



**Chan  
Zuckerberg  
Initiative** 

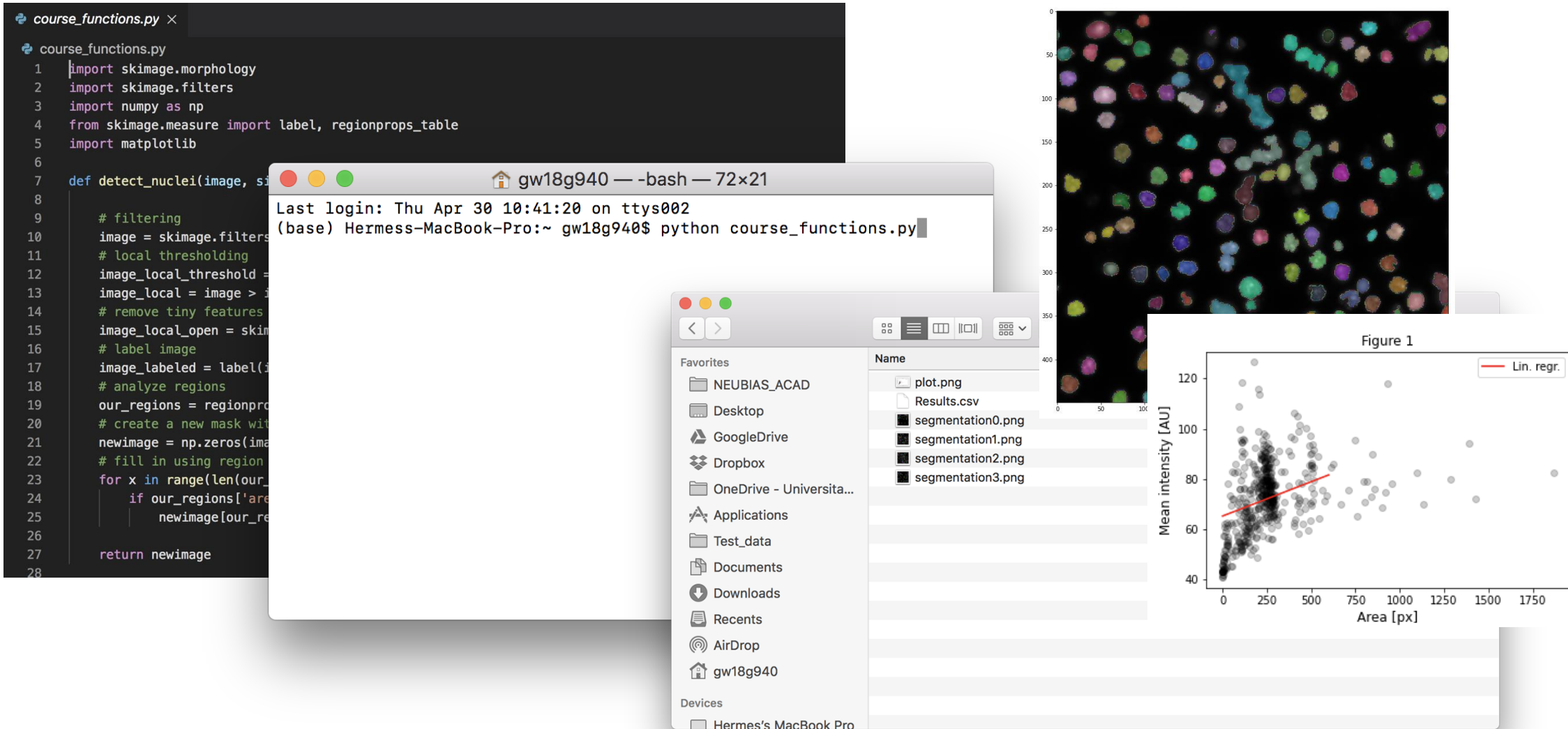


**Marcelo Leomil Zoccoler**

With materials from Robert Haase (PoL TU Dresden) and Guillaume Witz (Universität Bern)

August 2023

# “Classic” software vs. notebooks



# “Classic” software vs. notebooks

Code divided in parts

Dynamic: easy to test

Rich output: processing +  
visualisation + analysis in one place

Code + formatted text: easy  
documentation

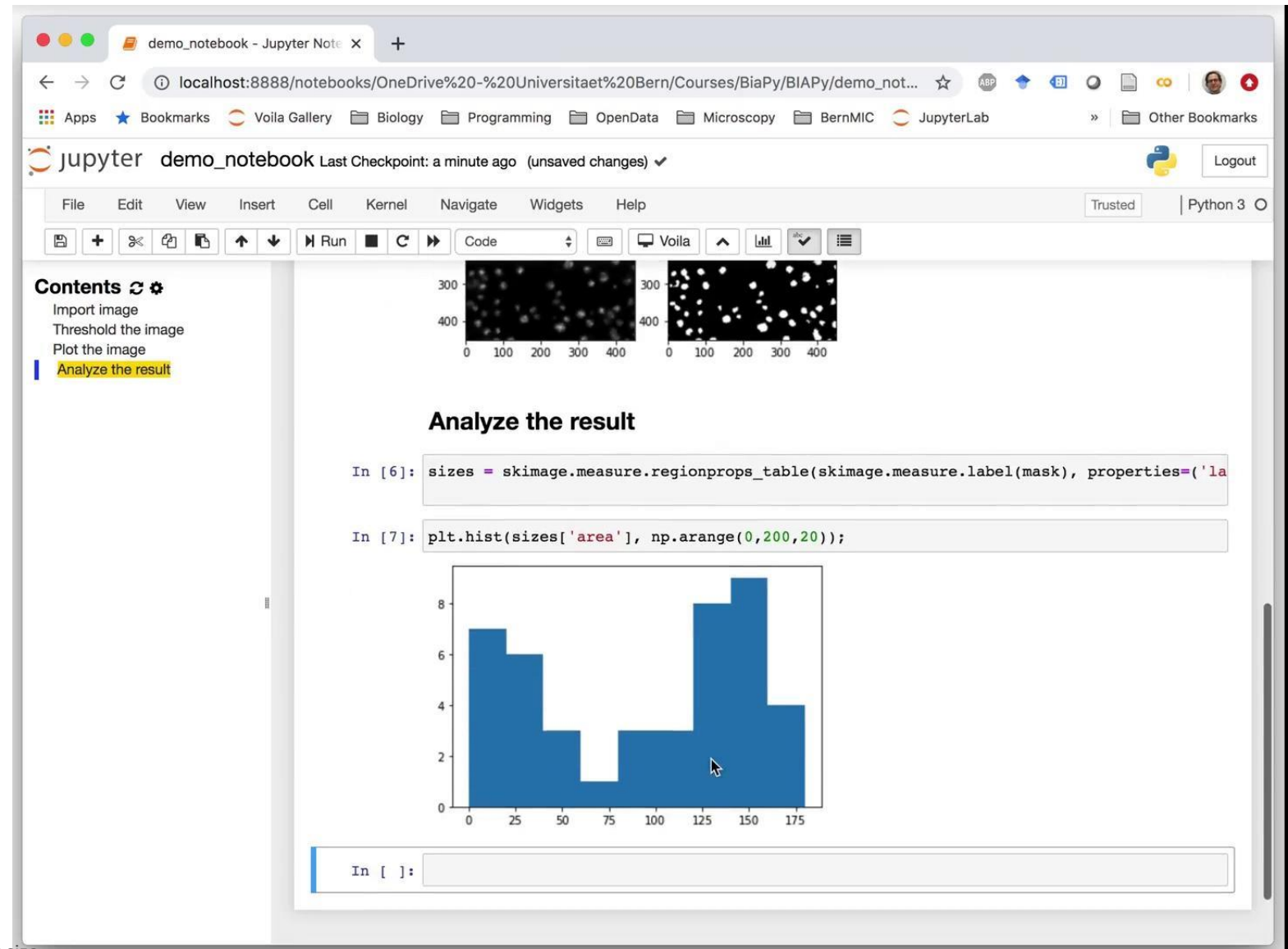


Illustration: load an image, threshold it, analyze object size



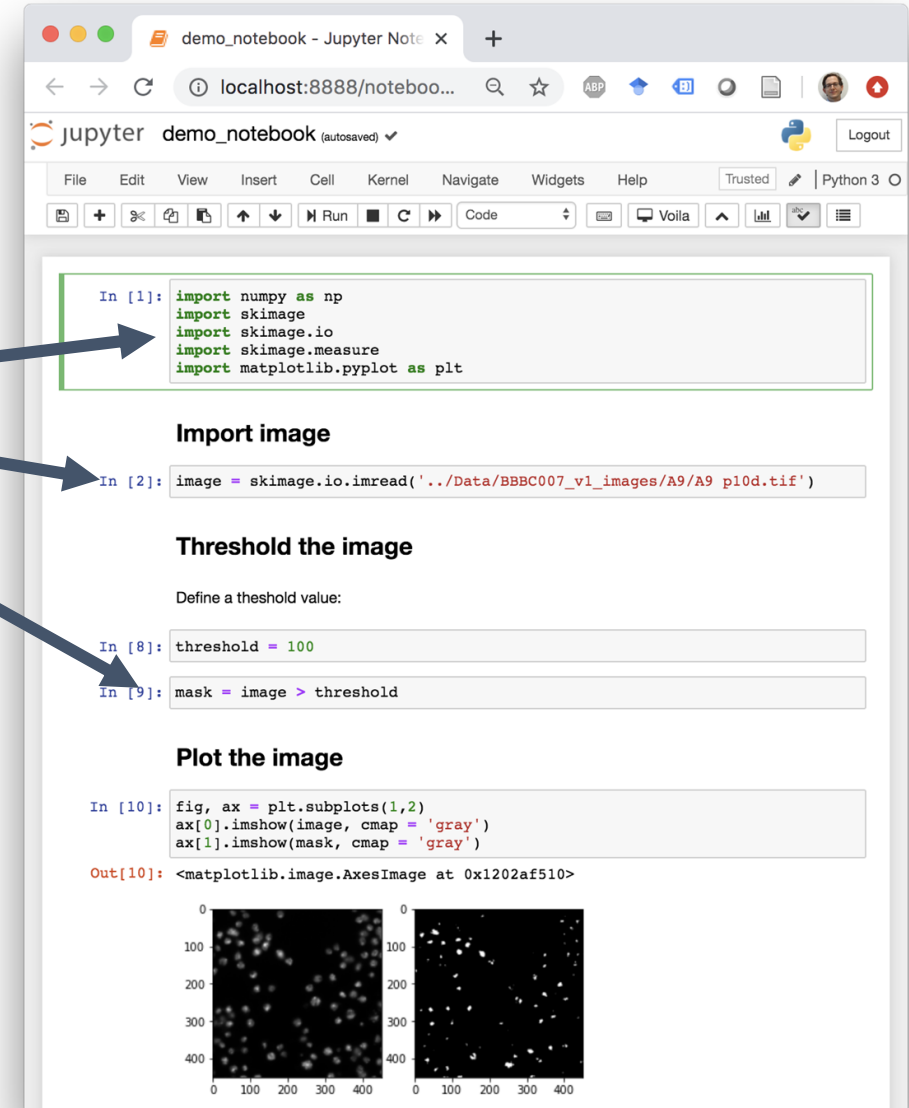
@zoccolermarcelo

# What is a jupyter notebook?

A text file (easily sent around)

Rendered by Jupyter in the browser

Split into sections called cells



# What is a jupyter notebook?

A text file (easily sent around)

Rendered by Jupyter in the browser

Split into sections called cells

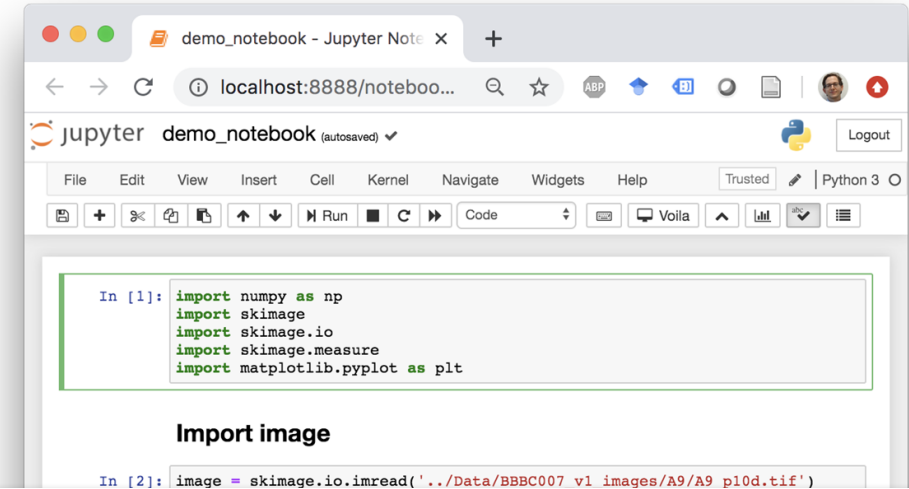
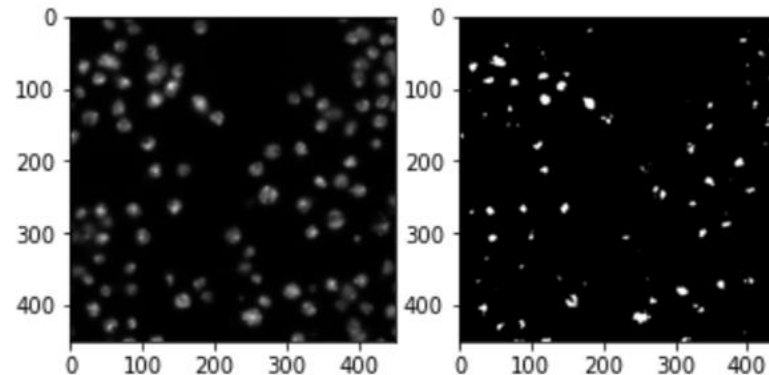
Cells can contain:

- Code
- Formatted text
- Rich output

In [2]: image

```
In [10]: fig, ax = plt.subplots(1,2)
ax[0].imshow(image, cmap = 'gray')
ax[1].imshow(mask, cmap = 'gray')

Out[10]: <matplotlib.image.AxesImage at 0x1202af510>
```



# Why using notebooks?

- Documenting (for your-(future)-self and for others) and enhanced reproducibility

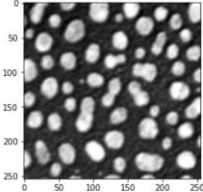
“First, we applied a Gaussian filter from scikit-image with  $\sigma = 1$ . Then, we applied another Gaussian filter to the original image with  $\sigma = 10$ . After that, we subtracted the first result from the second...

**Image Processing Workflow**

```
[6]: from skimage.filters import gaussian
     from skimage.io import imread
     import matplotlib.pyplot as plt

* [5]: image = imread(r'..\data\image.tif')

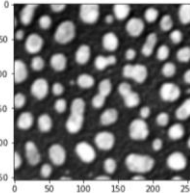
[7]: plt.imshow(image, cmap = 'gray')
[7]: <matplotlib.image.AxesImage at 0x1900ad27b50>
```



**1. Apply a gaussian filter with a small sigma**

We used  $\sigma = 1$  in this example.


```
[14]: image_gauss1 = gaussian(image, sigma = 1)
[15]: plt.imshow(image_gauss1, cmap = 'gray')
[15]: <matplotlib.image.AxesImage at 0x1900bf56670>
```



**2. Apply a gaussian filter with a large sigma**

We used  $\sigma = 10$  in this example.

```
[16]: image_gauss2 = gaussian(image, sigma = 10)
[17]: plt.imshow(image_gauss2, cmap = 'gray')
[17]: <matplotlib.image.AxesImage at 0x1900bfd5b0>
```

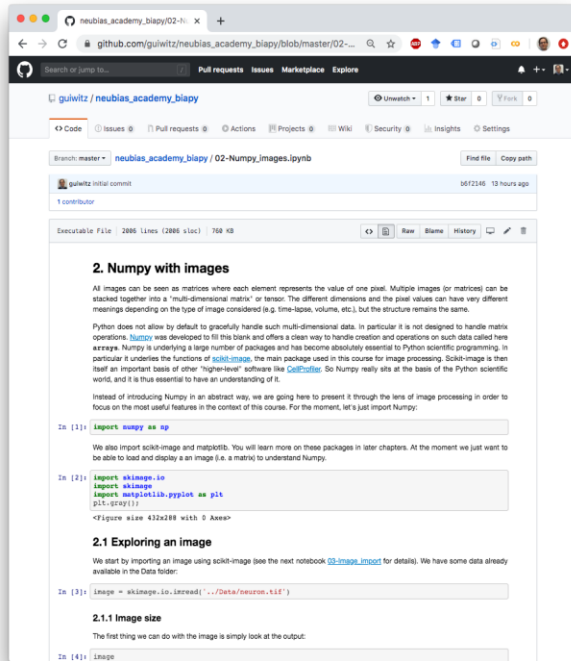


A large black arrow points from the original image to the first filtered image, and another large black arrow points from the first filtered image to the second filtered image, indicating the sequence of operations.

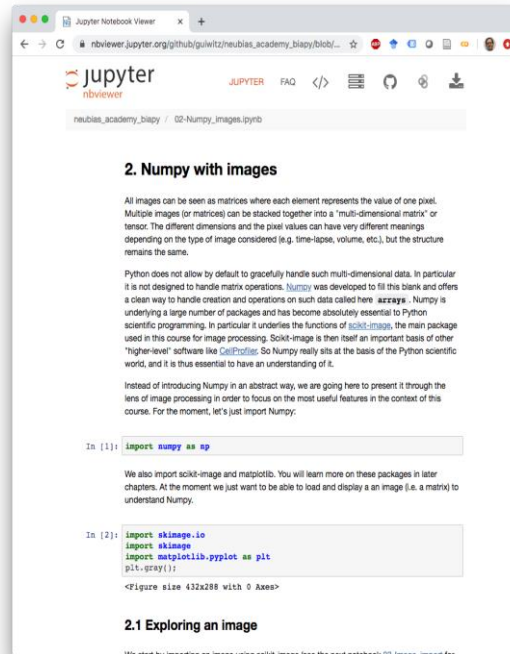


# Why using notebooks?

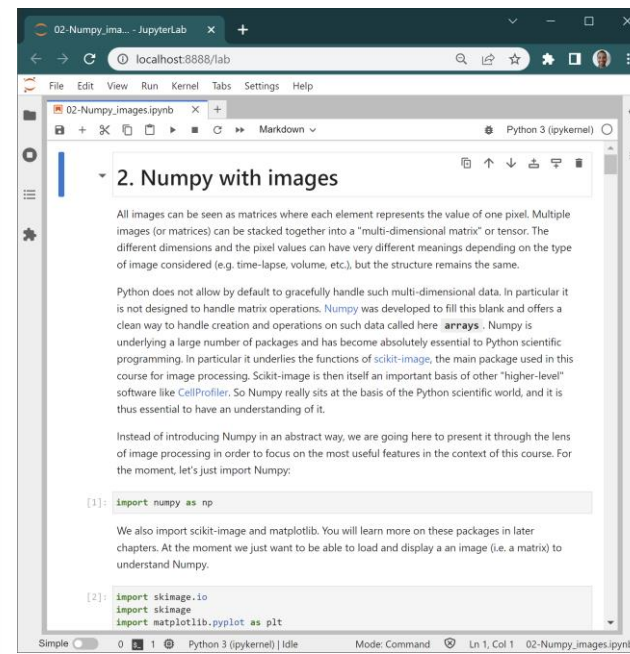
- Sharing
- - Send a single file with code, intermediary outputs and rich text explanations



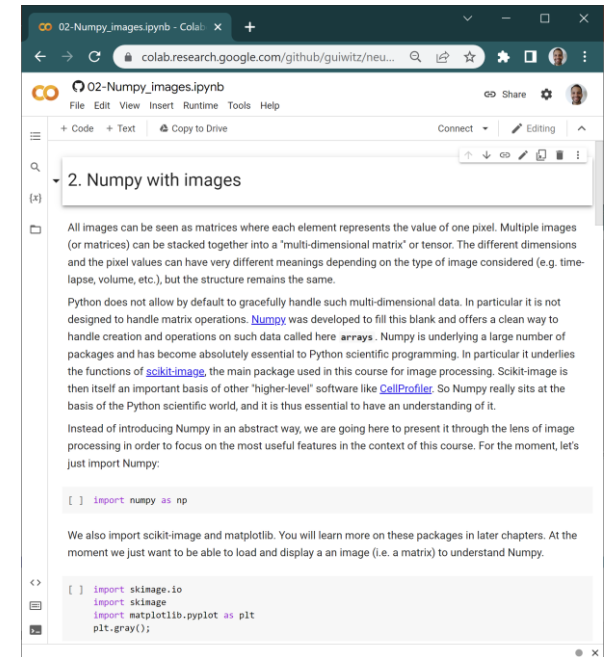
Github rendering



Nbviewer rendering



Jupyter lab (local or Binder)



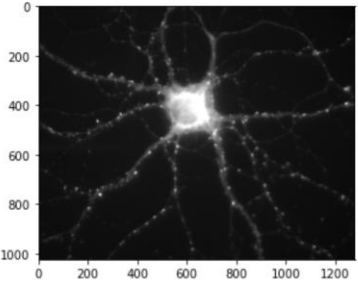
Google Colab

# Why using notebooks?

- Teaching
  - Small blocks of code with intermediary results are easier to understand than scripts that spit tons of outputs in sequence

```
This means that we have an image of 1024 rows and 1280 columns. We can now look at it using matplotlib (see next chapter for details) using the plt.imshow() function:
```

```
[6]: plt.imshow(image);
```



```
2.1.2 Image type
```

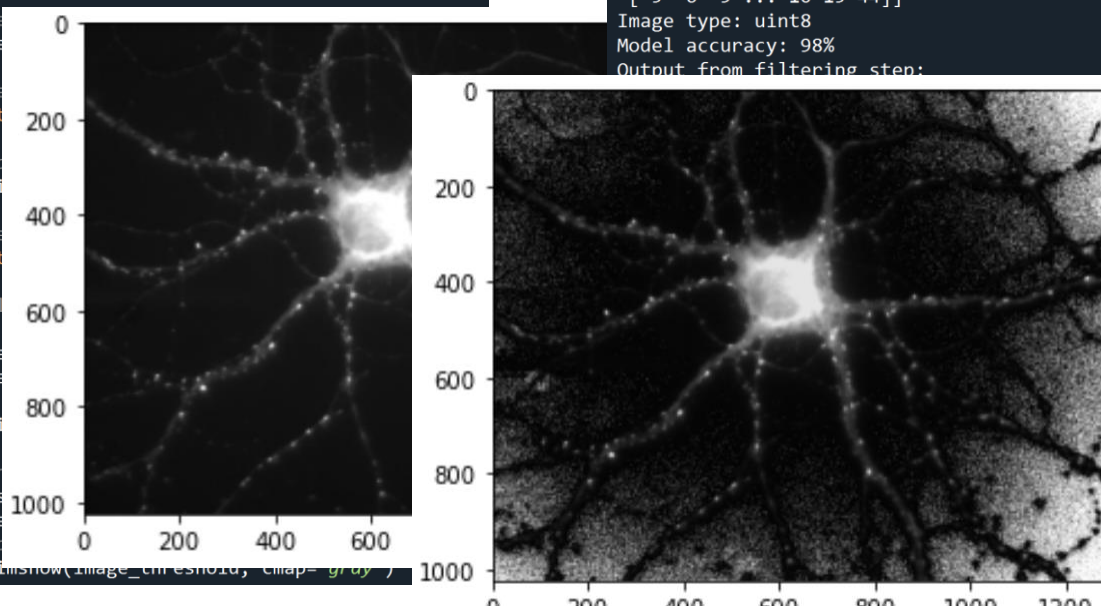
```
[7]: image
```

```
[7]: array([[32, 31, 31, ..., 18, 20, 22],  
         [27, 24, 25, ..., 24, 24, 23],  
         [26, 25, 25, ..., 17, 19, 20],  
         ...,  
         [25, 23, 24, ..., 19, 23, 23],  
         [25, 25, 25, ..., 19, 18, 19],  
         [26, 21, 23, ..., 19, 18, 22]], dtype=uint8)
```

In the output above we see that we have one additional piece of information: the array has `dtype`

```
4 import numpy as np  
5 import skimage.io  
6 import skimage  
7 import matplotlib.pyplot as plt  
8 plt.gray();  
9  
10 # Open image  
11 image =   
12  
13 # Check image shape  
14 print(image.shape)  
15  
16 # Display image  
17 plt.imshow(image)  
18  
19 # Check image type  
20 print(image.dtype)  
21  
22 # Calculate threshold  
23  
24 image_thresh =   
25 image_thresh =   
26  
27 plt.imshow(image_thresh)  
28  
29 # Manual thresholding  
30 thresh =   
31 image_thresh =   
32 # Display thresholded image  
33 plt.imshow(image_thresh, cmap=gray)
```

```
Pre-processing output: [[ 4 15  5 ... 69 83 86]  
 [ 5 10  8 ... 69 65 91]  
 [ 9  9  7 ... 42 67 73]  
 ...  
 [ 8  8 13 ... 12 28 51]  
 [ 6  6 10 ... 21 21 49]  
 [ 9  6  9 ... 10 19 44]]  
Image type: uint8  
Model accuracy: 98%  
Output from filtering step:
```





# Why using notebooks?

- Keep all the benefits from using code:
  - Batch processing
  - Running python functions/tools still unavailable as plugins

# Why using notebooks with napari?

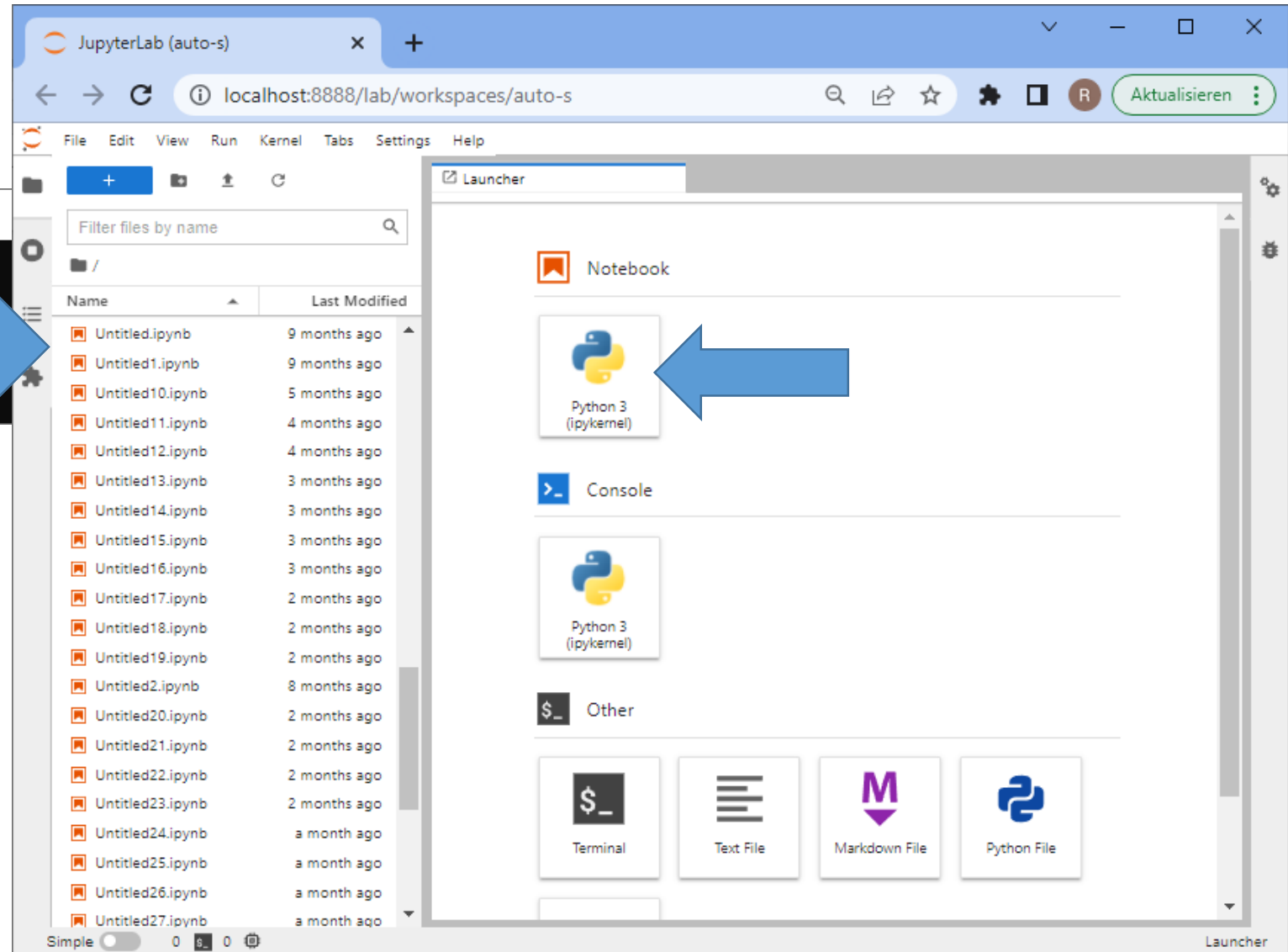
- Easy data interaction and visualization with napari:
  - Great for visualizing 3D (and more) data
  - Each processing step result can be displayed as a separate layer
- Data annotation

- Our programming environment for this course

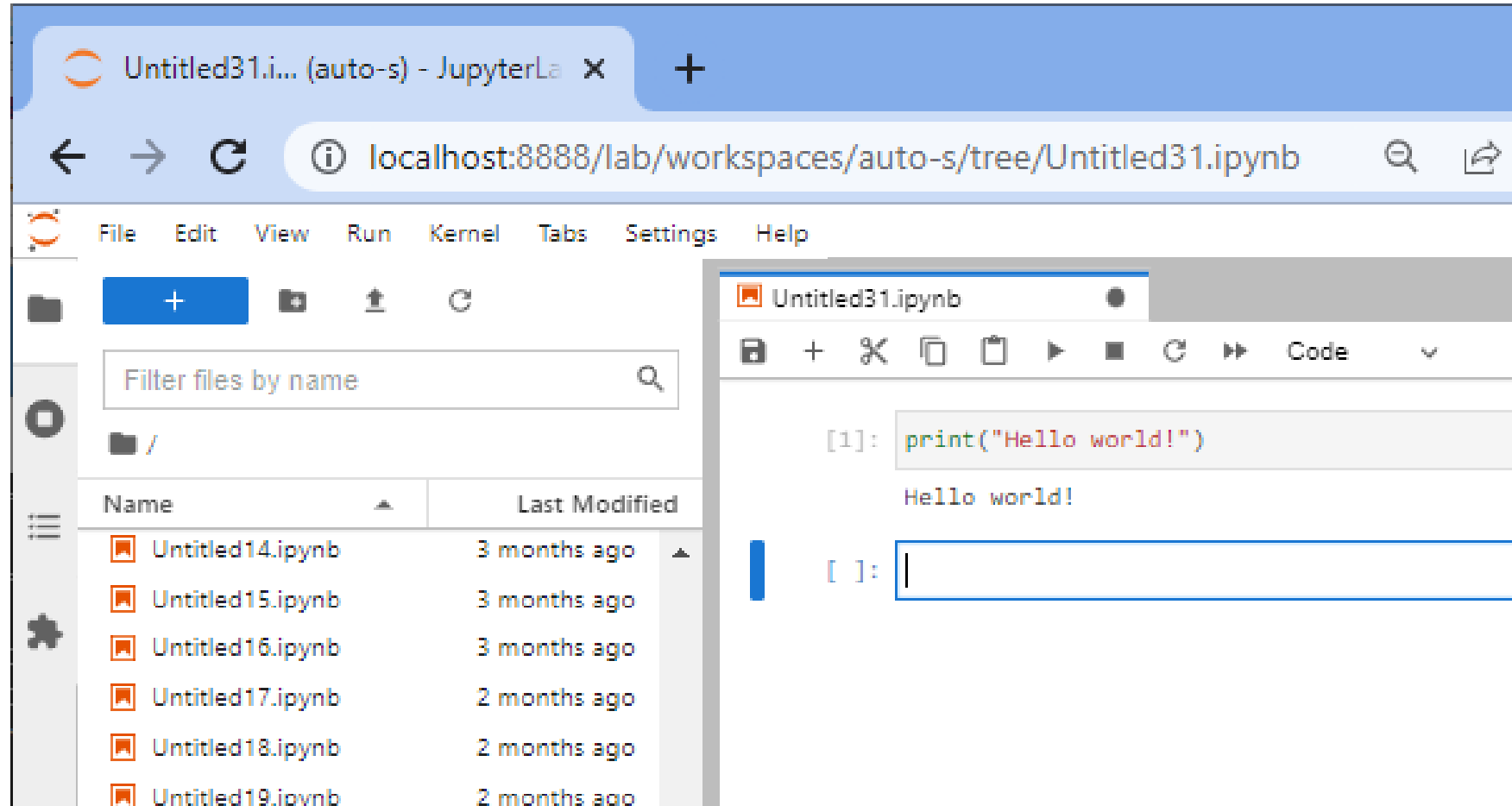
```
Miniforge Prompt - mamba deactivate  
  
(base) C:\Users\mazo260d>mamba activate napari-latam  
(napari-latam) C:\Users\mazo260d>jupyter lab
```

Current environment

Current path



- Execute code cell-by-cell and see results instantaneously



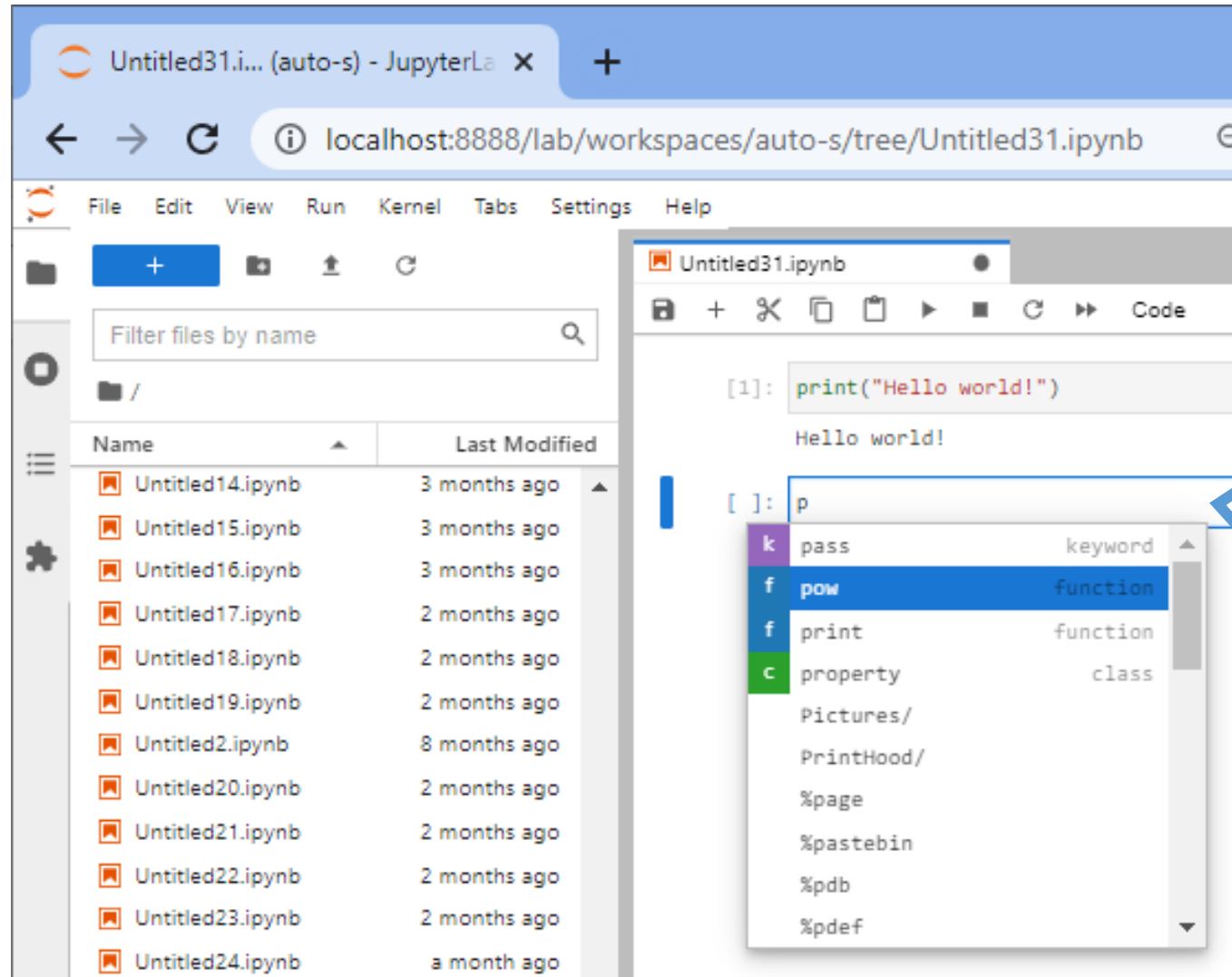
The screenshot shows the JupyterLab web interface. The top bar displays the browser tab 'Untitled31.i... (auto-s) - JupyterLa' and the address bar 'localhost:8888/lab/workspaces/auto-s/tree/Untitled31.ipynb'. The left sidebar contains a file browser with a search bar 'Filter files by name' and a list of files: 'Untitled14.ipynb', 'Untitled15.ipynb', 'Untitled16.ipynb', 'Untitled17.ipynb', 'Untitled18.ipynb', and 'Untitled19.ipynb'. The main area shows a code editor for 'Untitled31.ipynb' with a toolbar and a code cell containing the following text:

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

A blue box highlights the code cell, and a callout bubble points to it with the text:

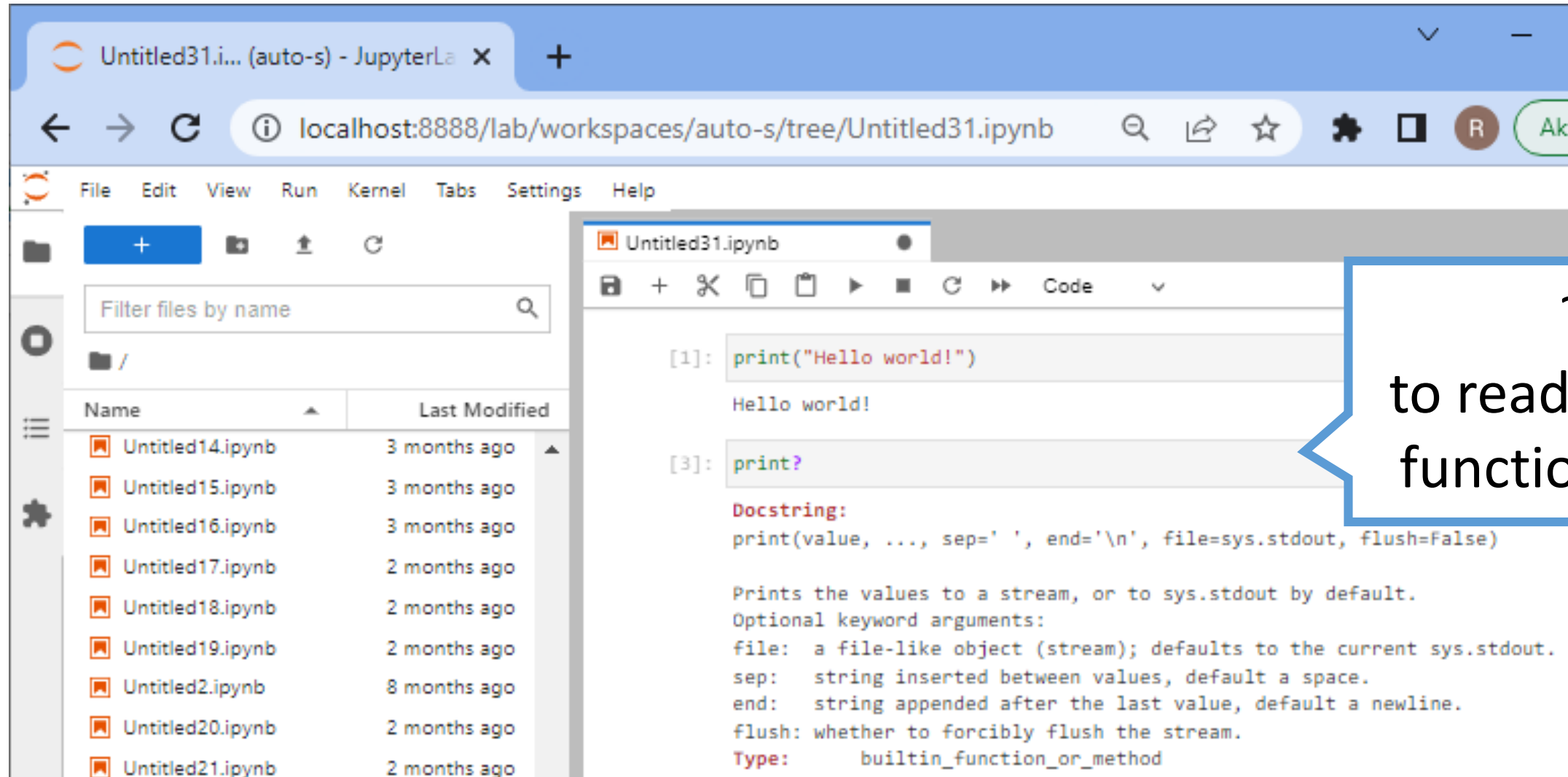
**SHIFT + ENTER**  
to execute a  
code cell

- Context-specific help, auto-completion



TAB  
to open auto-  
completion

- Help / “docstrings”



The screenshot shows the JupyterLab interface. On the left is a file browser with a search bar and a list of files. On the right is a code editor for 'Untitled31.ipynb'.

**File Browser:**

Name	Last Modified
Untitled14.ipynb	3 months ago
Untitled15.ipynb	3 months ago
Untitled16.ipynb	3 months ago
Untitled17.ipynb	2 months ago
Untitled18.ipynb	2 months ago
Untitled19.ipynb	2 months ago
Untitled2.ipynb	8 months ago
Untitled20.ipynb	2 months ago
Untitled21.ipynb	2 months ago

**Code Editor:**

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

[3]: print?

Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

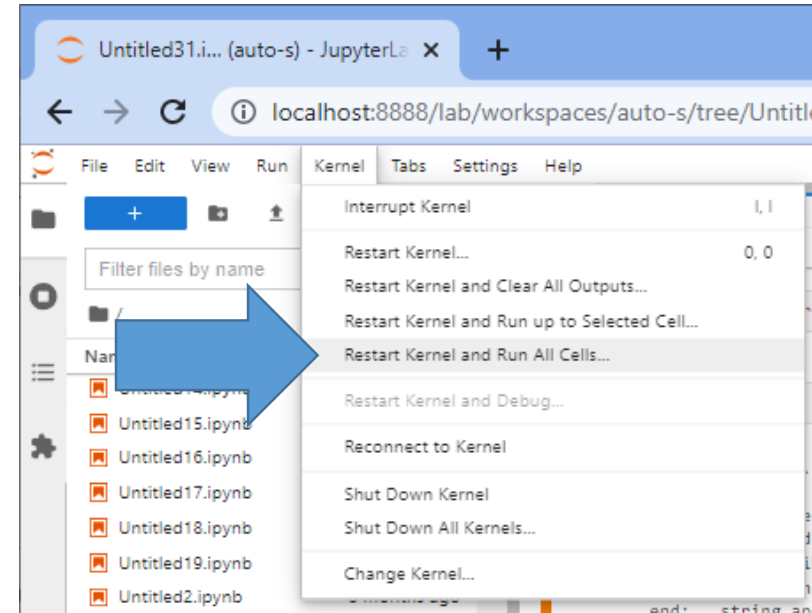
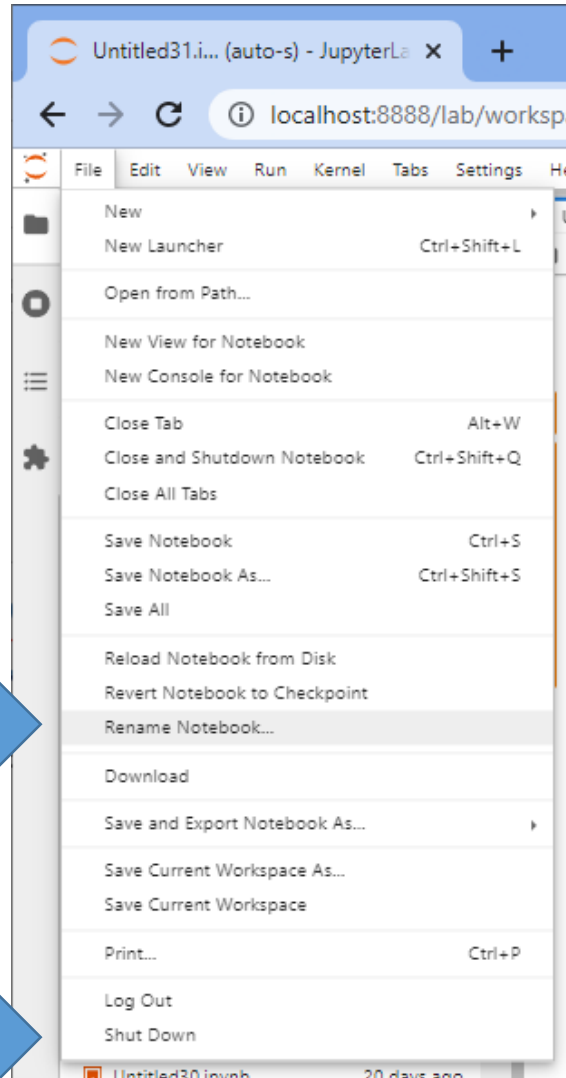
Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep:  string inserted between values, default a space.
end:  string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

?

to read what a  
function does



- Saving / renaming / closing



Enforcing a “clean” execution state is important for ensuring reproducibility and repeatability



**Chan  
Zuckerberg  
Initiative** 

# Python programming basics

Marcelo Leomil Zoccoler

With materials from Robert Haase

August 2023

- Variables can hold numeric values and you can do math with them

```
▶ # initialize program  
a = 5  
b = 3  
  
# run algorithm on given parameters  
sum = a + b  
  
# print out result  
print (sum)
```

8

- Math commands supplement operators to be able to implement any form of calculations

- Power

```
▶ pow(3, 2)
```

```
] : 9
```

- Absolute

```
▶ abs(-8)
```

```
] : 8
```

- Rounding

```
▶ round(4.6)
```

```
] : 5
```

Be careful with  
some of them!

```
▶ round(4.5)
```

```
] : 4
```

[https://en.wikipedia.org/wiki/Rounding#Round\\_half\\_to\\_even](https://en.wikipedia.org/wiki/Rounding#Round_half_to_even)

Comments should contain additional information such as

- User documentation
  - What does the program do?
  - How can this program be used?
- Your name / institute in case a reader has a question
- Comment why things are done.
- Do not comment what is written in the code already!

```
#  
# This program sums up two numbers.  
#  
# Usage:  
# * Run it in Python 3.8  
#  
# Author: Robert Haase, PoL TUD  
#         Robert.haase@tu-dresden.de  
# April 2021  
  
# initialise program  
a = 1  
b = 2.5  
  
# run complicated algorithm  
final_result = a + b  
  
# print the final result  
print( final_result )
```

- Also strings as values for variables are supported

Single and double quotes  
allowed

```
▶ firstname = "Robert"  
  lastname = 'Haase'  
  
print("Hello " + firstname + " " + lastname)
```

Hello Robert Haase



- String **f**ormatting is made easy using f-strings.

```
f"This is an f-string. a's value is {a}. Doubling the value of a gives {2*a}."
```

```
"This is an f-string. a's value is 5. Doubling the value of a gives 10."
```

- Using f-strings, you can also call code from within a string. Take care of code readability!

```
f"The first_name variable contains {first_name.lower().count('r')} r letters."
```

```
'The first_name variable contains 2 r letters.'
```

- Also strings as values for variables are supported
- When combining strings and numbers, you need to explicitly define what you want to do.

▶ *# mixing types*

```
a = 5
b = "2"

print (a + b)
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-4-51629e6a285f> in <module>
      4 b = "2"
      5
----> 6 print (a + b)
```

**TypeError:** unsupported operand type(s) for +: 'int' and 'str'

▶ *# mixing types to make numbers*

```
a = 5
b = "2"

print (a + int(b))
```

7

▶ *# mixing types*

```
a = "5"
b = 2

print (a + b)
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-5-85ae49867097> in <module>
      4 b = 2
      5
----> 6 print (a + b)
```

**TypeError:** can only concatenate str (not "int") to str

▶ *# mixing types to make strings*

```
a = "5"
b = 2

print (a + str(b))
```

52

- Conversion to a floating point number: `float()`