

QM Course 2024  
Charles University, Prague

# Bio-Image Analysis with napari

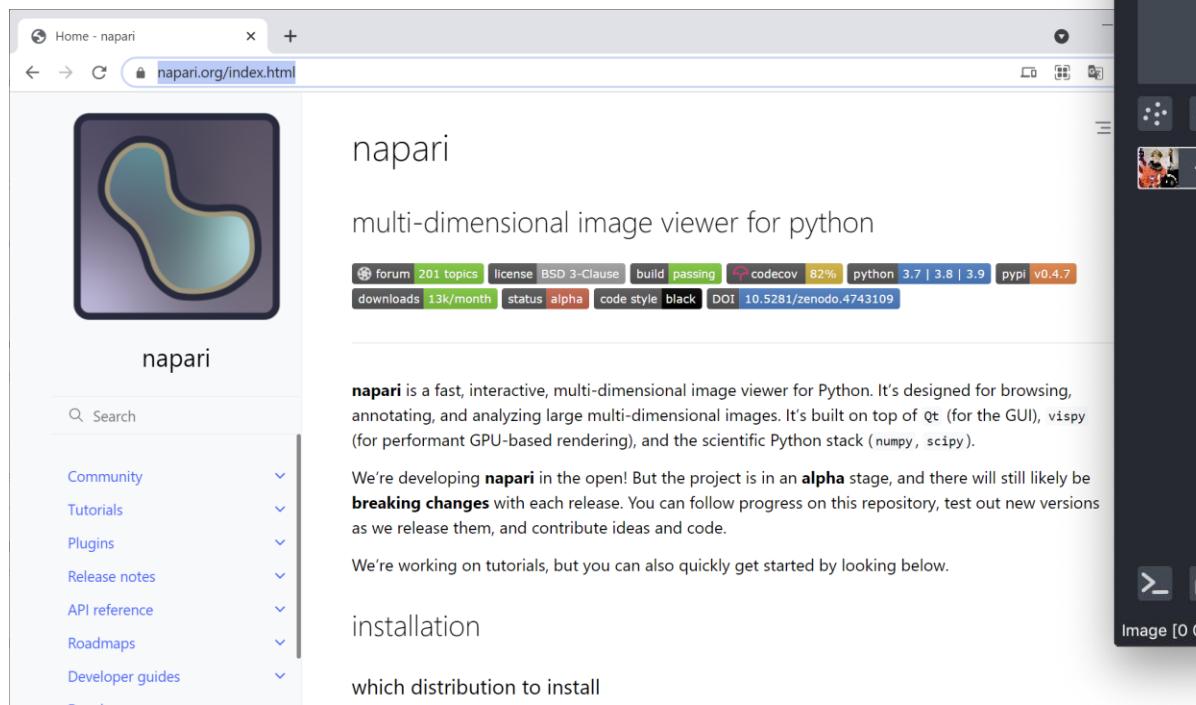
Marcelo Leomil Zoccoler

With material from:

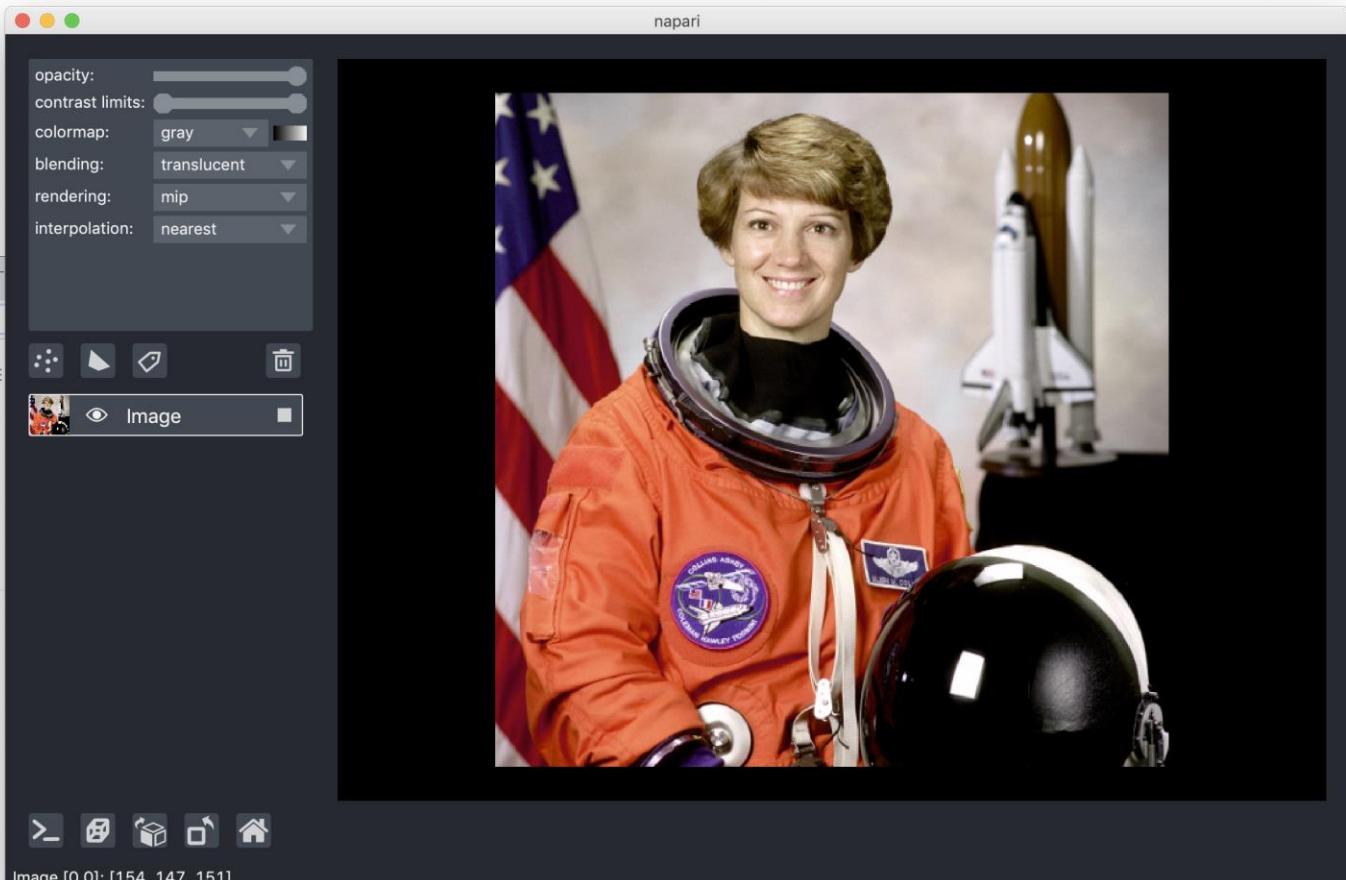
Robert Haase, PoL; Guillaume Witz, Universität Bern

# Napari: 3D viewer for Python

- Multi-dimensional image viewer in python
- <https://napari.org/>



The screenshot shows the official napari website at napari.org/index.html. The page features a sidebar with links for Home, Search, Community, Tutorials, Plugins, Release notes, API reference, Roadmaps, and Developer guides. The main content area displays the napari logo, a brief description of the software as a fast, interactive, multi-dimensional image viewer for Python, and sections for installation and distribution selection.



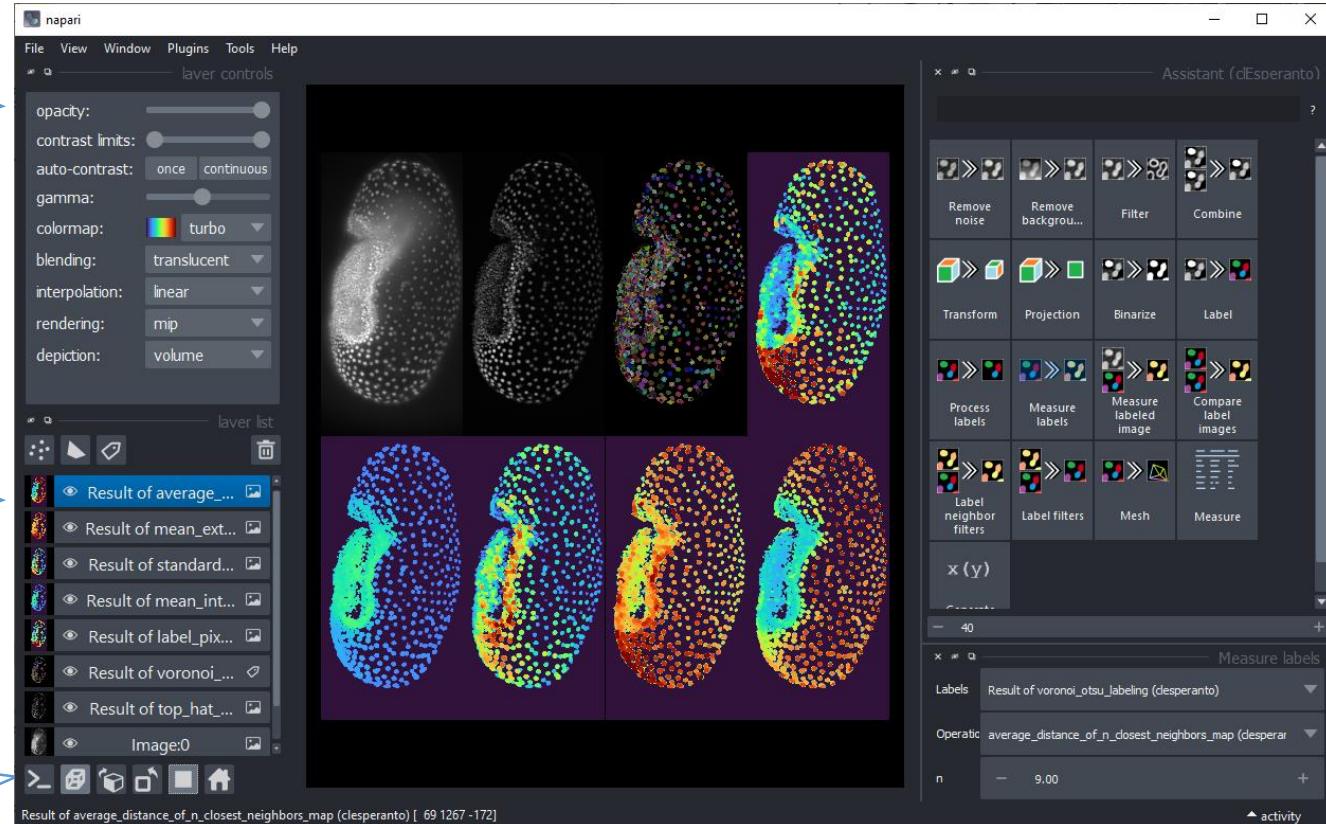
<https://napari.org/>

# Napari: 3D viewer for Python



<https://napari.org/>

# Napari user interface



View configuration / tools

```
layer.opacity = 0.5
```

Layers

```
layer.visible = False
```

Viewer controls

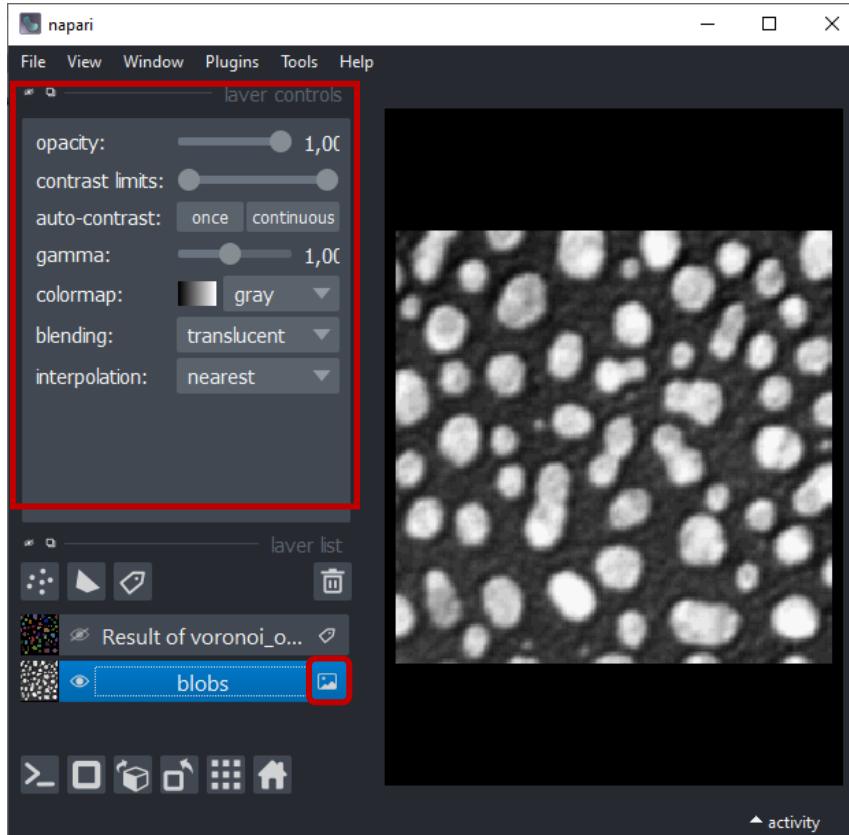
Dock widgets  
(custom plugins)

Function widgets  
(custom plugins)

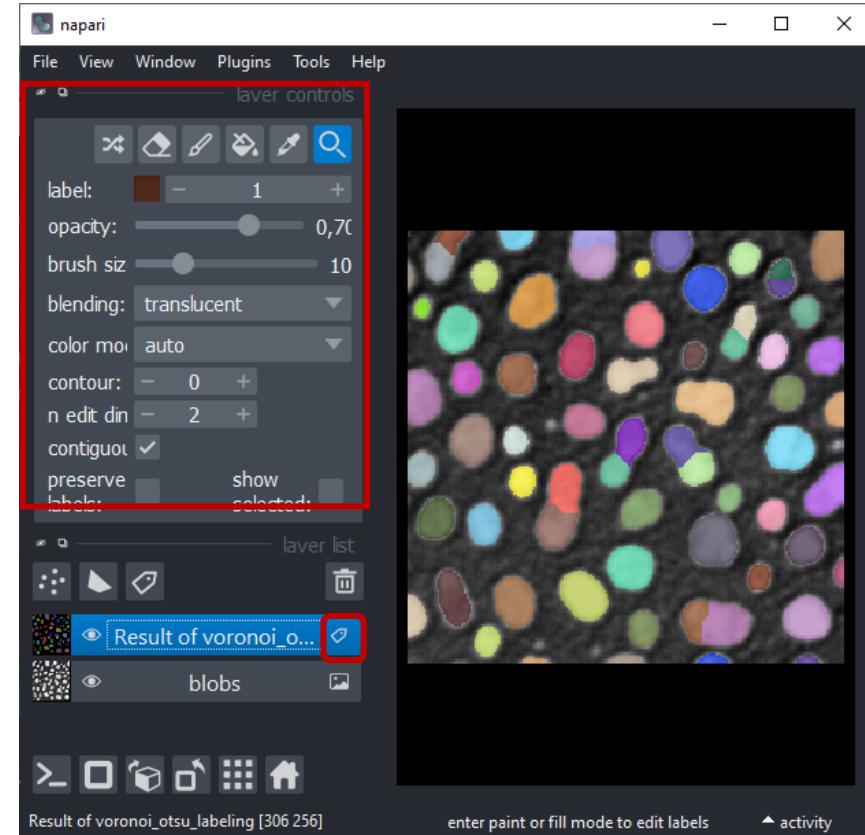
# Layer types

- Different layers have different tools and options

Image Layer

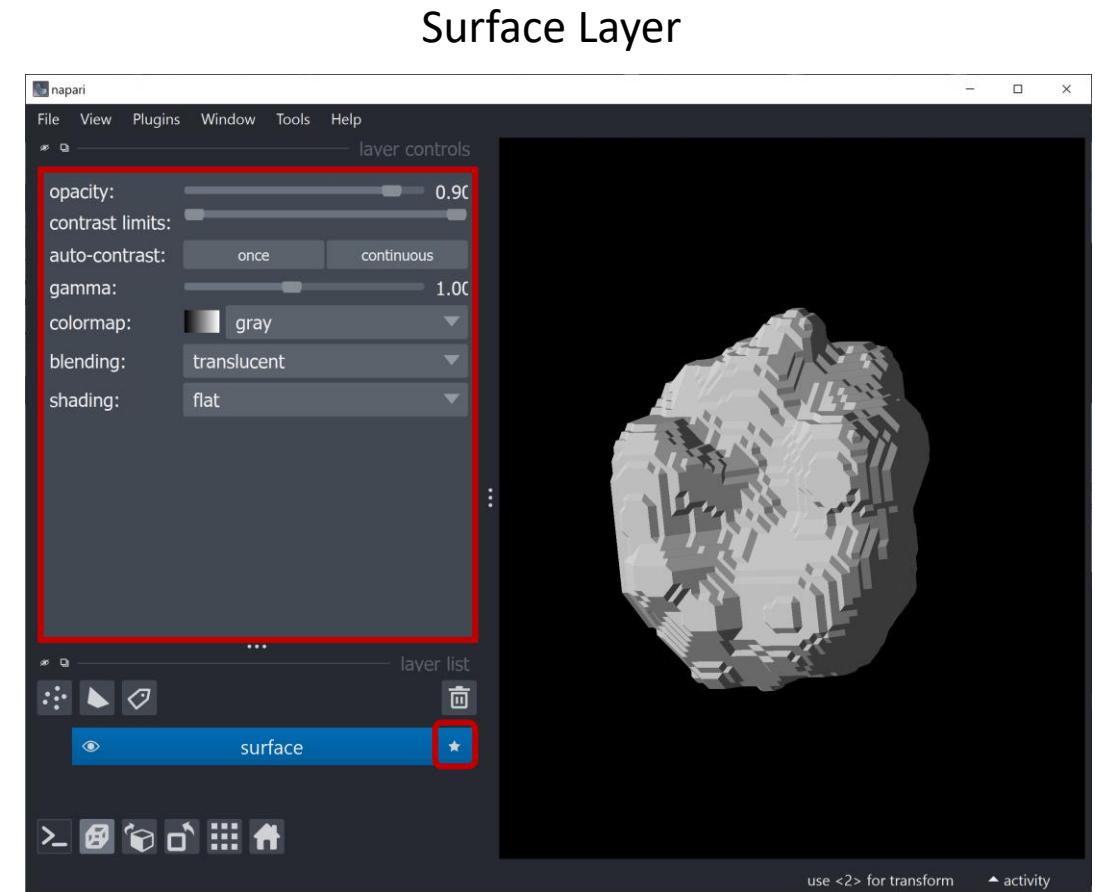
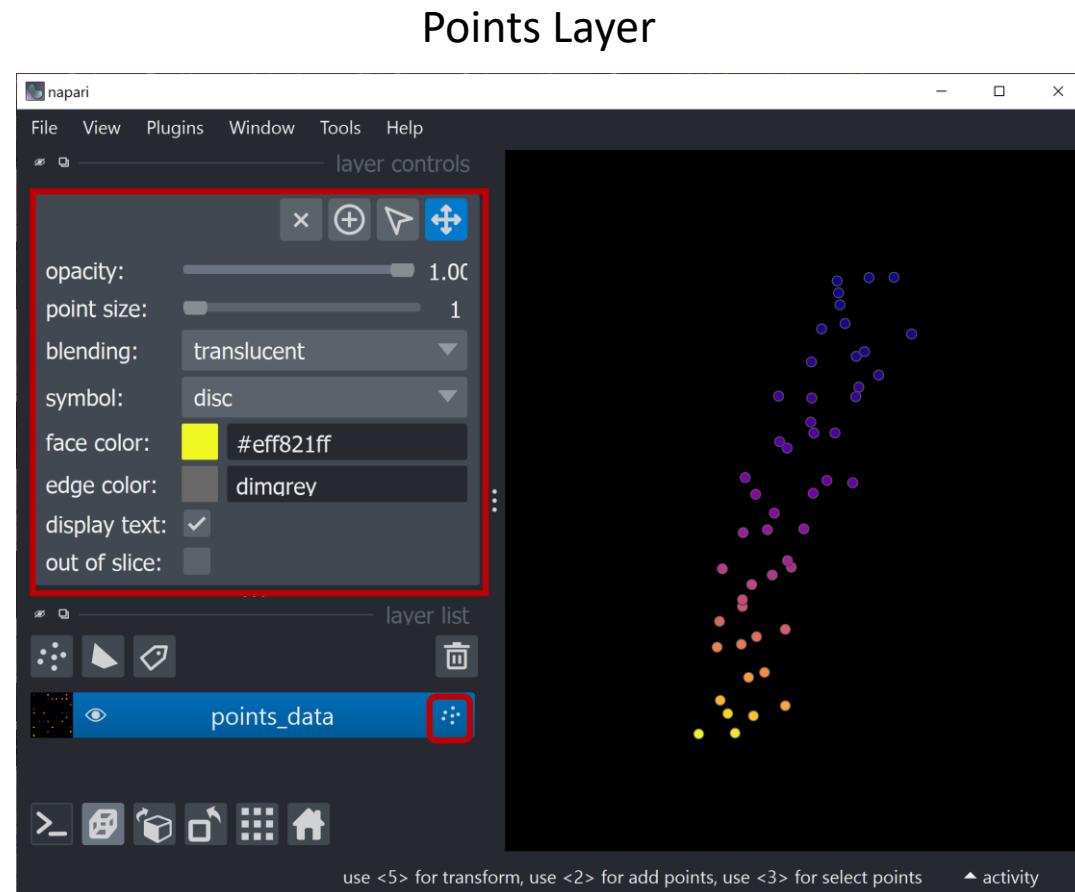


Labels Layer



# Layer types

- Different layers have different tools and options



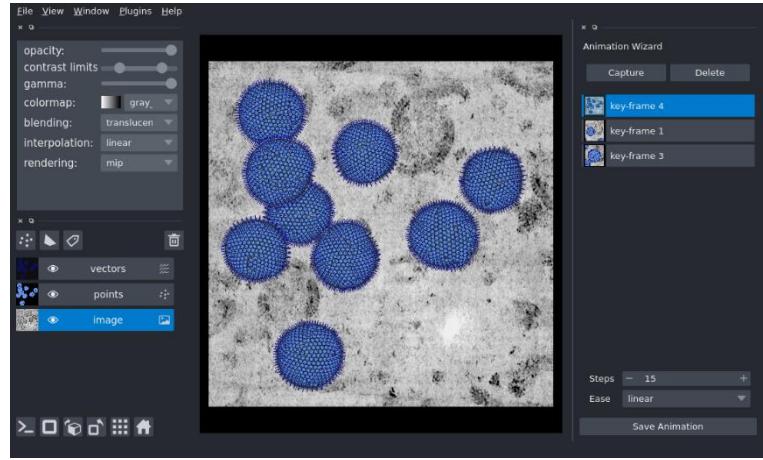
# napari plugins

## clusters-plotter



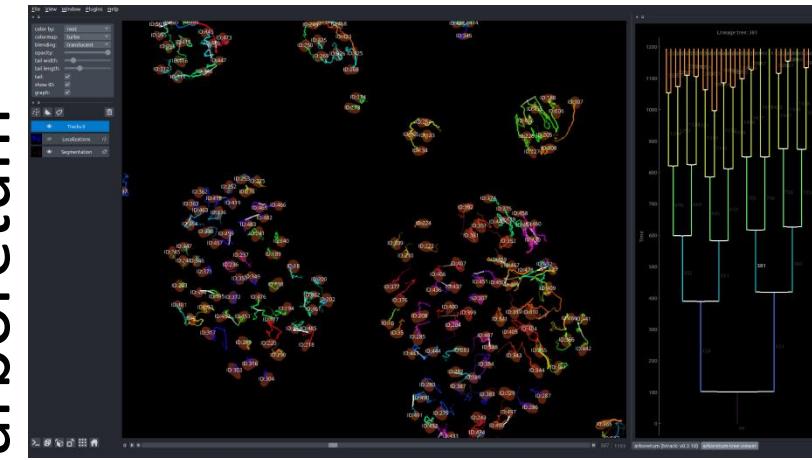
<https://github.com/BiAPoL/napari-clusters-plotter>

## animation



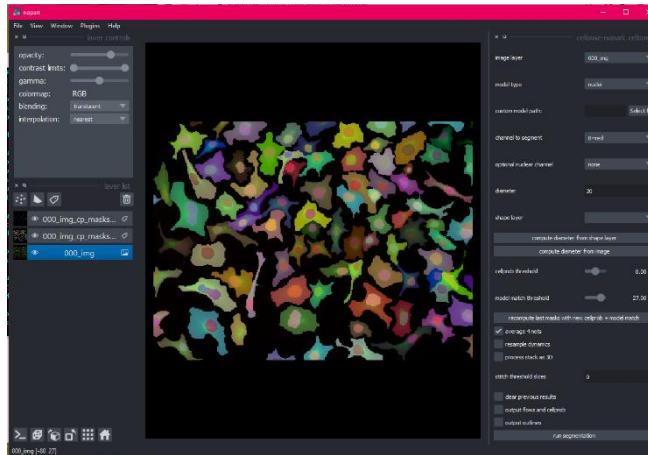
<https://github.com/napari/napari-animation>

## arboretum



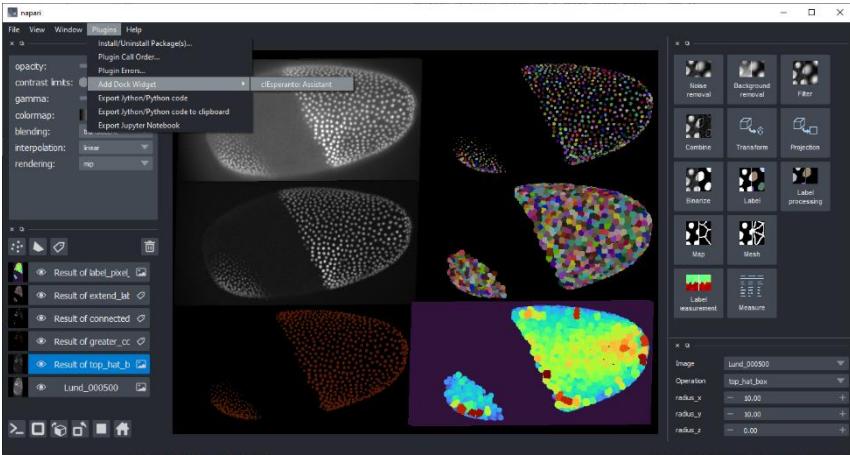
<https://github.com/quantumjot/arboretum>

## cellpose



<https://cellpose-napari.readthedocs.io/en/latest/>

## clesperanto



[https://github.com/c1Esperanto/napari\\_pyclesperanto\\_assistant](https://github.com/c1Esperanto/napari_pyclesperanto_assistant)

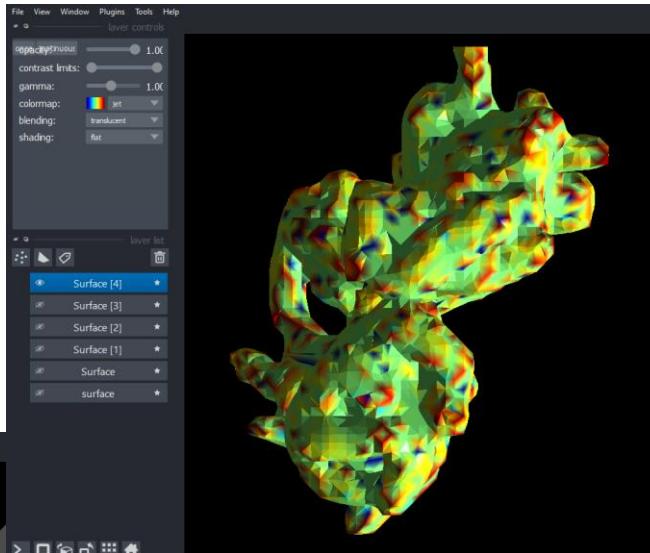
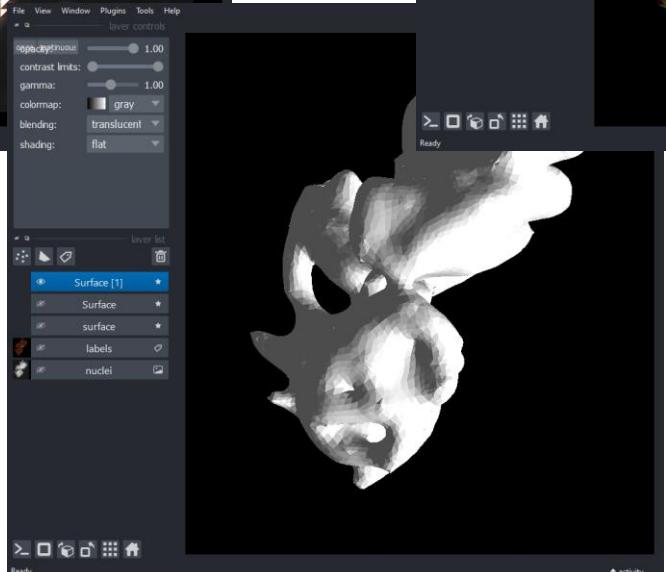
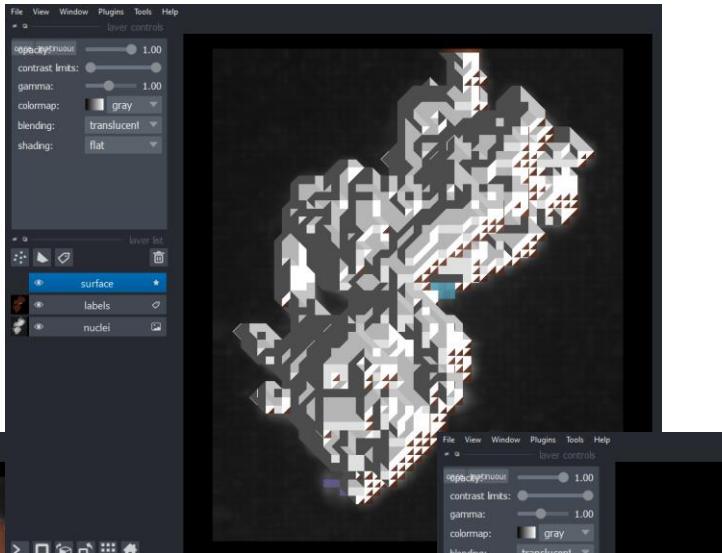
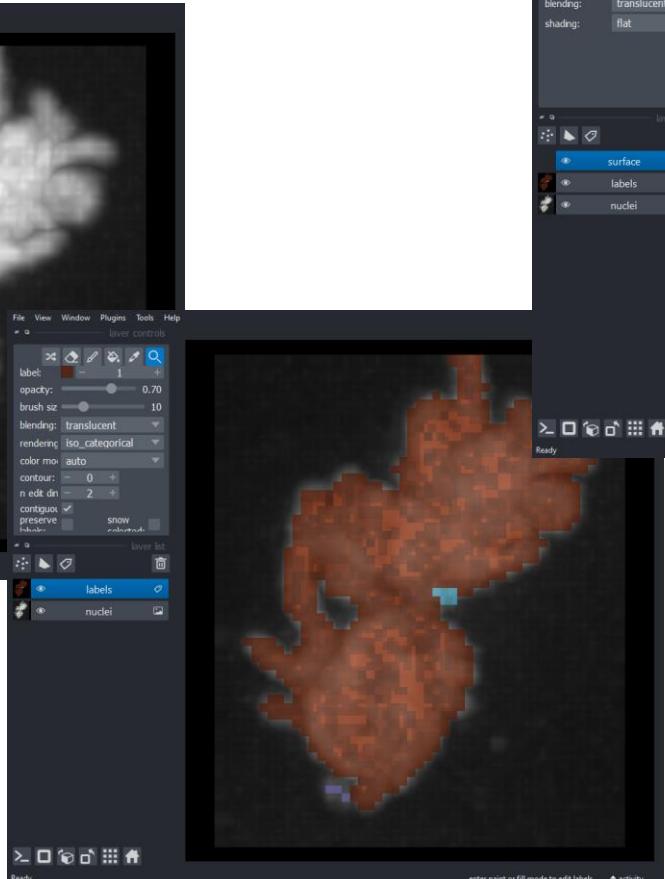
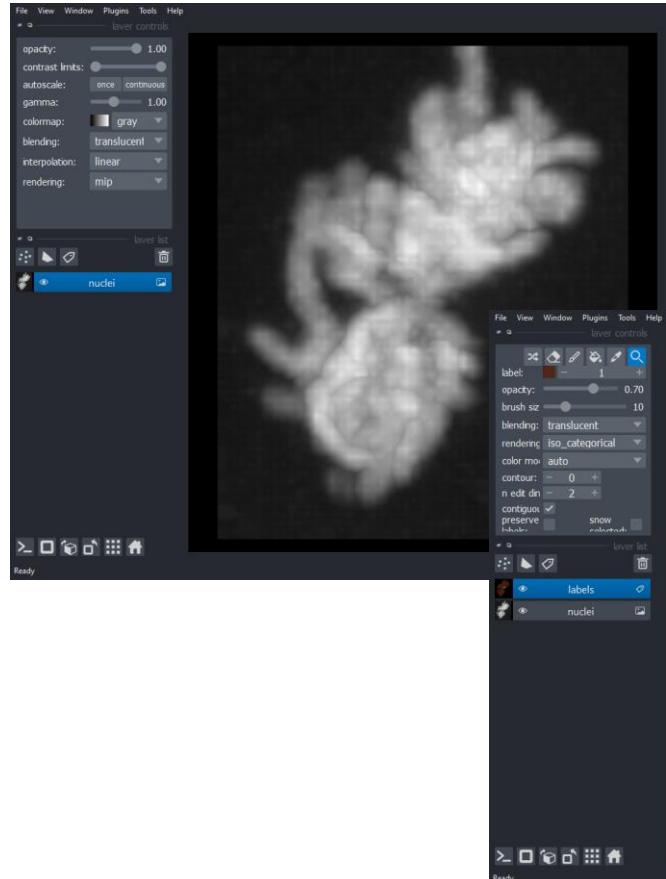


@zoccolermarcelo



# Napari process points and surfaces

- Surface extraction & analysis



Work with Zach Marin <https://github.com/zacsimile/napari-pymeshlab>  
<https://github.com/haesleinhuepf/napari-process-points-and-surfaces>

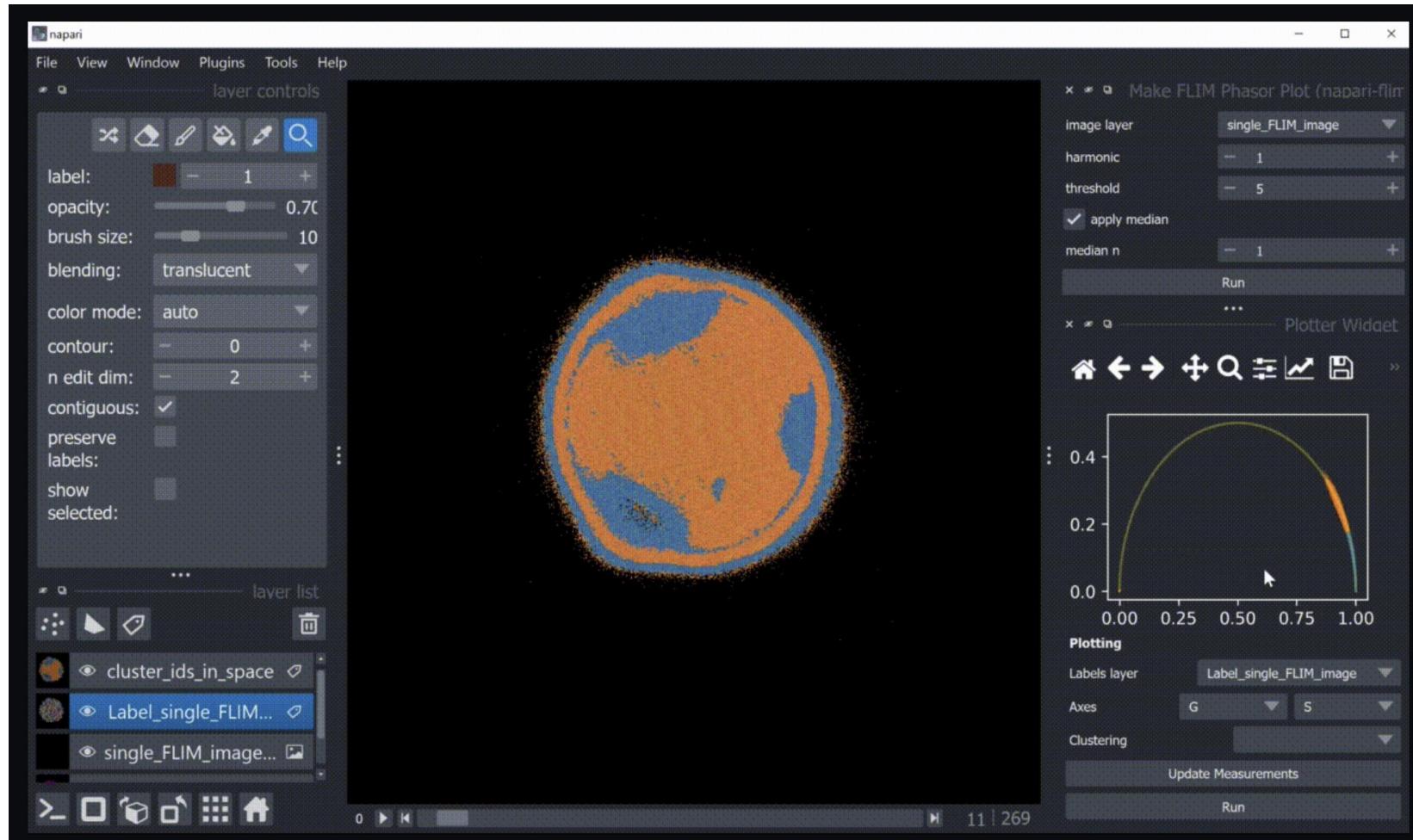
# Napari-flim-phasor-plotter



PoL  
Physics of Life  
TU Dresden



Read raw TCSPC FLIM data and cluster the phasor plot for segmentation

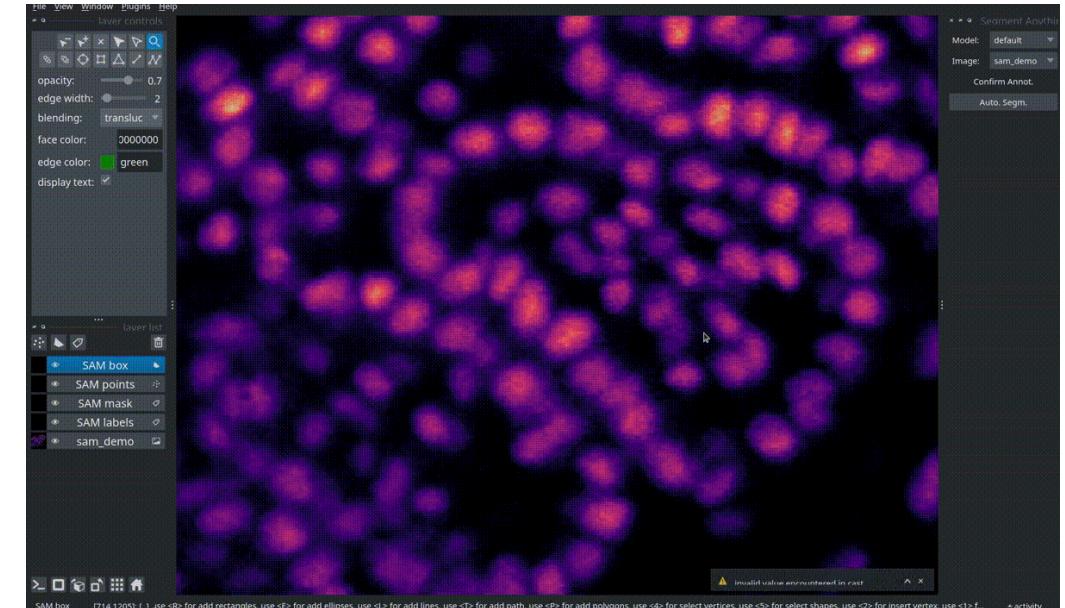


<https://github.com/zoccoler/napari-flim-phasor-plotter>

# Napari-sam plugins



<https://github.com/MIC-DKFZ/napari-sam>

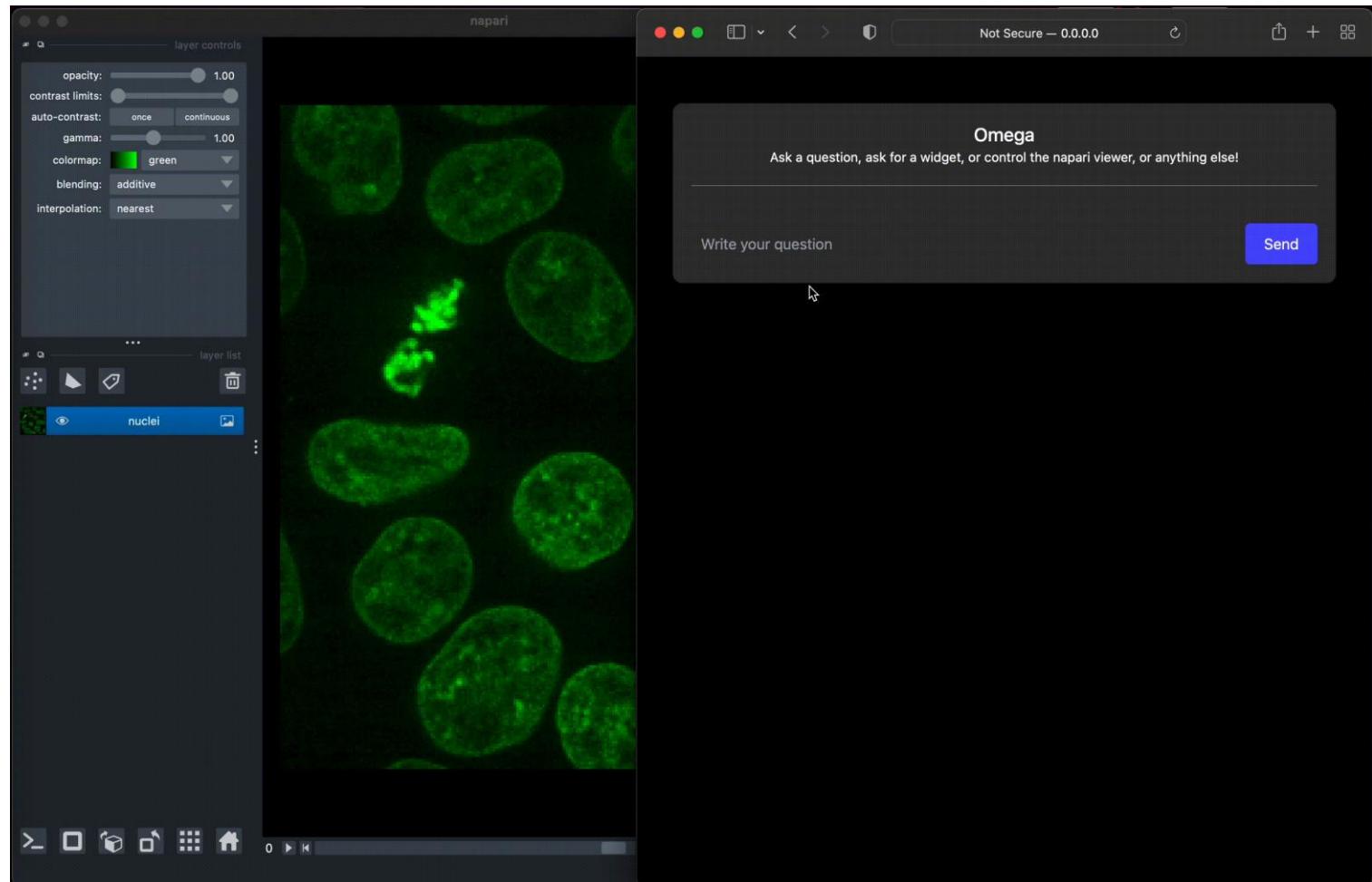


<https://github.com/royerlab/napari-segment-anything>

# chatGPT plugin



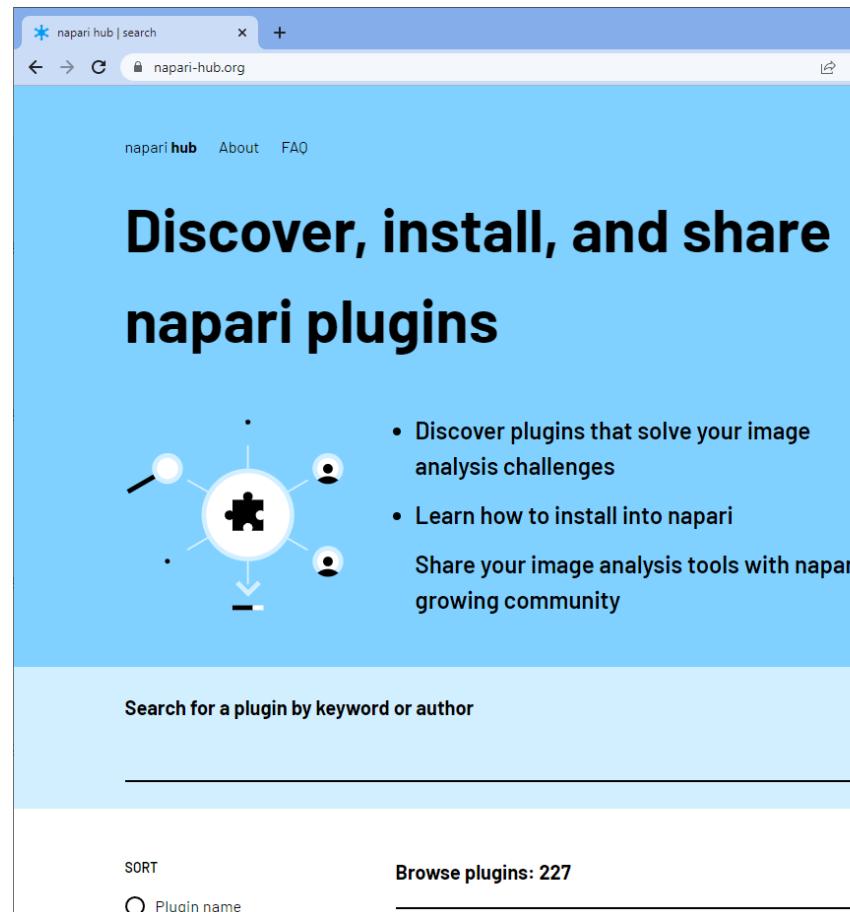
PoL  
Physics of Life  
TU Dresden



<https://github.com/royerlab/napari-chatgpt>

# The Napari Hub

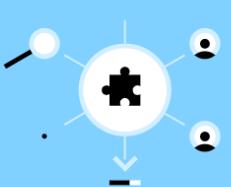
- The plugin you are looking for may be near you!
- Search engine for napari plugins



napari hub | search napari-hub.org

napari hub About FAQ

## Discover, install, and share napari plugins

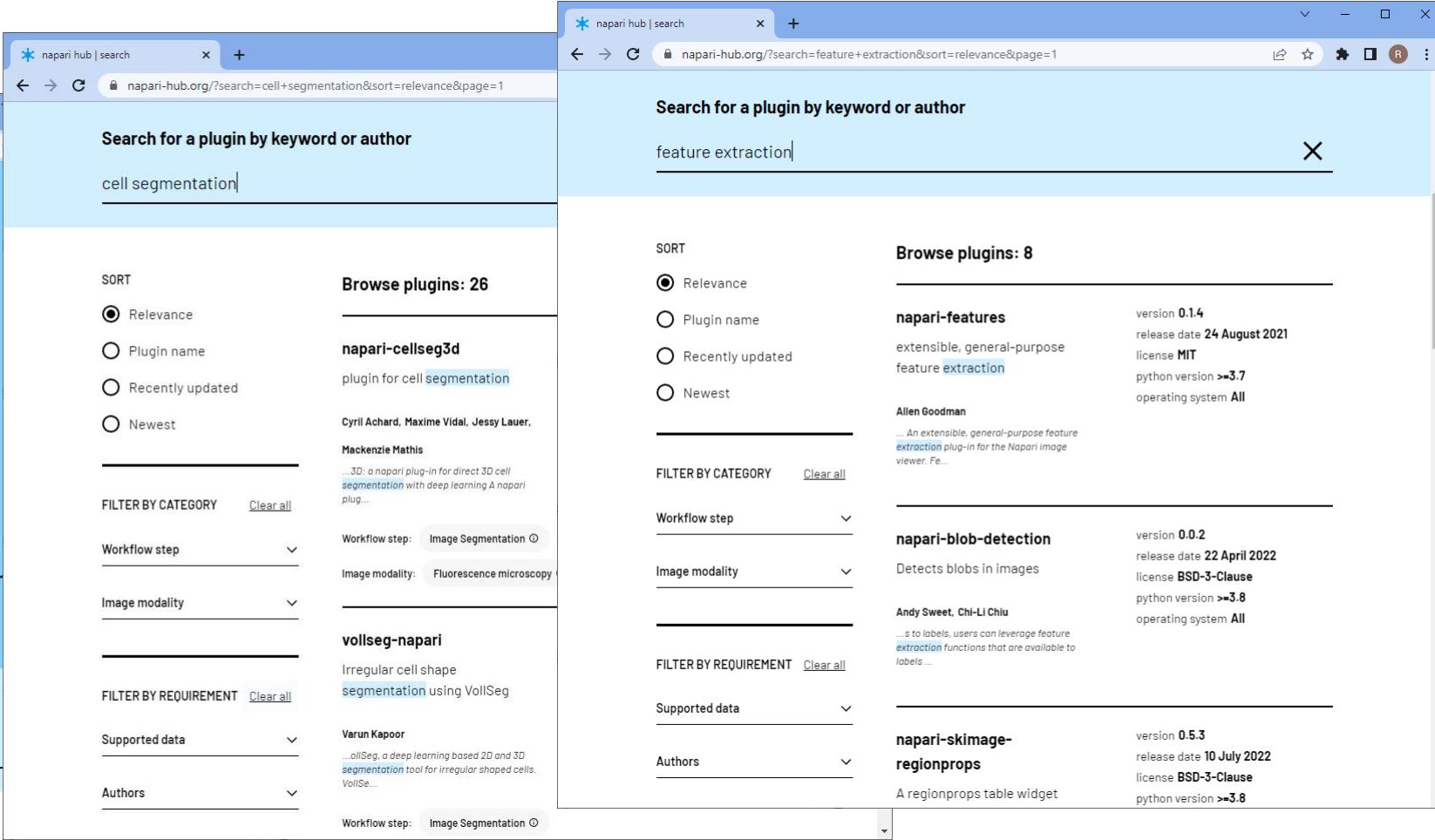


- Discover plugins that solve your image analysis challenges
- Learn how to install into napari
- Share your image analysis tools with napari's growing community

Search for a plugin by keyword or author

SORT  Plugin name

Browse plugins: 227



napari hub | search napari-hub.org/?search=feature+extraction&sort=relevance&page=1

Search for a plugin by keyword or author feature extraction X

SORT  Relevance  Plugin name  Recently updated  Newest

Browse plugins: 8

**napari-features** version 0.1.4 release date 24 August 2021 license MIT python version >=3.7 operating system All

*extensible, general-purpose feature extraction plug-in for the Napari image viewer. Fe...*

**Allen Goodman** ... An extensible, general-purpose feature extraction plug-in for the Napari image viewer. Fe...

**napari-blob-detection** version 0.0.2 release date 22 April 2022 license BSD-3-Clause python version >=3.8 operating system All

*Detects blobs in images*

**Andy Sweet, Chi-Li Chiu** ...s to labels, users can leverage feature extraction functions that are available to labels ...

**napari-skimage-regionprops** version 0.5.3 release date 10 July 2022 license BSD-3-Clause python version >=3.8

*A regionprops table widget*

FILTER BY CATEGORY [Clear all](#)

Workflow step: Image Segmentation

Image modality: Fluorescence microscopy

FILTER BY REQUIREMENT [Clear all](#)

Supported data

Authors

Workflow step: Image Segmentation

<https://www.napari-hub.org/>

# Image Analysis Basics with Python

# Arrays in Python

- Modifying lists entries

```
▶ numbers = [0, 1, 2, 3, 4]  
  
# write in one array element  
numbers[1] = 5  
  
print(numbers)  
  
[0, 5, 2, 3, 4]
```

Note: The first element has index 0!

- Creating lists of defined size



```
▶ zeros = [0] * 10  
print(zeros)
```

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

- Concatenating lists

```
▶ ones = [1, 1, 1]  
twos = [2, 2, 2]  
  
# concatenate arrays  
numbers = ones + twos  
  
print(numbers)
```

[1, 1, 1, 2, 2, 2]

+ means appending



```
▶ # Arrays
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(numbers)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Creating subsets of lists

Start

End

```
▶ subset = numbers[2:4]
print(subset)
```

```
[2, 3]
```

Step

```
▶ subset_with_gaps = arr[1:8:2]
print(subset_with_gaps)
```

```
[1, 3, 5, 7]
```

# data[start:stop:step]

# Indexing, cropping, subsets



- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

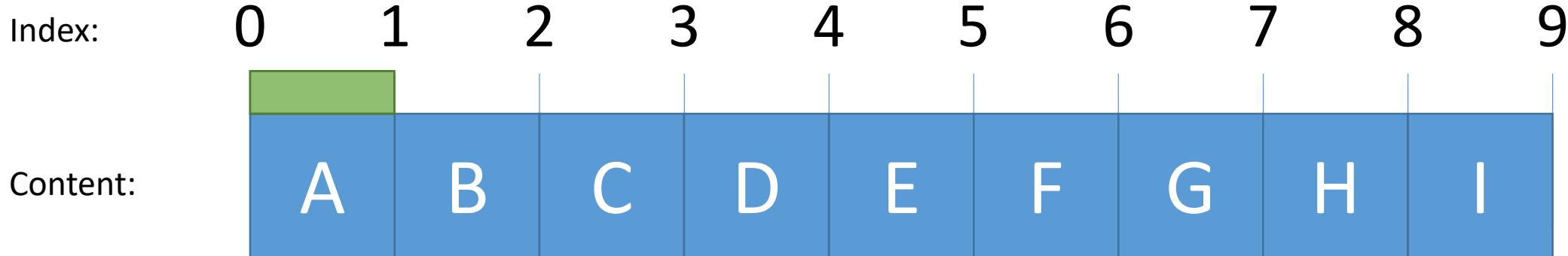
Index:	0	1	2	3	4	5	6	7	8	9
Content:	A	B	C	D	E	F	G	H	I	

# Indexing, cropping, subsets



- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0]
```

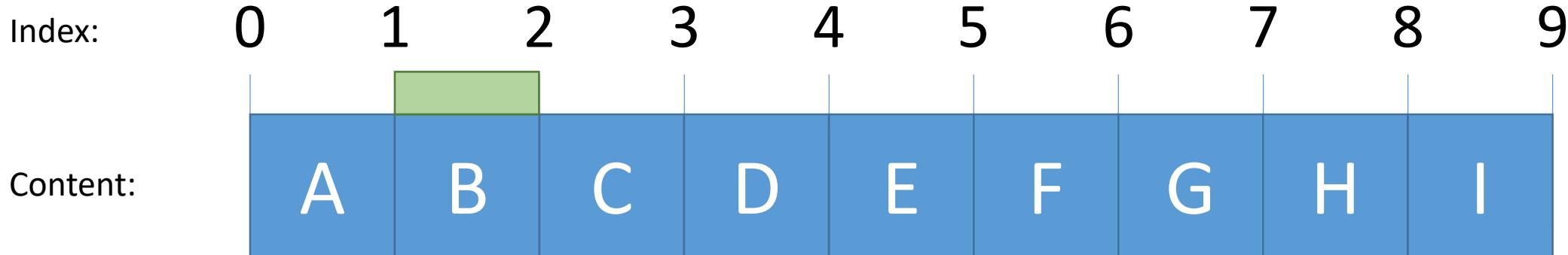
'A'

# Indexing, cropping, subsets



- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



data[0]

'A'

data[1]

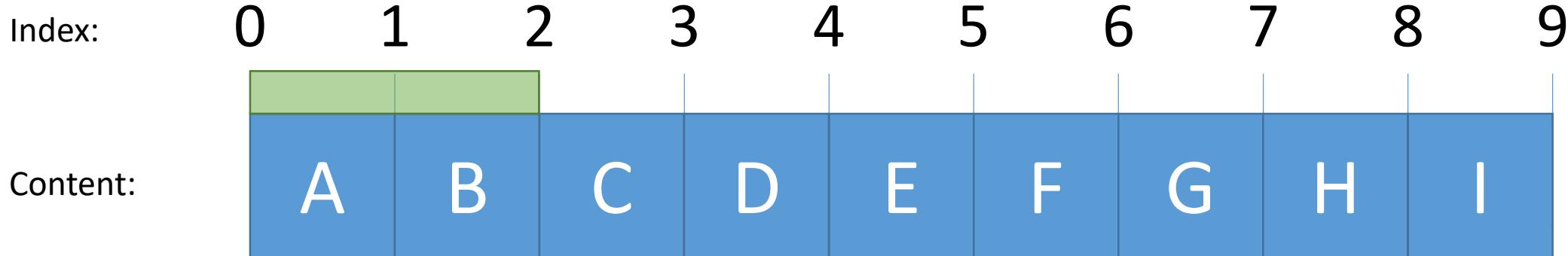
'B'

# Indexing, cropping, subsets



- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



data[0]

'A'

data[1]

'B'

data[0:2]

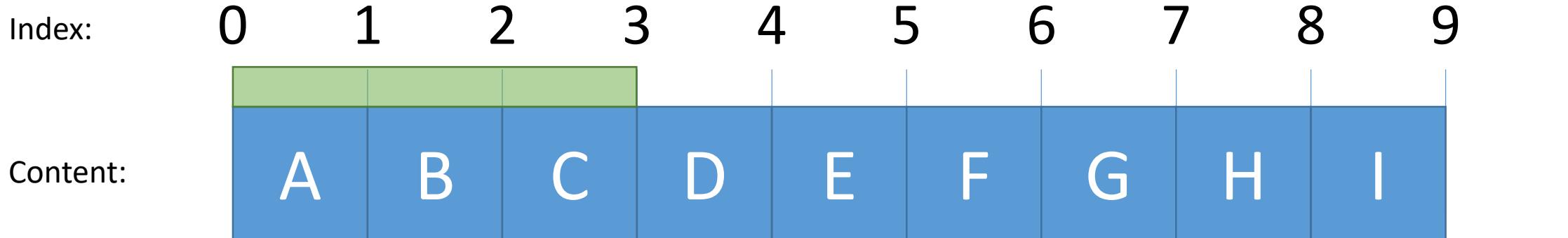
['A', 'B']

# Indexing, cropping, subsets



- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



`data[0]`

`data[1]`

`data[0:2]`

`data[0:3]`

`data[1:2]`

`len(data)`

'A'

'B'

['A', 'B']

['A', 'B', 'C']

['B']

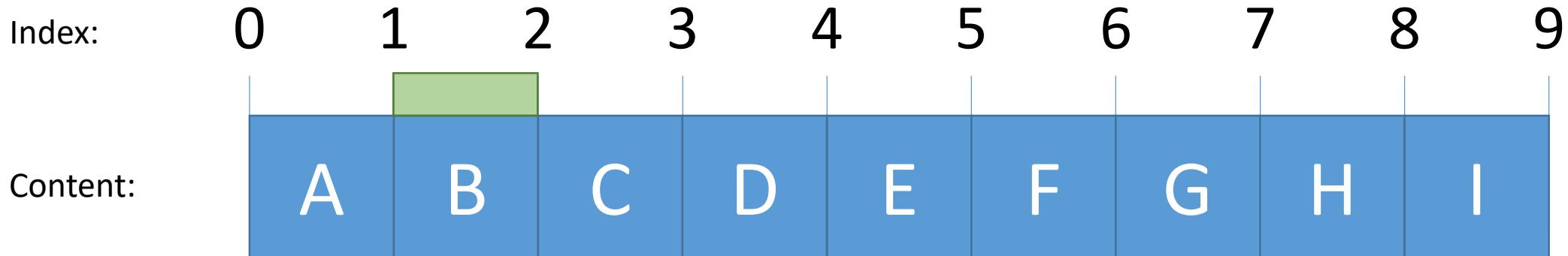
9

# Indexing, cropping, subsets



- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



data[0]

data[1]

data[0:2]

data[0:3]

data[1:2]

'A'

'B'

['A', 'B']

['A', 'B', 'C']

['B']

# Indexing, cropping, subsets

- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



`data[0]`

`data[1]`

`data[0:2]`

`data[0:3]`

`data[1:2]`

`len(data)`

'A'

'B'

['A', 'B']

['A', 'B', 'C']

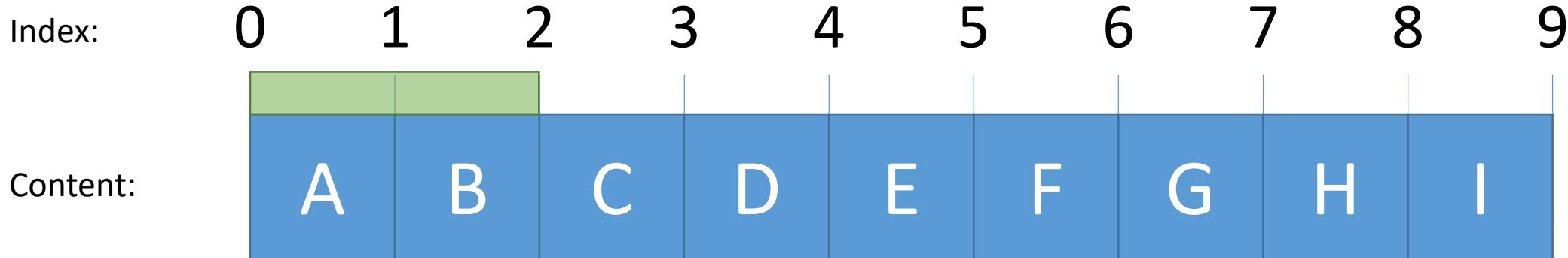
['B']

9

# Indexing, cropping, subsets

- You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



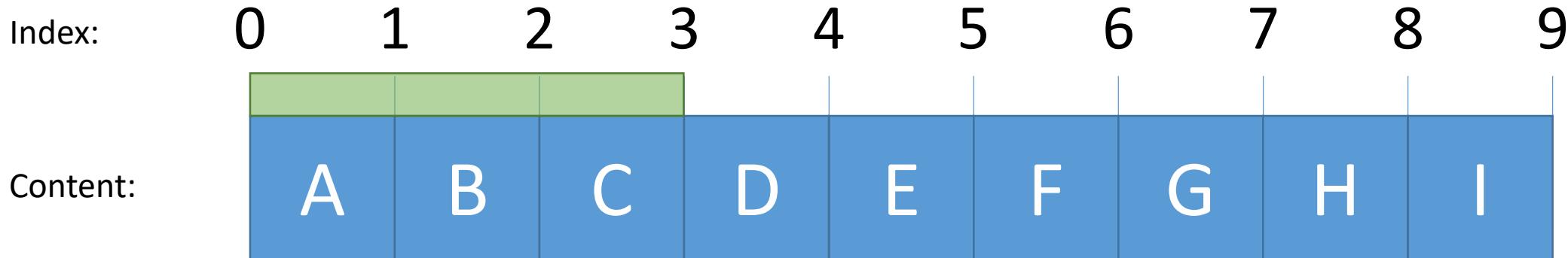
```
data[:2]
```

```
['A', 'B']
```

# Indexing, cropping, subsets

- You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:2]
```

```
data[:3]
```

```
['A', 'B']
```

```
['A', 'B', 'C']
```

# Indexing, cropping, subsets

- You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:2]
```

```
data[:3]
```

```
data[2:]
```

```
['A', 'B']
```

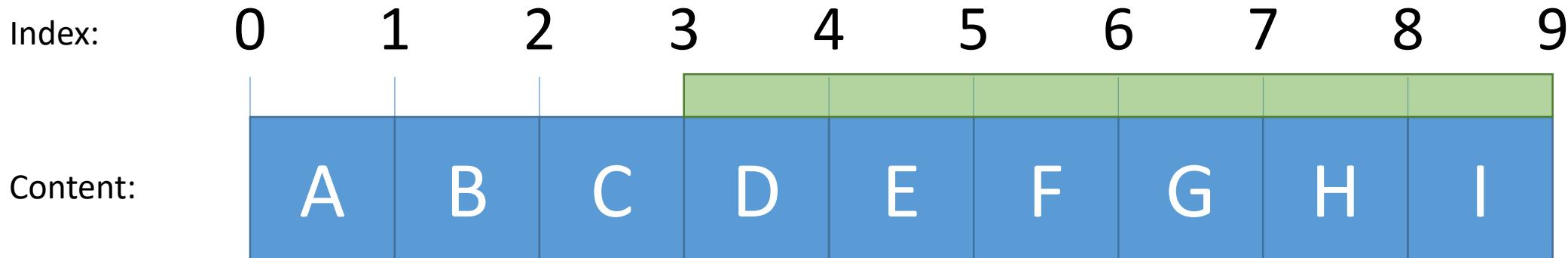
```
['A', 'B', 'C']
```

```
['C', 'D', 'E', 'F', 'G', 'H', 'I']
```

# Indexing, cropping, subsets

- You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



data[:2]

data[:3]

data[2:]

data[3:]

['A', 'B']

['A', 'B', 'C']

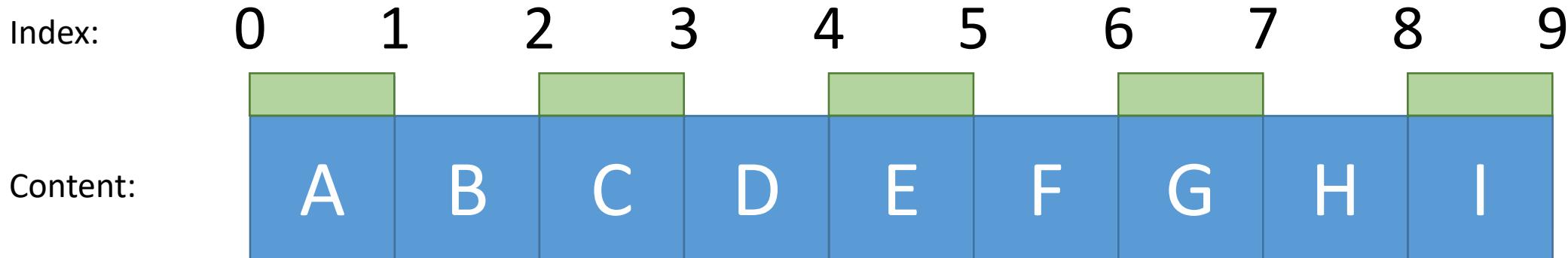
['C', 'D', 'E', 'F', 'G', 'H', 'I']

['D', 'E', 'F', 'G', 'H', 'I']

# Indexing, cropping, subsets

- The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0:10:2]
```

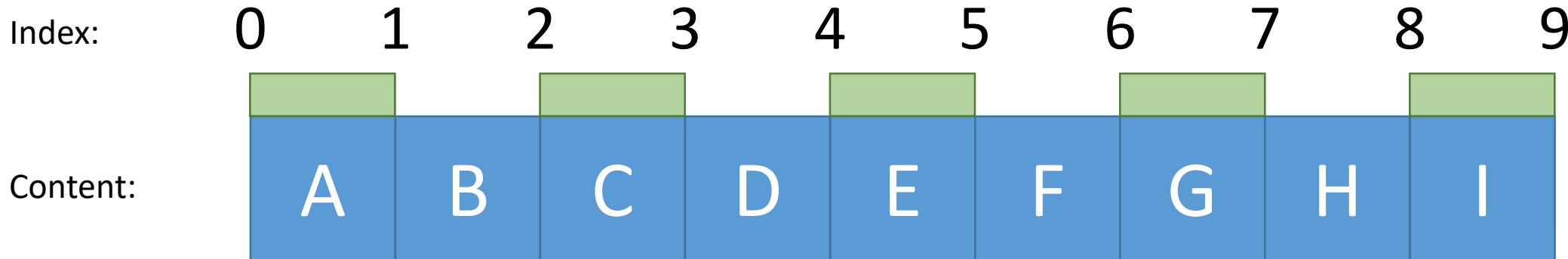
```
['A', 'C', 'E', 'G', 'I']
```

# Indexing, cropping, subsets



- The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0:10:2]
```

```
data[::2]
```

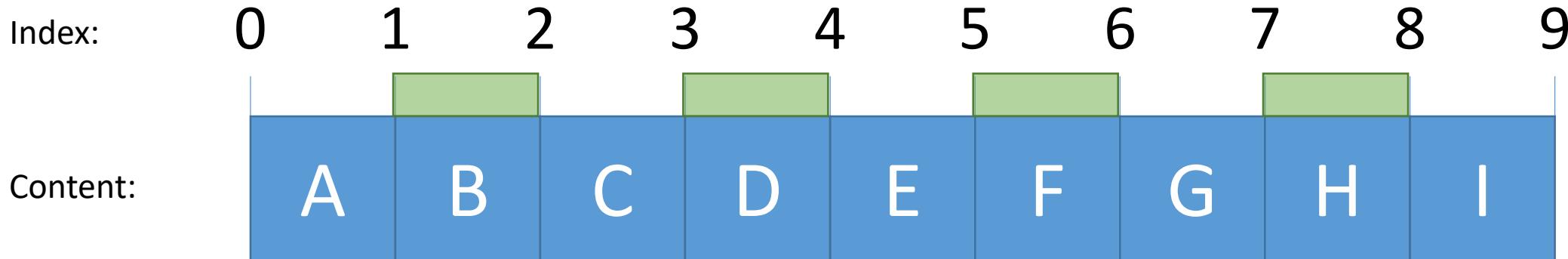
```
['A', 'C', 'E', 'G', 'I'] ['A', 'C', 'E', 'G', 'I']
```

# Indexing, cropping, subsets



- The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0:10:2]
```

```
['A', 'C', 'E', 'G', 'I']
```

```
data[::-2]
```

```
['A', 'C', 'E', 'G', 'I']
```

```
data[1::2]
```

```
['B', 'D', 'F', 'H']
```

# Indexing, cropping, subsets

- Indexing also works with negative indices

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[-2:]
```

```
['H', 'I']
```

# Indexing, cropping, subsets

- Indexing also works with negative indices

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[-2:]
```

```
data[:-2]
```

```
['H', 'I']
```

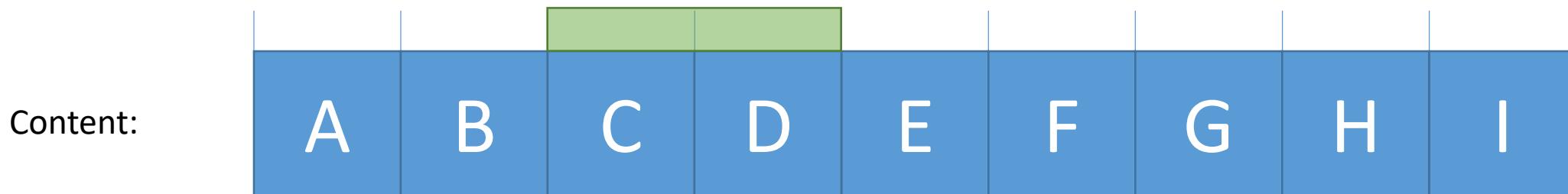
```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

# Indexing, cropping, subsets

- Indexing also works with negative indices

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

Index: -9 -8 -7 -6 -5 -4 -3 -2 -1



```
data[-2:]
```

```
data[: -2]
```

```
data[-7:-5]
```

```
['H', 'I']
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

```
['C', 'D']
```

# Indexing, cropping, subsets

- Indexing also works with negative indices

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

Index: -9 -8 -7 -6 -5 -4 -3 -2 -1

Content:



data[-2:]

data[: -2]

data[-7:-5]

data[-5:-7]

['H', 'I']

['A', 'B', 'C', 'D', 'E', 'F', 'G']

['C', 'D']

[]

# Indexing, cropping, subsets



- Negative stepping also works

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



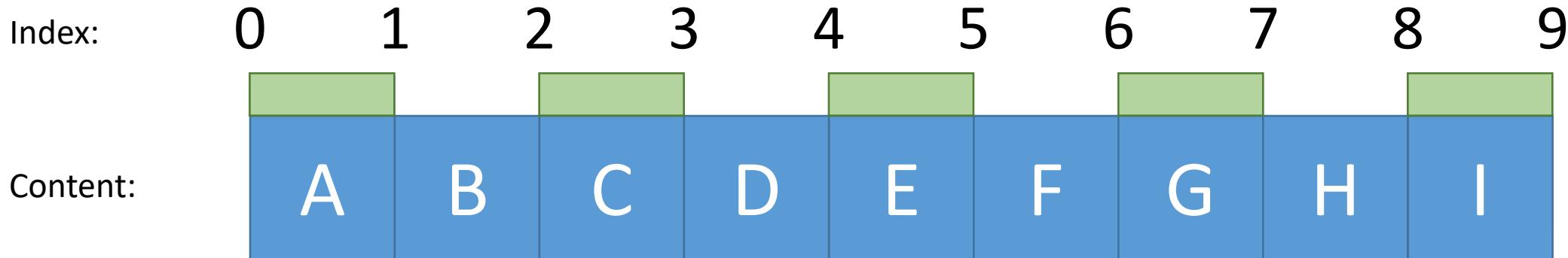
```
data[::-1]
```

```
['I', 'H', 'G', 'F', 'E', 'D', 'C', 'B', 'A']
```

# Indexing, cropping, subsets

- Negative stepping also works

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:: -1]
```

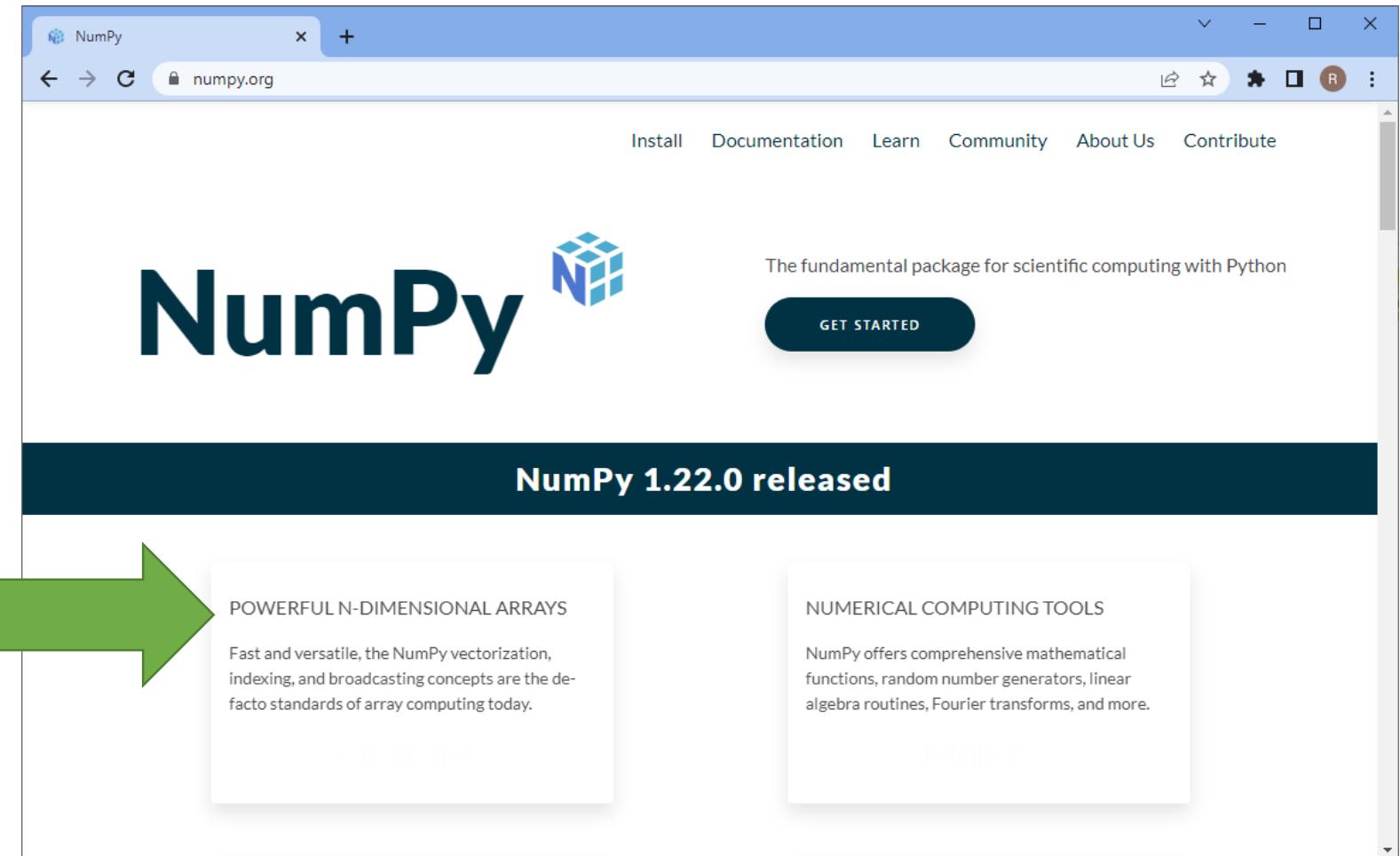
```
['I', 'H', 'G', 'F', 'E', 'D', 'C', 'B', 'A']
```

```
data[:: -2]
```

```
['I', 'G', 'E', 'C', 'A']
```

- The fundamental package for scientific computing with python.

- `conda install numpy`



- Simplifying mathematical operations on n-dimensional arrays
- Python arrays of arrays (lists of lists)

```
# multidimensional arrays
matrix = [
    [1, 2, 3],
    [2, 3, 4],
    [3, 4, 5]
]

print(matrix)
```

[[1, 2, 3], [2, 3, 4], [3, 4, 5]]

```
result = matrix * 2
print(result)
```

[[1, 2, 3], [2, 3, 4], [3, 4, 5], [1, 2, 3], [2, 3, 4], [3, 4, 5]]

- numpy arrays

```
import numpy as np

np_matrix = np.asarray(matrix)

print(np_matrix)
```

[[1 2 3]  
 [2 3 4]  
 [3 4 5]]

```
np_result = np_matrix * 2
print(np_result)
```

[[ 2 4 6]  
 [ 4 6 8]  
 [ 6 8 10]]

Tell python that  
you want to use  
a library called  
numpy

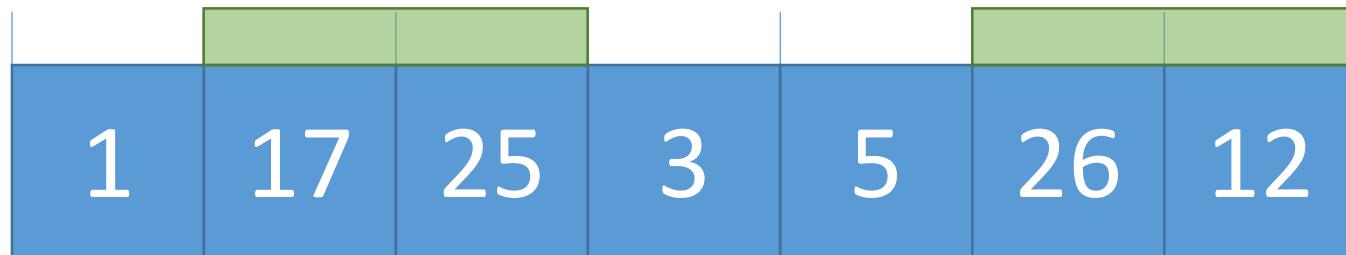
If “numpy” is too  
long, you can  
give an alias “np”

- “Masking” is addressing certain elements in numpy arrays, e.g. depending on their content

```
import numpy
measurements = numpy.asarray([1, 17, 25, 3, 5, 26, 12])
measurements
```

```
array([ 1, 17, 25,  3,  5, 26, 12])
```

Content:



```
mask = measurements > 10
```

```
mask
```

```
array([False,  True,  True, False, False,  True,  True])
```

```
measurements[mask]
```

```
array([17, 25, 26, 12])
```

# Basic descriptive statistics using numpy

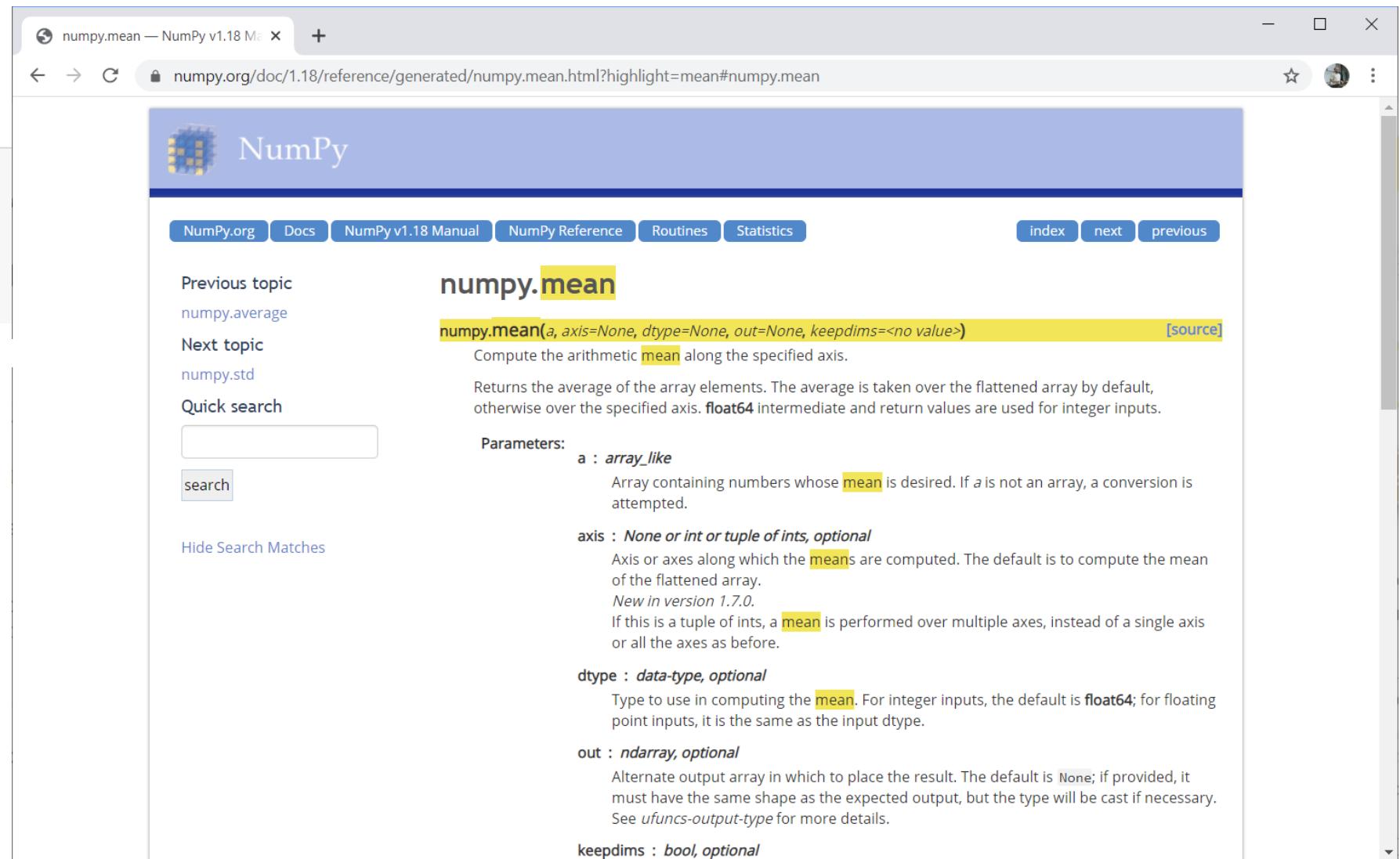
- Basic descriptive statistics

```
import numpy as np

measurements = [1, 4, 6, 7, 2]

mean = np.mean(measurements)
print("Mean: " + str(mean))

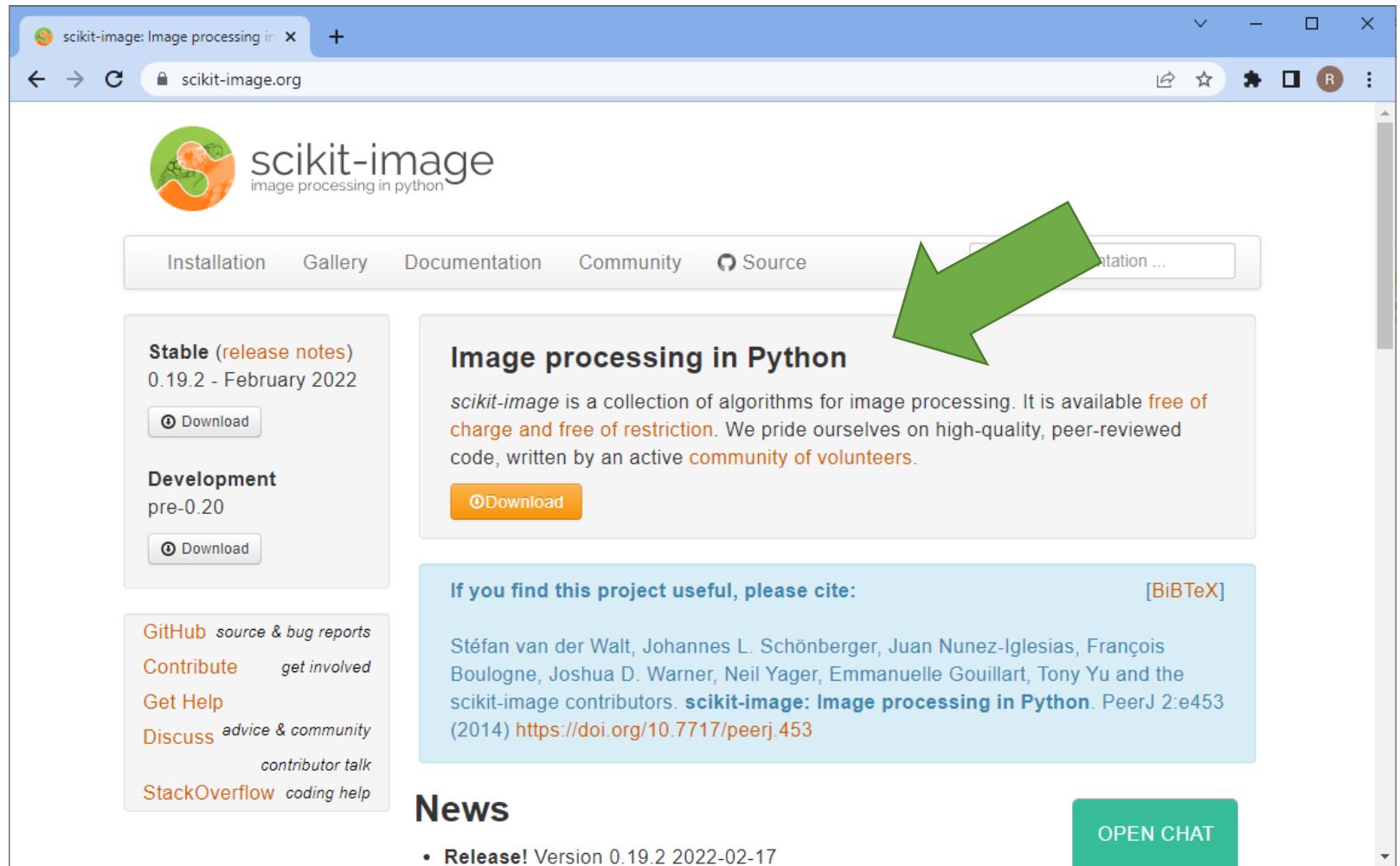
Mean: 4.0
```



The screenshot shows a web browser window displaying the NumPy documentation for the `numpy.mean` function. The URL in the address bar is `numpy.org/doc/1.18/reference/generated/numpy.mean.html?highlight=mean#numpy.mean`. The page title is "NumPy". The main content area is titled "numpy.mean" and contains the function signature `numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)` and a link to "[source]". Below the signature, it says "Compute the arithmetic mean along the specified axis." It then describes the behavior for flattened arrays and over multiple axes. The "Parameters" section includes detailed descriptions for `a`, `axis`, `dtype`, `out`, and `keepdims`. On the left sidebar, there are links to "Previous topic" (numpy.average), "Next topic" (numpy.std), and a "Quick search" input field.

- *scikit-image* is a collection of algorithms for image processing.

conda install scikit-image



The screenshot shows the official website for scikit-image (<https://scikit-image.org/>). The page features a header with the scikit-image logo and navigation links for Installation, Gallery, Documentation, Community, and Source. A large green arrow points to the "Image processing in Python" section, which contains a brief description of the project and a "Download" button. To the left, there are sections for Stable (release notes 0.19.2 - February 2022) and Development (pre-0.20), along with links to GitHub, Contribute, Get Help, Discuss, and StackOverflow. A "News" section at the bottom right mentions a release of Version 0.19.2 on 2022-02-17, and a "OPEN CHAT" button is also present.

# Tables

- Sneak preview: By the mid of the course, we will work with Pandas DataFrames, *fancy* Tables.

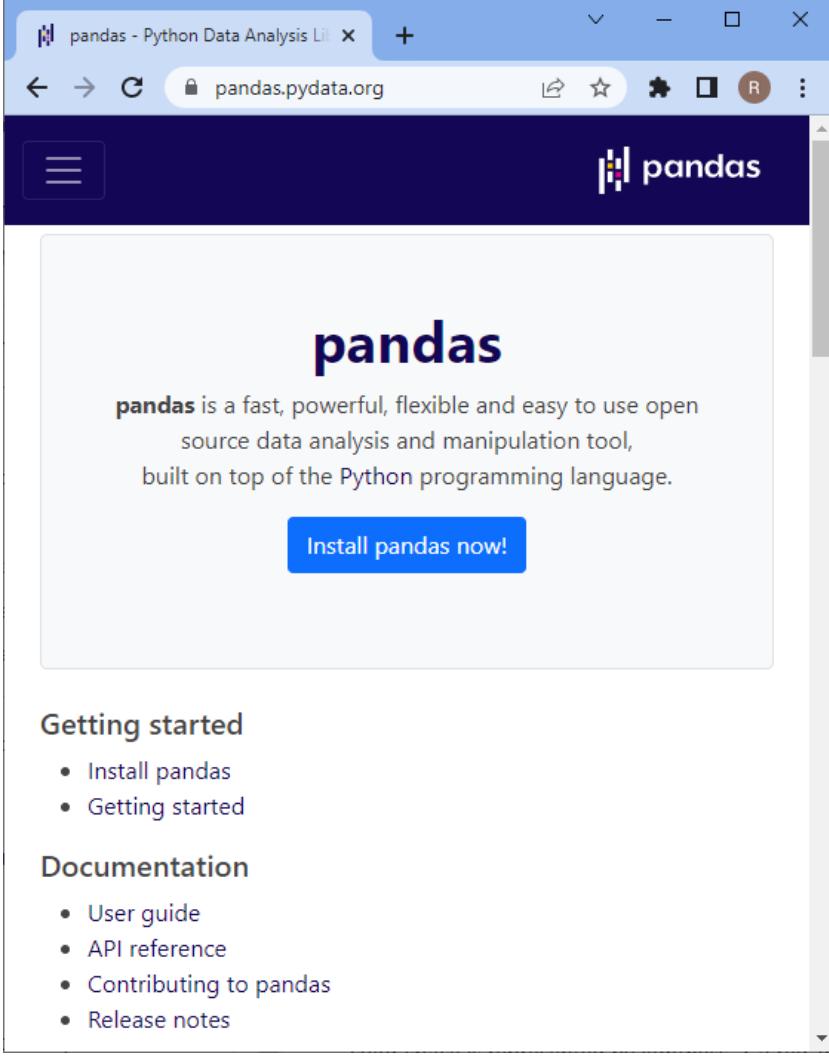
- conda install pandas

- Among many other features, Pandas allows to visualize tables nicely in Jupyter notebooks.

```
import pandas

pandas.DataFrame(measurements_week)
```

	Monday	Tuesday	Wednesday	Thursday	Friday
0	2.3	1.8	4.5	1.9	4.4
1	3.1	7.0	1.5	2.0	2.3
2	5.6	4.3	3.2	6.4	5.4

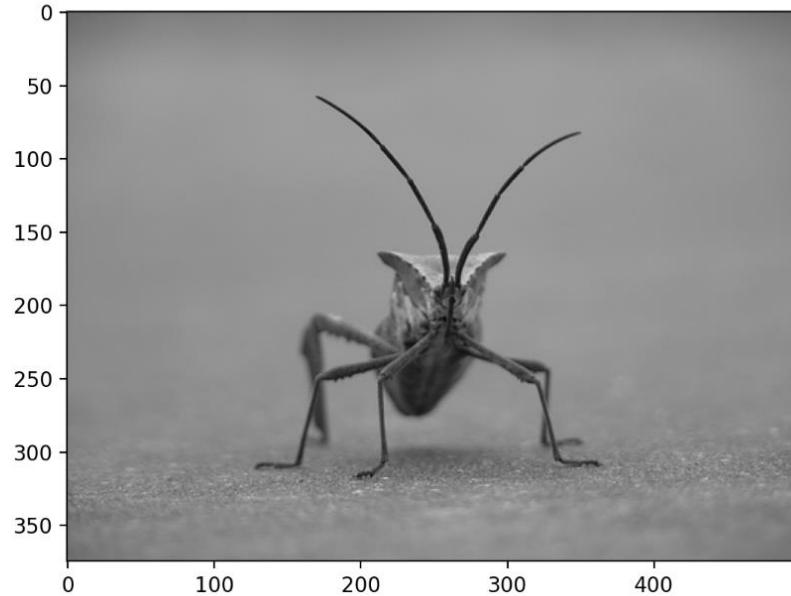


The screenshot shows the official pandas website at [pandas.pydata.org](https://pandas.pydata.org). The page features a dark header with the pandas logo and navigation links. Below the header, there's a large callout for "pandas" with a description: "pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language." A blue button labeled "Install pandas now!" is visible. On the right side, there are two sections: "Getting started" with links to "Install pandas" and "Getting started", and "Documentation" with links to "User guide", "API reference", "Contributing to pandas", and "Release notes". At the bottom of the page, there's a "Footer" section with links to "About", "Code of conduct", "Contributing", "Funding", "Privacy policy", and "Sitemap".

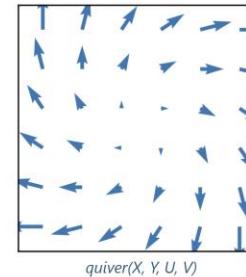
- *matplotlib* is the standard python library for plotting data.

conda install matplotlib

```
imgplot = plt.imshow(img)
```



Plot types Examples Tutorials Reference User guide Develop Releases



## Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

Try Matplotlib (on Binder) →

## Examples

This page contains example plots. Click on any image to see the full image and source code.

For longer tutorials, see our tutorials page. You can also find external resources and a FAQ in our user guide.

## Lines, bars and markers



<https://matplotlib.org/>



@zoccolermarcelo

- Open images

```
from skimage.io import imread  
  
image = imread("blobs.tif")
```

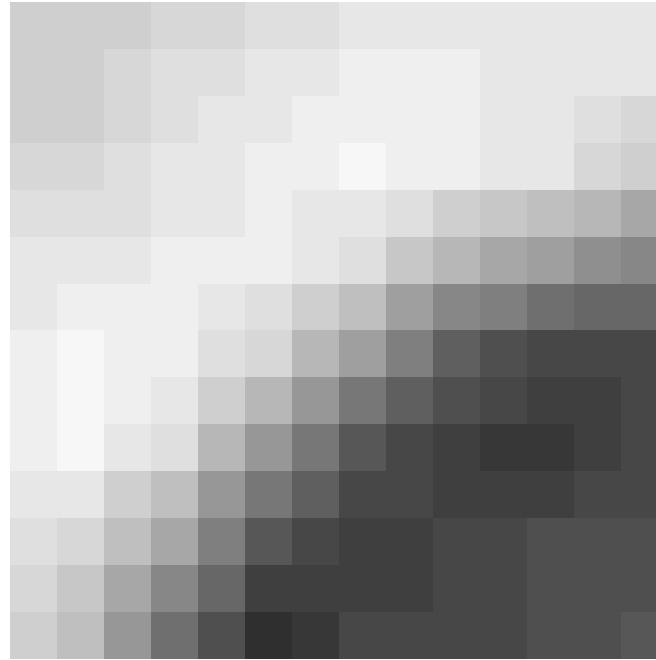
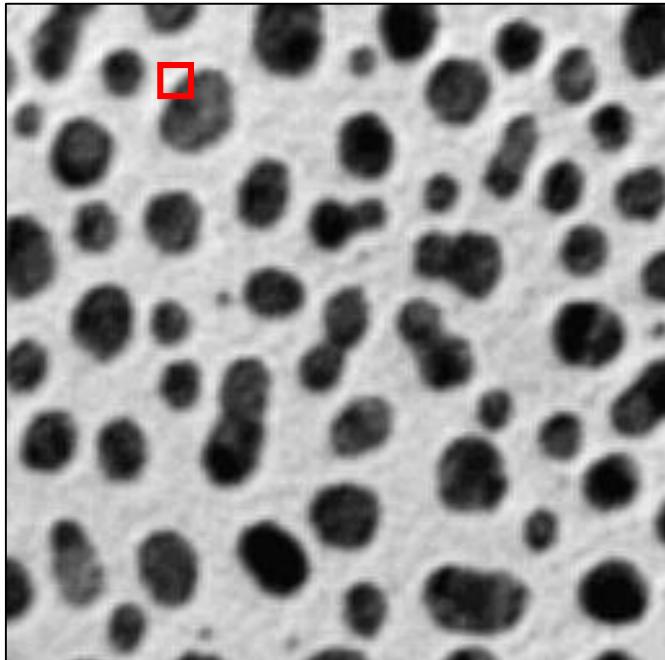
```
image
```

```
array([[ 40,  32,  24, ..., 216, 200, 200],  
       [ 56,  40,  24, ..., 232, 216, 216],  
       [ 64,  48,  24, ..., 240, 232, 232],  
       ...,  
       [ 72,  80,  80, ..., 48,  48,  48],  
       [ 80,  80,  80, ..., 48,  48,  48],  
       [ 96,  88,  80, ..., 48,  48,  48]], dtype=uint8)
```

Images are *just* multi-dimensional arrays or “arrays of arrays”.

# Images and pixels

- An image is just a matrix of numbers
- Pixel: “picture element”
- The edges between pixels are an artefact. In reality, they don’t exist!

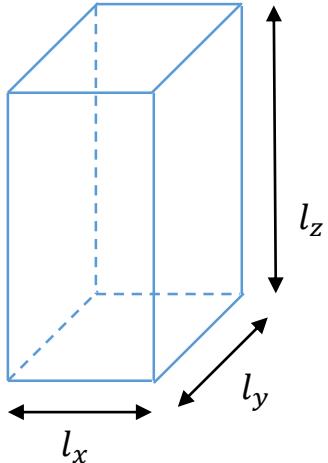


48	48	48	40	40	32	32	24	24	24	24	24	24	24	24
48	48	40	32	32	24	24	16	16	16	16	24	24	24	24
48	48	40	32	24	24	16	16	16	16	16	24	24	32	40
40	40	32	24	24	16	16	16	8	16	16	24	24	40	48
32	32	32	24	24	16	24	24	32	48	56	64	72	88	
24	24	24	16	16	16	24	32	56	72	88	96	112	120	
24	16	16	16	24	32	48	64	96	120	128	144	152	152	
16	8	16	16	32	40	72	96	128	160	176	184	184	184	
16	8	16	24	48	72	104	136	160	176	184	192	192	192	184
24	24	48	64	104	136	160	184	184	192	192	192	184	184	184
32	40	64	88	128	168	184	192	192	184	184	176	176	176	176
40	56	88	120	152	192	192	192	184	184	184	176	176	176	176
48	64	104	144	176	208	200	184	184	184	184	176	176	176	168



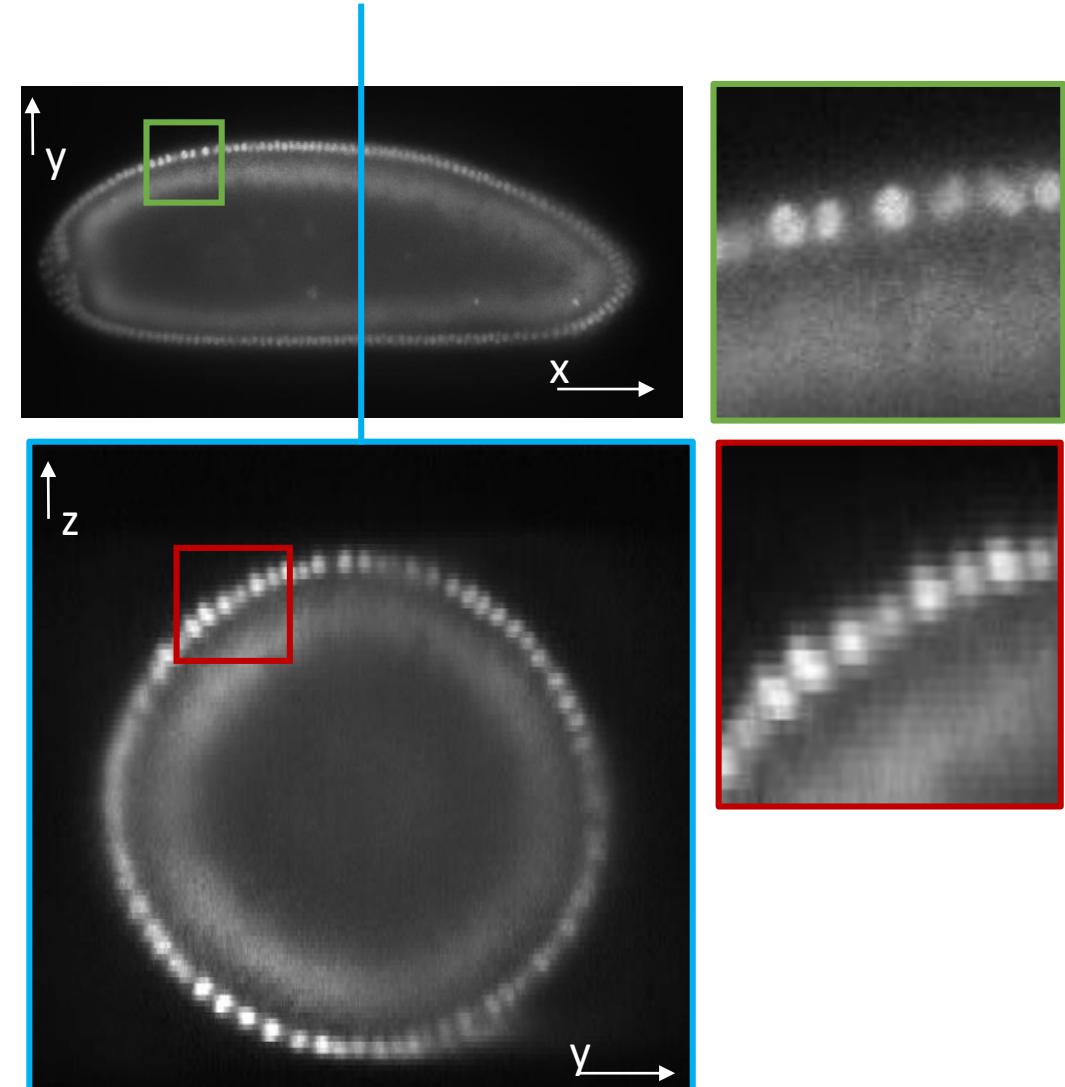
- Voxel: “Volume element”, usually anisotropic

- An-iso-tropy, from Greek:
  - ἀν- (not)
  - ἴσος *isos* (*equal*)
  - τρόπος *tropos* (*rotation, direction*)
- *Not the same in all directions*
- Usually in 3D image processing:



$$l_x = l_y \neq l_z$$

- Image analysts *love* to have isotropic voxels, but it's often not possible.

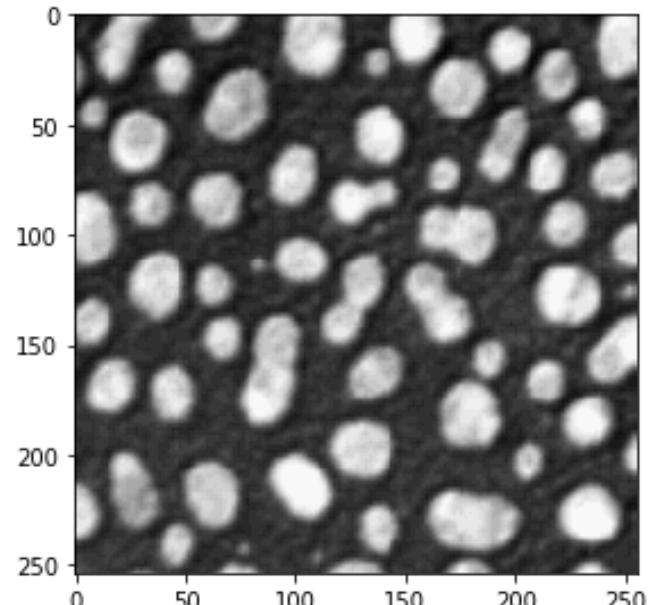


# Working with images in python

- Open images
- Visualize images

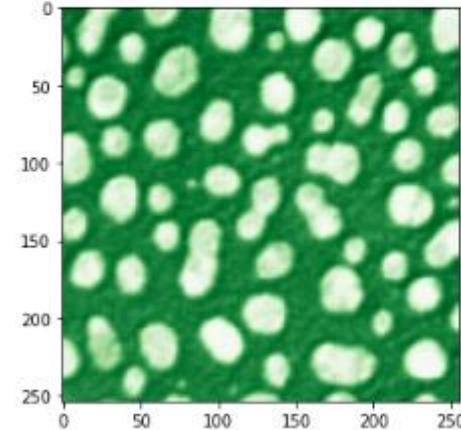
```
from skimage.io import imread  
  
image = imread("blobs.tif")
```

```
from skimage.io import imshow  
  
imshow(image)  
  
<matplotlib.image.AxesImage at 0x245e7e>
```



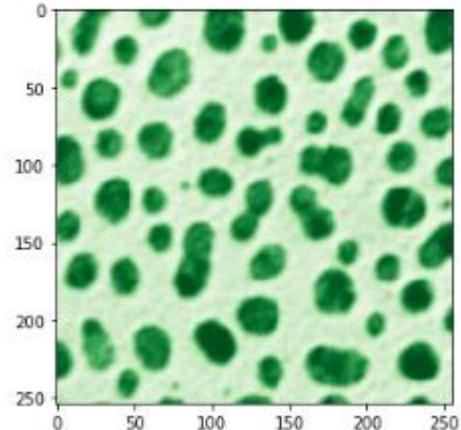
```
imshow(image, cmap="Greens_r")
```

```
<matplotlib.image.AxesImage at 0: <matplotlib.image.AxesImage at 0>
```



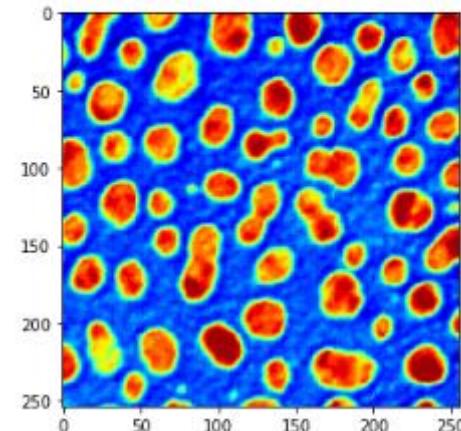
```
imshow(image, cmap="Greens")
```

```
<matplotlib.image.AxesImage at 0: <matplotlib.image.AxesImage at 0>
```



```
imshow(image, cmap="jet")
```

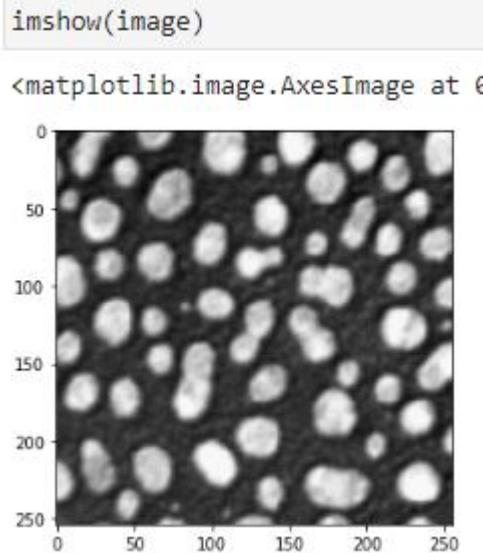
```
<matplotlib.image.AxesImage at 0: <matplotlib.image.AxesImage at 0>
```



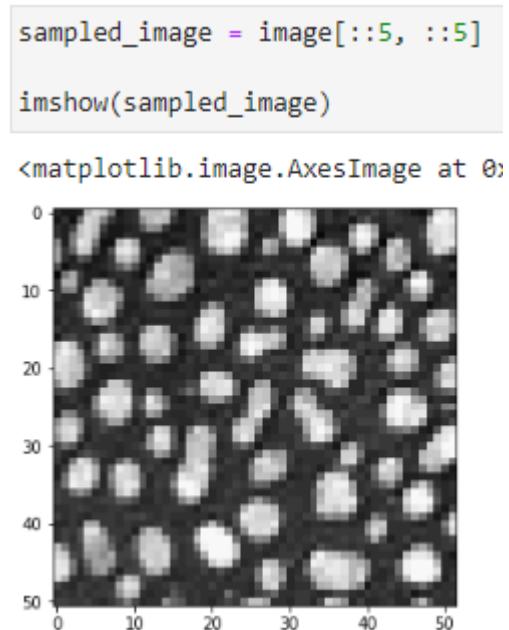
This does not modify the image data. The images are just shown with different colors representing the same values.

# Cropping, sampling and flipping images

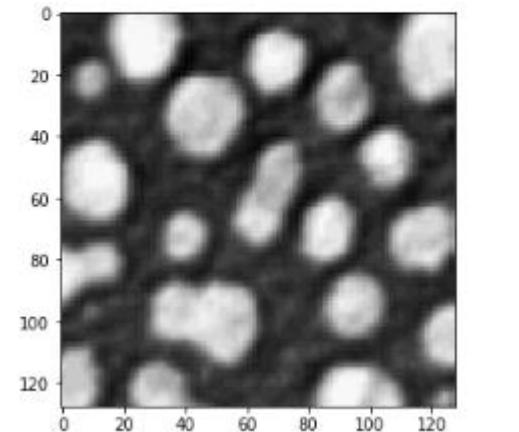
- Indexing and cropping *numpy-arrays* works like with python arrays.



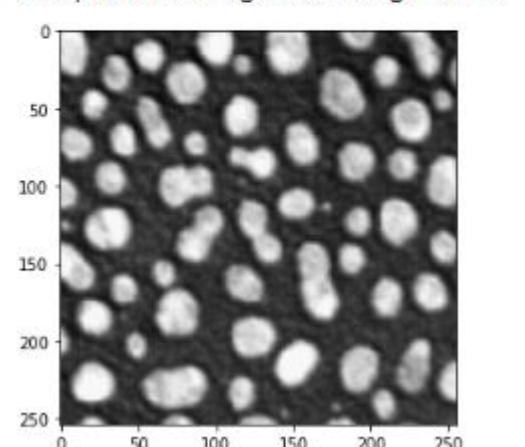
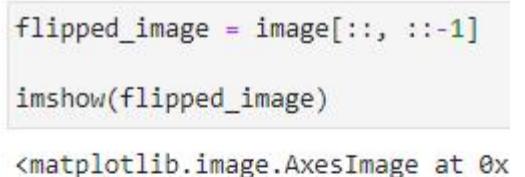
Original image



Sub-sampled image



Cropped image



Flipped image

# Napari and Jupyter Notebooks

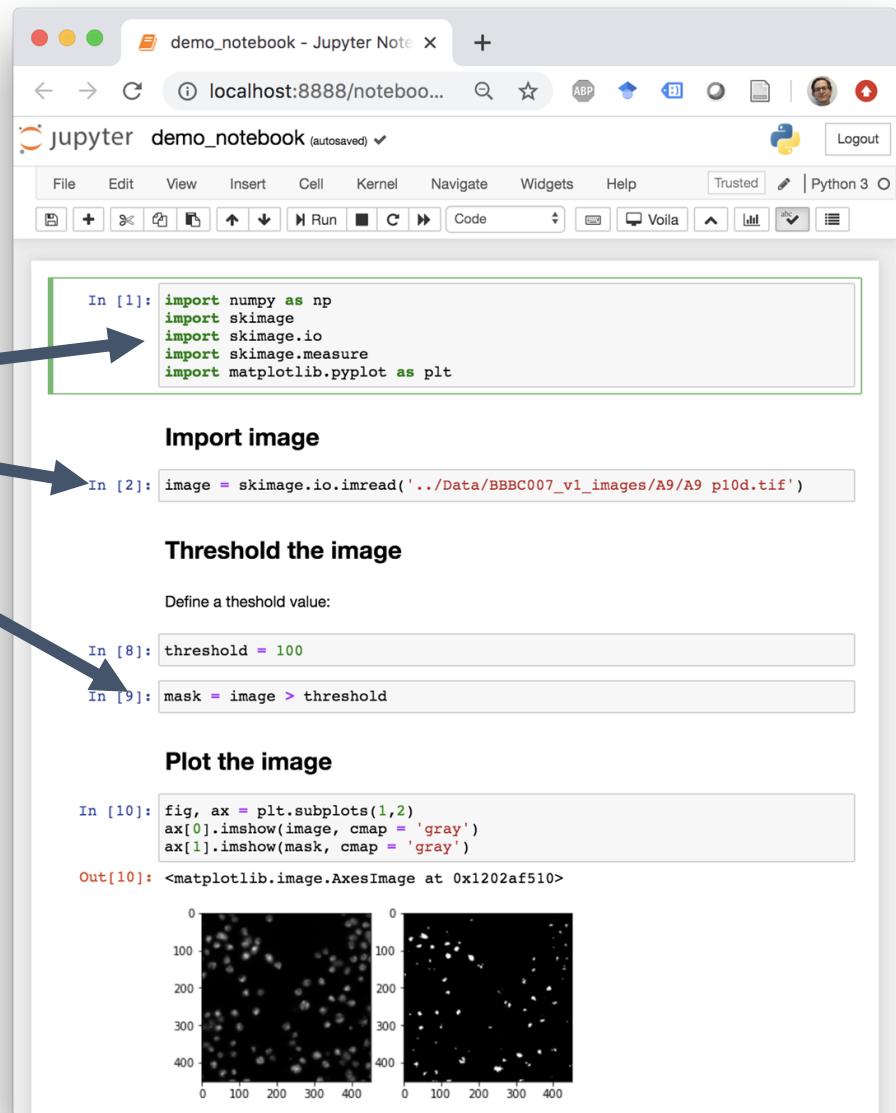
# Napari from jupyter notebooks

What is a jupyter notebook?

A text file (easily sent around)

Rendered by Jupyter in the browser

Split into sections called cells



The screenshot shows a Jupyter Notebook interface running in a browser window titled "demo\_notebook - Jupyter Note". The notebook contains several code cells and their corresponding outputs:

- In [1]:**

```
import numpy as np
import skimage
import skimage.io
import skimage.measure
import matplotlib.pyplot as plt
```

**Import image**
- In [2]:**

```
image = skimage.io.imread('../Data/BBBC007_v1_images/A9/A9_p10d.tif')
```

**Threshold the image**
- In [8]:**

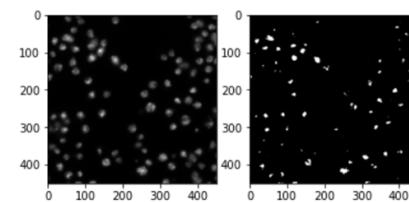
```
threshold = 100
```
- In [9]:**

```
mask = image > threshold
```

**Plot the 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>



# Napari from jupyter notebooks

What is a jupyter notebook?

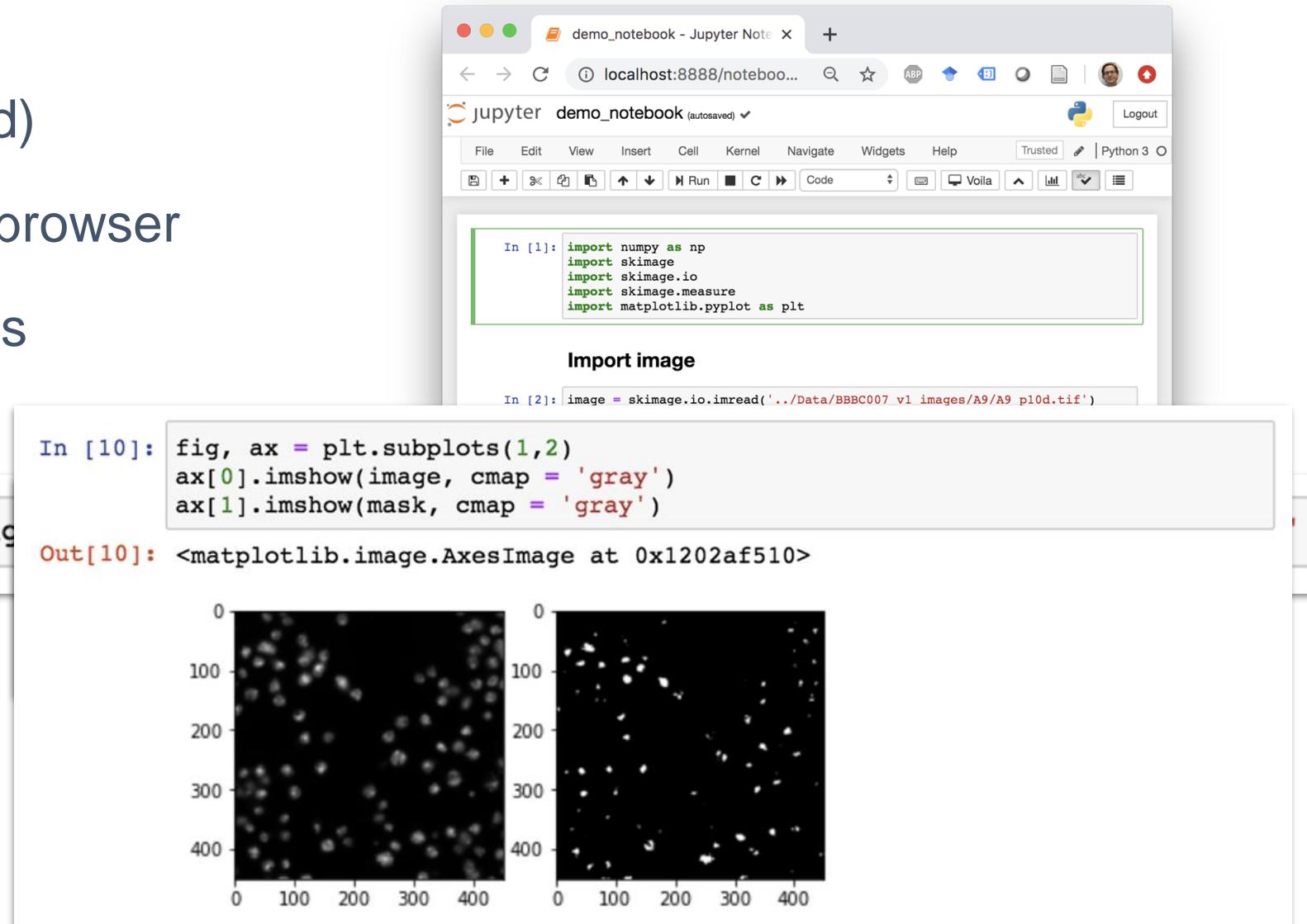
A text file (easily sent around)

Rendered by Jupyter in the browser

Split into sections called cells

Cells can contain:

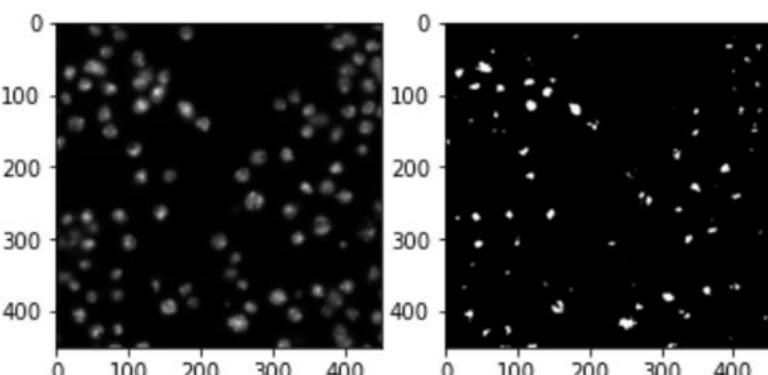
- Code
- Formatted text
- Rich output



The screenshot shows a Jupyter Notebook interface running in a web browser. The top cell, labeled 'In [1]', contains Python code for importing numpy, skimage, and matplotlib.pyplot, along with a descriptive comment 'Import image'. The bottom cell, labeled 'In [2]', contains code to read an image from a file path and displays its output as two grayscale plots. The first plot shows a noisy image, and the second plot shows a mask or segmented version of it.

```
In [1]: import numpy as np
import skimage
import skimage.io
import skimage.measure
import matplotlib.pyplot as plt
Import image
In [2]: image = skimage.io.imread('../Data/BBBC007_v1_images/A9/A9_p10d.tif')
```

```
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?

---

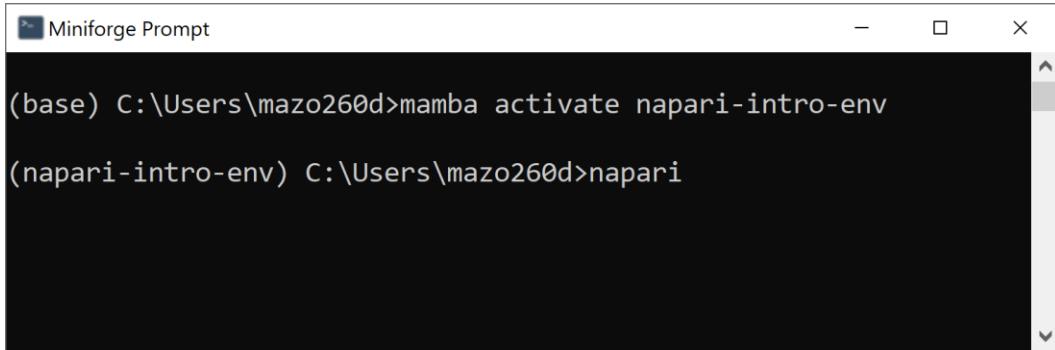
- Keep all the benefits from using code:
  - Batch processing
  - Running python functions/tools still unavailable as plugins

# 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

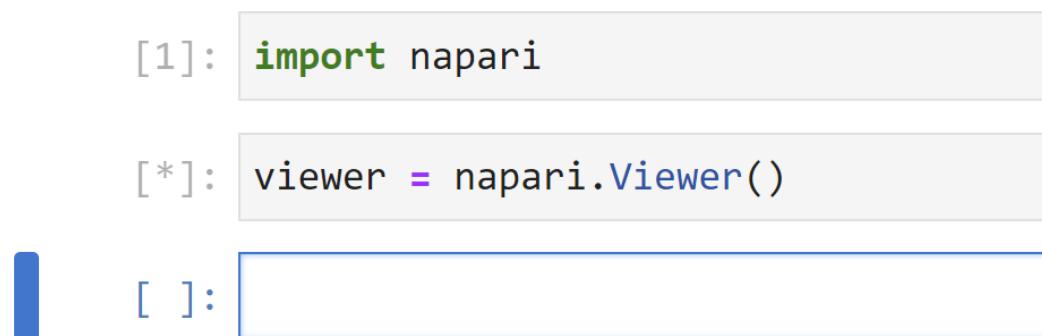
# Opening napari

- From command line:

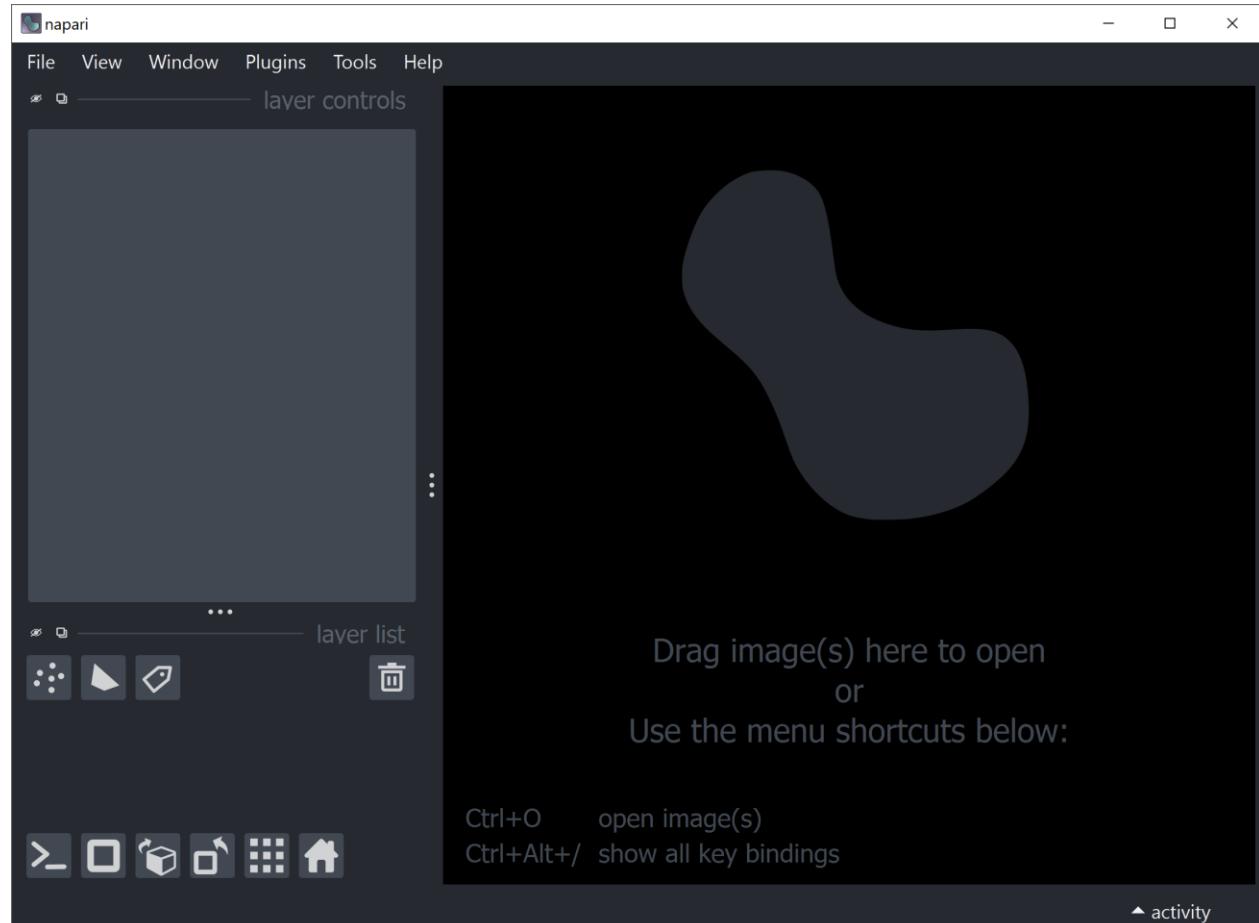


```
Miniforge Prompt
(base) C:\Users\mazo260d>mamba activate napari-intro-env
(napari-intro-env) C:\Users\mazo260d>napari
```

- From code (jupyter notebook):



```
[1]: import napari
[*]: viewer = napari.Viewer()
[ ]:
```



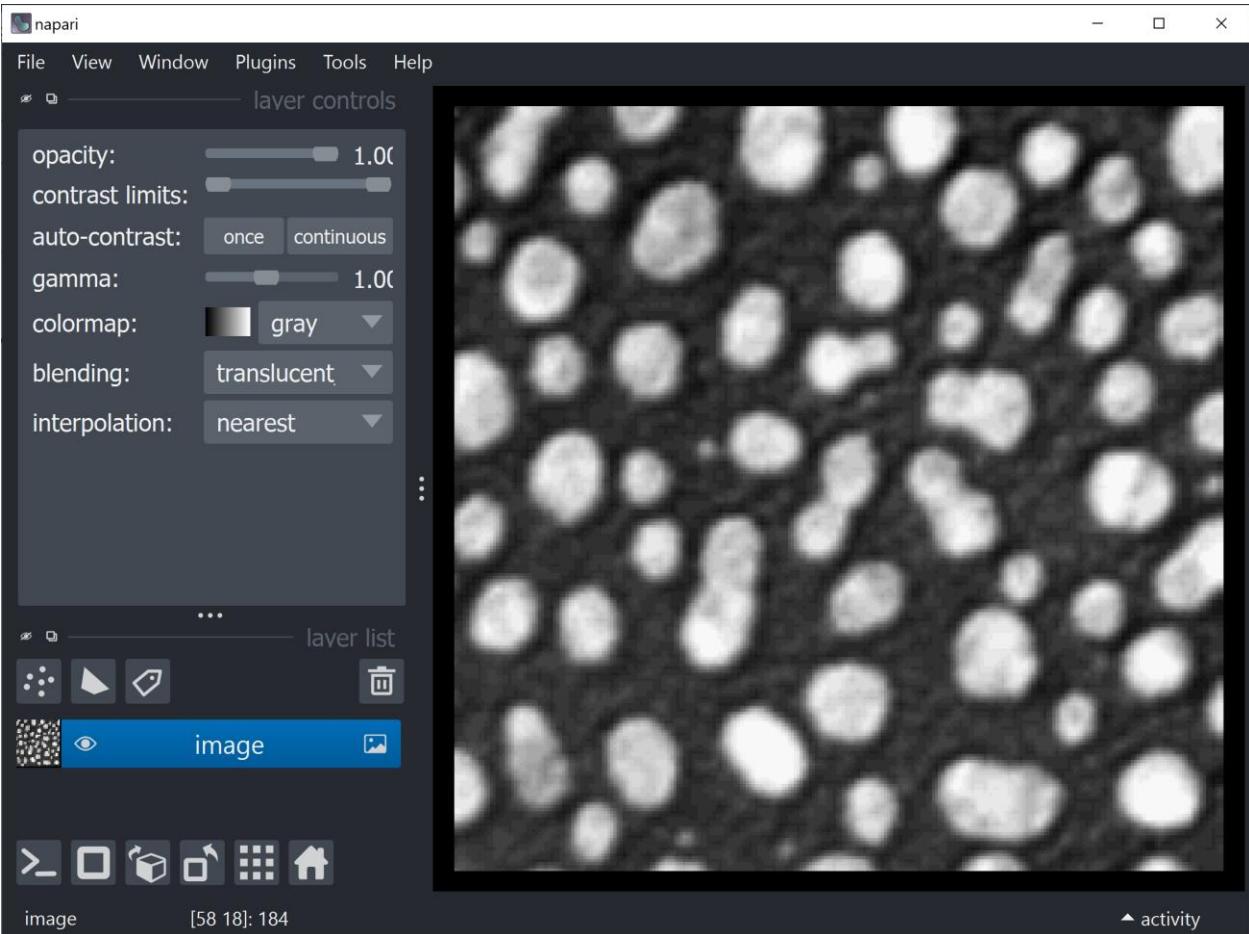
# Opening Images in napari

- Drag and drop
- File > Open File...
- From code:
  - Open an image with an “imread” function

```
image = imread("image.tif")
```

- Load the image to the viewer

```
viewer.add_image(image)
```



# Scripting napari in notebooks

- Add layers to napari to visualize intermediate processing results on top of each other or side by side.
- Change layer visualization within napari...

... or via code in a jupyter notebook:

```
viewer.layers[0].contrast_limits  
[0, 255]
```

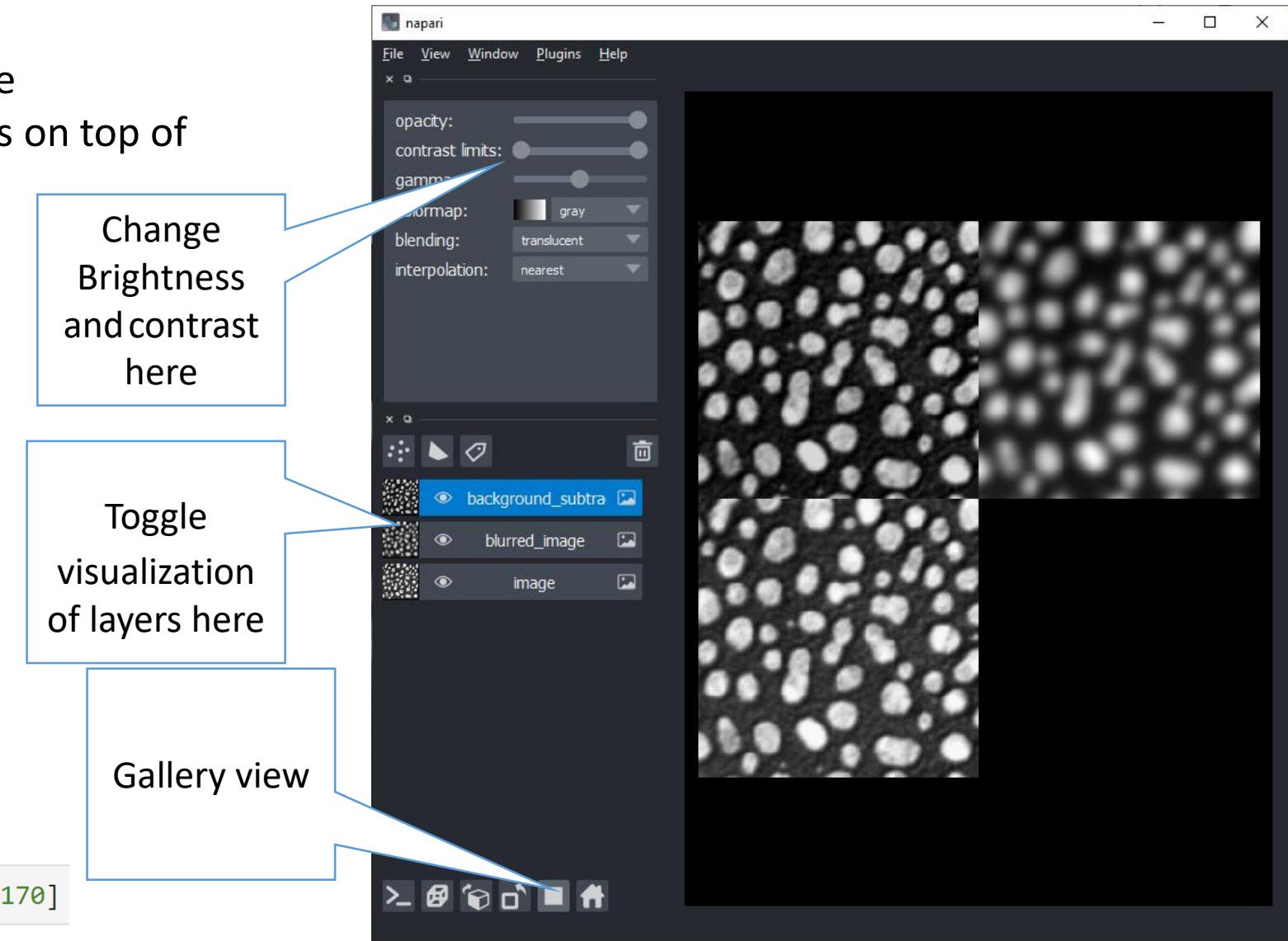
1. Access the viewer

2. Access the layers

3. Choose a layer (by index or name)

4. "Press TAB" and check out available properties

```
viewer.layers[0].contrast_limits = [30,170]
```



# Visualizing image segmentation

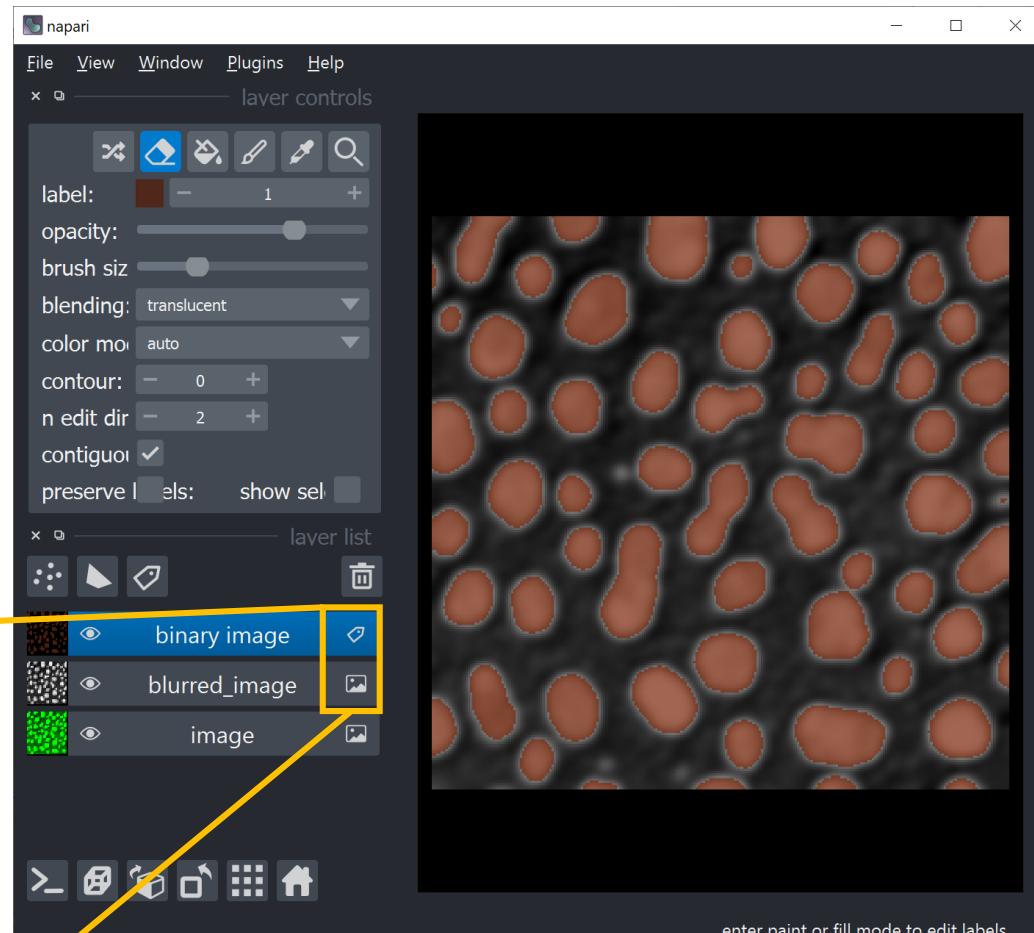
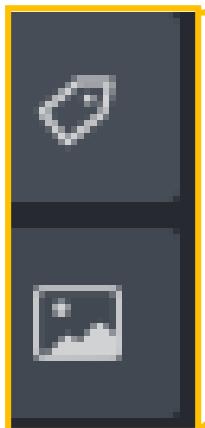
- Binary images and `label` images visualized as label layers

- `# Add a new labels layer containing an image`
- `viewer.add_labels(binary_image,`
- `name="binary image")`

Name your layers to keep track of what they contain

Labels Layer

Image Layer



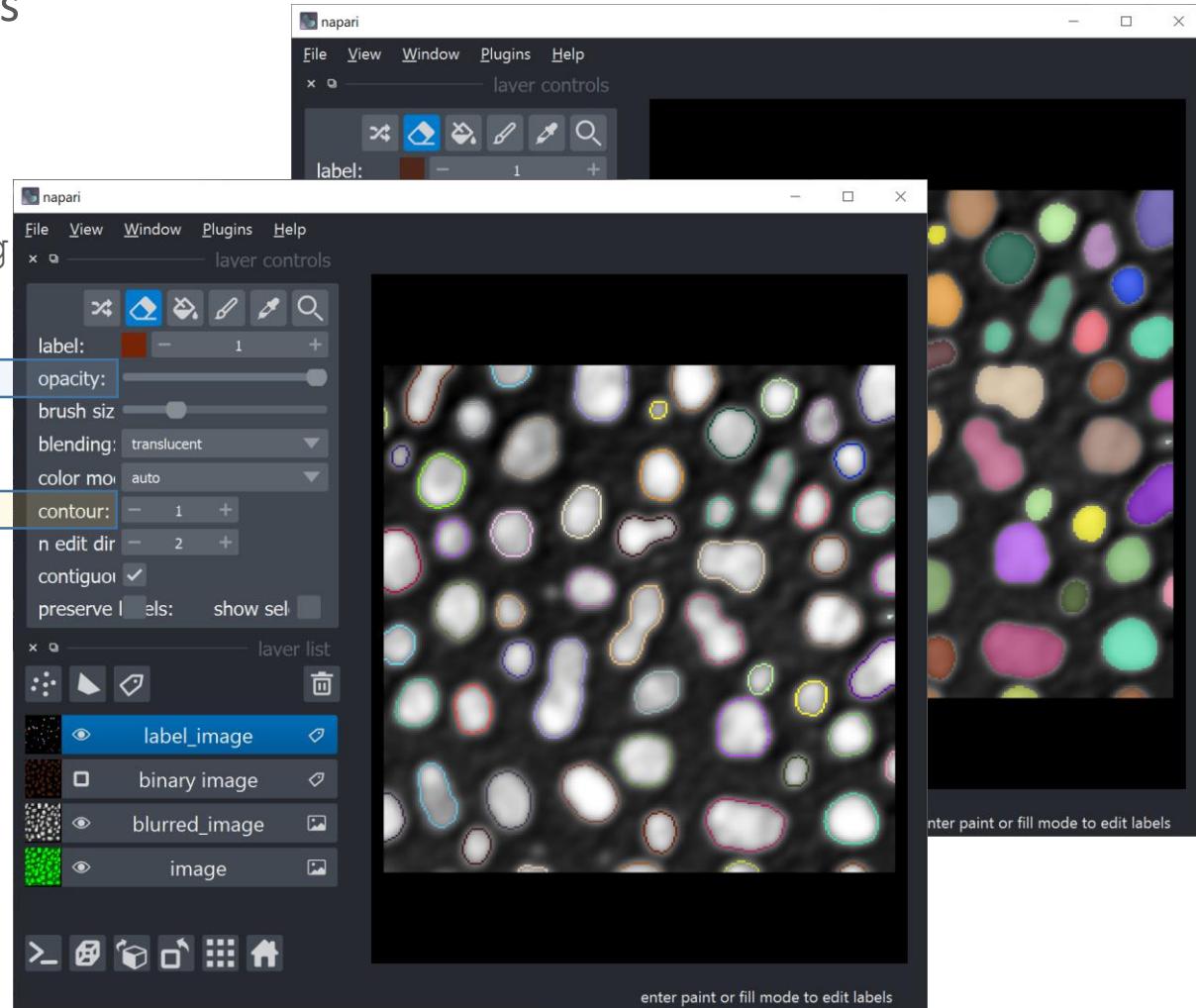
# Visualizing image segmentation

- Binary images and label images visualized as label layers

- ```
# add labels to viewer
```
- ```
label_layer = viewer.add_labels(label_image)
```

- Visualize contours instead of the overlay

- ```
label_layer.contour = 1
```
- ```
label_layer.opacity = 1
```

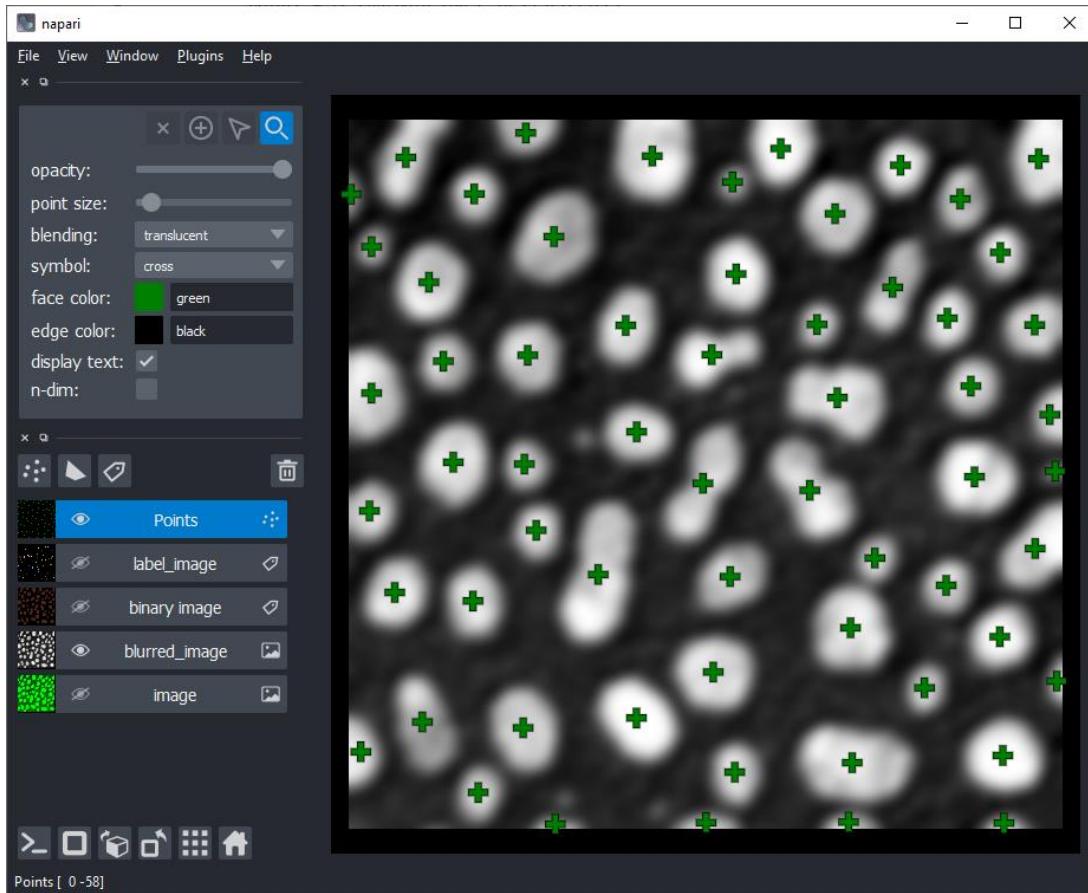


# Points layers

- There is also other layer types

- Shapes
- Points
- Surfaces
- Tracks
- Vectors

```
# add points to viewer
label_layer = viewer.add_points(points,
    face_color='green', symbol='cross', size=5)
```



1. Start up a terminal

2. Activate the environment using

```
mamba activate napari-intro-env
```

3. Run:

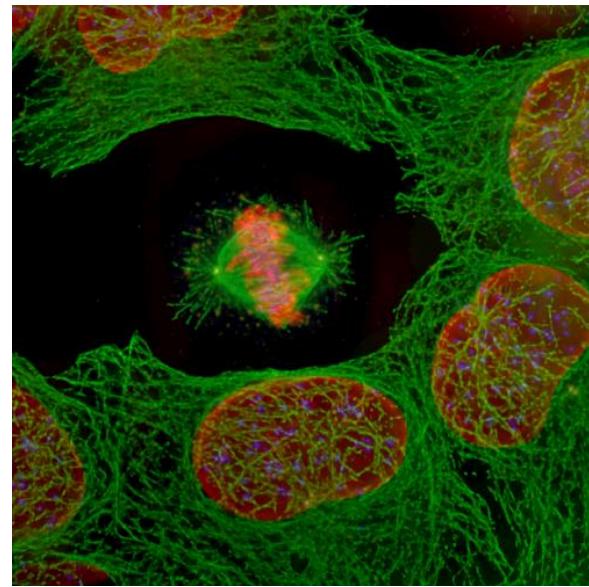
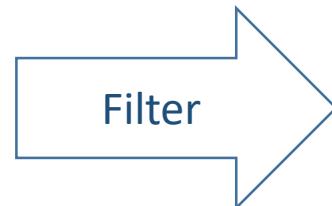
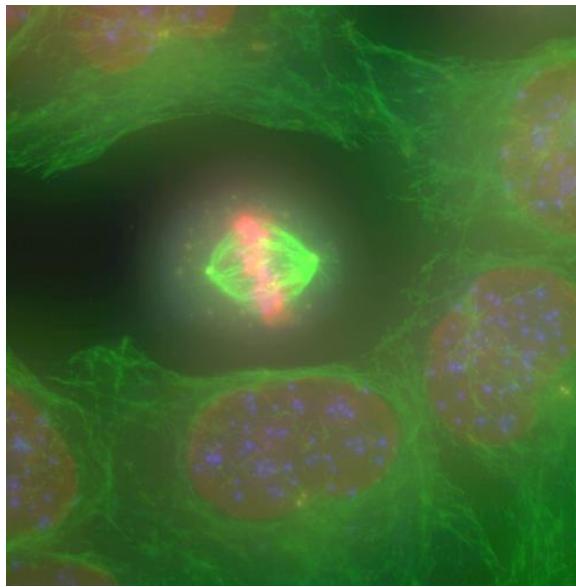
```
jupyter lab
```

- Run these notebooks to practice opening and visualizing images in a jupyter notebook and with napari:
  - [https://github.com/zoccoler/QM\\_Course\\_Bio\\_Image\\_Analysis\\_with\\_napari\\_2024/blob/main/docs/BioImage\\_Analysis\\_with\\_Jupyter\\_and\\_napari/01\\_images\\_are\\_numpy\\_arrays.ipynb](https://github.com/zoccoler/QM_Course_Bio_Image_Analysis_with_napari_2024/blob/main/docs/BioImage_Analysis_with_Jupyter_and_napari/01_images_are_numpy_arrays.ipynb)
  - [https://github.com/zoccoler/QM\\_Course\\_Bio\\_Image\\_Analysis\\_with\\_napari\\_2024/blob/main/docs/BioImage\\_Analysis\\_with\\_Jupyter\\_and\\_napari/02\\_Napari\\_from\\_Jupyter\\_introduction.ipynb](https://github.com/zoccoler/QM_Course_Bio_Image_Analysis_with_napari_2024/blob/main/docs/BioImage_Analysis_with_Jupyter_and_napari/02_Napari_from_Jupyter_introduction.ipynb)

# Image Processing

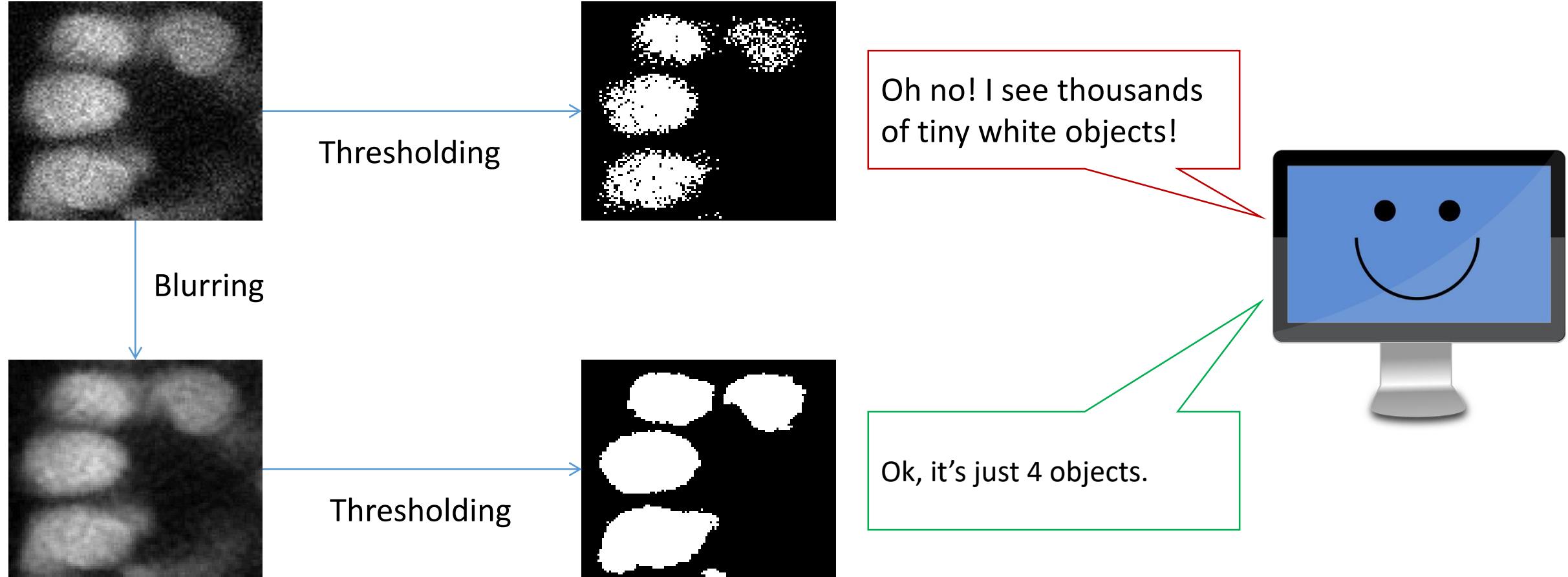
- An image processing filter is an operation on an image.
- It takes an image and produces a new image out of it.
- Filters change pixel values.
- There is no “best” filter. Which filter fits your needs, depends on the context.
- Filters do not do magic. They can not make things visible which are not in the image.

- Application examples
  - Noise-reduction
  - Artefact-removal
  - Contrast enhancement
  - Correct uneven illumination



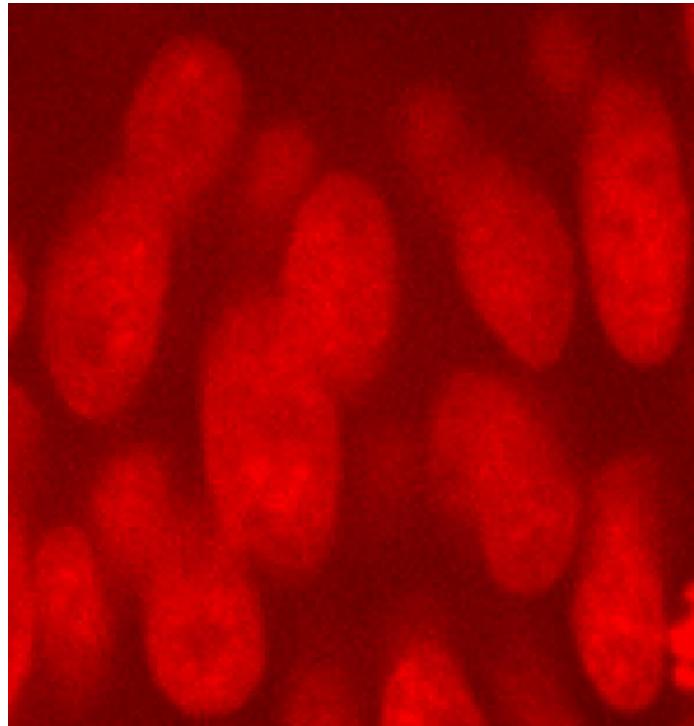
# Image correction / noise removal

- We need to remove the noise to help the computer *interpreting* the image

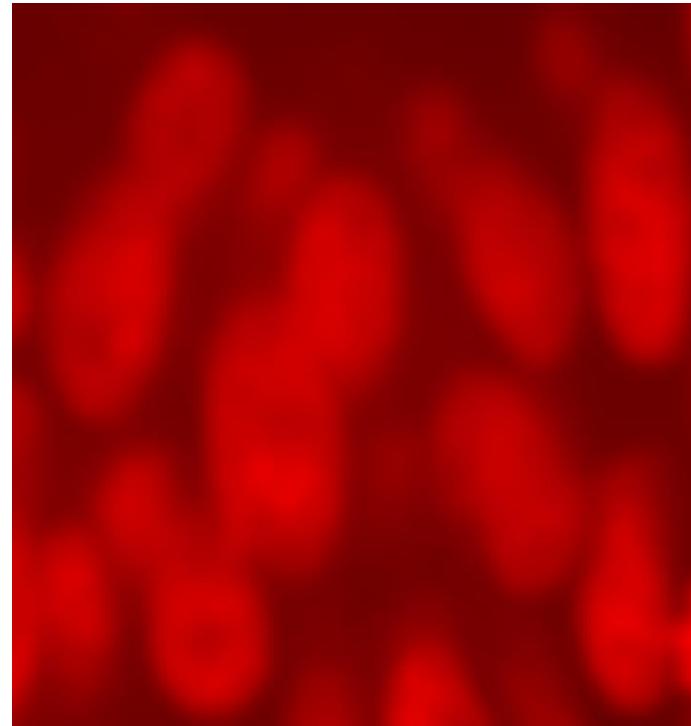


# Image correction / noise removal

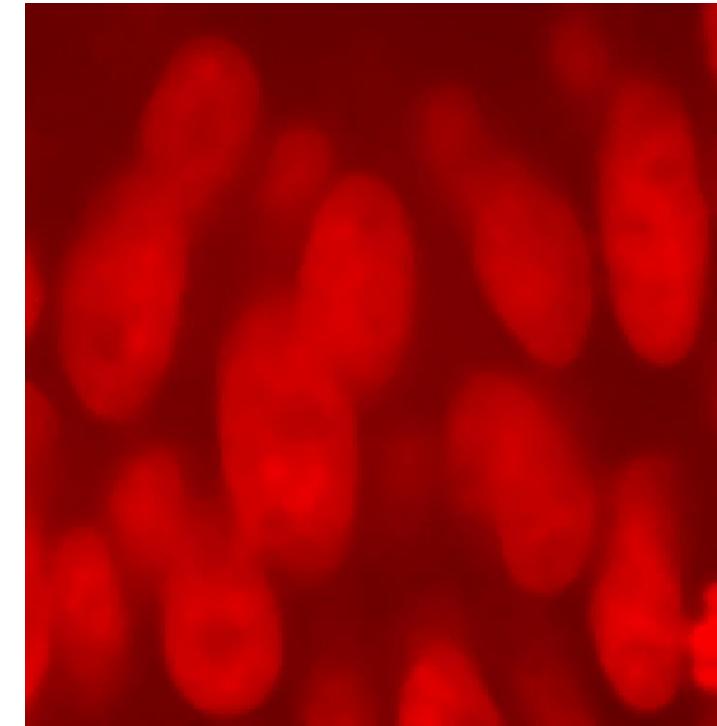
- Noise removal using *filters*



Original image



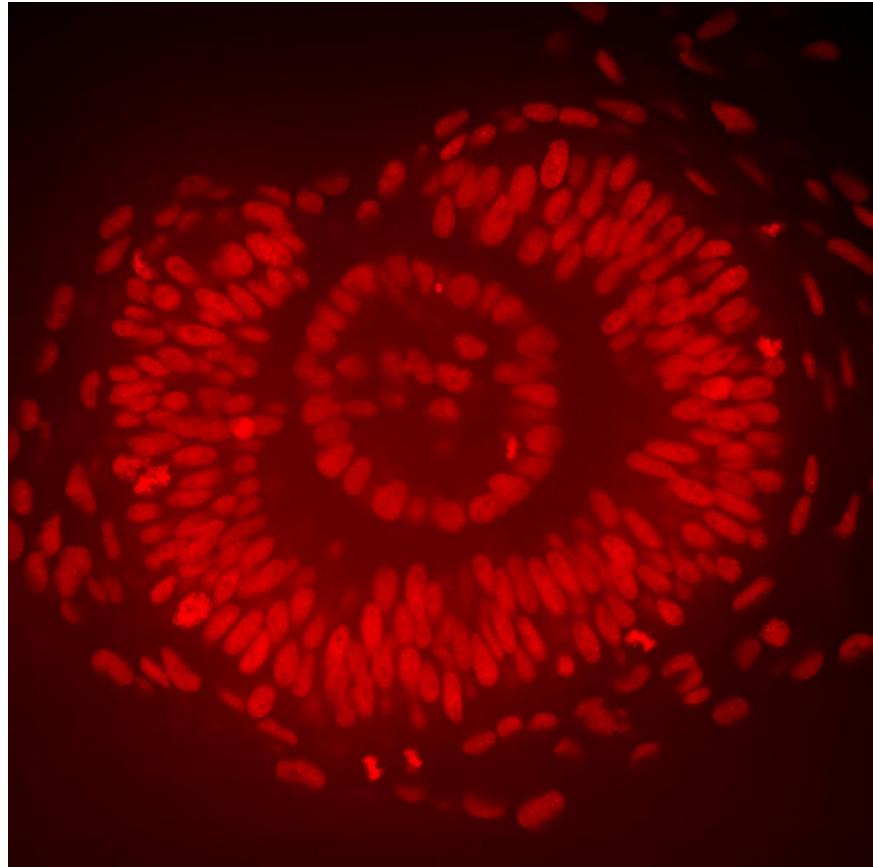
Gaussian blur filtered



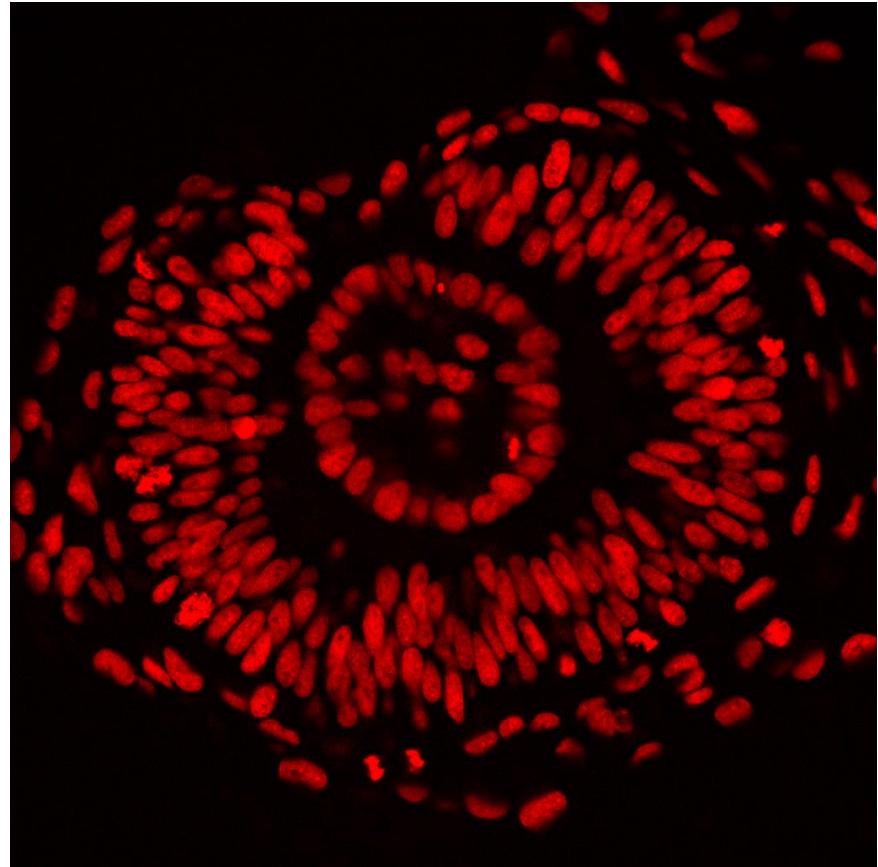
Median filtered

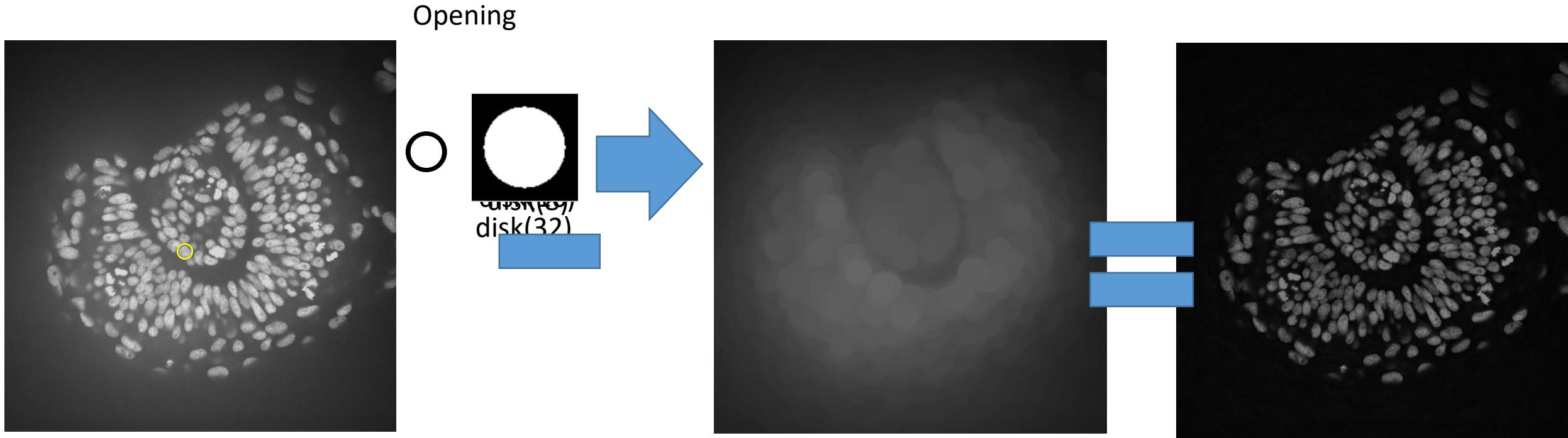
# Background removal

- Differentiating objects is easier if their background intensity is equal.



Subtract  
background





Original image

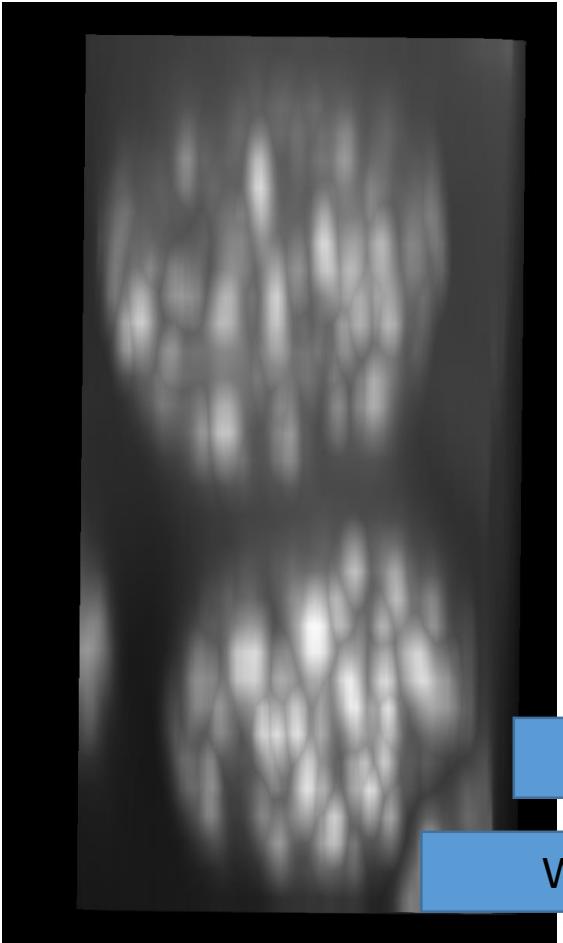
Structures have a radius  $\approx 12$

This is a good estimation of the background

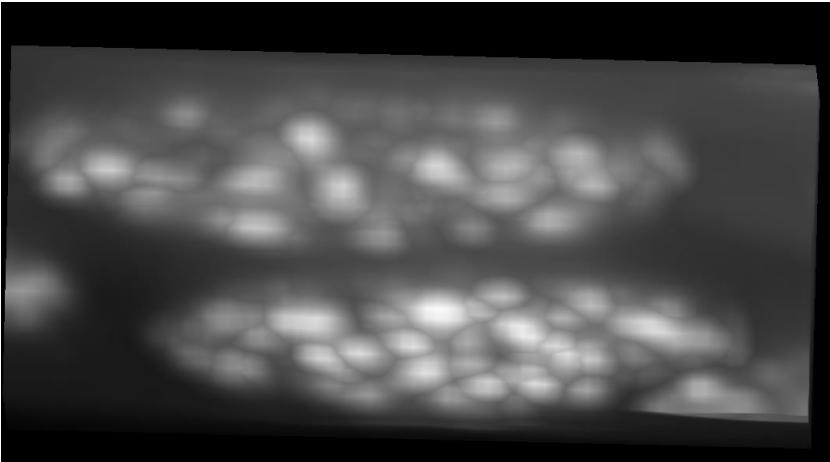
You have just learned the white tophat filter!

# Filtering in 3D with Anisotropy

An image that should look like this...



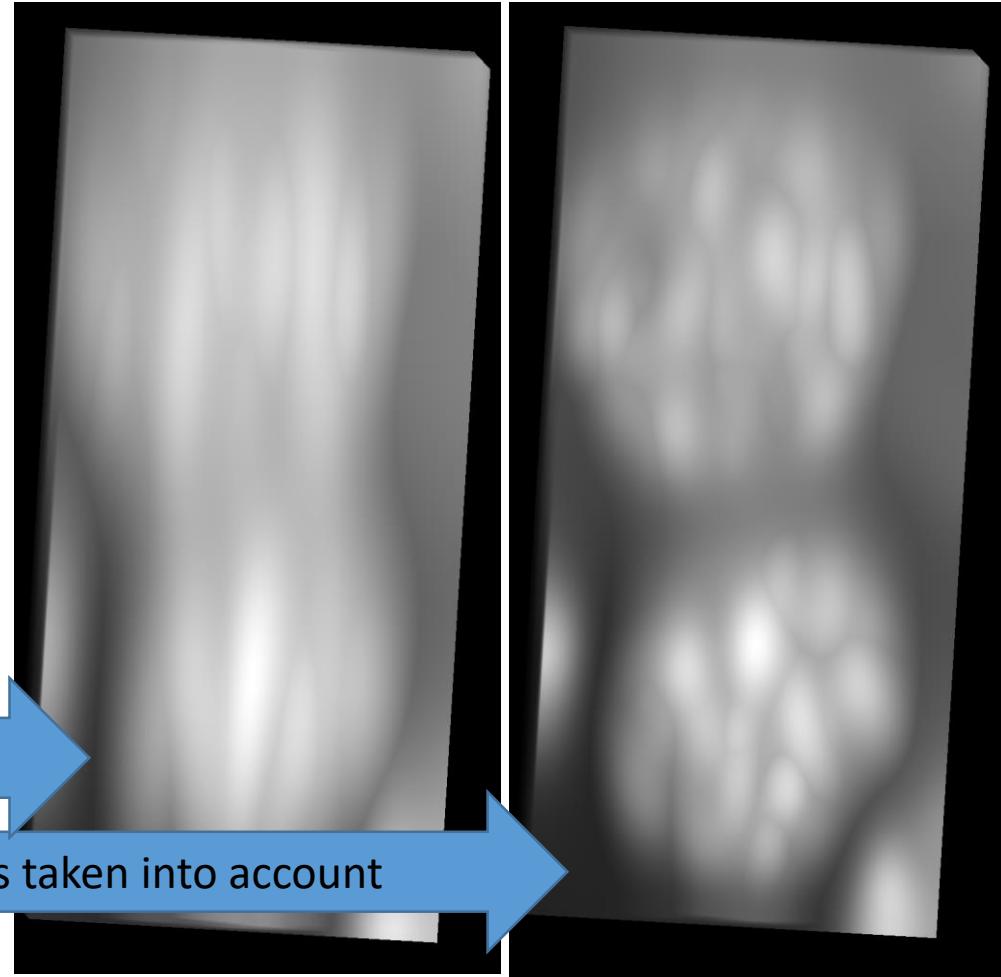
...may be displayed like this (if voxel size is ignored):



Many filter functions do not take anisotropy into account (always check documentation)

So applying a filter may do this

When in fact it should do this if anisotropy was taken into account



Re-scaling the image may be needed before filtering (unless function allows editing filter kernel)

1. Start up a terminal

2. Activate the environment using

```
mamba activate napari-intro-env
```

3. Run:

```
jupyter lab
```

- Run these notebooks to practice opening and visualizing images in a jupyter notebook and with napari:

- [https://github.com/zoccoler/QM\\_Course\\_Bio\\_Image\\_Analysis\\_with\\_napari\\_2024/blob/main/docs/BioImage\\_Analysis\\_with\\_Jupyter\\_and\\_napari/03\\_Filters\\_in\\_3D\\_Isotropic\\_Data.ipynb](https://github.com/zoccoler/QM_Course_Bio_Image_Analysis_with_napari_2024/blob/main/docs/BioImage_Analysis_with_Jupyter_and_napari/03_Filters_in_3D_Isotropic_Data.ipynb)

Optional:

- [https://github.com/zoccoler/QM\\_Course\\_Bio\\_Image\\_Analysis\\_with\\_napari\\_2024/blob/main/docs/BioImage\\_Analysis\\_with\\_Jupyter\\_and\\_napari/04\\_Brightness\\_and\\_Contrast\\_optional.ipynb](https://github.com/zoccoler/QM_Course_Bio_Image_Analysis_with_napari_2024/blob/main/docs/BioImage_Analysis_with_Jupyter_and_napari/04_Brightness_and_Contrast_optional.ipynb)
- [https://github.com/zoccoler/QM\\_Course\\_Bio\\_Image\\_Analysis\\_with\\_napari\\_2024/blob/main/docs/BioImage\\_Analysis\\_with\\_Jupyter\\_and\\_napari/05\\_Filters\\_optional.ipynb](https://github.com/zoccoler/QM_Course_Bio_Image_Analysis_with_napari_2024/blob/main/docs/BioImage_Analysis_with_Jupyter_and_napari/05_Filters_optional.ipynb)

## Aim:

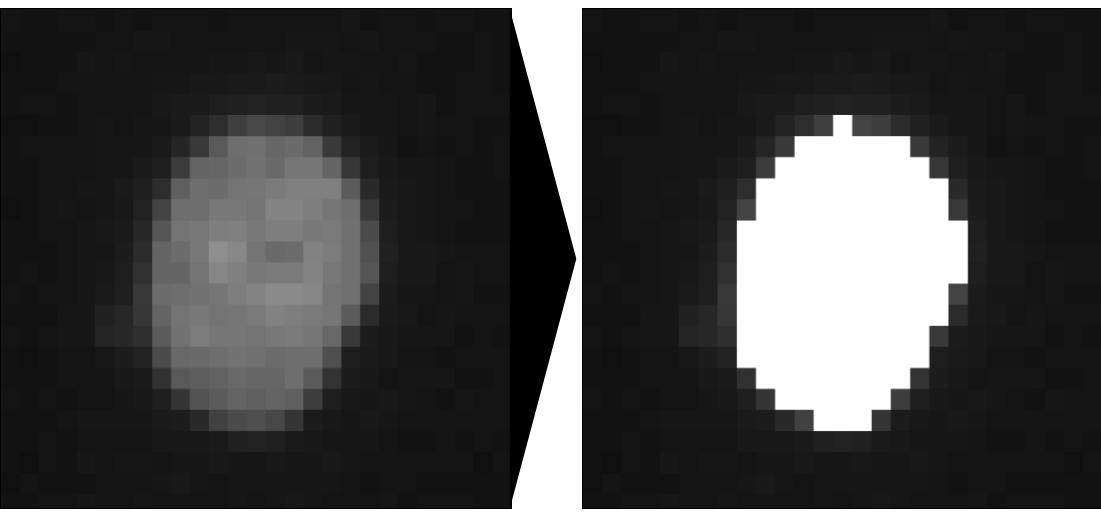
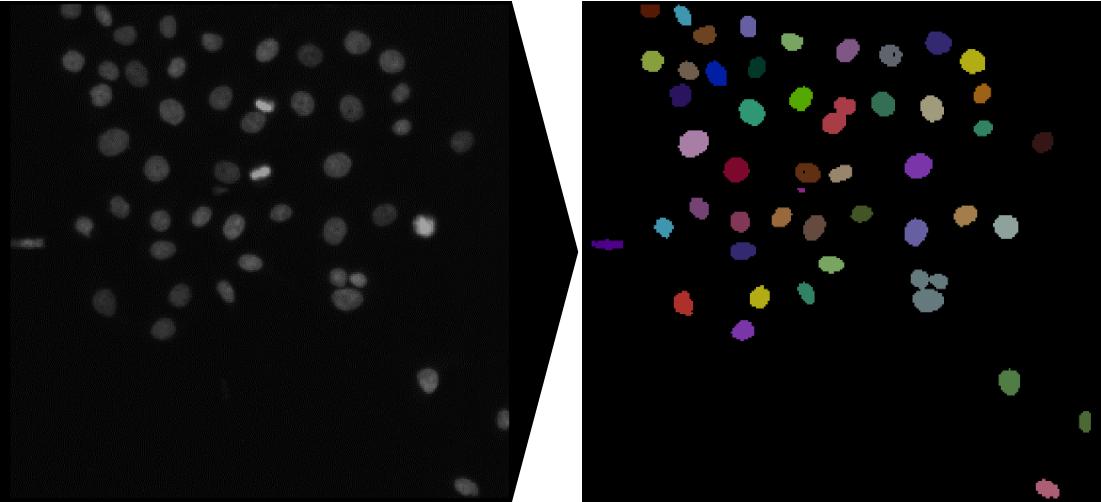
Separate background from foreground

## Vocabulary:

- **Segmentation:**
  - Assigning a meaningful *label* to each pixel
  - Segmentation is a *classification* problem
- **Semantic segmentation:**

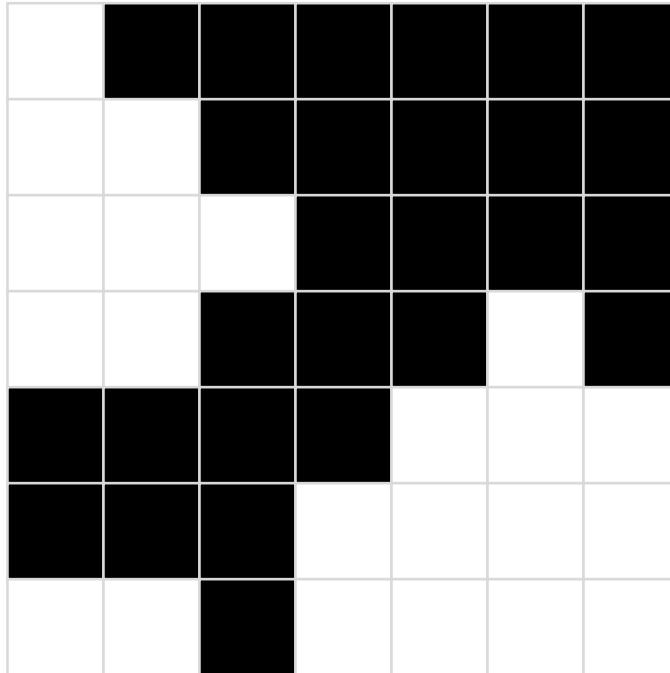
Differentiate pixels into multiple *classes* (e.g., membrane, nucleus, cytosol, etc.)
- **Instance segmentation:**

Differentiate multiple occurrences of the same class into separate instances of this class (e.g., separate *label* for each cell in image)



# Instance segmentation

- In order to allow the computer differentiating objects, connected component analysis (CCA) is used to mark pixels belonging to different objects with different numbers
- Background pixels are marked with 0.
- The maximum intensity of a labelled map corresponds to the number of objects.

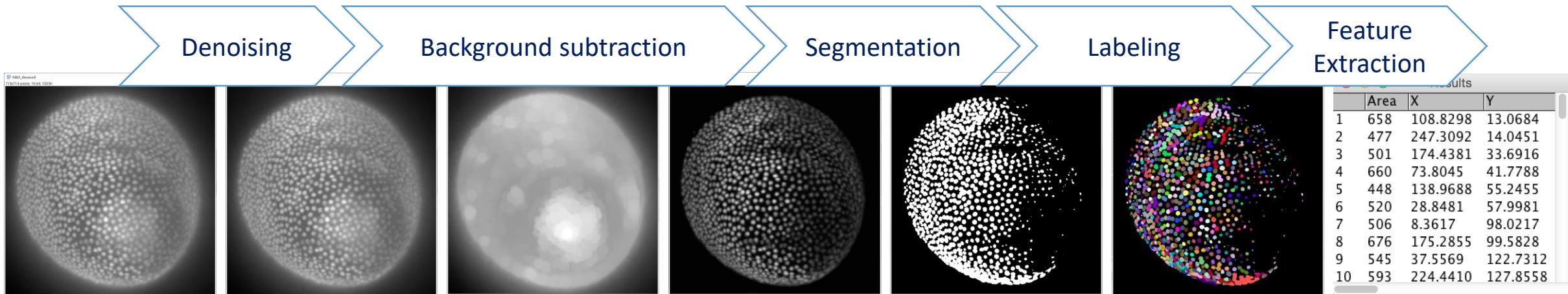


CCA

1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	3	0	0
0	0	0	0	3	3	3	3
0	0	0	3	3	3	3	3
2	2	0	3	3	3	3	3

# Image Analysis Workflows

- Image analysis workflow is a series of processes/functions (not always linear) applied to images to achieve a certain goal (usually some measurements)
- Here is a classic example:



- Batch processing is the process of applying the same image analysis workflow to several images (in a folder or data repository)
- In Python, that typically means putting the workflow inside a **for** loop that runs over all images of the folder/repository

1. Start up a terminal

2. Activate the environment using

```
mamba activate napari-intro-env
```

3. Run:

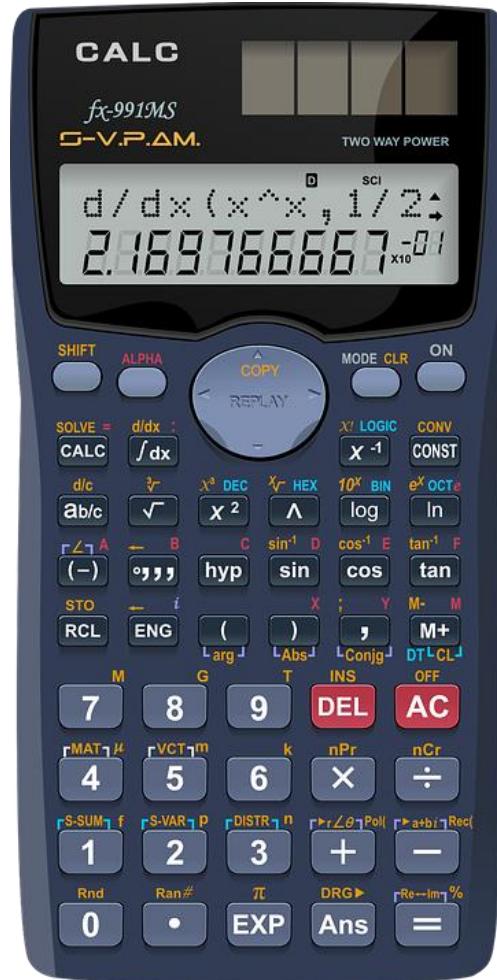
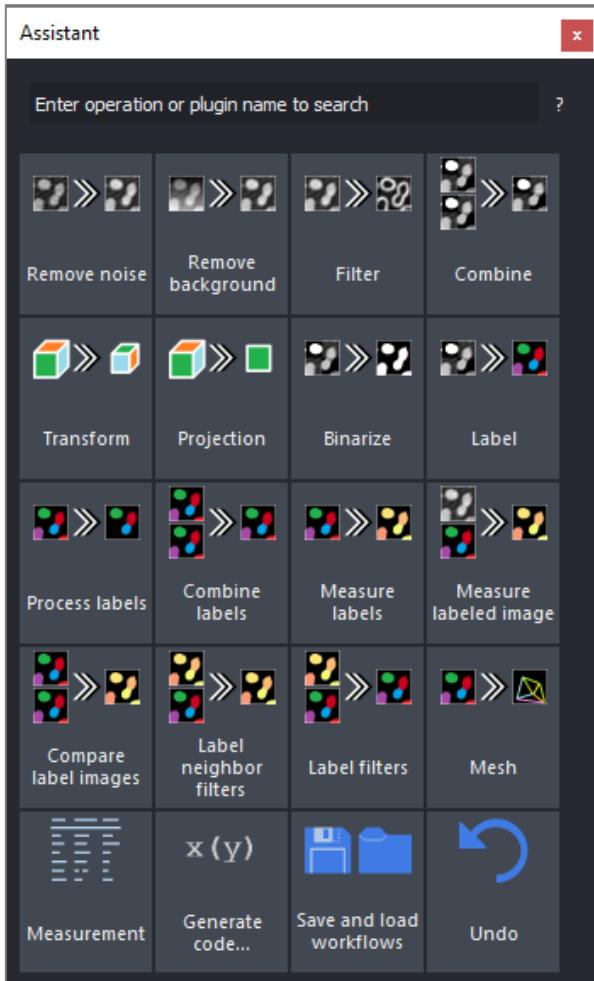
```
jupyter lab
```

- Run these notebooks to practice opening and visualizing images in a jupyter notebook and with napari:
  - [https://github.com/zoccoler/QM\\_Course\\_Bio\\_Image\\_Analysis\\_with\\_napari\\_2024/blob/main/docs/BioImage\\_Analysis\\_with\\_Jupyter\\_and\\_napari/06\\_Segmentation\\_workflow\\_example.ipynb](https://github.com/zoccoler/QM_Course_Bio_Image_Analysis_with_napari_2024/blob/main/docs/BioImage_Analysis_with_Jupyter_and_napari/06_Segmentation_workflow_example.ipynb)
  - [https://github.com/zoccoler/QM\\_Course\\_Bio\\_Image\\_Analysis\\_with\\_napari\\_2024/blob/main/docs/BioImage\\_Analysis\\_with\\_Jupyter\\_and\\_napari/07\\_batch\\_segmentation\\_workflow\\_example.ipynb](https://github.com/zoccoler/QM_Course_Bio_Image_Analysis_with_napari_2024/blob/main/docs/BioImage_Analysis_with_Jupyter_and_napari/07_batch_segmentation_workflow_example.ipynb)

# Napari-Assistant

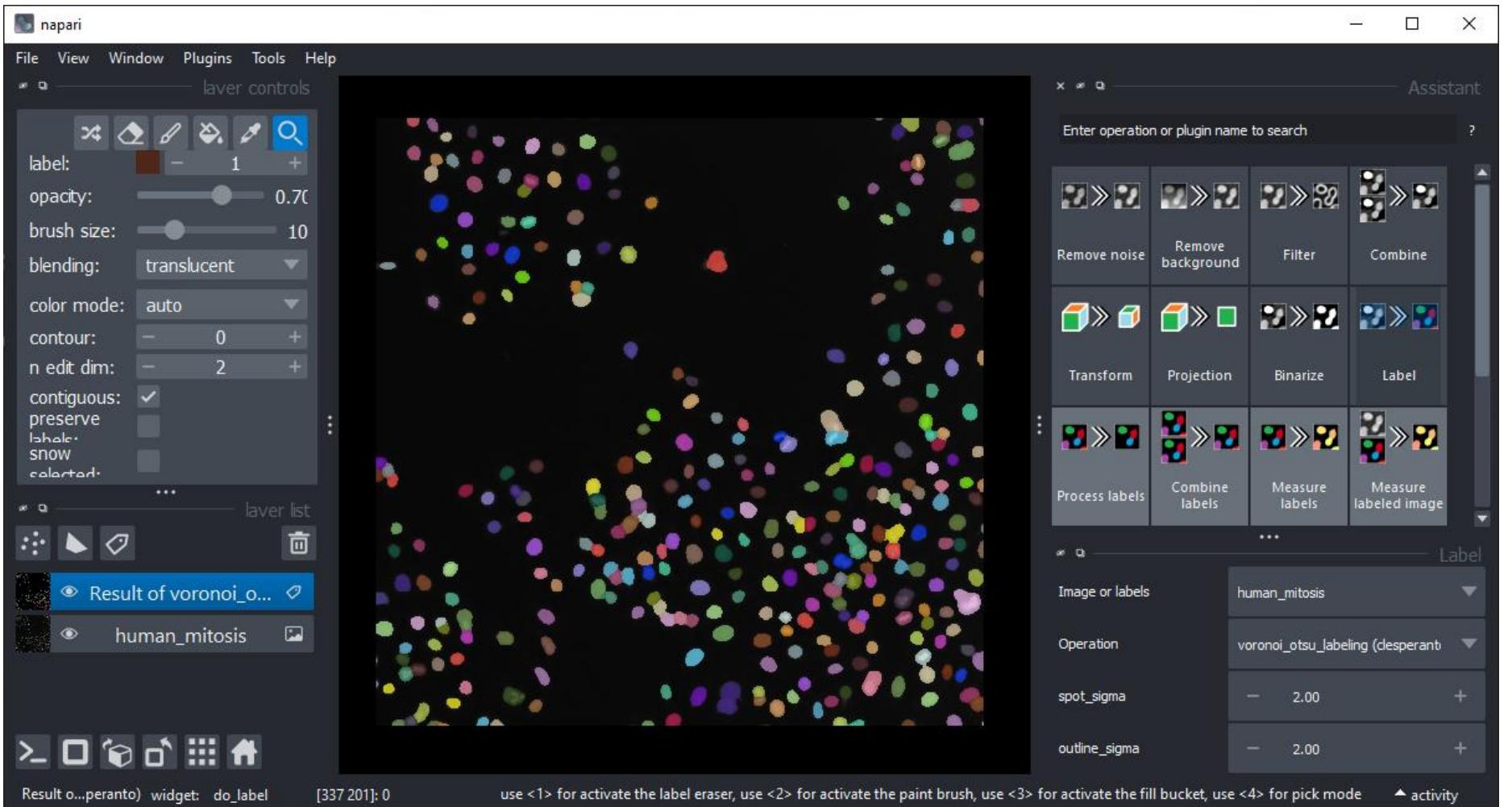
# The Napari Assistant

- A pocket-calculator-like interface to build image analysis workflows



<https://www.napari-hub.org/plugins/napari-assistant>

# The Napari Assistant



Viewer  
controls

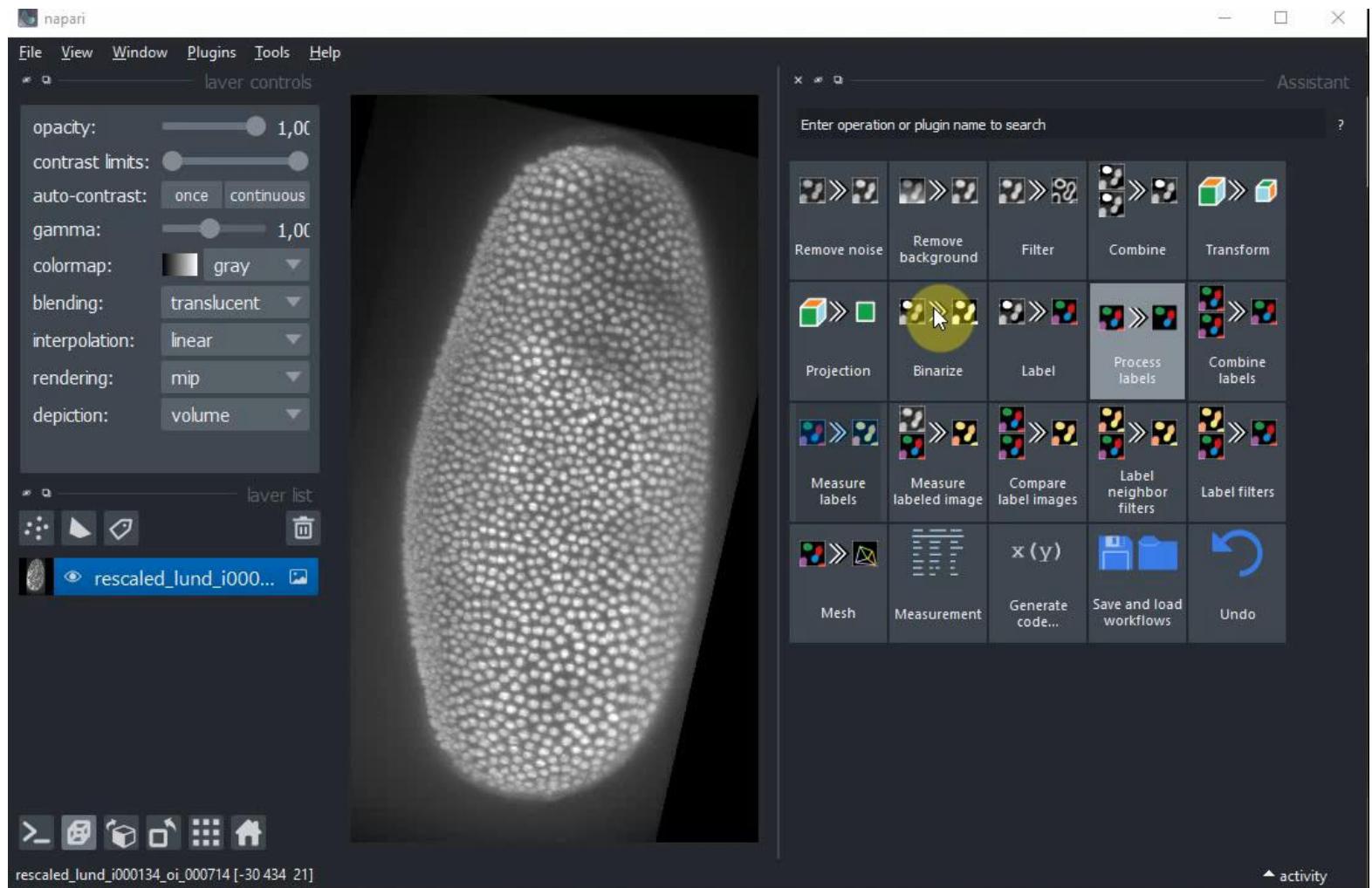
(Napari core)

Image  
Analysis

(Napari Assistant)

<https://www.napari-hub.org/plugins/napari-assistant>

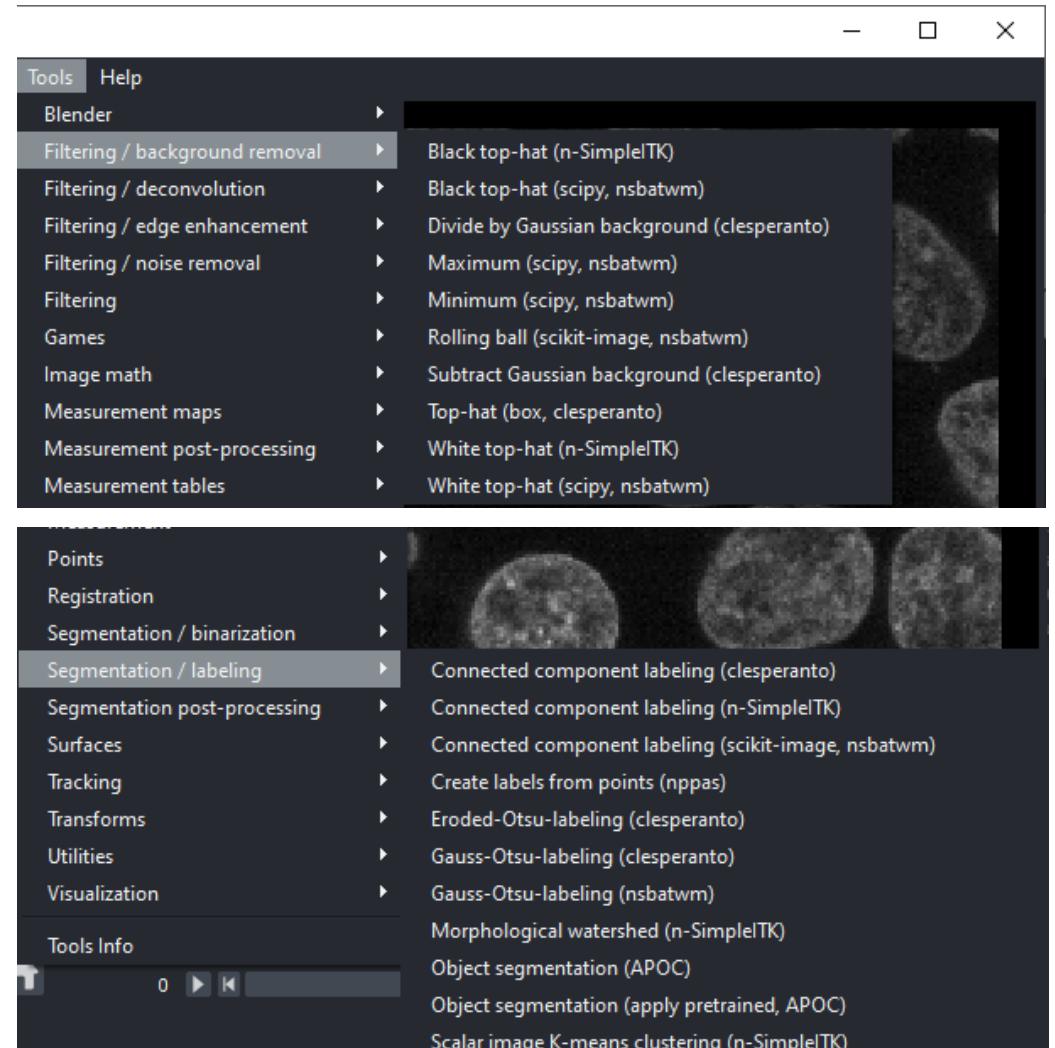
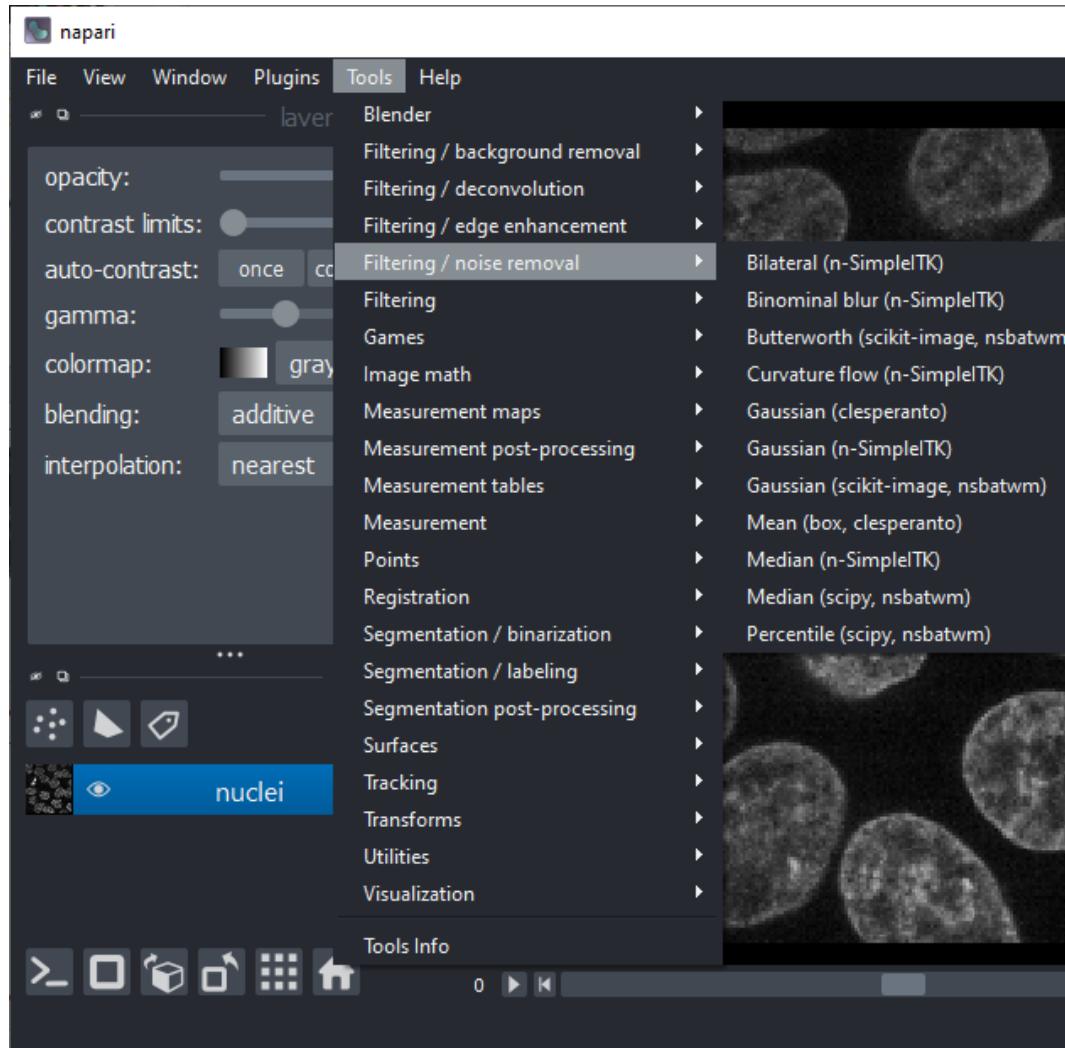
# The Napari Assistant



<https://www.napari-hub.org/plugins/napari-assistant>

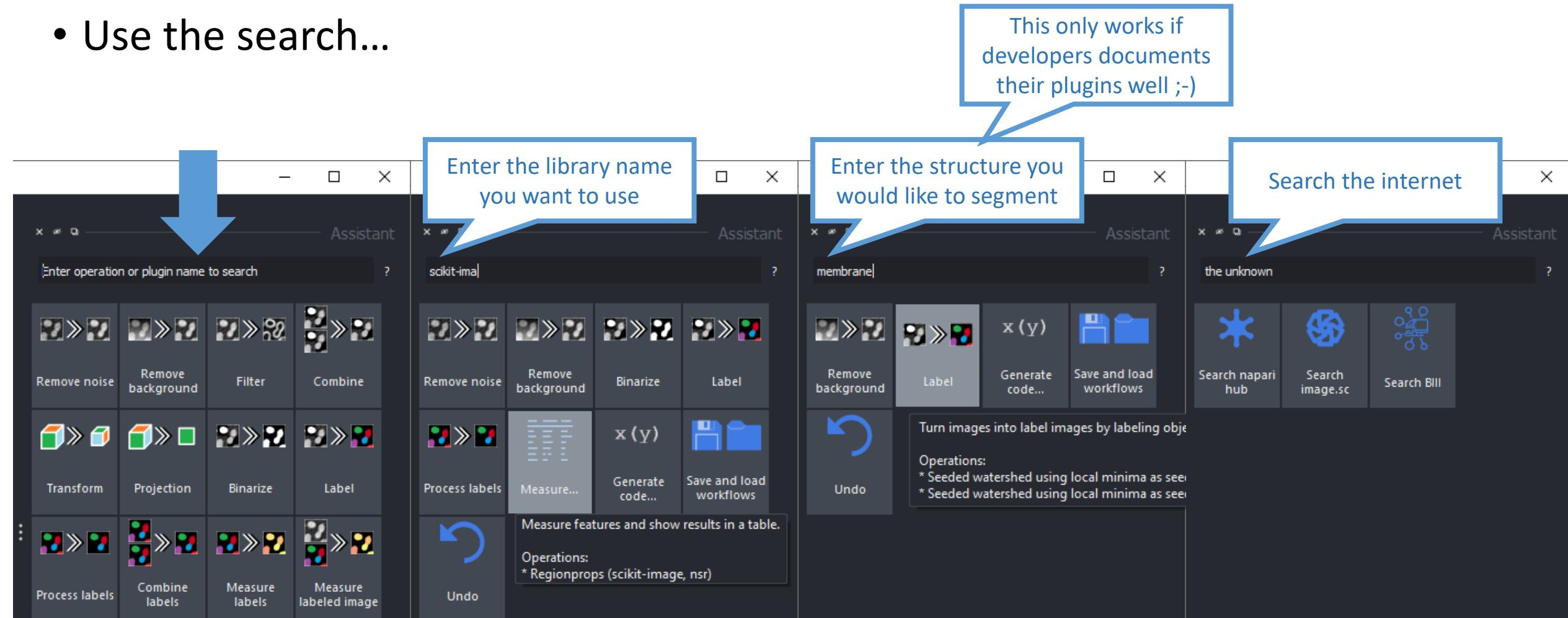
# The Tools menu

- Organized in categories



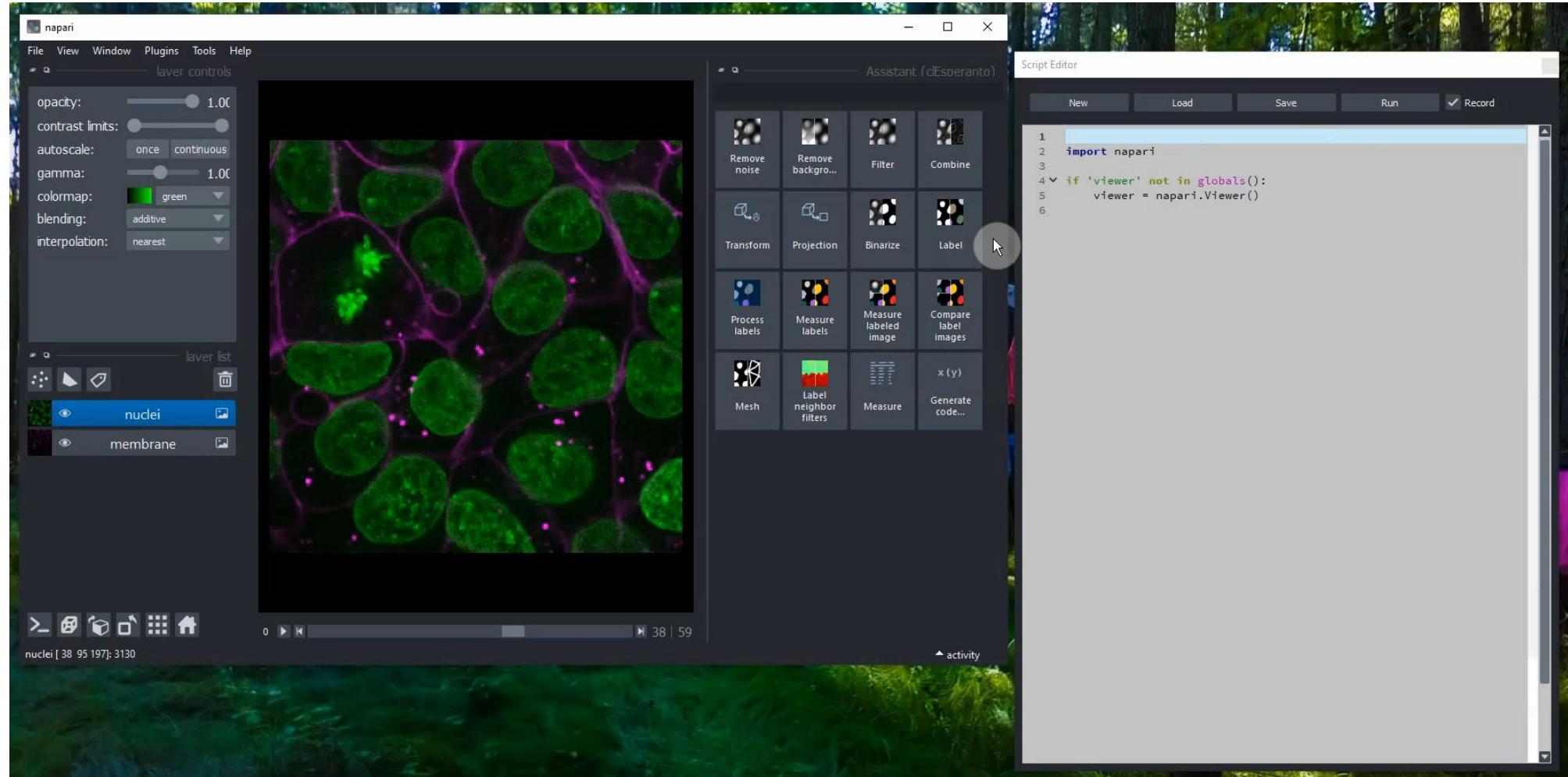
# Browse operations

- Use the search...



<https://www.napari-hub.org/plugins/napari-assistant>

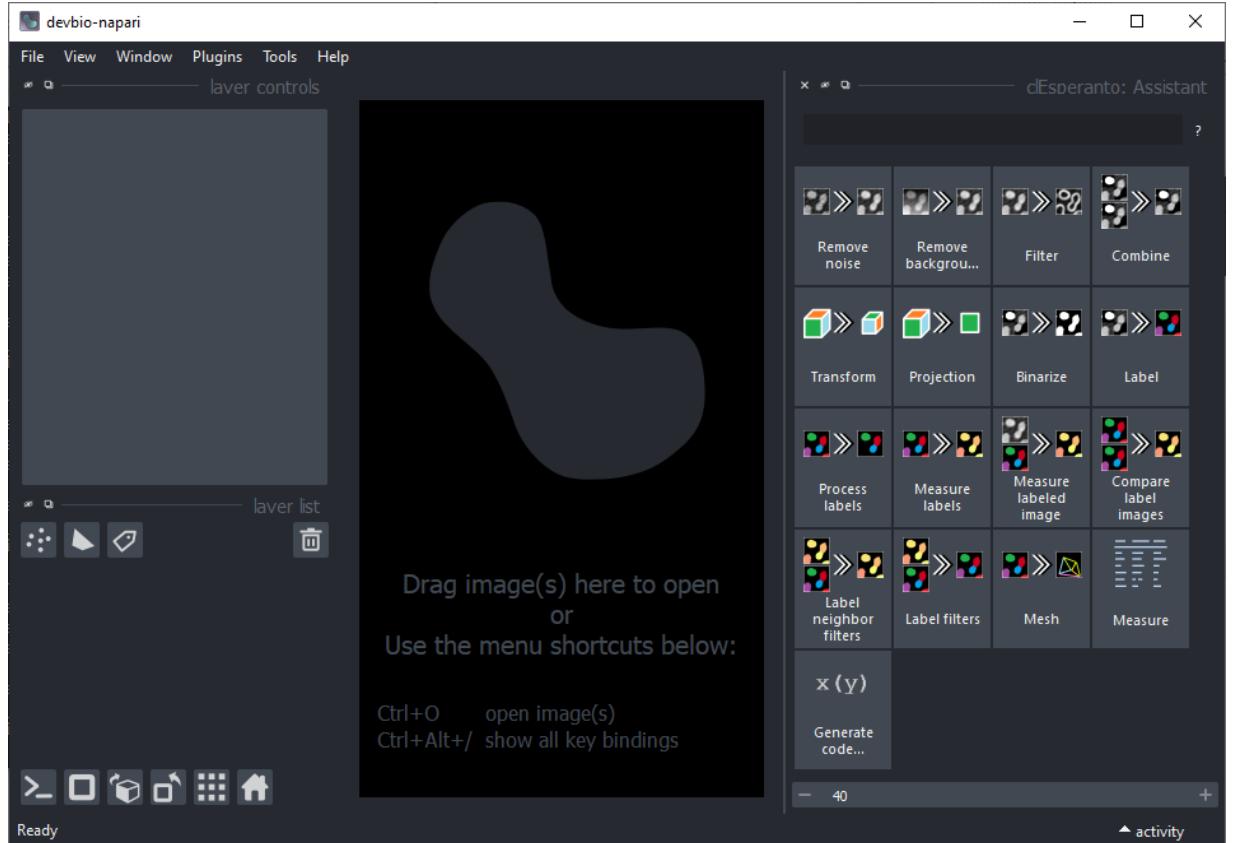
# Export code to Python



<https://www.napari-hub.org/plugins/napari-script-editor>

# Exercise: run napari-assistant

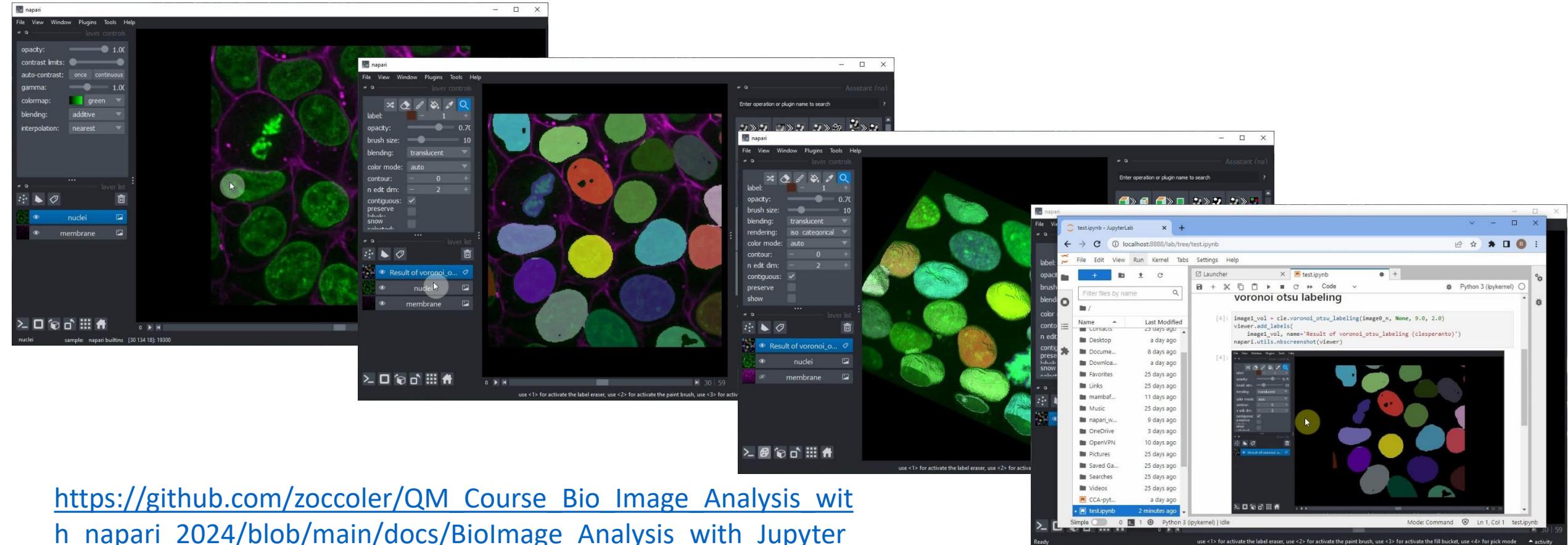
1. Start up a terminal
  2. Activate the environment using  
mamba activate napari-intro-env
  3. Run:  
naparia
- 
- [https://github.com/zoccoler/QM\\_Course\\_Bio\\_Image\\_Analysis\\_with\\_napari\\_2024/blob/main/\\_docs/BioImage\\_Analysis\\_with\\_Jupyter\\_and\\_napari/08\\_napari-assistant.md](https://github.com/zoccoler/QM_Course_Bio_Image_Analysis_with_napari_2024/blob/main/_docs/BioImage_Analysis_with_Jupyter_and_napari/08_napari-assistant.md)



[https://github.com/clEsperanto/pyclesperanto\\_prototype#troubleshooting-graphics-cards-drivers](https://github.com/clEsperanto/pyclesperanto_prototype#troubleshooting-graphics-cards-drivers)

# Exercise

- Use the Napari Assistant to generate a Jupyter Notebook



[https://github.com/zoccoler/QM Course Bio Image Analysis with napari 2024/blob/main/docs/BioImage%20Analysis%20with%20Jupyter%20and%20napari/09\\_notebook\\_export.md](https://github.com/zoccoler/QM Course Bio Image Analysis with napari 2024/blob/main/docs/BioImage%20Analysis%20with%20Jupyter%20and%20napari/09_notebook_export.md)

# Acknowledgements



## BiAPoL team

- Marcelo Zoccoler
- Johannes Soltwedel
- Maleeha Hassan
- Stefan Hahmann
- Somashekhar Kulkarni

### Former lab members:

- Robert Haase
- Allyson Ryan
- Till Korten
- Mara Lampert
- Svetlana Iarovenko
- Ryan George Savill
- Laura Zigutyte

## Networks



## Funding



**Chan  
Zuckerberg  
Initiative** 

