Software

Course documentation for students of Dead Language Processing

P. S. Langeslag

April 6, 2025

Contents

Overview

This course will have you producing and running Python code. Rather than working in a terminal-based interpreter as described in our textbook, we will take advantage of Jupyter Notebooks, an environment allowing us to work either in-browser or in an integrated development environment (IDE, essentially an advanced code editor) and divide our code up into cells that we can save, modify, reorder, and rerun as needed, as well as intersperse with full Markdown documentation. This guide will help you get set up. If you're new to this

sort of thing, set aside plenty of time and prepare to follow a lot of manuals, ideally before the start of term! If you run into any obstacles, you may find solutions online; you can also use the Stud.IP forum to ask your peers for tech support as required.

Local IDE: The Better Way

If you can stomach setting up Python, Jupyter, and Git locally, I recommend you do so, and then install an IDE with Jupyter support; try Visual Studio Code (aka VS Code or Code) with the Jupyter extension, following the below instructions. A local installation has the major advantage that you will only have to install the necessary Python packages once. However, if you choose to go this way, you should be aware that one of the libraries we'll use, CLTK, is only officially compatible with Python 3.7, 3.8, and 3.9, while another, `gensim`, requires a Python version no greater than 3.11. For this reason, and also for reasons of general best practice, you are strongly advised to set up a Python installation and environment manager, for instance by installing the Anaconda distribution of Python that comes standard with Jupyter and environment management included; or else look into `pyenv`. VS Code has its own guides to installing Python and using it in VS Code, and to Python environments. There's a great introduction to `pyenv` at Real Python.

JupyterLab Remote: The Fallback Option

If you are new to all this, and setting up Python 3.9, Jupyter, and Git is beyond you, you may alternatively choose to work in-browser in GWDG's remote JupyterLab instance at https://jupyter-cloud.gwdg.de, following the instructions below. The main obstacle to recommending this as the easier approach is that you will ideally have to set up SSH keys in order to access the course repository on GitLab; but if you can't manage that, you can alternatively download the code from GitLab manually and upload it to your JupyterLab environment yourself. An additional drawback to using the remote instance is that although your work is safely stored, Python packages are lost between sessions and will have to be reinstalled every time you do a stint of work. Furthermore, GWDG's instance of JupyterLab no longer offers Python 3.9, so you may run into compatibility issues when using CLTK, affecting perhaps a week or two of your work. (Once they upgrade to 3.12, work involving `gensim` may also be affected.) On the other hand, an advantage of the GWDG environment is that it allows your instructor to reproduce any issues exactly, and all demo notebooks used in the course have been tested against it, whereas your instructor cannot help you with any local package conflicts or other software issues if you install Python on your own system. Additionally, the remote option saves you from having to set up Python, Jupyter, and Git yourself. Even so, you are probably best off trying a local installation first, and switching to the online option only if you can't get that to work.

Local IDE Walkthrough

Software and Plugins

In order to make use of VS Code's Python, Jupyter, and Git functionality, you will have to install their standalone distributions first. For Python and Jupyter, you have a choice

between two overall strategies.

Anaconda: The Easy Way

The easiest solution for Python and Jupyter is to install Anaconda, a Python distribution that comes standard with Jupyter and Python version/environment management. To install Anaconda, head to anaconda.com and select "Free Download", look for the small print to "skip registration" to download the installer without sharing your email address (unless you don't mind), and select your operating system and processor architecture. You will need a few gigabytes of free disk space. When opening Anaconda Navigator, you can, if you like, dismiss prompts to set up an account to gain access to AI assistance and other online features. At this point you may want to update Anaconda Navigator to the latest version before undertaking further action. Next, to set up an environment with Python 3.9.18, launch the Anaconda Prompt from within Anaconda Navigator and enter

```
conda create --name nlp -c anaconda python=3.9.18
```

and confirm when prompted; or follow these instructions to use the graphical Navigator app to do the same. Once done, the second dropdown menu at the top of your Anaconda Navigator Home screen will give you access to a new environment "nlp" in addition to "base (root)". Activate the new environment in the "Environments" tab by clicking on it, then return to the Home screen and continue with Git and Visual Studio Code below. Going forward, always just check that the correct environment is active before launching any of your apps from within Anaconda Navigator.

`pip` and `pyenv`: The Traditional Approach

If you prefer to use the more traditional `pip` package manager over `conda`, you will want to go over the installation instructions at Real Python, or at least the briefer installation notes at the top of VS Code's Get Started With Python tutorial. You will have to install Jupyter Notebooks separately. Basic installation instructions are at jupyter.org, but they assume you are comfortable with Python package management already; `pip` instructions are here, and even they assume you know to open a terminal (in Windows, use PowerShell) and issue your commands there; as per usual, the relevant Real Python article has more detail. This is also the point at which you'll want to install a Python version and environment manager; `pyenv` does both (see the relevant Real Python article for guidance).

Git and Visual Studio Code

Once you are set up with a Python distribution, your next step is to install and configure Git. Configuration can be as simple as entering two lines like the following into your terminal (or entering your name and email into a graphical cofiguration assistant):

```
git config --global user.name "Firstname Lastname"
git config --global user.email your.email@stud.uni-goettingen.de
```

Now if Visual Studio Code does not already show up in Anaconda Navigator, look for it in your operating system's package manager ("app store"), or download it from https://code.visualstudio.com and install it to your system. Depending on your operating system,

you may have a choice of distributions. In that case, installing the proprietary Microsoft release ensures you will have access to all the necessary extensions.

Once you have it installed, launch Code and open the extensions manager (`Ctrl+Shift+X`) to install the extensions entitled Python (by Microsoft), Python Environments (by Microsoft; agree to install the pre-release if no final release is available), Jupyter (by Microsoft), and GitLab Workflow (by GitLab). *If you cannot find Python Environments, it may be because you have installed the open-source implementation of VS Code rather than the proprietary Microsoft release.*

GitLab Access

We'll want to enable VS Code to access the course repository directly using its GitLab Workflow extension. This requires that you create a personal access token for your GitLab account. Log in at https://gitlab.gwdg.de, click on your avatar and select "Edit profile", then open "Access tokens" from the navigation bar on the left and "Add new token". Call it "VS Code" or similar, select the `api` scope, set an expiry date up to a year in the future but at least beyond the end of term, save the token and then copy it to your clipboard (and leave this page open just in case). In VS Code, open the Command Palette (Ctrl+Shift+P), start typing "GitLab" and select "GitLab: Authenticate". Select "Manually enter instance URL", enter `https://gitlab.gwdg.de` as your instance URL, then paste in the access token.

Now you have everything you need to clone the repository. Make sure to close any active folder within VS Code (File → Close Folder), then head to Source Control (Ctrl+Shift+G). You should now be promted to enter a repository URL (and if everything has gone well so far, you should see an option "Clone from GitLab (https://gitlab.gwdg.de - username)" below the input field; but don't select it). Copy in `https://gitlab.gwdg.de/langeslag-teaching/dlp.git`; or if prompted to enter the server address first, select or enter `https://gitlab.gwdg.de` and then copy in the repository URL as the next step. At this stage, you may be prompted for your user name and the access token a second time, so keep your GitLab tab open to the Access token page, as you won't be able to access it afterwards. Select the containing folder within which you wish your repository folder to be located (e.g. `Documents/Uni/DLP/`). Once the download is complete, you should be prompted to open and trust the contents of the new folder. Now when you open any of the Jupyter Notebooks in the folder (that's the files with a `.ipynb` extension), the option "Select Kernel" in the topright corner should give you a choice from among the environments you have set up; for instance, if you have used `pyenv` or `conda` to install Python 3.9.18 and have created an environment `nlp` associated with this Python version, then clicking "Python Environments..." should give you access to "nlp (Python 3.9.18)", and if it's already active it will be preceded by a star. If you don't use Python much outside this course, you may as well make this environment your default. Take a breather; you are now all set up, and can return to your work in future simply by opening VS Code (and reopening the course repository folder if it's not still open on your return).

If you run into obstacles in any part of this process and fail to find solutions online or among your peers, please get in touch with your instructor early so you don't get behind on your homework. At the same time, please understand that every machine has its own

configuration entailing its own complications, and your instructor can do no more than offer suggestions. While you wait for guidance, set up a JupyterLab remote environment following the instructions below just in case.

Package Management

Python package management is traditionally done from the command line, using a command `conda` or `pip`. In VS Code, with our repository folder open, select "Terminal" and "New Terminal" (or Ctrl+Shift+` on US keyboards) to open a terminal window with the current folder as its working directory. You can install individual packages by issuing commands like `conda install gensim` or `pip install gensim`, depending on your package manager; but our repository contains a file `requirements.txt` listing all the packages used for your homework. To install all of these with `conda`, issue

```
conda install --file requirements.txt
```

or for `pip`:

```
pip install -r requirements.txt
```

from within the repository folder. Please take note of any errors you encounter; these will generally result from conflicts between installed and required packages. Also remember that if you have activated a virtual environment for this folder (e.g. by selecting it in Anaconda, or by issuing `pyenv local nlp` to permanently associate the current folder with a virtual `pyenv` environment named `nlp`), packages are installed in that virtual environment, whereas if you haven't, they are installed to your global Python installation or Anaconda base environment.

JupyterLab Remote Walkthrough

Accessing JupyterLab Remote

If you run into any insurmountable issues when trying the local install, first activate your GitLab account by logging in at https://gitlab.gwdg.de if you have never done so before. Then, in a second browser tab, log in at https://jupyter-cloud.gwdg.de. You may select the default image if prompted. If you aren't returning to an active or saved workspace, JupyterLab should open to a launcher window (see Figure 1). From here, you can technically select Python 3 in the Notebook (or Console, see below) category to start coding. However, you'll probably want to create a new directory for each course or project you undertake; and furthermore, you'll want to clone the course's GitLab repository, which creates a course directory and copies in all the required files. So that's what we'll do next.

GitLab Access

In the JupyterLab launcher window, from the category "Other" select "Terminal". This opens a command-line interface answering to many of the same commands you may know from UNIX-like systems such as Linux or macOS. We now get to the trickiest part of the

remote JupyterLab solution: setting up SSH keys and exchanging them with GitLab. The following steps should do the trick:

1. In your JupyterLab terminal, enter `ssh-keygen -t ed25519 -C "your.name@stud.uni-goettingen.de"` and hit Enter to confirm defaults at each of three steps (including the passphrase prompts) until you are returned to the command line prompt.
2. Now enter `cat ~/.ssh/id_ed25519.pub`.
3. The output of the above command should start with `ssh-ed25519` and end with your email address, or whatever other value you entered after `-C` in step 1. Select this complete output, nothing more or less, with your mouse or touch interface, and right-click (or Command+click; or long press?) to bring up the context menu and select "Copy".
4. Now return to your GitLab browser tab, click on your avatar, "Edit profile", "SSH Keys", "Add new key", and paste your public key into the Key field. Give it a title like "GWDG JupyterLab", ensure that the usage type includes at least authentication, and set an expiration date beyond the current term.

(Cf. these fuller instructions, the relevant sections being Generate an SSH key pair and Add an SSH key to your GitLab account; follow the instructions for Linux, but note that the clipboard solution here given won't work in JupyterLab).

If everything has gone according to plan, you should now be able to clone the repository by pasting the following command into your JupyterLab terminal and hitting Enter:

```
git clone git@gitlab.gwdg.de:langeslag-teaching/dlp.git
```

You may be prompted to type `yes` to trust the fingerprint. This should create a folder `dlp` and populate it with our course files. The folder will appear in the file system in the left-hand pane. Once it's done cloning, you can close the terminal window (click `x` or type `exit`) and double-click the `dlp` folder on the left. Nothing changes except the current folder (or "working directory"). Now if you start Python 3, the files you've pulled in are available within your working directory.

If you fail to set up GitLab access, you can instead point your browser to https://gitlab.gwdg.de/langeslag-teaching/dlp, select "Code" and "Download source code: zip", extract the archive locally, then use the Upload function in JupyterLab to store the files in a folder below your home folder in the remote instance of JupyterLab.

If the course materials should ever be updated in the course of the term, you'll have to update your working copy to take advantage of these changes. To do so, enter the `dlp` folder in the file system, launch a terminal window, and enter `git pull`. Once the line `Writing objects` reaches 100%, your working copy is up to date and you can close the window. If you receive a warning that your working copy contains changes not in the master branch, that's where Git gets more complicated; try this guide. If you have manually uploaded the course materials, you can simply repeat that process for updated files. Do make sure not to lose any files you have yourself created; this can be avoided in either scenario by writing only to files with file names other than those tracked by the repository.

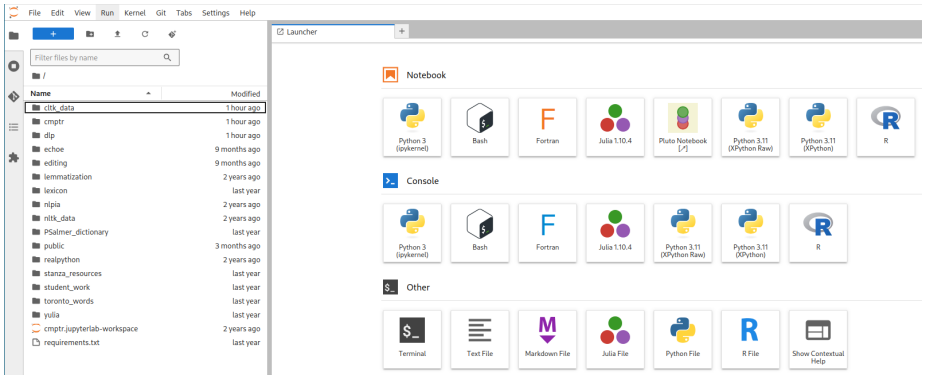

Figure 1: JupyterLab's interface: file system on the left, launcher on the right.

Package Management

Our textbooks assume that you have access to a Python package manager, usually `pip` or `conda`, on the command line. GWDG's remote instance of JupyterLab in fact incorporates `pip` directly into the interpreter, so to install a Python package like `nltk` you can type `pip install nltk` directly into your notebook or console, or you can take the more conventional route and do your package management from within a terminal window.

The remote instance of JupyterLab does not save your Python libraries: the next time you log in, your packages will be gone. To install all the packages required for this course before you start a stint of work, you can open a terminal window and run `pip install -r requirements.txt` from within the `dlp/` folder. This takes time, though, and is overkill if you just need one or two packages. Instead, whenever you start out work with one or more non-stock libraries, you can just install the ones you need (e.g. `pip install nltk gensim`). Likewise, if in your work you ever encounter an error along the lines of `ModuleNotFoundError: No module named 'gensim'`, simply install the missing package (`pip install gensim`) and get on with your work. `cltk` has a lot of package dependencies, so `pip install cltk` takes a little longer, but also covers `nltk`, `numpy`, `scipy`, and `scikit-learn` in one go; so simply starting any notebook with `pip install cltk` (ignore the warnings) will often get the job done, too.

Notebook or Console?

JupyterLab offers two interfaces: Notebook and Console. You can technically do your homework in either one, but if you want to save or submit any of your work you may want to opt for the notebook. The console offers a traditional interactive interpreter experience; the notebook is often considered more useful in a learning setting because you can save and export your work, you can reorder and rerun your code blocks, and you have formatting options (Markdown, or even LaTeX) for any notes you wish to take. You can also open a console alongside a notebook that is aware of functions and variables set in the notebook (File → New Console for Notebook). Finally, to ease your workload I have done some of your coding for you and made it available in the repository in the form of notebooks. So

all things considered, you'll probably want to do your final work in a notebook but any experimental work and "exploration" in a console spawned within your notebook session. The textbooks we'll use assume a regular interactive interpreter, but their exercises will work just as well in a notebook.

Notebook Modes and Commands

Like the Vim editor popular among programmers, the Jupyter Notebook interface has a command mode and an edit mode. When you open a notebook, it is in command mode and will respond to keyboard commands like the following:

`Enter`	Enter edit mode
`Shift+Enter`	Run cell code
`k` or arrow	Navigate up
`j` or arrow	Navigate down
`a`	Add a cell above
`b`	Add a cell below
`y`	Change cell mode to code
`m`	Change cell mode to Markdown
`dd`	Delete current cell

Once in edit mode, you have access to commands like the following:

`Esc`	Enter command mode
`Shift+Enter`	Run cell code and command new cell below

For more keyboard shortcuts, see e.g. here.

Markdown

Markdown is an efficient structured text format which it is well worth mastering, as it is the most convenient and powerful format in which to write nearly all the documents you need in your professional and personal life (except spreadsheets or complex tables). Fuller guides are all over the web (my own is here), but the short version is that it relies on formatting cues like the following:

```
# Top-level headings receive one hash/pound sign

## Lower-level headings receive two

Paragraphs are ended by double line breaks.

Like so.
```

```
[Links](https://are.encoded.thus) or they may be encoded <https://thus.ly>

```python
Code demonstrations ("listings") are set off within triple backtick delimiters.
```
```

The relevance of Markdown to Jupyter Notebooks is that it is the format for note-taking cells. You can toggle the format of a selected cell from code to Markdown with the `y` and `m` keybindings (press `Esc` first if in edit mode), or using the dropdown menu. A Markdown cell serves to document or annotate your code, or to create logical sections within a Python script.