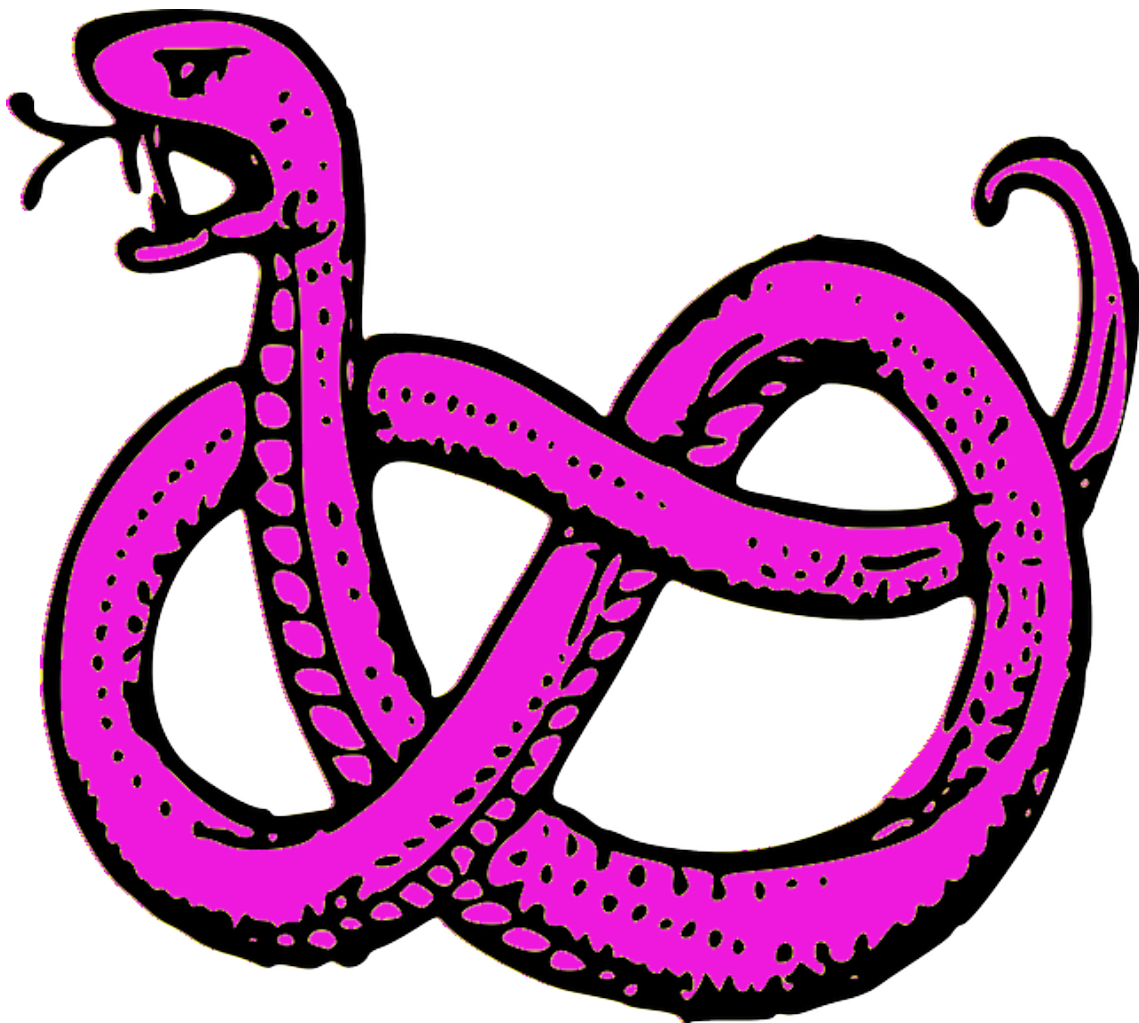


Handreichungen zur Aufzucht und Pflege von Python in außergewöhnlichen Lebensräumen: Jupyter Notebooks und Jupyterlab

Prof. Dr. Christian Münker



Zuletzt aktualisiert: 6. Oktober 2020

chipmuenk@gmail.de

Aufzucht und Pflege von Python in außergewöhnlichen Lebensräumen

(c) 2017-2020 Prof. Dr. Christian Münker

Titelbild:

(c) Clker-Free-Vector-Images, <https://pixabay.com/de/schlange-reptil-gef%C3%A4hrlich-hiss-37585/> unter CC0 Lizenz

Die genialen XKCD-Comics in diesem Skript:

(c) Randall Munroe (<https://xkcd.com/>)

unter CC BY-NC 2.5 Lizenz (<http://creativecommons.org/licenses/by-nc/2.5/>).

Diese Publikation kann frei heruntergeladen werden unter

<https://github.com/chipmuenk/dsp>.



Dieses Dokument steht unter Creative-Commons-Lizenz CC BY-SA 4.0:

<https://creativecommons.org/licenses/by-sa/4.0/deed.de>.

- Bei Verwendung dieses Werks müssen Autor, Titel und URL zu Werk und / oder Autor genannt werden und es muss auf die entsprechende CC-Lizenzurkunde verwiesen werden („BY“, attribution).
- Dieses Werk oder Teile daraus dürfen nur unter gleichen Lizenzbedingungen weiterverteilt werden („SA“, share alike) .

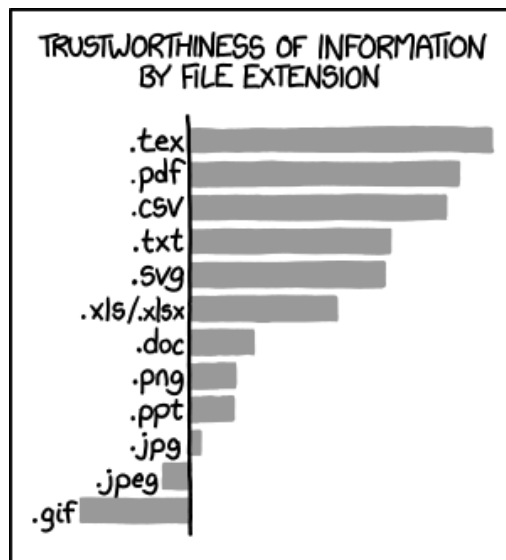


Abb. 1: Diese Unterlagen wurden mit L^AT_EX verfasst [<http://xkcd.com/1301/>]

Inhaltsverzeichnis

Open Source und Python: Aber warum denn nur?	v
1 Jupyter Notebooks	1
1.1 Einführung	1
1.1.1 Beispiel-Notebooks	3
1.2 Notebooks in der Cloud	4
1.2.1 Gesis Notebook Server	5
1.2.2 Microsoft Azure Notebooks	6
1.2.3 Software Installation / Update	7
1.3 Notebooks auf dem eigenen Rechner	7
1.4 Jupyter-Client: Arbeiten im Browser	7
1.4.1 Code - Zellen	9
1.4.2 Text / Markdown - Zellen	10
1.4.3 Zellen mit Medien	11
1.5 JupyterLab - IDE im Browser	11
1.6 Blogs und Videos zu Jupyter-Notebooks / JupyterHub / JupyterLab	13
1.6.1 Jupyter Notebooks - Computing im Browser	13
1.6.2 JupyterLab - Next Generation Browser IDE	14
1.6.3 JupyterHub - Und jetzt alle!	15
2 Git für Jupyter Notebooks	17
2.1 Repository Clonen	17
2.1.1 Microsoft Azure	17
2.1.2 Gesis	17
2.1.3 Terminal (alle Jupyter Server)	18
2.2 Update vom Remote Repo holen	18
2.3 Einchecken	19
2.4 Update Remote Repo	20
2.4.1 Neues Notebook	20
2.4.2 Update von Notebooks auf lokalem PC	20
2.4.3 Update von Notebooks direkt auf dem Azure Repo (nur Owner)	21
2.5 Ein paar Git-Kommandos	21
2.5.1 Basics	21
2.5.2 Stand des Remote-Repas erzwingen	22
2.5.3 Versionierung von Notebooks mit nbdime	23
2.6 GitLab	24
2.6.1 Projekte	24
Abbildungsverzeichnis	25
Tabellenverzeichnis	26

Listings**27**

Open Source und Python: Aber warum denn nur?

Wenn man außerhalb eines Computerlabors gemeinsam an Code arbeiten möchte, ist ein BYOD (bring your own device) Szenario zunächst verlockend aufgrund der geringen Kosten und weil man selbst nicht die Computer administrieren muss. Außerdem können Studierende so auch Aufgaben zu Hause vorbereiten. Aber ...

Die Installationshürde: Bei komplexerer Software ist es oft schwierig, Studierende zur Installation zu bewegen, vor allem wenn drei andere Kollegen bereits vier weitere Softwareinstallationen verlangen.

Kein Rechner gleicht dem anderen: Betriebssystem(version), Hardware und viele andere unvorhersehbare Faktoren machen den Installationsupport zu einer Angelegenheit, die nie langweilig wird.

Mit der Lizenz zum Coden: Bei kommerzieller Software kommt erschwerend das Verteilen von Lizenzschlüsseln, das Installieren von Lizenzservern etc. hinzu.

Ich sehe was, was Du nicht siehst: Die Verteilung von Code-Schnipseln, Programmieraufgaben, Musterlösungen etc. muss so organisiert sein, dass alle Studierenden die gleichen Versionen sehen und man bei Bedarf auch noch während des Semesters Korrekturen und Updates nachliefern kann.

Aber es gibt Hoffnung:

Web Based Computing: Die komfortabelste Lösung ist es, die gewünschte Software auf einem möglichst aus dem Internet erreichbaren Server so zu installieren, dass Studierende über Browser oder einen anderen Remote Access / Remote Desktop (z.B. VNC) darauf zugreifen können. **Jupyter Notebooks** sind interaktive Dokumente, die im Webbrowser bearbeitet werden und die u.a. Text und math. Formeln, Grafiken, Videos und ausführbaren Python-Code enthalten können. Jupyter Server laufen u.a. in der Microsofts Azure Notebook Cloud (<https://notebooks.azure.com/>).

Embedded Experiments: Embedded Systems wie PYNQ, Raspberry Pi oder Arduino kann man weitgehend mit den benötigten Files „bestücken“ bevor man sie an die Studierenden herausgibt. Dafür braucht es ein System zum einfachen Clonen von Betriebssystem und Dateien. Updates während des Semesters können bei einfachen Systemen ohne Internetzugang schwieriger sein.

Open Source: Ich verwende überwiegend Free Open Source Software (FOSS) und Freeware (z.B. LTSpice) bzw. selbst entwickelte Software (pyFDA), um der Lizenzproblematik zu entgehen. Allerdings ist die Dokumentation oft spärlich, und man muss zusätzliche Arbeit einplanen z.B. für Tutorials. Auf der anderen Seite ist der Support für FOSS z.B. in Foren oft besser als der offizielle Support für kommerzielle Software.

git: Das verteilte Versionierungssystem mit Hosting Services wie GitHub und GitLab (an der HM installiert) ist hervorragend geeignet, um Code von einem zentralen Punkt aus zu verteilen und so gleiche Versionsstände für alle zu ermöglichen. Problematisch ist allerdings, dass Jupyter Notebooks nicht nur Code sondern auch Grafiken, Versionsnummern etc. enthalten, die ein Diff / Merge von Notebooks schwierig machen. Abhilfe schafft das Projekt **nbdime** (siehe 2.5.3).

In den folgenden Unterlagen erfahren Sie wie Sie möglichst einfach mit interaktiven Jupyter Notebooks arbeiten können:

- Loggen Sie sich bei einem **Cloud-Service** ein, der Jupyter Server zur Verfügung stellt (Abschnitt 1.2) und clonen Sie meine Notebooks von GitHub.
- Laden Sie die Notebooks auf Ihren **lokalen Rechner** herunter und führen Sie sie dort aus (Python - Installation erforderlich, Abschnitt 1.3).
- Schließlich können Sie Notebooks statisch betrachten auf **GitHub** (<https://github.com/chipmuenk/dsp>)
- Einen eigenen **Notebook-Server an der Hochschule München** gibt es mangels Ressourcen momentan nicht.

... und jetzt wünsche ich viel Spaß mit der Pythonpflege:



Abb. 1: Python ist überall! [<http://xkcd.com/107/>]

1 Jupyter Notebooks

1.1 Einführung

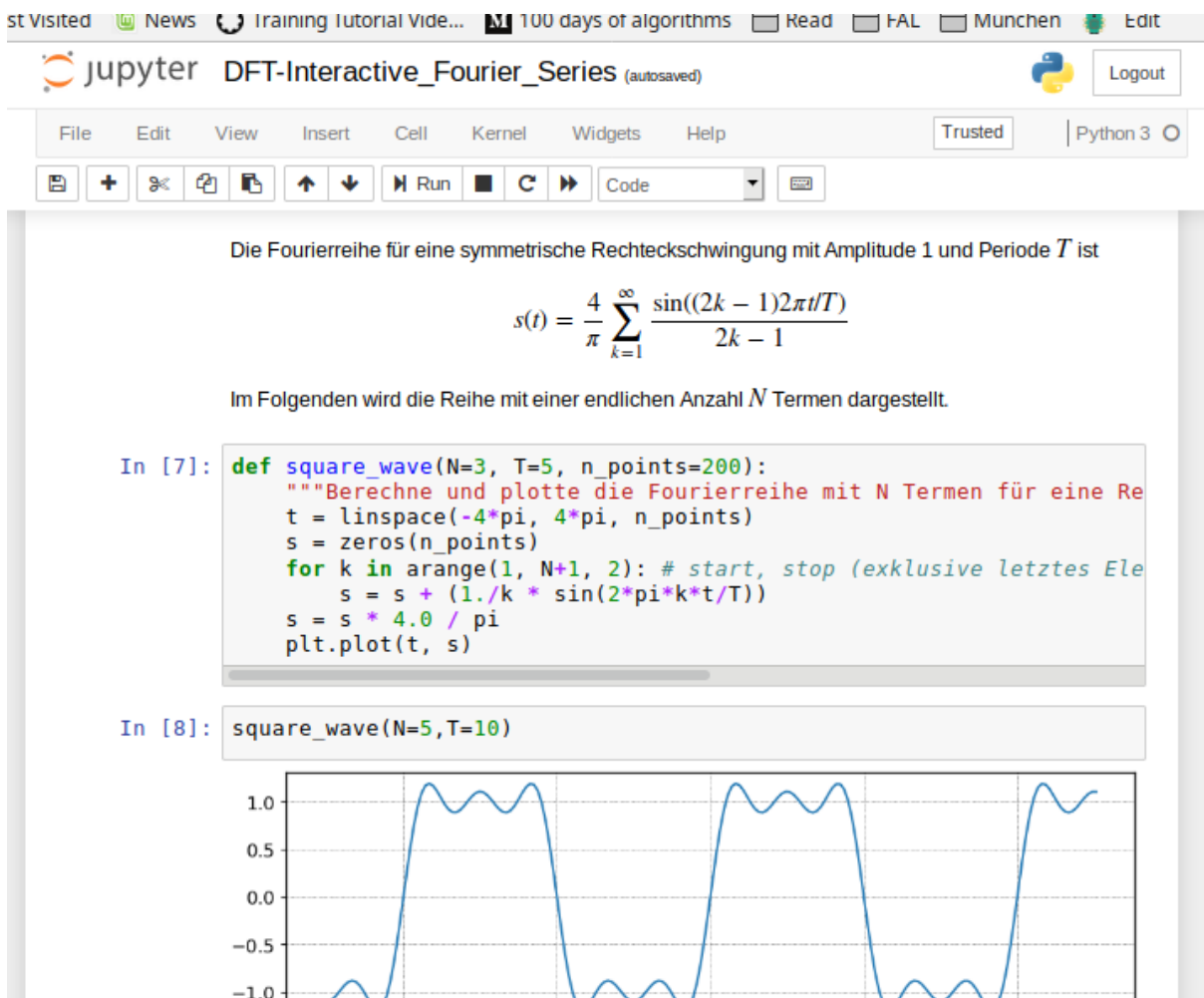


Abb. 1.1: Jupyter Notebook

In diesem Kurs arbeiten wir mit „Jupyter Notebooks“ (Abb. 1.1), das sind interaktive Dokumente, die Sie im Browser („Jupyter Client“) darstellen und bearbeiten können. Ein paar Python-Kenntnisse schaden nicht dabei.

Jupyter Notebooks werden in einer Server-Client Architektur (Abb. 1.2) zur Verfügung gestellt: Der Jupyter Client (das grafische „Frontend“) ist in Javascript programmiert und läuft im Ihrem Browser. Sie müssen nichts installieren.

Der Jupyter Server besteht aus dem Kernel, der den Code (Python oder eine andere Sprache) ausführt und dem Notebook Server, der Notebooks und Medien aufbereitet und verteilt. Der Server kann in der „Cloud“ (Abschnitt 1.2), auf dem eigenen PC (Abschnitt 1.3) oder natürlich

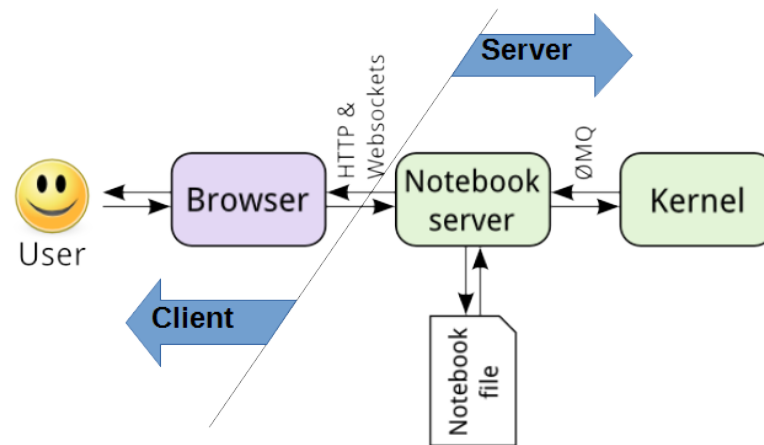


Abb. 1.2: Jupyter Architektur

auch im Intranet (wenn man denn Ressourcen hätte) werkeln. Wenn Sie sich für eine Variante entschieden haben, schildert Abschnitt 1.4 wie Sie mit dem Client Notebooks bearbeiten.

Notebooks sind zellenweise organisiert; eine Zelle kann entweder ausführbaren Code (Abschnitt 1.4.1) enthalten oder Rich Text (siehe 1.4.2) im Markdown Format (<http://de.wikipedia.org/wiki/Markdown>), einem Superset von HTML. Neben einfachen Textformatierungen wie **kursiv** oder ****fett**** und Überschriften können Markdown Zellen auch Latex-Formeln enthalten. Medien (Plots, Grafiken, Audio, Video, ...) können ebenfalls dargestellt bzw. abgespielt werden.

Notebooks *.ipynb werden als einzelne textbasierte JSON - Files¹ abgespeichert und können daher leicht weitergegeben werden. Sie können auch als statisches HTML oder als PDF dargestellt werden

```

{
  "metadata": {
    "name": "Part 1 - Running Code"
  },
  "nbformat": 3,
  "nbformat_minor": 0,
  "worksheets": [
    {
      "cells": [
        {
          "cell_type": "heading",
          "level": 1,
          "metadata": {},
          "source": [
            "Running Code in the IPython Notebook"
          ]
        }
      ]
    }
  ]
}

```

Lst. 1.1: Beispiel für JSON-Quelltext eines Jupyter-Notebooks

¹Java Script Object Notation, ein kompaktes Format zum Austausch von Daten zwischen Anwendungen, das von Menschen und Computern gelesen werden kann, ähnlich XML.

Plots werden auf dem Server berechnet, als BLOBs (Binary Large OBjects) übertragen und statisch im Browser dargestellt, daher können sie leider (noch) nicht gezoomt o.ä. werden. Mit Hilfe von <http://nbviewer.ipython.org> kann man sich Notebooks im Internet durch Angabe der URL auch ohne Python Installation als statische HTML Seiten anzeigen lassen. Notebooks auf GitHub werden auf diese Art automatisch gerendert.

Verwendet man die **plotly** Bibliothek (<https://plot.ly/python/line-charts/>) anstelle von **matplotlib**, sorgt Javascript für interaktive Plots (auch im statischen HTML-Format).

Notebooks lassen sich auch super als interaktives Laborbuch einsetzen, man kann gleichzeitig experimentieren und dokumentieren, so werden sie auch in dieser Lehrveranstaltung eingesetzt. Große Softwareprojekte sollte man nicht mit Jupyter entwickeln, dafür ist es nicht gedacht.

Jupyter Notebooks werden bereits seit 2001 entwickelt, zunächst unter dem Namen IPython Notebooks (man sieht es immer noch an der Dateiendung *.ipynb). Im Zuge einer Modularisierung des Codes wurde das Projekt umbenannt in **Jupyter** (<https://jupyter.org>). Der Name „JuPyteR“ setzt sich aus Julia, Python und R zusammen, den ersten drei unterstützten Programmiersprachen und spiegelt so wieder, dass außer Python zahlreiche andere (mittlerweile mehr als 60) Programmiersprachen unterstützt werden.

Inzwischen gehören IPython und Jupyter zum „Tafelsilber“ der Python-Community und stehen für offenes, kollaboratives und reproduzierbares Scientific und Technical Computing.

Lorena A. Barbas ausführlicher Artikel *Teaching and Learning with Jupyter*, <https://jupyter4edu.github.io/jupyter-edu-book/index.html> schildert die Vorzüge von Jupyter Notebooks in der Lehre.

1.1.1 Beispiel-Notebooks

Ein guter Startpunkt ist die Seite des Jupyter Projekts, ansonsten habe ich überwiegend Notebooksammlungen rund um die DSP ausgewählt. Sie können alle Notebooks (statisch) im Browser betrachten.

Jupyter-Projekt, *IPython-in-depth tutorial*, <https://github.com/ipython/ipython-in-depth>. Notebook-Tutorial mit Notebooks von den Jupyter-Machern.

Galerie mit interessanten Notebooks, <https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks>.

Karlijn Willems, *Jupyter Notebook Tutorial: The Definitive Guide*, <https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook>. Schönes kompaktes Tutorial zu Jupyter Notebooks.

J.R. Johansson, *Lectures on scientific computing with Python*, Notebook-basierte Einführung in Numpy / Scipy / Matplotlib und Sympy: <https://github.com/jrjohansson/scientific-python-lectures>

Kyle Mandli, <https://github.com/mandli> hat Notebooks zu Numerik und Statistik und Open Source Software, viele Sammlungen befassen sich mit Themen rund um Python.

Notebooks speziell zu DSP und verwandten Themen finden Sie unter

Allen Downey, *Think DSP*, <https://github.com/AllenDowney/ThinkDSP>. Sehr gute Kombination aus Buch (PDF oder Papier) und Notebooks zum Einstieg in die digitale Signalverarbeitung mit vielen Audiobeispielen.

Jean-François Bercher, „A Journey in Signal processing with Jupyter“, übersichtlich gestalteter Kurs zu DSP Grundlagen mit Buch und Notebooks (<https://github.com/jfbercher/LecturesSignalProcessing>), muss allerdings lokal installiert werden und benötigt Jupyter Extensions.

Sascha Spors, Uni Rostock:

- „Lecture notes for a masters course on Digital Signal Processing“, <https://github.com/spatialaudio/digital-signal-processing-lecture>
- „Selected Topics in Audio Signal Processing“, <https://github.com/spatialaudio/selected-topics-in-audio-signal-processing-lecture>
- „Wave Field / Sound Field Synthesis“, <http://python.sfstoolbox.org/en/latest/>

1.2 Notebooks in der Cloud

Wenn man Internetverbindung hat, ist es am bequemsten sich einen Notebookserver in der „Cloud“ zu suchen (und zu hoffen, dass der Server nicht ausgelastet ist).

Einen guten Überblick über kostenlose Services (Stand 2019) bietet <https://www.dataschool.io/cloud-services-for-jupyter-notebook/>.

Gesis <https://notebooks.gesis.org/>: Das Leibniz-Institut für Sozialwissenschaften stellt einen relativ flotten Server zur freien Verfügung, eigentlich nur für Sozialwissenschaftler. Mal schauen, wie lange es diesen Service noch gibt ...

Microsoft <https://notebooks.azure.com/> ist komfortabel, aber oft sehr langsam mit Time-Outs

LRZ <https://doku.lrz.de/display/PUBLIC/How+to+setup+a+Jupyter+Server+in+the+Cloud+for+Courses>, auch in unserem Netzwerk kann man natürlich Notebookserver einrichten, darf / muss sich aber um alles selbst kümmern. Das hätte Potenzial für die Zukunft ...

Google <https://colab.research.google.com> ist mehr auf AI-Anwendungen ausgelegt. Ich habe es nicht geschafft, Notebooks dort persistent abzulegen. Die Anbindung an Google Drive fand ich auch ziemlich unverständlich.

CoCalc <https://www.cocalc.com> ist auf Kollaboration ausgelegt (man kann zu mehreren gleichzeitig im gleichen Notebook arbeiten!). Das k.o. Kriterium für die kostenlosen Version ist der fehlende Zugang zum Internet (kein Github!)

Alle Services haben ihre Vor- und Nachteile; mich haben nur die Angebote von Gesis und Microsoft überzeugt.

1.2.1 Gesis Notebook Server

Erstellen Sie einen kostenlosen Account unter <https://notebooks.gesis.org/>. Das System bezeichnet sich als „Persistent BinderHub“, d.h. bei jedem Start wird ein Container ausgepackt, aber es gibt persistente Verzeichnisse so dass man nicht jedes Mal alles neu installieren muss.

Standard Libraries, Jupyterlab und git sind auch hier vorinstalliert, aber Sie müssen im Terminal ein paar Python Module nachinstallieren:

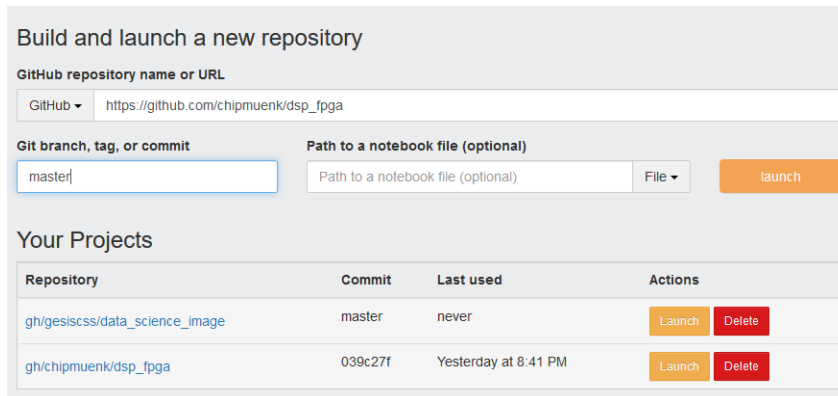
```
pip install numpy
pip install scipy
pip install matplotlib
pip install nbdime
```

Nach jedem Serverstart müssen die Module erneut installiert werden. Das können Sie vereinfachen, indem Sie den File `requirements.txt` im obersten Verzeichnis ablegen mit den zu installierenden Modulen:

```
numpy
scipy
matplotlib
nbdime
```

`pip install -r requirements.txt` installiert dann die enthaltenen Module, beim nächsten Start des Servers sollte der requirements File automatisch eingelesen werden.

Nach dem Einloggen können Sie ganz einfach existierende Repos z.B. von Github oder Ihrem lokalen Rechner auf dem Gesis Notebook Server installieren (Abb. 1.3).



The screenshot shows the Gesis Notebook Server interface. At the top, there's a section titled "Build and launch a new repository". It contains a form with the following fields:

- GitHub repository name or URL:** A dropdown menu set to "GitHub" and a text input field containing "https://github.com/chipmuenk/dsp_fpga".
- Git branch, tag, or commit:** A text input field containing "master".
- Path to a notebook file (optional):** A text input field and a dropdown menu set to "File".
- Launch button:** An orange button labeled "launch".

Below the form, there's a section titled "Your Projects" which contains a table with the following data:

Repository	Commit	Last used	Actions
gh/gesiscss/data_science_image	master	never	Launch Delete
gh/chipmuenk/dsp_fpga	039c27f	Yesterday at 8:41 PM	Launch Delete

Abb. 1.3: Clonen eines GitHub Repos in Gesis

Diese Repos und die git Infos bleiben zum Glück auch beim Neustart erhalten.

Sie können im Textfile `.gitignore` Files und Ordner angeben, die von git ignoriert werden sollen.

Beide Files sind bei meinem chipmuenk/dsp Repo bereits eingerichtet, bei eigenen Repos müssen Sie diese Anpassungen selbst vornehmen.

Clonen Sie jetzt das Notebook Repository (Branch main):

```
git clone https://github.com/chipmuenk/dsp
```

1.2.2 Microsoft Azure Notebooks

Sie benötigen nur ein Microsoft-Konto und können dann sofort loslegen und Github Repos clonen oder eigene Notebooks hochladen.

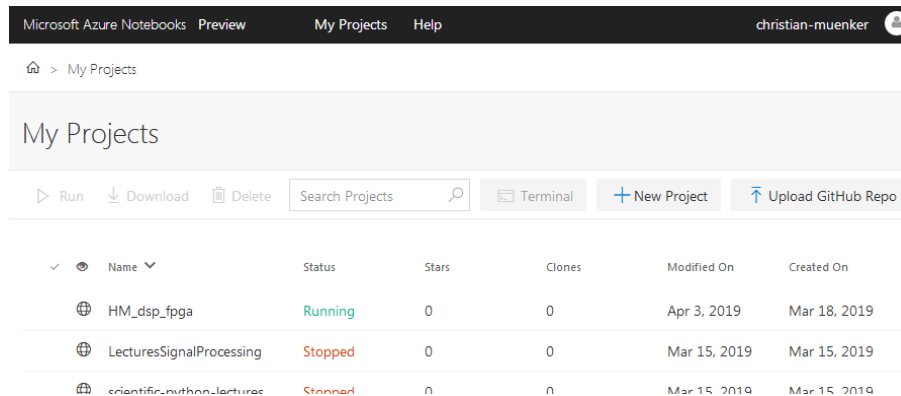


Abb. 1.4: Git Projekte auf Azure Notebooks

- Wechseln Sie in die Ansicht „My Projects“ (Abb. 1.4) und geben Sie über „Upload GitHub Repo“ den Namen des zu clonenden GitHub Repos an.
- Projekte, die Sie von GitHub gecloned haben, stehen unter git Management, Sie können mit git einfach aktualisierte Versionen von GitHub holen (`git pull` oder `git fetch origin`; `git merge`) oder Änderungen zu GitHub kopieren (`git push` wenn Sie die entsprechende Berechtigung haben). Öffnen Sie dazu aus dem Projekt ein Terminal (blau umrandet in Abb. 1.5) und wechseln Sie mit `cd library` in das eigentliche (git verwaltete) Projektverzeichnis.
- Eigene Files kann man leicht vom Computer oder von einer URL hochladen (rot gestrichelt umrandet in Abb. 1.5)

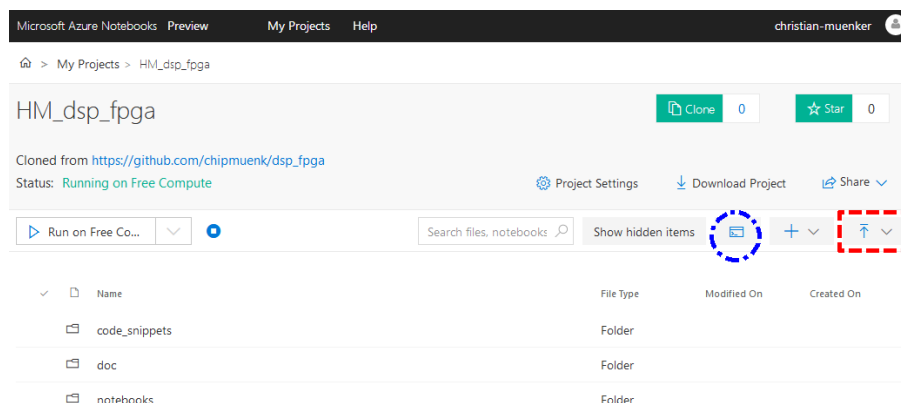


Abb. 1.5: Terminal öffnen in Azure Notebooks

Mit der rechten Maustaste können Sie hier Notebooks auch im Jupyterlab (Abschnitt 1.5) öffnen.

Konfigurieren des Projekts

Informationen unter <https://docs.microsoft.com/de-de/azure/notebooks/>. Softwareversionen können über einen `requirements.txt` File gewählt werden oder aus dem Terminalfenster mit `pip` installiert bzw. aktualisiert werden.

1.2.3 Software Installation / Update

Generell können Sie mit **pip** bzw. **pip3** Python Libraries installieren, z.B. für **nbdime** (Abschnitt 2.5.3):

```
pip3 install nbdime
```

bzw. updaten mit

```
pip3 install nbdime -U
```

1.3 Notebooks auf dem eigenen Rechner

Wenn Sie die Anaconda Distribution lokal installiert haben, starten Sie aus der Linux Console bzw. vom Anaconda Prompt

ein Jupyter Notebook mit **jupyter notebook**

oder JupyterLab mit **jupyter lab**

Achtung: Für Jupyter muss u.U. eine Ausnahmeregel zur (Windows-)Firewall hinzugefügt werden, da immer ein localhost Server gestartet wird - auch wenn Jupyter Server und Client auf dem gleichen Maschine laufen.

Im Browser öffnet sich nun das Jupyter Dashboard mit einer localhost URL (<http://127.0.0.1:8888/> oder ähnlich) in einem neuem Tab (Abb. 1.6).

Falls Python Module fehlen, müssen Sie diese selbst nachinstallieren (meist mit **conda install xxx**) oder **pip install xxx**):

- Erforderlich: Scientific Python Stack (numpy, scipy, matplotlib), jupyter
- Optional: ipywidgets, jupyterlab

Der Server wird beendet mit **<CTRL>-C <CTRL>-C** im Terminalfenster.

1.4 Jupyter-Client: Arbeiten im Browser

Haben Sie sich für eine Servervariante entschieden und sich eingeloggt, wird Ihnen der Toplevel des DSV Kurses mit allen Kapiteln zum Kurs angezeigt (Abb. 1.6).

Aufgrund der sehr reduzierten Benutzeroberfläche ist der Umgang mit Notebooks etwas gewöhnungsbedürftig. Einstiegshilfen bieten:

- <https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook>

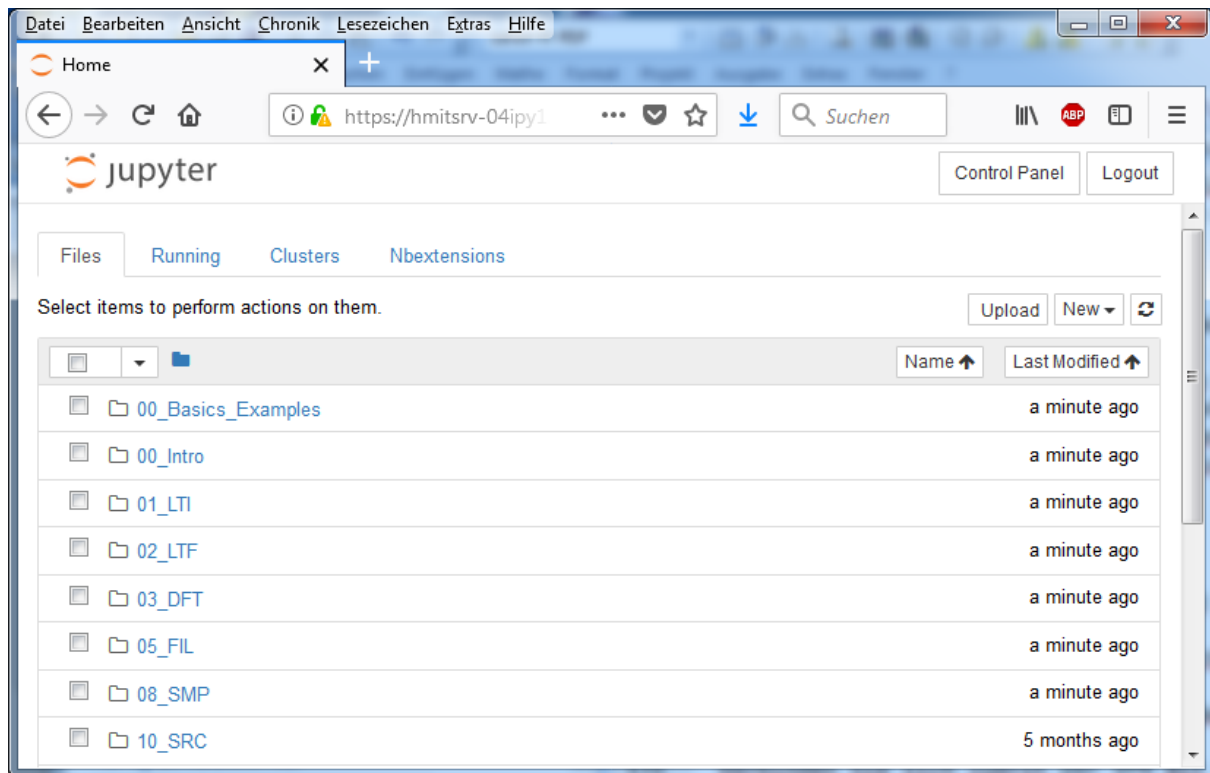


Abb. 1.6: Jupyter Navigator (Toplevel)

- http://nbviewer.jupyter.org/github/nicolasfauchereau/Python_NIWA_Wellington/blob/master/notebooks/IPython_notebook.html

Beim Download einzelner Notebooks aus dem Internet, bekommt man manchmal den HTML-Code anstelle des *.ipynb Files. Workarounds: In Git das gesamte Repository als zip-File herunterladen oder den File im Raw (JSON) Format kopieren.

Manchmal wirken Notebooks „tot“, das kann daran liegen dass der Python Kernel abgestürzt ist oder kein Python Kernel ausgewählt war. Mit Kernel → Restart oder Kernel → Change Kernel können Sie das Notebook wiederbeleben. Manchmal hängt auch nur eine Zelle in einer Schleife (erkennbar an dem [*] vor der Zelle), dann können Sie mit Kernel → Interrupt oder <CTRL>-C die Ausführung stoppen.

- Zellen können als „Code“ oder „Markdown“ (= Rich Text) gekennzeichnet werden (über Pulldownmenu in Jupyter).
- Die aktuell ausgewählte(n) Zelle(n) sind durch einen breiten blauen Balken gekennzeichnet. Wenn Sie darauf klicken, wird die Zelle kollabiert (Nur JupyterLab). Ausgewählte Zellen sind in einem von zwei Modi:

Edit Mode nach Eingabe von <RETURN>: Die Zelle ist blau umrandet, hat einen weißen Hintergrund und einen blinkender Cursor, Sie können jetzt beliebigen Text bzw. Code eingeben.

Command-Mode nach Eingabe von <ESC>: Die Zelle hat einen grauen Hintergrund und keinen Cursor, Sie können jetzt ganz Zelle löschen, verschieben, einfügen ...

- Mit **<SHIFT>-<RETURN>** wird eine Textzelle gerendert bzw. eine Code-Zelle aufgeführt und die nächste Zelle selektiert.
- **<ALT>-<RETURN>** macht fast das gleiche, allerdings bleibt die ursprüngliche Zelle selektiert.

Zellen können man einzeln („Cell → Run“) oder alle („Cell → Run All“) nacheinander ausgeführt bzw. gerendert werden; Variablen und Funktionen aus bereits ausgeführten Zellen können verwendet werden.

Genau wie bei Matlab, „merkt“ sich der Interpreter Variablen zwischen den Läufen, auch wenn Sie die im Quelltext u.U. gelöscht haben. Starten Sie daher im Notebook ab und zu den Kernel neu, damit „tote“ Variablen aus dem Speicher gelöscht werden.

1.4.1 Code - Zellen

<TAB> im löst im Command Mode wie üblich eine Command Completion aus. Achtung:

```
numpy as np
np.arr<TAB>|
```

innerhalb einer Zelle funktioniert nicht, da zum Zeitpunkt der Command Completion np noch nicht bekannt ist. Abhilfe: numpy in einer vorherigen Zelle importieren (oder Zelle einmal erfolgreich ausführen).

<SHIFT><TAB> zeigt den Tooltip an, ausführlichere Infos werden mit angehängtem ? angezeigt, z.B. mit print?

%magic: Ein %% am Anfang einer Code-Zelle leitet ein „Zell-Magic“ ein, ein Jupyter Makros das sich auf die ganze Zelle bezieht, z.B. misst **%%timeit** die Laufzeit des gesamten Codes in dieser Zelle.

Zeilen, die mit einem % anfangen sind „Line-Magics“:

- **%lsmagic** listet alle verfügbaren Line- und Zell-Magics auf. Siehe auch <https://www.youtube.com/watch?v=zxkd007L29Q>.
- **%time** misst die Laufzeit des Codes in der restlichen Zeile, z.B. `%time sum(range(1000))`.
%timeit misst die Laufzeit mehrfach und zeigt den Mittelwert an.
- **%matplotlib inline** zeigt matplotlib Plots im Notebook an anstatt in einem Extrafenster
- **%run** führt externe Python Skripte und Notebooks aus und zeigt die Ausgabe im Notebook an
- **%debug** kann in einer folgenden Zelle verwendet werden, nachdem die Code-Ausführung mit einem Fehler (exception) angehalten wurde. Sie können z.B. Variablenwerte vor der Exception anzeigen lassen.

1.4.2 Text / Markdown - Zellen

Text-Zellen werden in Markdown beschrieben, einem Superset von HTML. Für spezielles Layout, z.B. für Tabellen oder Sonderzeichen kann man daher auch mit HTML nachhelfen.

Ganz wichtig: Abschnitte in Markdown Zellen müssen durch Leerzeilen von einander getrennt werden, Einrückungen sind Teil der Syntax!

Überschriften: Ein bis vier #, gefolgt von einem Leerzeichen,

```
#    Titel
##   Überschrift Level 1
###  Überschrift Level 2
#### Überschrift Level 3
```

Hervorhebungen: Fett: `__Text__` oder `**Text**`, kursiv: `_Text_` oder `*Text*`

Formeln und mathematische Zeichen können wie in \LaTeX eingegeben werden, z.B. `$c^2 = a^2 + b^2$`, sie werden im Browser mit MathJax gerendert, der Browser muss die mathjax.js Bibliothek beim ersten Mal aus dem Netz nachladen können. Formeln in einer eigenen Zeile werden zwischen `$$... $$` eingeschlossen.

Monospace Font für z.B. Code oder Filenamen bekommt man mit einzelnen Back Ticks, also z.B. ``Text``.

Zeilenumbrüche kann man erzwingen mit `
` [HTML].

Farben erzielt man z.B. mit `Text`. Nicht alle Markdown-Konstrukte funktionieren zwischen Font-Tags, also nicht wundern. [HTML]

Einrückungen erhält man mit einem „Größer“ Zeichen (`>`) gefolgt von einem Leerzeichen und dem Text. Alles bis zum nächsten Carriage Return wird eingerückt.

Bullets: Bindestrich oder Stern gefolgt von zwei Leerzeichen oder Leerzeichen, Bindestrich, Leerzeichen am Anfang einer Zeile erzeugen eine Bullet-Liste. Unterlisten müssen mit einem Tab eingerückt werden.

Aufzählungen: Der erste Punkte beginnt mit `1.` gefolgt von einem Leerzeichen, die nächsten Zeilen müssen mit irgendeiner Zahl, einen Punkt und einem Leerzeichen beginnen. Unter-aufzählungen müssen mit einem Tab eingerückt werden..

Tabellen: Einfache Tabellen können mit Markdown gestaltet werden:

```
| Header1 | Header2 |
|-----|-----|
| Data 1  | Data 2   |
```

ansonsten kann man HTML-Code verwenden.

Sonderzeichen kann man über UTF-8 Zeichen über deren (hexa)dezimalen Code erzeugen mit `&#reference_number`, siehe z.B. https://www.w3schools.com/charsets/ref_html_utf8.asp. Beispiel: `■` oder `■` ergibt ein schwarzes Quadrat [HTML].

Horizontale Linien bekommt man mit drei Sternen `***`

Interne Links: Einen Link zu einer Überschrift erstellt man mit `[section title] (#section-title)`. Die Sprungmarke (Text zwischen Klammern) darf keine Leerzeichen oder Sonderzeichen enthalten, diese müssen durch Bindestriche ersetzt werden.

Mit `` kann man einen Anker oberhalb eines Abschnitttitels setzen und darauf verlinken mit `__[section titel] (#section_ID)`

Externe Links bekommt man mit `__[link text] (http://meine_url)__`

1.4.3 Zellen mit Medien

Abbildungen: Man kann Bilder per Drag & Drop in Markdown Zellen ziehen, sie werden dann als Binärobjekt im Notebook abgespeichert. Sauberer ist es, Abbildungen zu referenzieren:
``

In Codezellen können nur Grafiken aus dem Netz eingebunden werden, Bildunterschriften sind (noch) nicht möglich:

``

Videos: Lokale Videos können in Markdown-Zellen über HTML5 Tags eingebunden werden:

`<video controls src="../../../images/animation.m4v" />`

Videos aus dem Internet werden in Code-Zellen eingebunden über

```
from IPython.display import HTML, YouTubeVideo
YouTubeVideo("r18Gi8lSkfM", start=0, autoplay=0, theme="light", color="red")
HTML('<iframe width="600" height="350"
src="https://youtube.com/embed/r18Gi8lSkfM" </iframe>')
```

1.5 JupyterLab - IDE im Browser

Jupyter Notebooks werden manchmal als „amazingly featureless“ bezeichnet, JupyterLab wurde daher als „Next-Generation Web-Based User Interface for Project Jupyter“, also quasi als browserbasierte IDE konzipiert. Das Notebook-Dateiformat (*.ipynb) blieb dabei unverändert, ebenso wie die Kommunikation mit dem Notebook Server (JupyterHub funktioniert nach wie vor).

Die beiden Seiten

http://jupyterlab.readthedocs.io/en/stable/getting_started/starting.html

und

<https://jupyterlab.readthedocs.io/en/stable/user/interface.html>

geben einen bunten ersten Einblick.

Neu in JupyterLab sind:

- Multi-Tab, Multi-Window Interface mit Drag & Drop
- Side-by side editing: Z.B. Markdown-Editor und Viewer
- Zellen können per Doppelklick ausgeblendet werden
- Single-Document Mode (View -> Single-Document Mode) für mehr Platz

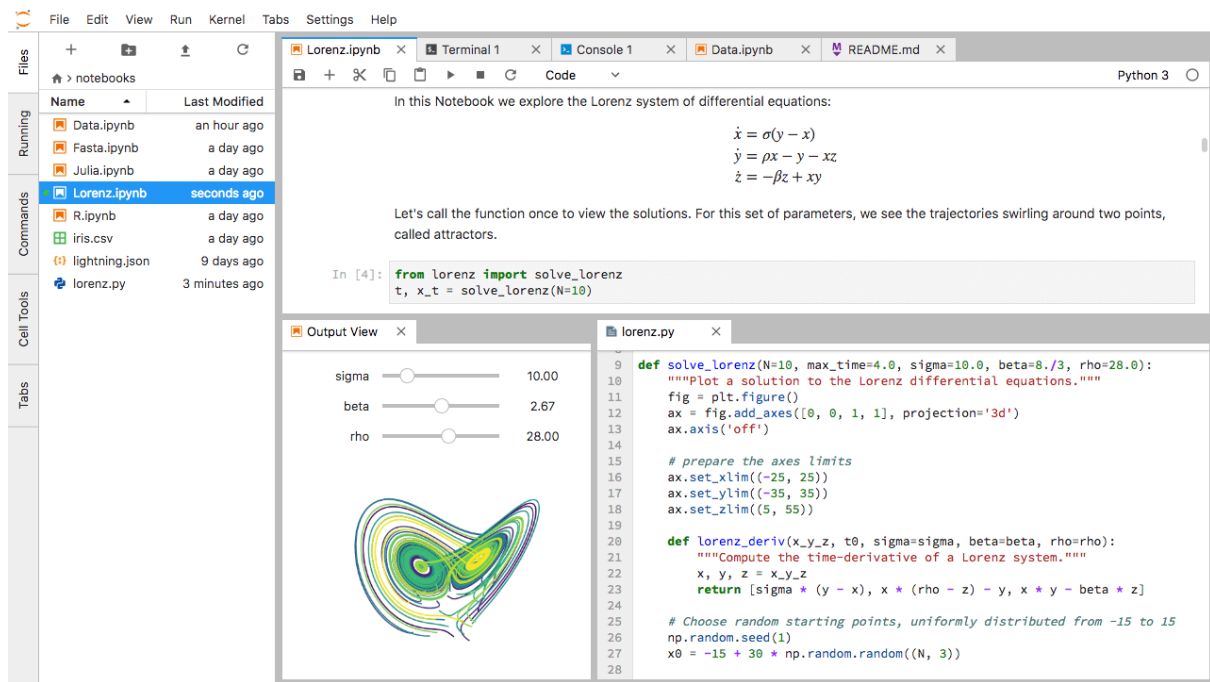


Abb. 1.7: IDE im Browser: JupyterLab

- Side-bar Navigation
- Verbesserter File-Browser
- Einfacher Wechsel zwischen Kernels
- Terminal, Texteditor, Image-Viewer
- Schneller CSV-Viewer - Scrollen von Tabellen mit 1 Million Zeilen ist problemlos möglich.

Sie starten JupyterLab aus dem Terminal mit **jupyter lab**, ggf. installieren Sie es mit

conda install jupyterlab bzw. **pip install jupyterlab**.

Da JupyterLab eine Erweiterung des klassischen Jupyter Notebook Servers ist, kann man auch zunächst ein Jupyter Notebook starten (`jupyter notebook`) und dann die JupyterLab URL (normalerweise <http://localhost:8888/lab>) anstatt der Default-URL <http://localhost:8888/tree> besuchen.

Man kann jederzeit zwischen JupyterLab und Jupyter wechseln, sollte aber vorher Notebooks speichern und die Kernel stoppen, sonst können u.U. Inkonsistenzen auftreten.

Umgekehrt kann man von JupyterLab aus der Seitenleiste aus **Help -> Launch Classic Notebook** ein klassisches Notebook-Interface öffnen. Alternativ ändert man den Pfad von **/lab** to **/tree**.

Wer wissen möchte, wie man möglichst effizient mit Notebooks arbeitet, dem empfehle ich den Artikel unter https://florianwilhelm.info/2018/11/working_efficiently_with_jupyter_lab/.

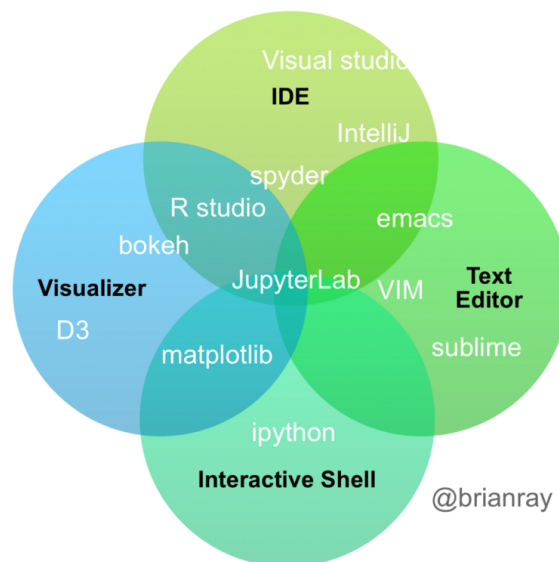


Abb. 1.8: JupyterLab: Im Mittelpunkt

1.6 Blogs und Videos zu Jupyter-Notebooks / JupyterHub / JupyterLab

Brian Granger et al., „Jupyter Frontends: From the Classic Jupyter Notebook to JupyterLab, nteract, and Beyond“, <https://www.youtube.com/watch?v=YKmJvHjTGAM>, JupyterCon NY, Aug. 2017, 30 + 10 min. Die Geschichte und Motivation von Jupyter Notebooks mit technischen Details der Implementierung.

1.6.1 Jupyter Notebooks - Computing im Browser

Jupyter4edu „Teaching and Learning with Jupyter“, <https://jupyter4edu.github.io/jupyter-edu-book/>, 2019. DAS Buch zum Einsatz von Jupyter in der Lehre!

, [TeachingremotelywithJupyter:tips,resources,andbest-practices](#)

Google Groups, „Teaching with Jupyter Notebooks“:

„Reproducible Code with a Narrative“. <https://groups.google.com/forum/#!forum/jupyter-education>

Carl Sandrock, [Are we allowing students to avoid learning via notebooks?](#), 2018.

Corey Schaefer, „Jupyter Notebook Tutorial: Introduction, Setup, and Walkthrough“, Video (30“), 2017. <https://www.youtube.com/watch?v=HW29067qVWk>

Mark Jay, „Learn Jupyter Notebooks (Pt. 1): Plotting“, https://www.youtube.com/watch?v=Hr4yh1_4GlQ&t=1s, 2017, 14 min. Sehr gute Einführung! Mehr Teile bei Youtube.

5agado, „Interactive Visualizations in Jupyter-Notebooks“, 2017. <https://towardsdatascience.com/interactive-visualizations-in-jupyter-notebook-3be02ab2b8cd>, Überblick über verschiedene Grafikbackends (statisch / interaktiv)

Nazif Berat, „Boost Your Jupyter Notebook Productivity“, 2017. <https://towardsdatascience.com/jupyter-notebook-hints-1f26b08429ad>

Adrien Lina, „Present Your Data Science Results in a Jupyter Notebook, the Right Way: How to import one Jupyter notebook into another“, 2017. <https://blog.sicara.com/present-data-science-results-jupyter-notebook-import-into-another-108433bc8505> - a neat trick to circumvent fiddling with the python path.

Kyle Kelley, „nteract on jupyter“, 2018. <https://blog.nteract.io/nteract-on-jupyter-53cc2c38290d>

Ehi Aigiomawu, „Introduction to Matplotlib - Data Visualization in Python“, 2018. <https://heartbeat.fritz.ai/introduction-to-matplotlib-data-visualization-in-python-d9143287ae39>

Will Koehrsen, „Interactive Controls in Jupyter Notebooks“, <https://towardsdatascience.com/interactive-controls-for-jupyter-notebooks-f5c94829aee6>, 2019.

1.6.2 JupyterLab - Next Generation Browser IDE

Schockwellenreiter, „Wird JupyterLab das neue R-Studio?“, <http://blog.schockwellenreiter.de/2017/12/2017121201.html>, Dez. 2017.

Brian Granger et al., „JupyterLab: The Next-Generation Jupyter Frontend“, <https://www.youtube.com/watch?v=w7jq4XgwLJQ>, JupyterCon NY, Aug. 2017, 30 + 10 min.

Brian Granger et al., „JupyterLab+Real Time Collaboration“, <https://www.youtube.com/watch?v=dSjvK-Z3o3U>, 2017. Live-Demo von Jupyter auf der PyData Conference Seattle 2017 mit Fokus auf Data Science und kollaborativem Arbeiten - sehr eindrucksvoll! (20 min. + 10 min. Diskussion).

Brian Ray, „JupyterLab first impressions“, https://medium.com/@brianray_7981/jupyterlab-first-impressions-e6d70d8a175d, 2017.

Chad Lagore, „Jupyter Notebooks are Breathtakingly Featureless - Use Jupyter Lab“, 2017. <https://towardsdatascience.com/jupyter-notebooks-are-breathtakingly-featureless-use-jupyter-lab-be858a67b59d>

JupyterLab kann auch leicht gemeinsam mit JupyterHub verwendet werden, da Dateiformat und Sicherheitskonzept wie bei Jupyter sind:

<http://jupyterlab.readthedocs.io/en/stable/user/jupyterhub.html>

We use the Jupyter Labs interface, which we access by adding the /lab address to the standard `pynq:9090` (<http://pynq:9090/lab>). This opens up a range of new features for developing and running our PYNQ applications.

Upon first glance, Jupyter lab appears much more like a traditional IDE, and includes new features such as the ability to drag and drop cells between note books and associate terminals with notebooks for scratch outputs.

1.6.3 JupyterHub - Und jetzt alle!

Carol Willing, PyData Carolinas, *JupyterHub: A „things explainer overview“*, <https://www.youtube.com/watch?v=4GJFNQBB26s>, 2016.

Ryan Lovett und Yuvi Panda, UC Berkeley, *Managing a 1,000+ Student JupyterHub without Losing Your Sanity*, <https://www.youtube.com/watch?v=ivswAxysfTk>, 2017.

2 Git für Jupyter Notebooks

Es liegt nahe, **git** zur Versionierung und zum Update der Notebooks in diesem Kurs zu verwenden, da die Notebooks auf GitHub gehostet sind und git auf allen Server und Client Plattformen installiert oder installierbar ist.

In diesem Kurs arbeiten wir mit dem Repo

<https://github.com/chipmuenk/dsp>, Branch: **main**.

Anmerkung: Mit „Remote Repo“ oder „origin“ ist hier immer das „Ur-Repo“ gemeint auf GitHub / GitLab

Kurzanleitung: <https://thomas-leister.de/git-fuer-einsteiger/>

2.1 Repository Clonen

Mit dieser Operation holen Sie ein remote Repository auf Ihren Rechner (oder Ihren Cloudspeicher), behalten aber im Gegensatz zu einer einfachen Kopie den Bezug dazu. Wenn das Remote Repo später geändert wird, können Sie die Änderungen mit Ihrem lokalen Stand zusammenführen (merge).

2.1.1 Microsoft Azure

Mit Microsofts Azure Notebooks ist das besonders einfach: Mit „Upload GitHub Repo“ auf der Startseite erstellt man schnell eine Kopie eines beliebigen (öffentlichen) GitHub Projekts. Alternativ kann man mit einem öffentlichen Link (z.B. <https://notebooks.azure.com/christian-muenker/projects/dsp>) ein Notebook direkt von Microsoft Azure klonen.

2.1.2 Gesis

Auch hier gibt es ein komfortables UI (Abb. 2.1) für die Konfiguration, Sie müssen nur die URL eines öffentlichen Repos eintragen (s.o.).

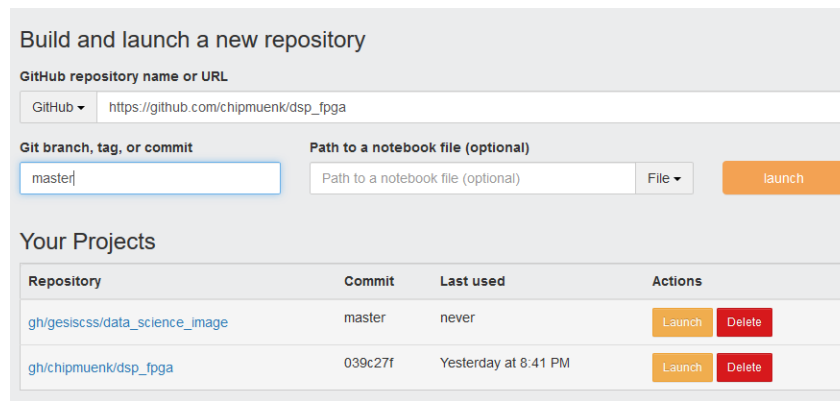


Abb. 2.1: Clonen eines GitHub Repos in Gesis

2.1.3 Terminal (alle Jupyter Server)

Vom Terminal ist das nicht viel schwieriger: Gehen Sie in das Notebook Home Verzeichnis:

Azure /library/

Gesis /

und setzen Sie von dort den Befehl ab

git clone https://github.com/chipmuenk/dsp

Dort wird dann das Verzeichnis **dsp** unter git Management angelegt, das eine Kopie des angegebenen Repos enthält.

2.2 Update vom Remote Repo holen

Bevor Sie loslegen, sollten Sie jedes mal Ihr System auf den Stand des Remote Repos bringen. Und wenn Ihr Dozent mal wieder alle Notebooks geändert hat und Sie zähneknirschend Ihr geclontes Repo aktualisieren müssen, gehen Sie am besten so vor:

git status

zeigt den Zustand Ihres Workspaces. Dort stehen alle Änderungen, die noch nicht in Ihr lokales oder das remote Repo eingchecked sind.

Vermutlich sehen Sie jetzt einige geänderte Notebooks: Jedes Mal wenn Sie ein Notebook ausführen, ändern sich Metadaten und Output Zellen. Unterschiede lassen Sie sich mit

nbdiff <geändertes_notebook.ipynb>

aus dem nbdime Module anzeigen. Wenn die Unterschiede unwichtig sind, können Sie sie verwerfen mit:

git checkout -- . (alle Änderungen verwerfen) oder

git checkout -- geaenderter_file.ipynb (ein File verwerfen) oder

git checkout -- notebooks/*.ipynb (alle Notebooks verwerfen)

Änderungen, die Sie behalten möchten, speichern Sie *vorher* unter einem neuen Filenamen ab.

Alternativ können Sie mit

git stash

die lokalen Änderungen auf einen „Stack“ schieben und sie später mit

git stash apply

wieder zurückholen.

Wenn in alle lokalen Änderungen committed / eingchecked / verworfen worden sind, holen Sie den Stand des Remote Repos ab mit

git fetch

und vereinigen die Version mit Ihrem lokalen Repo:

git merge

Die Anweisung **git pull** führt beide Anweisungen nacheinander aus.

Änderungen, die Sie mit **git add** bereitgestellt (s.u.) haben, muss man explizit „unstagen“ und den Zeiger auf die Spitze des lokalen Repos setzen mit

git reset HEAD <Filename> bzw. für alle Files mit **git reset HEAD .**

Treten beim **git merge** Konflikte auf, kann man die Übernahme der Version aus dem Remote Repo erzwingen mit

```
git pull
```

```
git checkout --theirs .
```

```
git merge
```

Alternativ kann man einen harten Reset auf den Stand des Remote Repos machen:

```
git fetch origin
```

```
git reset --hard origin/master
```

Die lokale Variante behält man mit:

```
git checkout --ours .
```

```
git add .
```

```
git commit .
```

2.3 Einchecken

git status

zeigt „Änderungen“ in Ihrem Workspace an.

Änderungen, die Sie nicht behalten möchten können Sie zurücksetzen auf den Stand Ihres lokalen Repos mit

git checkout -- . (alle Änderungen verwerfen) oder

git checkout -- geaenderter_file.ipynb (ein File verwerfen)

Änderungen, die Sie behalten möchten (z.B. eigene Notebooks), stellen Sie zunächst bereit (staging) für das Einchecken in Ihr lokales Repo mit

git add . (alle geänderten Files) bzw. **git add my_file**

Dann „legen Sie sich fest“ (commit) mit einer Änderungs-Message

git commit -m "was ist neu"

2.4 Update Remote Repo

Wenn Sie Schreibberechtigung im Remote Repo haben, können Sie auch die lokalen Änderungen dorthin schreiben. Holen Sie zunächst den aktuellen Stand des remote Repos mit

git fetch

Dann „stagen“ Sie alle Änderungen (d.h. stellen sie bereit fürs Einchecken in Ihr lokales Repo):

git add . (alle Änderungen) bzw. **git add my_file**

git status zeigt Ihnen jetzt nicht einfach „Änderungen“, sondern „Zum Commit vorgemerkte Änderungen“ an.

2.4.1 Neues Notebook

1. Notebook **mynotebook.ipynb** auf dem eigenen PC in lokalem git Repo erstellen, testen und committen
2. Upload zum (remote) GitHub / GitLab Repo mit **git push mynotebook.ipynb** [nur Dozent]
3. **git pull** vom GitHub / GitLab Repo ins Azure Repo oder auf eigenen PC [Studierender]

Alternativ kann man ohne Versionierung (git / GitHub Notebooks) direkt vom Rechner ins Azure Projekt hochladen.

2.4.2 Update von Notebooks auf lokalem PC

1. Editieren eines Notebooks **changed_nb.ipynb** auf dem lokalen PC
2. Stagen des geänderten Notebooks mit **git add .** (oder **git add changed_nb.ipynb**)
3. Committen des geänderten Notebooks mit **git commit .** (oder **git commit changed_nb.ipynb**)
4. Weitergeben der Änderungen an das Remote Repo mit **git push** [nur Dozent]
5. Weiter wie oben mit **git pull** aus dem Azure Projekt

2.4.3 Update von Notebooks direkt auf dem Azure Repo (nur Owner)

Alternativ kann man Notebooks auch direkt auf dem Azure Repo editieren oder neu anlegen.

1. Auf Azure Repo anmelden und Notebook neu anlegen oder editieren
2. Änderungen per `git push` vom Azure Repo an den Remote Repo weitergeben [nur Dozent]
3. Änderungen per `git pull` vom Remote Server auf den lokalen PC holen

2.5 Ein paar Git-Kommandos

Aus dem Azure Repo kann man ein Terminal öffnen, das Projekt befindet sich im Unterverzeichnis **library**. Dort kann man mit **git** im Textmodus arbeiten, und sich z.B. mit **git status** den Zustand den Repos anzeigen lassen. Auf das Remote Repo kann man natürlich nur mit entsprechender Berechtigung z.B. mit `git push` zurückschreiben.

2.5.1 Basics

git status Status des lokalen Repos und Workspace

git clone https://xxx Mache eine Kopie des angegebenen Remote Repos

git fetch Hole Änderungen vom Remote Repo

git fetch; git reset --hard origin/master Hole Änderungen vom Remote Repo

git merge Führe einen Merge zwischen Local Repo und den geholten Stand des Remote Repos durch

git pull (= git fetch und git merge) Hole Änderungen vom Remote Repo und merge sie

git checkout --theirs .; git add . Wenn man bereits im Conflict Mode ist und einfach alle Konflikte mit „--theirs“ lösen will (s.u.). Den umgekehrten Fall erreicht man, indem man „--ours“ angibt. Anstelle von „.“ kann man auch einzelne Files spezifizieren.

git add . Stagen (Bereitstellen) aller lokalen Änderungen

git commit Commit-Messages werden in vi eingegeben, „a“ fügt daher eine Zeile hinzu, „<ESC>:wq“ beendet den „add“ Modus, schreibt und beendet vi.

git push Schiebe Änderungen zum Remote Repo

Tipp: Bei jedem Aufruf eines Notebooks werden dessen Metadaten aktualisiert. Auch wenn man sonst nichts verändert hat, ist das Notebook für git daher „modified“. Solange die Files nicht per `git add` gestaged wurden, kann man die Änderungen per

```
git checkout -- <Filename>
```

(einzelnes File) bzw.

```
git checkout .
```

(alle Files) auf den letzten eingetragten lokalen Stand zurücksetzen. Nach einem `git add` muss man explizit „unstagen“ und den Zeiger auf die Spitze des lokalen Repos setzen mit

```
git reset HEAD <Filename>
```

bzw. für alle Files mit

```
git reset HEAD .
```

Tipp: Bei Konflikten nach einem `git pull` vom Remote Repo erzwingt man die Übernahme der Version aus dem GitHub Repo (lokale Änderungen werden überschrieben!) mit

```
git pull
git checkout --theirs .
git merge
```

Alternativ kann man einen harten Reset auf den Stand des Remote Repos machen:

```
git fetch origin
git reset --hard origin/master
```

Die lokale Variante behält man mit:

```
git checkout --ours .
git add .
git commit .
git push
```

2.5.2 Stand des Remote-Repos erzwingen

Wenn man in seinem Workspace mit Notebooks „herumgespielt“ hat (Öffnen und Schließen reicht schon), ist das Notebook dort auf einem anderen Stand als auf dem Remote Repo, ein automatisches `git pull` bzw. `git merge` ist nicht mehr möglich. Möchte man den Stand des lokalen Repos auf den Stand des Remote Branches zwingen, holt man zunächst diesen Stand und setzt dann den lokalen Branch auf genau diesen Stand zurück:

```
git fetch origin
git reset --hard origin/master
```

Falls man doch eine relevante Änderung am Notebook vorgenommen hat, kann man den lokalen Stand vorher in einen neuen Branch „my-saved-work“ sichern:

```
git commit -a -m "Saving my work, just in case"
git branch my-saved-work
```

Seit git Version 1.6.1 gibt es bei Mergekonflikten die einfachere Variante: Nach `git fetch`; `git merge` (oder kürzer `git pull`)

Dabei ist **git fetch** das Kommando, das die lokale Kopie des Remote Repos (irgendwo unter `.git/refs/remote/original` auf den aktuellen Stand des Remote Repos bringt. **git merge** versucht den Stand des lokalen Workspace

siehe auch <http://ndpsoftware.com/git-cheatsheet.html> (interaktives Cheatsheet) oder die Diskussion unter https://stackoverflow.com/questions/292357/what-is-the-difference-between-git-pull-and-git-fetch?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

```
git checkout --theirs .
git add .
```

bzw. um die lokale Variante zu behalten:

```
git checkout --ours .
git add .
```

„Ours“ bezieht sich dabei auf den momentanen Branch; „theirs“ auf den branch, in den gemerged wird.

2.5.3 Versionierung von Notebooks mit nbtime

Wie bereits beschrieben, tut man sich bei Notebooks schwer, in git zwischen wesentlichen und unwesentlichen Änderungen zu unterscheiden.

Das Projekt **nbtime** (Doku) stellt Tools zur Verfügung für komfortables Jupyter Notebook Diff und Merge unter git.

Unter Azure Notebooks ist **nbtime** bereits vorkonfiguriert, bei Gesis und auf Ihrem eigenen Rechner können Sie es installieren mit **pip install nbtime** bzw. **conda install -c conda-forge nbtime**

Sie integrieren es in git mit **nbtime config-git --enable --global**

Danach kann man im Terminal Unterschiede zwischen Notebooks anzeigen lassen mit `nbdiff`:

```
nbdiff my_notebook.ipynb
```

```
nbdiff notebook_1.ipynb notebook_2.ipynb
```

git diff oder **git merge** sollten jetzt Aliase auf die entsprechenden nbtime Kommandos sein. (alias für das entsprechende nbtime Kommando).

or viewing a rich web-based rendering of the diff with `nbdiff-web`:

```
nbdiff-web notebook_1.ipynb notebook_2.ipynb
```

2.6 GitLab

Kostenfrei kann man auf GitHub nur öffentliche Projekte hosten, alternativ gibt es an der HM einen GitLab Server, den man für nicht-öffentliche Zwecke nutzen kann. Die Funktionalität von GitLab ist sehr ähnlich zu der von GitHub.

2.6.1 Projekte

Allgemein kann man in gitlab genau wie in github Projekte managen, kann aber hier den Zugriff auf eingeloggte oder sogar ausgewählte User einschränken. In gitlab legt man ein leeres Projekt `tolles_projekt` an und clond das Projekt dann auf den eigenen Rechner mit z.B. **`git clone hm-cmuenker@gitlab.lrz.de/hm-cmuenker/tolles_projekt.git`**

An der Stelle, an der git aufgerufen wurde, entsteht ein (leerer) Projektordner `tolles_projekt` (genau wie beim Arbeiten mit github), den Sie mit Dateien und Ordnern bestücken können. Sie können auch innerhalb von `tolles_projekt` andere git Projekte clonen mit z.B. mit `git clone https://github.com/python-acoustics/python-acoustics.git` Das Projekt wird mit gitlab synchronisiert mit:

```
git add .
git commit „initial commit“
git push -u origin master
```

Abbildungsverzeichnis

1	XKCD: Latex rules	ii
1	XKCD: Python ist überall!	vi
1.1	Jupyter Notebook	1
1.2	Jupyter Architektur	2
1.3	Clonen eines GitHub Repos in Gesis	5
1.4	Git Projekte auf Azure Notebooks	6
1.5	Terminal öffnen in Azure Notebooks	6
1.6	Jupyter Navigator (Toplevel)	8
1.7	IDE im Browser: JupyterLab	12
1.8	JupyterLab: Im Mittelpunkt	13
2.1	Clonen eines GitHub Repos in Gesis	18

Tabellenverzeichnis

Listings

1.1 Beispiel für JSON-Quelltext eines Jupyter-Notebooks 2