PSC NCSA NSF Award 2417789

Carnegie Mellon University

University of Pittsburgh

CI Pathways

Leading the Way to Effective Cyberinfrastructure Use

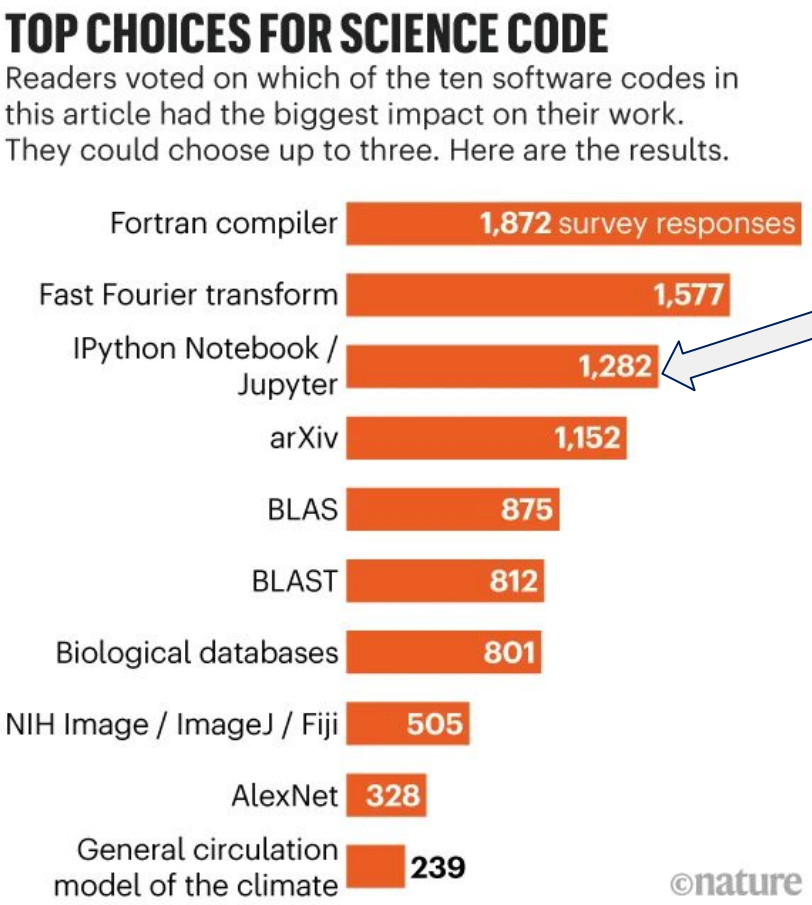
CI Awareness - Session 2: Interactive HPC Jupyter Notebooks

Soham Pal - Research Computing & Data Facilitator, NCSA

Overview

1. What is Jupyter?
2. How to access Jupyter?
3. Basic Jupyter concepts - architecture, kernels, cells
4. Jupyter + Python examples
5. Suggestions

What is Jupyter?



NEWS FEATURE | 20 January 2021 | Correction 22 January 2021 | Update 19 February 2021 | Correction 08 April 2021

Ten computer codes that transformed science

From Fortran to arXiv.org, these advances in programming and platforms sent biology, climate science and physics into warp speed.

Is it really that good?

What really is Jupyter?

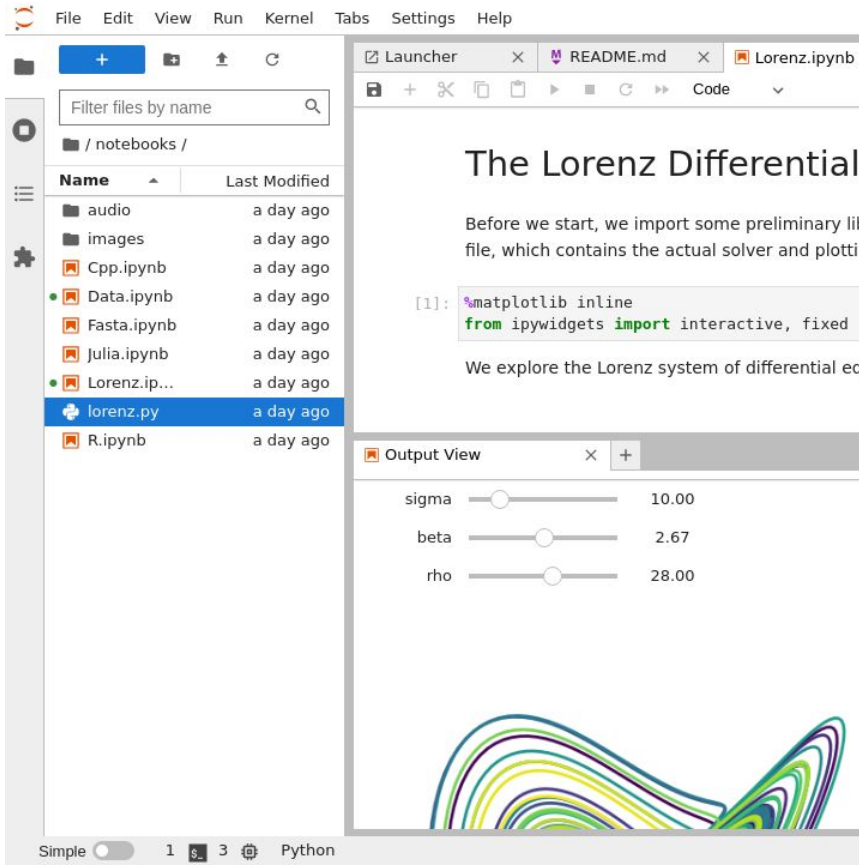
Jupyter is an ecosystem of tools and protocols for interactive coding with computational notebooks.

What is a computational notebook?

Just like your usual paper notebooks, but now in addition to text and math you have

- code
- visualization
- etc

This is more than comments in your programs. With a notebook you can explain complex information and ideas along with the necessary code.



More information: [What is Jupyter?]

Brief history of the Computational Notebook

1984 — Literate programming

Don Knuth introduced the idea of literate programming - code snippets embedded in natural language text.

1988 — Mathematica

Mathematica was the first scientific computation tool to introduce the notebook interface. Still going strong in some fields.

1992 — Maple

Maple introduced its "worksheet" interface which integrated text, code, and graphics into one document.

2001 — IPython

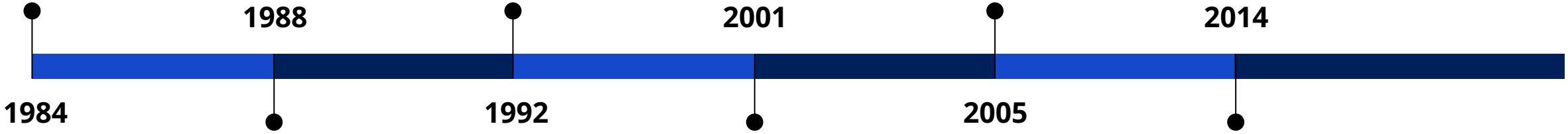
An open source project that provides an interactive environment for programming in Python. Added notebook interface in 2011.

2005 — SageMath

An open source project that leverages various Python and R libraries to provide an interactive computational interface.

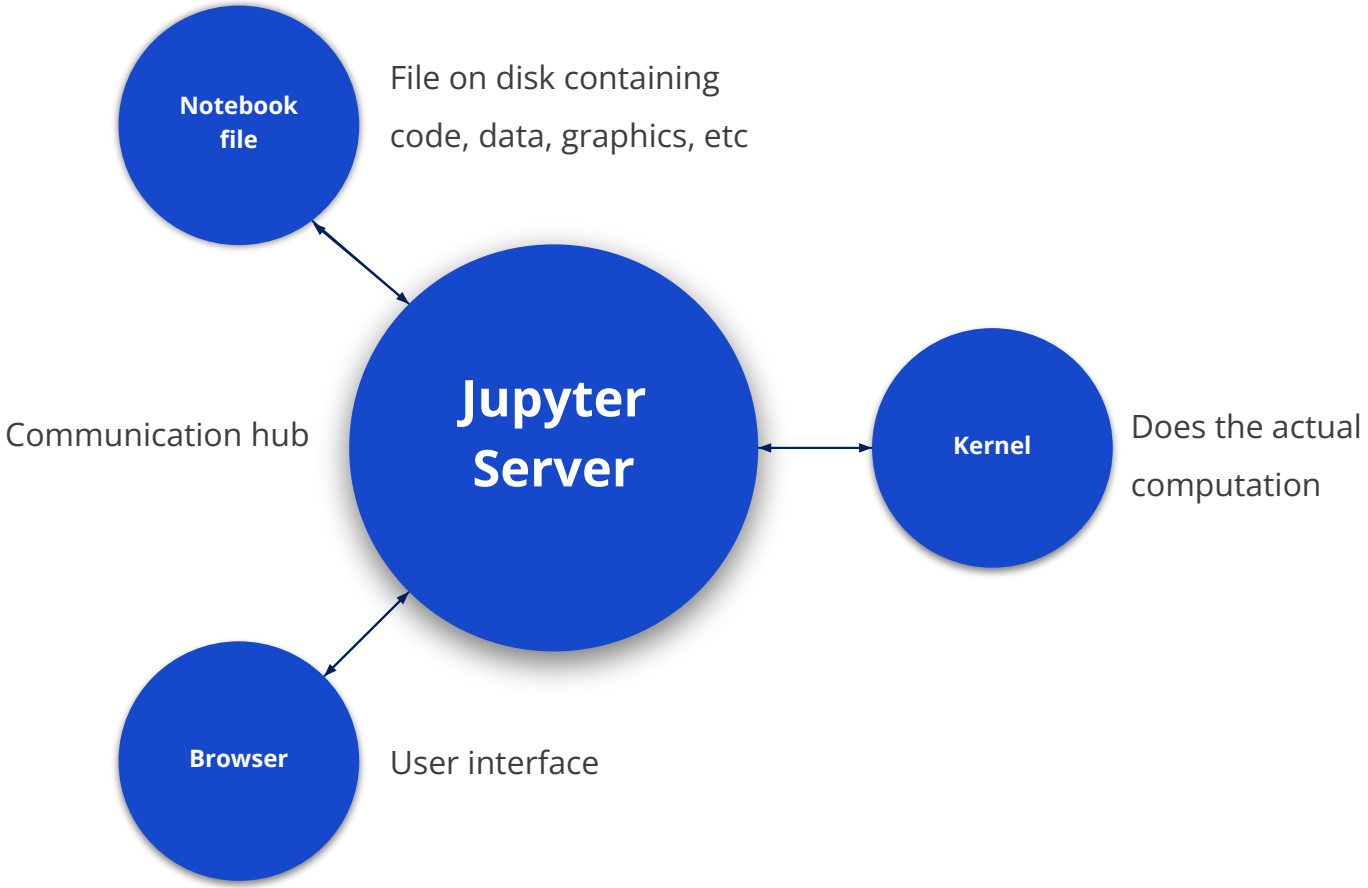
2014 — Project Jupyter

Started as an evolution of IPython notebooks to support interactive programming in **Ju**lia, **Py**thon, and **R**. Now supports hundreds of languages.



More information: [A Brief History of Jupyter Notebooks]

How does Jupyter work



More information: [Jupyter architecture]

- User only directly interacts with the user interface (UI).
- All communication is mediated through the Jupyter Server.
- The Jupyter Server, the kernel, and the notebook file do not need to be on the local machine - **they can be on a separate server**.
- The default kernel is the IPython kernel. For a list of supported kernels, see [Jupyter kernels].

Jupyter interfaces

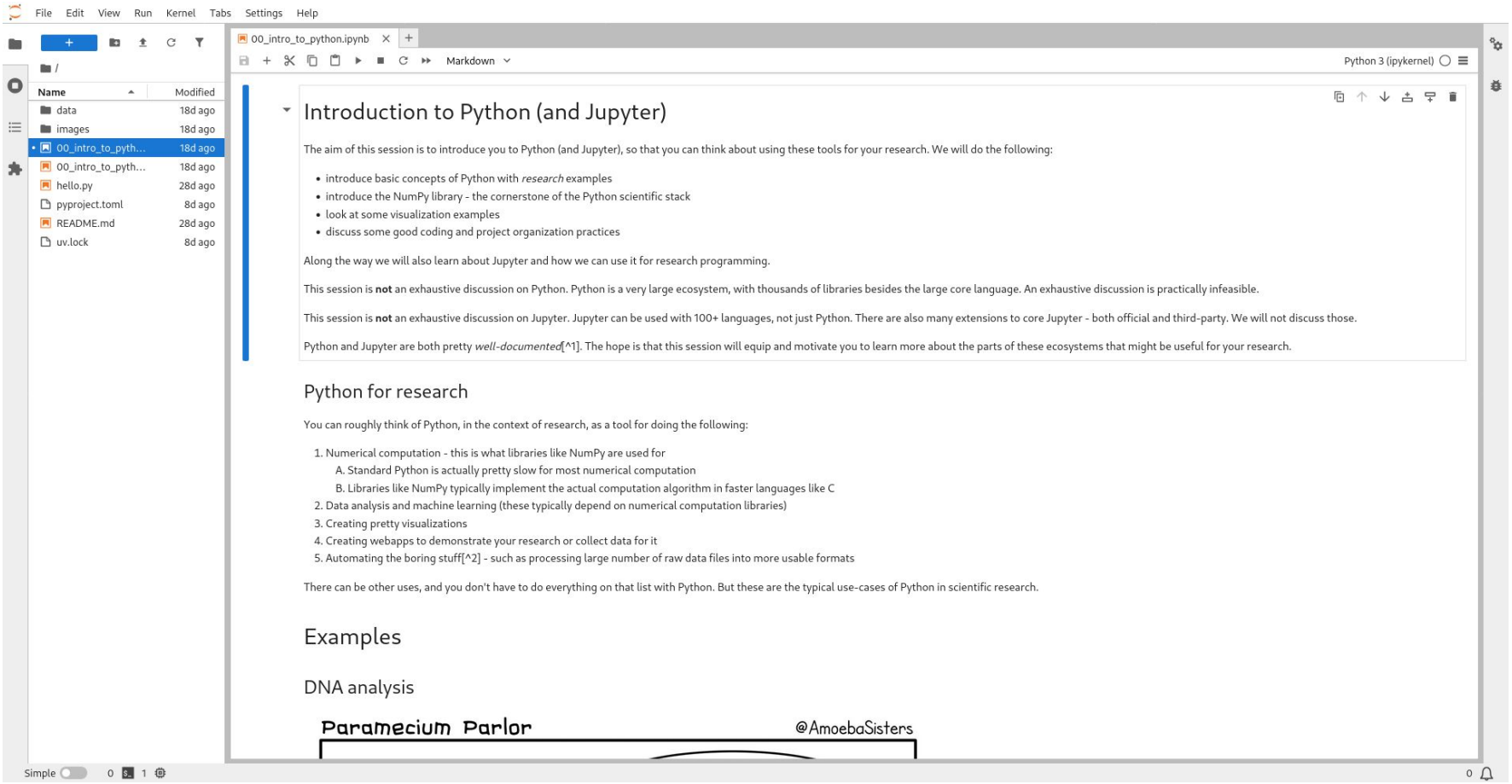
Project Jupyter provides multiple user interfaces, most of them browser based. Additionally, text editors/IDEs like VS Code, Emacs, Spyder, etc provide alternative UIs.

The two main interfaces are:

1. Jupyter Notebook: This is the original Jupyter interface. Often "Jupyter" is used to mean this interface.
2. **JupyterLab**: A modern version of the notebook interface with more IDE like capabilities.

(The word "*notebook*" is overloaded in the Jupyter world. Depending on the context it can mean either the interface, the file on disk, or both.)

JupyterLab interface



Jupyter use cases

Jupyter can be useful in the following or similar cases:

- communicating scientific ideas
- iterating ideas or implementations
- collecting, visualizing, analyzing data
- machine learning/deep learning workflows
- developing interactive course materials

Jupyter is not necessarily the best tool for writing large software libraries.

You might want to consider a text editor or IDE instead.

How to access Jupyter

- Try Jupyter: Just to get a feel of the interface, not for serious computation.
- Install locally.
- Cloud providers: **SageMaker Studio Lab**, Lightning Studio, Google Colab, etc. These are often modified to provide “extra” features.
- **HPC** offerings: For example, Delta provides JupyterLab through Open OnDemand.

We will look at SageMaker Studio Lab, and the “vanilla” JupyterLab provided by Delta.

SageMaker Studio Lab

SageMaker Studio Lab is an Amazon-hosted JupyterLab with access to GPUs. It is free to use.

You get:

- 4 hours of CPU/GPU per session
- Total of 8 hours of CPU/GPU in a 24 hour period

Useful for learning, experimentation, prototyping, particularly in the fields of data science, machine learning, scientific computation.

To access SageMaker Studio Lab, go to https://studiolab.sagemaker.aws/ and request an account. Usually approved pretty fast.

Google Colab

Google Colab is a Google-hosted (modified) Jupyter Notebook with access to GPUs and TPUs. It offers both free and paid tiers.

In the free tier, notebooks can run for 12 hours. The types and usage limits of GPUs and TPUs depend on usage pattern and availability.

Useful for learning, experimentation, prototyping, particularly in the fields of data science, machine learning, scientific computation. Can be potentially used for bigger problems than SageMaker Studio Lab.

All you need to access Google Colab is a Google account.

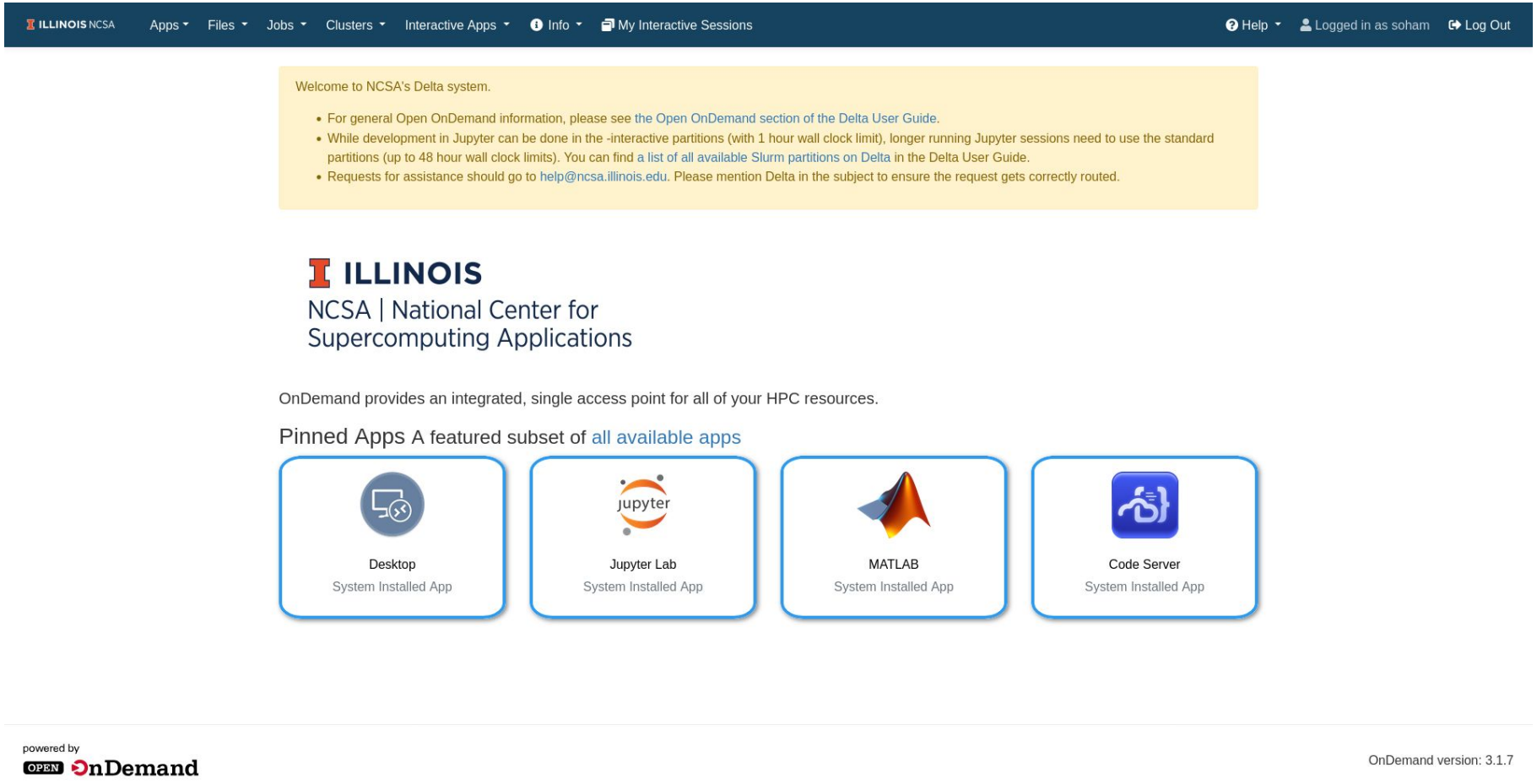
Accessing JupyterLab on Delta

Delta offers a JupyterLab interface with access to Delta's CPUs and GPUs. To use JupyterLab on Delta, you need to have a NCSA identity and a allocation on Delta.

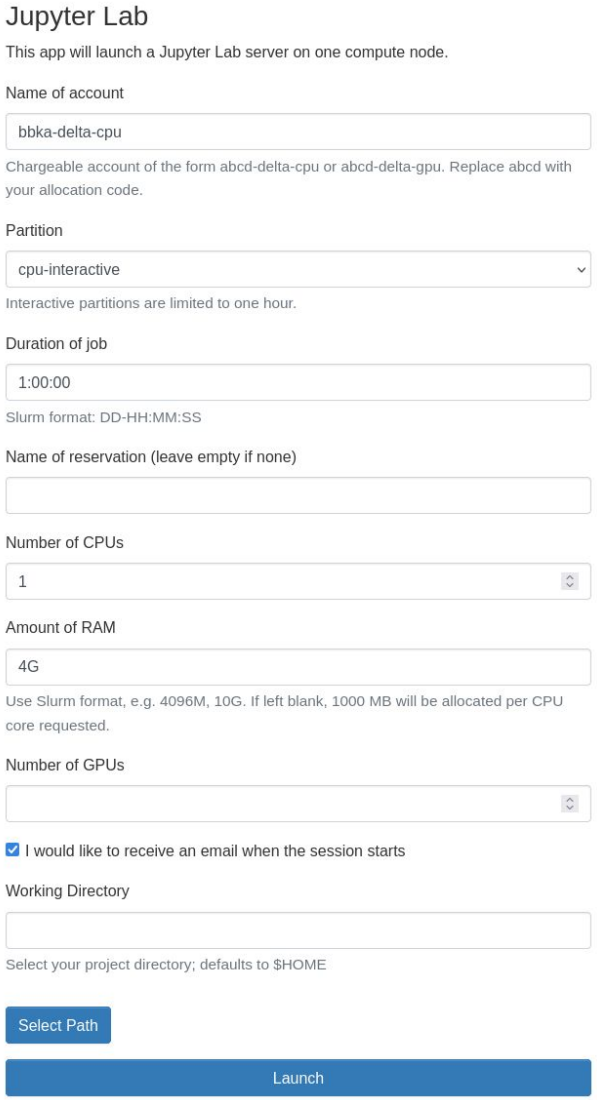
1. Navigate to openondemand.delta.ncsa.illinois.edu
2. Login through CILogon with your **NCSA identity** (will need Duo authentication)
3. Choose **Jupyter Lab** from the Open OnDemand dashboard
4. Fill in the form (specify account, # CPUs, # GPUs, etc) and select **Launch**
5. Once your session is ready, select **Connect to Jupyter**

More information: Delta documentation

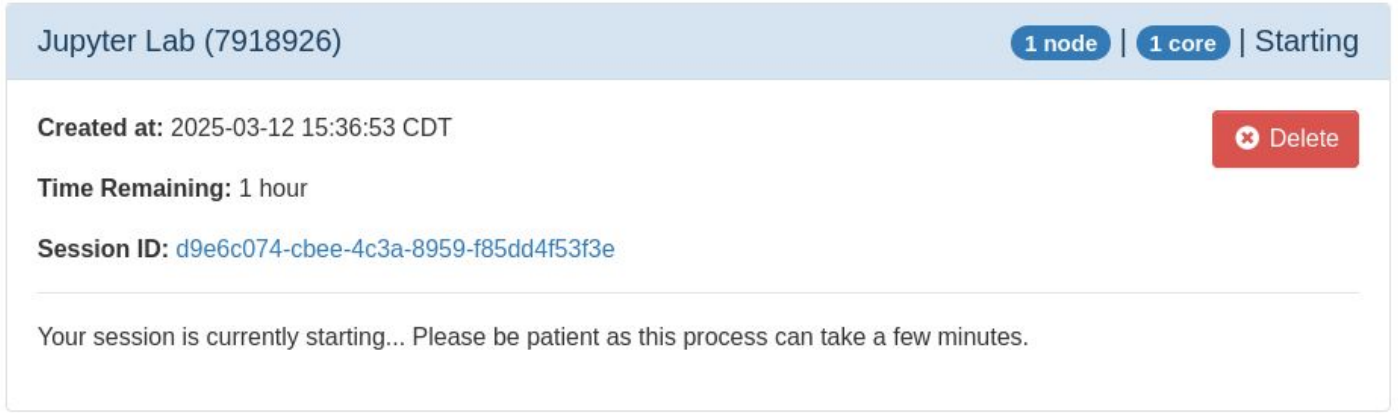
Delta OOD dashboard

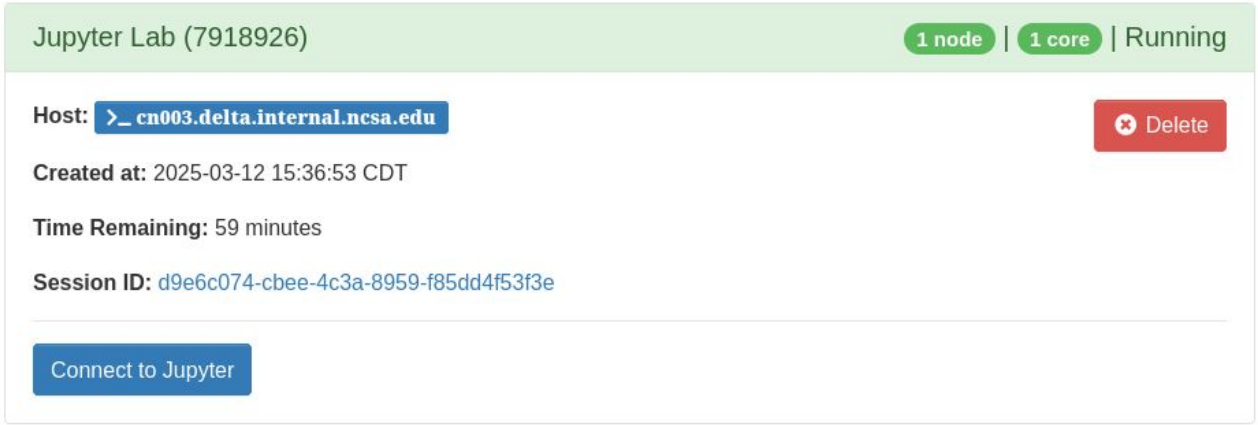


Delta OOD request form



Launching Jupyter on Delta





Installing Jupyter kernels

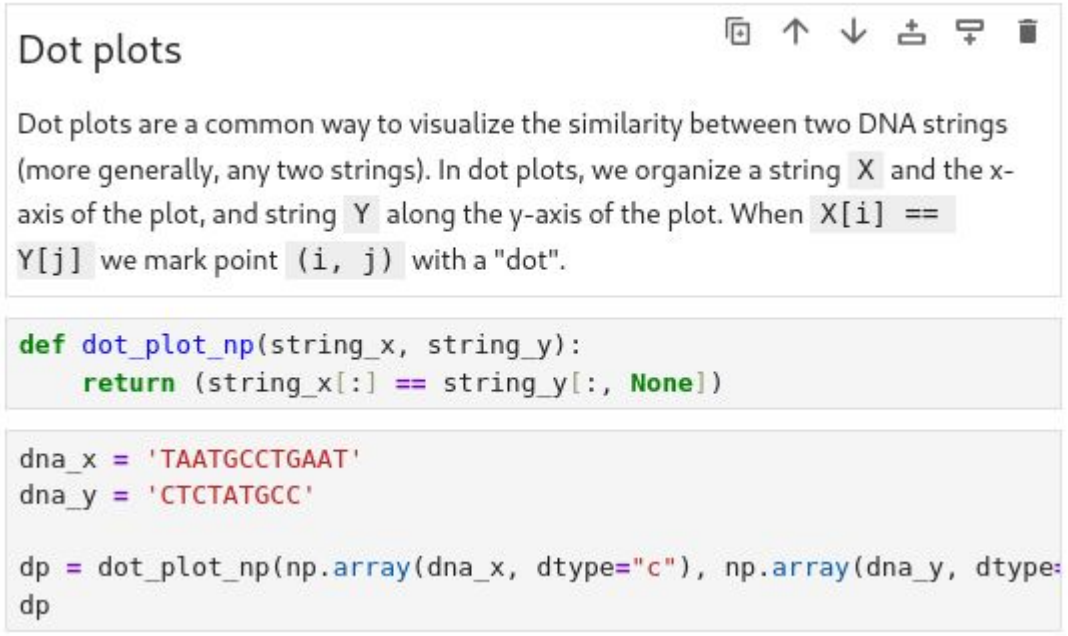
We will only look at Python kernels. The process will be mostly similar for R or Julia. Python has the ***curse of package and environment managers***. To keep things uniform we will use conda.

1. First create a conda environment: `conda create -n <env-name>`
2. Activate conda environment: `conda activate <env-name>`
3. Install packages (including Jupyter): `conda install <package>` or `pip install <package>`
4. **Install IPython kernel**: `ipython kernel install --user --name <kernel-name>`

On Delta, you will have to initialize “conda” first. More information: Delta documentation

Cells in Jupyter notebooks

Content in Jupyter notebooks are organized in cells.



Dot plots

Dot plots are a common way to visualize the similarity between two DNA strings (more generally, any two strings). In dot plots, we organize a string `X` and the x-axis of the plot, and string `Y` along the y-axis of the plot. When `X[i] == Y[j]` we mark point `(i, j)` with a "dot".

```
def dot_plot_np(string_x, string_y):
    return (string_x[:] == string_y[:, None])
```

```
dna_x = 'TAATGCCTGAAT'
dna_y = 'CTCTATGCC'

dp = dot_plot_np(np.array(dna_x, dtype="c"), np.array(dna_y, dtype=
dp
```

Text cell

Jupyter uses Markdown for narrative text. You can even add mathematical content with LaTeX syntax.

Code cells

You can only run code that is in a code cell.

Demonstration

Things to be careful about

Code execution and state management

Out-of-order execution

You can run cells in any order, which can lead to a misleading or inconsistent program state — a variable defined in cell 10 can affect cell 3 if you run it after.

Hidden state

If you modify data or imports in a cell and rerun other cells, the notebook might rely on "hidden" state that isn't visible. Restarting the kernel clears this, but it’s easy to forget.

File and environment management

Path confusion

Working directories can get messy. If you move a notebook file, relative paths to data or modules might break.

Dependency drift

Reproducing a notebook months later might fail if package versions changed.

- This is less likely to happen if you use pre-built modules, like the “anaconda” modules on Delta.
- You can track environments with tools like “pip freeze” or “conda env export”, and ensure that your notebook is reproducible.

Version control

Hard to diff and merge

Notebooks are saved as JSON, making them hard to version-control properly with version control tools like git.

Collaboration friction

Real-time collaborative editing (like Google Docs) isn't natively supported on the Jupyter Notebook interface. JupyterLab provides an extension which can help with this, but might not work with all JupyterLab providers.

Third-party tools like Jupytext and nbdime can help with alleviating some of these difficulties. If you are used to working with scripts, then Jupytext is highly recommended.

Suggestions/Tips

Code organization

Use Functions & Classes

- Avoid writing long, linear notebooks
- Wrap code into reusable functions (or classes, if needed)
- Create libraries, if needed, and import them into notebooks

Split large projects

- Instead of one large notebook, have multiple smaller ones
- Run separate notebooks like modules using the “%run” magic

```
%run data_preprocessing.ipynb
```

Profiling and debugging

Profile your code to identify bottlenecks

Profiling a single line of code

```
%timeit sum([i**2 for i in
range(10**6)])
```

Profiling a code cell

```
%%timeit
a = np.arange(10**6)
np.sum(a**2)
```

Use the JupyterLab debugger or the %%debug magic to identify bugs in your code.

Interactive plots

Use modern packages (Python)

- Plotly or Bokeh
- Both are highly recommended, but Bokeh is personal favorite
- Both cases you will have to install the necessary libraries

If using Matplotlib

- Use `%matplotlib ipympl` magic
- Newer versions prefer ipympl
- You will likely have to install ipyml