Introduction to Python

Serge Sharoff

Introduction to Corpus Linguistics for Translators

1 Objectives

At the end of these classes, you should be able to:

- Use the basic functions of Google Colab/Jupyter Lab
- Understand a simple python script
- Edit it to improve its logic and output
- Start your own corpus collection from the Web
- Classify texts automatically using existing models
- Investigate keywords for subcorpora in your collection

2 First experiments

2.1 Learning about Google Colab or Jupyter Lab

1. Open `https://colab.research.google.com/` or Jupyter Lab on your computer
2. Create a new notebook and type the examples from Slide 6
3. What are the commands to run the cells? What are their shortcuts?
4. You can add new cells at the very end or between existing cells. Find a way how to do it.
5. Once you've run all of the cells, what is the value of variable *s*?
6. How many words does it contain?
7. Print the second word in this string.

2.2 Running your first script

1. Save the notebooks from this week's folder to your M Drive
2. Upload healthy-linguist.ipynb to Python environment
3. Run it cell by cell until you reach the end.
4. Modify the variables for your name and age and run again.
5. Now upload word-frequency.ipynb and run it.

2.3 Understanding word frequency distribution

1. What does it do when it sees a word NOT in its dictionary yet?
2. How many lines does it print?
3. How does it compute IPM? What is the total word count for this text?
4. How does it choose which words to output?
5. Can you check the total number of words recorded in the dictionary?

2.4 Modify the word frequency script

1. Change the last line to print the frequency report like:
   The frequency of the word "parrot" is 8, ipm 10335. (or whatever counts are reported)
2. At the end of the script, print the number of words it outputs.
3. Change the word selection conditions to include more words.
4. Look into the Python documentation for `defaultdict`. Simplify our script by using it.
5. Add a function to save the dictionary a plain text file with the structure:
   word<TAB>real word frequency<TAB>normalised ipm frequency.

2.5 Getting the keywords

1. You can experiment with your own data or with the files available under the Corpora folder on Minerva (you will have to **uncompress** them to make the usable). If using Colab, upload them to a folder and make sure that `gdrive` point to the right folder.
2. Upload word-keyness.ipynb and run it.
3. Repeat the same process with more files.
4. Add a function to save the most important keywords. (Usually they need to have the LL-score above 11, check statistics textbooks).
5. Modify the script to include the possibility to repeat it over several files.
6. Experiment with your languages. For Chinese try using the jieba library, for Japanese try using mecab. There is also a multilingual interface of spaCy.

y

3 Process web pages

3.1 Installation of additional modules

Run: `!pip install trafilatura`

This should install the trafilatura module for processing web pages. When you need any extra module, you can install it in this way by replacing `trafilatura` with the name of that module.

3.2 Processing Web pages

1. Use the notebook corpus-collect-list.ipynb
2. Make sure it has access to your Google Drive (running Cell 3 will request this connection). If you use your own environment, modify the script to link the file collection to the right location.
3. If you want more broad crawling, use corpus-collect-list.ipynb and apply it to a website you have identified as highly relevant to your specialised corpus domain. There is a variable to control this.
4. How can you save your results to a different folder? You're likely to need to save them to a different subfolder for each language?
5. What happens with scraping from different websites? Please check the description of Javascript and the role it plays in rendering websites.

4 Text processing

4.1 Extract linguistic features

1. Read a text from your corpus (by modifying the filename in linguistic-annotation.ipynb) and determine:
   - its genre (use HuggingFace model `ssharoff/genres`)
   - its mood (model `bhadresh-savani/distilbert-base-uncased-emotion`; you might need to collect less formal texts first; keep your file structure separate for each corpus)
   - its named entities
2. Try this processing with a text in a different language.
3. Modify linguistic-annotation.ipynb to run this processing to annotate **all** files in your corpus. (Keep in mind that you have files for two languages). If your filenames follow a pattern with numbers, check the documentation for `range` to generate a list of numbers. See also the example of how names are generated in a loop in corpus-collect.ipynb. Otherwise, you can run your for-loop through an explicit list of filenames.
4. What is the statistics of each genre or emotion label? Can you create a dictionary of the more common names in each category?
5. What happens if you work with a language other than English? Are the classifiers still accurate? You can explore more models at `https://huggingface.co/models` (focus on Token classification and Text classification models)

5 Keyword analysis

1. Save the predictions of the emotion classifier to separate your files into those with positive and negative emotions.
2. Make separate frequency lists for positive and negative emotions. Do the words for your projects occur in those files?

3. Compare the keywords for the respective frequency lists. As a template for this task use word-keyness.ipynb.

6 Terminology extraction

1. Run terminology extraction with terminology-experiments.ipynb
2. Check the codes it outputs. Anything other than `LABEL_0` indicates a term
3. Generate a list of terms from your corpus in English
4. Generate a list of terms from your corpus in another language

7 Debugging

You can feel frustration when a python script does not work as intended. This is common experience with many tools including python. They do not always follow your expectations, but they can process many more things, so that you can devote your time to something more interesting. Find ways to adapt. In particular:

1. Please experiment by modifying the script: you can always return back to the original script as posted by the tutor. Once you know your own script does at least something important, you can save it under a new name, so that if you make any further modifications, they do not impact the version you know works.
2. If you do not understand why the script misbehaves, use `print` more often. For example, you can print the length of the url list or each item before processing them. This can help you in understanding what goes wrong. Use `assert` to make sure the next step proceeds with the correct data.
3. When you restart your Colab or Jupyter environment, the previous cells have not run to initialise the variables. Check different options in the Runtime menu.
4. Use python documentation and discussion forums. Documentation at `https://python.org` is extensive. Even more information is searchable through discussion forums.
5. Often errors in Colab are linked to Q&A at the Python StackExchange forum. They can give lots of information about the problems experienced by other python users.

8 Other sources to learn about Python

- Dirk Hovy's Python for Linguists, see `http://www.dirkhovy.com/portfolio/papers/download/pfl_handout.pdf`
- Digiling Python Intro `https://learn.digiling.eu/`
- NLTK book: `http://www.nltk.org/book/`
- How to think like a computer scientist: `http://openbookproject.net/thinkcs/`
- Collection of tutorials from `https://pythonbasics.org`
- Another collection from `https://wiki.python.org/moin/BeginnersGuide/NonProgrammers`