Data collection

Goals

- **Learn to automatically download multiple web pages with the command line**
- Understand how HTML data is structured
  - Know what are markup languages
  - Know the data model behind XML
  - Understand the XML syntax
  - Gain familiarity with the most common elements and attributes
- Retrieve pertinent information from HTML files and create a datasheet with this cleaned content
- Learn the basics of regular expressions and how to use them to clean data

Web scraping

Web scraping is a technique to retrieve data from websites:

- Fetch the webpage
- Extract the pertinent data

Fetching websites

Web crawlers. See Wikipedia article for a list with several software proposals.

In this session we will use:

- Command `curl`
- Open Refine

How to install

- Depending on your OS, cURL might already be installed (e.g. Ubuntu or Windows version 1803 or later). There are system-specific instructions on how to install it depending on your OS (see for example: https://help.ubidots.com/en/articles/2165289-learn-how-to-install-run-curl-on-windows-macosx-linux)
- OpenRefine can be downloaded from https://openrefine.org/download.html (for installation instructions see: https://docs.openrefine.org/manual/installing)

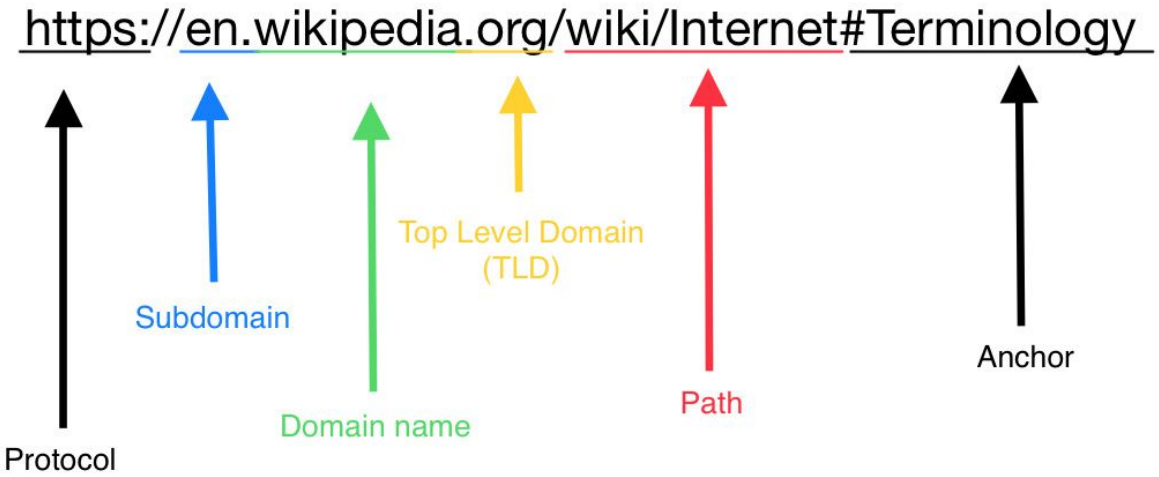
Test

- To see if cURL is properly installed, open the terminal and run:

  ```
  curl --help
  ```

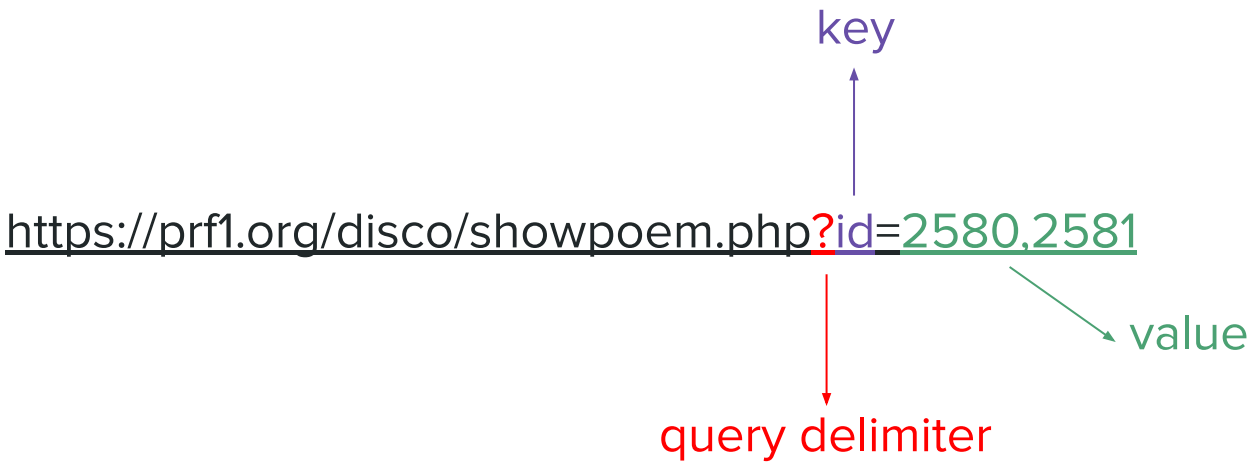
- To check that OpenRefine is running, you can access the application via your browser at http://127.0.0.1:3333/

URL structure



URL structure by Noémie2602 (link)

URLs that contain queries



Fetching data with the command `curl`

- Useful flags:
  - `-f, --fail` If the server returns an error, *curl* fails silently and returns error 22. Example: `curl -f https://example.com`
  - `-L, --location` Allow curl to follow any redirections. Example: `curl -L https://example.com`
  - `-O, --remote-name` Specify that the local file should have the name of the remote file that was downloaded. Example: `curl -O https://example.com/filename`
  - `-o, --output <file>` Store output in a file. Example: `curl -o file https://example.com -o file2 https://example.net`
- To see all flags: `curl --help`

cURL syntax

- Multiple URLs that differ in one part are written together using braces. E.g.:

  ```
  http://example.{first,second,third}.com
  ```

- Alphanumeric series are written with brackets. E.g.:

  ```
  ftp://ftp.url.com/file[1-100].txt
  ```

- Multiple sequences are allowed. E.g.:

  ```
  http://url.com/archive[010-020]/vol[1-4]/part{a,b}.html
  ```

Exercise

Download a multi-chaptered work from:
https://www.corpusthomisticum.org/iopera.html

Solution example

```
curl -fLO https://www.corpusthomisticum.org/sth0000.html
-fLO https://www.corpusthomisticum.org/sth[1001-4084].html
```

Goals

- Learn to automatically download multiple web pages with the command line
- **Understand how HTML data is structured**
  - **Know what are markup languages**
  - **Know the data model behind XML**
  - **Understand the XML syntax**
  - **Gain familiarity with the most common elements and attributes**
- Retrieve pertinent information from HTML files and create a datasheet with this cleaned content
- Learn the basics of regular expressions and how to use them to clean data

HTML

- *Hypertext Markup Language*: one of the languages used to create websites. The HTML elements define the structure, the links and the metadata of the website.

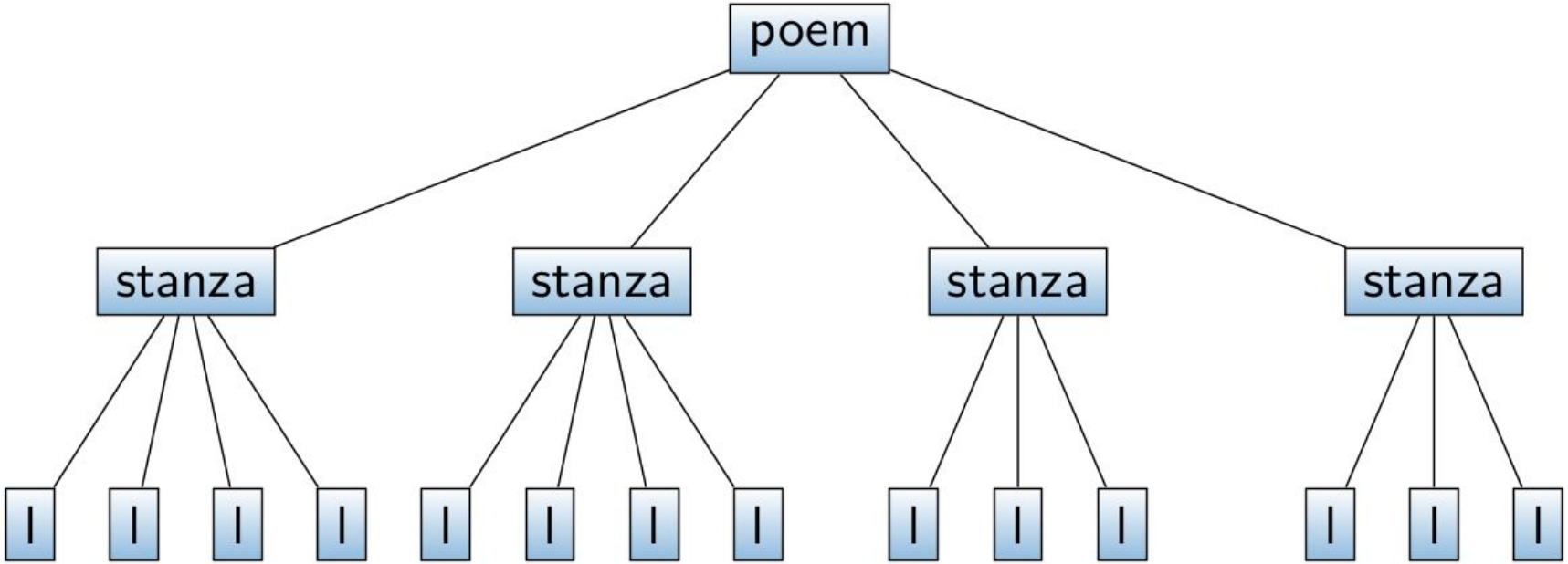
HTML

- *Hypertext* ***Markup Language***: one of the languages used to create websites. The **HTML elements** define the structure, the links and the metadata of the website.

Markup entails...

... modelling the inherent structure of a text and its semantic properties though:

- Hierarchies
- Ordered structures
- Human language + computational language


poem
stanza
stanza
stanza
stanza
l
l
l
l
l
l
l
l
l
l
l
l
l
l
l

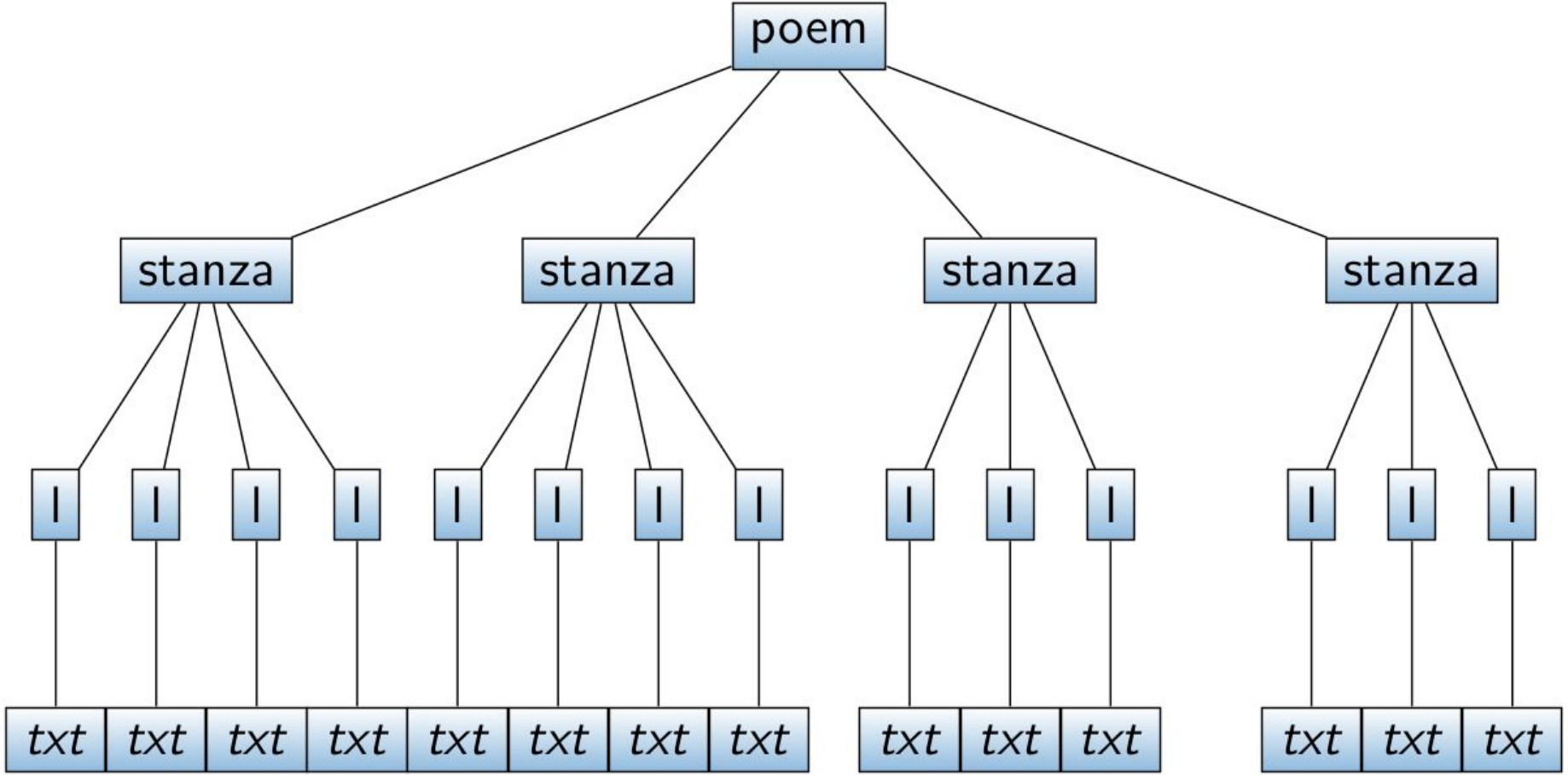
Types of markup

- Descriptive
- Presentational
- Procedural
- **Multipurposing**

What is XML?

- Format for the storage and transmission of data
- Defined by the World Wide Web Consortium (W3C)
- Very extensive use

```
<poem>
    <stanza>
        <l>Valenz Senher, rei dels Aragones</l>
        <l>a qi prez es honors tut iorn enansa,</l>
        <l>remembre vus, Senher, del Rei franzes</l>
        <l>qe vus venc a vezer e laiset Fransa</l>
    </stanza>
    <stanza>
        <l>Ab dos sos fillz es ab aqel d'Artes;</l>
        <l>hanc no fes colp d'espaza ni de lansa</l>
        <l>e mainz baros menet de lur paes:</l>
        <l>jorn de lur vida said n'auran menbransa.</l>
    </stanza>
    <stanza>
        <l>Vostre Senhier faccia a vus compagna</l>
        <l>per qe en ren no vus qal[la] duptar;</l>
        <l>tals quida hom qe perda qe gazaingna.</l>
    </stanza>
    <stanza>
        <l>Seigner es de la terra e de la mar,</l>
        <l>per qe lo Rei Engles e sel d'Espangna</l>
        <l>ne varran mais, si.ls vorres aiudar.</l>
    </stanza>
</poem>
```


poem
stanza
stanza
stanza
stanza
l
txt

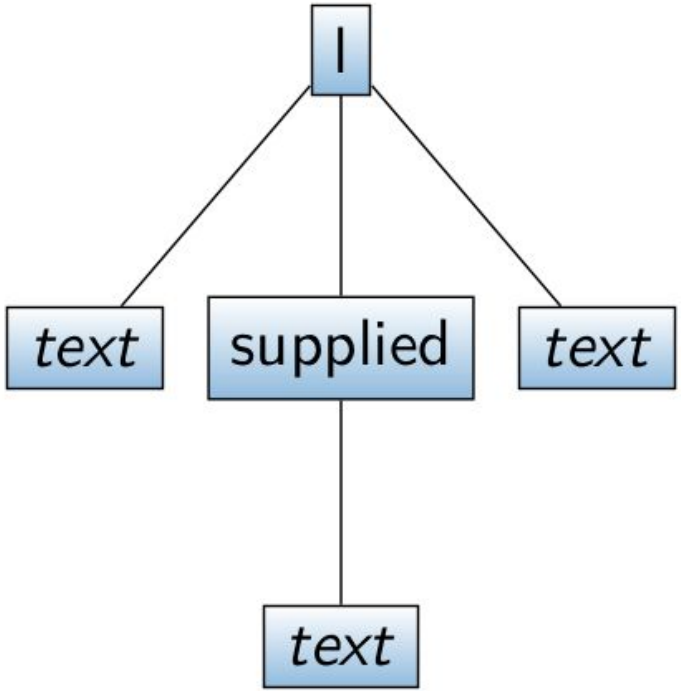
XML syntax

- Element: `<l> </l>`
- Attribute: `<l met="3" rhyme="a">`
- Textual nodes:

```
<l met="3" rhyme="a">Frutales</l>

<l met="3" rhyme="b">cargados.</l>

<l met="3" rhyme="b">Dorados</l>

<l met="3" rhyme="a">trigales…</l>
```

```
<poem>
    <stanza type="quatrain">
        <l>Valenz Senher, rei dels Aragones</l>
        <l>a qi prez es honors tut iorn enansa,</l>
        <l>remembre vus, Senher, del Rei franzes</l>
        <l>qe vus venc a vezer e laiset Fransa</l>
    </stanza>
    <stanza type="quatrain">
        <l>Ab dos sos fillz es ab aqel d'Artes;</l>
        <l>hanc no fes colp d'espaza ni de lansa</l>
        <l>e mainz baros menet de lur paes:</l>
        <l>jorn de lur vida said n'auran menbransa.</l>
    </stanza>
    <stanza type="tercet">
        <l>Vostre Senhier faccia a vus compagna</l>
        <l>per qe en ren no vus qal[la] duptar;</l>
        <l>tals quida hom qe perda qe gazaingna.</l>
    </stanza>
    <stanza type="tercet">
        <l>Seigner es de la terra e de la mar,</l>
        <l>per qe lo Rei Engles e sel d'Espangna</l>
        <l>ne varran mais, si.ls vorres aiudar.</l>
    </stanza>
</poem>
```

```
<poem>
    <stanza type="quatrain">
        <l>Valenz Senher, rei dels Aragones</l>
        <l>a qi prez es honors tut iorn enansa,</l>
        <l>remembre vus, Senher, del Rei franzes</l>
        <l>qe vus venc a vezer e laiset Fransa</l>
    </stanza>
    <stanza type="quatrain">
        <l>Ab dos sos fillz es ab aqel d'Artes;</l>
        <l>hanc no fes colp d'espaza ni de lansa</l>
        <l>e mainz baros menet de lur paes:</l>
        <l>jorn de lur vida said n'auran menbransa.</l>
    </stanza>
    <stanza type="tercet">
        <l>Vostre Senhier faccia a vus compagna</l>
        <l>per qe en ren no vus qal<supplied>la</supplied>
            duptar;</l>
        <l>tals quida hom qe perda qe gazaingna.</l>
    </stanza>
    <stanza type="tercet">
        <l>Seigner es de la terra e de la mar,</l>
        <l>per qe lo Rei Engles e sel d'Espangna</l>
        <l>ne varran mais, si.ls vorres aiudar.</l>
    </stanza>
</poem>
```


I
text
supplied
text
text

XML syntax

- The XML tree has a single root, that is, a single element that contains all other elements
- All contents are delimited
- It cannot contains the characters `&`, `<` (replace with the corresponding entities `&` and `<`)
- All elements must be properly nested: no overlaps!

Why XML?

- Easy and simple syntax
- Readable
- Control over input and output
- Software and hardware independent
- Supported by a wide range of software (open + proprietary)

The XML family of standards

- Schema languages
- XPath
- XSLT
- XQuery
- SVG
- **HTML**
- Schematron
- KML
- XSL-FO, XForms, XProc, OOXML, OpenOffice.orgXML

Skeleton of an HTML document

```
<!DOCTYPE html>
<html>
    <head>
        <title>Title of the web page</title>
    </head>
    <body></body>
</html>
```

Most frequent elements

- Paragraphs: **<p>**
- Headings, with different hierarchies: **<h1>**, **<h2>**, **<h3>**, **<h4>**...
- Unordered lists (list items will be marked with bullets by default):

```
<ul>
    <li>A list item</li>
    <li>Another list item</li>
</ul>
```

- Ordered lists (list items will be marked with numbers by default):

```
    <ol>
        <li>First list item</li>
        <li>Second list item</li>
    </ol>
```

Most frequent elements

- Links: `<a href="url">Text of the link</a>`
- Emphasis (by default, browsers present its contents in italics): `<em>`
- Strong emphasis (by default, browsers present its contents in bold): `<strong>`
- Generic division (block): `<div>`
- Generic division (inline): `<span>`
- Elements to semantically differentiate sections (see this post): `<main>`, `<header>`, `<footer>`, `<article>`, `<section>`, `<nav>`, `<aside>`
- Figures and images: `<figure>` (and `<figcaption>`) and `<img>`

Very brief task

1. Go to the folder “Exemplars” and open with any browser: `html_example.html`
2. Click `Ctrl + U`
3. Go the tab with the html file (not with the source) and right click on it. Select the option “Inspect”

To learn more

- W3C HTML Tutorial (every section has a “Try it Yourself” functionality that allows you to test and manipulate HTML snippets to further understand how each element and attribute work)

Related resources

- W3C Markup Validation Service (to validade websites/HTML files)

Goals

- Learn to automatically download multiple web pages with the command line
- Understand how HTML data is structured
  - Know what are markup languages
  - Know the data model behind XML
  - Understand the XML syntax
  - Gain familiarity with the most common elements and attributes
- **Retrieve pertinent information from HTML files and create a datasheet with this cleaned content**
- Learn the basics of regular expressions and how to use them to clean data

Scraping with OpenRefine

- Task: create a corpus with these four essays taken from WikiSource:
  - https://en.wikisource.org/wiki/Some_Account_of_a_Proposed_New_College_for_Women
  - https://en.wikisource.org/wiki/The_Criterion/Volume_4/Number_1/On_Being_Ill
  - https://en.wikisource.org/wiki/Mrs_Dalloway_in_Bond_Street
  - https://en.wikisource.org/wiki/Militant_Pacifism

Brief introduction to GREL

- General Refine Expression Language. To learn more see: https://openrefine.org/docs/manual/grel
- To retrieve pertinent information, we will look into the source (the HTML file) and select the element that contains the relevant data. We will be using expressions similar to this one:
  `value.parseHtml().select("span[id=ws-author]")[0].htmlText()`
  - `value`: variable to refer to the source column (from where we extract the information)
  - `parseHtml()`: function to create a tree out of the string contained in the source column
  - `select()`: this function navigates the three and selects the content expressed within the parenthesis

Brief introduction to GREL

- `value.parseHtml().select("span[id=ws-author]")[0].htmlText()`
  - `span[id=ws-author]`: value of the `select()` function. It looks for the `<span>` elements that contain and `@id` attribute with the value `“ws-author”`. Note that the syntax is as follows: `elementName[attributeName=attributeValue]`
  - `[0]`: Numeric filter (this languages starts counting at 0). This means we select the first `<span>` element with the specified attribute.
  - `htmlText()`: Function that retrieves the textual content of the element

Brief introduction to GREL

- `forEach(value.parseHtml().select("span[id=ws-author]"), author, author.htmlText()).join(", ")`: This function has the following syntax: `forEach(sequence, variableName, expression)`

  The function iterates over each one of the `<span>` elements with that specific attribute-value (instead of getting the first one as before): this is our sequence. Then we give the name of the variable (`author`) thus the word “author” is used to designate each of the iterations. Then, for each occurrence of author, we retrieve the textual content.

  After the results of the function, `join(", ")` concatenates all the authors using the comma followed by a space delimiter.

Instructions: create a project

1. Open and run Open Refine. You should a browser opened at http://127.0.0.1:3333/
2. Copy the three URLs of the previous slides and paste its contents in Create project > Clipboard
3. Make sure that there is no whitespace before and after the url and Click “next”
4. After loading, in the menu below select “Line-based text files”
5. Give a name to the project in the top menu (e.g. Woolf) and click on “Create project”

Instructions: retrieve the name of the author

1. Open the dropdown menu of the column that contains the URLs (likely named “Column 1”)
2. Select Edit column > Add column by fetching URLs
3. In the pop-up dialogue, give a name to the column (e.g. “HTML source”) and select OK
4. Open the dropdown menu of this new column and select Edit column > Add column based on this column
5. Give a name to the new column, e.g. “Author”
6. In the Expression box, get the content of the element that contains the author of the work, this is:
   `value.parseHtml().select("span[id=ws-author]")[0].htmlText()`

Instructions: retrieve the title

1. As to create the “Author” column, open the dropdown menu of the column “HTML source” and select Edit column > Add column based on this column
2. Give a name to the new column, e.g. “Title”
3. In the Expression box, get the content of the element that contains the title of the work:
   `value.parseHtml().select("span[id=ws-title]")[0].htmlText()`

Instructions: retrieve the text

1. As to create the “Author” column, open the dropdown menu of the column “HTML source” and select Edit column > Add column based on this column
2. Give a name to the new column, e.g. “Text”
3. In the Expression box, get the content of each paragraph of the text and create a new line to separate the paragraphs:

```
forEach(value.parseHtml().select("div[class=prp-pages-output
]")[0].select("p"), paragraph,
paragraph.htmlText()).join("\n")
```

Goals

- Learn to automatically download multiple web pages with the command line
- Understand how HTML data is structured
  - Know what are markup languages
  - Know the data model behind XML
  - Understand the XML syntax
  - Gain familiarity with the most common elements and attributes
- Retrieve pertinent information from HTML files and create a datasheet with this cleaned content
- **Learn the basics of regular expressions and how to use them to clean data**

Required software

Visual Studio Code

What are regular expressions?

- A regular expression is a **pattern** describing a certain amount of text
- Many applications and programming languages have their own implementation of regular expressions
- Some resources:
  - Regex tester: https://www.regexpal.com
  - Regex tutorial: https://www.regular-expressions.info/quickstart.html

Literal characters

- The most basic regular expression consists of a single literal character
- It matches the first occurrence of that character in the string
- E.g.: **a**

Special characters

- the question mark `?`, the asterisk or star `*`, the plus sign `+`
- the backslash `\`, the caret `^`, the dollar sign `$`, the period or dot `.`, the vertical bar or pipe symbol `|`
- the opening parenthesis `(`, the closing parenthesis `)`, the opening square bracket `[`, and the opening curly brace `{`

Special characters are escaped with the backslash `\` (see this slide)

Classes

- A “character class” matches only one out of several characters
- Classes are enclosed in square brackets
- E.g.: `reali[sz]e`

Ranges

- You can express numeric or alphabetic ranges with character groups. They are enclosed in square brackets and the hyphen expresses the range.
- E.g.
  - `[1-4]`
  - `[0-3][0-9]`
  - `[A-Za-z]`

Shorthand character classes

- `\s` = white space
- `\w` = alphanumeric
- `\d` = digit

Negating a class

- Typing a caret after the opening square bracket negates the character class
- E.g.:
  - `q[^u]`
  - `[^\w]`
  - `[\W]`

Non-printable characters

- `\t` = tab character
- `\r` = carriage return
- `\n` = line feed

The dot

- The dot matches a single character, except line break characters
- Most applications have a “dot matches all”
- E.g.: `reali.e`

Anchors

- `^` = matches at the start of the string
- `$` = matches at the end of the string
- E.g.: `^\d`

Alternation

- `|` = regular expression equivalent of “or”
- E.g.:
  - **`British|American English`** (will find occurrences of the string “English” and the string “American English”
  - **`(British|American) English`** (will find occurrences of the string “British English” and the string “American English”

Repetition

- `?` = makes the preceding token in the regular expression optional
- `*` = matches the preceding token zero or more times
- `{n}` = specific amount of repetition
- E.g.:
  - `favou?r` (will find occurrences of both “favor” and “favour”
  - `love.*` (will find occurrences of, for example, “love”, “loves”, “lovers”)
  - `\d{4}` (will find occurrences of four digits, as in “2023”)
  - `\d{2,4}` (will find occurrences of two, three and four digits (the comma thus establishes a range)

Grouping and capturing

- We can place parentheses around multiple tokens to group them together
- Quantifiers can be applied to these groups
- `$x` = backreference capturing group number *x* (depending on the regex interpreter, a backslash could be used instead)
- E.g.:
  - Find: `(\d{2})-(\d{2})-(\d{4})`
  - Replace: `$3-$2-$1`

The backslash

- `\` = scapes a character, that is, the following character works as a literal
- E.g.: `\(sic\)` = finds occurrences of the string “(sic)”

Tips

- `\n\n` = matches two consecutive newline characters, that is, locates blank lines
- `^.*$` = matches an entire line from initial to final character
- `^string.*$` and `^.*string$` = matches a line beginning or ending, respectively, with a specific string *string*
- `".*?"` = matches everything inside a pair of quotation marks
- `\d{1,2}[./-]\d{1,2}[./-]\d{2,4}` = matches dates in the *dd-mm-yyyy* or *mm-dd-yyyy* format, allowing single- or double-digit numbers for days and months, and numeric strings of length two to four for the year. The delimiter can be a period (“.”), forward slash (“/”), or dash (“-”).

Practical exercises

In the folder “regex” you will find the different texts mentioned in this exercises.

Task 1: In the file `social-significance-drama_emma-goldman.txt`, there is pseudo-markup to indicate certain formatting information. In particular, underscores are used to delimit strings in italics (e.g. `_string_`). Semantically, we can see that there are three main uses of italics: character names, emphasis, foreign words.

1. Look for strings delimited by underscores that contain accents, to find (at least some of) the foreign words. Replace the underscores of these words with `{string}`
2. Look for strings delimited by underscores that begin with a capital letter to detect the characters’ names. Replace the underscores of these words with angle brackets `<string>`

Practical exercises

Task 2: In the file `woman-church-state_matilda-gage.txt`, replace all the footnotes references (numbers between square brackets, e.g. `[8]`).

Task 3: In the file `emmeline-orphan-castle_charlotte-smith.txt`, replace all the chapter headings (e.g. `CHAPTER I`) with just the Roman numeral between square brackets (this is, `[I]`)

Solutions

Solutions

- Task 1:
  - Find: `_(.*[áéóíúâêôîûàèòìùëïüçñ].*?)_`
  - Replace: `{$1}`
- Task 2:
  - Find: `\[\d+\]`
  - Replace:
- Task 3:
  - Find: `^\s*CHAPTER\s+([IVXLCDM]+)\s*$`
  - Replace: `[$1]`