Jupyter with OnDemand

Introduction: This workshop will discuss how to use Jupyter Notebooks and Jupyter Labs on ARCC HPC clusters and introduce a series of best practices.

Course Goals:

- Introduce what Jupyter is and why it’s useful
- Identify the difference between Jupyter Lab and Jupyter Notebooks and when to use one tool over another
- Demonstrate the Jupyter service within OnDemand across a variety of available languages and kernels
- Provide the steps to convert an existing Conda environment into a kernel that can be used within a Jupyter session

Notes:

- The workshop modules work best in a sequential manner as a story introducing concepts and providing examples, but sections can be used separately to focus on a particular concept.
- You will need to *modify* usernames, project names, and folder locations, to apply to yourself.

Jupyter

Goals:

- Introduce what Jupyter is and why it's useful
- Differentiate between Jupyter Notebooks and Jupyter Labs and when to use each
- Identify cell types in a notebook and how they're used
- Things to be

What is Jupyter?

Jupyter, formerly known as an ipython notebook, is a popular tool used in data science and data analysis.

- An open-source, browser-based, web application with a wide variety of functions
    - Allows users to create and share computational documents, called notebooks
        - Notebooks facilitate the development of live code that can then be run in a number of different coding languages.
        - Code can be run step by step in "chunks" called cells.
        - Users can combine live code cells with other cells - Markdown text, images, plots, and other rich media in a single interactive canvas.
        - Can produce a wide variety of interactive output including HTML, videos, LaTeX, and custom MIME types.
        - Can be shared through e-mail, GitHub, or other cloud storage and sharing services.
        - Easily exported to other formats like, books, slides, web apps, static web pages, or PDF documents.

- Allows users to run code and share ideas and share in a “live” and easily available format.

- Requires a kernel to launch

Jupyter Notebooks

<table>
<tr>
<td>

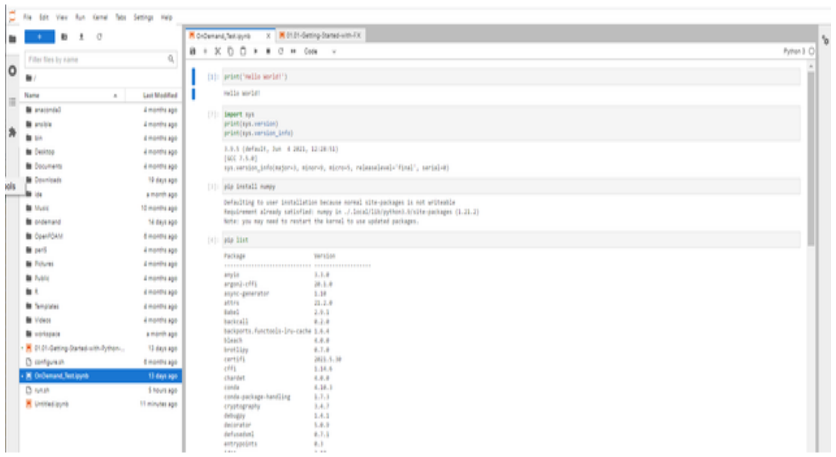
Jupyter Notebooks are just the Notebook application itself

- **Simple interface where users can open and run notebooks**
- **Straightforward linear flow, where you can create and run cells in a single notebook**
- **Beginners may find Notebooks easier to use, initially**

Lacks some of the functionality of Jupyter Lab

- **Supports extensions, but the process of installing and managing them more difficult.**
- **Available extensions is smaller compared to JupyterLab.**
- **Does not have a built-in terminal or text editor. Users need to rely on external tools or extensions for these tasks.**

</td>
<td>



File Edit View Run Kernel Tabs Settings Help
OnDemand_Test.ipynb
01.01-Getting-Started-with-FX
Code
Python 3
Name
Last Modified

```
[1]: print('Hello world!')

     Hello world!

[2]: import sys
     print(sys.version)
     print(sys.version_info)

     3.9.5 (default, Jun  4 2021, 12:28:51)
     [GCC 7.5.0]
     sys.version_info(major=3, minor=9, micro=5, releaselevel='final', serial=0)

[3]: pip install numpy

     Defaulting to user installation because normal site-packages is not writeable
     Requirement already satisfied: numpy in ./.local/lib/python3.9/site-packages (1.21.2)
     Note: you may need to restart the kernel to use updated packages.

[4]: pip list

     Package                       Version
     ----------------------------- -------------------
     anyio                         3.3.0
     argon2-cffi                   20.1.0
     async-generator               1.10
     attrs                         21.2.0
     Babel                         2.9.1
     backcall                      0.2.0
     backports.functools-lru-cache 1.6.4
     bleach                        4.0.0
     brotlipy                      0.7.0
     certifi                       2021.5.30
     cffi                          1.14.6
     chardet                       4.0.0
     conda                         4.10.1
     conda-package-handling        1.7.3
     cryptography                  3.4.7
     debugpy                       1.4.1
     decorator                     5.0.9
     defusedxml                    0.7.1
     entrypoints                   0.3
```

</td>
</tr>
</table>

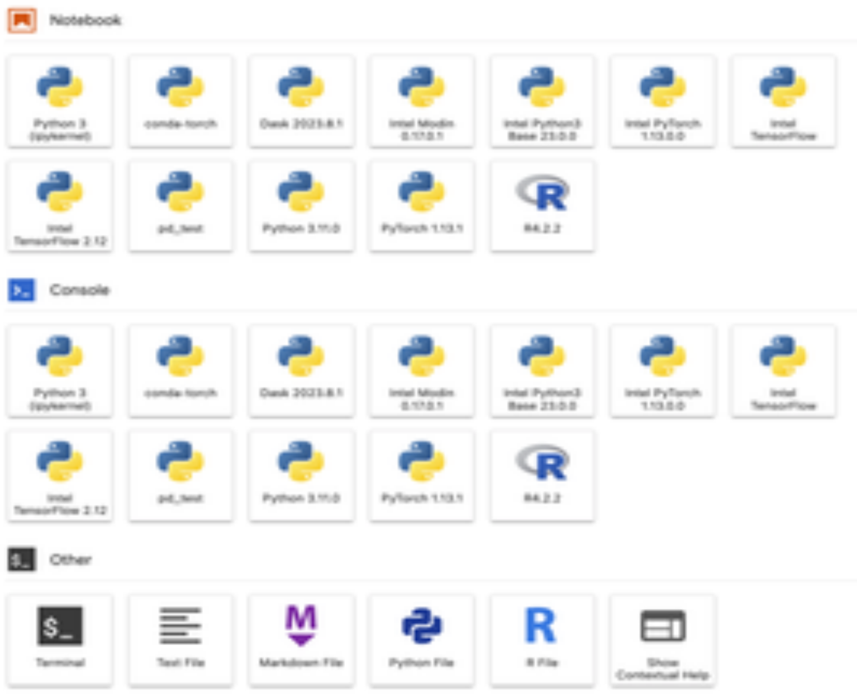
Notebook cell types

By default, there are 4 types of Notebook cells:

- Markdown
- Code
- NBConvert
- Heading

Jupyter Labs

- Jupyter's "next generation interface to work with notebooks, code, and data
- Includes notebooks, but extends to consoles, terminals, CSV editors, markdown editors, interactive maps, etc.
  - Users can easily write their own plugins.
  - Workspace consists of a main work area, where you can open multiple documents and activities, and a collapsible left sidebar that provides access to the file browser, running kernels and terminals, command palette, and notebook cell tools.
- Has a modular structure, allowing you to open several notebooks and added files like HTML, Text, markdown in the same window - more like an IDE.
  - The main work area in JupyterLab uses a tab-based layout, allowing you to switch between multiple open documents easily.
  - Users can drag and drop tabs to rearrange the layout, split the view to see multiple documents side-by-side, or even create new windows for a more customized workspace.
- Lab also allows users to execute code in a python console



Starting Jupyter from OnDemand

Goals:

- Go through the steps of starting a Jupyter session through the OnDemand web interface

---

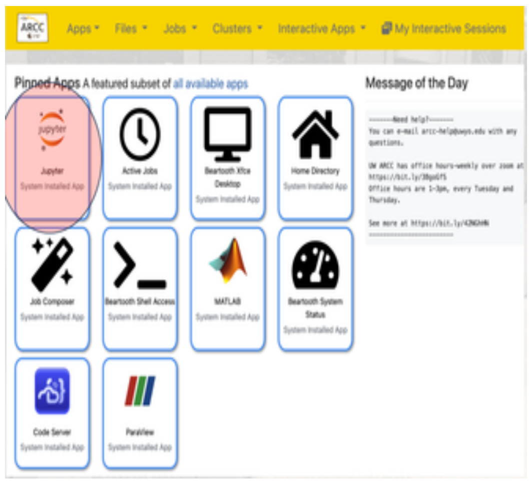
Click to start

On your local desktop, you launch jupyter through command line, by typing a command to launch it like:

```
jupyter notebook
```

Alternatively, OnDemand interactive applications can be launched from OnDemand with graphics. This should appear similar to a remote desktop window that only gives you remote access to the launched application.

After logging into OnDemand on your favorite ARCC HPC resource, you can request a Jupyter Session by clicking on the app from the main Dashboard:



```
--------Need help?--------
You can e-mail arcc-help@uwyo.edu with any questions.

UW ARCC has office hours-weekly over zoom at https://bit.ly/3BgoGf5
Office hours are 1-3pm, every Tuesday and Thursday.

See more at https://bit.ly/42NGhHN
========================
```

Fill out the Jupyter Session Request Form

After clicking the jupyter app, you are taken to a web form to tailor and specify the Jupyter environment you'd like to run in your session

Jupyter Interface: Select from Jupyter Notebook or Jupyter Lab

Account: The associated investment account or project you're using to run the session

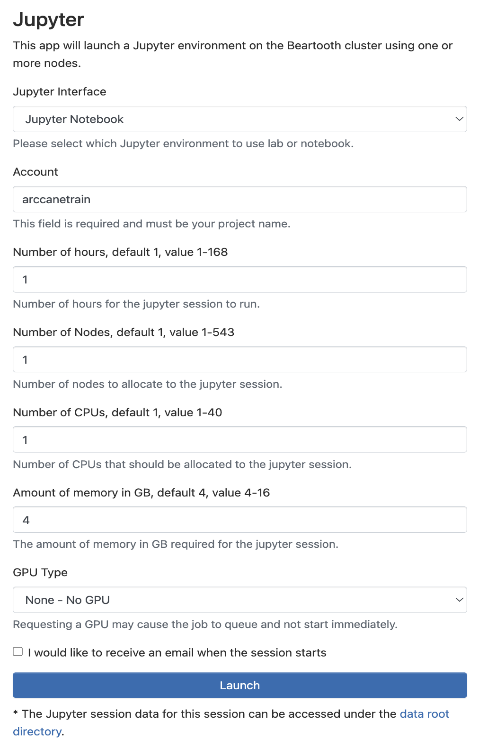
Number of hours: How long you plan to use the notebook

Number of Nodes: how many nodes you want allocated to perform work while using this notebook.

Number of CPUs: how many cores you will need access to perform your work while using this notebook.

Amount of Memory: Memory in GB required to run throughout the course of this Jupyer session

GPU Type: Which GPU hardware you'd like to perform your work in the Jupyter Notebook or Lab on



Jupyter

This app will launch a Jupyter environment on the Beartooth cluster using one or more nodes.

Jupyter Interface: Jupyter Notebook

Please select which Jupyter environment to use lab or notebook.

Account: arccanetrain

This field is required and must be your project name.

Number of hours, default 1, value 1-168: 1

Number of hours for the jupyter session to run.

Number of Nodes, default 1, value 1-543: 1

Number of nodes to allocate to the jupyter session.

Number of CPUs, default 1, value 1-40: 1

Number of CPUs that should be allocated to the jupyter session.

Amount of memory in GB, default 4, value 4-16: 4

The amount of memory in GB required for the jupyter session.

GPU Type: None - No GPU

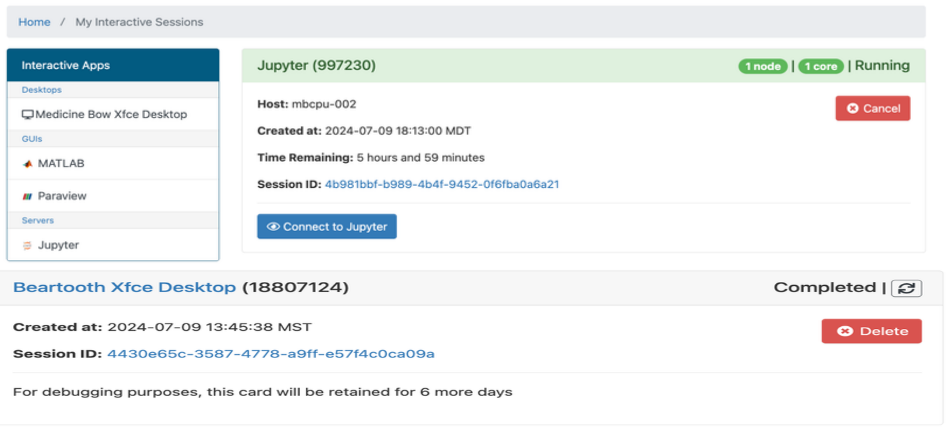
Requesting a GPU may cause the job to queue and not start immediately.

☐ I would like to receive an email when the session starts

* The Jupyter session data for this session can be accessed under the data root directory.

Your interactive sessions

- When you click *launch*, you're redirected to a page showing a list of your most recent interactive sessions.
- The Slurm scheduler assigns a compute node with a specified number of cores, memory, hardware and timeframes as requested from the input you provided in your webform.
- When your session is ready for use, the heading will turn green.
  - Completed sessions are denoted with gray headings
  - Pending sessions are denoted with blue headings



Jupyter (997230) 1 node | 1 core | Running

Host: mbcpu-002
Created at: 2024-07-09 18:13:00 MDT
Time Remaining: 5 hours and 59 minutes
Session ID: 4b981bbf-b989-4b4f-9452-0f6fba0a6a21

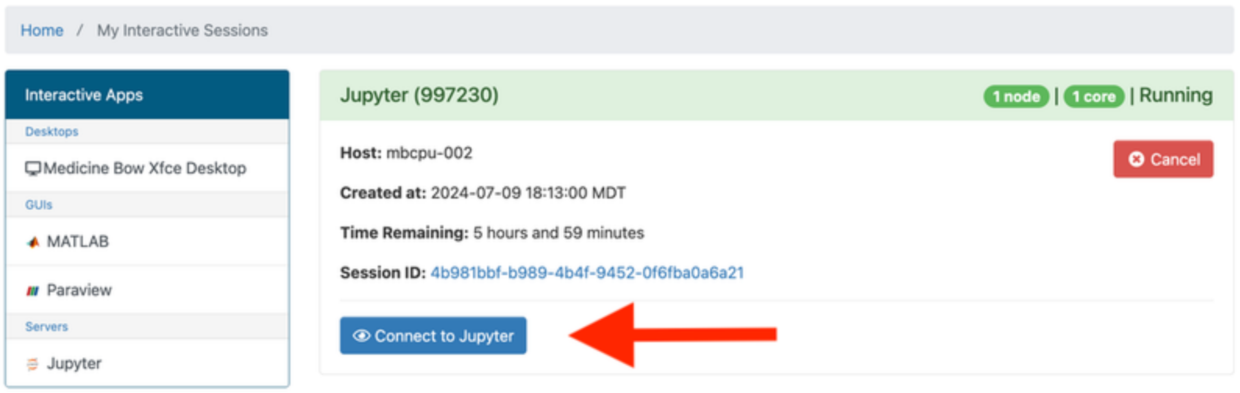
Beartooth Xfce Desktop (18807124) Completed |

Created at: 2024-07-09 13:45:38 MST
Session ID: 4430e65c-3587-4778-a9ff-e57f4c0ca09a

For debugging purposes, this card will be retained for 6 more days

Connect to your session

To open Jupyter, click on the connect button within the active session



Jupyter (997230) 1 node | 1 core | Running

Host: mbcpu-002
Created at: 2024-07-09 18:13:00 MDT
Time Remaining: 5 hours and 59 minutes
Session ID: 4b981bbf-b989-4b4f-9452-0f6fba0a6a21

You will be directed to a Jupyter notebook or lab environment to start using Jupyter!

Dive into Jupyter Notebooks

Goals:

- Walk through a navigating within a Jupyter Notebook session
- Demonstrate options and features available in Jupyter Notebooks

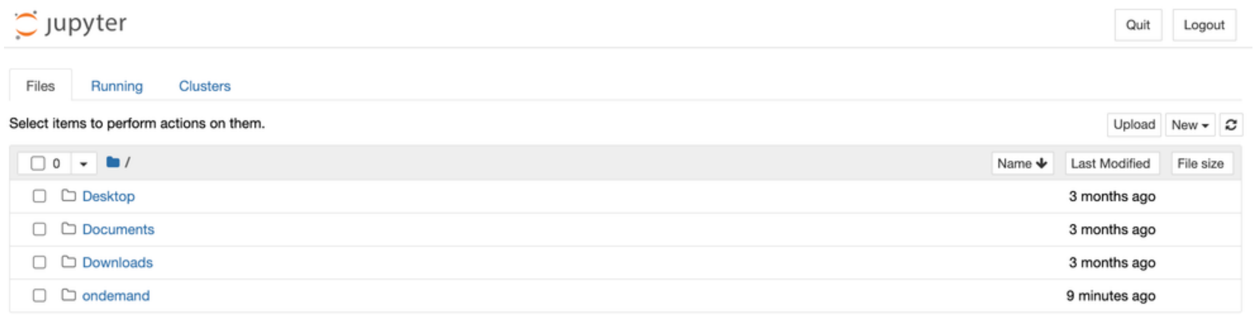
---

Initial Screen Navigation and Options

Upon connecting, you are presented the jupyter dashboard which serves as your home page for jupyter notebook. The Jupyter Notebook screen is rather simple with 3 tabs:

- Files: (Default selected) Interactive view of the portion of the filesystem accessible by the user, rooted by the directory in which the notebook was launched from.

- Running: Displays currently running notebooks known to the server. (You can manage notebook kernels from here)
- Clusters: Gives a summary of iPython Parallel clusters
  (More about this later)



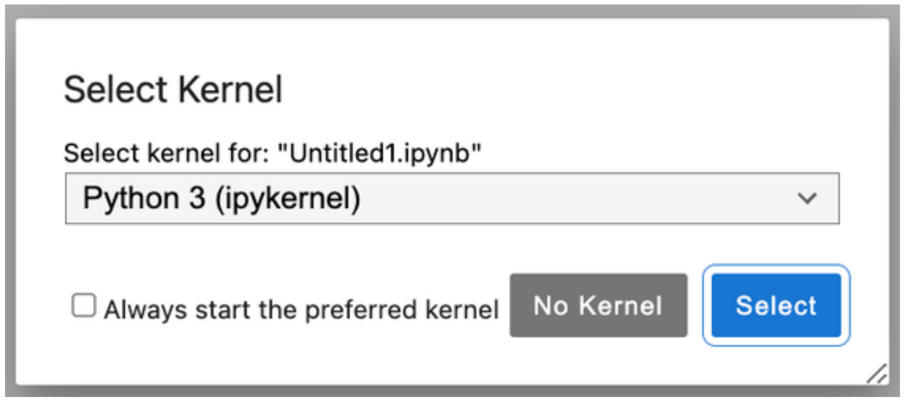
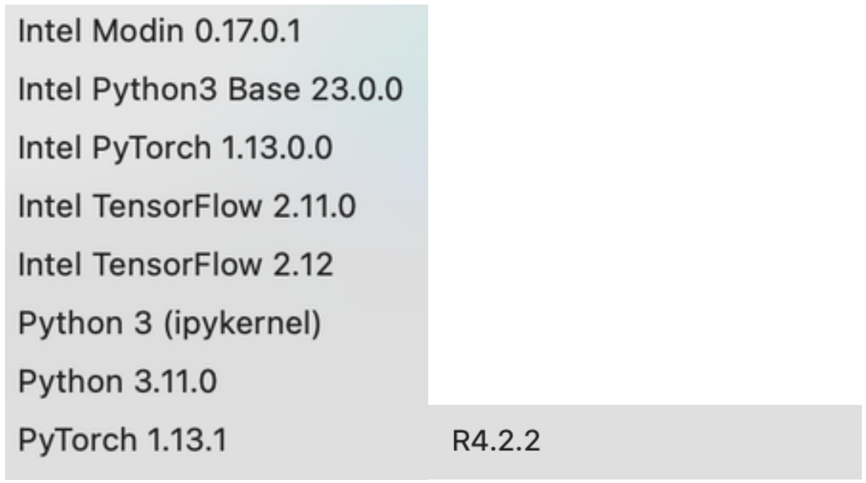
Select items to perform actions on them.

Name	Last Modified
Desktop	3 months ago
Documents	3 months ago
Downloads	3 months ago
ondemand	9 minutes ago

What are Kernels?

A Jupyter kernel is the computational engine behind the code execution in Jupyter notebooks.

Most users think of this as the “compiler” or programming language used when running code cells.
The Kernel empowers you to execute code in different programming languages like Python, R, or Julia or other languages and instantly view the outcomes within the notebook interface.

After opening a new notebook, you will be prompted to select a kernel - If you have never created a kernel to use, you will only see a list of default Jupyter kernels available on the cluster - You may check the box to start with the preferred kernel every time you open a notebook	 Select Kernel Select kernel for: "Untitled1.ipynb" Python 3 (ipykernel) ☐ Always start the preferred kernel
Default Kernels on ARCC HPC Resources include: - Python Kernels - R Kernels HPC-wide kernels are titled by packages installed and available when launched Users can also create user-defined kernels from conda environments (Covered in a subsequent module. See: Launching Jupyter Kernels from Conda Environments)	 Intel Modin 0.17.0.1 Intel Python3 Base 23.0.0 Intel PyTorch 1.13.0.0 Intel TensorFlow 2.11.0 Intel TensorFlow 2.12 Python 3 (ipykernel) Python 3.11.0 PyTorch 1.13.1 R4.2.2

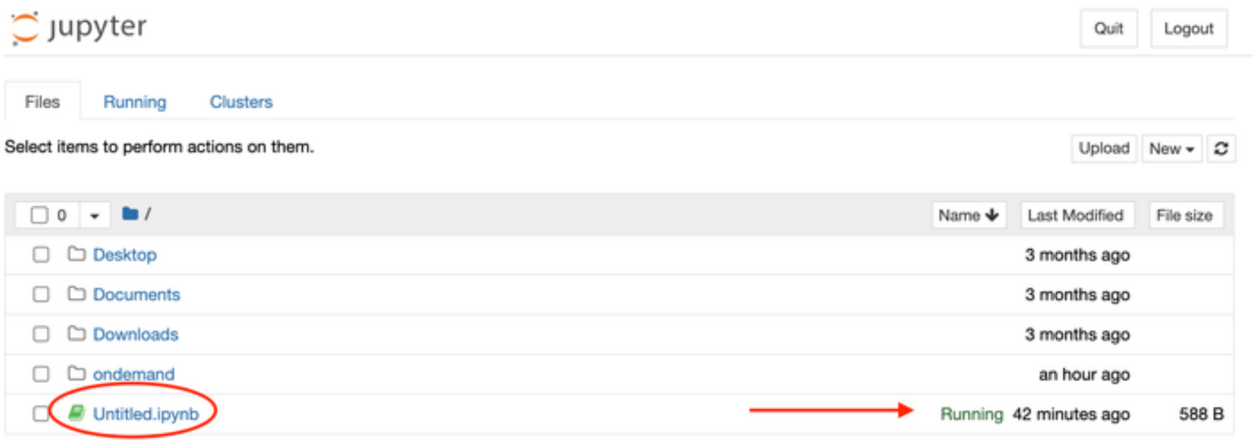
Open a New Blank Notebook

From the Right side of the File Management Tab:

New->Notebook-> Select from a list of kernels. Choose `Python 3 (ipykernel)`

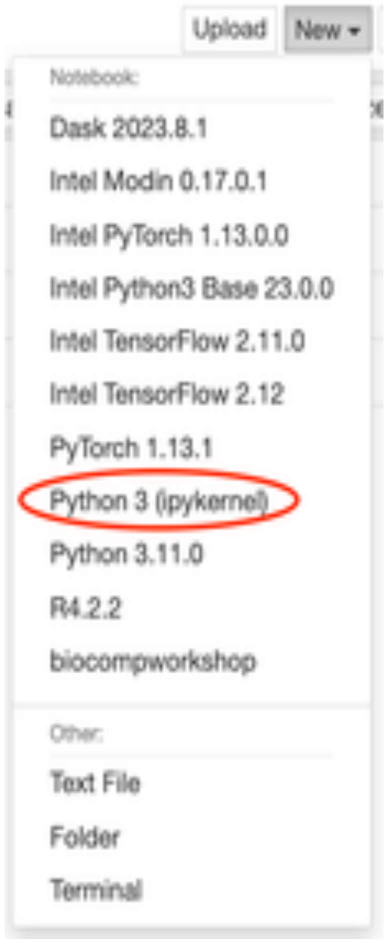
This should open a new browser tab/window with a blank Jupyter notebook named: `Untitled.ipynb`

If we go back to our previous Jupyter tab/window containing the file browser from which we launched our notebook, this new file shows up in the list, and has a green icon to it's left, meaning it is currently running:



Select items to perform actions on them.

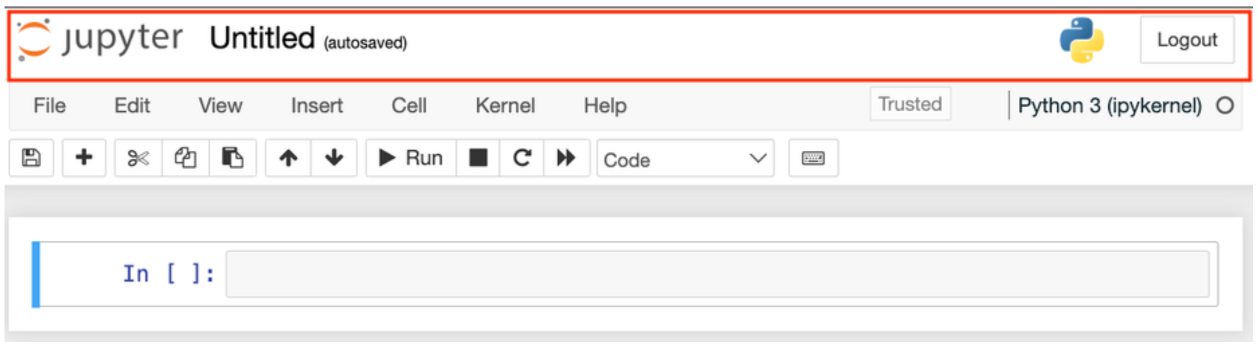
	Name	Last Modified	File size
	Desktop	3 months ago	
	Documents	3 months ago	
	Downloads	3 months ago	
	ondemand	an hour ago	
	Untitled.ipynb	Running 42 minutes ago	588 B



New Notebook - New Options

When a notebook is open a new browser tab is created showing the notebook user interface (UI). This allows for interactive editing and running of the notebook document.

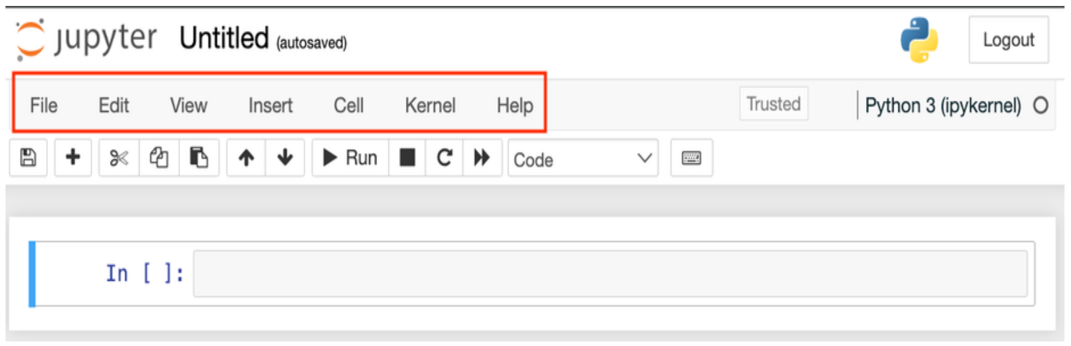
- **Header:** Top has the document name (editable).
- **Menu bar with drop-downs & loaded kernel**
- **Toolbar**
- **Body**


jupyter Untitled (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code
In [ ]:

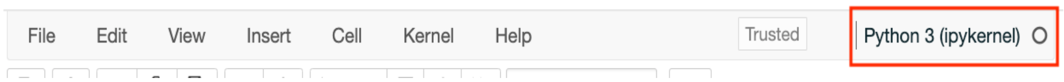
Menu Bar with Dropdowns

- Has top-level menus that expose actions available in Jupyter Notebook:
  - **File**: actions related to files and folders
  - **Edit**: actions related to editing notebooks
  - **View**: Options to alter appearance of Notebook
  - **Insert:** Limited options for cell insertion
  - **Kernel**: actions for kernel management
  - **Help**: a list of Jupyter help links

Note: [Jupyter extensions](#) can create new top-level menus in the menu bar.


jupyter Untitled (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code
In [ ]:

Right of the menu bar, the current kernel is listed



Toolbar Actions

- Save and checkpoint notebook

- Add a cell below the current one

- Cut/Delete this cell

- Copy contents of current cell

- Paste in new cell below active cell

- Up 1 cell

- Down 1 cell

Run - Run current cell

- Stop running cell

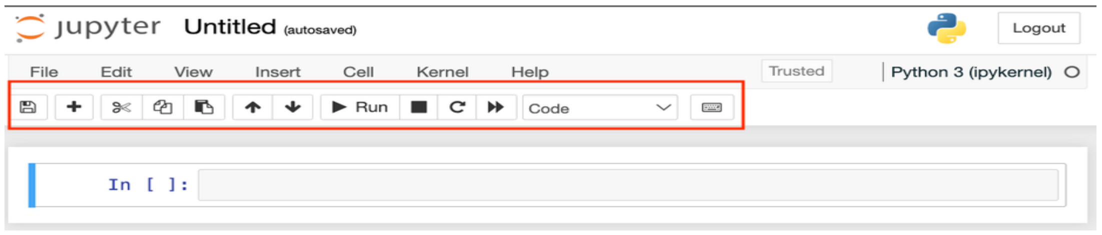
- **Reload/Restart Kernel

- **Restart Kernel & Re-run entire notebook

Code - Select current cell type

- Display full list of keyboard shortcuts for Jupyter Notebooks

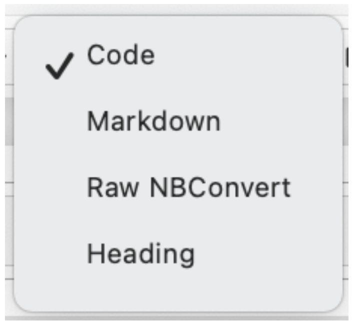
** - Will restart entire kernel and you will lose all current output. (Is output easily regenerable?)



Notebook Cell Types

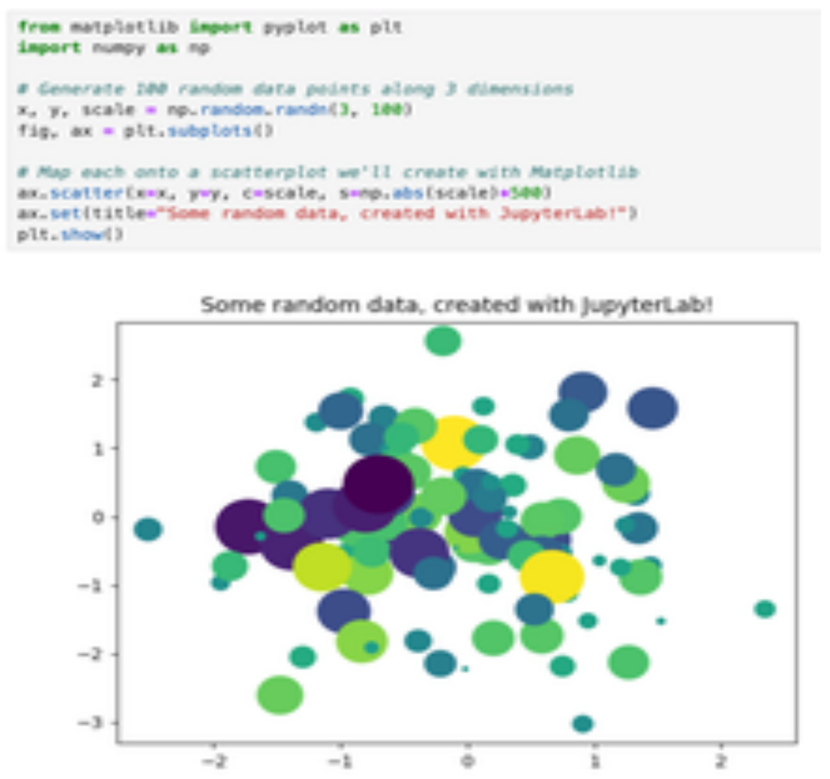
We can use the cell type option in the toolbar to set cell type in the notebook body:

- **Code:** Define computational code (language = from kernel) in the document.
  - If the kernel is python cell type, the cell will expect input in the form of python code.
  - This is our default code type when new cells are created.
- **Markdown:** Uses Markdown language to build nicely formatted narratives around the code in the rest of the document. Click here for Markdown Cheat Sheet
- **Raw NBconvert:** Used when text should be kept in raw form for conversion to another format (such as HTML or Latex). When you use these, cells marked as Raw are converted in a way specific to your targeted output format.
- **Heading:** For making headings. Somewhat redundant - you can also make headings in a markdown cell.

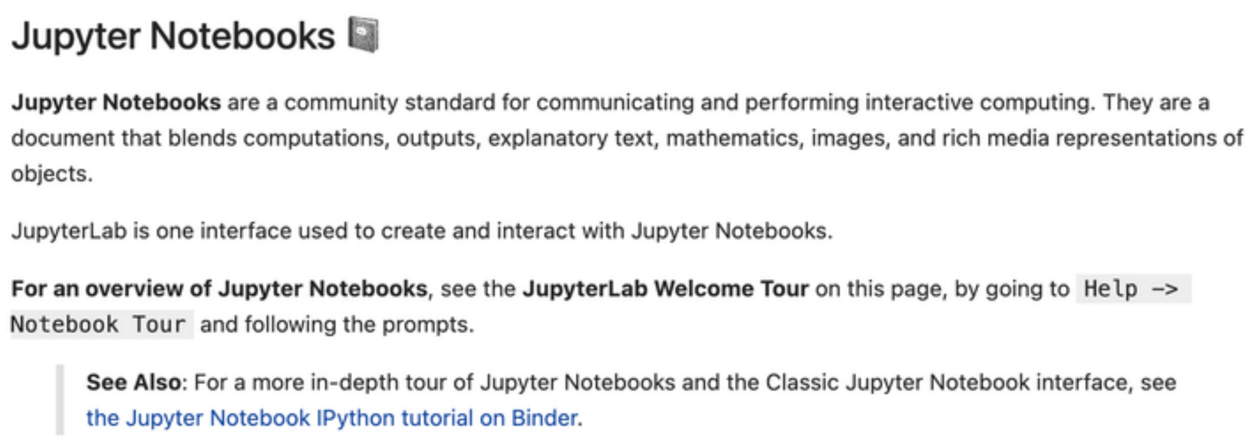


Code

- **Code cells allow you to write and run programming code in a language of your choosing (e.g., `Python`)**
- **Languages supported in Jupyter include Python, R, Julia, and many others**
- **On ARCC HPC resources, we support jupyter code in Python and R**
- **After running, they can and usually do provide some form of output**

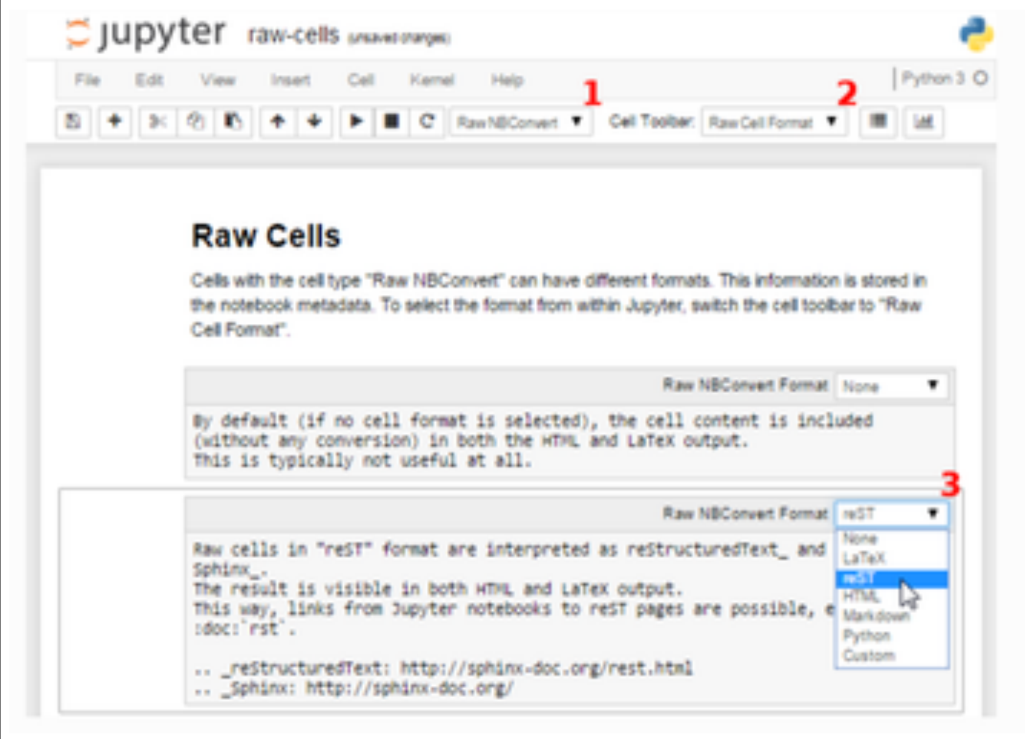


```python
from matplotlib import pyplot as plt
import numpy as np

# Generate 100 random data points along 3 dimensions
x, y, scale = np.random.randn(3, 100)
fig, ax = plt.subplots()

# Map each onto a scatterplot we'll create with Matplotlib
ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500)
ax.set(title="Some random data, created with JupyterLab!")
plt.show()
```

Markdown - **Text Cells allowing you to write and render `Markdown` syntax** - **Where you describe and document your workflow**	 Jupyter Notebooks **Jupyter Notebooks** are a community standard for communicating and performing interactive computing. They are a document that blends computations, outputs, explanatory text, mathematics, images, and rich media representations of objects. JupyterLab is one interface used to create and interact with Jupyter Notebooks. **For an overview of Jupyter Notebooks**, see the **JupyterLab Welcome Tour** on this page, by going to `Help -> Notebook Tour` and following the prompts. > **See Also**: For a more in-depth tour of Jupyter Notebooks and the Classic Jupyter Notebook interface, see the Jupyter Notebook IPython tutorial on Binder.

Raw NBConvert

- Stands for "Raw Notebook Convert"
- Retains any text in these cells in their raw form and does not run them
- Enables the conversion of your notebook to another format as given by the FORMAT string using Jinja templates.
    - Presenting: PDF
    - Publishing: LaTeX
    - Collaboration
    - Sharing: HTML
- Setting to "none" just makes it a "Raw" cell in which nothing is run on it.


jupyter raw-cells (unsaved changes)
File Edit View Insert Cell Kernel Help
Python 3
1
Raw NBConvert
2
Cell Toolbar: Raw Cell Format
3
None
LaTeX
reST
HTML
Markdown
Python
Custom

Raw Cells

Cells with the cell type "Raw NBConvert" can have different formats. This information is stored in the notebook metadata. To select the format from within Jupyter, switch the cell toolbar to "Raw Cell Format".

Raw NBConvert Format: None

```
By default (if no cell format is selected), the cell content is included
(without any conversion) in both the HTML and LaTeX output.
This is typically not useful at all.
```

Raw NBConvert Format: reST

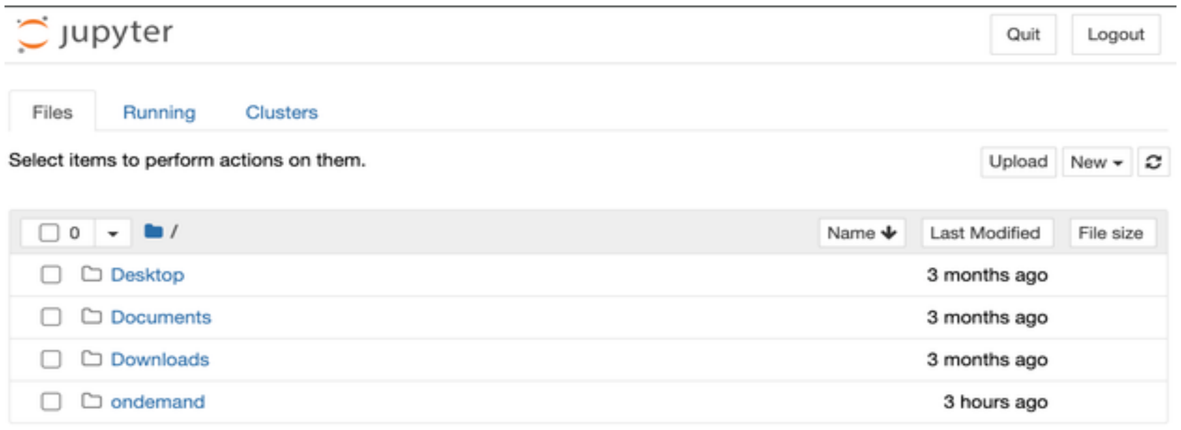
```
Raw cells in "reST" format are interpreted as reStructuredText_ and
Sphinx_.
The result is visible in both HTML and LaTeX output.
This way, links from Jupyter notebooks to reST pages are possible, e
:doc:`rst`.

.. _reStructuredText: http://sphinx-doc.org/rest.html
.. _Sphinx: http://sphinx-doc.org/
```

Where are we?

Previously, we said the file management tab shows the filesystem accessible to the user, rooted by the directory from which the notebook was launched.

In the file management tab we can see root directory, and within it, the Desktop, Documents, and Downloads, and ondemand directories.

We could just assume the file manager is showing our home directory. But how would we find out for certain?



Select items to perform actions on them.

/

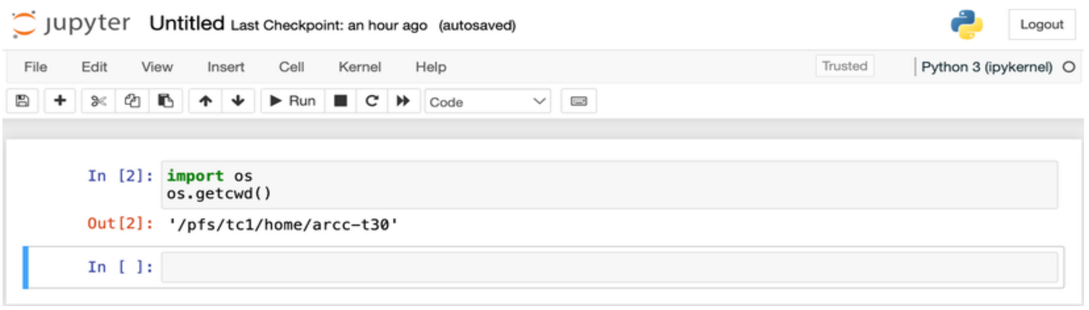
Name	Last Modified	File size
Desktop	3 months ago	
Documents	3 months ago	
Downloads	3 months ago	
ondemand	3 hours ago	

Running with a Python kernel, we can use our jupyter notebook to get this information from the system:

1. import the python OS module (to let us interact with the native OS on the cluster that Jupyter is running on top of)
2. On the next line, type `os.getcwd()` (AKA: get current working directory)
3. Click the run button ▶ Run to run our cell and generate a new output cell, which also creates a new input cell below that.

Note: New input cells are code cell types by default

With the information from our output cell, we can conclude that OnDemand launches Jupyter from your $HOME


jupyter Untitled Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [2]: import os
        os.getcwd()
Out[2]: '/pfs/tc1/home/arcc-t30'
In [ ]:
```

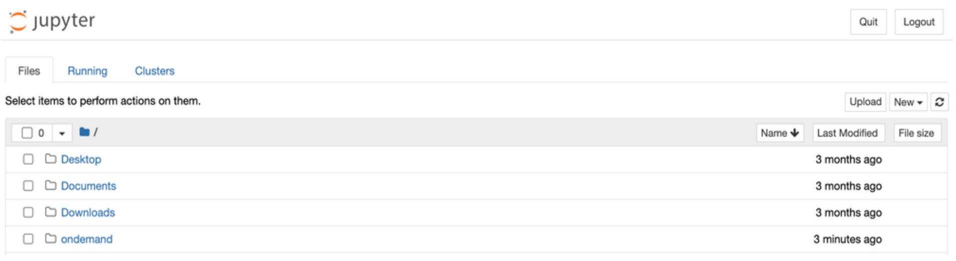
`import os` : import a python module allowing us to use python kernel running this notebook to interact with underlying HPC cluster's OS

`os.getcwd()`: a python command to output the full system path in which our active jupyter notebook resides.

How to get to directories outside of $HOME?

If we select the default Python 3 (ipykernel), we are presented with the file explorer showing our home directory as it's rooted location. This means we can't go up any further in system's directory structure.

- With our local root location for the notebook set to our $HOME, we are unable to see our `/project` and `/gscratch` directories on the cluster.
- To expose these folders to the jupyter environment, create a symbolic link (aka shortcut) within our /home.



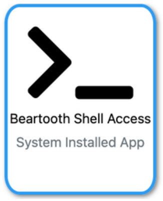
Select items to perform actions on them.

/	
Desktop	3 months ago
Documents	3 months ago
Downloads	3 months ago
ondemand	3 minutes ago

Instructions for creating a symbolic link may be found here or expanded in the cell to the right

Steps to create a symbolic link

1. Open an ssh connection to the HPC cluster with:
   `ssh your_username@clustername.arcc.uwyo.edu`
   or open a shell through OnDemand:



2. In the shell/terminal interface, create a symbolic link to your project (replacing `project_name` with the name of your project) with:

```
[~] ln -s /project/project_name/ project
```

3. In the shell/terminal interface, create a symbolic link to your gscratch (replacing `username` with the your username on the HPC) with:

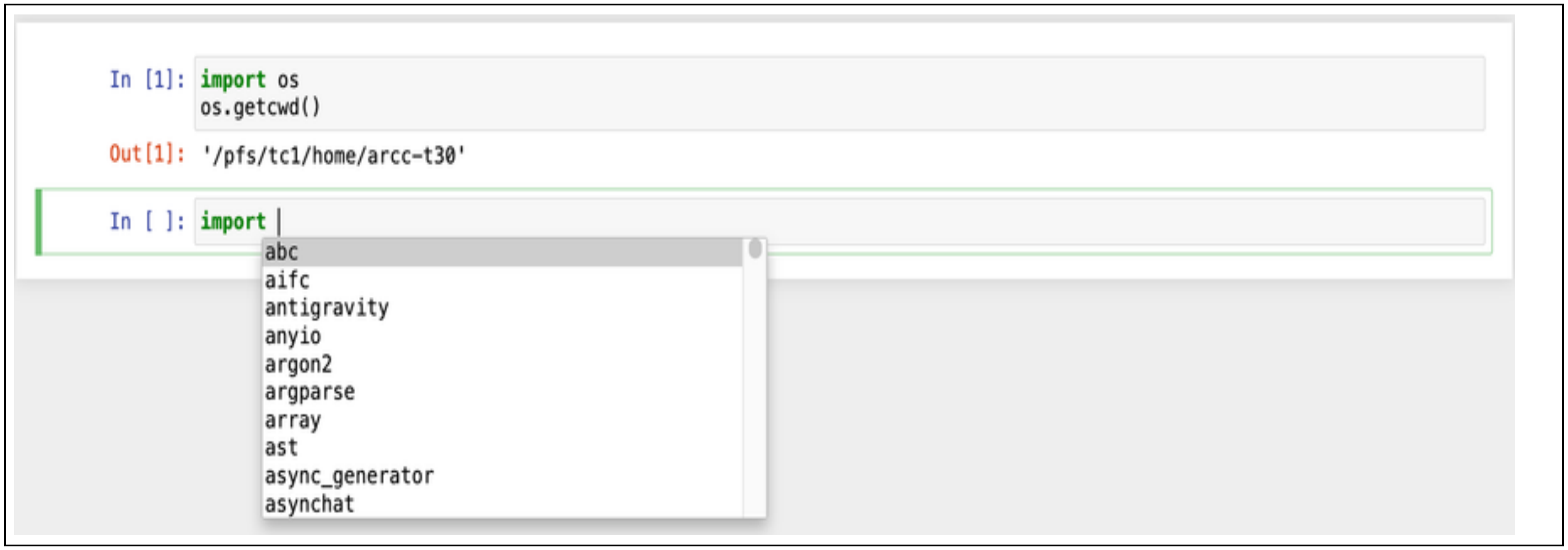
```
[~] ln -s /gscratch/username/ gscratch
```

What Packages are Available in our Kernel?

In our notebook, we can see which modules are available by opening a new cell with the + button.

In our cell box, set as "code" use the python `import` command, followed by a space, then hit tab to get a list of options.

Hitting tab after `import` runs autocomplete options for the import command. This list of options has populated all modules available to us in our jupyter notebook:

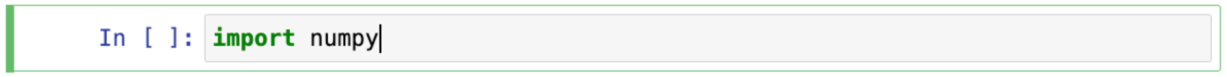


```
In [1]: import os
        os.getcwd()

Out[1]: '/pfs/tc1/home/arcc-t30'

In [ ]: import |
```

New Cell in our Notebook

Since we appear to have a large number of packages available in this environment, we'll import one we expect to be there.

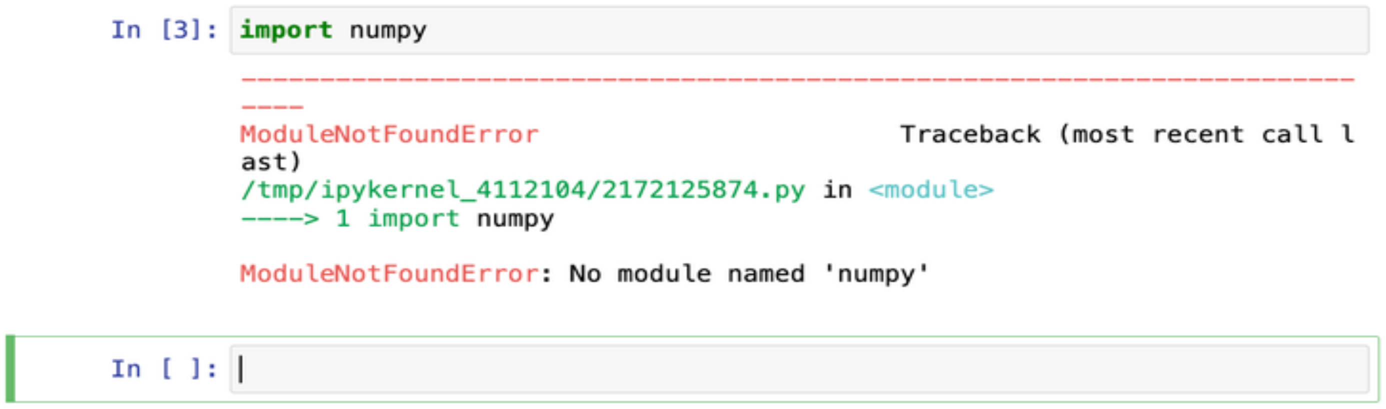
In our bottom-most cell, add to the import command by typing an import for a common package used in mathematic and multi dimensional matrix computations - numpy.



```
In [ ]: import numpy
```

Run this command with the run button

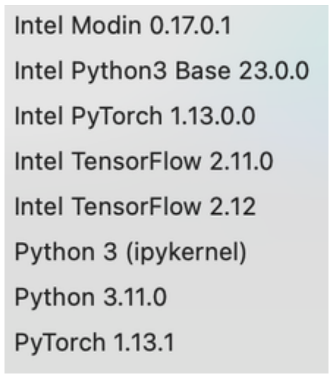
Our output results in an error:



```
In [3]: import numpy

        ---------------------------------------------------------------------------
        ModuleNotFoundError                       Traceback (most recent call l
        ast)
        /tmp/ipykernel_4112104/2172125874.py in <module>
        ----> 1 import numpy

        ModuleNotFoundError: No module named 'numpy'

In [ ]: |
```

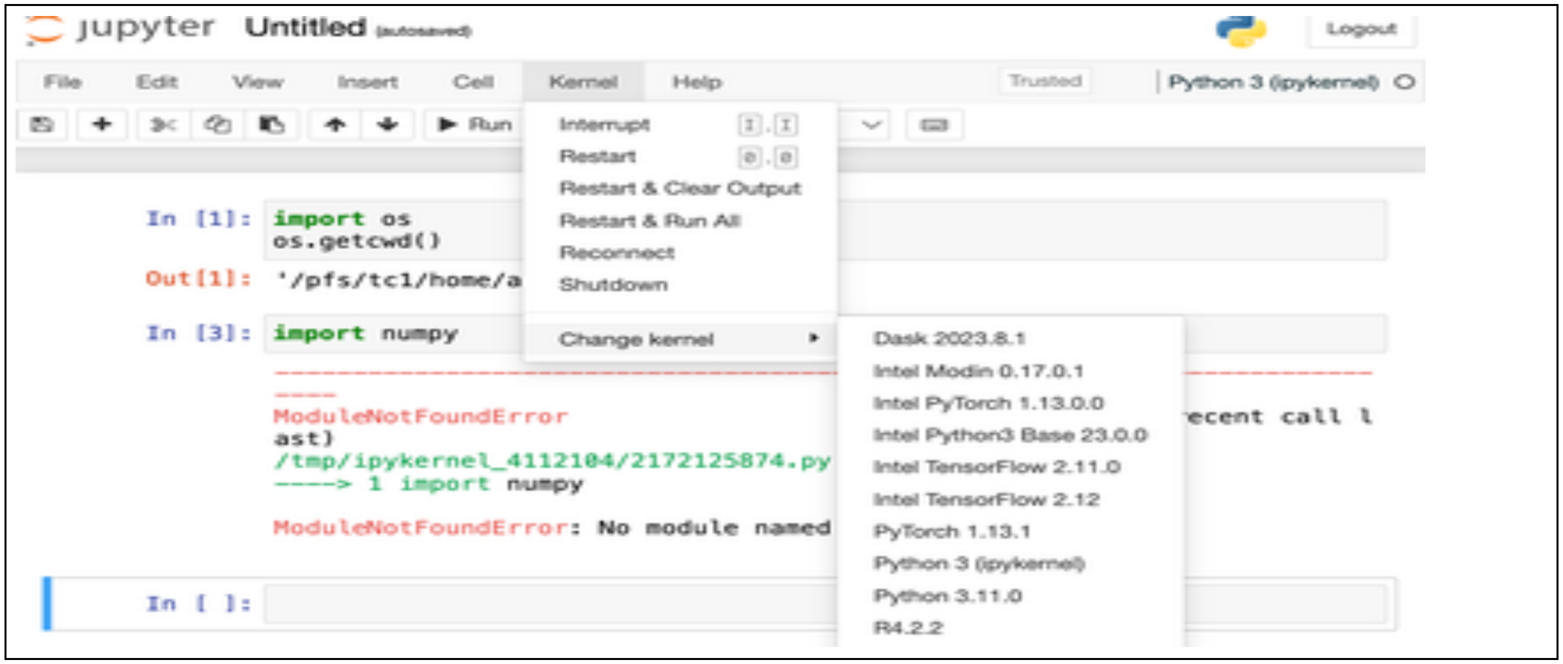
- The error means this particular module is not available in the kernel we have loaded, despite being a commonly used software package for researchers and computations.
- While many packages were listed when we autocompleted an import command, most of them were installed as part of the jupyter installation and underlying OS environment.
- Most software we'd need to perform even more simple and common activities for our research would still need to be installed or made available somehow. What are our options?

Load a different kernel

Depending on the HPC's native environment, you may have other kernels available.	 Intel Modin 0.17.0.1 Intel Python3 Base 23.0.0 Intel PyTorch 1.13.0.0 Intel TensorFlow 2.11.0 Intel TensorFlow 2.12 Python 3 (ipykernel) Python 3.11.0 PyTorch 1.13.1

To load a different kernel, we go to the Kernel option in our drop down menu then navigate to "Change kernel".

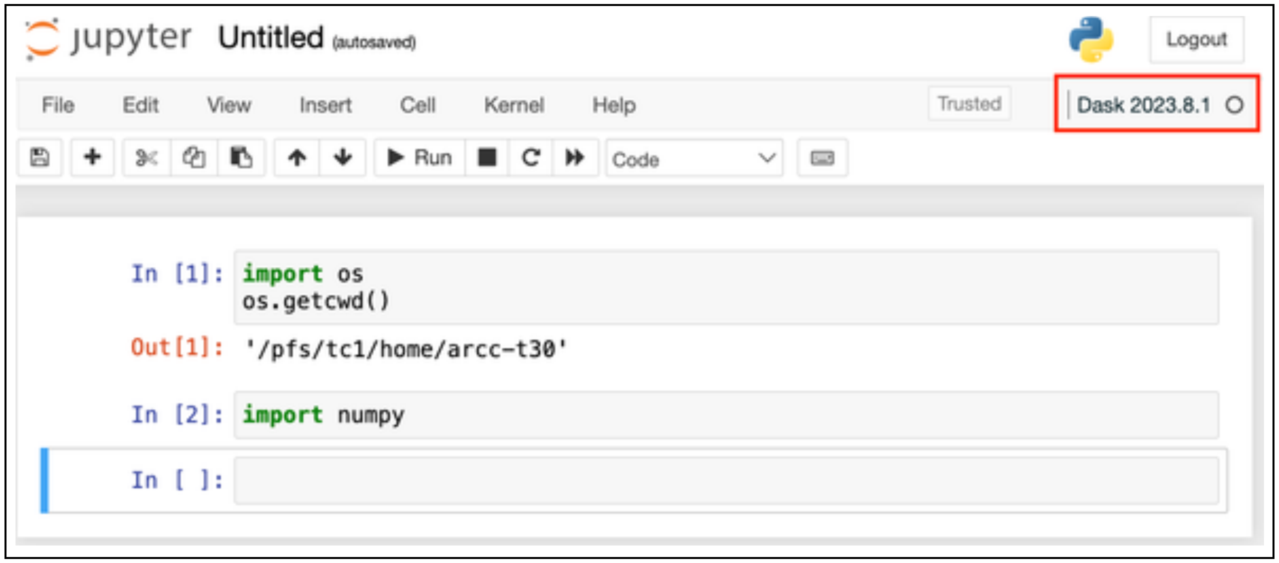
- To load a different kernel, we go to the Kernel option in our drop down menu then navigate to "Change kernel".
- Select a different kernel than the current one, based on your own preference



```
In [1]: import os
        os.getcwd()
Out[1]: '/pfs/tc1/home/a
In [3]: import numpy
        ----------------------------------------------------------------------------
        -----
        ModuleNotFoundError                                               ecent call l
        ast)
        /tmp/ipykernel_4112104/2172125874.py
        ----> 1 import numpy

        ModuleNotFoundError: No module named
In [ ]:
```

The new kernel is loaded as shown in the top right of our notebook.

- If we rerun our 2 cells again, what happens this time?



```
In [1]: import os
        os.getcwd()
Out[1]: '/pfs/tc1/home/arcc-t30'

In [2]: import numpy

In [ ]:
```

No available kernels have all the software I need - Now what?

To be continued…

Launching Kernels from Conda Environments

Goals:

- Provide logic and reasoning for using Conda environments, versus other available options
- Walk through the process for creating personalized kernels from a Conda environment

---

Why make your own kernels?

Individual researchers on ARCC systems often need access to specific software and modules - specific to their focus of study or research.

While ARCC provides general “base” kernels, they typically lack all software packages specific to any one researcher’s needs. To solve this, researchers may perform installs within the jupyter notebook (with `!pip` for Python or `install.packages()` for R).

ARCC recommends against this practice because by default this will install the packages within the user’s $HOME creating a new set of issues in the future.

Issue 1: Storage Quotas

1. `$HOME` directories on HPC are usually relatively small compared to storage in `/project` or `/gscratch`. ARCC provides a default quotas on HPC for each user’s `$HOME`, but this can fill up quickly when performing frequent python `pip` or R `install.package()` installations, which leads to exceeding the storage quota in your `$HOME` directory.
   a. While installation locations can be redirected, several other configuration changes should be made to make them available and work appropriately.
   b. It’s far more straightforward to create Conda environments and redirect Conda installations with the `-p` flag (to specify installation path).

Issue 2: Software Conflicts

2. By default, local installs will install to one central location within your home directory (`pip` under `~/.local`, `install.packages()` under `~/R`)
    a. Package installations are separated by kernel software versions, but if conflicts exist within the overall install location, packages are overwritten to make the software and dependencies "fit" with your most recently requested installation. This can change installation and available packages for your user profile, breaking older installations and software that you still want to use.
    b. Installing in your `$HOME` directory makes packages and versions of software in your /home available to you regardless of whether you want them at the time, or not.
    c. This causes software conflicts between versions native to the HPC system, those you may want to load off and on with modules, and those you've installed in `$HOME`, meaning that loaded modules or native HPC software may not work properly or crash due to underlying dependencies can be superseded by packages in `$HOME`.

Kernels from Conda Environments

In the Conda module, we learned that conda allows software environments to be contained, meaning they do not conflict with one another, and can be loaded and unloaded so that they are exposed to you only when you want them to be.

In OnDemand Jupyter sessions you can launch a Jupyter session utilizing the software packages from a Conda environment you've set up yourself.

- This becomes very useful if you are using a package that requires extensions be installed in the environment that is launching the Jupyter session.
- To configure your environment so that it launches as you want, you should ensure that the appropriate packages are installed within the Conda environment.
- In this module, we will go through steps needed to correctly create your environment and configure it as a kernel available to you within your Jupyter sessions.

Exporting your Conda Environment to a Kernel (Python)

In these steps, we assume you've already created your desired Conda environment. To learn how to create a Conda environment, please see our Conda module.

With our Conda environment already installed and configured, we can now set it up to be used as a jupyter kernel. (To learn about how to make your own Conda environments, see our training on Conda)
1. Open a Command Line Terminal on the HPC resource.
2. Load Miniconda
3. Activate your Conda environment
4. Install your kernel:
`conda install ipykernel` for python kernels
5. Set your environment to be recognized as an available kernel
`python -m ipykernel install --user --name=<kernelname>`
	For Python:
	`#open command line interface`
	`module load miniconda3`
	`conda activate <insert_environment_name_or_path_here>`
	`conda install ipykernel`
	`python -m ipykernel install --user --name=mypythonkernel`

Exporting your Conda Environment to a Kernel (R)

In these steps, we assume you've already created your desired Conda environment. To learn how to create a Conda environment, please see our Conda module.

With our Conda environment already installed and configured, we can now set it up to be used as a jupyter kernel. (To learn about how to make your own Conda environments, see our training on Conda)

1. Open a Command Line Terminal on the HPC resource.
2. Load Miniconda
3. Activate your Conda environment
4. Install your kernel:
   `conda install r-irkernel` for R kernels
5. Set your environment to be recognized as an available kernel
   a. note R location in conda
   b. Load R: `R`
   c. Install kernel: `R prompt:`
      `> IRkernel::installspec(name,displayname)`
   d. Create specific R version kernel folder in `~/.local/share/jupyter/kernels/`
   e. Copy config from <conda-env> appending the following to path after <conda-env> name with `/lib/R/library/IRkernel/kernelspec/*` to kernels folder in your `.local/share/jupyter/kernels/` folder.

For R:

```
#Step (1) open command line interface:
#Step (2) Load miniconda:
module load miniconda3
#Step (3) :
conda activate <insert_environment_name_or_path_here>
#Step (4):
conda install r-irkernel
#Step (5a):
which R
/project/<project_name>/software/conda/r/rtest/bin/R
#Launch R (step 5b):
R
#Install kernel in R prompt (step 5biii):
> install.packages('IRkernel')
>  IRkernel::installspec(name='r4.4.1',  displayname='
4.4.1 Kernel')
> q()
#Step (5c):
#Change directory to jupyter kernel directory:
cd ~/.local/share/jupyter/kernels/
#Step (5d): make a directory for your kernel:
mkdir r4.4.1
#change directory to new directory:
cd r4.4.1
#Step (5e) Copy kernelspec from conda to .local:
cp
/project/<project_name>/software/conda/r/rtest/lib/R/
library/IRkernel/kernelspec/*
~/.local/share/jupyter/kernels/r4.4.1/.
```

Running a Console Kernel

Open a new Jupyter Session

Select your new kernel from the dropdown list

Next Steps

Previous	Workshop Home
Dive into Jupyter Notebooks	Jupyter with OnDemand

Use the following link to provide feedback on this training: https://forms.gle/qBBwXpKeTNqSR5516 or use the QR code below.