Italian Glosser

This script, developed by Beatrice Turano for the University of Milano Bicocca, implements a glosser for Italian sentences.

The script takes an excel file as input and returns the same excel file, with a new sheet containing the glossed sentences.

A brief description of how the glosser works is reported below.

1. Morphological Analysis

The input sentence is first analysed using [spacy](https://spacy.io/). The following information are extracted for each word:

- text
- Part of Speech (pos)
- Lemma
- Morphological information where available, in particular:
    - mood, tense, person and number for finite verbs
    - mood for infinite verbs
    - number and grammatical gender for nouns, articles, participles, and adjectives
    - other morphological information for parts of speech that will not be specified in the gloss but that will be useful for the subsequent translation.

N.B. If the pos is `PUNCT` or `X`, the token is ignored.

2. Annotation

The morphological information extracted in the previous step are converted in order to follow the Leipzig Glossing Rules. The conversion happens thanks to a custom glossary. The parts of speech that get annotated are the following: articles, nouns, adjectives, auxiliaries, verbs, participles.

3. Sentence translation

The whole sentence is translated in English using DeepL.

4. Lemma translation

Each lemma is translated using:

- A custom glossary for specific cases (ex. Clitics)
- DeepL for every other case.

Since deepL is a Machine Translation tool that works best when there is a lot of context, passing only the lemmas to the translation won't work. A custom logic was implemented in order to use Italian syntactic rules and the glossing logic (as already said, some parts of speech do not get annotated) to form bigger chunks of lemmas to translate. For example, since in most cases an article is followed by a noun, instead of translating the lemmas by themselves, we put them together.

Example: let's imagine this input sentence: *"Una ragazza sta pettinando una bambola*".

The first NP, "*una ragazza*", should be glossed as "*uno-F.SG ragazza-F.SG*". However, if we pass the lemma "*uno*" to DeepL, it returns "*one*" instead of "*a*". That is why we implement a logic that identifies bigger chunks ("*una ragazza*") in order to obtain the correct translation ("*a-F.SG girl-F.SG*"). Usually, the chunks are identified according to the parts of speech that get annotated - when there is an annotation, we create the chunk.

The case of articles

As we said, articles also get annotated, so according to this logic they should create a chunk by themselves. This does not happen because when there is an article, the logic "hides" the annotation until another annotated term is reached (a noun, in this case). The annotation of the article is "revealed" and added to the article only after the chunk is translated.

5. Creation of the final gloss and output file

All the translated and annotated chunks are put together in a single string. A new worksheet is created in the excel file we pass as input and the following columns are added:

- "FRASE": the original sentence, one per line
- "TRADUZIONE": the translation of the original sentence
- "GLOSSA_IT": the glossed sentence, with lemmas in italian
- "GLOSSA_EN": the glossed sentence, with lemmas in English.

Technical documentation

Project files

It is important that all the project files are in the same folder. Make sure to have the following files:

- code_official.py
- language_data.py
- Pipfile
- Pipfile.lock
- README.md (contains the same overview as the first page of this file and the Extra: Deep Dive in the code at the end of the technical documentation).

Installation

Requirements

- Python 3.10 (download the specific version from https://www.python.org/downloads/) - it must be added to PATH.
    - On Mac, it must be manually added to the PATH variable of the system in order to make the terminal recognize "python3" as a command (https://www.mygreatlearning.com/blog/add-python-to-path/)
    - On Windows, select the **Add Python to PATH** flag in the installation window.

  Check the correct installation by open the Terminal and running:

```
python3 --version
```

- Pip
    - Once Python is installed, usually pip is installed too. If it is not, open the Terminal (on Mac) or Windows PowerShell (on windows) and run:

```
python3 get-pip.py
```

If an error occurs, try:

```
python3 -m pip install --upgrade pip
```

You can check the correct installation by running the command:

```
pip3 --version
```

- Visual Studio Code: it's a code editor. Download here: https://code.visualstudio.com/

    - Open the app, select the column on the left with the puzzle icon ("Extensions") and install the following extensions:
        - Python
        - Pylance
        - isort
    - On the column on the left, see the button "Open folder". Click on the button and select the folder with all the project files.
    - On the top menu, choose the "Terminal" option. All the next steps can be executed from the terminal.

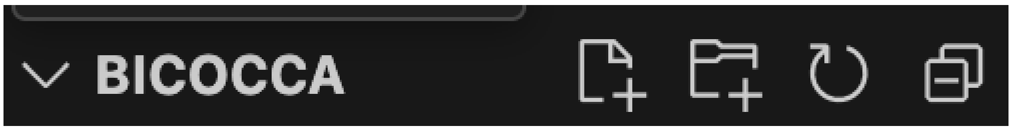
- Pipenv: python library for the management of virtual environments. To install, open the Terminal (from VS code, from Terminal app, from Powershell as preferred) and run the command:

```
pip3 install pipenv
```

Setup

Virtual Environment: DeepL API Key

Open the project folder from Visual Studio Code. You will see the folder name and these icons:



Click on the first icon on the left to add a file to the folder, and name the file ".env". Inside this file, write the DEEPL API KEY (shared among the Researchers involved in the project; ask Prof. Guasti for the key) in this format:

```
DEEPL_API_KEY = 'abcdefghijklmnopqrstuvwxyz'
```

SpaCy

We use the library spaCy to perform the morphological analysis. This library has specific requirements.

1. Open the Terminal and install spacy by running:

```
pip3 install -U spacy
```

2. After it is done processing, run also this command:

```
python3 -m spacy download it_core_news_lg
```

Setting up the rest of the virtual environment

After setting up spacy, the virtual environment needs to be set up.
Open the terminal and run the following:

```
pipenv install
```

This should install all the other requirements automatically.
Otherwise, you can run the following one after the other:

```
pipenv install deepl
pipenv install openpyxl
```

Running the script

Input

Create an excel file where the first column contains only the sentences to be glossed. Do not add any header.
Put the file in the same folder of the rest of the project.

Enter the environment

Enter the newly created virtual environment by running

```
pipenv shell
```

from the Visual Studio Code terminal.

Run script

Run the script by writing the following command in the terminal:

```
python code_official.py -f <file>
```

where <file> is the name of the input file.

NB you will see a lot of prints on the console. It is normal.

Check the output

Open again the excel file using excel. You should see another worksheet with the name "Glossed" and the 4 columns mentioned in the previous pages.

Extra: Deep dive into the code (technical explanation)

Glossing

First of all, the input sentence is given to the spacy Italian model and the aforementioned information is extracted from the output and saved in a dictionary. The dictionaries are appended to a list.

The list loops through all of its items, which are the token objects extracted from spaCy. A number of nested conditions are checked in order to proceed. The conditions are listed below.

1. POS

If the POS is not `DET`, `PRON` or `ADP`, a gloss is created by converting the extracted morphological information into the Leipzig Glossing Rules using a custom glossary. The gloss is saved as a separate element in the token dictionary. If the POS is in the list of exceptions, the gloss is saved as an empty string.

2. Gloss

We first check two main conditions.

Case 1: The gloss is an empty string.

If the gloss is an empty string, it means that we can compact the lemmas into a more complex piece of text before sending it to translation. In particular, we save it into a list of tokens that need to be translated at a later time.

1. However, if the POS of the token is `DET`, we convert the morphological information first and then add the token dictionary to the list of items to translate.

    Why adding glosses to DETs only now?

    Because the logic implemented decides where to break the sentence and translate it according to where the glosses are. However, the more context we give to the automatic translation, the more accurate it is. So adding the glosses to the DETs only after allows to create more complex chunks of text for the translation to be accurate without discarding the gloss.

2. If the token is the last one to be translated, then it is translated.
    a. If the token has a gloss (and therefore is a `DET`), the gloss is concatenated to the translation.

Case 2: The gloss is not at empty string.

The translation can happen only in this step, as we are dealing with one of the elements that determine the end of the chunk. The token is added to the list of items to translate and passed to the translation function. The list of items to translate is reset to its original state.

Custom translation

A custom translation function is implemented in order to avoid using automatic translation for specific cases, as mentioned before.
The custom translation makes use of manually compiled dictionaries to correctly translate:

- Clitic pronouns
- Pronouns
- Some articles

The custom translation logic takes a chunk of lemmas as input, and loops through them. It first checks whether the token is in the custom dictionary, and in that case, saves the corresponding translation.
If the token is a pronoun, it checks whether it is to be found in the clitics dictionary, and appends the correct translation.
If the token is not to be found in any of the custom dictionaries, it is translated with DeepL.