# Example of the use of different languages

Saul SL

July 2025

## Contents



## 1 Environment setup

For the example described in the next section, a number of packages are needed thus, it may be better to create a virtual environment where these can be installed. Snippet 1 list all the commands for setting up the environment.

```
conda create --name webscrap -y
conda activate webscrap

# Install python libraries
conda install anaconda::pandas
conda install anaconda::bs4
conda install conda-forge::lxml
pip install requests

# Install R
conda install r::r
```

Figure 1: **Code to create a new virtual environment in Anaconda.** Then, on an R session install the required R packages

```
install-packages("dplyr")
```

Figure 2: **Code to install the required R package to process the dataframe.**

## 2 Extract a table from a website and process it

Having all the dependencies installed we can write all the code blocks required for the example. This will attempt to extract a table of all supported-languages in org-mode that work out-of-the-box, check if specific languages are supported and then print the information in a table. To this end, two code blocks and a call for the results are written. The first one (written in Python) scraps the relevant community documentation web page, extract the table that lists all the supported languages, transform it into a pandas' data frame and return this object. The second code block, (written in R), will import the data frame, filter it for a selected set of languages and rename the first column. Finally, the selected table will be printed with a caption that informs about the selection process.

The code below will process the table of supported-languages in Emacs org-mode, note that it is necessary to return the object of interest in the last line for it to be exported and processed by the next code block.

```
import pandas as pd
from bs4 import BeautifulSoup
import requests
from io import StringIO

# URL of the web page containing the table
url = 'https://orgmode.org/worg/org-contrib/babel/languages/'

# Send a GET request to the URL
response = requests.get(url)

# Parse the HTML content of the web page using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Find the table element
table_html = soup.find('table')
```

```
tb_languages = pd.read_html(StringIO(str(table_html)))[0]
tb_languages
```

Figure 3: **Python code to retrieve a web page, extract and process a table within.**

The code below will use the exported object by storing it in a custom variable (`df`) and select if a there is support for 4 languages of interest (`LaTeX`, `Python`, `R` and `C++`) then it will rename the first column (row numbers) to see in which position (alphabetically) these languages are placed.

```
library(dplyr)
my_lang <- c("LaTeX", "Python", "R", "C++", "CLI")
df  |>
  filter(Language %in% my_lang) |>
  rename(Lang_N=X)
```

Figure 4: **R code to import the data frame from the previous code-block and filter it.**

The processing steps in the code above will return the filtered table so there is no need to store in an object just use the `#+RESULTS:` keyword with the name of the code block.

Table 1: **Built-In language support for selected languages on Emacs.**

| Lang_N | Language | Identifier | Documentation | Maintainer |
|---|---|---|---|---|
| 3 | C++ | cpp | ob-doc-c | Thierry Banel |
| 5 | CLI | shell | ob-doc-shell | Matthew Trzcinski |
| 22 | LaTeX | latex | ob-doc-LaTeX | nan |
| 36 | Python | python | ob-doc-python | Jack Kamm |
| 37 | R | R | ob-doc-R | Jeremie Juste |

## 3 Notes

Note that in the pdf version of this file, the caption for all the code blocks has the label, 'Figure'. This is an issue of the exporter that requires additional configuration that was not meant to be included in this example.