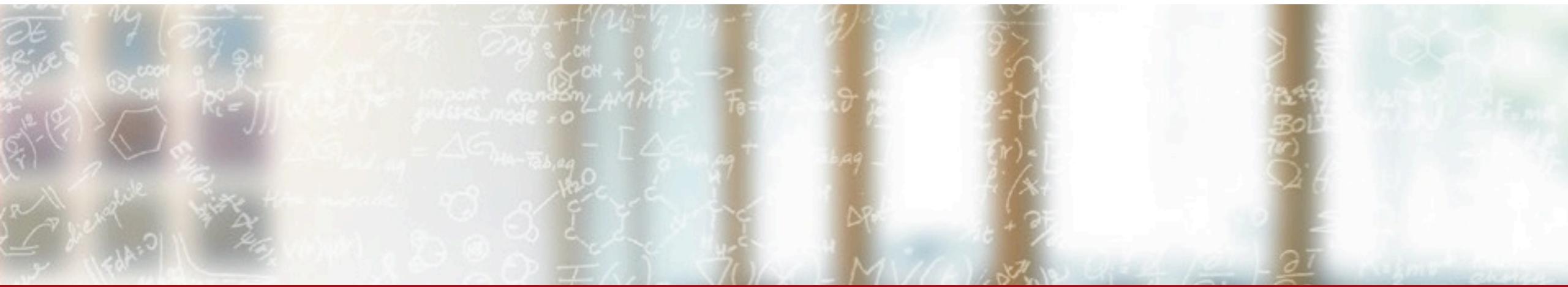


Agenda

1. New features available in JupyterLab and JupyterHub
2. Supercomputing with Julia



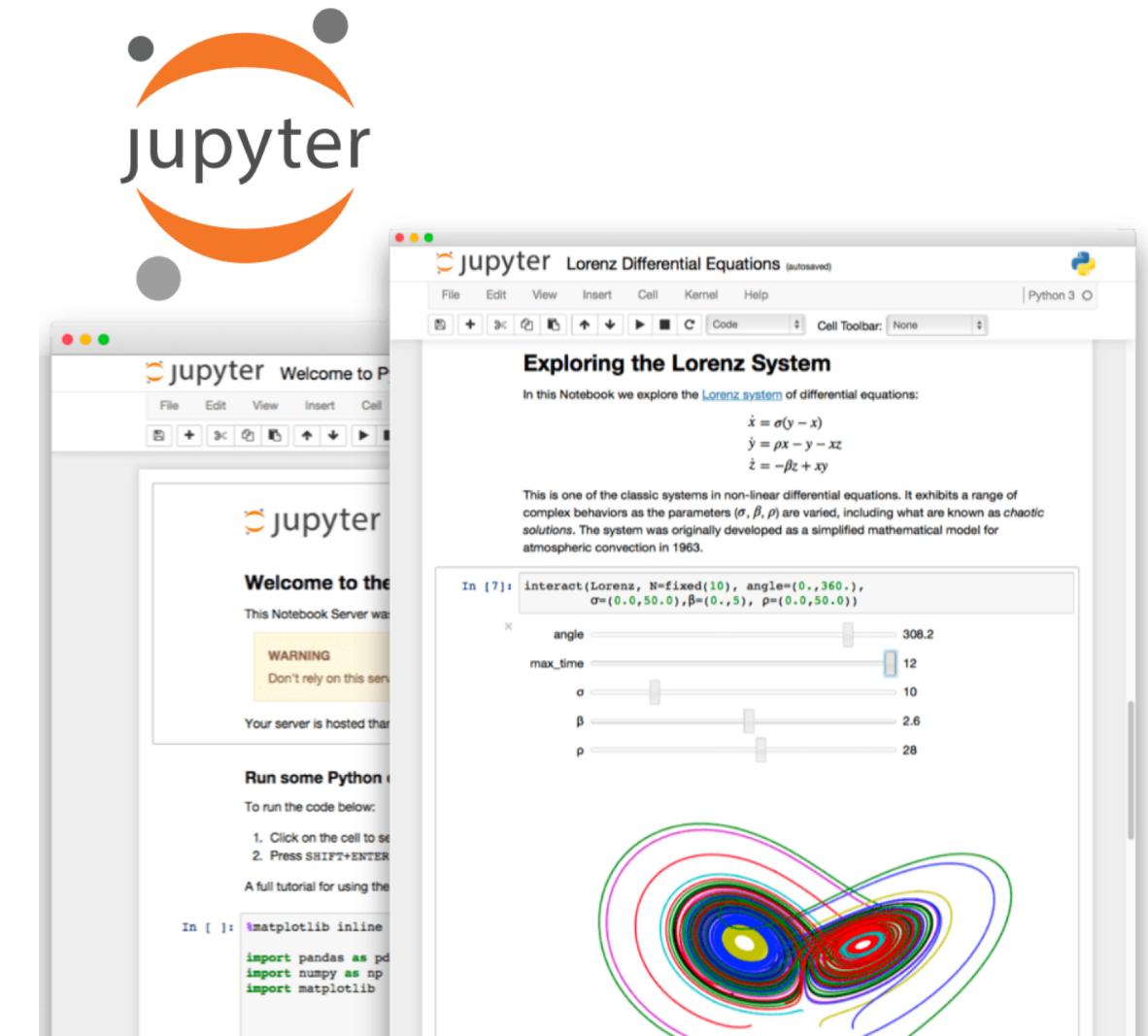
Interactive Supercomputing with Jupyter

Announcing JupyterLab 1.1.1 and JupyterHub 1.0

CSCS User Lab Day
Tim Robinson (ETH Zurich / CSCS)
September 9, 2019

Introduction to Jupyter Notebook

- Jupyter Notebook is an open-source web app for creating reproducible computational narratives
- Documents can contain live code, equations, narrative text, visualizations, rich media, etc..
- Documents can be shared or exported to PDF, HTML, LaTeX, etc.
- Working environment includes
 - File browser
 - Terminal
 - Support for many languages: Python, R, Julia, C++, ...
 - Extensible design
 - Many server/client plugins
- Notebook is attached to a “kernel”, which does the actual computation



JupyterHub for the User Lab

- <https://jupyter.cscs.ch>
- JupyterHub provides Jupyter servers on demand for users of Piz Daint
- Log in with your CSCS credentials
- Spawns a server on a dedicated compute node of Piz Daint (gpu or mc)
- Launch time should be 5 minutes maximum
- To load modules or activate virtual environments, add these commands to `$HOME/.jupyterhub.env`

The screenshot shows a web browser window for the JupyterHub service at <https://jupyter.cscs.ch/>. The page header includes the CSCS logo, navigation links for Home, Token, Admin, and the ETH Zürich logo. A message in a pink box states: "The JupyterHub service is under maintenance the first Wednesday of every month, 8:00-12:00. The default version of JupyterLab is now version 1.1.1 (05.09.2019)". The main content area is titled "Spawner Options". It contains several configuration fields:

- Piz Daint node type: gpu (dropdown)
- Queue: JupyterHub dedicated queue (single node only) (dropdown)
- Training course reservation: [empty input field]
- Number of nodes: 1 (dropdown)
- Job duration: 1 hour (dropdown)
- Account (leave empty for default): [empty input field]
- JupyterLab Version: 1.1.1 (dropdown)
- Start IPyParallel automatically with MPI? (for JupyterLab 0.35.2): No (checkbox)
- If yes, how many processes per node? (default: one process per virtual core): [empty input field]
- Start Dask.distributed automatically? (for JupyterLab 0.35.2): No (checkbox)
- If yes, how many tasks per node? (default: one task per node): 1 (dropdown)

A large red "Spawn" button is located at the bottom of the form.



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

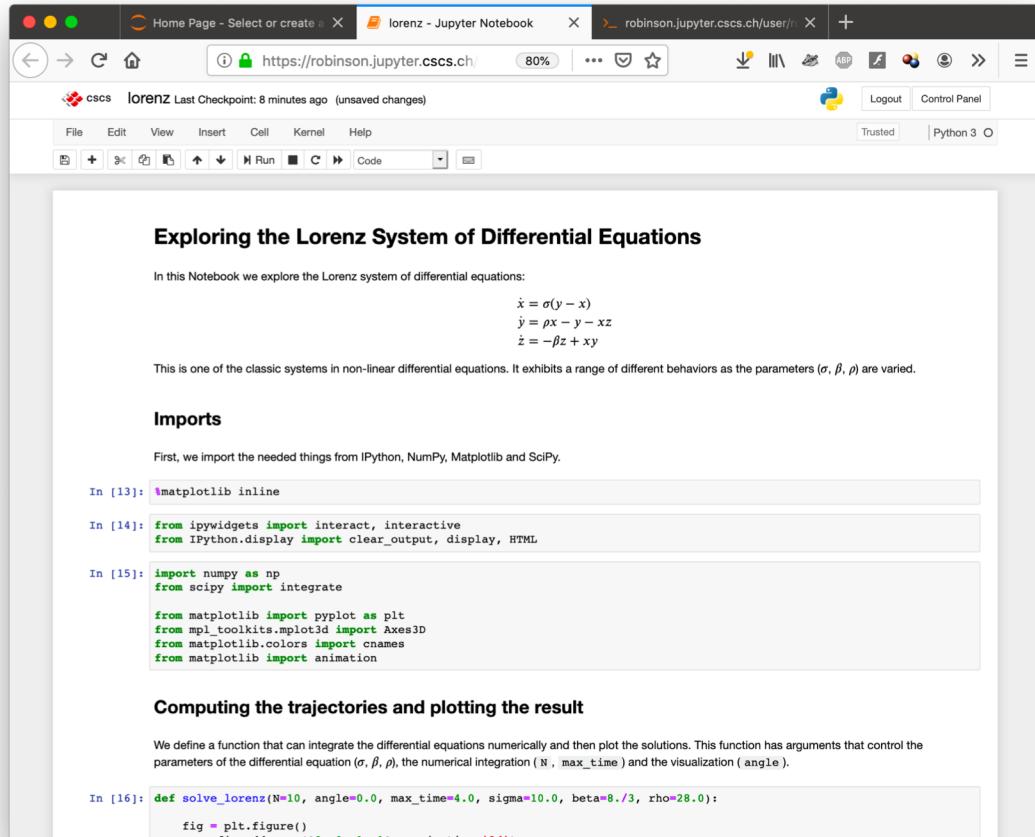
ETHzürich

Evolution from Jupyter Notebook to JupyterLab

JupyterLab

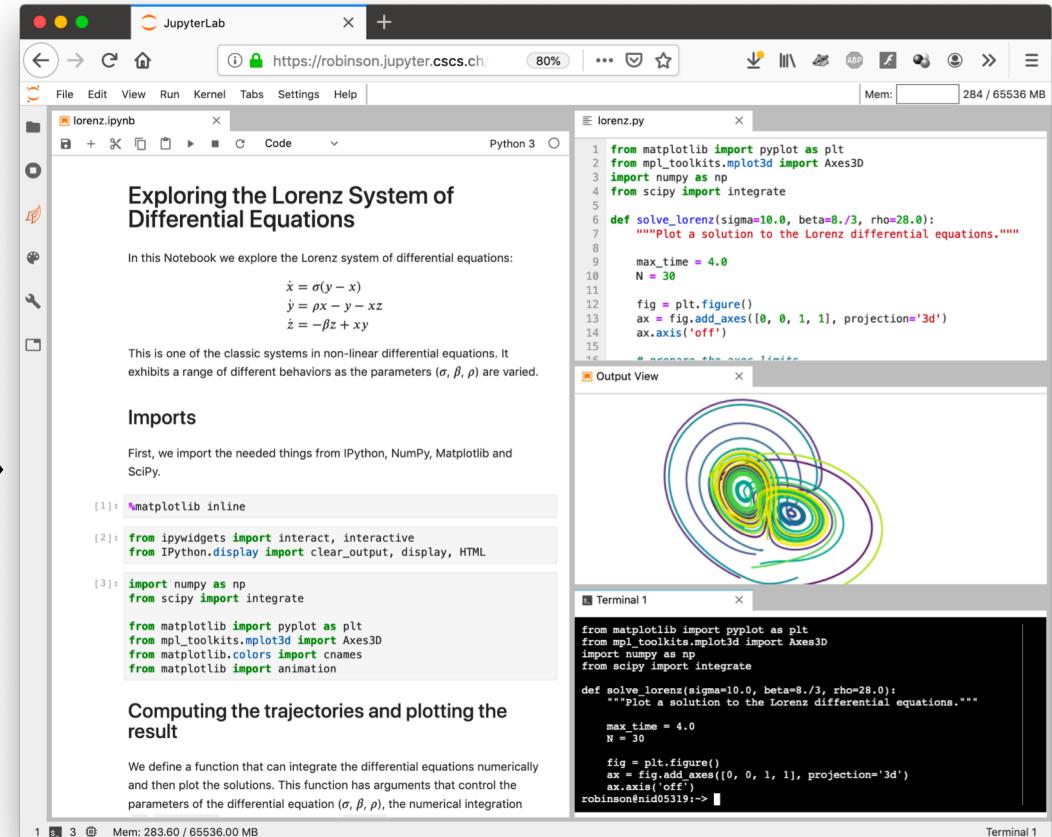
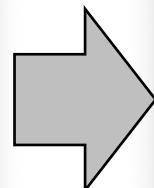
- Next-generation web-based user interface for Jupyter
- Provides higher degree of interaction between notebooks, documents, text editors and other activities (arrange with tabs/splitters)
- Advanced interactive development environment
- Served from same server and uses same notebook document format
- JupyterLab available to the User Lab since March 2019 (version 0.35.2)
- Upgraded to version 1.1.1 in September 2019

Evolution of Jupyter Notebook to JupyterLab



The screenshot shows a Jupyter Notebook interface with the title "Exploring the Lorenz System of Differential Equations". It contains text about the Lorenz system, mathematical equations, imports, and code for solving and plotting the trajectories. A large grey arrow points from this interface to the right.

```
In [13]: %matplotlib inline
In [14]: from ipywidgets import interact, interactive
         from IPython.display import clear_output, display, HTML
In [15]: import numpy as np
         from scipy import integrate
         from matplotlib import pyplot as plt
         from mpl_toolkits.mplot3d import Axes3D
         from matplotlib.colors import cnames
         from matplotlib import animation
In [16]: def solve_lorenz(N=10, angle=0.0, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):
             fig = plt.figure()
             ax = fig.add_axes([0, 0, 1, 1], projection='3d')
```



The screenshot shows a JupyterLab interface with the title "Exploring the Lorenz System of Differential Equations". It includes imports, code for solving and plotting the Lorenz system, and a 3D plot of the resulting trajectories. A terminal window at the bottom shows the command used to run the code.

```
In [1]: %matplotlib inline
In [2]: from ipywidgets import interact, interactive
         from IPython.display import clear_output, display, HTML
In [3]: import numpy as np
         from scipy import integrate
         from matplotlib import pyplot as plt
         from mpl_toolkits.mplot3d import Axes3D
         from matplotlib.colors import cnames
         from matplotlib import animation
In [4]: def solve_lorenz(sigma=10.0, beta=8./3, rho=28.0):
             """Plot a solution to the Lorenz differential equations."""
             max_time = 4.0
             N = 30
             fig = plt.figure()
             ax = fig.add_axes([0, 0, 1, 1], projection='3d')
             ax.axis('off')
             # ... (rest of the function)
In [5]: solve_lorenz()
In [6]:
```

Output View



```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from scipy import integrate
def solve_lorenz(sigma=10.0, beta=8./3, rho=28.0):
    """Plot a solution to the Lorenz differential equations."""
    max_time = 4.0
    N = 30
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')
    # ... (rest of the function)
robinson@nid05339:~>
```

Terminal 1

Expand and collapse cells

The screenshot shows a JupyterLab interface with a notebook titled "lorenz.ipynb". The notebook content is as follows:

Exploring the Lorenz System of Differential Equations

In this Notebook we explore the Lorenz system of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of different behaviors as the parameters (σ , β , ρ) are varied.

Imports

First, we import the needed things from IPython, NumPy, Matplotlib and SciPy.

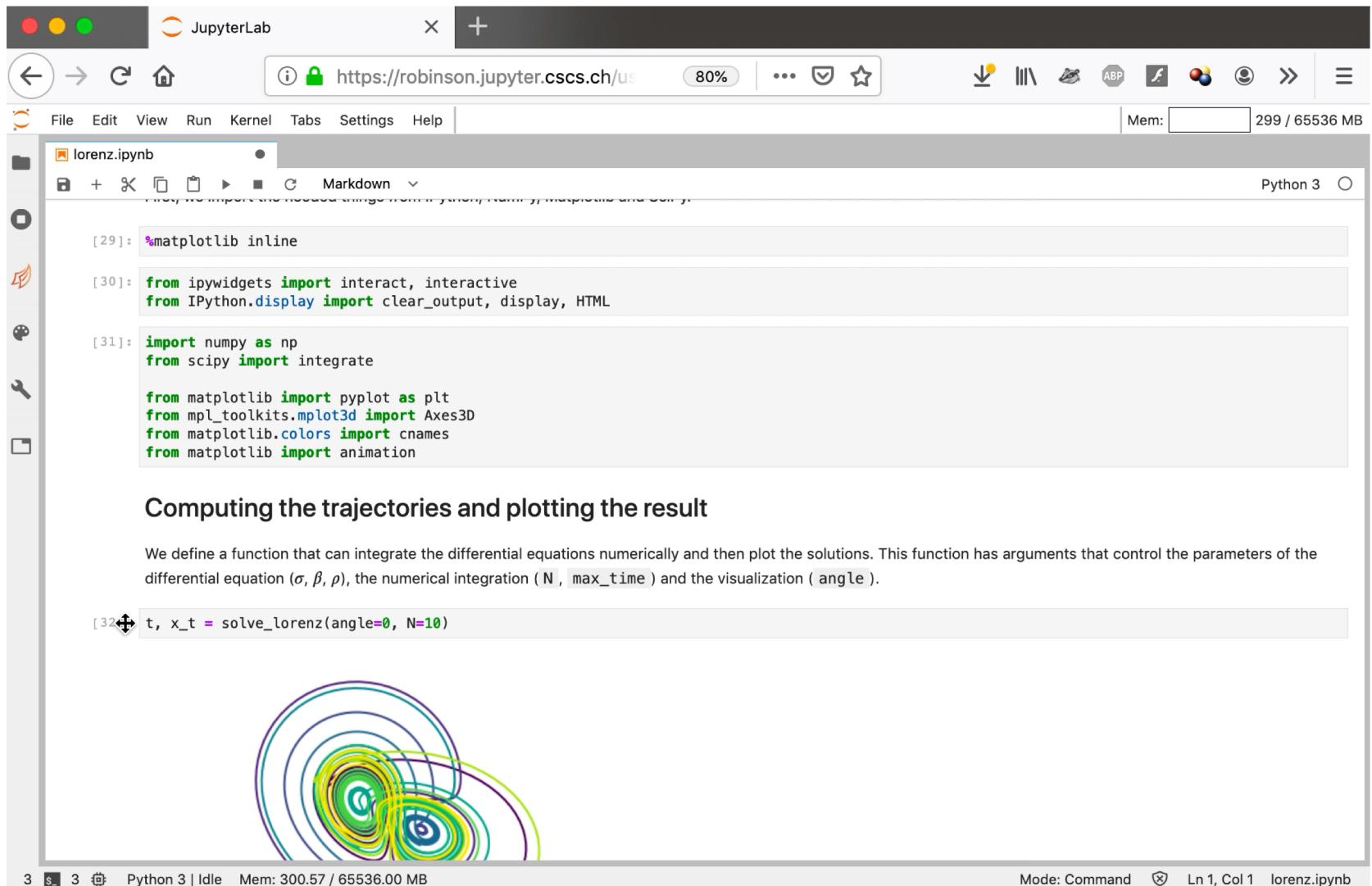
```
[17]: %matplotlib inline
[18]: from ipywidgets import interact, interactive
       from IPython.display import clear_output, display, HTML
[19]: import numpy as np
       from scipy import integrate

       from matplotlib import pyplot as plt
       from mpl_toolkits.mplot3d import Axes3D
       from matplotlib.colors import cnames
       from matplotlib import animation
```

Computing the trajectories and plotting the result

At the bottom of the interface, the status bar shows: 3 s 3 Python 3 | Idle Mem: 299.13 / 65536.00 MB Mode: Command L 1, Col 3 lorenz.ipynb

Drag and drop cells with a notebook



The screenshot shows a JupyterLab interface with a single notebook file named "lorenz.ipynb". The code cell at line 32 contains the command `t, x_t = solve_lorenz(angle=0, N=10)`. Below the code, a 3D plot of the Lorenz attractor is displayed, showing three interlocking trajectories in red, blue, and green. The plot is rendered using the Matplotlib library.

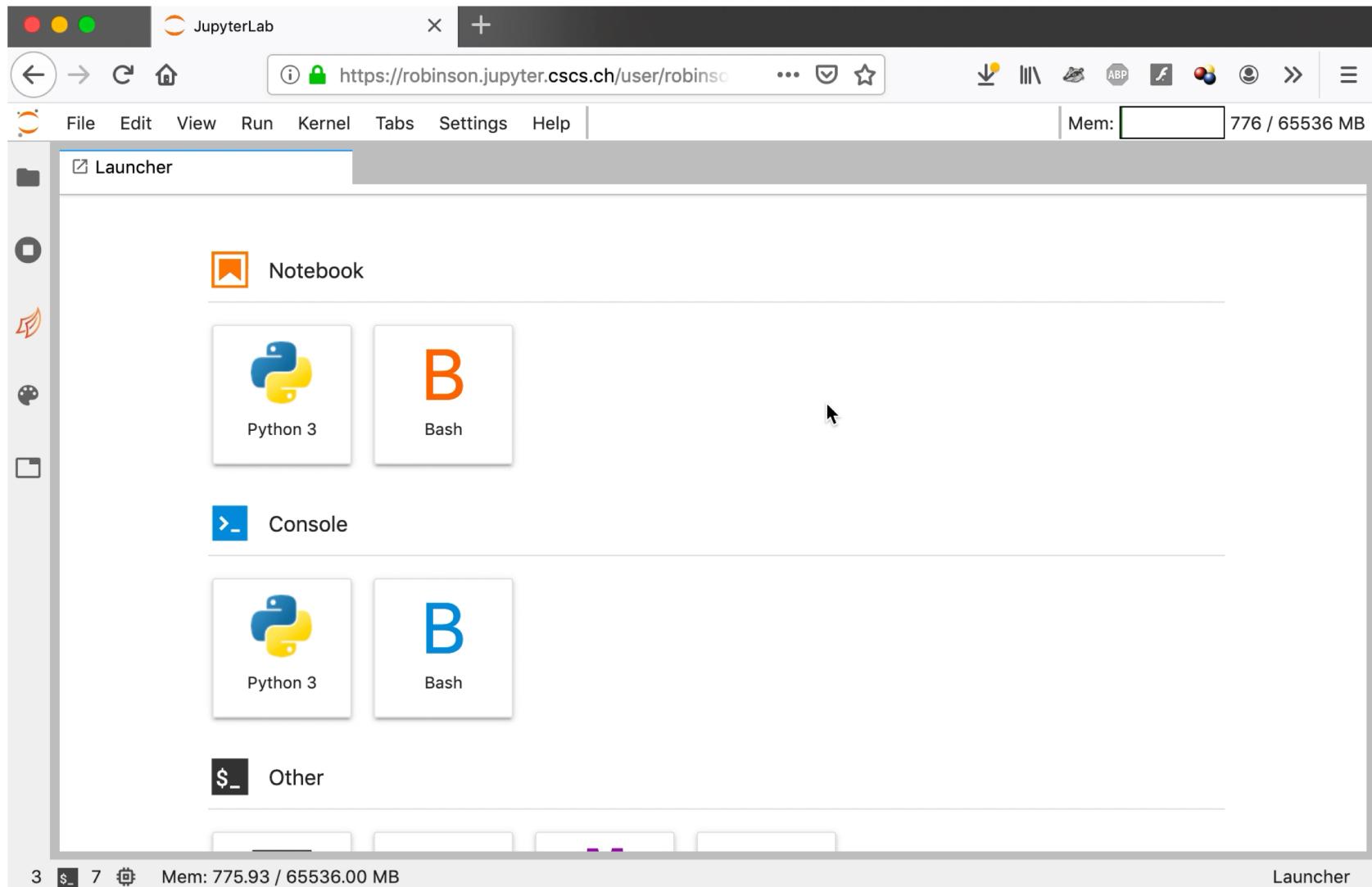
```
%matplotlib inline
from ipywidgets import interact, interactive
from IPython.display import clear_output, display, HTML
import numpy as np
from scipy import integrate
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
```

Computing the trajectories and plotting the result

We define a function that can integrate the differential equations numerically and then plot the solutions. This function has arguments that control the parameters of the differential equation (σ, β, ρ), the numerical integration (`N`, `max_time`) and the visualization (`angle`).

```
t, x_t = solve_lorenz(angle=0, N=10)
```

Tab autocomplete in the text editor



Interactive computing – new output view

The screenshot shows the JupyterLab interface with a notebook titled "lorenz.ipynb". The code cell contains Python code for solving the Lorenz system. The code defines a function to set up a 3D plot, a derivative function for the Lorenz equations, and initializes random starting points. A descriptive text block explains the purpose of the function.

```
from matplotlib.colors import cnames
from matplotlib import animation

def solve_lorenz(N=10, angle=0.0, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):

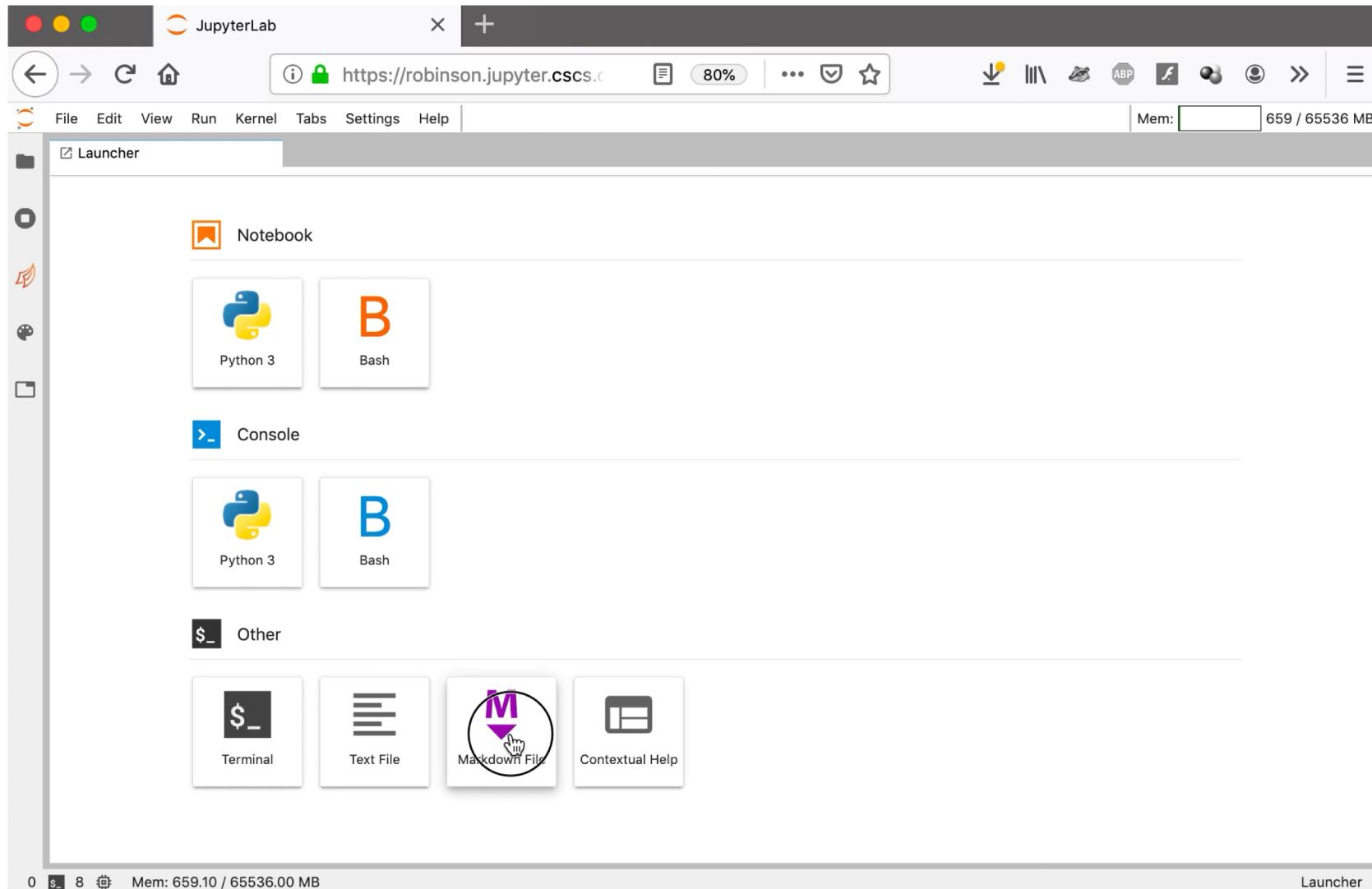
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')

    # prepare the axes limits
    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))

    def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
        """Compute the time-derivative of a Lorenz system."""
        x, y, z = x_y_z
        return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]

    # Choose random starting points, uniformly distributed from -15 to 15
    np.random.seed(1)
    x0 = -15 + 30 * np.random((N, 3))
```

Simplifying code documentation





CSCS

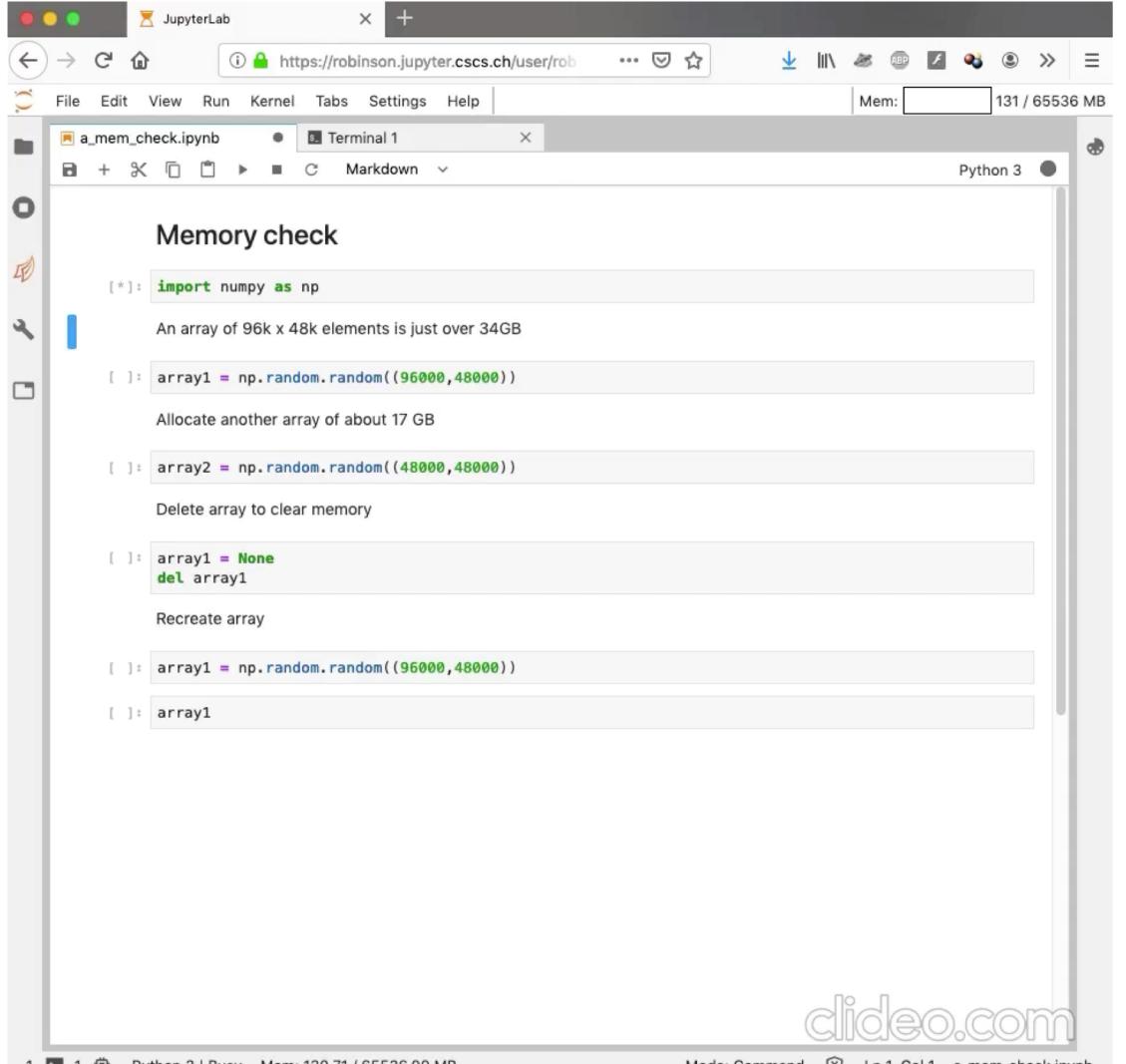
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

New extensions enabled with JupyterLab 1.1.1

JupyterLab system monitor

- JupyterLab extension to display system information
- Provides a frontend for the *nbresuse* package
- Currently limited to memory usage
- CPU usage and other metrics may be added in the future



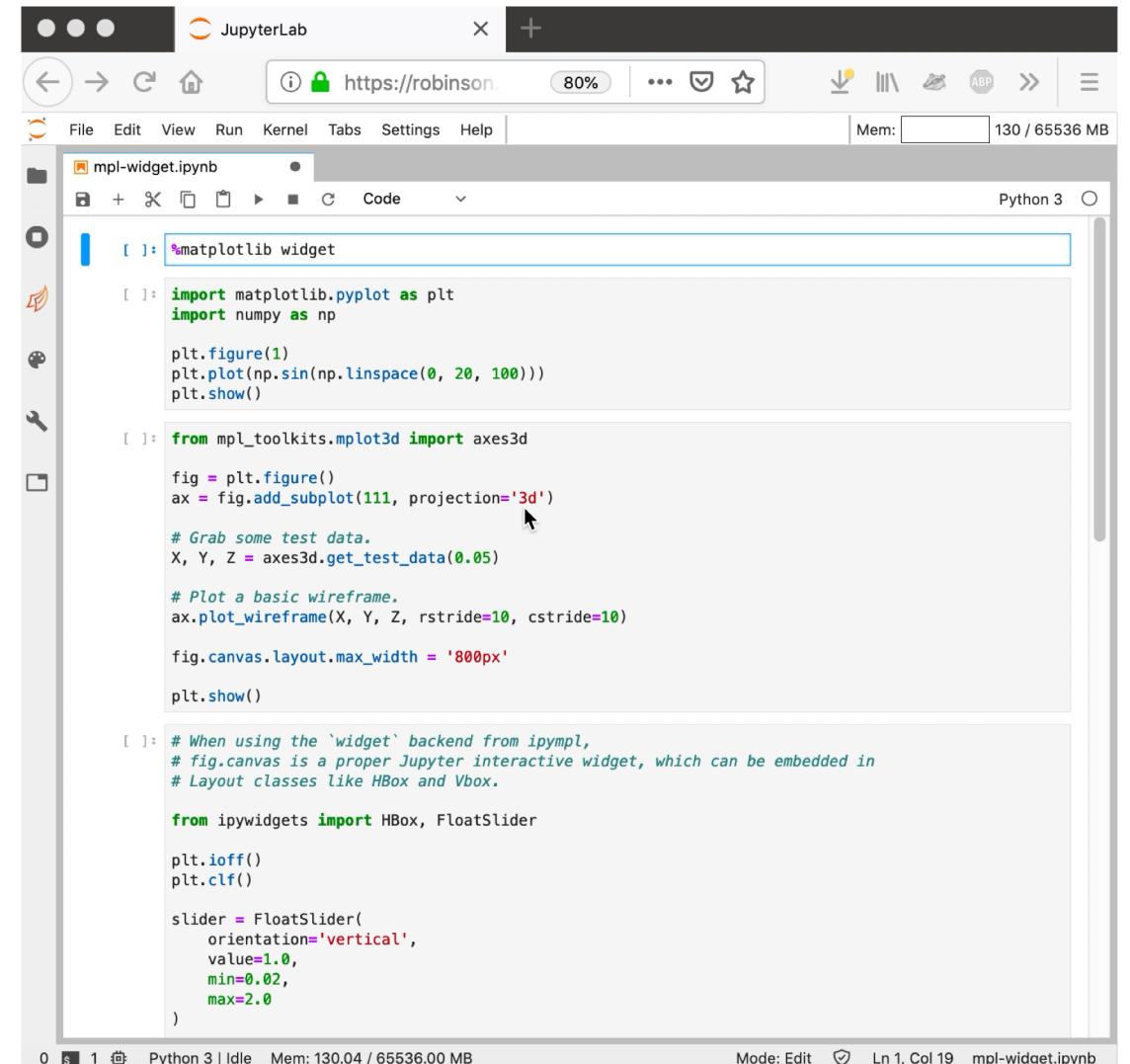
The screenshot shows a JupyterLab interface with a single notebook tab titled "a_mem_check.ipynb". The notebook contains the following code:

```
[*]: import numpy as np
An array of 96k x 48k elements is just over 34GB
[ ]: array1 = np.random.random((96000,48000))
Allocate another array of about 17 GB
[ ]: array2 = np.random.random((48000,48000))
Delete array to clear memory
[ ]: array1 = None
del array1
Recreate array
[ ]: array1 = np.random.random((96000,48000))
[ ]: array1
```

The status bar at the bottom indicates "Python 3 | Busy" and "Mem: 130.71 / 65536.00 MB". A watermark for "clideo.com" is visible in the bottom right corner.

jupyter-matplotlib widgets

- Matplotlib is a commonly used 2D plotting library for producing figures in hardcopy formats, and interactive environments across platforms
- Jupyter-matplotlib enables the interactive features of matplotlib in JupyterLab
- The figure canvas element is a Jupyter interactive widget that can be positioned in interactive widget layouts
- `%matplotlib widget`



The screenshot shows a JupyterLab interface with a code cell containing Python code. The code uses the `%matplotlib widget` magic command to enable interactive plotting. It imports matplotlib.pyplot and numpy, creates a 2D plot of a sine wave, and then imports mpl_toolkits.mplot3d to create a 3D wireframe plot. The code also demonstrates how to use ipywidgets' HBox and FloatSlider to create a vertical slider for the 3D plot's canvas width.

```
%%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np

plt.figure(1)
plt.plot(np.sin(np.linspace(0, 20, 100)))
plt.show()

from mpl_toolkits.mplot3d import axes3d

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Grab some test data.
X, Y, Z = axes3d.get_test_data(0.05)

# Plot a basic wireframe.
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)

fig.canvas.layout.max_width = '800px'

plt.show()

# When using the 'widget' backend from ipympl,
# fig.canvas is a proper Jupyter interactive widget, which can be embedded in
# Layout classes like HBox and VBox.

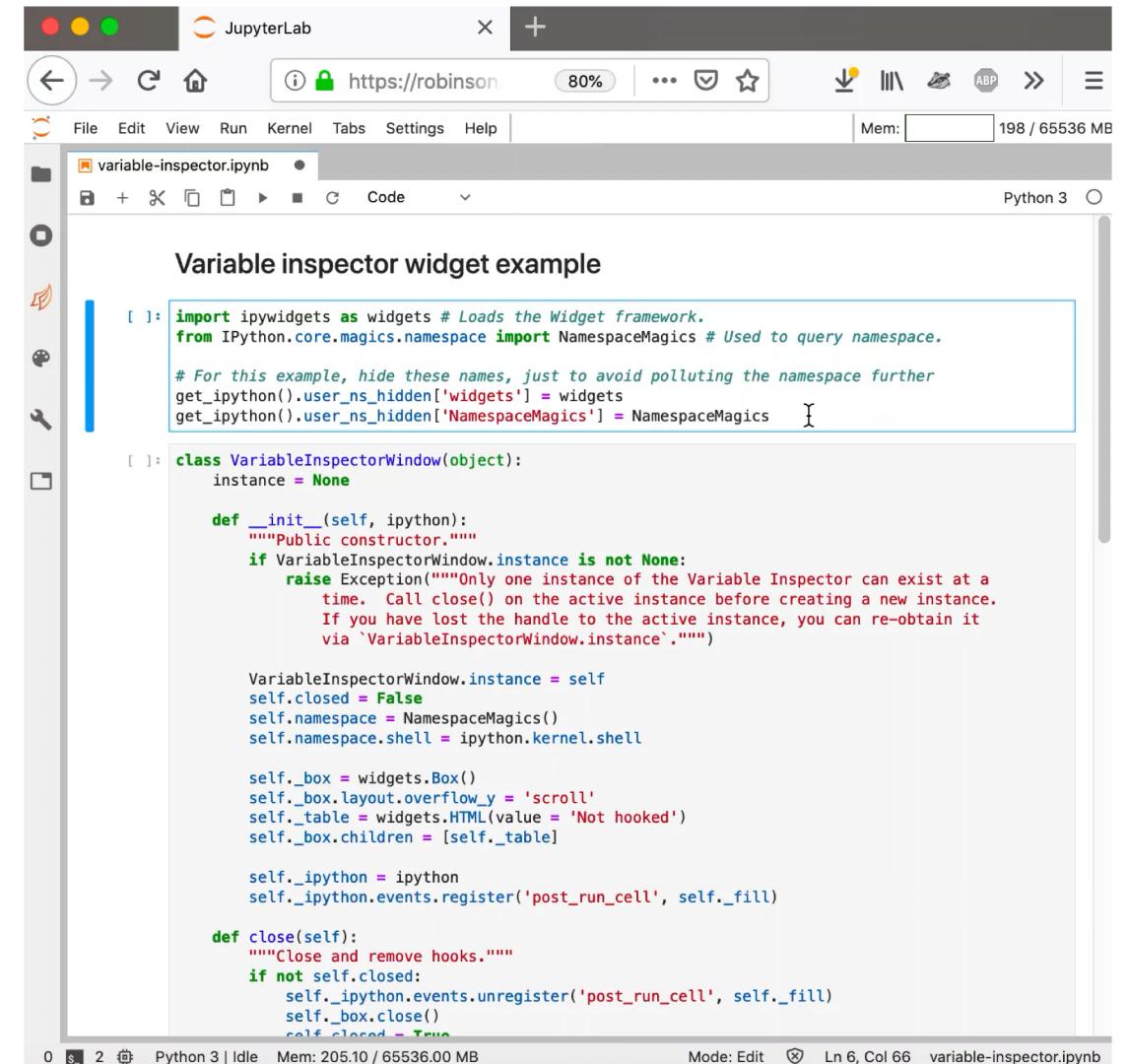
from ipywidgets import HBox, FloatSlider

plt.ioff()
plt.clf()

slider = FloatSlider(
    orientation='vertical',
    value=1.0,
    min=0.02,
    max=2.0
)
```

ipywidgets

- Ipywidgets are interactive HTML widgets for Jupyter notebooks and the IPython kernel
- Core widgets
 - Sliders
 - Progress bars
 - Text boxes
 - Toggle buttons and checkboxes
 - Display areas
- Besides the widgets already provided with the library, the framework can be extended with custom widget libraries



The screenshot shows a JupyterLab interface with a code editor window titled "variable-inspector.ipynb". The code implements a "Variable inspector widget example". It starts by importing ipywidgets and NamespaceMagics, then hides the 'widgets' and 'NamespaceMagics' names from the user namespace. It defines a class "VariableInspectorWindow" that checks if an instance already exists and raises an exception if so. It then initializes a Box widget with a scrollable table containing the message "Not hooked". It registers a "post_run_cell" event hook to fill the table with variable data. Finally, it defines a close method to remove the hook and close the box. The code editor has syntax highlighting for Python and uses a light theme.

```
import ipywidgets as widgets # Loads the Widget framework.
from IPython.core.magic.namespace import NamespaceMagics # Used to query namespace.

# For this example, hide these names, just to avoid polluting the namespace further
get_ipython().user_ns_hidden['widgets'] = widgets
get_ipython().user_ns_hidden['NamespaceMagics'] = NamespaceMagics

class VariableInspectorWindow(object):
    instance = None

    def __init__(self, ipython):
        """Public constructor."""
        if VariableInspectorWindow.instance is not None:
            raise Exception("""Only one instance of the Variable Inspector can exist at a
                           time. Call close() on the active instance before creating a new instance.
                           If you have lost the handle to the active instance, you can re-obtain it
                           via `VariableInspectorWindow.instance`.""")
        VariableInspectorWindow.instance = self
        self.closed = False
        self.namespace = NamespaceMagics()
        self.namespace.shell = ipython.kernel.shell

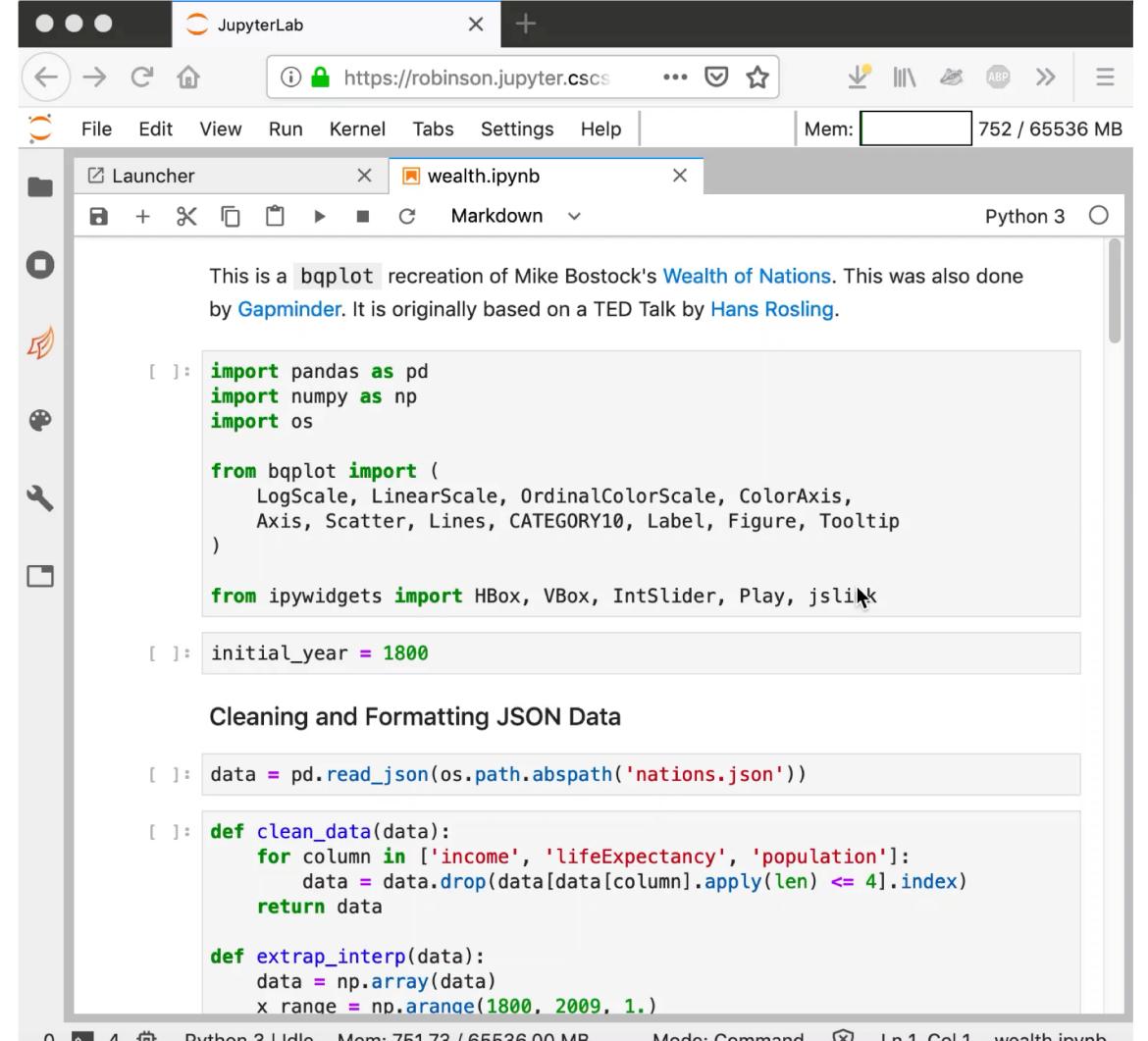
        self._box = widgets.Box()
        self._box.layout.overflow_y = 'scroll'
        self._table = widgets.HTML(value = 'Not hooked')
        self._box.children = [self._table]

        self._ipython = ipython
        self._ipython.events.register('post_run_cell', self._fill)

    def close(self):
        """Close and remove hooks."""
        if not self.closed:
            self._ipython.events.unregister('post_run_cell', self._fill)
            self._box.close()
            self.closed = True
```

bqplot

- 2D interactive plotting framework for the Jupyter notebook
- In bqplot, every component of a plot is an interactive ipython widget – this means after a plot is drawn you can change almost any aspect of it
- Offers a bridge between Python and d3.js, allowing the ability to build complex interactive GUIs
- Two APIs:
 - Simple “matplotlib inspired” pyplot
 - More verbose API which exposes every element of a plot individually



This is a bqplot recreation of Mike Bostock's [Wealth of Nations](#). This was also done by [Gapminder](#). It is originally based on a TED Talk by Hans Rosling.

```
[ ]: import pandas as pd
import numpy as np
import os

from bqplot import (
    LogScale, LinearScale, OrdinalColorScale, ColorAxis,
    Axis, Scatter, Lines, CATEGORY10, Label, Figure, Tooltip
)

from ipywidgets import HBox, VBox, IntSlider, Play, jslink

[ ]: initial_year = 1800

Cleaning and Formatting JSON Data

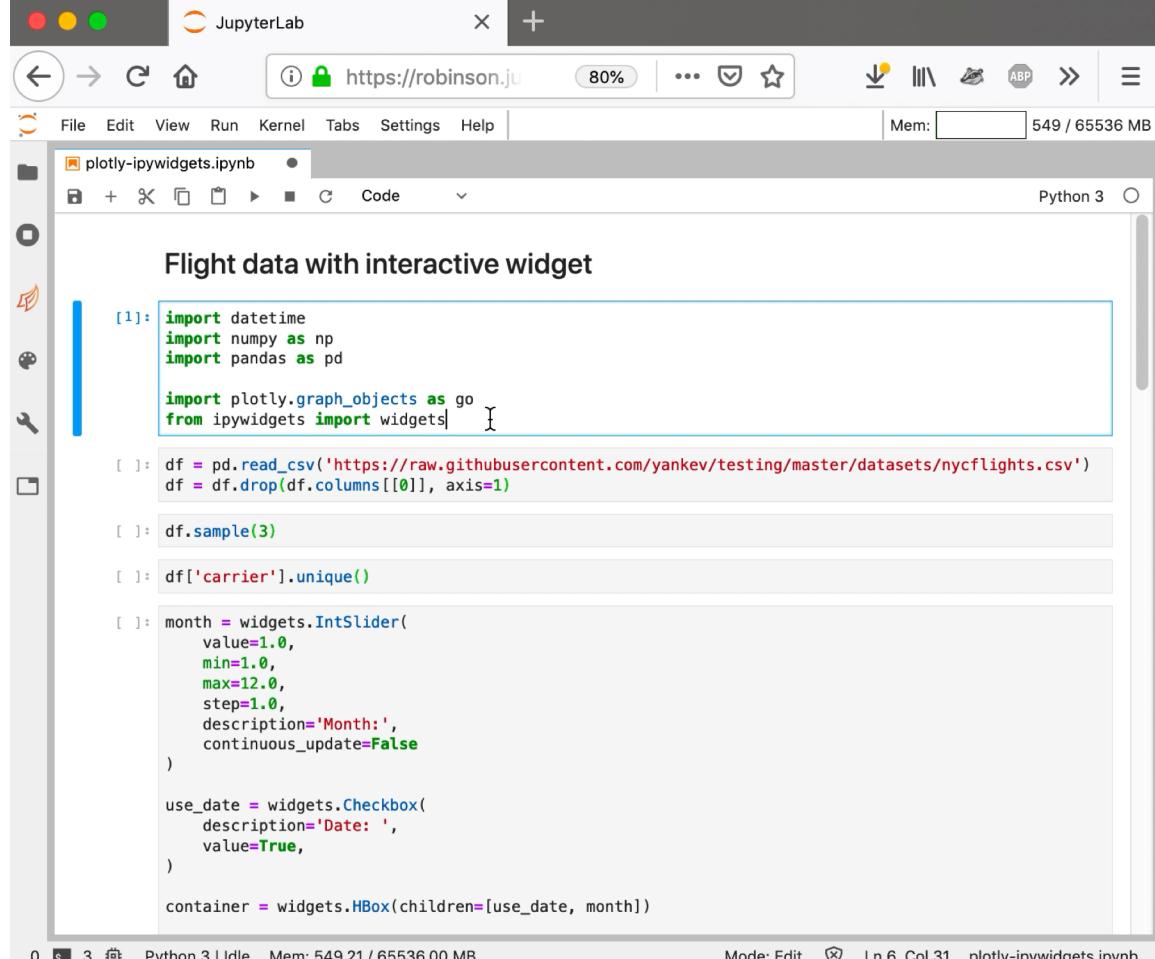
[ ]: data = pd.read_json(os.path.abspath('nations.json'))

[ ]: def clean_data(data):
    for column in ['income', 'lifeExpectancy', 'population']:
        data = data.drop(data[data[column].apply(len) <= 4].index)
    return data

def extrap_interp(data):
    data = np.array(data)
    x_range = np.arange(1800, 2009, 1.)
```

plotly widgets / jupyterlab-plotly

- plotly.py is an interactive graphing library for Python
- Over 30 chart types (scientific charts, 3D graphs, statistical charts, SVG maps, financial charts,...)
- Plotly 3.0.0 introduced a new Jupyter widget class for notebook and JupyterLab:
`plotly.graph_objs.FigureWidget`



The screenshot shows a JupyterLab interface with a code cell titled "Flight data with interactive widget". The code imports datetime, numpy, pandas, and the required Plotly modules. It reads a CSV file of flight data, samples three rows, and finds unique carriers. It then creates an IntSlider for the month (1 to 12) and a Checkbox for using date information. Finally, it creates an HBox container for the date checkbox and the month slider.

```
import datetime
import numpy as np
import pandas as pd

import plotly.graph_objects as go
from ipywidgets import widgets

df = pd.read_csv('https://raw.githubusercontent.com/yankev/testing/master/datasets/nycflights.csv')
df = df.drop(df.columns[[0]], axis=1)

df.sample(3)

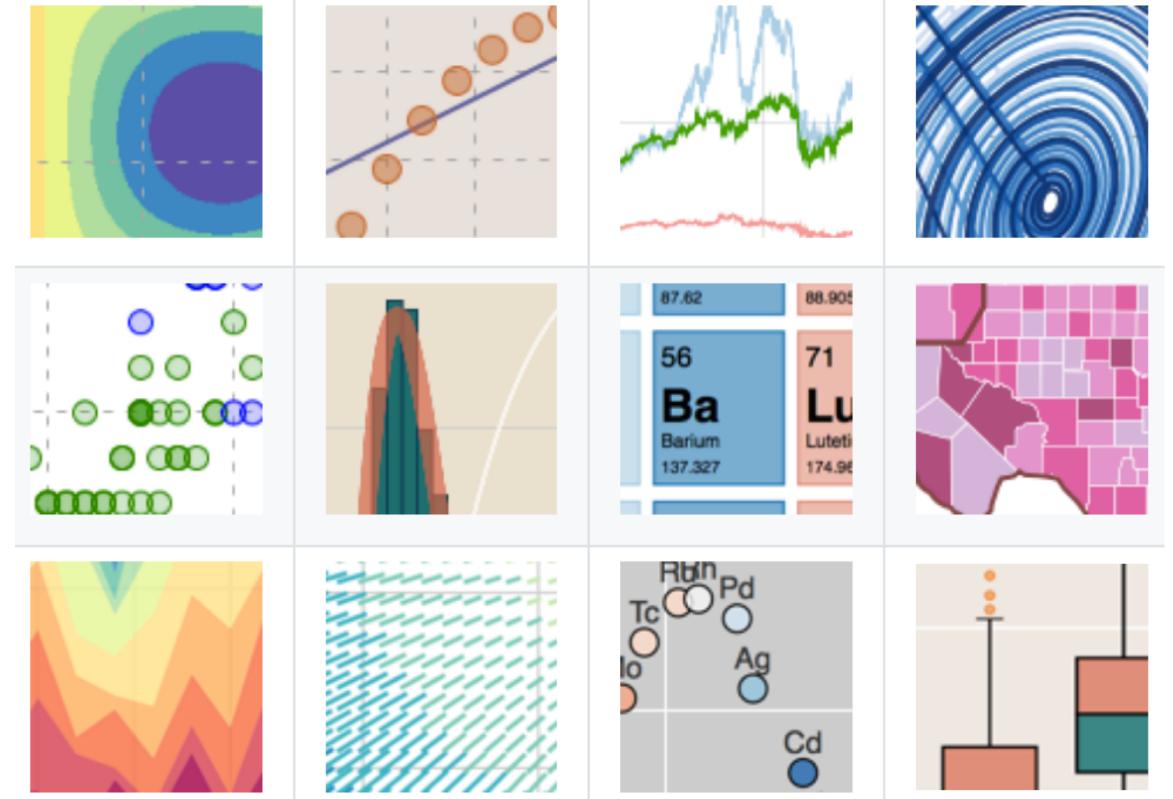
df['carrier'].unique()

month = widgets.IntSlider(
    value=1.0,
    min=1.0,
    max=12.0,
    step=1.0,
    description='Month:',
    continuous_update=False
)

use_date = widgets.Checkbox(
    description='Date: ',
    value=True,
)
container = widgets.HBox(children=[use_date, month])
```

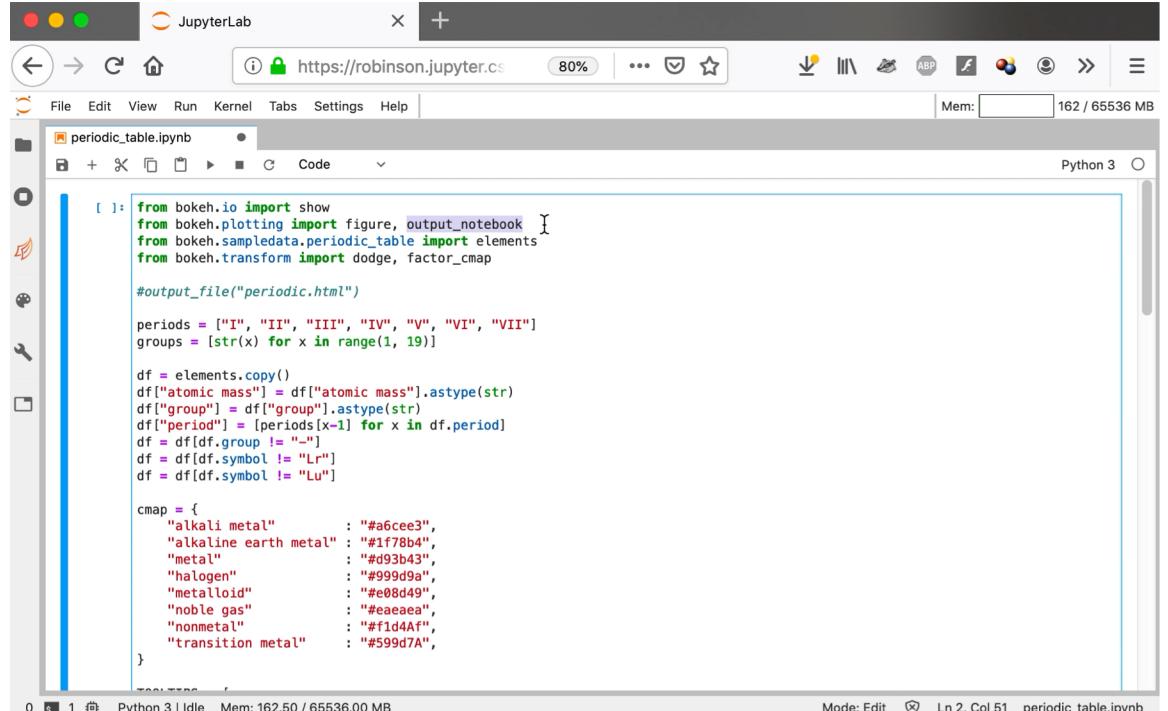
Jupyterlab_bokeh

- Bokeh is an interactive visualization library for Python that enables visual presentation of data in web browser
- With Bokeh, you can create interactive plots, dashboards, and data applications



jupyterlab_bokeh

- Bokeh is an interactive visualization library for Python that enables visual presentation of data in web browser
- With Bokeh, you can create interactive plots, dashboards, and data applications
- Jupyterlab_bokeh is a server extension for rendering this content in JupyterLab



The screenshot shows a JupyterLab interface with a code cell containing Python code to generate a periodic table plot using Bokeh. The code imports necessary modules, reads a periodic table dataset, and applies styling to create a grouped bar chart where each element's atomic mass is represented by a colored bar corresponding to its group and period.

```
[ ]: from bokeh.io import show
from bokeh.plotting import figure, output_notebook
from bokeh.sampledata.periodic_table import elements
from bokeh.transform import dodge, factor_cmap

#output_file("periodic.html")

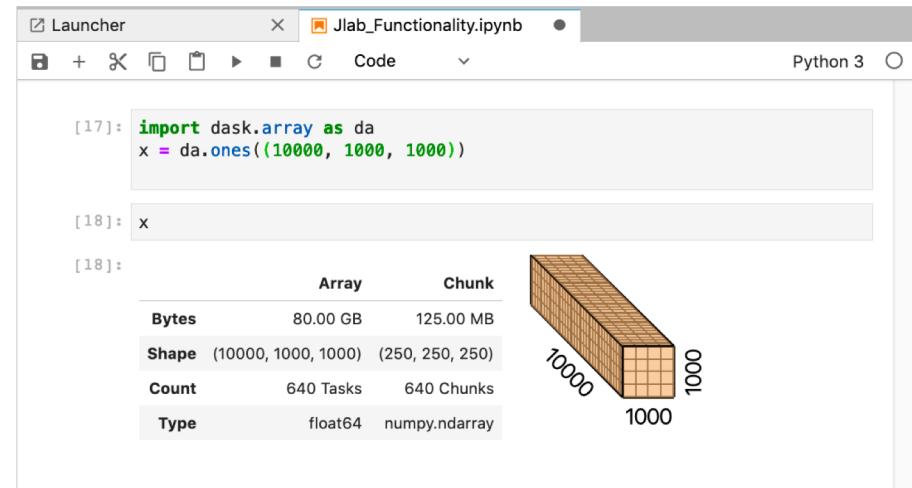
periods = ["I", "II", "III", "IV", "V", "VI", "VII"]
groups = [str(x) for x in range(1, 19)]

df = elements.copy()
df["atomic mass"] = df["atomic mass"].astype(str)
df[["group"] = df[["group"]].astype(str)
df[["period"] = periods[x-1] for x in df.period]
df = df[df.group != "-"]
df = df[df.symbol != "Lr"]
df = df[df.symbol != "Lu"]

cmap = {
    "alkali metal" : "#a6cee3",
    "alkaline earth metal" : "#1f78b4",
    "metal" : "#d93b43",
    "halogen" : "#999d9a",
    "metalloid" : "#e08d49",
    "noble gas" : "#eaeaea",
    "nonmetal" : "#f1d4af",
    "transition metal" : "#599d7a",
}
```

Dask and dask JupyterLab extension

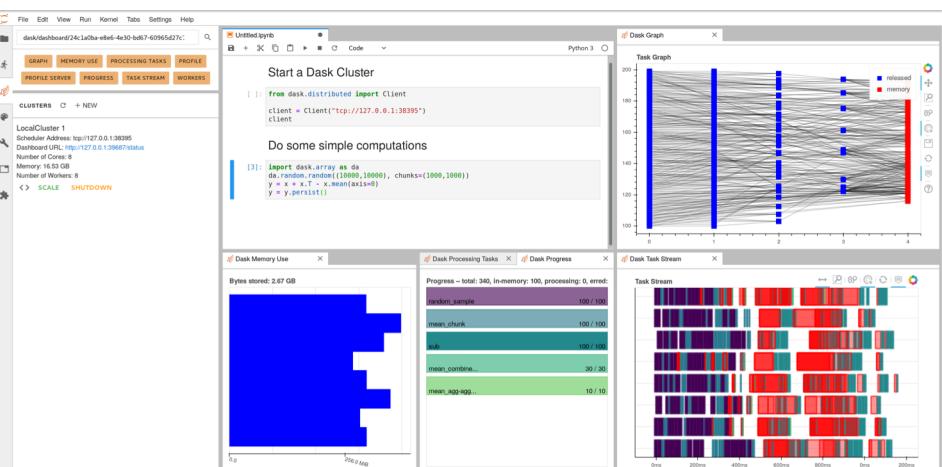
- Dask is a flexible library for parallel computing in Python
- Extends interfaces like *NumPy*, *Pandas*, or *Python iterators* to larger-than-memory / distributed environments
- These parallel collections run on top of dynamic task schedulers



A screenshot of a JupyterLab interface. In the code editor, cell [17] contains the code `import dask.array as da` and `x = da.ones(10000, 1000, 1000)`. Cell [18] contains the variable `x`. Below the code editor, a table provides details about the array:

	Array	Chunk
Bytes	80.00 GB	125.00 MB
Shape	(10000, 1000, 1000)	(250, 250, 250)
Count	640 Tasks	640 Chunks
Type	float64	numpy.ndarray

To the right of the table is a 3D bar chart representing the 10000x1000x1000 array.



Dask JupyterLab extension

The screenshot shows a JupyterLab interface with two tabs open, both titled "JupyterLab". The left tab is active and displays a notebook cell for "user_day_2019_dask_distri". The cell contains the following content:

```
[2]: from dask import bag as db
```

If you have Bokeh installed, you should have access to the Dask Dashboard. The dashboard can give an idea of what's happening across the dask distributed cluster.

```
[3]: email = db.read_text('/scratch/sn3000/robinson/enron_mail/mailbag.*.json.gz')
```

This call produced one bag element for each file. We want to split each of these elements into words (`.split`), and then combine them (`.flatten`). We will also save this in memory in case we want to do further analysis:

```
[4]: emailwords = email.str.split().flatten()
words = client.persist(emailwords)
```

Next let's find the 20 most frequent words, and tell the Dask scheduler to start the work.

```
[5]: wordcount = words.frequencies().topk(20, lambda x: x[1])
wc_future = client.compute(wordcount)
progress(wc_future)
```

The status bar at the bottom indicates "Python 3 | Idle" and "Mem: 215.31 / 65536.00 MB".



CSCS

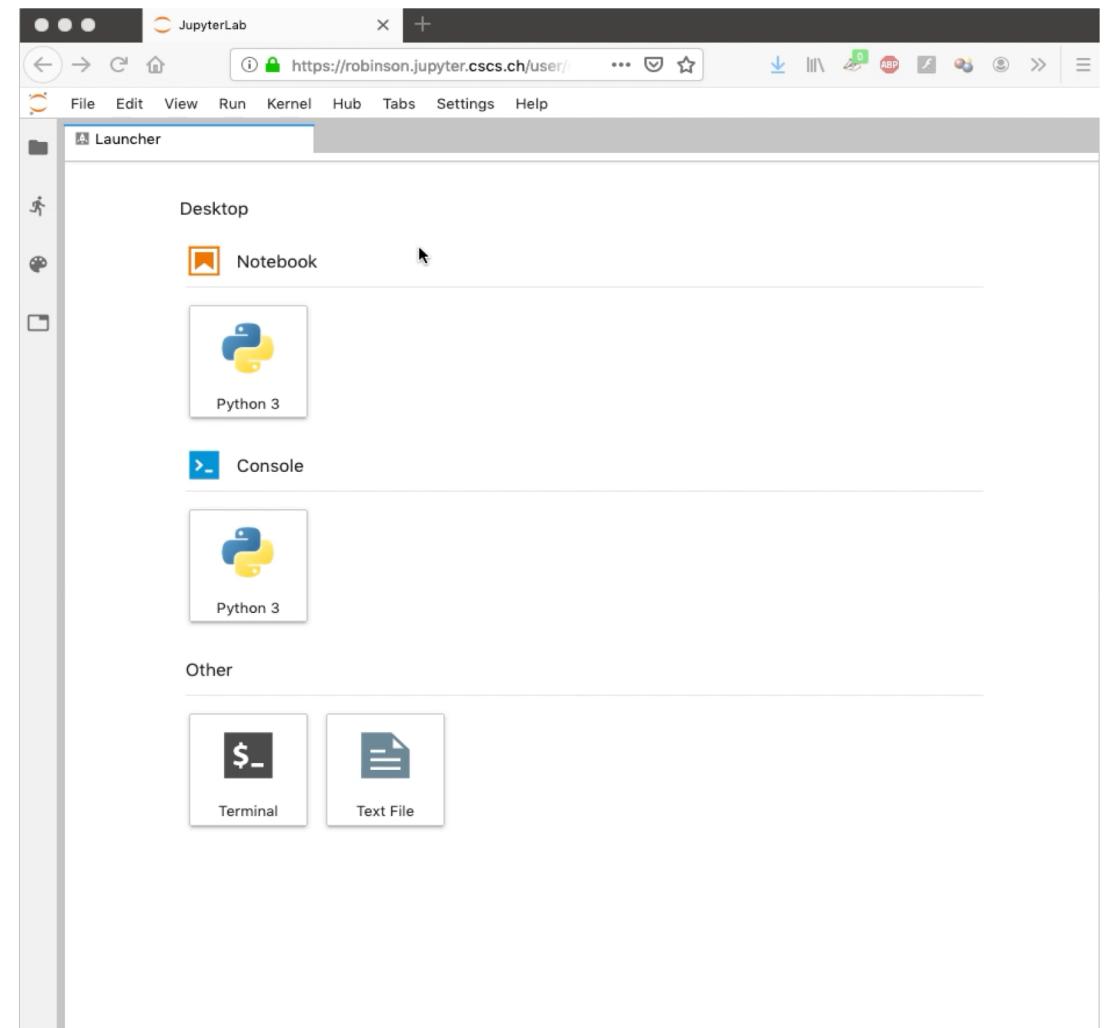
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich

Changes and New Functionality in JupyterLab 1.0+

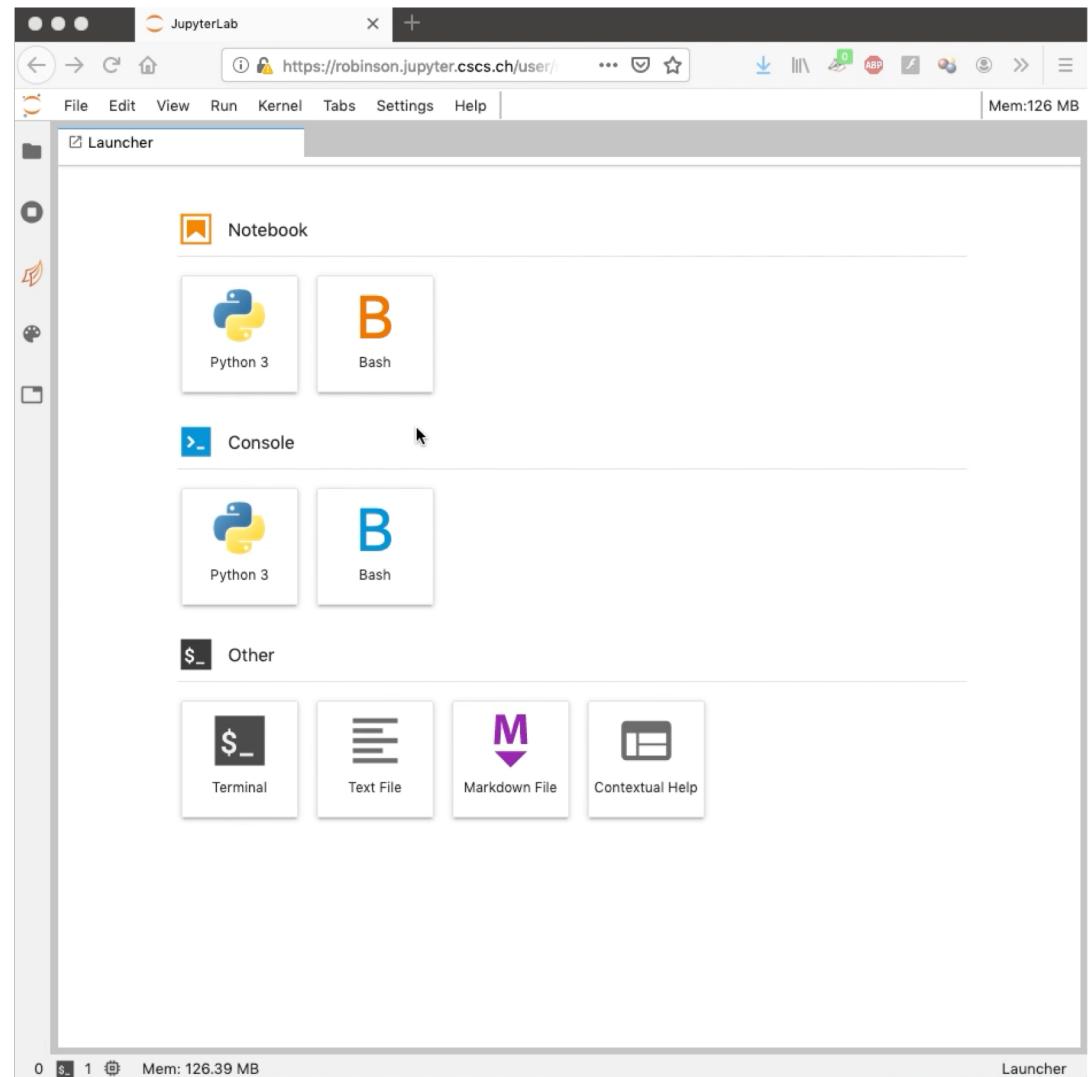
JupyterHub integration

- Previously, to stop your server you needed to use Hub > Control Panel



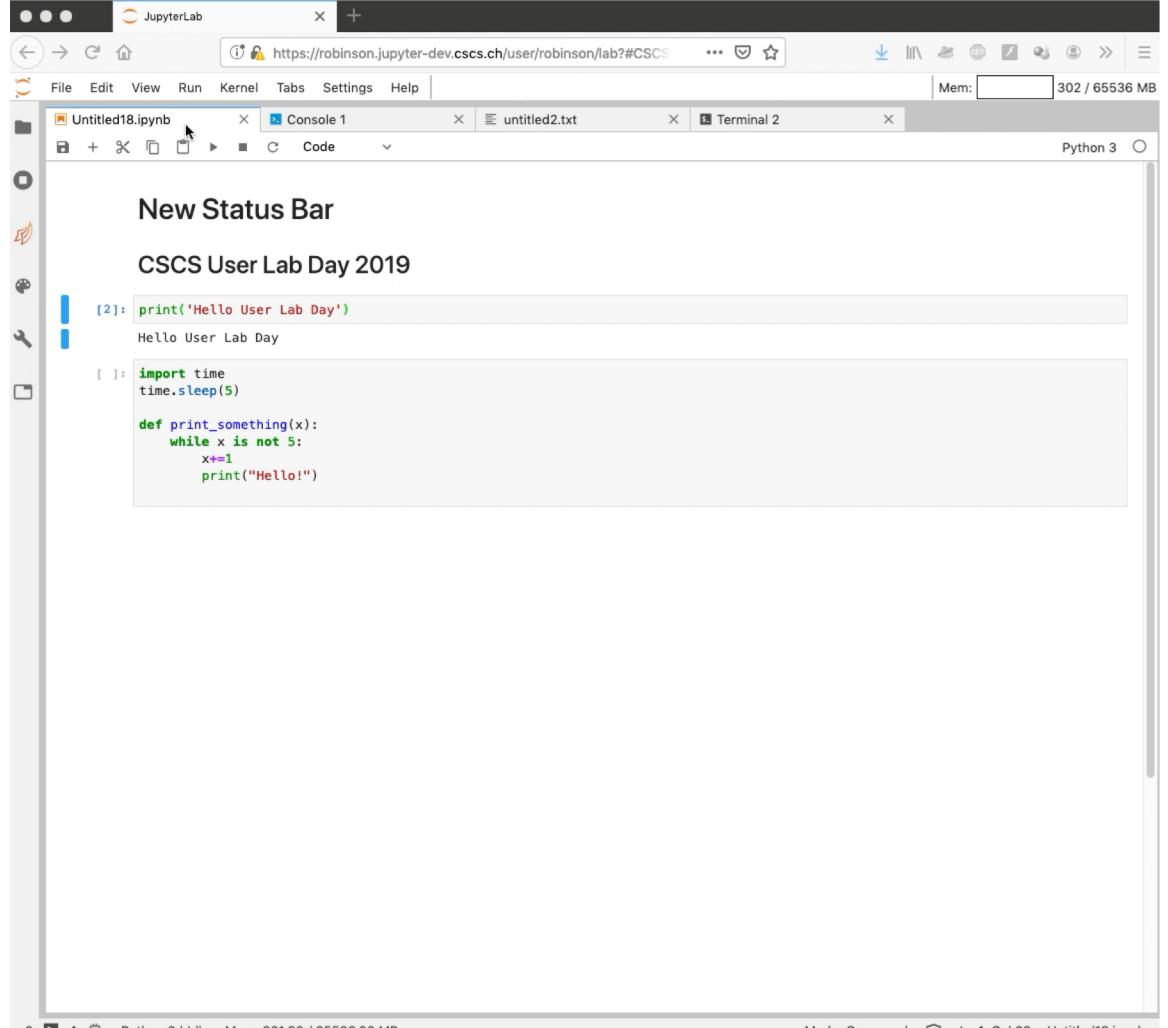
JupyterHub integration

- Previously, to stop your server you needed to use Hub > Control Panel
- Now, the JupyterHub extension is included in the core JupyterLab distribution
- To stop your sever: File > Control Panel
- Note: File > Logout will not stop your server!



Status bar

- Jupyterlab-statusbar provides a generic status bar to showcase the various states of JupyterLab
- Different components render depending on the active context: notebook, console, file editor, and terminal
- Can be used by other extensions to add custom elements into the status bar



The screenshot shows the JupyterLab interface with the "New Status Bar" extension installed. The top navigation bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The status bar at the bottom displays "Mode: Command" and "Ln 1, Col 28 Untitled18.ipynb". The main workspace shows a notebook titled "Untitled18.ipynb" containing the following code:

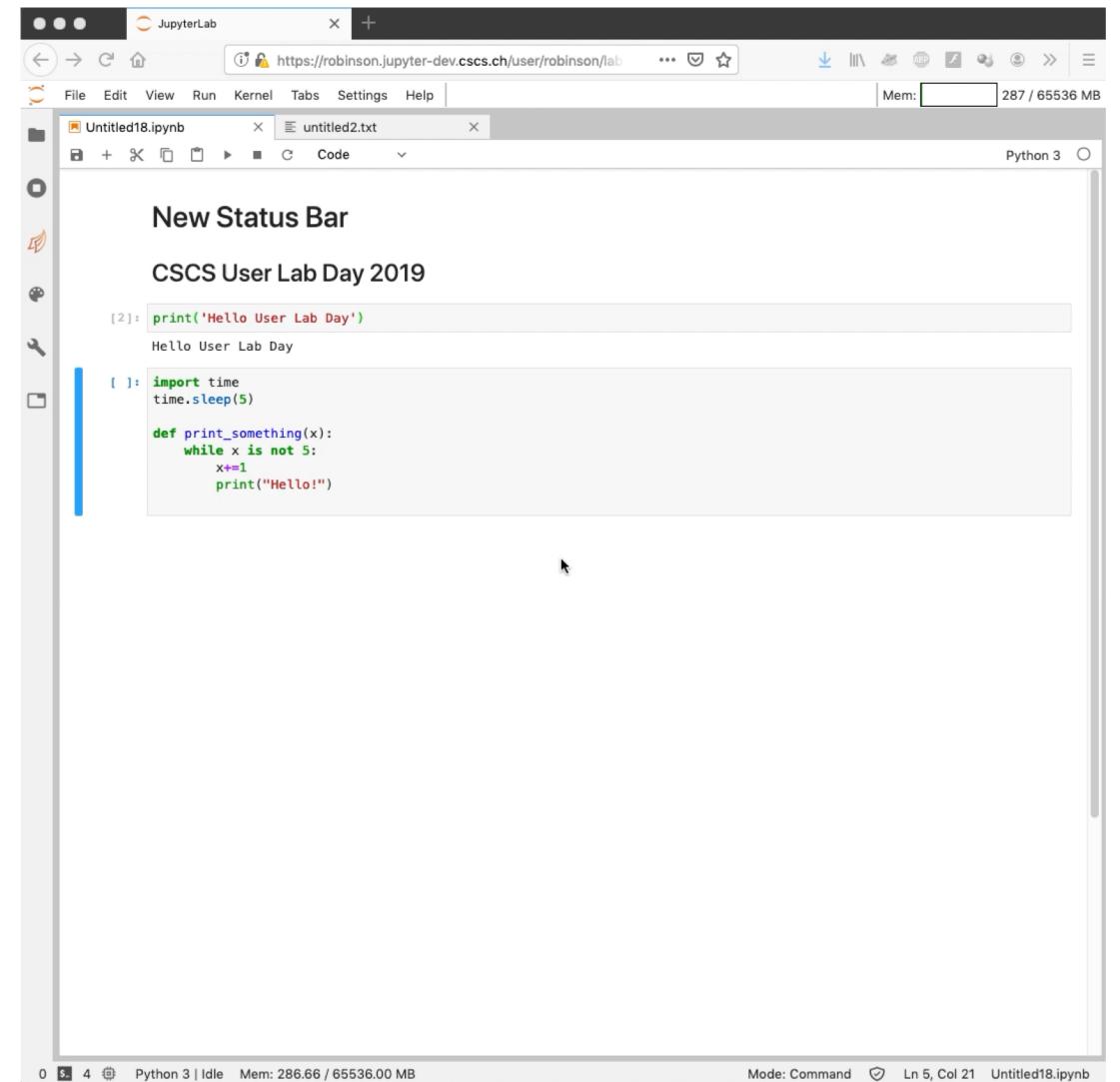
```
[2]: print('Hello User Lab Day')
Hello User Lab Day

[ ]: import time
time.sleep(5)

def print_something(x):
    while x is not 5:
        x+=1
        print("Hello!")
```

Status bar (2)

- Jupyterlab-statusbar provides a generic status bar to showcase the various states of JupyterLab
- Different components render depending on the active context: notebook, console, file editor, and terminal
- Can be used by other extensions to add custom elements into the status bar



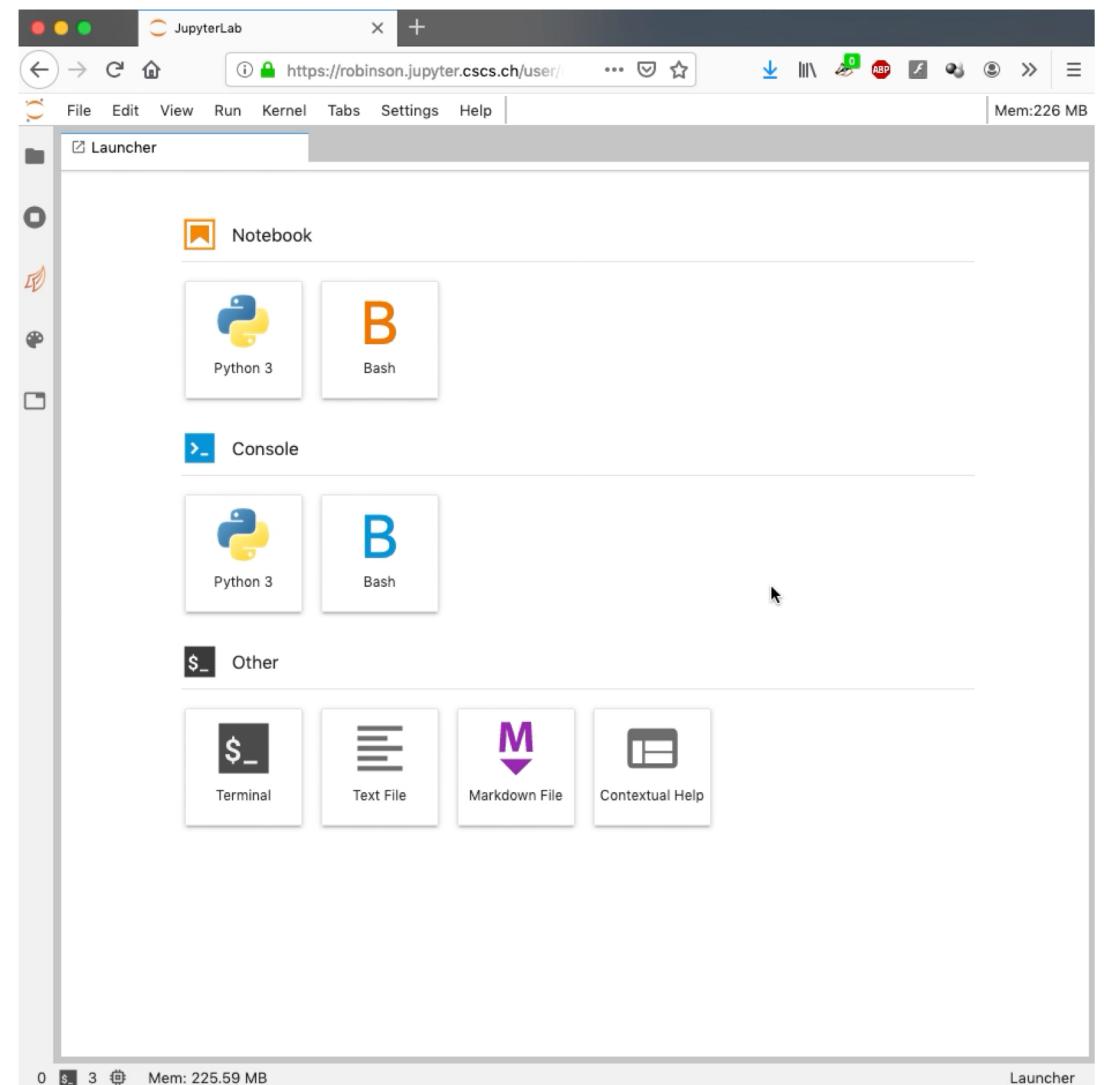
The screenshot shows a JupyterLab interface with the "New Status Bar" extension active. The top navigation bar includes "File", "Edit", "View", "Run", "Kernel", "Tabs", "Settings", and "Help". The status bar at the bottom displays "Mode: Command", "Ln 5, Col 21", and "Untitled18.ipynb". The main workspace shows two tabs: "Untitled18.ipynb" and "untitled2.txt". The notebook tab contains the following code:

```
[2]: print('Hello User Lab Day')
Hello User Lab Day
[1]: import time
time.sleep(5)

def print_something(x):
    while x is not 5:
        x+=1
        print("Hello!")
```

New kernels

- Bash kernel now available in the launcher
- IJulia kernel *coming soon...*



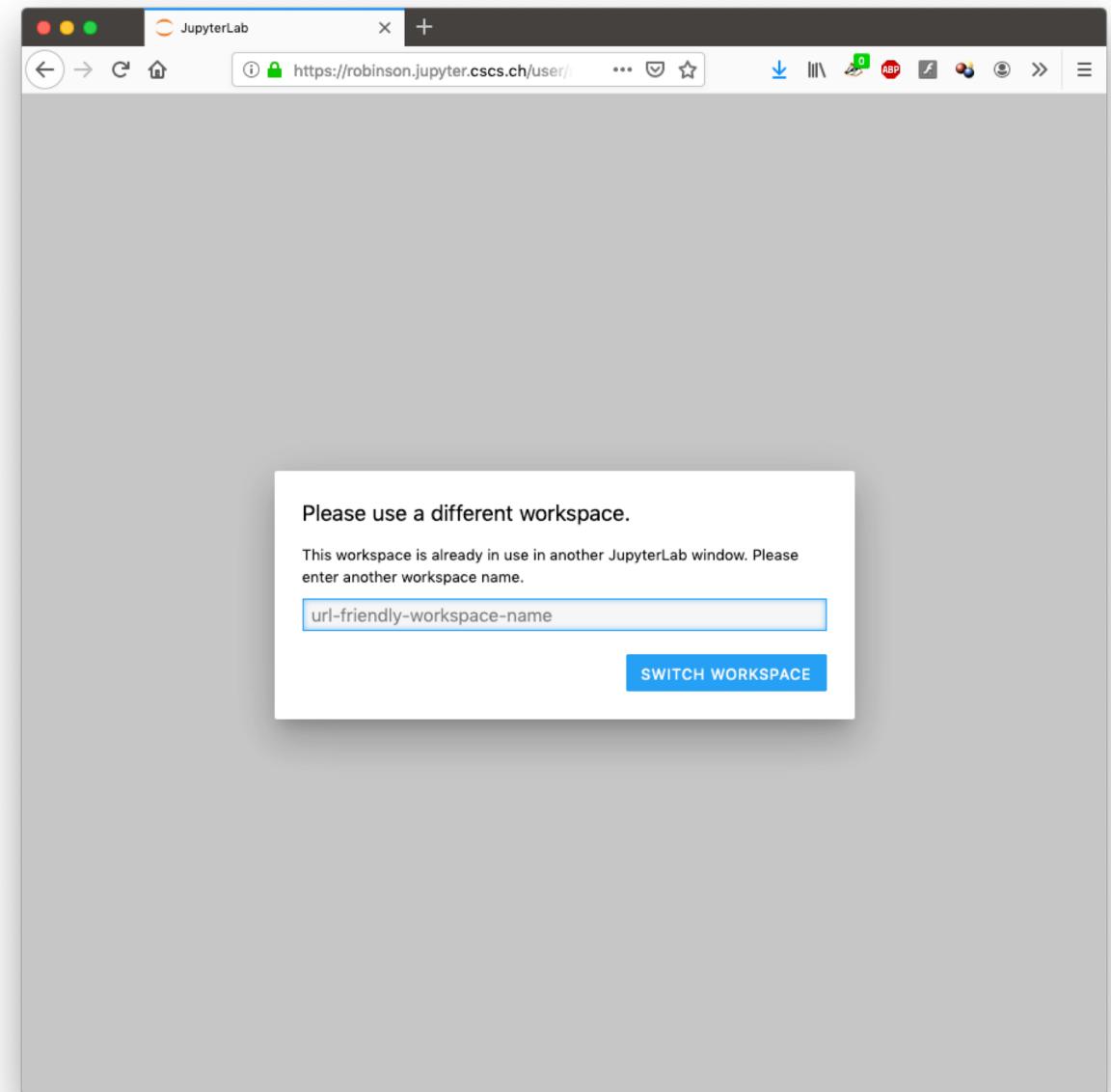
Find and replace

- First class support for find and replace across JupyterLab
- Currently works in notebooks and text files
- Supports regex
- Extensible to other widgets

The screenshot shows a JupyterLab interface. At the top, there's a toolbar with icons for file operations, search, and help. Below it is a menu bar with File, Edit, View, Run, Kernel, Tabs, Settings, and Help. A status bar at the bottom indicates "Mem:200 MB". The main area has a sidebar on the left with various icons. In the center, a code cell contains the text "Let's do a find and replace" followed by "[19]: a = 1 + 2". The number "19" is blue, while "a = 1 + 2" is highlighted with a light blue border. The status bar at the bottom also shows "Mode: Edit" and "Ln 1, Col 11 Untitled27.ipynb".

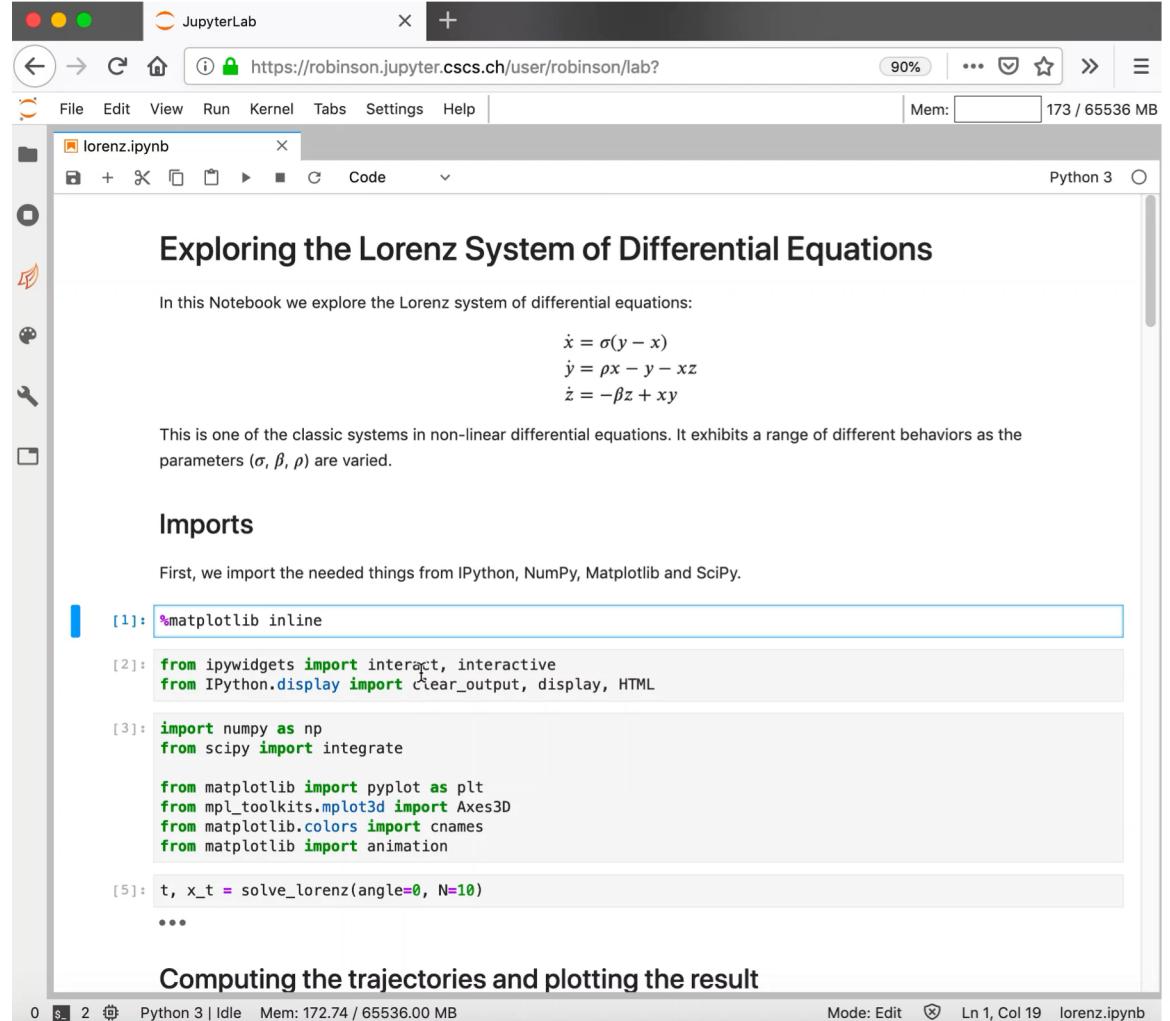
New workspaces are auto-generated

- Previously, you would receive an annoying pop-up message ***Please use a different workspace*** if you have a workspace open in another browser tab



New workspaces are auto-generated

- Previously, you would receive an annoying pop-up message ***Please use a different workspace*** if you have a workspace open in another browser tab
- Now, **new workspaces are automatically generated** when you create a new window with the same workspace name
- User is redirected to a randomly generated workspace name of the form auto-x where x is randomly drawn from [A-Z,a-z,0-9].



The screenshot shows a JupyterLab interface in a web browser. The title bar says "JupyterLab". The address bar shows the URL <https://robinson.jupyter.cscs.ch/user/robinson/lab?>. The sidebar on the left has icons for file operations like Open, Save, and Delete. A central panel displays a notebook titled "lorenz.ipynb". The title of the notebook is "Exploring the Lorenz System of Differential Equations". Below the title, it says: "In this Notebook we explore the Lorenz system of differential equations:" followed by the Lorenz equations: $\dot{x} = \sigma(y - x)$, $\dot{y} = \rho x - y - xz$, $\dot{z} = -\beta z + xy$. It notes that this is one of the classic systems in non-linear differential equations. The code section shows imports for matplotlib, ipywidgets, IPython, numpy, scipy, and other plotting libraries. The status bar at the bottom indicates "Python 3 | Idle" and "Mem: 172.74 / 65536.00 MB".

Other User Facing Changes in JupyterLab 1.X

- “Inspector” is renamed “Contextual Help” and is moved to the Help menu
- Drag and drop notebook cells into console or text editor
- Command to render all markdown cells
- Ability to “Merge Selected Cells” in the context menu
- Notification when a kernel is automatically restarted, e.g., out-of-memory crash
- Many other changes to the user interface; various bug fixes



CSCS

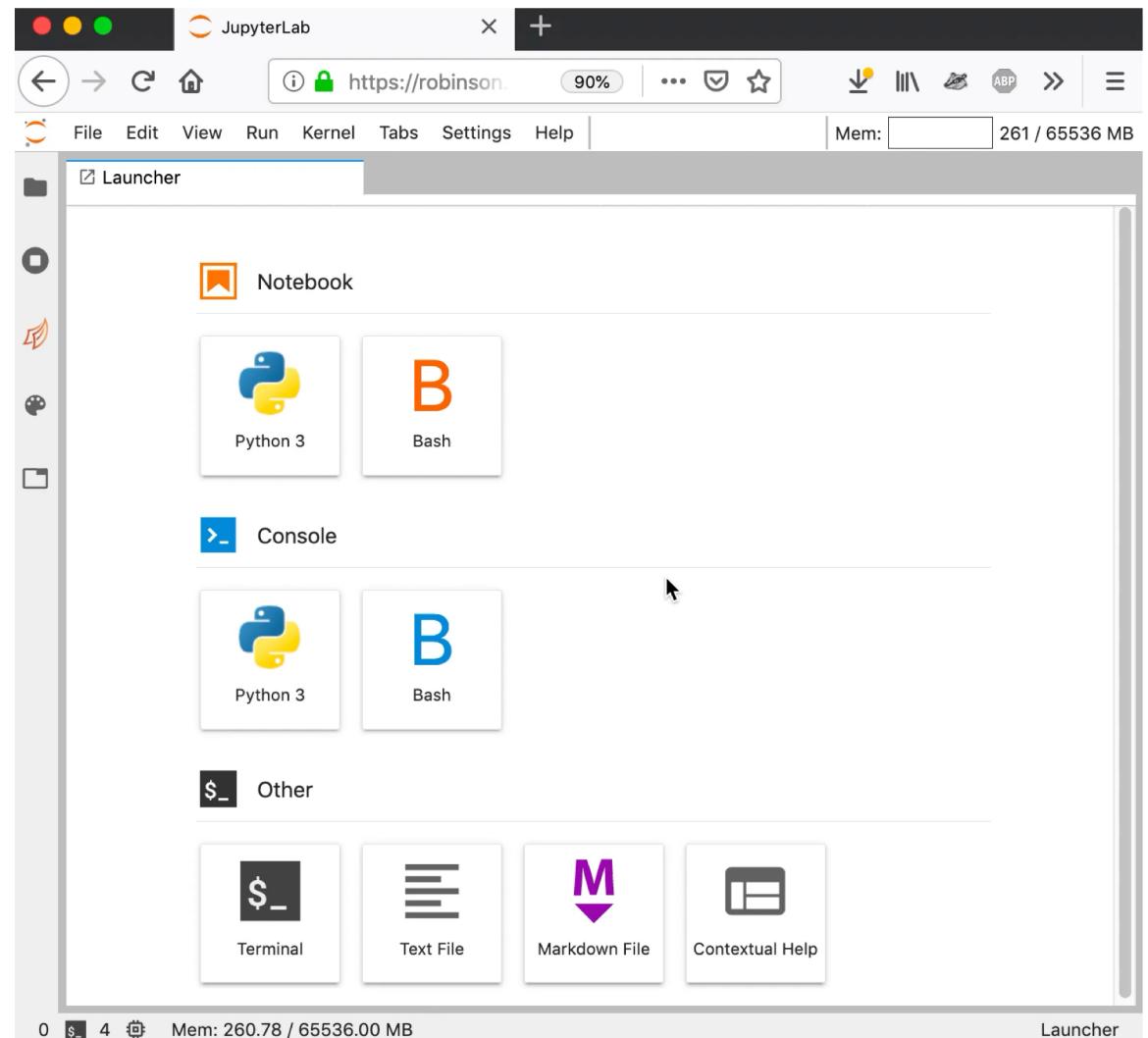
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

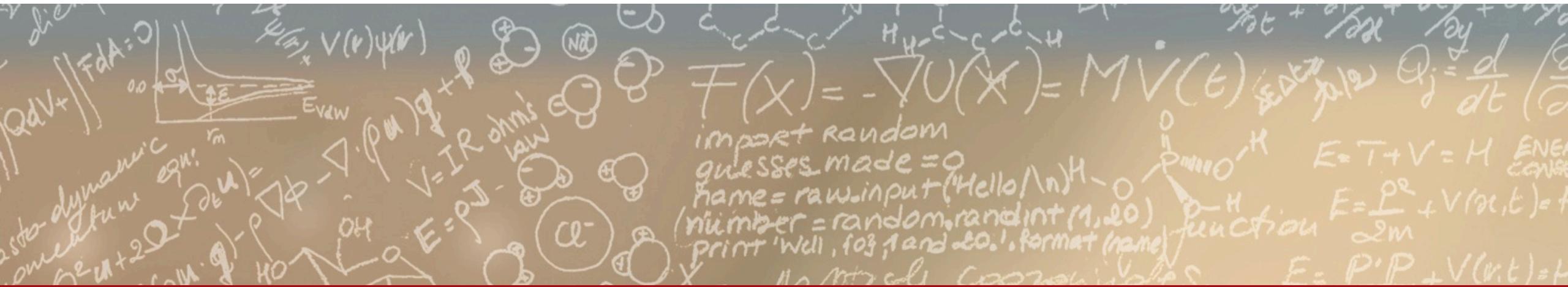
ETHzürich

New Functionality in JupyterHub 1.0

Named servers

- In addition to your default server, you can now have additional server(s) with names
- This allows you to have more than one server running at the same time
- For example, you might want one server running on a “gpu” node, and another server on an “mc” node
- Managed through a new user interface





Thank you. We welcome your feedback on the Jupyter service!

help@cscs.ch

Coming soon: survey on interactive computing use & requirements