Web scraping and social media scraping – selectors

Jacek Lewkowicz, Dorota Celińska-Kopczyńska

University of Warsaw

March 12, 2019

Procedure of scraping

- Identify the information
- Choose an appropriate strategy
- Make the data retrieval
- Extract the information
- Prepare the data
- Validate the data
- Do some debugging and maintenance
- Prepare generalization

Before we start coding...

- Viewing a page in a web browser usually requires downloading content encoded in HTML
- The most common task during scraping is extracting data from HTML source
- HTML is a markup language, a particular "dialect" of XML
- Markup languages use set of tags or rules to organize and provide information about their content
- Summing it up, you will need a tool to quickly address parts of the HTML/XML documents!

Selectors

- Selectors "select" certain parts of the HTML document either by XPath or CSS expressions
- XPath is a language for getting content of XML documents
- CSS is a language for applying styles to HTML websites

Why learn XPath or XML/HTML?

- You will be able to navigate everywhere inside the XML tree
- It is a must-have skill for accurate web data extraction
- XPaths are usually more powerful than CSS selectors

Why should I care?

- It is true, that for the most of the practical purposes you will use built-in tools for retrieving XPaths or CSS selectors...
- However, those automatic tools have limited possibilities. If you want to make a more sophisticated query, you will have to write it yourself
- Also, sometimes the returned XPaths for some reasons do not work... and you have to rewrite them (*there are more ways to do something!*)
- Other options include struggling with later data processing or even giving up on getting some sort of data

XML – overview

- **Ex**tensible **M**arkup **L**anguage
- Extensible means that the tags are not predefined – you can (actually have to) define your own
- It is a way to store and organize data, irrespectible of how it will be presented
- It is a public, open standard developed by World Wide Web Consortium (W3C)
- **It is not a programming language** – it does not allow for performing computations or writing algorithms!

XML – structure

- XML document is structured using **nodes** which include **element** nodes, **attribute** nodes and **text** nodes
- Element nodes must have an opening and closing tags, e.g. `<tag>` opening tag and `</tag>` closing tag. Everything between those tags is contained within the element
- The elements have to be properly nested, i.e., all opened tags must be closed!

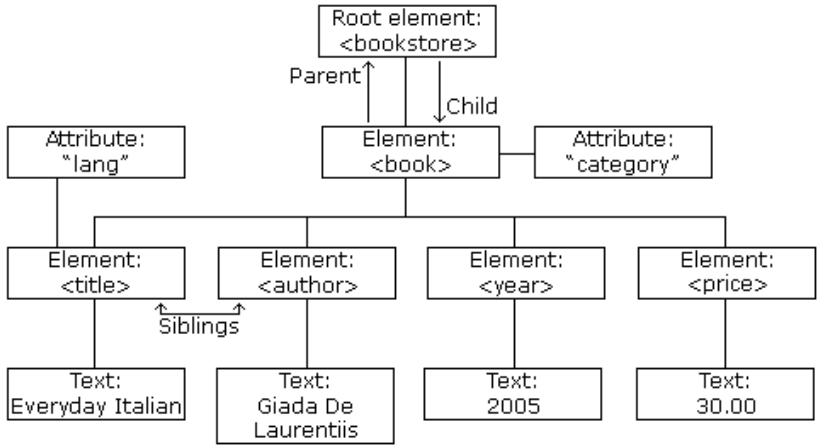
XML – naming rules

- Element names are case sensitive: `<sometag>` is not the same as `<SomeTag>`
- Element names must start with a letter or underscore (but cannot start with string "xml")
- Element names can contain letters, digits, hyphens, underscores, periods, but cannot contain spaces
- Attribute values must be quoted

Note: In HTML (XML dialect) some of the rules are relaxed!

XML tree – basic definitions

- XML documents form a tree structure which starts at the **root** and branches to **leaves**. Leaves are **child elements** of root. All elements can have child elements
- Terms **parent**, **child** and **sibling** describe the relationships among elements
- Children are called **descendants** and parents are called **ancestors**



Source: http://www.w3schools.com/xml/xml_tree.asp

XML tree – code

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="fantasy">
    <title lang="en">Silmarillion</title>
    <author>J. R. R. Tolkien</author>
    <year>2005</year>
    <price>23.99</price>
  </book>
  <book category="science">
    <title lang="pl">Ekonometria</title>
    <author>Jerzy Mycielski</author>
    <year>2010</year>
    <price>20.00</price>
  </book>
</bookstore>
```

XML tree – additional remarks

- Every node has exactly one parent, **except the root**
- Attribute and text nodes have no children
- Attributes cannot contain multiple values or tree structures (elements can)
- **Path** is the sequence of connections from node to node

XPath – Intuition

- Using XPath is similar to searching in a library catalogue, however we are allowed to state which metadata fields to specify. Another example is a traditional computer file system
- While using XPath we do not have to know in advance the exact form of the content
- It is a kind of a search query: we provide an instruction **how to get to the element**

There are many ways to retrieve information

- Imagine you stand at the Ratusz Arsenał underground station and ask how to get to FES UW
- **XPath**: go down the Długa street, pass the next two buildings (including Museum of Archeology) and FES UW will be the next building
- **Other methods, e.g., regexp**: Look for the pink building in Długa street
- Note: each method has its pros and cons! FES UW is one building, but think about the Central Campus (a few similarly looking buildings). Or what if there are more pink buildings over there?

XPath – terminology

- There are seven kinds of nodes: **element**, **attribute**, **text**, **namespace**, **processing-instruction**, **comment** and **document**
- **Atomic values** are nodes with no children or parent
- **Items** are atomic values or nodes
- *Family-like* terminology of XML trees also applies here

Location paths

- The most common XPath expression
- Used to move in any direction from a starting point (**the context node**) to any node(s) in the tree
- A string with a series of "steps": `step 1 / step 2 / step 3`
- Processed from left to right, whitespace does not matter

Selecting nodes

- XPath uses path expressions to select nodes in an XML document
- The most useful path expressions:

nodename	selects all nodes with the name *nodename*
/	selects from the root node
//	selects nodes in the document from the current node that match the selection no matter where they are (ability to skip levels)
.	selects the current node
..	selects the parent of the current node
@	selects attributes

Selecting nodes – examples

`book`	selects all nodes with the name book
`/book`	selects the root element book – if the path starts with a slash (/) it always represents an absolute path to an element!
`//author`	selects all nodes with name author no matter where they are
`//author/..`	selects all of the parent elements of the nodes with name author no matter where they are
`/author/book/title/@ed`	selects all values of attributes named ed of all of the titles of the books written by all authors in the database

Node tests – Wildcards

- We may select different **node types** along the axes
- We use wildcards to select unknown XML nodes

`*`	matches any element node
`@*`	Matches any attribute node
`node()`	matches any node of any kind
`text()`	matches text nodes
`comment()`	matches comment nodes

Wildcards – examples

/book/*	Selects all child element nodes of the book element
//*	Selects all elements in the document
//title[@*]	Selects all title elements that have at least one attribute of any kind

Predicates – operators

- We may use predicates to find a specific node or a node that contains a specific value
- Contained within []

=	Equivalent comparison, can be used for numeric or text values
!=	Not equivalent comparison
>, >=	Greater than, greater than or equal to
<, <=	Less than, less than or equal to
or	Boolean or
and	Boolean and
not	Boolean not

Predicates – examples

/bookstore/book[1]	selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	selects the last book element that is the child of the bookstore element
/bookstore/book[position()<3]	selects the first two book elements that are children of the bookstore element
//title[@lang]	selects all the title elements that have an attribute named lang
//title[@lang='en']	selects all the title elements that have a lang attribute with a value of "en"
/bookstore/book[price>42.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 42.00
/bookstore/book[price>42.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 42.00

Source: Adapted from http://www.w3schools.com/xml/xpath_syntax.asp

Selecting several paths

- You can select several paths with | operator

`//book/title | //book/year`	Selects all title or year elements which are children of the nodes book no matter where they are

In-text search

- We can perform in-text searching with built-in functions

`node[contains(@name, value)]`	Matches on all nodes node that contain in the argument name given value value
`node[starts-with(.,''A'')]`	Matches on all nodes node, in current node starts with character A (case sensitive)
`node[ends-with(text(),''a'')]`	Matches on all nodes node which text() element ends with character a (case sensitive)

How to use built-in tools in web browsers?

- Start with viewing the source code of the web page or "inspecting element"
- In Chrome/Chromium: inspect element / find the element you are interested in / right click on it / copy XPath
- In Mozilla Firefox: Inspect Element (open the console) / right click on interesting element / copy xpath (you will probably have to install an add-on for XPaths)

Translations from Polish for the classroom software

Polish	English
Zbadaj element	Inspect element
Kopiuj	Copy
Selektor CSS	Selector CSS
Ścieżka CSS	CSS Path
Zbadaj	Inspect element
Pokaż źródło strony	View page source