



Python

Data structures

Robert Haase

April 2022

- Variables are memory blocks where you can store stuff

```
measurement = 5
```

Computer memory

measurement

5

name

"Drosophila"

combination

"Drosophila5"

- Arrays are variables, where you can store multiple values

Give me a "0", five times!

```
array = [0] * 5
```

Computer memory

array

1	0	5	0	Rab bit
---	---	---	---	------------

```
▶ # 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 arrays

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” is addressing certain elements in arrays. 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” is addressing certain elements in arrays. 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	

```
data[0]
```

'A'

- “Indexing” is addressing certain elements in arrays. 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	

```
data[0]
```

'A'

```
data[1]
```

'B'

- “Indexing” is addressing certain elements in arrays. 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	

```
data[0]
```

'A'

```
data[1]
```

'B'

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

['A', 'B']

- “Indexing” is addressing certain elements in arrays. 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	

```
data[0]
```

```
'A'
```

```
data[1]
```

```
'B'
```

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

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

```
data[0:3]
```

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

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

```
['B']
```

```
len(data)
```

```
9
```

- “Indexing” is addressing certain elements in arrays. 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	

```
data[0]
```

'A'

```
data[1]
```

'B'

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

['A', 'B']

```
data[0:3]
```

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

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

['B']

- “Indexing” is addressing certain elements in arrays. 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	

```
data[0]
```

'A'

```
data[1]
```

'B'

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

['A', 'B']

```
data[0:3]
```

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

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

['B']

```
len(data)
```

9

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

```
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	

```
data[:2]
```

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

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

```
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	

```
data[:2]
```

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

```
data[:3]
```

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

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

```
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	

```
data[:2]
```

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

```
data[:3]
```

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

```
data[2:]
```

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

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

```
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	

```
data[:2]
```

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

```
data[:3]
```

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

```
data[2:]
```

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

```
data[3:]
```

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

- The step-size allows skipping elements

```
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	

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

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


- The step-size allows skipping elements

```
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	

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

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

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

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

- The step-size allows skipping elements

```
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	

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

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

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

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

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

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

- 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	0
Content:	A	B	C	D	E	F	G	H	I	

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

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

- 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	0
Content:	A	B	C	D	E	F	G	H	I	

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

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

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

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

- 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	0
Content:	A	B	C	D	E	F	G	H	I	

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

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

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

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

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

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

- 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	0
Content:	A	B	C	D	E	F	G	H	I	

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

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

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

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

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

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

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

```
[]
```

- Negative stepping also works

```
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	

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

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

- Negative stepping also works

```
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	

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

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

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

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


- Modifying array elements

```
▶ 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 arrays of defined size

What? How many?

```
▶ zeros = [0] * 10
print(zeros)

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

- Concatenating arrays

```
▶ ones = [1, 1, 1]
twos = [2, 2, 2, 2]

# concatenate arrays
numbers = ones + twos

print(numbers)

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

+ means appending

- Lists can be modified

```
measurements = [5.5, 6.3, 7.2, 8.0, 8.8]
```

```
measurements[1] = 25
```

```
measurements.append(10.2)
```

```
measurements
```

```
] [5.5, 25, 7.2, 8.0, 8.8, 10.2]
```

- Tuples not

```
immutable = (4, 3, 7.8)
```

Note: round brackets

```
immutable[1] = 5
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-49-a01b13633c23> in <module>  
----> 1 immutable[1] = 5
```

```
TypeError: 'tuple' object does not support item assignment
```

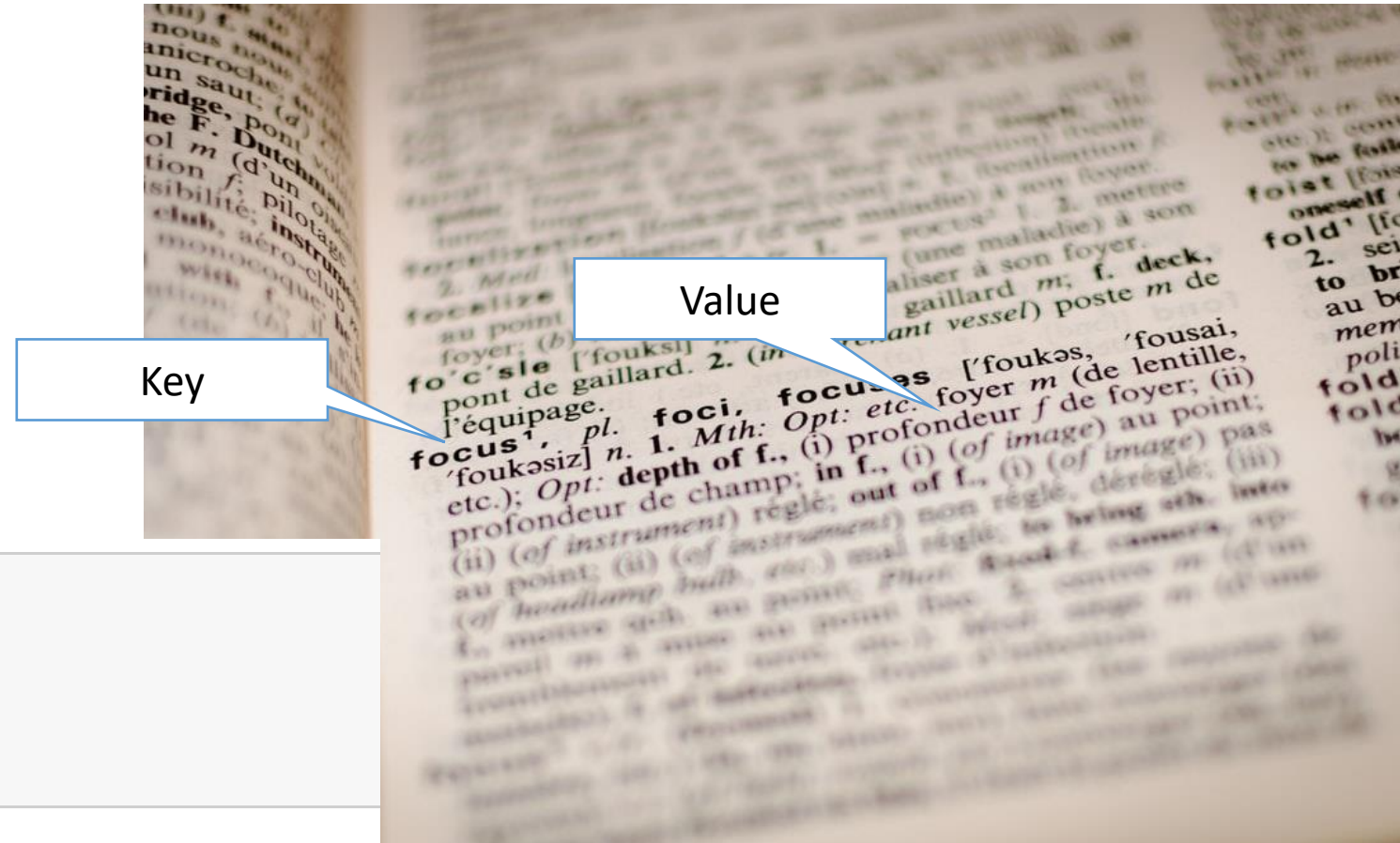
- Dictionary: a list of key-value pairs

Key Value

```
▶ german_english_dictionary = {  
    'Vorlesung': 'Lecture',  
    'Gleichung': 'Equation'  
}
```

▶ german_english_dictionary

```
]: {'Vorlesung': 'Lecture', 'Gleichung': 'Equation'}
```



- Dictionary: a list of key-value pairs

```
▶ german_english_dictionary = {  
    'Vorlesung': 'Lecture',  
    'Gleichung': 'Equation'  
}
```

- Look up something in the dictionary: it's an array with named entries!

```
▶ german_english_dictionary['Vorlesung']  
]: 'Lecture'
```

- Tables can be dictionaries with arrays as values

```
► measurements_week = {  
    'Monday': [2.3, 3.1, 5.6],  
    'Tuesday': [1.8, 7.0, 4.3],  
    'Wednesday': [4.5, 1.5, 3.2],  
    'Thursday': [1.9, 2.0, 6.4],  
    'Friday': [4.4, 2.3, 5.4]  
}
```

```
► measurements_week
```

```
]: {'Monday': [2.3, 3.1, 5.6],  
    'Tuesday': [1.8, 7.0, 4.3],  
    'Wednesday': [4.5, 1.5, 3.2],  
    'Thursday': [1.9, 2.0, 6.4],  
    'Friday': [4.4, 2.3, 5.4]}
```

- Retrieve a column

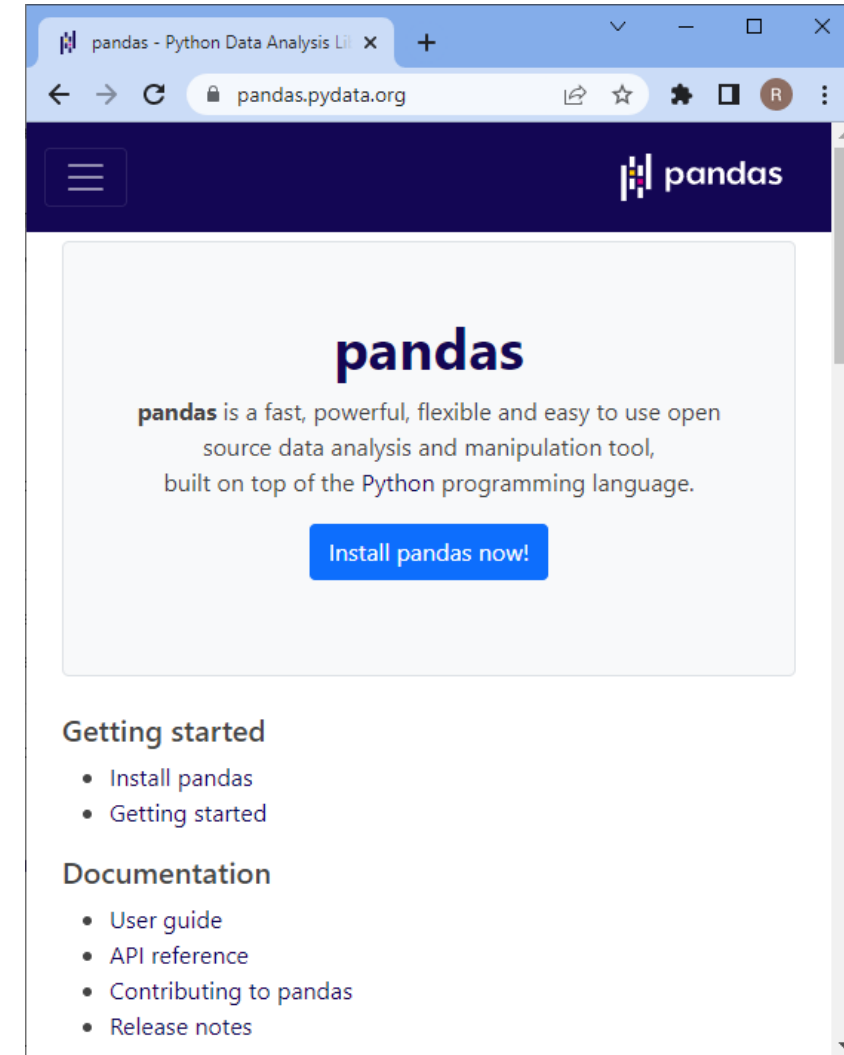
```
► measurements_week['monday']
```

```
]: [2.3, 3.1, 5.6]
```

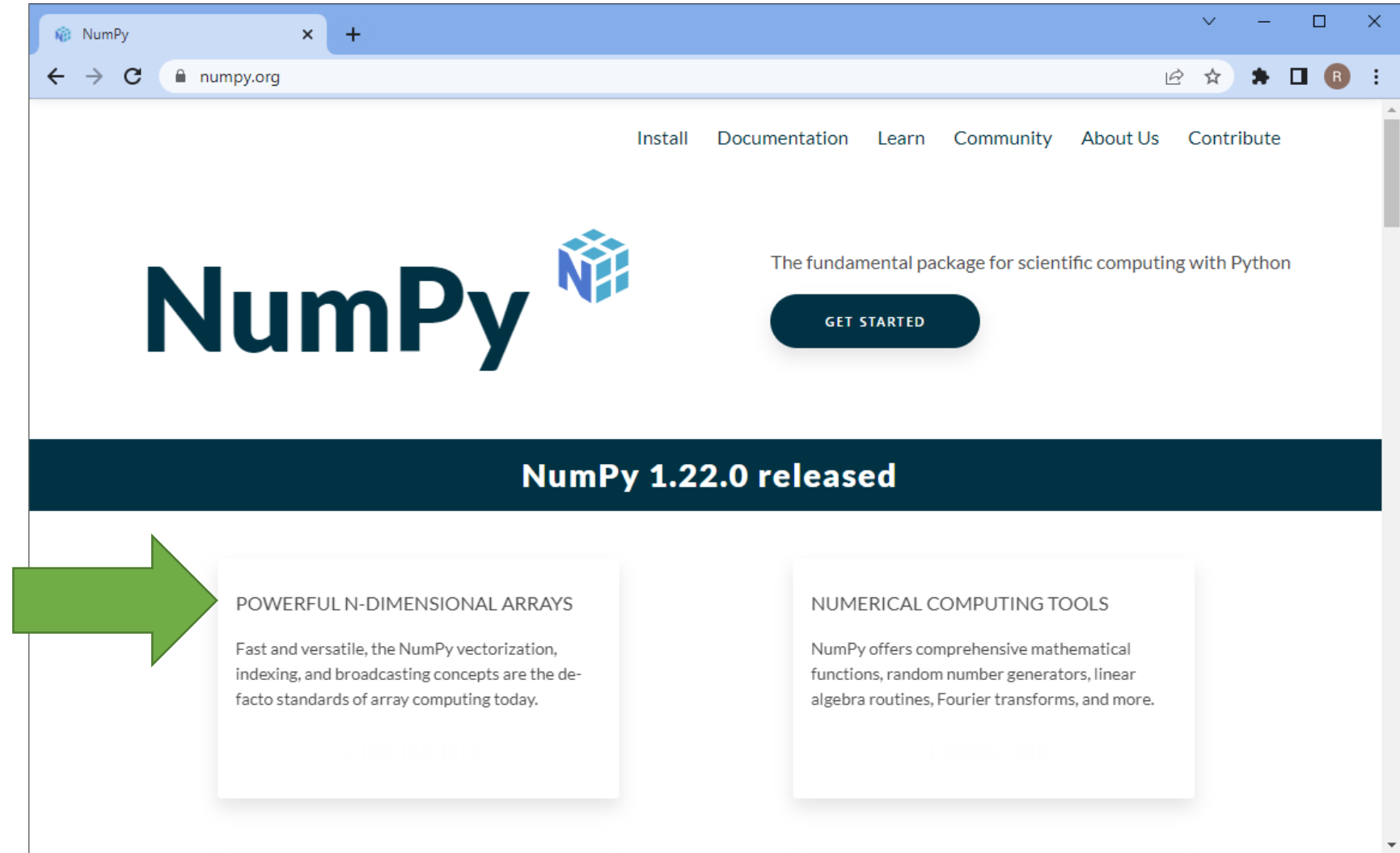
- Sneak preview: By the mid of the semester, we will work with Pandas DataFrames, *fancy* Tables.
- `conda install pandas`
- Among many other features, Pandas allows to visualize tables nicely in Jupyter notebooks.

```
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 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:

1	17	25	3	5	26	12

```
mask = measurements > 10
mask
```

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

```
measurements[mask]
```

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

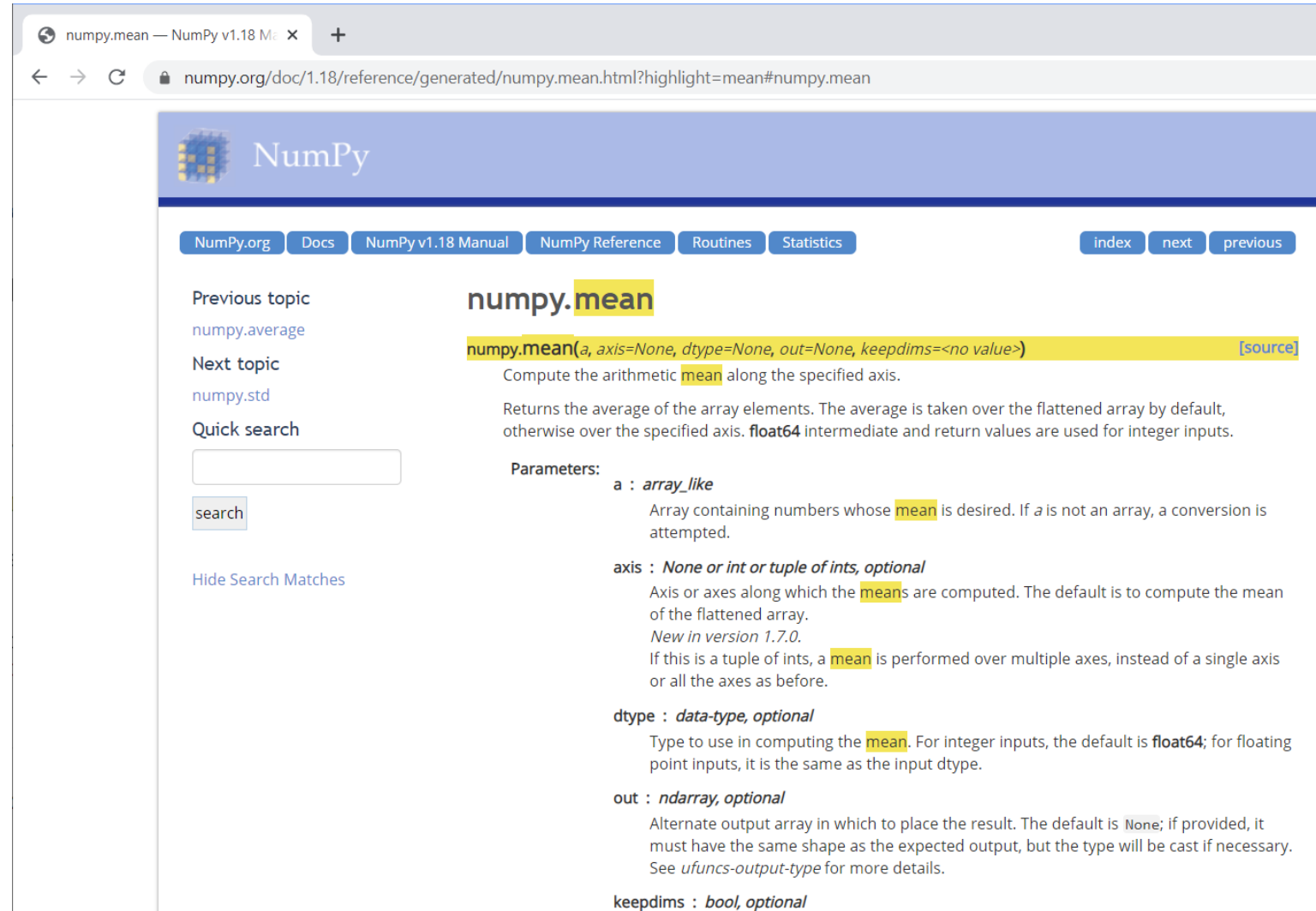
- Basic descriptive statistics

```
import numpy as np

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

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

Mean: 4.0



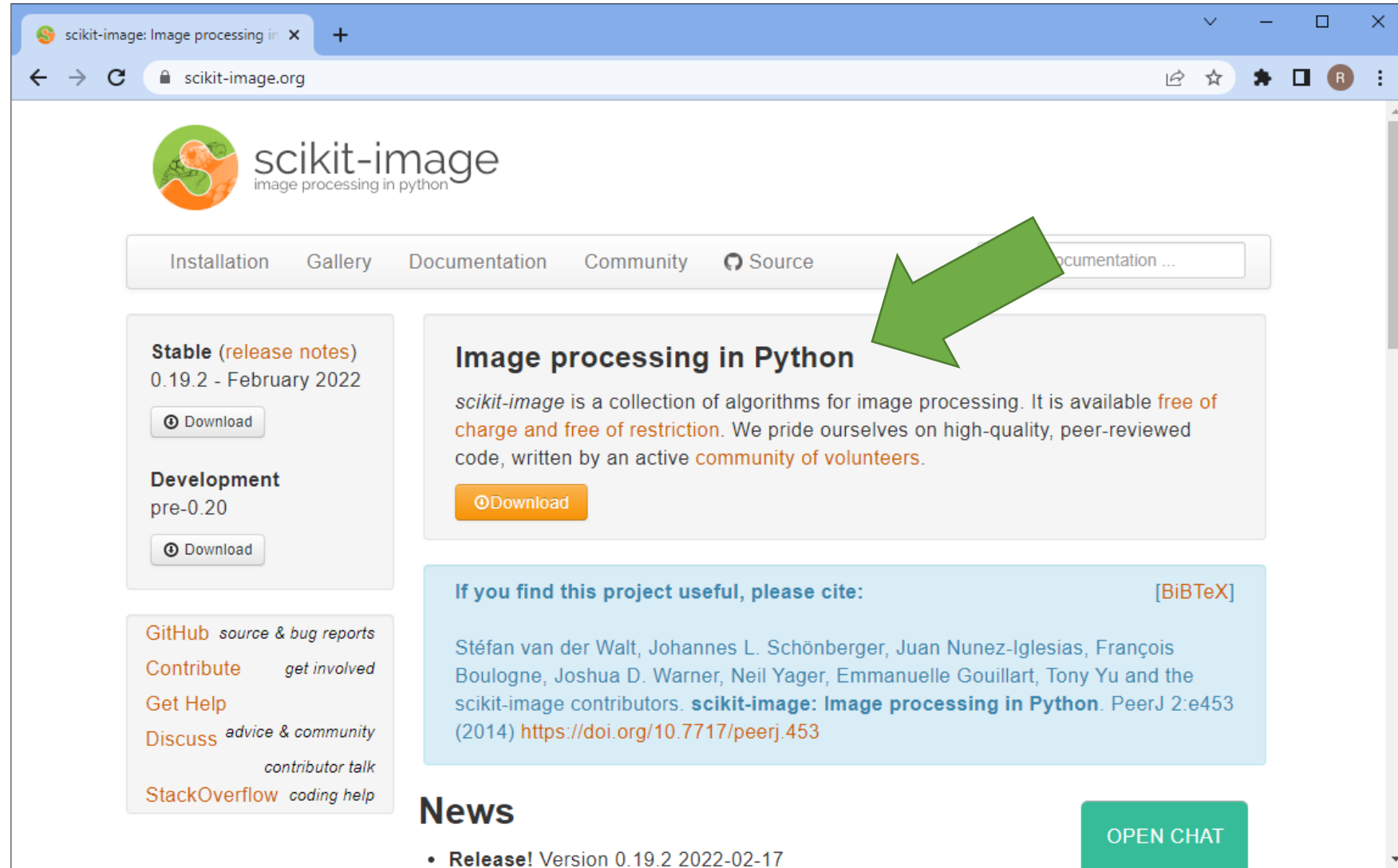
The screenshot shows the NumPy documentation page for the `numpy.mean` function. The page title is "numpy.mean" and the URL is "numpy.org/doc/1.18/reference/generated/numpy.mean.html?highlight=mean#numpy.mean". The page includes a navigation bar with links to "NumPy.org", "Docs", "NumPy v1.18 Manual", "NumPy Reference", "Routines", and "Statistics". The main content area shows the function signature `numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)` and a description: "Compute the arithmetic mean along the specified axis. Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs." The parameters are listed as follows:

- a** : *array_like*
Array containing numbers whose mean is desired. If *a* is not an array, a conversion is attempted.
- axis** : *None or int or tuple of ints, optional*
Axis or axes along which the means are computed. The default is to compute the mean of the flattened array.
New in version 1.7.0.
If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.
- dtype** : *data-type, optional*
Type to use in computing the mean. For integer inputs, the default is `float64`; for floating point inputs, it is the same as the input dtype.
- out** : *ndarray, optional*
Alternate output array in which to place the result. The default is `None`; if provided, it must have the same shape as the expected output, but the type will be cast if necessary. See *ufuncs-output-type* for more details.
- keepdims** : *bool, optional*

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

```
conda install scikit-image
```

- Today, we will only use it for loading and visualizing images.



- 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)
```

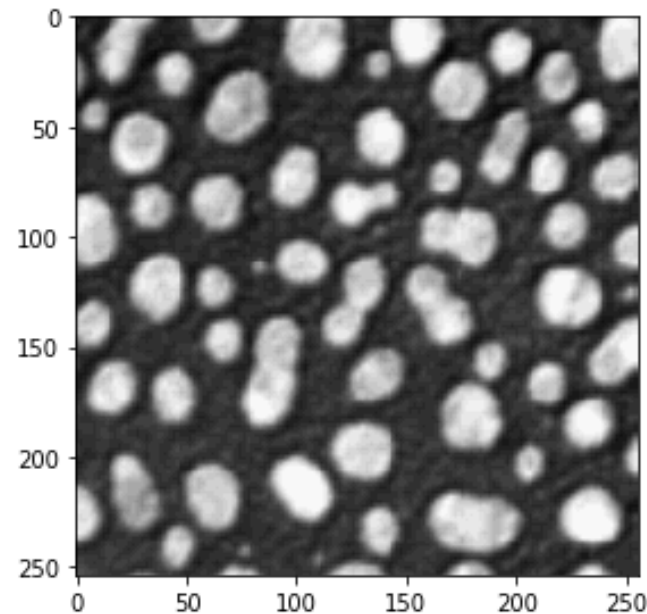
Images are *just* multi-dimensional arrays or “arrays of arrays”.

- Open images
- Visualize images

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

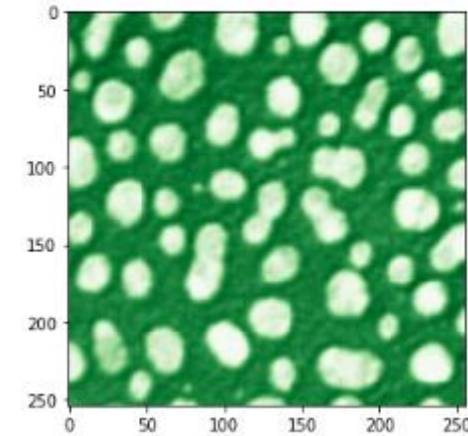
```
from skimage.io import imshow  
imshow(image)
```

<matplotlib.image.AxesImage at 0x245e74>



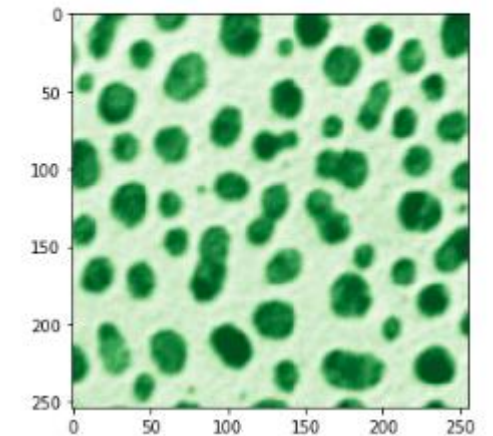
```
imshow(image, cmap="Greens_r")
```

<matplotlib.image.AxesImage at 0x...>



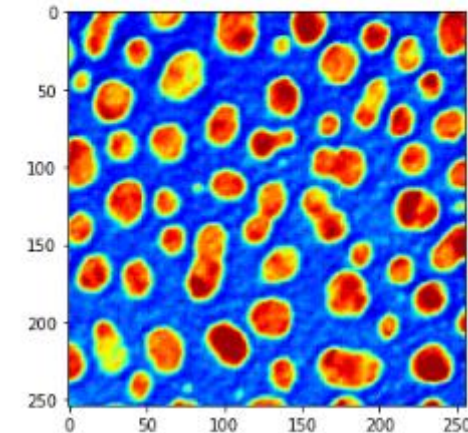
```
imshow(image, cmap="Greens")
```

<matplotlib.image.AxesImage at 0x...>



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

<matplotlib.image.AxesImage at 0x...>

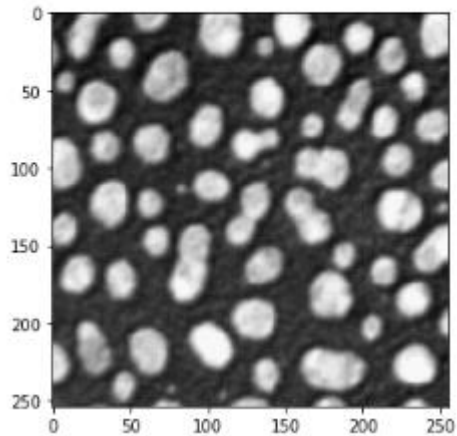


This does not modify the image data. The images are just shown with different colors representing the same values.

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

```
imshow(image)
```

<matplotlib.image.AxesImage at 0x29e

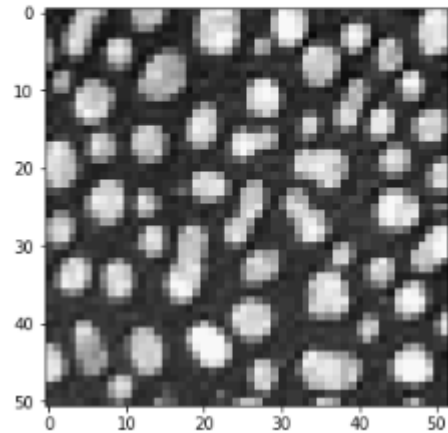


Original image

```
sampled_image = image[::5, ::5]
```

```
imshow(sampled_image)
```

<matplotlib.image.AxesImage at 0x

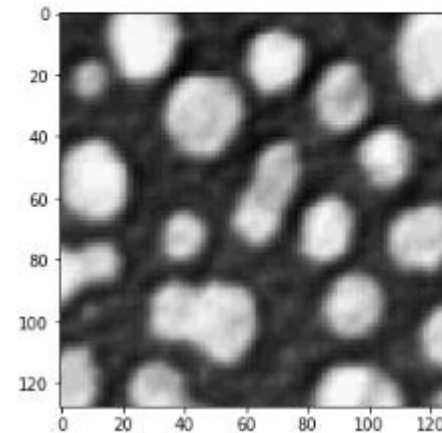


Sub-sampled image

```
cropped_image2 = image[0:128, 128:]
```

```
imshow(cropped_image2)
```

<matplotlib.image.AxesImage at 0x29e

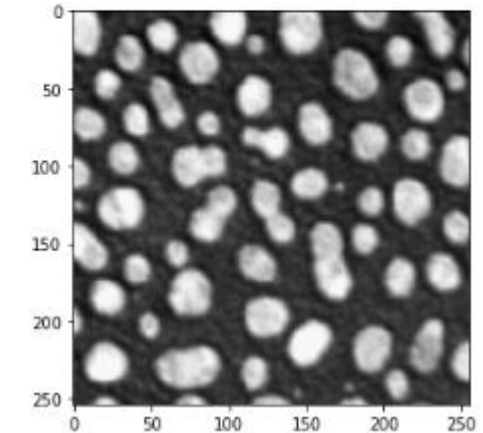


Cropped image

```
flipped_image = image[:, ::-1]
```

```
imshow(flipped_image)
```

<matplotlib.image.AxesImage at 0x



Flipped image

If your program throws error messages:

- Don't panic.
- *"There are two ways to write error-free programs; only the third one works."*

Alan J. Perlis, Yale University

- Read where the error happened.
 - You may see your fault immediately, when looking at the right point.
- Read what appears to be wrong.
 - If you know, what's missing, you may see it, even if it's missing in a slightly different place.
 - Sometimes, something related is missing

```
▶ print(round(4.5))
```

File "<ipython-input-15-09a9be4a90c5>", line 1
print(round(4.5))
^

SyntaxError: unexpected EOF while parsing

Take home messages

- Arrays can be accessed like this:

`data[start:stop:step]`

- Strings are arrays
- Lists are arrays
- Tuples are arrays
- Dictionaries are arrays with named elements
- Columns in tables are arrays
- Images are multi-dimensional arrays
- Learning how to deal with arrays in Python is key.

Coming up next

- Loops
- Conditions
- Functions
- Libraries

```
▶ animal_set = ["Cat", "Dog", "Mouse"]  
  
for animal in animal_set:  
    print(animal)
```

```
Cat  
Dog  
Mouse
```