propietaris de pàgines web disposen d'altres arguments legals per a lluitar contra el *web scraping*. És el cas de la Llei d'ús indegit d'ordinadors de 1990, que prohíbeix l'accés i modificació no autoritzats de material informàtic.

Enllaç d'interès

Per a més informació sobre el cas entre Ebay i Bidder's Edge, consulteu l'enllaç següent: https://en.wikipedia.org/wiki/ebay_v._bidder_%27s_Edge

Així, es pot observar com el *web scraping*, especialment quan es fa a gran escala o per a ús comercial, pot anar acompanyat d'implicacions legals complexes. Per això, és aconsellable consultar un advocat o els experts apropiats abans d'embarcar-se en aquest tipus de projectes, a més de tenir en compte els principis clau següents:

- **Obtenir permís per escrit.** La millor pràctica per a evitar problemes legals consisteix a obtenir permís escrit del propietari d'un lloc web, en el qual s'especifiqui fins a quin punt se'n pot extreure informació.
- **Verificar les condicions d'ús.** Aquestes inclouran sovint disposicions explícites contra l'extracció automàtica de dades. Així mateix, les API d'un lloc web solen anar acompanyades dels seus propis termes d'ús, per la qual cosa també és aconsellable revisar aquests casos.
- **Rastrejar només informació pública.** Generalment, quan un lloc web exposa informació públicament, sense ser necessari acceptar una sèrie de termes i condicions, es considera que l'ús moderat de *web scraping* és adequat. Aquells llocs en els quals és necessari iniciar sessió per a accedir a la informació d'interès, per contra, són més delicats des del punt de vista legal.
- **No causar dany.** No sobrecarregar el servidor amb gaires peticions, mantenir-se allunyat dels equips protegits i no intentar accedir als servidors als quals no es té accés.
- **Utilitzar la informació extreta de manera justa.** No utilitzar amb finalitats comercials les dades protegides per drets d'autor.

8. Millors pràctiques i consells

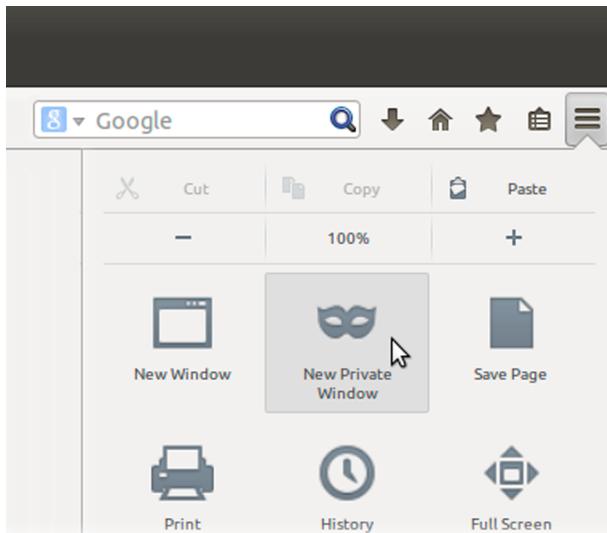
La llista següent, basada en Vanden Broucke i Baeyens (2018), resumeix una sèrie de bones pràctiques i consells a l'hora de fer *web scraping*:

- 1) Abans de fer *web scraping*, **verifiqueu si ja existeix una API** que permeti recuperar la informació d'interès, sense limitacions de descàrrega.
- 2) **No analitzeu l'HTML manualment.** L'ús de llibreries com BeautifulSoup facilita considerablement la tasca.
- 3) **No satureu de peticions el servidor web**, ja que això augmentarà les probabilitats de ser bloquejats. Així mateix, com que l'administrador de la web (*webmaster*) es pot adonar que s'està fent una gran quantitat de peticions a la seva pàgina, pot ser interessant contactar-hi per a trobar la manera de treballar conjuntament.
- 4) **Modifiqueu el *user agent***, ja que molts llocs revisen aquesta capçalera per a prevenir el *web scraping*.
- 5) **Reviseu el navegador.** Si desconeixem la causa d'un problema, pot ser interessant obrir una nova sessió en el navegador, preferiblement utilitzant els modes d'«incògnit» o de «navegació privada» (*private browsing*) per a assegurar que el conjunt de galetes es troba buit. Així mateix, es pot utilitzar la instrucció `curl` per a depurar els casos més complexos.

Exemple de navegació privada

Per a navegar privadament, per exemple per Mozilla Firefox, s'ha d'accedir al navegador com es mostra a la figura 21. Cal esmentar que, en aquests casos, el navegador mostra una màscara a la part superior.

Figura 21. Navegació en mode privat



Font: pantalla d'ajuda de Mozilla.

6) Assumiu que el *web scraper* deixarà de funcionar. Les pàgines web són dinàmiques, per la qual cosa pot ser molt útil implementar un codi que de seguida proporcioni advertiments detallats quan algun fragment deixi de funcionar.

7) Tingueu en compte la qualitat i robustesa de les dades obtingudes. La International Data Management Association del Regne Unit (DAMA UK) defineix la qualitat de les dades a partir de les sis dimensions següents:

- a) **Completesa** (*completeness*). Fa referència a la proporció de dades emmagatzemades davant del potencial de dades completes.
- b) **Unicitat** (*uniqueness*). En comparar les dades amb altres conjunts, aquestes han de ser úniques.
- c) **Puntualitat** (*timeliness*). El grau en què les dades representen la realitat, des del punt requerit en el temps.
- d) **Validesa** (*validity*). Les dades es consideren vàlides si s'ajusten a la sintaxi (format, tipus, rang) de la seva definició.
- e) **Exactitud** (*accuracy*). El grau en què les dades s'ajusten a la realitat que s'està descrivint.
- f) **Consistència** (*consistency*). Fa referència a l'absència de diferències quan es comparen dues o més representacions d'alguna cosa, pel que fa a la seva definició.

8) Recordeu els aspectes legals associats al *web scraping*, amb l'objectiu de fer un bon ús de les dades obtingudes.

Enllaç d'interès

DAMA UK Working Group.
The six primary dimensions for data quality assessment. Defining data quality dimensions.
<https://bit.ly/2qcimnf>

9. Exemples de web scraping i casos d'èxit

El *web scraping* pot aplicar-se a dominis molt variats, amb diferents finalitats. A continuació, se citen diversos exemples d'ús potencial en diferents camps.

- **Analitzar la competència.** El *web scraping* pot ser útil per a comparar productes i preus dels nostres principals competidors.
- **Gestionar la reputació d'una marca, comercial o personal, a internet.** Això es pot fer mitjançant grafs de relacions obtinguts de Viquipèdia, fent ànàlisis de sentiments procedents de xarxes socials, etc.
- **Analitzar el client.** Conèixer els gustos i preferències del client pot ajudar a donar millors serveis, i també a desenvolupar un pla de màrqueting més adaptat i, per tant, més eficient.
- **Fer tasques específiques.** En ocasions, és necessari rastrejar informació d'internet puntualment, per a solucionar problemes concrets.
- **Obtenir correus electrònics.** El màrqueting per correu electrònic (*e-mail marketing*) és una tècnica que les empreses utilitzen sovint per a contactar amb possibles clients. El *web scraping* permet trobar les adreces de correu electrònic d'aquests clients potencials per a enviar-los informació comercial.
- **Detectar opinions fraudulentes.** El *web scraping* permet recuperar informació amb l'objectiu de desenvolupar sistemes de detecció precoç de comportaments fraudulents a internet.
- **Millorar el posicionament:** optimització natural (SEO) per les sigles en anglès de *search engine optimization*. El posicionament SEO es basa en diversos paràmetres, un dels quals és el contingut de la pàgina web. Així, analitzar aquest contingut mitjançant *web scraping* pot ajudar en l'optimització del posicionament.
- **Resumir informació.** En ocasions, la informació d'interès pot estar distribuïda en diferents pàgines web, per la qual cosa el *web scraping* pot ajudar a centralitzar tota aquesta informació en un mateix lloc.

Bibliografia recomanada

S. Vanden Broucke; B. Baesens (2018). *Practical Web Scraping for Data Science*. Springer.

Però, en la pràctica, qui és que fa actualment *web scraping*? Encara que les possibles aplicacions d'aquesta eina són pràcticament il·limitades, la llista següent, recollida de Broucke i Baesens (2018), mostra alguns exemples interessants de casos d'ús reeixits:

- Molts productes de Google es beneficien d'aquesta tècnica. El traductor de Google (Google Translate), per exemple, utilitza text emmagatzemat al web per a entrenar-se, aprendre i millorar contínuament. Altres traductors com DeepL es basen en el mateix principi.
- En el camp dels recursos humans, aquesta tècnica guanya cada vegada més força. L'empresa emergent hiQ, per exemple, s'ha especialitzat en la venda d'informació sobre empleats, o potencials empleats, mitjançant l'ús de dades recollides d'internet, principalment extretes dels perfils públics de LinkedIn.
- Les empreses de màrqueting digital, i també els artistes digitals, utilitzen les dades presents a internet per a dissenyar tot tipus de projectes creatius. El projecte «We Feel Fine», presentat a l'apartat 1, en el qual Jonathan Harris i Sep Kamvar van rastrejar totes les frases presents en blogs començant per «I feel/I am feeling», va donar lloc a una manera molt creativa de mostrar com se sentia el món al llarg del dia.
- En un altre estudi, els missatges descarregats de Twitter, blogs i altres xarxes socials van ser analitzats per a construir una base de dades que permetés construir un model predictiu per a la identificació de patrons de depressió i pensaments suïcides. Encara que aquesta eina podria tenir un valor incalculable, hi ha certa controvèrsia relacionada amb la privadesa de les persones els missatges de les quals estan sent evaluats.
- En un article de Cavallo i Rigobon (2016), titulat «The Billion Prices Project: Using Online Prices for Measurement and Research» (El projecte dels mil milions de preus: l'ús de preus en línia per al mesurament i la recerca), es va utilitzar *web scraping* per a recollir un conjunt de dades format per gran quantitat de preus presents a internet, amb l'objectiu de construir un índex robust de preus diaris, per a diferents països.
- Els bancs i altres institucions financeres utilitzen el *web scraping* per a analitzar la competència. Per exemple, els bancs analitzen sovint les pàgines web de la competència per a saber on s'obren o es tanquen sucursals, i també per a fer un seguiment de les taxes de préstecs ofertes. De la mateixa manera, les empreses d'inversió solen utilitzar *web scraping* per a rastrejar les notícies relacionades amb els actius de la seva cartera.
- Els científics sociopolítics també utilitzen el *web scraping* per a analitzar els sentiments de la població, a més de la seva orientació política. Un famós article titulat «Dissecting Trump's Most Rabid Online Following» descriu els resultats d'un d'aquests estudis, en el qual es van analitzar discussions entre usuaris de la plataforma Reddit, amb l'objectiu de caracteritzar els seguidors de Donald Trump.

Enllaç d'interès

Podeu accedir al traductor DeepL en aquest enllaç: <https://www.deepl.com/translator>.

Enllaç d'interès

Podeu accedir a la pàgina web d'hiQ en aquest enllaç: <https://www.hiqlabs.com/>.

Enllaç d'interès

Per a més informació sobre el projecte, consulteu l'enllaç: [wefefine.org/](http://wefeelfine.org/).

Enllaç d'interès

Accediu a l'estudi sobre Twitter en aquest enllaç: <https://bit.ly/2wigpsy>.

Enllaç d'interès

Accediu a l'estudi de Cavallo i Rigobon (2016) en aquest enllaç: <http://www.nber.org/papers/w22111>.

Enllaç d'interès

Accediu a l'article a la pàgina web: <https://fivethirtyeight.com/features/dissecting-trumps-most-rabid-online-following/>.

- La informació extreta d'imatges procedents de Tinder i Instagram va permetre crear un model predictiu que identifiqués si una imatge era considerada «atractiva». Una dada interessant és que els fabricants de telèfons intel·ligents estan incorporant aquest tipus de models en les seves aplicacions fotogràfiques per a millorar la qualitat (o percepció de la qualitat) en les seves imatges.

Enllaç d'interès

Accediu a l'article sobre Tinder i Instagram en aquest enllaç: <https://bit.ly/2zaqjgs>.

Resum

En aquest mòdul didàctic s'han revisat els aspectes fonamentals relacionats amb *el web scraping*. En primer lloc, s'ha presentat la utilitat i potencial d'aquesta eina, particularment útil quan cal obtenir informació d'un lloc web que no disposa d'una API per a obtenir-la, o quan la seva API no satisfa del tot les necessitats del nostre projecte de ciència de dades.

Després de revisar alguns passos previs necessaris per a planificar qualsevol procés de *web scraping* de manera òptima, s'han revisat les principals eines disponibles a les llibreries Requests i BeautifulSoup que permeten recuperar textos, imatges o qualsevol altre contingut audiovisual procedents d'internet.

Posteriorment, a l'apartat 4, s'han presentat els formats estandarditzats més comunament utilitzats en l'emmagatzematge de dades rastrejades: CSV i JSON. Així mateix, s'han introduït les eines Flask mitjançant les quals és possible implementar una API que contingui i accedeixi fàcilment a aquestes dades. Finalment, s'han enumerat diferents repositoris públics en els quals es poden compartir les bases de dades resultants.

En l'apartat següent, s'han introduït diverses mesures que poden aplicar-se a qualsevol lloc web per a evitar el *web scraping*. A continuació, a l'apartat 6, s'han presentat diversos mètodes que permeten simular el comportament humà amb l'objectiu de resoldre aquestes mesures o obstacles, com l'ús de múltiples adreces IP, la configuració de temporitzacions (*timeouts*), la gestió d'inicis de sessió (*logins*) i galetes (*cookies*) de sessió, etc.

A continuació, s'han plantejat els principals aspectes legals relacionats amb l'extracció de dades procedents d'internet, i després s'ha presentat una llista de millors pràctiques i consells que ens permetin fer un bon ús del *web scraping*.

Finalment, amb l'objectiu de mostrar el potencial d'aquesta eina, s'han presentat alguns exemples d'aplicació i també casos d'èxit reals del *web scraping*.

Exercicis d'autoavaluació

1. Expliqueu amb les vostres pròpies paraules quan és útil fer *web scraping*. Imagineu que teniu un negoci, expliqueu quan podria ser útil aplicar-hi *web scraping*.
2. Poseu un exemple en què publicar dades obtingudes mitjançant *web scraping* sigui legal, i un altre en què no.
3. Enumereu els cinc passos previs necessaris per a planificar qualsevol procés de *web scraping* de manera òptima.
4. Els errors en la descàrrega de pàgines web poden ser temporals? Expliqueu per què.
5. És necessari establir un agent d'usuari (*user agent*)? Expliqueu per què.
6. Enumereu els mètodes utilitzats per a resoldre els obstacles més habituals en *web scraping*.
7. L'objectiu d'aquesta activitat és la creació d'un conjunt de dades (*dataset*) a partir de les dades contingudes en un lloc web. Primer, s'haurà d'analitzar si la pàgina web disposa d'un arxiu *robots.txt*, i també d'un mapa del lloc web, quina grandària té, la tecnologia emprada i el seu propietari. A continuació, s'han d'indicar les següents característiques del conjunt de dades (*dataset*) general:
 - a) Títol descriptiu del conjunt de dades.
 - b) Subtítol del conjunt de dades. Descripció àgil del conjunt de dades creat.
 - c) Imatge. Representació gràfica que identifiqui el conjunt de dades.
 - d) Context. Quina és la matèria del conjunt de dades?
 - e) Contingut. Quins camps inclou? Quin és el període de temps de les dades i com s'han recollit?
 - f) Agraiaments. Qui és el propietari del conjunt de dades?
 - g) Inspiració. Per què és interessant aquest conjunt de dades? Quines preguntes li agradaria respondre a la comunitat?
 - h) Llicència. S'ha de seleccionar una de les següents llicències i justificar l'elecció: Released Under CC0: Public Domain License, Released Under CC BY-NC-SA 4.0 License, Released Under CC BY-SA 4.0 License, Database Released Under Open Database License, Individual Contents Under Database Contents License, Other (specified above) o Unknown License.
 - i) Codi. S'ha d'especificar el codi utilitzat per a generar el conjunt de dades.
 - j) Conjunt de dades resultant, en format CSV.
8. Imagineu que voleu crear una empresa amb l'objectiu de vendre un producte o servei. Després d'explicar en tres línies la missió d'aquesta empresa, detalleu diferents conjunts de dades, ja existents o creats mitjançant *web scraping*, que utilitzarieu per a millorar el producte o servei ofert.

Solucionari

Exercicis d'autoavaluació

1. És útil fer *web scraping* quan no disposem d'API per a accedir a les dades web, o quan les API disponibles no aporten informació suficient per al nostre projecte de dades. Com a exemple de negoci, podem suposar una sabateria que pretén fer un seguiment dels preus de la competència. Podríem analitzar el lloc web del principal competitor cada dia, amb l'objectiu de comparar els preus de les diferents sabates a la venda; però això requeriria molt temps i no permetria controlar canvis freqüents en l'oferta. Per tant, una alternativa consistiria a reemplaçar aquest procés manual i repetitiu per una solució automatitzada, basada en tècniques de *web scraping*.

2. Exemple legal: llistes de telèfons. Exemple il·legal: opinions (associades a drets d'autor).

3. Revisar l'arxiu *robots.txt*, examinar el mapa del lloc web, calcular-ne la grandària, identificar la tecnologia utilitzada i conèixer l'administrador del web o propietari.

4. Sí, els errors de descàrrega poden ser temporals. Un exemple d'error temporal és el codi d'estat 503 Service Unavailable, després del qual es pot intentar la descàrrega més tard.

5. No és necessari, però és recomanable ja que alguns llocs web bloquegen l'usuari per defecte, per a evitar l'ús de rastrejadors automàtics.

6. Modificar l'agent d'usuari (*user agent*), gestionar l'inici de sessió i les galetes de sessió, analitzar l'arxiu *robots.txt*, espaiar les peticions HTTP, emprar múltiples adreces IP mitjançant servidors intermediaris, configurar temporitzacions i evitar els paranyos d'aranya.

7. La solució d'aquesta activitat es troba en el repositori Github (<https://github.com/datalifeclcieuoc/web-scraping>).

8. La missió de l'empresa és oferir calçat fabricat amb matèries primeres cent per cent reciclables i ecològiques. Tant la matèria primera com l'elaboració del producte seran de proximitat, i fomentaran l'ocupació local i el comerç just. Alguns conjunts de dades ja existents d'utilitat podrien ser els següents:

- a) Women's Shoe Prices obtingut de Kaggle.
- b) Men's Shoe Prices obtingut de Data World.
- c) Un altre exemple de conjunt de dades és UT Zappos50K.

Així mateix, amb l'objectiu de fer un estudi de mercat, es podria aplicar *web scraping* per a recuperar informació sobre els productes i preus que ofereix la competència.

A més, la informació disponible en pàgines com Viquipèdia o DBpedia podria proporcionar dades rellevants sobre la reputació de l'empresa amb les quals desenvolupar un graf de relacions que podríem analitzar amb programes com Gephi.

Una anàlisi de sentiments permetria avaluar la polaritat i subjectivitat d'aquesta informació, cosa que permetria detectar opinions fraudulentes.

D'altra banda, seria interessant analitzar les opinions dels nostres clients en el lloc web o en xarxes socials, amb l'objectiu d'adaptar tant el producte/servei ofert com el pla de màrqueting.

El *web scraping* també podria servir per a trobar adreces de correu electrònic de clients potencials, interessats en el comerç just, a protegir el medi ambient i en el calçat de disseny.

Finalment, disposar d'un resum de tota la informació disponible sobre calçat ecològic de comerç just en un mateix lloc podria ser interessant a l'hora de prendre decisions sobre el nostre model de negoci.

Glossari

API *f* Vegeu interfície de programació d'aplicacions.

Associació de gestió de dades *f* Associació sense ànim de lucre i independent, dedicada al desenvolupament de la gestió de recursos de dades (*data resource management* o DRM) i de la gestió de recursos d'informació (*information resources management* o IRM).

Sigla **DAMA**
en data management association

atom feeds *m* Format de redifusió web basat en un fitxer XML, desenvolupat com a alternativa al format RSS.

bot *m* Afèresi de robot. Programa informàtic autònom, capaç de dur a terme tasques concretes i repetitives per internet, la realització de les quals per un humà seria impossible o molt tediosa.

CAN-SPAM *m* Vegeu control de la invasió de pornografia i publicitat no sol·licitada.

CAPTCHA Sigles de *completely automated public turing test to tell computers and humans apart* (prova de Turing completament automàtica i pública per a diferenciar ordinadors d'humans). Prova de tipus desafiament-resposta, utilitzada en informàtica per a determinar quan un usuari és o no humà.

control de la invasió de pornografia i publicitat no sol·licitada *m* Llei federal estatunidenca que estableix les regles i sancions aplicables al correu electrònic comercial.
Sigla CAN-SPAM
en Controlling the Assault of Non-Solicited Pornography and Mârqueting

galeta *f* Petit arxiu amb dades procedents d'un lloc web, emmagatzemades en el navegador de l'usuari. Atès que proporciona informació sobre l'activitat prèvia de l'usuari en aquest lloc web, permet agilitar la navegació.

en cookie

CSS *f* Vegeu full d'estil en cascada.

CSS sprites *f* En una pàgina web, conjunt d'imatges o icones agrupades en una mateixa imatge.

CSV *m* Vegeu valor separat per comes.

DAMA *f* Vegeu associació de gestió de dades.

DOM *m* Vegeu model d'objectes del document.

Github *m* Plataforma de desenvolupament col·laboratiu de programari per a allotjar projectes utilitzant el sistema de control de versions Git.

full d'estil en cascada *f* Llenguatge de disseny gràfic que permet presentar, de manera estructurada, un document escrit en llenguatge d'etiquetatge. S'usa principalment en el disseny visual de documents web i interfícies d'usuari escrites en XML o HTML.

Sigla CSS
en cascading style sheets

HTML *m* Vegeu llenguatge d'etiquetatge d'hipertext.

honeypot (esquer, en català) Eina de seguretat informàtica dissenyada per a ser l'objectiu d'un possible atac, amb la finalitat de detectar-lo i obtenir informació tant de l'atac com de l'atacant.

HTTP Sigles d'*hypertext transfer protocol* (protocol de transferència d'hipertext). Protocol de comunicació que permet les transferències d'informació en la World Wide Web.

interfície de programació d'aplicacions *f* Conjunt de rutines que permeten accedir a funcions d'un determinat programari; a internet, les API permeten accedir al contingut d'un lloc web.

Sigla **API**
en application programming interface

JSON Acrònim de *javascript object notation*. Format de text lleuger utilitzat per a l'intercanvi de dades.

Llenguatge d'etiquetatge extensible *m* Metallenguatge extensible d'etiquetes, desenvolupat pel World Wide Web Consortium (W3C) i adaptat de l'SGML (*standard generalized markup language*).

Sigla XML

en extensible markup language

Llenguatge d'etiquetatge d'hipertext *m* Llenguatge d'etiquetatge utilitzat per a l'elaboració de pàgines web.

Sigla HTML

en hypertext markup language

mapa de lloc web *m* Conjunt de pàgines d'un lloc web, accessibles per a cercadors i usuaris.
en sitemap

marc de descripció de recursos *m* Família d'especificacions del World Wide Web Consortium (W3C), originalment dissenyat com un model de dades per a metadades.

Sigla RDF

en resource description framework

model d'objectes del document *m* Interfície de plataforma que proporciona un conjunt estàndard d'objectes per a representar, de manera jeràrquica, documents HTML, XHTML i XML. Mitjançant el DOM, els programes poden accedir i modificar el contingut, l'estructura i l'estil d'aquests documents.

Sigla DOM

en document object model

Llenguatge extensible de full d'estil *m* Família de llenguatges basats en l'estàndard XML que permet descriure com la informació continguda en un document XML qualsevol ha de ser transformada o formatada per a la seva presentació en un mitjà.

Sigla XSL

en extensible stylesheet language

optimització per a motors de cerca *f* Tècnica que consisteix a optimitzar l'estructura i informació d'un lloc web amb l'objectiu de millorar-ne la visibilitat en els resultats orgànics dels diferents cercadors d'internet.

Sigla SEO

en search engine optimization

parse Analitzar recorrent-los tots els registres d'una base de dades. Un *parser*, en informàtica, és un analitzador sintàctic, és a dir, un programa informàtic que analitza la sintaxi d'un document escrit en un llenguatge en particular.

profunditat (d'una pàgina web) *f* Nombre mitjà de clics necessaris per a arribar a una determinada pàgina del lloc.

RDF *m* Vegeu marc de descripció de recursos.

RSS feed Sigles de *really simple syndication* (sindicació realment simple). Format XML utilitzat per a la difusió de contingut web.

sindicació realment simple *f* Format XML utilitzat per a la difusió de contingut web.

Sigla RSS

en really simple syndication

SEO *m* Vegeu optimització per a motors de cerca.

servidor proxy *m* Equip dedicat o sistema de programari que actua com a intermediari en les peticions de recursos que fa un client a un altre servidor.

parany d'aranya *f* Conjunt de pàgines web que, intencionadament o no, poden causar que un rastrejador web o bot de cerca es bloquegi entre un nombre infinit de peticions.
en spider trap

valor separat per comes *m* Arxiu de text que emmagatzema les dades en forma de taula, on les columnes se separen per comes (o punt i coma en els idiomes en els quals la coma és el separador decimal) i les files per salts de línia.

Sigla CSV

en comma-separated values

XLSX *m* Format i extensió d'arxiu emprat a Microsoft Excel a partir de la seva versió 2007 (anteriorment XLS).

en Excel Microsoft Office Open XML Format Spreadsheet

XML *m* Vegeu llenguatge d'etiquetatge extensible.

XSL *m* Vegeu llenguatge extensible de full d'estil.

webmaster *m* Persona responsable del desenvolupament, coordinació i manteniment d'un lloc web.

Bibliografía

Acodemy (2015). *Learn Web Scraping With Python In A Day: The Ultimate Crash Course to Learning the Basics of Web Scraping With Python In No Time*. CreateSpace Independent Publishing Platform.

Bosch, O. (2017). *An introduction to web scraping, IT and Legal aspects*. <<https://bit.ly/2pMUYKC>>

Brody, H. (2017). *The ultimate guide to web scraping*. LeanPub.

Broucke, S. Vanden; Baesens, B. (2018). *Practical Web Scraping for Data Science*. Springer.

Casas, J.; Conesa, J. (2016). *Datos abiertos y enlazados*. Editorial UOC.

Cavallo, A.; Rigobon, R. (2016). *The Billion Prices Project: Using Online Prices for Measurement and Research. Journal of Economic Perspectives* (vol. 30, núm. 2, pàg. 151-178).

Dale, K. (2016). *Data Visualization with Python and JavaScript*. O'Reilly.

Heydt, M. (2018). *Python Web Scraping Cookbook: Over 90 proven recipes to get you scraping with Python, microservices, Docker, and AWS*. Packt Publishing.

Kouzis-Loukas, D. (2016). *Learning Scrapy*. Packt Publishing.

Lawson, R. (2015). *Web Scraping with Python*. Packt Publishing Ltd.

Minguillón, J. (2016). *Fundamentos de Data Science*. Editorial UOC.

Mitchell, R. (2015). *Web Scraping with Python: Collecting Data from the Modern Web*. O'Reilly.

Munzert, S.; Rubba, C.; Meißner, P.; Nyhuis, D. (2014). *Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining*. Hoboken, NJ; Chichester; West Sussex: John Wiley & Sons.

Nair, V.G. (2014). *Getting started with BeautifulSoup*. Packt Publishing Ltd. Open Source Collaborative framework in Python. [Data de consulta: 15 de març del 2018]. <<https://scrapy.org>>