Teaching and Learning with Jupyter

Lorena A. Barba et al.

2018-11-30

Contents

Chapter 1

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Chapter 2

Introduction

Project Jupyter is a broad collaboration that develops open-source tools for interactive and exploratory computing. The tools include: IPython, the Jupyter Notebook, Jupyter Hub, and an ecosystem of extensions contributed by a large community. Jupyter Notebook exploded in popularity since late 2014, fueled by its adoption as the favorite environment for doing data science. It has also grown as a platform to use in the classroom, to develop teaching materials, to share lessons and tutorials, and more. Notebooks are documents containing text narratives, combined with executable code (many languages are supported) and the output of that code. This marriage of content and code makes for a powerful new form of data-based communication. Educators everywhere are adopting Jupyter for teaching.

This handbook is for any educator teaching a topic that includes data analysis or computation to support the learning—not just courses in engineering or science, but also data journalism, business and quantitative economics, data-based decision sciences and policy, quantitative health sciences, and others. It aims to give an entry point, and a broad overview of Jupyter in education. Whether you are already using Jupyter to teach, you have found learning materials built on Jupyter that piqued your curiosity, or have never heard of Jupyter, the material in this open book can help you empower your teaching with this new technology.

Educators newly adopting Jupyter can be overwhelmed by having to navigate the ecosystem of tools and content. They could study many examples, or consume myriad blog posts and talk videos to distill the patterns of good practices and technical solutions to best serve their students. Several early adopters, having much experience to share, decided to begin collecting this know-how, and sharing open documentation about using Jupyter for teaching and learning.

The Jupyter Community Workshop in DC (November 2018) began that process, with a book sprint aimed at producing the first version of this handbook. The collaboratively written book consolidates explanations and examples covering key topics, including: what is Jupyter, how to try Jupyter, sharing notebooks with students, locally installing Jupyter, cloud offerings, finding example notebooks, writing lessons in Jupyter, making collections for a course, exporting to other formats with nbconvert, writing textbooks with Jupyter, using Binder, JupyterHub, making assignments and auto-grading, making online courses, teaching with Jupyter in the classroom, active learning and flipped learning pedagogies with Jupyter, guiding learners to create their own content in Jupyter, and more. This open handbook will grow to encompass all you need to know about Jupyter in Teaching and Learning.

If you find these materials helpful or inspiring, give us a shout-out on Twitter using #Jupyter4Edu. We hope you do!

2.1 Acknowledgements

The book sprint was held at the George Washington University in Washington, DC, on 28–30 November 2018, and organized by Lorena A. Barba. Funding to support the logistics and travel of all participants was possible thanks to a grant from Bloomberg to Project Jupyter, and managed by NumFOCUS. The group was fêted at a reception sponsored by Leidos. Participants traveled from all over the country and volunteered their precious time and hard work to give this work to the Jupyter community, with a heartfelt sense of gratitude to all the contributors to the software projects we love and depend on. Thank you!

GitHub repository for this book: https://github.com/jupyter4edu/jupyter-edu-book

Chapter 3

Chapter 6. Getting Your Class Going with Jupyter

You have several options on how to get Jupyter Notebooks to your students. You can ask students to install Jupyter on their own computers, install Jupyter on lab computers for students to use, or run Jupyter on a remote server that your students access on the internet.

3.1 Local installation on students' or lab computers

"Local installation" means that each computer is running the software that includes the Jupyter Notebook. Typically, this requires installing a distribution that includes Jupyter, Python, and possibly other language kernels.

A popular distribution for Jupyter is Anaconda, which is easy to install on Windows, Mac, and Linux. Because it can install everything at the user level, it does not require the installer to have administrator (or root) permissions.

Two other popular open source projects that can run Jupyter notebooks are nteract and Hydrogen. nteract is installed by downloading a binary installer from their website and double-clicking the installation file. Once nteract is installed, any Jupyter notebook on a student's local system can be double-clicked and it will open within nteract. nteract's simple user interface make it an excellent choice for students new to computer programming. Hydrogen is a very popular plugin for the open source Atom editor; it's currently used by over 700,000 people. Hydrogen lets a user edit, display, and execute a notebook within the Atom editor.

You can ask students to install Jupyter on their own computer or make it possible for them to use lab computers. These can also be combined: give students the instructions to install it on their own, but also tell them that it's available in the lab if they can't get it to work on their laptop. This way you don't need a large enough computer lab for everyone, and don't need to worry that not everyone can get it to work on their own.

3.1.1 Jupyter on student-owned computers

The benefits of installation on student-owned computers include:

- Once students have the software on their computers, they always have access to it; they can work anywhere, and they can use it for internships, jobs, and other non-school activities.
- It is easy for them to install additional packages later.

- Students learn to install and set up Jupyter, and software in general, which is a skill they are likely to need.
- The total computing power for the class scales with the number of students, as long as each student has enough CPU power and memory to support the intended applications.
- You can adopt Jupyter without support or resources from your institution.
- Students learn to use Jupyter on their preferred OS, e.g. Linux, Mac, or Windows, which means they are already familiar with the basic idioms of their OS.

Drawbacks include:

- This approach is only possible if every student owns a computer with enough capacity.
- Students with less powerful computers might be at an unfair disadvantage.
- Although installation is generally easy, it still takes time. The time you spend at the beginning of a class can be worthwhile for a semester-long course that uses Jupyter throughout, but it is a barrier to using Jupyter for a single module or one-off assignment in a course about something else.
- Also, the amount of time spent debugging esoteric problems scales with the number of students: a class of 25 students is bound to have a few people with 32-bit processors, incompatible libraries, out-of-date operating systems, over-zealous virus checkers, etc., and a class with 100 students will have four times as many. One work-around is to have students work in pairs: the probability that more than half of the students cannot get it working is reduced.
- Discrepancies in installed library versions can cause issues for students and may lead to different behaviors when students run code.

Although Jupyter is cross-platform and ideally behaves the same on Windows, Mac, or Linux, and distributions such as Anaconda also behave very similarly on all platforms, the instructions for installing and launching it are slightly different on each operating system, so fine-grained instructions such as "double click here" or "type this command" need different versions for Linux, Mac, and Windows users, which can be challenging when the instructor presenting the material has only one platform at their disposal. It is worth developing detailed instructions that the students can go through at their own pace, rather than relying only on a live demo in class that will only apply to a fraction of the students.

3.1.2 Jupyter on lab computers

Using lab computers instead of student-owned computers has the benefits of uniformity and improved equity. Each student will have exactly the same setup, and the instructions will work the same for everyone. This reduces the amount of individual tech support required and guarantees that all students have access to enough computational power.

However, this deployment has some disadvantages:

- Depending on how much control you have of the computer lab, you might need institutional permission and support.

- Students might be limited to working on assignments only when they are on campus and when computer labs are open, which might be an unfair disadvantage for non-resident students or those with full time jobs.

- It might be difficult to install additional packages as the need arises, and students might not be allowed to install packages they need for projects.

- Even in a computer lab, it can be difficult to maintain consistency across machines, and to keep all installations functional.

  — **Box**

  What is Anaconda? You will see the Anaconda distribution recommended by many educators and course authors. Anaconda is a package manager, an environment manager, a Python distribution, a collection of over 1,500+ open source packages, and also Jupyter. It is free to download, open source, and easy to install, on any computer system (Windows, Mac OS X or Linux). It also includes the conda packaging utility to update and install new packages of the Python and R ecosystems, and to manage computational environments. According to the company's webpage, Anaconda has more than 6 million users. https://www.anaconda.com/what-is-anaconda/ . The Software Carpentry project provides installation instructions with videos for anaconda: https://swcarpentry.github.io/python-novice-gapminder/setup/

— **Box**

Two other popular open source projects that can run Jupyter notebooks are nteract and Hydrogen. nteract is installed by downloading a binary installer from their website and double-clicking the installation file. Once nteract is installed, any Jupyter notebook on a student's local system can be double-clicked and it will open within nteract. nteract's simple user interface make it an excellent choice for students new to computer programming. Hydrogen is a very popular plugin for the open source Atom editor; it's currently used by over 700,000 people. Hydrogen lets a user edit, display, and execute a notebook within the Atom editor.

3.2 Jupyter on remote servers

Even when Jupyter runs locally, it runs as a web application; that is, it runs in a browser connected to a server. In a local installation, the browser and the server run on the same machine. But it is also possible to run the server remotely.

In that case, students don't have to install anything; they only have to run a browser and load a URL.

There are several ways to run Jupyter on a remote server:

1. You can run Jupyter on a server owned by you or your institution.
2. You can run Jupyter in a temporary environment running in the cloud.
3. You can run Jupyter in a persistent environment running in the cloud.

Running Jupyter remotely has many of the advantages of running in a lab: you can provide a consistent environment and guarantee that all students have access to sufficient computation resources. And it mitigates one of the drawbacks of a lab installation, since students have access to cloud resources from anywhere, not just on campus.

Working in the cloud also means that students do not have to manage their own backups of a laptop hard drive. Although a student could still inadvertently overwrite, delete, or destroy the contents of a notebook stored in the cloud, they will not lose their entire work if a laptop is damaged or lost.

For simple, one-off uses of Jupyter (say, for a single assignment or in-class activity) the cloud option is very attractive as it requires little in-class time to discuss installation of additional software.

3.2.1 Running in a temporary environment in the cloud

The easiest option for running Jupyter in the cloud is to use a cloud service that provides temporary environments. Some of these services are free of cost, and you can use them without installing anything.

These environments are well-suited for short examples in classes that do not use Jupyter extensively. Students can open a notebook and start running with the push of a button.

However, there are some limitations to these services:

- If your notebooks depend on particular packages, or particular versions of packages, it can be difficult to satisfy these requirements.

- These services run notebooks in a temporary environment that disappears if it is left idle. So they might not be suitable for managing student work.

- Some of these services do not guarantee a level of service and may not be as reliable as you need for a class or workshop.

  — **Box**

  Binder (https://mybinder.org) is an open-source service provided by Project Jupyter. It allows the owner of a set of notebooks residing in a git repository to pre-build an image in the Binder service, and get a shareable link that any visitor can click to obtain a working instance of JupyterHub, with the notebooks in the repository. The session is temporary (any changes the user makes will be deleted when closing the tab or window), but it's fully interactive. Binder is currently one of the favorite services for running one-off workshops or tutorials.

3.2.2 Running on servers you control

If you have access to a server or cluster with enough computing power to support your class—including CPU and especially memory—you can provide a Jupyter as a service using JupyterHub.

JupyterHub is open-source software that provides a cloud-based Jupyter application for each user in a group. Each user has their own account and home directory on the server. The Hub, JupyterHub's central system, allows authenticating users and starting individual Jupyter notebook servers. Programs that start notebook servers can use a variety of technical solutions. For more details, see: https://github.com/jupyterhub/jupyterhub/wiki/Spawners

Once the Hub starts a user's notebook server, the Jupyter Notebook running in the cloud behaves just like Jupyter does when installed on an individual's computer, but JupyterHub will be running notebooks and storing files on a remote cloud computer. Students can download notebooks stored in the cloud to their local computer if they wish to work with a local installation as well. Additionally, students can upload notebooks (and other files) from their local computer to the cloud.

While anyone can run a JupyterHub server on their own Linux or Mac computer, installing and configuring JupyterHub requires sophisticated knowledge spanning the Linux/Unix operating system, system administration, and networking. For more information, see:

- https://github.com/jupyterhub/jupyterhub (the basic JupyterHub project, which can be installed on a bare-metal server, a virtual private server (VPS), or a commercial cloud cluster)
- https://github.com/jupyterhub/the-littlest-jupyterhub (a simplified installation of JupyterHub on a remote server or VPS)
- https://github.com/jupyterhub/zero-to-jupyterhub-k8s (a step-by-step guide to install JupyterHub on a Kubernetes cloud system)

Providing a JupyterHub service offers several benefits. First, students get up and running immediately—they spend no time installing software. They navigate to a web URL, log in to JupyterHub, and begin using Jupyter. This ability to quickly log in and begin computing is a powerful way to get students to engage with the lesson, builds confidence, and avoids the sometimes-stressful experience of installing software on the student's computer.

However, running JupyterHub on your own server has drawbacks:

- Getting started is not easy; most instructors would require (or at least benefit from) institutional support that may not be available.
- It can be difficult to scale: if the number of students increases, you might need more computing power. And the load students generate can be uneven; for example, if everyone runs a computationally-intensive example at the same time, your server might not be able to handle it.
- This option can be expensive, unless you already have servers with sufficient power.

3.2.3 Running Jupyter in the cloud

If you or your institution don't own computing hardware with the power to support your class, you can run JupyterHub on virtual servers provided by cloud services like AWS and Microsoft Azure. In those environment, you can install JupyterHub as described in the previous section

Commercial offerings also exist to use Jupyter in the cloud, some of which provide free trials or a "freemium" pricing model. They include:

- CoCalc, previously MathSageCloud (https://cocalc.com) is a subscription service with a free trial plan. The service also includes the ability to share files with project collaborators. The no-cost version does not allow network access. This has some important limitations (for example, you cannot install additional packages or kernels).
- Gryd (https://gryd.us) is another subscription service with a free tier. It includes course-management features, like a way to create a course, invite students, and deploy auto-graded assignments.
- codio (https://codio.com/features/ide)
- Microsoft Azure notebooks (https://notebooks.azure.com/)
- Amazon Sagemaker (https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-prepare.html)
- Gradient by Paperspace (https://www.paperspace.com/gradient)
- Google Colaboratory (https://colab.research.google.com/)

The biggest advantage of these services is that they require no installation and minimal setup by instructors, and some of them provide features that integrate with learning management systems. However, instructors generally have to create student accounts and set up student environments.

These services are highly scalable; that is, they can handle large numbers of students and uneven loads. However, they are not infallible; they might require some tending to make sure students have access to enough resources.

The biggest drawback of these services is that they can be expensive. Some charge on a per-student basis, with limits on computation and memory use. Some charge on the basis of actual use, which can be unpredictable (and might require instructors to enforce limits on student activities).

Other drawbacks include:

- It may be difficult or impossible to install packages you, or particular versions of packages.
- Some of these services impose limits on what students can do; for example, they might have limited ability to access external services.
- Many of these services are relatively new, and they sometimes expose instructors and students to rough edges.
- Students generally lose access to their accounts when the class ends (or a limited time after).
- There may be privacy concerns with sharing student information on commercial servers. Some institutions have agreements with one or more of these providers that address privacy.

Chapter 4

Distribution and Collection of Materials

You may want to distribute course materials to and collect them from students. A variety of options are available. Some important things to consider:

- Do you want to share your notebooks publicly, or do they require privacy?
- Can the notebooks that the students create or edit be public? Or do they require privacy?
- How do you plan to assess collected notebooks?
- Do you need integration with your LMS?
- Do you need integration with a file-sharing system?
- Do you want to distribute with the cell output showing?
- Do students need software that is not easily available on their own (or laboratory) computers?

Jupyter notebooks are plain text computer files, so you can distribute them to students and collect them using any system that handles text files, including GitHub, Google Drive, and (as a last resort) email.

4.0.1 Learning Management Systems

Many instructors use a Learning Management System (LMS) to communicate with students. These tools offer private file sharing and assignments that connect to the students' institutional computing accounts and they can be used to distribute and collect notebooks as text files. However, most LMS tools are not yet notebook-aware, so they don't render notebooks or make it easy for instructors to comment on or grade them.

Some tools and workflows are being actively developed to connect the Jupyter ecosystem to the LMS ecosystem using the Learning Tools Interoperability (LTI)(https://open.edx.org/learning-tools-interoperability) standard. By the time you read this, you might find that the options have improved.

4.0.2 Web hosting

Notebooks can be publicly hosted on any website, so students can download the files by clicking on a link. Most web hosting software is not notebook-aware,

but nbviewer is a web service that renders notebooks (https://nbviewer.jupyter.org/). Also, some browser extensions can open notebook files via nbviewer (https://github.com/jiffyclub/open-in-nbviewer).

GitHub Pages (and other similar services) can be used to host rendered notebooks, and continuous integration services can build the web pages from the notebooks and then display content. See Jupyter Book (https://github.com/choldgraf/jupyter-book) and use of doctr to do this.

4.0.3 Git

One of the most popular tools for distributing and collecting notebooks is Git, which is a version control system. Files under Git control are often hosted on services like GitHub, GitLab, and Bitbucket. Many of these services are notebook-aware; for example, when you view a notebook on GitHub, you see a rendered notebook that includes formatted text, typeset mathematics, code highlighting, and the output of the code, including figures.

Educators at academic institutions can use GitHub Classroom, which allows instructors to set up assignments for a class. Students click on a link for an assignment and a copy of the assignment repo is created and initialized with the assignment content, which can be a notebook. Each student's repository can be made private, with access only granted for the student and instructor. This can be an efficient way to distribute assignments to a large class.

A drawback of Git is that it is hard to use. It might be worth spending time in your class to teach Git if it is valuable for students to learn about version control. But if this is not one of the learning goals for your class, you can minimize the students' exposure to Git using graphical interfaces like GitHub Desktop and Git for Windows.

The default Git tools for comparing files and merging changes do not work well with Jupyter notebooks. However, there are specialized tools to help with these tasks (see Notebooks Under Version Control).

4.0.4 JupyterHub

If your students are using JupyterHub, you can place notebooks and any related files directly into the students' directories manually or via a script. If nbgrader is available on your JupyterHub instance you can use it to collect and distribute notebooks (whether or not you choose to use nbgrader's assessment features). This allows you to develop the notebooks and incrementally make them visible to the students for them to "fetch". They can then edit the notebooks or create new ones in the directory created in their storage space, and then publish their notebooks back to you for downloading, viewing, or assessing with the nbgrader tools (see the next section for details on this tool).

```
**- Box**


**What is nbgrader?** `nbgrader` is a tool for creating, handling, and automatically grading assignment


[https://nbgrader.readthedocs.io](https://nbgrader.readthedocs.io)
```

4.0.5 Using an LMS and nbgrader together:

Integration of nbgrader with learning management systems is still primitive, but the following is a strategy that works with current tools.

1. The instructor creates an assignment notebook using nbgrader, then distributes the assignment to students via an LMS.
2. Students complete the assignment and upload the solution to the LMS.
3. The instructor downloads the completed assignments as zip file and extracts the students' solutions in a Jupyter environment.

4. Instructors and graders use nbgrader to grade the assignment and save the grades to a CSV file.
5. The CSV file is then uploaded to the LMS.

There are already some tools that make this workflow easier, including the Extractor plugin to the ZipCollect feature in nbgrader (https://nbgrader.readthedocs.io/en/stable/plugins/zipcollect-plugin.html).

Chapter 5

Assessing student learning with Jupyter Notebooks

Many educators develop course-assessment activities as Jupyter Notebooks. This includes exams, in-class activities, homework assignments, and projects.

One simple way to handle the assessment of a notebook-based submission is to have students either print them out, email them, submit them as a standard electronic document (say into a Course Management System), or drop them into a shared folder. At that point, the instructor can mark and grade them in a traditional manner, for example by simply writing comments on a printout or adding annotations to a PDF.

`**_Pro Tip: Printing out a notebook can sometimes result in wasted space on pages, especially for noteb`

nbgrader allows code cells in a notebook to be marked to be auto-graded or manually graded. An instructor can then create an assignment that can be completely auo-graded, requiring little work after the notebook has been created. This makes grading much easier and scales well with large class sizes. However, creating such an auto-graded notebook in nbgrader can be quite time-consuming. In addition, pedagogically a completely auto-graded notebook may have serious downsides. For example, studies suggest that students learn better when they can actively connect a topic to their own interests [CITATION NEEDED]. One method of encouraging this is to have a "reflection" question on each submission. Such a reflection question can encourage students to comment on the material in a personal way, but it cannot be auto-graded. Another downside is that simply autograding code with unit tests is unlikely to assess many of the learning objectives you might have for an assignment, e.g., ability to use specific software-design patterns. To address this, you can create manually graded cells for a portion of an assignment and provide written feedback to the student.

GOTCHA: At the time of this writing, nbgrader has some limitations that require careful use. For example, using it in a multi-class setting (say, on JupyterHub) requires that instructors coordinate the naming of assignments so that they do not collide.

nbgrader is a sophisticated tool that can be set up to allow multiple graders, teaching assistants, and more. For more information on using nbgrader, see https://github.com/jupyter/nbgrader.

There are some third-party notebook-based assessment solutions. For example CoCalc (www.cocalc.com) and Vocareum (www.vocareum.com) provide a cloud notebook platform that can also perform assessment similar to nbgrader.

For example, cocalc.com offers... [are there other third-party course management notebook-oriented solutions?] and Berkeley uses DataHub for their large Data8 course. Vocareum (https://www.vocareum.com)

5.1

Chapter 6

How do you create Jupyter Notebooks for reuse and sharing?

As you create notebooks for your lectures, computational essays, or homework assignments, you may wish to think about how to make it possible that they can be reused by yourself and others.

First, you may want to make the materials openly accessible and findable via the internet. This suggests avoiding keeping the notebooks behind a “walled garden,” such as a Course Management System. That is, users may have access to some material, but be prevented from seeing other materials. You will have to decide whether you want others to have full access. For example, many teachers do not want students to be able to see notebooks that may have solutions, or hints of solutions and therefore limit their access.

Callout box: To share your notebook with others you can submit it to https://www.engage-csedu.org/ This curated collection of open educational computing resources is maintained by the National Center for Women in Information Technology (NCWIT).

If you decide to make your notebooks reusable by others, make it clear under which license the materials can be used. For example, you can include a Creative Commons attribution and share-alike statement at the bottom of your notebook. Adding a license allows people to reuse your materials without asking for permission explicitly. https://creativecommons.org/licenses/by-sa/3.0/us/

GitHub may be the most common service to host and share notebooks, where they can be viewed (including rendering), downloaded, or forked by others. (Private repositories can also be used to limit visibility to colleagues, students, or other organizations.). Make sure to be aware of some of the pitfalls of keeping notebooks version controlled however (see Notebooks Under Version Control for details).

Another potential issue with sharing deals with external files that you may want to include in your notebook. This is in contrast with content (say a plot) that can be directly created by the notebook’s code. Possible content includes data, images and videos using code and embedding tags in markdown or HTML. The implication is that if you share your notebook you must include the external files along with the notebook. This can be done a number of ways including using a version control repository, a zip-file, or a file sharing service. Another external dependency issue with sharing notebooks involves software libraries. In this case you share a configuration file that a user can use to setup the same environment. Examples of these files include a conda env.yml, a pip requirements.txt, or dockerfile.

Because Jupyter notebooks embed the output of cells into the ipynb file itself (e.g., images, videos, etc.), the files can grow large. To make it possible to display the cell output via the renderers on Github, Gitlab, or nbviewer, save the notebook after it is executed and then upload to those services. If instead you want to reduce file size and provide the notebooks to someone with the code cell output cleared, choose this option in the notebook’s dropdown interface. Then the user will need to execute the notebook themselves to see the output.

Jupyter: a 21st Century Genre of Open Educational Resources and Practices

Educators create teaching and learning materials. With the appearance of the internet, a community of educators began producing open access traditional teaching materials. In parallel, a community of software developers began creating open source software. Each community developed their own development patterns. In particular open source software communities gravitated to the bazaar style[1] of distributed and collaborative work. Jupyter notebooks may be the first time that these communities are merging. Jupyter notebook authors are applying the content creation patterns they use to the creation of open educational resources that teach computation or teach through computation.

Open Education encompasses a large community, with its own conferences and journals, with leaders and advocated practices. The most visible efforts are related to Open Educational Resources (OER): the creation and adoption of openly licensed learning materials. In 1994, Wayne Hodgins coined the term "learning object" and the idea spread that digital materials could be designed and made to be *reused*. This was followed by efforts to develop metadata standards, content exchanges, and so on (addressing the concern of how to find the objects to reuse them). In 1998, David Wiley coined the term "open content" and spread the idea that principles of Free and Open Source Software (FOSS) could be applied to content on the World Wide Web. The Creative Commons non-profit organization was founded in 2001 to provide ready-made license agreements for sharing content and served a vital infrastructure role on the spread of OER. The Creative Commons licenses are now the most widely used licensing framework for open education. The year 2001 also saw the launch of MIT OpenCourseWare (OCW). MIT promised free public access for non-commercial uses of their course materials. It was a unique commitment at an institutional level, strengthened by the MIT brand. Other universities joined the OCW movement: Rice with the OpenStax project (now formerly Connexion), CMU with the Open Learning Initiative (OLI), Utah State University with the Center for Open and Sustainable Learning, and so on. Today, the Open Education Consortium has hundreds of members from around the world. The recurrent topics in OER are: reducing costs for students buying textbooks, increasing access, and dealing with copyright and licenses.

In the last few years, educators using Jupyter have been creating and sharing all kinds of educational materials in the form of notebooks, typically under a Creative Commons Attribution license (CC-BY). In fact, Jupyter is *a new genre of OER*. But in addition to creating open content, educators using Jupyter often take active part in the Jupyter community and adopt the *culture* of open-source software. This is a culture with strong ethical commitments, related to freedom of access, transparency, and governance (Coleman, 2012). The content they create has the value of giving access (the very definition of OER), under an open model. But open-source culture also promotes a culture of collaboration. In this regard, engaging in teaching with Jupyter opens new possibilities for educators to engage in *open development* and collaborate with others in producing lessons, tutorials, courses, and even books.

REF—Coleman, E.G., 2012. Coding freedom: The ethics and aesthetics of hacking. Princeton University Press.

NOTE from Carol: Installation on a student computer or web-based access. Installation on computer lab computers or web-based access. Web-based access requires additional decisions and system administration resources.

6.1 Notes

[1]The bazaar style is a method of collectively creating software that isn't top down directed like a traditional company hierarchy.

Chapter 7

About the Authors

7.1 Project Lead

7.1.1 Lorena A. Barba

- George Washington University
- labarba@email.gwu.edu
- @LorenaABarba

Lorena A. Barba is Associate Professor of Mechanical and Aerospace Engineering at the George Washington University. She adopted Jupyter in 2013 and since then used it in every course she teaches. Her open course materials are well known and used by thousands of learners: CFD Python and Numerical MOOC are the best examples.

7.2 Authors at the sprint

7.2.1 Lecia J. Barker

- University of Colorado Boulder
- lecia.barker@colorado.edu
- @leciab

Lecia Barker is an Associate Professor and Associate Chair of Undergraduate Studies in the Department of Information Science at the University of Colorado Boulder. She is also a Senior Research Scientist for the National Center for Women & IT. Her research group is studying the diffusion and adoption of teaching practices in undergraduate computer science. Lecia holds a Ph.D. in Communication from CU Boulder and an MBA in Marketing from San Diego State University.

7.2.2 Douglas Blank

- Bryn Mawr College
- dblank@brynmawr.edu
- @dougblank

Douglas Blank is Associate Professor in the Department of Computer Science at Bryn Mawr College, a small, all-women's college outside of Philadelphia, PA, USA. He has a joint Ph.D. in Cognitive Science and

Computer Science from Indiana University, Bloomington. For over 20 years, Douglas has taught all levels of Computer Science. For the last 4 years, he has used Jupyter notebooks exclusively in the classroom. Douglas has published in the areas of Computer Science Education, Robotics, Artificial Intelligence, and Deep Learning. He is on the advisory board of Engage-CSEdu.org, a joint project between Google and the National Center for Women and Information Technology (NCWIT). Douglas also writes text and code at his website douglasblank.com.

7.2.3 Jed Brown

- University of Colorado Boulder
- jed@jedbrown.org
- @five9a2

Jed Brown is an Assistant Professor of Computer Science at the University of Colorado Boulder. He has been teaching numerical and scientific computing courses using Jupyter Notebook and nbgrader for three years, and leads a research group that develops computational methods and community software for computational science.

7.2.4 Allen Downey

- Olin College
- downey@allendowney.com
- @AllenDowney

Allen Downey is a professor of Computer Science at Olin College and the author of a series of open-source textbooks related to software and data science, including *Think Python*, *Think Bayes*, and *Think Complexity*, published by O'Reilly Media. These books, and the classes based on them, use Jupyter notebooks extensively. Prof Downey holds a Ph.D. in computer science from U.C. Berkeley, and M.S. and B.S. degrees from MIT.

7.2.5 Tim George

- Project Jupyter
- tgeorgeux@gmail.com

Timothy George is the Lead UI/UX Designer for Project Jupyter, focusing primarily on JupyterLab. In addition to his formal duties, Tim is also in working with Jupyter on design strategy, future products, governance, diversity and inclusion. He studied HCI at UC Irvine's Donald Bren School of Informatics and Computer Science where he received a Master's Degree.

7.2.6 Lindsey Heagy

- University of California Berkeley
- lindseyheagy@gmail.com
- @lindsey_jh

Lindsey Heagy is a Postdoctoral Researcher at the University of California Berkeley working on Project Jupyter and Jupyter in the geosciences. She recently completed her PhD at the University of British Columbia in geophysics. She is a project leader of GeoSci.xyz, an effort to build collaborative, interactive, web-based textbooks in the geosciences, and a core contributor to SimPEG, an open source framework for geophysical simulation and inversions. The GeoSci.xyz project relies heavily on Jupyter for making the content come to life.

7.2.7 Kyle Mandli

- Columbia University
- kyle.mandli@columbia.edu
- @KyleMandli

Kyle Mandli is an Assistant Professor in the Department of Applied Physics and Applied Mathematics at Columbia University. He has developed a set of openly available course notes centered around Jupyter notebooks and uses Jupyter for homework in conjunction with nbgrader. His other research interests include development of computational methods for coastal hazards such as storm surge and tsunamis.

7.2.8 Jason K. Moore

- University of California, Davis
- jkm@ucdavis.edu
- @moorepants

Jason K. Moore is an Assistant Teaching Professor of Mechanical and Aerospace Engineering at the University of California, Davis. He currently teaches dynamics and mechanical design related courses. He utilizes Jupyter notebooks to teach modeling and simulation and is working on a textbook about Mechanical Vibrations. He is responsible for the Jupyter related features in the LibreTexts project and is also a core developer of the SymPy and PyDy projects which utilizes Jupyter for training workshops, e.g. PyDy Tutorial and SymPy Code Generation Tutorial. Jason has PhD, MSc, and BSc degrees in mechanical engineering from UC Davis and Old Dominion University.

7.2.9 David Lippert

- Leidos

David Lippert is a software engineer at Leidos in Arlington, Virginia. He utilizes Jupyter notebooks primarily for exploratory data analysis and for training and evaluating machine learning algorithms. He has written Jupyter notebooks to create new Dr. Seuss sonnets and to evaluate if the Rotten Tomatoes Tomatometer can be trusted. He has a BA in computer science from Middlebury College.

7.2.10 Kyle Niemeyer

- Oregon State University
- kyle.niemeyer@oregonstate.edu
- @kyleniemeyer

Kyle Niemeyer is an Assistant Professor of Mechanical Engineering in the School of Mechanical, Industrial, and Manufacturing Engineering at Oregon State University. He teaches courses in numerical and analytical methods for solving differential equations as well as gas dynamics, and recently developed a graduate course on software development for engineering research. His research group develops and applies methods for modeling combustion and chemically reacting fluid flows. He is also on the steering committee of the Cantera open-source project for chemical kinetics, thermodynamics, and transport processes.

7.2.11 Ryan Watkins

- George Washington University
- rwatkins@gwu.edu
- @parsingscience

Ryan Watkins is a Professor of Educational Technology at George Washington University in Washington DC. He leads the Human-Technology Collaboration (HTC) PhD program area, and he teaches courses in needs assessment, instructional design, and research methods. Ryan's research focuses on how people and organizations define and assess needs. He is co-host of Parsing Science, a podcast where researchers share the stories behind their science. He also developed the We Share Science platform for sharing video abstracts of research.

7.2.12 Richard West

- Northeastern University
- R.West@northeastern.edu
- @richardhwest

Richard West is Associate Professor of Chemical Engineering at Northeastern University in Boston. He leads a research group in computational modeling for complex reacting systems like combustion or catalysis. He is a core member of the Cantera open-source project. As well as in an elective on "computational modeling in chemical engineering", he has integrated Python and Jupyter into core classes on chemical kinetics and reactor design, at both the undergraduate and graduate levels. As part of his NSF CAREER award, he is developing modules to teach students to use Python and SciPy to solve chemical engineering problems.

7.2.13 Elizabeth Wickes

- University of Illinois at Urbana-Champaign
- wickes1@illinois.edu
- @elliewix

Elizabeth Wickes is a Lecturer at the School of Information Sciences at the University of Illinois at Urbana-Champaign. She teaches foundational programming from an information and data sciences perspective, as well as other coursework on open data and reproducibility. Her programming course lectures are written in Jupyter Notebooks and the class is taught via live coding.

7.2.14 Carol Willing

- Cal Poly San Luis Obispo
- willingc@gmail.com
- @WillingCarol

Carol Willing is a Research Software Engineer at Cal Poly San Luis Obispo working full-time on Project Jupyter. She is a Python Software Foundation Fellow and former Director; a Project Jupyter Steering Council member; and a core developer on CPython and Jupyter. Carol has an M.S. in Management from MIT and a B.S.E. in Electrical Engineering from Duke.

7.2.15 Michael Zingale

- Stony Brook University
- Michael.Zingale@stonybrook.edu
- @Michael_Zingale

Michael Zingale is an Associate Professor and computational astrophysicist at Stony Brook University. He has a PhD from University of Chicago (2000). He frequently teaches numerical methods and Python for scientific computing graduate courses, relying on Jupyter notebooks and python for much of the presentation. He is an advocate for open educational resources, as a founder of the Open Astrophysics Bookshelf project where he hosts his *Introduction to Computational Astrophysical Hydrodynamics* text.

Chapter 8

Glossary

Anaconda: a free, open-source package manager, environment manager, Python distribution, and collection of over 1,500+ open source packages including and also Jupyter. https://www.anaconda.com/what-is-anaconda/

API (Application Programming Interface): the exact details of interacting with software, usually used by other software.

Binder: a hosted service that allows anyone to launch their own sandboxed notebook environment from a Git repository. https://mybinder.org

cell: the area in a Jupyter Notebook where you can enter markdown, or computer code.

cloud, in the: used to describe software or documents hosted on a remote computer accessed over the internet.

CSV (Comma Separated Values): referring to a comma-separated value file. A plain-text file format such that each line is a list of data separated by commas.

execute: technical term for having the computer perform the instructions of your program. Alias for "run it."

extension, Jupyter: in this instance, it is not a request for more time. Rather, a Jupyter extension is a bit of code, often developed by a third-party, that adds additional functionality to Jupyter. For example, a popular extension is a Table of Contents creator.

flipped classroom: a teaching style where students work on their own outside of class to learn new material (sometimes by watching recorded lectures or reading descriptive/interactive notebooks) and the come together in the classroom to practice what they've learned through exercises or experiments.

Git: a popular version control system (VCS) used for keeping track of changes of files over time.

IDE (Integrated Development Environment): software that assists in the development of additional software.

Jupyter: The term "Jupyter" may refer to one of a couple of different things: a community of users and developers focused on the open source software; the collection of tools and standards that, together, allow projects like the Jupyter Notebook to operate. The name refers to the three core programming languages supported: Julia, Python, and R.

JupyterHub: a cloud service that can provide access to Jupyter notebooks and environments to multiple users via a modern web browser. http://jupyter.org/hub

kernel: In Jupyter, a kernel is the packaging up of a language, and related programs needed to run it. For example, Python2 and Python3 are separate kernels.

LMS (Learning Management System): a cloud service that helps instructors manage aspects of classrooms.

load: how many students can a computer support?

Markdown: a text format that allows for basic formatting (headers, text styles, links) mixed inline with the text. Markdown files usually have the extension `.md` and can be rendered natively by GitHub and other tools.

magic: a meta-command typically starting with one or two percent signs. Changes the meaning of the contents of a line (one percent sign, `%`) or the cell (two percent signs, `%%`) from code to a particular meta-instruction. For example, `%%R` indicates that the cell contents will be interpreted as commands to the R language. Magics are kernel-specific (e.g., vary with the kernel in use).

nbgrader: a tool for creating, handling, and automatically grading assignments based on Jupyter notebooks. https://nbgrader.readthedocs.io

nbviewer: a web application for rendering Jupyter notebooks as static web pages, providing a URL to share and view them with a modern web browser. https://nbviewer.jupyter.org

nbconvert: a tool for converting Jupyter notebooks into other formats such as PDF, HTML, LaTeX, Markdown, reStructuredText, and others. https://nbconvert.readthedocs.io

notebook hidden state: a technical term referring to the value of variables that may have surprising results due to cells having been executed in a non-sequential order.

open source: software and documents that are created in a manner that give you rights to be able to use, and reproduce.

pattern: A "pattern" is a technical term referring to an abstract description of a labeled process. For example, "wash, rinse, repeat" is a common pattern for cleaning various objects.

service, JupyterHub: JupyterHub can take advantage of additional separate, but integrated, software extensions. These are called "services."

script: a colloquial term for a computer program.

unit test: a technical term for a "test" for checking to see if software is operating correctly.

URL (Universal Resource Locator): the address of a resource (e.g., webpage) on the internet.

widget: a user interface (such as buttons, sliders, and checkboxes) that allow the easy control of hidden computer code.