

## CHAINING SEARCH NOTEBOOK – QUICK START

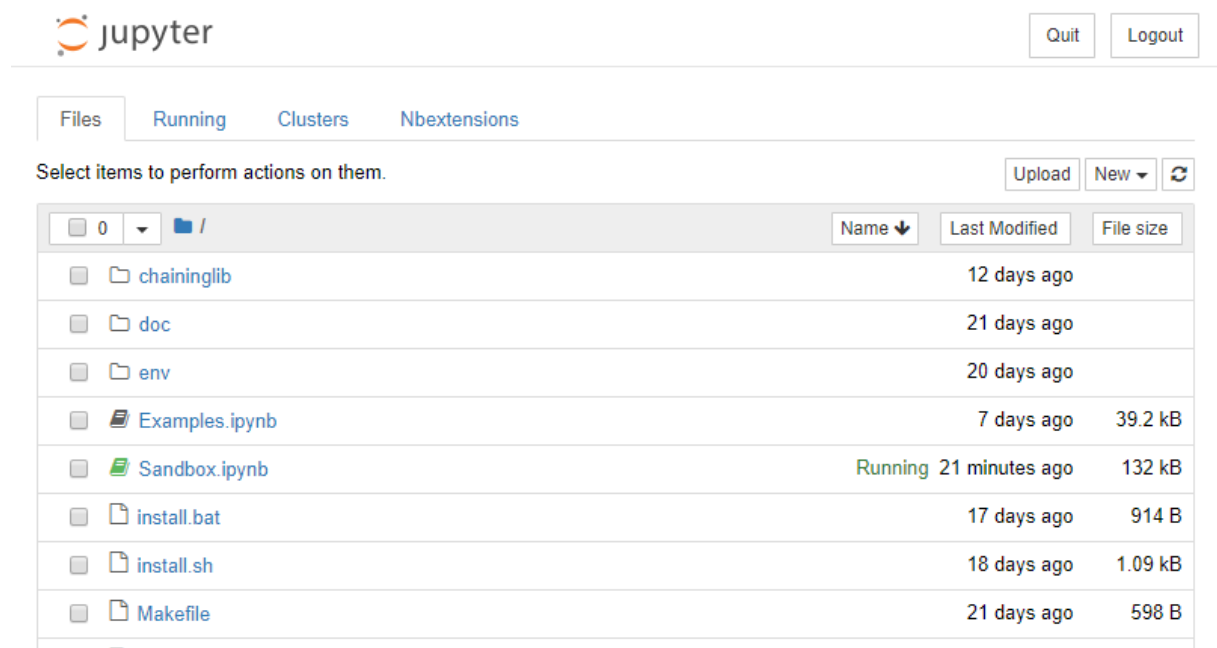
The Chaining search notebook is an advanced digital environment for linguistic research. It supplies more than common tools like an graphic user interface for searching a lexicon or a corpus. The notebook provides the user with a set of functions allowing highly customizable search and data processing operations. One can write custom Python scripts to do multiple searches and process the results, draw charts, and of course ‘chain’ operations. That is: the results of the one search can be programmatically re-used in another search, in another source, in another way.

## INSTALLATION

The installation procedure of the notebook is fully described in the `README.md` file on GitHub. Please refer to this file. Installation scripts are provided for both Linux/Mac and Windows.

## LET’S START

Once the notebook is installed and launched as indicated in the `README.md` file, your browser will automatically open the following page:



As shown hereabove, the notebook consists of some folders and files. Only two files are relevant to you now:

- `Examples.ipynb` and
- `Sandbox.ipynb`

The first file, `Examples.ipynb`, is a notebook filled with case studies. These are examples showing how to perform search and data processing operations of several types and complexity.

Once these examples have inspired you, you'll be ready to go ahead with the second file, `Sandbox.ipynb`, which is an empty notebook, where you can try things out or build a complex script straight away.

The following sections will present both files and explain what you can do with those.

### Folder structure

- The 'env' folder contains code for running the notebook environment and must be left alone.
- The 'chaininglib' folder contains the libraries that enables the functions you'll be able to use in your scripts (the so-called API).
- The 'doc' folder contains full documentation about those functions. You don't need to browse it, as you will be provided with a link straight to the relevant documentation file in a short moment.

## CASES STUDIES / EXAMPLES

The `Examples.ipynb` file can be accessed but just clicking on it. The file will open in a new tab called 'Examples' which should look like this:

jupyter Examples (autosaved) Python Logout

File Edit View Insert Cell Kernel Widgets Help Trusted | env

Run

### Examples Chaining search

This notebook contains a number of examples of chaining linguistic resources: corpora, lexica and treebanks. Try the examples, or copy the code and customize the examples in the [Sandbox](#).

### List of examples

#### Corpora

- [Corpus search](#)
- [Frequency of zeker+verb and vast+verb compared](#)
- [Train a POS tagger on an annotated corpus](#)
- [Search in corpus and filter on metadata](#)

---

## Make sure it's working

First of all, when opening this file for the first time, we have to make sure that we are using the right kernel and that the kernel is trusted.



Here is how to do that:

- First: The name of the kernel in use is shown in the horizontal grayish top bar (see picture). On its right edge, it should show the word 'env'. If it does, you're ok. If it doesn't, click menu item 'Kernel' in the same bar and choose option 'Change kernel'; this will open a pulldown menu where you must click the 'env' option. The kernel name on the right side of the bar must now read 'env' as well.
- Second: On the left side of the kernel name, there must be a small box saying 'Trusted'. If it does, we're done. But if it says 'Not trusted' instead: click on it, and when a confirmation window pops up, click on 'Trust'. Done!

---

## OVERVIEW

Now we are ready to have a look at the examples. We have quite a list of those:

### ▼ List of examples

#### Corpora

- [Corpus search](#)
- [Frequency of zeker+verb and vast+verb compared](#)
- [Train a POS tagger on an annotated corpus](#)
- [Search in corpus and filter on metadata](#)
- [Visualizing h-dropping](#)
- [Generate lexicon from several corpora](#)

#### Lexica

- [Lexicon search](#)

#### Corpus + lexicon

- [Retrieve synonyms from DiaMaNT, look up in Gysseling](#)
- [Build a frequency list of the lemma of some corpus output](#)
- [Find occurrences of attributive adjectives not ending with -e, even though they are preceeded by a definite article](#)
- [Look up inflected forms and spelling variants for a given lemma in a corpus](#)
- [Corpus frequency list of lemmata from lexicon with given lemma](#)
- [Build a frequency table of some corpus, based on lemmata of a given lexicon](#)
- [Search corpus for wordforms of lemma not included in lexicon](#)

#### Treebanks

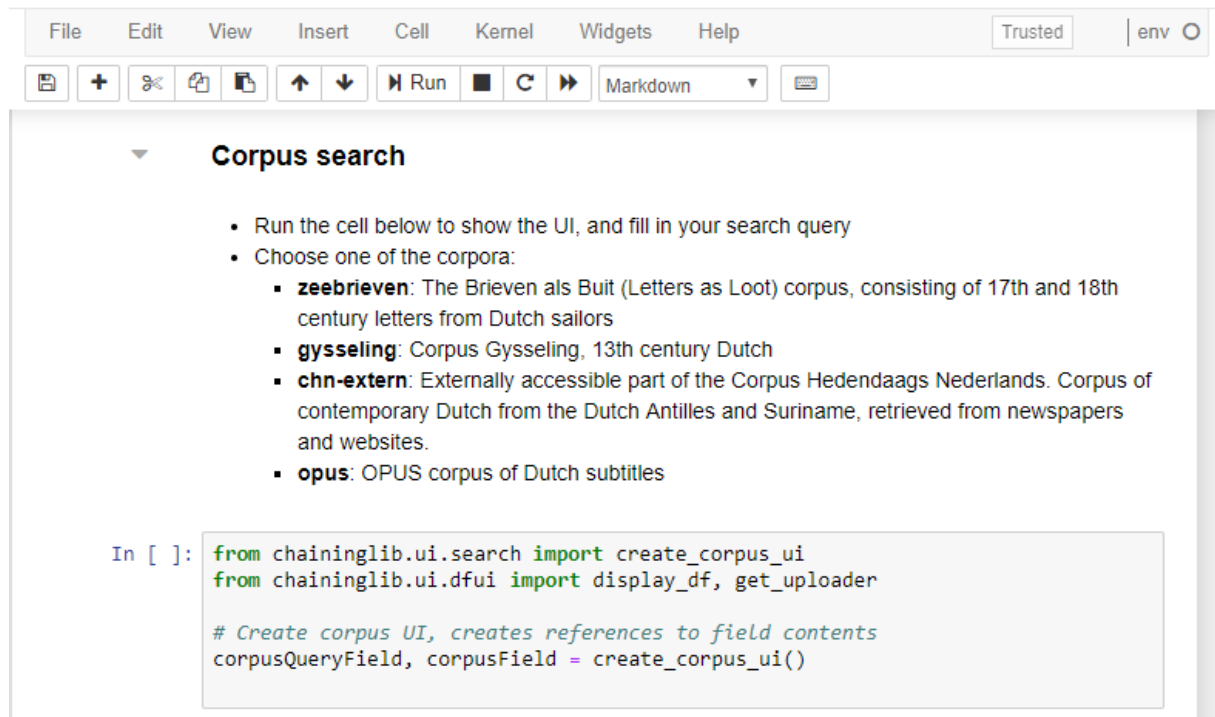
- [Treebank search](#)
- [Which objects of verb \*geven\* occur?](#)

As the screenshot shows, the examples are ordered by category. You can visit an example straight by clicking on its name.

---

## RUNNING A CASE STUDY

We've just seen the list of available case studies/examples. Now, clicking on the first example ('corpus search') will lead you to this screen section:



**Corpus search**

- Run the cell below to show the UI, and fill in your search query
- Choose one of the corpora:
  - **zeebrieven**: The Brieven als Loot (Letters as Loot) corpus, consisting of 17th and 18th century letters from Dutch sailors
  - **gysseling**: Corpus Gysseling, 13th century Dutch
  - **chn-extern**: Externally accessible part of the Corpus Hedendaags Nederlands. Corpus of contemporary Dutch from the Dutch Antilles and Suriname, retrieved from newspapers and websites.
  - **opus**: OPUS corpus of Dutch subtitles

```
In [ ]: from chaininglib.ui.search import create_corpus_ui
        from chaininglib.ui.dfui import display_df, get_uploader

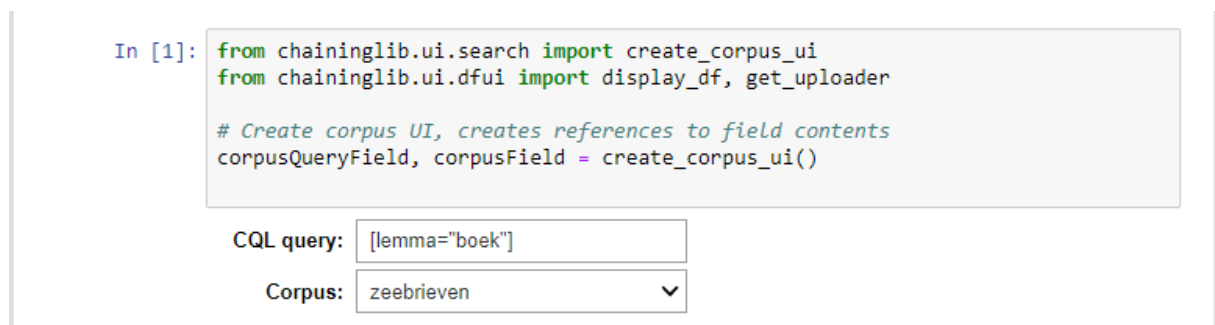
        # Create corpus UI, creates references to field contents
        corpusQueryField, corpusField = create_corpus_ui()
```

Each example is built quite the same way as above. The title of the example is followed by some explanation, and a cell with some Python code in it.

To run the code:

- Click into the cell
- Then go to the horizontal top bar and click on 'Run'

Once we've done that, some output should appear under the code cell. In this particular case, our output is a search interface, in which we can enter a search query and choose a corpus to search:



```
In [1]: from chaininglib.ui.search import create_corpus_ui
        from chaininglib.ui.dfui import display_df, get_uploader

        # Create corpus UI, creates references to field contents
        corpusQueryField, corpusField = create_corpus_ui()
```

CQL query:

Corpus:

To start the search, go to the next code cell.<sup>1</sup> This following piece of code reads your input out of the interface and start a search with that:

- Click the cell below and press Run to perform the given query

```
In [ ]: from chaininglib.search.CorpusQuery import *

#from chaininglib import search
query= corpusQueryField.value
corpus_name = corpusField.value
df_corpus = create_corpus(corpus_name).pattern(query).search().kwic()
#df_corpus = load_dataframe('mijn_resultaten.csv')
display_df(df_corpus, labels="Results")
```

When this code cell is clicked upon and subsequently run, the search results will appear underneath as expected:

### Results

	left context	lemma 0	pos 0	word 0	right context
0	heeft 4 gl 0 Aen	boek	NOU	boeken	en pampier en pennen en
1	lijeue man stelt alles te	boek	NOU	boeck	waet ghij uijt gefit dat
2	alzo0 hij niet op de	boek	NOU	bouck	en stondt en hij heeft
3	Schrijfpampier a 6 Sr t	boek	NOU	boek.	50 ditto ongsneeden ditto a
4	voorne missive, dat UEDs mijn	boek	NOU	boeken	bij bemtrop laat verkoopen, als
5	verbeeld hebbe, dat wat geleerde	boek	NOU	boeken	aan gaat, schouten altoos gepraefereert
6	heilde accuratesse goede order der	boek	NOU	boeken	& Voorsichtinheid, die nriice ik

This is all you need to know to be able to explore the following examples! For each example, click the code cell you want to run the code of, click on 'Run' in the horizontal top bar, and see the results.

### Important remark

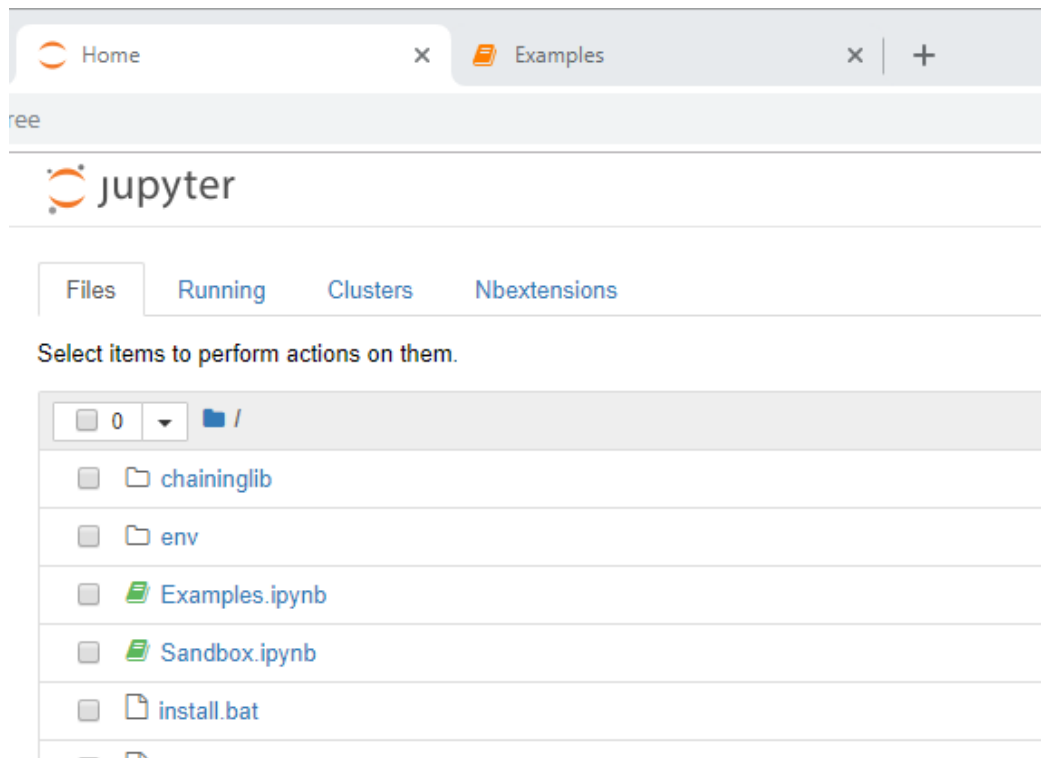
Note that in some rare cases, an example can only be run when the previous example was run before. But in such cases, the introduction text indicates that.

<sup>1</sup> A mistake commonly made in this particular case study is to start the search by clicking on 'Run' in the horizontal top bar again. But since the cell that contains the code for generating the search interface is still active (because that's the last element you clicked on or typed in), clicking on 'Run' again will just cause the interface to be rebuilt. So the search won't start at all. To be able to start the search, you need to go to the next code cell. This cell contains code that picks up the input from the interface and start the search with it.

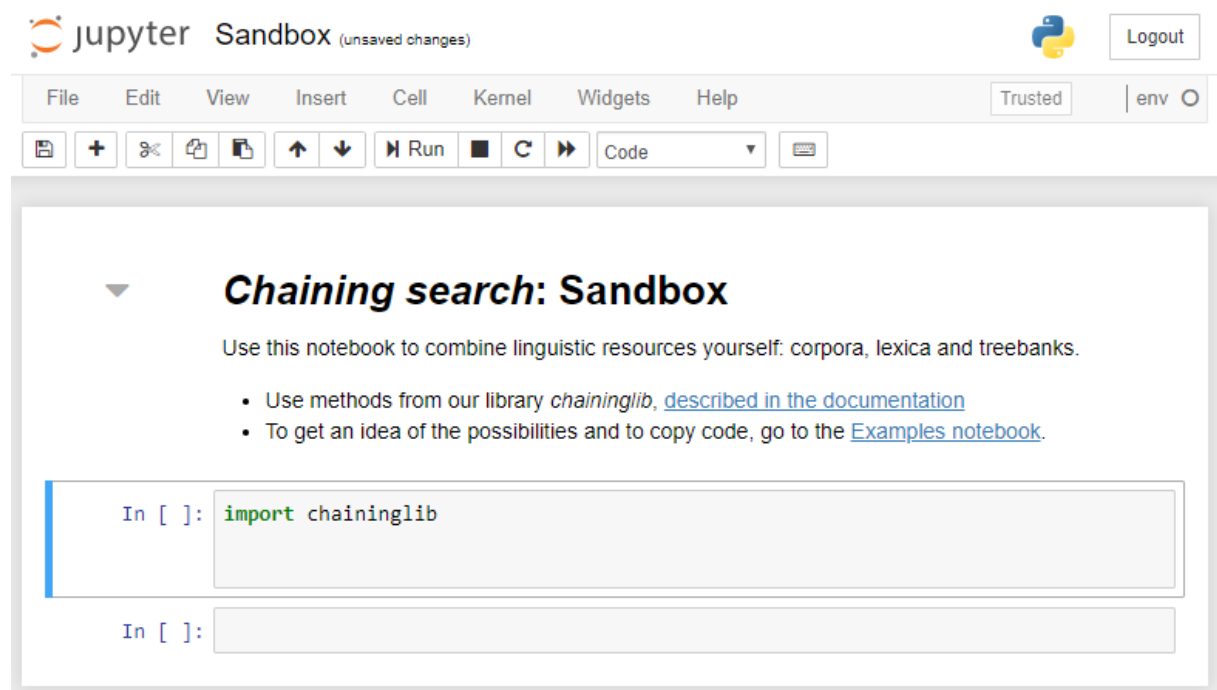
## YOU OWN SANDBOX

Now you're finished with the case studies, let's look at the `Sandbox.ipynb` file. That file is the place where we'll be building our own scripts.

To access this file, we'll first have to go back to the browser tab called 'Home'. This tab shows the list of files or folders available in our notebook installation.



Look up the `Sandbox.ipynb` file in the list and click on it. The file will open in a new tab and should look like this:



Just like in the Case studies screen, you have cells at your disposal. You can type your own code there, following what we've seen in the Case studies, and also the methods documentation. Both can be accessed again from this screen by clicking the corresponding link (see picture). Finally, click on the 'Run' button to execute your code.

Happy coding!