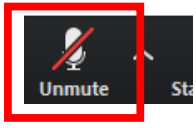
Workshop 4

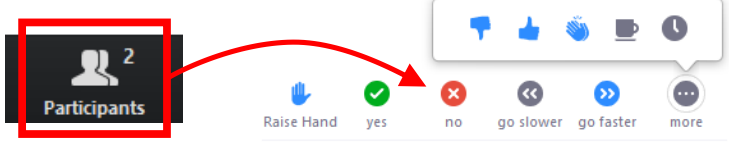
COMP20008 Elements of Data Processing

Zoom in workshops

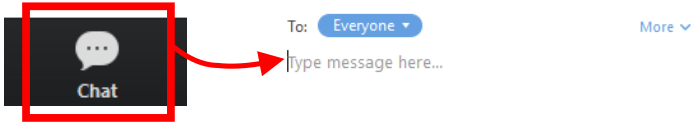
Ensure your microphone is muted unless you have the floor



Make use of non-verbal feedback (e.g. raise hand)



Use chat to talk to me (or others) without interrupting the class



Learning outcomes

By the end of this class, you should be able to:

- use regular expressions for text processing
- scrape tabular data from a webpage
- implement a basic web crawler
- apply stemming and lemmatization for natural language text processing

Regular expressions

Regular expressions in Python

- Supported by the `re` module in the Python standard library
- Documentation: docs.python.org/library/re
- Uses Perl syntax (just be aware that there are others)

Function	Description
`re.search(pattern, string)`	Find first occurrence of a matching pattern in `string`
`re.findall(pattern, string)`	Return all occurrences of a matching pattern in `string` as a list of strings
`re.sub(pattern, repl, string, count=0)`	Replace all occurrences of a matching pattern in `string` by `repl`, and return the result

Regular expressions in Python

Metacharacter	Description
`^`	Matches start of string
`$`	Matched end of string
`.`	Matches any character (except new line)
`*`	Match zero or more repetitions of preceding RE
`+`	Match one or more repetitions of preceding RE
`?`	Matches zero or one repetitions of preceding RE
`{m,n}`	Matches at least m, but not more than n repetitions of the preceding RE
`\`	Escape special characters
`[]`	Define a set of characters
`()`	Define a subexpression/block/group
`|`	Matches the RE before or after

Character class	Description
`\d`	Matches a digit
`\D`	Matches a non-digit
`\w`	Matches a “word” character
`\W`	Matches a “non-word” character
`\s`	Matches whitespace characters
`\S`	Matches non-whitespace characters

Web scraping

Scraping tabular data from HTML

HTML snippet:

```
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Corresponding table:

Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94

HTML tags:

- `table`: a table
- `tr`: table row
- `th`: header table cell
- `td`: standard table cell

Scraping tabular data from HTML

Workflow:

1. View the source code in a web browser
2. Identify where the table occurs in the HTML document tree (e.g. look for an `id` attribute)
3. Extract the `table` element (using `bs4`)
4. Nested for loop: iterate over the rows and cells within each row

BeautifulSoup (bs4)

- A library for extracting data from HTML and XML files
- Traverse the underlying document tree (similar to `lxml`)
- Documentation: www.crummy.com/software/BeautifulSoup/doc/

Method	Description
`el.find_all()`	Look in the children of this element, and return a list of elements that match the given criteria
`el.name`	Get the name of this element
`el.string`	Get the text/string content of this element
`el['attrname']`	Access an attribute of this element (like a dictionary)

Web crawling

Basic web crawling in Python

Recommended libraries:

- `requests`: for sending HTTP requests
- `bs4`: for finding links in HTML
- `urllib.parse`: for parsing links

Pseudocode

Initialize a queue of URLs (seeds)

While queue is full:

1. Get a URL from the queue
2. If the page can be crawled, fetch the page
3. Extract URLs from the page and add them to the queue

Text processing

Stemming and lemmatization

Stemming

- Crude process: chop off the end of words
- Example:
  - “picture” → “pictur”
  - “pictured” → “pictur”
  - “picturing” → “pictur”
- Porter stemmer available under `nltk.stem`

Lemmatization

- Return the base or dictionary form of a word
- Example:
  - “octopi” → “octopus”
  - “better” → “good”
- Wordnet lemmatizer available under `nltk.stem`