# API AND WEB SCRAPING

## DATA COLLECTION AND DATA PROCESSING

> "The streets of the Web are paved with data that can't wait to be collected."
>
> Munzart, Rubba, Meissner, Nyhuis, Automated Data Collection with R

# WORLD WIDE WEB

There was a time in the recent past where both scarcity and inaccessibility of data was a problem for researchers and decision-makers. That is **emphatically** not the case anymore.

Data abundance carries its own set of problems:

- tangled masses of data
- traditional data collection methods and classical (small) data analysis techniques may not be sufficient anymore

# WEB DATA SCRAPPING EXAMPLE – NEW PHONE

Let's say you want to know what people think of a new phone. Standard approach: market research (e.g. telephone survey, reward system, etc.)

**Pitfalls:**

- unrepresentative sample: the selected sample might not represent the intended population
- systematic non-response: people who don't like phone surveys might be less (or more) likely to dislike the new phone
- coverage error: people without a landline can't be reached, say
- measurement error: are the survey questions providing suitable info for the problem at hand?

# WEB DATA QUALITY – NEW PHONE

These solutions can be **costly**, **time-consuming**, **ineffective**.

**Proxies** – indicators that are strongly related to the product's popularity, without measuring it directly.

If **popularity** is defined as large groups of people preferring one product over a competitor, then sales statistics on a commercial website may provide a proxy for popularity.

Rankings on Amazon could provide a more **comprehensive** view of the phone market vs. traditional survey.

# POTENTIAL ISSUES – NEW PHONE

**Representativeness** of the **listed products**

- Are all phones listed?
- If not, is it because that website doesn't sell them?
- Is there some other reason?

**Representativeness** of the **customers**

- Are there specific groups buying/not-buying online products?
- Are there specific groups buying from specific sites?
- Are there specific groups leaving/not-leaving reviews?

**Truthfulness** of customers and **reliability** of reviews.

# IS WEB SCRAPING LEGAL?

**What is a spider?**

- Programs that graze or crawl the web for information rapidly
- Jumps from one page to another, grabbing the entire page content

**Scraping** is taking specific information from specific websites (which is the goal): how are these **different**?

"Scraping inherently involves **copying**, and therefore one of the most obvious claims against scrapers is copyright infringement."

# LEGAL CASES – WEB SCRAPING

## eBay vs. Bidder's Edge (BE)

- BE used automated programs to crawl information from different auction sites.
- Users could search listings on the BE webpage instead of going to individual auction sites.
- BE accessed eBay's sites ~100 000 times / day (1.53% of # of requests, 1.1% of total data transferred by eBay) in 1999.
- eBay alleged damages of up to $45k- $62K in a 10 month period.
- BE didn't steal information that wasn't public, but excessive traffic was demanding on eBay's servers.
- **Your verdict?**

# FRIENDLY COOPERATION WITH APIS

What is an API? API stands for application program interface which is a set of routines, protocols and tools for building software applications.

Many APIs restrict the user to a certain amount of API calls per day (or some other limits).

These limits should be obeyed.

# Supplemental Material

# WHY AUTOMATED DATA COLLECTION?

## With regards to social scientific data:

- sparse financial resources
- little time or desire to collect data by hand
- want to work with up to date, high quality data rich sources
- document process from beginning (data collection) to end (publication) so it can be reproduced

## Issues with manual collection:

- non-reproducible process
- prone to errors and cumbersome
- subject to heightened risks of “death by boredom”

# WHY AUTOMATED DATA COLLECTION?

**Advantages of program-based solutions:**

- reliability
- reproducibility
- time-efficient
- assembly of higher quality datasets

# AUTOMATED COLLECTION CHECKLIST

Is **web scraping** or **statistical text processing** (automated or semi-automated data collection) really necessary?

**Criteria:**

- Do you plan to repeat the task from time to time e.g. to update your database?
- Do you want others to be able to replicate your data collection process?
- Do you deal with online sources of data frequently?
- Is the task non-trivial in terms of scope and complexity?

# AUTOMATED COLLECTION CHECKLIST

**Criteria:** (continued)

- If the task can be done manually, do you lack the resources to let others do the work?
- Are you willing to automate the process by means of programming?

If most of the answers are positive then an automated approach may be the right choice.

# WORLD WIDE WEB

The way we **share**, **collect**, and **publish** data has changed over the past few years due to the ubiquity of the *World Wide Web* (WWW).

**Private businesses**, **government**, and **individual users** are posting and sharing all kinds of data and information.

At every moment, new channels generate vast amounts of data on human behaviour.

# OPEN SOURCE SOFTWARE

Another trend:

- growth and increasing popularity and power of **open source software** (source code can be inspected, modified, and enhanced by anyone).

Community aspect → ever-changing and improving

**R** and **Python** are open source software that can be used for data analysis in the social sciences and other domains

They incorporate **interfaces** to other programming languages and software **solutions**.

# DATA CLEANING AND DATA PROCESSING

Data collection proper is only the tip of the iceberg.

Data cleaning and data processing is **essential** (as well as time-consuming).

**Tasks:**

- Selecting the columns (variables) of interest
- Re-labeling these columns
- Modifying the data type of the columns so that the data can be used the way we want

# DATA CLEANING AND DATA PROCESSING

**Tasks:** (continued)

- Editing and/or extracting data in a column
- Deciding how to deal with missing data (which can be a challenge)
- Multiple other tasks, depending on the data and its uses

Certain tasks can be automated, others cannot.

# QUESTIONS ABOUT DATA QUALITY

1. What type of data is most suited to answer your questions?
2. Is the quality of the data sufficiently high to answer your question?
3. Is the information systematically flawed?

---

Can you avoid the dreaded: “well, it’s the best data we have…”?

# DATA QUALITY

**First-hand information:** for example, a tweet, or a news article

**Second-hand data:** data that has been copied from an offline source or scraped from elsewhere.

Sometimes you can't remember or retrace the source of data when it is second-hand.

Does it still make sense to use it? It depends.

**Cross-validation** is standard for use of any secondary data.

# DATA QUALITY AND USER'S PURPOSE

Data quality depends on the **application**.

For example,

- Sample of tweets collected on a random day could be used to analyze the use of a hashtags or the gender-specific use of words
- Not as useful if collected on Election Day to predict the election outcomes (**collection bias**)

# DATA SOURCES (TRADE-OFFS)

Automated vs. Traditional

Accuracy vs. Completeness

Coverage vs. Validity

Speed vs. Cost

etc.

# DATA COLLECTION PROCESS (5 STEPS)

## 1. Know exactly what kind of information you need

- Specific: GDP of all OECD countries for last 10 years; sales of top 10 shoe brands in 2017
- Vague: people's opinion on shoe brand X

## 2. Find out if there are any web data sources that could provide direct or indirect information on your problem

- Easier for specific facts: shoe store's webpage will provide information about shoes that are currently in demand i.e. sandals, boots, etc.
- Tweets may contain opinion trends on *anything*
- Commercial platforms can provide information on product satisfaction

# DATA COLLECTION PROCESS (5 STEPS)

**3. Develop a theory of the data generation process when looking into potential sources**

- When was the data generated?
- When was it uploaded to the Web?
- Who uploaded the data?
- Are there any potential areas that are not covered? consistent? accurate?
- How often is the data updated?

# DATA COLLECTION PROCESS (5 STEPS)

## 4. Balance advantages and disadvantages of potential data sources

- Validate the quality of data used
- Are there other independent sources that provide similar information to crosscheck against
- Can you identify original source of secondary data

## 5. Make a decision

- Choose data source that seems most suitable
- Document reasons for this decision
- Collect data from several sources to validate data sources

# IS WEB SCRAPING LEGAL?

**Ethical Guidelines:**

- Work as transparently as possible
- Document data sources at all time
- Give credit to those who originally collected and published the data
- If you did not collect the information, you probably need permission to reproduce it
- Don't do anything illegal.

Crawling another company's information to process and resell it is a common complaint.

# LEGAL CASES – WEB SCRAPING

## Associated Press (AP) vs. Meltwater

- Meltwater offers software that scrapes news information based on specific keywords.
- Clients order summaries on topics containing excerpts of news articles.
- AP said their content was stolen and that Meltwater needed a license before distributing the information that was scraped.
- The judge found in favour of AP arguing that Meltwater is a competitor.
- **Your verdict?**

# LEGAL CASES – WEB SCRAPING

## Facebook vs. Pete Warden

- Pete Warden scraped basic information from Facebook users' profiles, to offer services to manage communication and networks.
- His process, according to him, was in line with robots.txt.
- After his first blog post using the data he scraped from Facebook, he was asked to delete the data.
- Facebook contends that robots.txt has no legal force and they could sue anyone for accessing their site even if they complied with the scraping instructions, that the only legal way to access any website with a crawler was to obtain prior written permission.
- **Your verdict?**

# LEGAL CASES – WEB SCRAPING

## United States vs Aaron Swartz

- Swartz co-created RSS, markdown and Infogami.
- He was arrested in 2011 for having illegally downloaded millions of articles from the archives of JSTOR.
- The case was dismissed after his suicide in January 2013.
- **Your verdict?**

# LESSONS LEARNED

Not clear which scraping actions are illegal and which are legal.

Re-publishing content for commercial purposes is considered more problematic than downloading pages for research/analysis.

**Robots.txt:** *Robots Exclusion Protocol* is a file that tells scrapers what information on the site may be harvested.

**Be friendly!** Not everything that can be scraped requires to be scraped. Scraping programs should behave "nicely", provide the data you seek, and be efficient, in this order.

cqads.carleton.ca/robots.txt

```
#
# robots.txt
#
# This file is to prevent the crawling and indexing of certain parts
# of your site by web crawlers and spiders run by sites like Yahoo!
# and Google. By telling these "robots" where not to go on your site,
# you save bandwidth and server resources.
#
# This file will be ignored unless it is at the root of your host:
# Used:    http://example.com/robots.txt
# Ignored: http://example.com/site/robots.txt
#
# For more information about the robots.txt standard, see:
# http://www.robotstxt.org/robotstxt.html

User-agent: *
Crawl-delay: 10
# Directories
Disallow: /includes/
Disallow: /misc/
Disallow: /modules/
Disallow: /profiles/
Disallow: /scripts/
Disallow: /themes/
# Files
Disallow: /CHANGELOG.txt
Disallow: /cron.php
Disallow: /INSTALL.mysql.txt
Disallow: /INSTALL.pgsql.txt
Disallow: /INSTALL.sqlite.txt
Disallow: /install.php
Disallow: /INSTALL.txt
Disallow: /LICENSE.txt
Disallow: /MAINTAINERS.txt
Disallow: /update.php
Disallow: /UPGRADE.txt
Disallow: /xmlrpc.php
```

```
User-agent: Twitterbot
Allow: /

User-agent: *
Disallow: /esi/
Disallow: /webview
Disallow: /vueweb
Disallow: /news/sponsored
Disallow: /search
Disallow: /19849159/
```

theweathernetwork.com/robots.txt

```
User-agent: *
Disallow:
Crawl-delay: 10
```

cfl.ca/robots.txt

# CONTACT DATA PROVIDERS

Any data accessed by HTTP forms is stored in some sort of database.

Ask proprietors of the data first if they will grant access to the database or files.

The larger the amount of data you want, **the better it is for both parties to communicate before starting to harvest data**.

For small amounts of data, that's less important.

# SCRAPING DO'S AND DON'T'S

## 1. Stay identifiable

## 2. Reduce traffic

- Accept compressed files
- If scraping the same resources multiple times, check first if it has changed before accessing again
- Retrieve only parts of a file

# SCRAPING DO'S AND DON'T'S

## 3. Do not bother server with multiple requests

- Many requests per second can bring smaller servers down
- Webmasters may block you if your scraper behaves this way
- One or two request per second is fine

## 4. Write modest scraper (efficient and polite)

- No reason to scrape pages daily or repeat same task over and over; make your scraper as efficient as possible
- Do not over-scrape pages
- Select resources you want to use and leave the rest untouched

# DEVELOPER TOOLS

Developer tools allow us (among other things) to see the correspondence between the HTML code for a page, and the rendered version we see in the browser.

Unlike “View Source”, developer tools shows the *dynamic* version of the HTML (i.e. the HTML is shown with any changes made by JavaScript since the page was first received).

Inspecting a page’s various elements and discovering where they reside in the HTML file is crucial to efficient web scraping.

# DEVELOPER TOOLS

## Firefox

- right click page → Inspect Element

## Safari

- Safari → Preferences → Advanced → Show Develop Menu in Menu Bar
- Develop → Show Web Inspector

## Chrome

- right click page → Inspect

Nice Peter - ERB

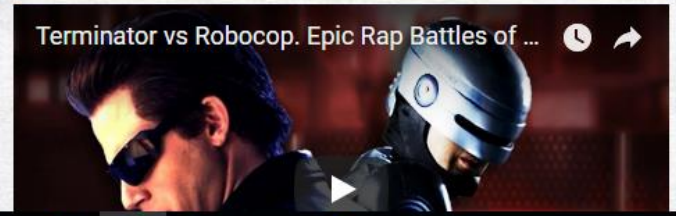
Secure | https://nicepeter.com/erb


HOME | LIVE SHOWS | ERB | MUSIC | VIDEOS | GALLERY | SHOP | PRESS | ARCHIVE | CONTACT

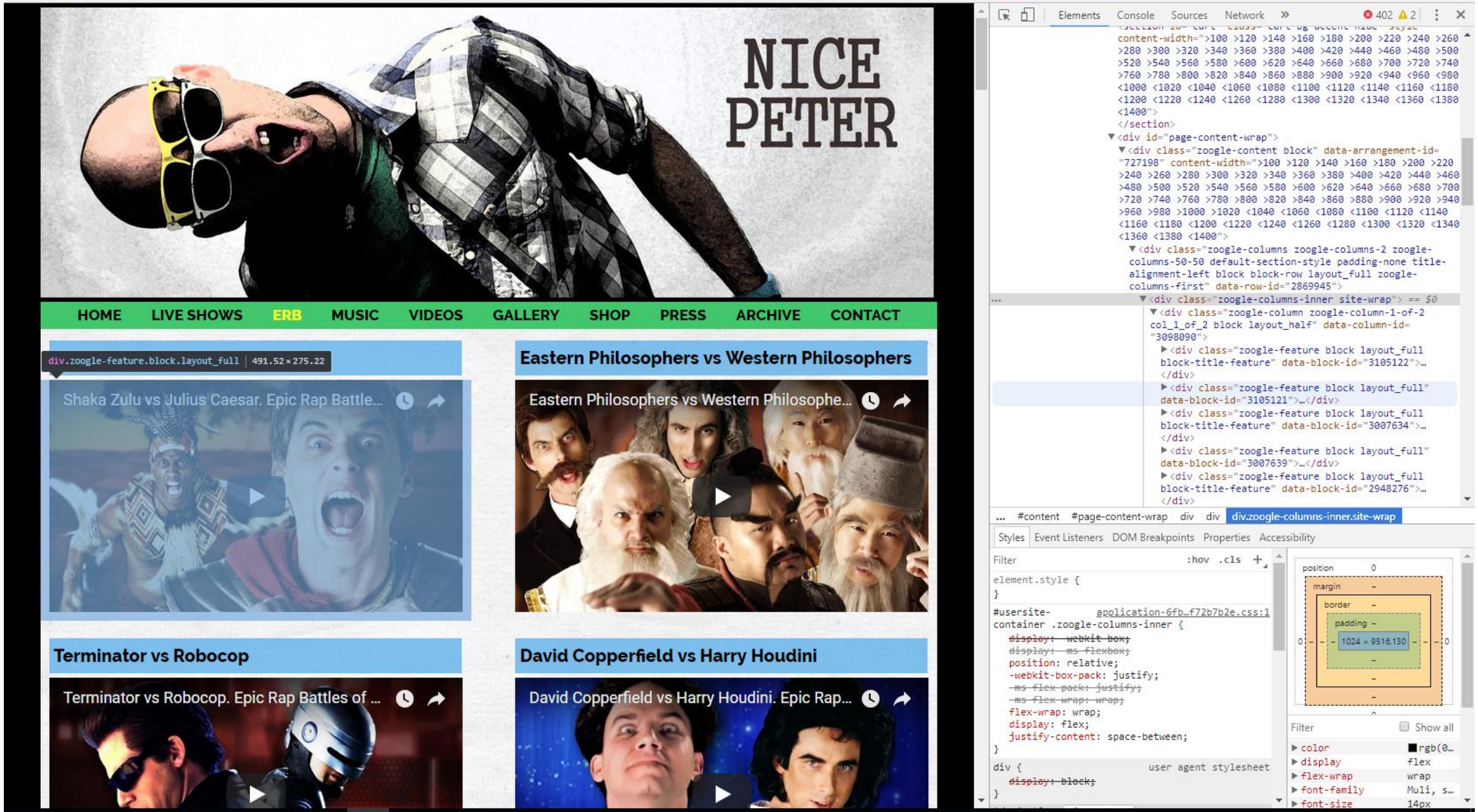
## Shaka Zulu vs Julius Caesar


## Eastern Philosophers vs Western Philosophers


## Terminator vs Robocop



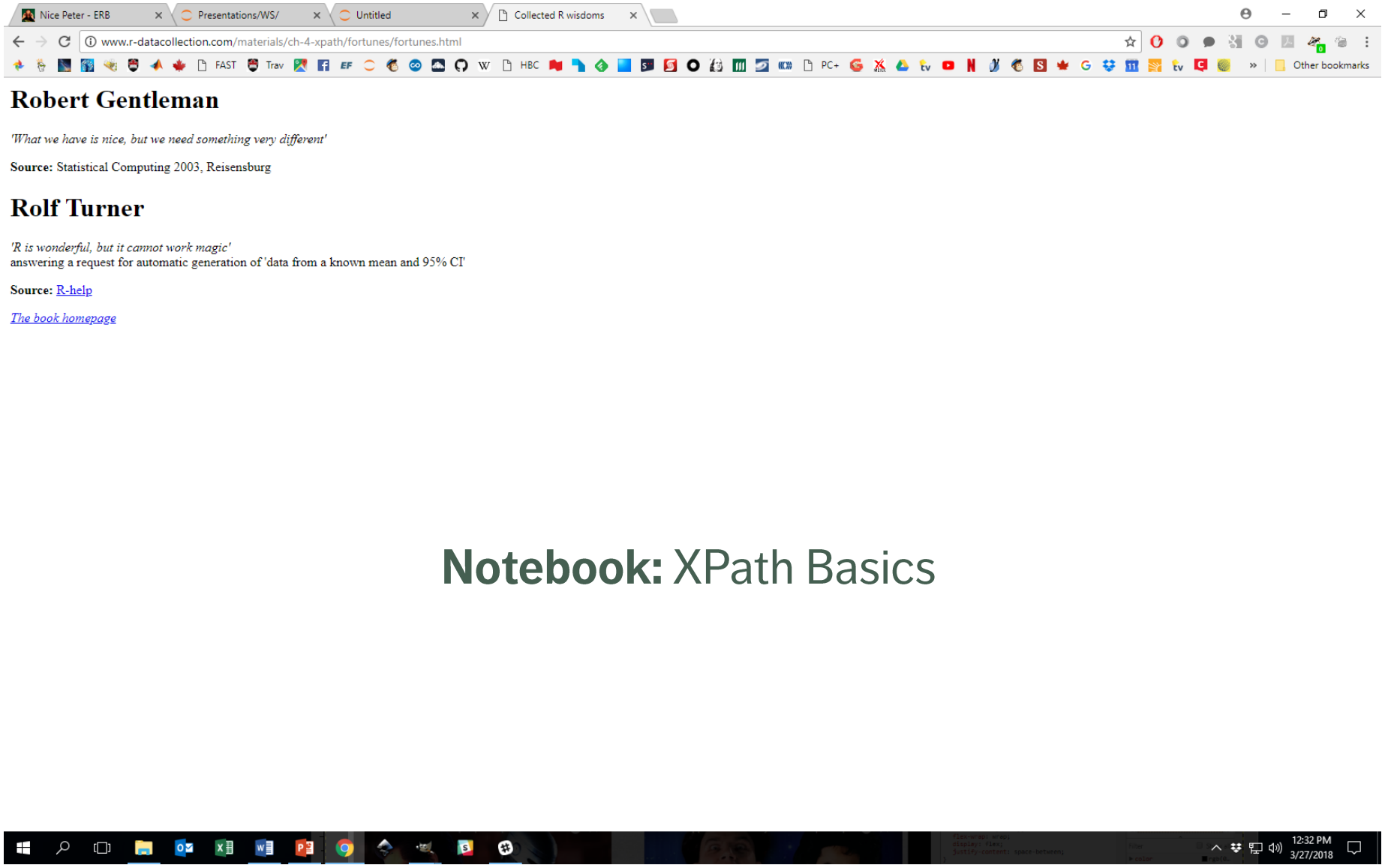

## David Copperfield vs Harry Houdini


Nice Peter - ERB
Secure | https://nicepeter.com/erb
NICE PETER
HOME
LIVE SHOWS
ERB
MUSIC
VIDEOS
GALLERY
SHOP
PRESS
ARCHIVE
CONTACT
div.zoogle-feature.block.layout_full | 491.52 × 275.22
Shaka Zulu vs Julius Caesar. Epic Rap Battle...
Eastern Philosophers vs Western Philosophers
Eastern Philosophers vs Western Philosophe...
Terminator vs Robocop
Terminator vs Robocop. Epic Rap Battles of ...
David Copperfield vs Harry Houdini
David Copperfield vs Harry Houdini. Epic Rap...
Elements
Console
Sources
Network
#content
#page-content-wrap
div
div
div.zoogle-columns-inner.site-wrap
Styles
Event Listeners
DOM Breakpoints
Properties
Accessibility
position
margin
border
padding
1024 × 9516.130
12:11 PM
3/27/2018

# XPATH

**XPath** is a query (domain-specific) language

- Used to select specific pieces of information from marked-up documents such as HTML, XML or variants such as SVG, RSS
- Information stored in marked-up documents needs to be converted into formats suitable for processing and statistical analysis
- Implemented in R package `XML`
- Process steps:
    1. Specifying the data of interest
    2. Locating it in a specific document
    3. Tailoring a query to the document to extract the desired info.



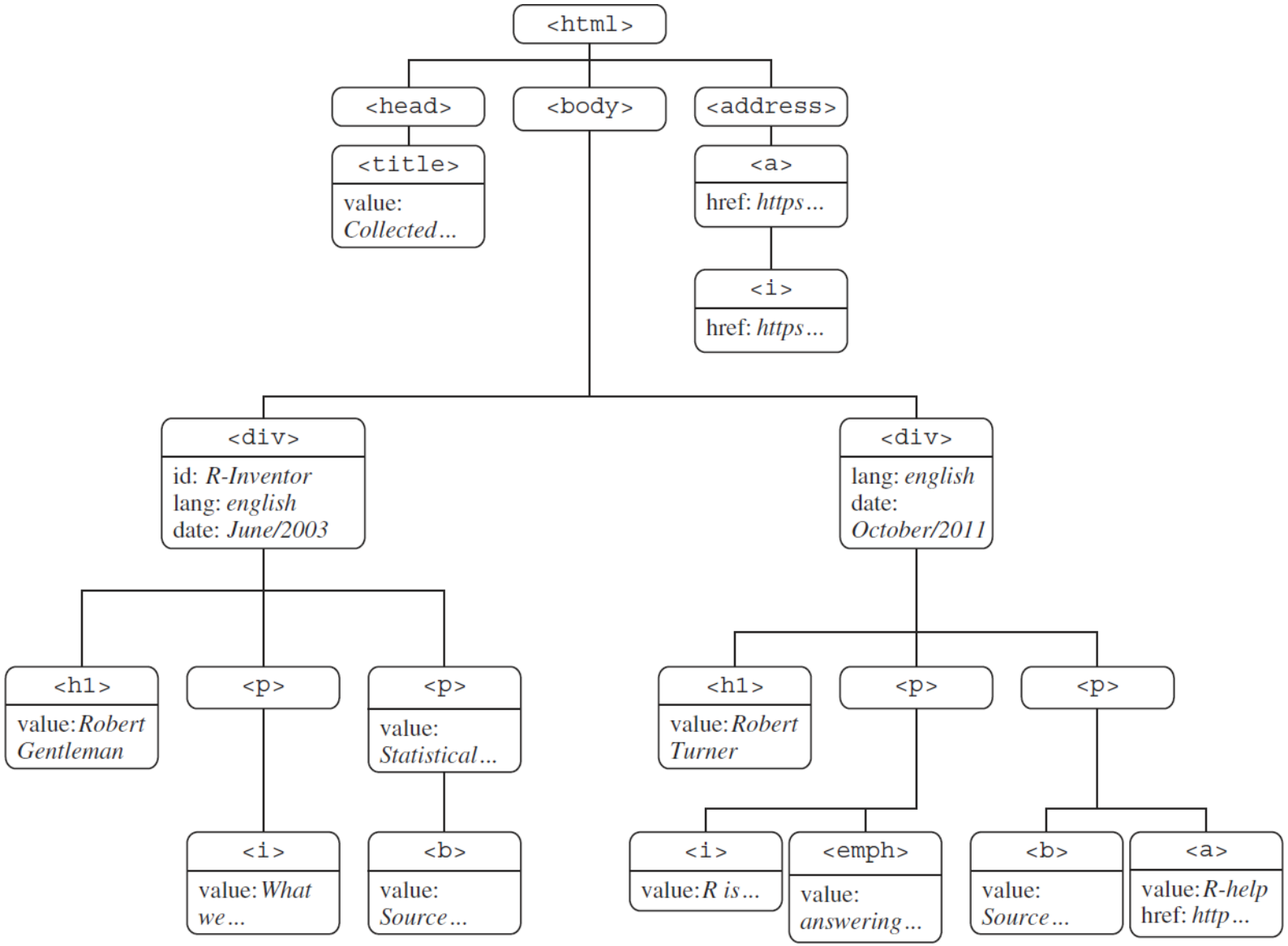

# **Notebook:** XPath Basics

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head><title>Collected R wisdoms</title></head>
<body>
<div id="R Inventor" lang="english" date="June/2003">
  <h1>Robert Gentleman</h1>
  <p><i>'What we have is nice, but we need something very different'</i></p>
  <p><b>Source: </b>Statistical Computing 2003, Reisensburg</p>
</div>

<div lang="english" date="October/2011">
  <h1>Rolf Turner</h1>
  <p><i>'R is wonderful, but it cannot work magic'</i> <br><emph>answering a request
for automatic generation of 'data from a known mean and 95% CI'</emph></p>
  <p><b>Source: </b><a href="https://stat.ethz.ch/mailman/listinfo/r-help">R-help</a>
</p>
</div>

<address>
<a href="http://www.r-datacollectionbook.com"><i>The book homepage</i></a><a></a>
</address>

</body>
</html>
```


<html>
<head>
<body>
<address>
<title>
value: Collected...
<a>
href: https...
<i>
href: https...

id: R-Inventor
lang: english
date: June/2003

lang: english
date: October/2011
<h1>
value: Robert Gentleman
<p>
<p>
value: Statistical...
<h1>
value: Robert Turner
<p>
<p>
<i>
value: What we...
<b>
value: Source...
<i>
value: R is...
<emph>
value: answering...
<b>
value: Source...
<a>
value: R-help
href: http...

# XPATH – BASIC STRUCTURE

HTML/XML tags have **attributes** and **values**.

HTML files must be parsed before they can be queried by XPath.

XPath queries require a path and a document to search.

- paths consist of hierarchical addressing mechanism (succession of nodes, separated by forward slashes ["/"])
- a query takes the form `xpathSApply(doc,path)`:
  - `xpathSApply(parsed_doc,"/html/body/div/p/i")`

would find all `<i>` tags inside a `<p>` tag inside a `<div>` tag in the `body` of the `html` file.

# XPATH – NODE RELATIONS

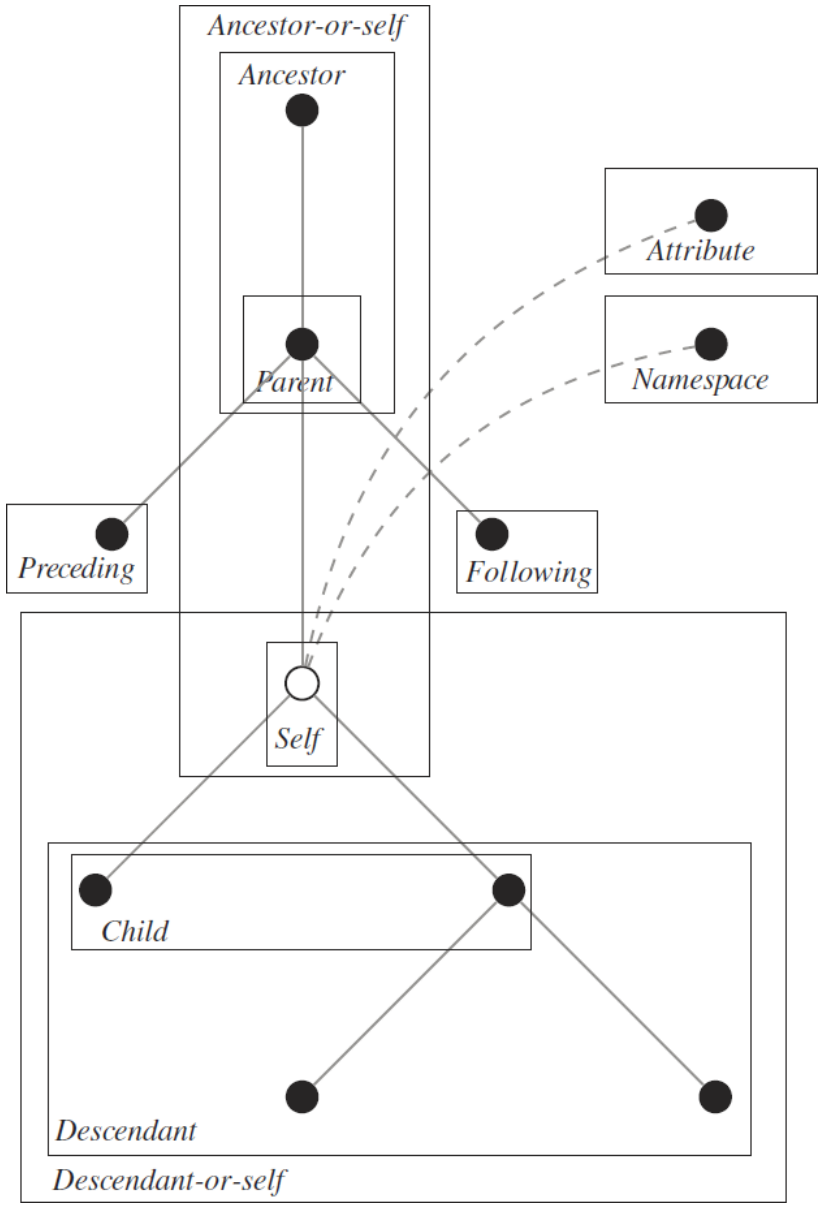
Absolute (or even relative) paths cannot always succinctly select nodes in large or complicated files.

Family tree analogy: nodes' placement in the parsed tree often mimic the relations in extended families.

Relations are denoted according to `node1/relation::node2`.

**Examples:**

- `"//a/ancestor::div"` returns all `<div>` nodes that are ancestor to an `<a>` node.
- `"//a/ancestor::div//i"` returns all `<i>` nodes contained in a `<div>` node that is an ancestor to an `<a>` node, etc.




| Axis name | Result |
|---|---|
| ancestor | Selects all ancestors (parent, grandparent, etc.) of the current node |
| ancestor-or-self | Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself |
| attribute | Selects all attributes of the current node |
| child | Selects all children of the current node |
| descendant | Selects all descendants (children, grandchildren, etc.) of the current node |
| descendant-or-self | Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself |
| following | Selects everything in the document after the closing tag of the current node |
| following-sibling | Selects all siblings after the current node |
| namespace | Selects all namespace nodes of the current node |
| parent | Selects the parent of the current node |
| preceding | Selects all nodes that appear before the current node in the document except ancestors, attribute nodes, and namespace nodes |
| preceding-sibling | Selects all siblings before the current node |
| self | Selects the current node |

# XPATH – PREDICATES

A predicate is a function that applies to a node's name, value, or attributes and that returns a logical *TRUE* or *FALSE*.

Predicates modify the path input of an XPath query. Nodes for which the relation is true are selected by the query.

Predicates are denoted by square brackets, placed after a node.

**Examples:**

- `"//p[position()=1]"` returns the first `<p>` node relative to its parent node
- `"//p[last()]"` returns the last `<p>` node relative to its parent node
- `"//div[count(./@*)>2]"` returns all `<div>` nodes with 2+ attributes, etc.

| Function | Description | Example |
|---|---|---|
| `name(<node>)` | Returns the name of `<node>` or the first node in a node set | `//*[name()='title']`; Returns: `<title>` |
| `text(<node>)` | Returns the value of `<node>` or the first node in a node set | `//*[text()='The book homepage']`; Returns: `<i>` with value *The book homepage* |
| `@attribute` | Returns the value of a node's *attribute* or of the first node in a node set | `//div[@id='R Inventor']`; Returns: `<div>` with attribute *id* value *R Inventor* |
| `string-length(str1)` | Returns the length of `str1`. If there is no string argument, it returns the length of the string value of the current node | `//h1[string-length()>11]`; Returns: `<h1>` with value *Robert Gentleman* |
| `translate(str1, str2, str3)` | Converts `str1` by replacing the characters in `str2` with the characters in `str3` | `//div[translate(./@date, '2003', '2005')='June/2005']`; Returns: first `<div>` node with date attribute value *June/2003* |
| `contains(str1,str2)` | Returns `TRUE` if `str1` contains `str2`, otherwise `FALSE` | `//div[contains(@id, 'Inventor')]`; Returns: first `<div>` node with id attribute value *R Inventor* |
| `starts-with(str1,str2)` | Returns `TRUE` if `str1` starts with `str2`, otherwise `FALSE` | `//i[starts-with(text(), 'The')]`; Returns: `<i>` with value *The book homepage* |
| `substring-before (str1,str2)` | Returns the start of `str1` before `str2` occurs in it | `//div[substring-before(@date, '/')='June']`; Returns: `<div>` with date attribute value *June/2003* |
| `substring-after (str1,str2)` | Returns the remainder of `str1` after `str2` occurs in it | `//div[substring-after(@date, '/')=2003]`; Returns: `<div>` with date attribute value *June/2003* |
| `not(arg)` | Returns `TRUE` if the boolean value is `FALSE`, and `FALSE` if the boolean value is `TRUE` | `//div[not(contains(@id, 'Inventor'))]`; Returns: the `<div>` node that does not contain the string *Inventor* in its id attribute value |
| `local-name(<node>)` | Returns the name of the current `<node>` or the first node in a node set—without the namespace prefix | `//*[local-name()='address']`; Returns: `<address>` |
| `count(<node>)` | Returns the count of a nodeset `<node>` | `//div[count(.//a)=0]`; Result: The second `<div>` with one `<a>` child |
| `position(<node>)` | Returns the index position of `<node>` that is currently being processed | `//div/p[position()=1]`; Result: The first `<p>` node in each `<div>` node |
| `last()` | Returns the number of items in the processed node list `<node>` | `//div/p[last()]`; Result: The last `<p>` node in each `<div>` node |

# UK GOV PRESS RELEASES – BACKGROUND

The United Kingdom Government publishes all of its press releases online, at gov.uk/government/announcements.

As of 29 March 2018, there were 65K+ press releases available on the site.

**Questions:**

- Can we predict which agency released an announcement based on its textual content alone?
- Are there topics that seem to return to the forefront over and over?

GOV.UK

Search

Departments Worldwide How government works Get involved
Policies Publications Consultations Statistics Announcements

# Announcements

You can use the filters to show only results that match your interests

**Contains**

keywords

**Announcement type**

All announcement types

**Policy area**

All policy areas

**Department**

All departments

**Person**

All people

**World locations**

All locations

**65,716** announcements

Get updates to this list email feed

**Welsh innovation is key to Britain's future export success**
24 March 2018 WO Speech

**Preventing Hunger as a Weapon of War**
23 March 2018 FCO Speech

**"Our vote today against this resolution is a vote against the politicization of the Commission on the Status of Women."**
23 March 2018 FCO Speech

**Lord Ahmad welcomes conclusions of the 37th Session of the UN Human Rights Council**
23 March 2018 FCO Speech

**Rt Hon Mark Field MP speech at Global FinTech Investor Forum**
23 March 2018 FCO Speech

Press release

# Foreign Secretary statement on Iran

The Foreign Secretary has made the following statement on the protests in Iran.

Published 1 January 2018
From: Foreign & Commonwealth Office and The Rt Hon Boris Johnson MP

The Foreign Secretary Boris Johnson said:

" The UK is watching events in Iran closely. We believe that there should be meaningful debate about the legitimate and important issues the protesters are raising and we look to the Iranian authorities to permit this."

" We also believe that, particularly as we enter the 70th anniversary year of the Universal Declaration on Human Rights, people should be able to have freedom of expression and to demonstrate peacefully within the law."

" We regret the loss of life that has occurred in the protests in Iran, and call on all concerned to refrain from violence and for international obligations on human rights to be observed."

## Individual press releases contain:

- title
- date of publication
- publishing organisations/individuals
- text of the release

## Focus on 2017, and releases from

- Wales Office
- Foreign Office
- Department of Science and Technology
- Department for Environment, Food & Rural Affairs

---

**Notebook:** UK Gov Press Releases

# REGULAR EXPRESSIONS

Main task in web scraping is to collect **relevant** information for the research problem from lots of textual data.

We care about systematic elements of textual data, especially if quantitative methods are eventually going to be applied.

Systematic structures can be

- numbers
- names (countries, etc.)
- addresses (mailing, e-mailing, URLs, etc.)
- specific character strings, etc.

# REGULAR EXPRESSIONS

Regular expressions (regexps) allow for the systematic extraction of the information components.

**Regexps** are abstract sequences of strings that match concrete recurring patterns in text.

Can be used to extract from plain text, HTML, and XML.

Useful when information is hidden within *atomic* values.

---

**Notebook:** Python Regular Expressions and More

# BEAUTIFUL SOUP

Simple web requests require some networking code to fetch a page and return the HTML contents.

Browsers do a lot of work to intelligently parse totally improper HTML syntax, something like:

```
<a href="crummy.com> <b>link text<a> </b>
```

**Beautiful Soup** is a Python library that helps extract data out of HTML and XML files. It parses HTML files, even if they're broken.

# BEAUTIFUL SOUP

BS does not simply convert bad HTML to good XHTML, to be parsed with an XML parser.

BS allows a user to fully inspect the (proper) HTML structure it produces, programmatically.

When BS is done its work on an HTML file, the result is an API for traversing, searching, and reading the document's elements.

# BEAUTIFUL SOUP

Typical HTML elements to be extracted/read come in various formats:

- text
- tables
- form field values
- images
- videos
- etc.

It provides **idiomatic** ways of navigating, searching, modifying the parse tree of the HTML file (it's a huge time-saver).

[Beautiful Soup documentation, https://www.crummy.com/software/BeautifulSoup/bs4/doc/]

```
html_doc = """
<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 href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
```

[Beautiful Soup documentation, https://www.crummy.com/software/BeautifulSoup/bs4/doc/]

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')

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>
```

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

soup.title.name
# u'title'

soup.title.string
# u'The Dormouse's story'

soup.title.parent.name
# u'head'

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

soup.p['class']
# u'title'

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

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>]

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

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

```
print(soup.get_text())
# The Dormouse's story
#
# 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.
#
# ...
```

# SELENIUM

**Selenium** is a tool to automate web browser interactions (in Python). It is used primarily for automating web applications for testing purposes, but it has other applications.

Mainly, it allows the user to open a browser and to perform tasks as a human being would, such as:

- clicking buttons
- entering information in forms
- searching for specific information on the web pages
- etc.

# SELENIUM

Selenium requires a driver to interface with the chosen browser. Firefox, for example, requires *geckodriver*.

Other supported browsers will have their own drivers available:

| | |
|---|---|
| Chrome: | https://sites.google.com/a/chromium.org/chromedriver/downloads |
| Edge: | https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/ |
| Firefox: | https://github.com/mozilla/geckodriver/releases |
| Safari: | https://webkit.org/blog/6900/webdriver-support-in-safari-10/ |

# SIMULATING A WEB BROWSER

Selenium automatically controls a complete browser, including rendering the web documents and running JavaScript.

This is useful for pages with a lot of dynamic content that isn't in the base HTML.

Selenium can program actions like "click on this button", or "type this text", and at any point you have access to the dynamic HTML of the current state of the page, like what you see in Developer Tools.

# USING APIS

An API is a website's way of giving programs access to their data, without the need for scraping.

That is, an API provides **structured access** to **structured data**.

For example, a finance site might offer an API with financial data, or the *New York Times* might offer an API for news articles.

In either case, the data will be in a pre-defined, structured format (often JSON).

# USING APIS

The APIs we'll consider have R/Python libraries that encapsulate all required networking and encoding.

This means that it suffices to read the library documentation to know what to do.

---

Exercise: Use Zomato to find which Canadian city has the best sushi restaurants (https://github.com/fatihsucu/pyzomato).

# YOUTUBE API – KHAN ACADEMY

Millions of videos are available through **YouTube**.

It's not obvious how one would scrape video content off the web in general (other than the URLs); some videos have associated text content (**transcripts**).

We use the YouTube API to scrape that content.

---

**Notebook:** YouTube Transcripts

[https://www.youtube.com/playlist?list=PL1328115D3D8A2566]

YouTube CA

Search

SIGN IN

- Home
- Trending
- History

BEST OF YOUTUBE

- Music
- Sports
- Gaming
- Movies
- TV Shows
- News
- Live
- 360° Video

Browse channels

Sign in now to see your channels and recommendations!

SIGN IN


# Statistics

68 videos • 3,290,303 views • Last updated on Jul 2, 2014

Khan Academy

SUBSCRIBE

Introduction to statistics. Will eventually cover all of the major topics in a first-year statistics course (not there yet!)

1 

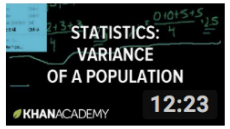

**Statistics: The average | Descriptive statistics | Probability and Statistics | Khan Academy**

Khan Academy

2 

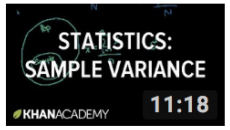

**Statistics: Sample vs. Population Mean**

Khan Academy

3 

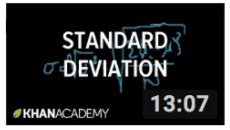

**Statistics: Variance of a population | Probability and Statistics | Khan Academy**

Khan Academy

4 


**Statistics: Sample variance | Descriptive statistics | Probability and Statistics | Khan Academy**

Khan Academy

5 

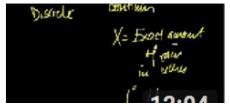

**Statistics: Standard deviation | Descriptive statistics | Probability and Statistics | Khan Academy**

Khan Academy

6 


**Statistics: Alternate variance formulas | Probability and Statistics | Khan Academy**

Khan Academy

7 

**Introduction to Random Variables**

Khan Academy