Introduction to coding with

Workshop 1 – 13-09-2024

Willem Kroese – w.s.j.kroese@uu.nl – BBG 6.17
Institute of Marine and Atmospheric Research

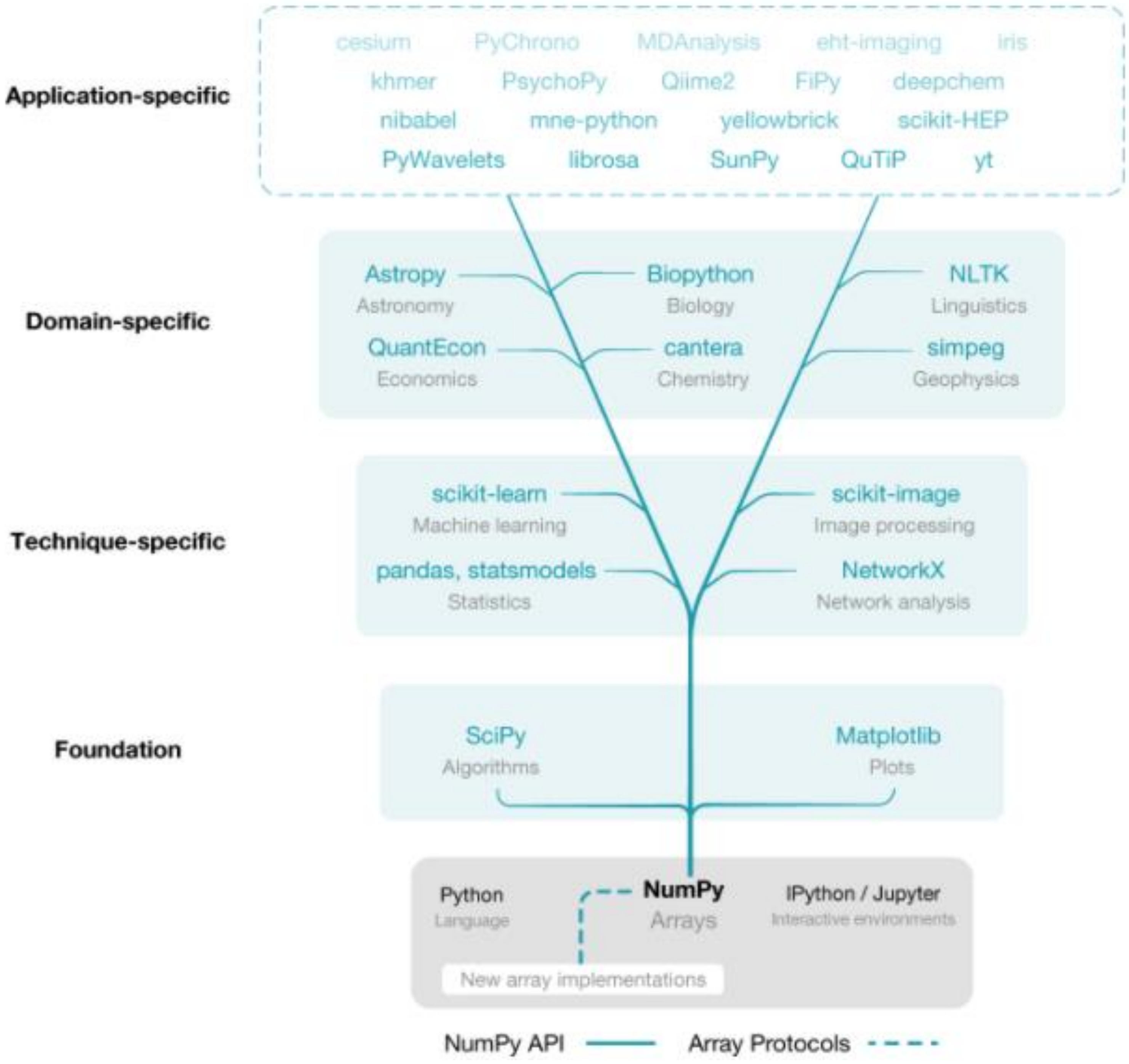
Module 0

- Notebooks:
  - Cell types
  - Creating and deleting cells
- Python:
  - Data types
  - Lists and dictionaries
  - For and while loops
  - Conditionals
  - Functions
  - Classes

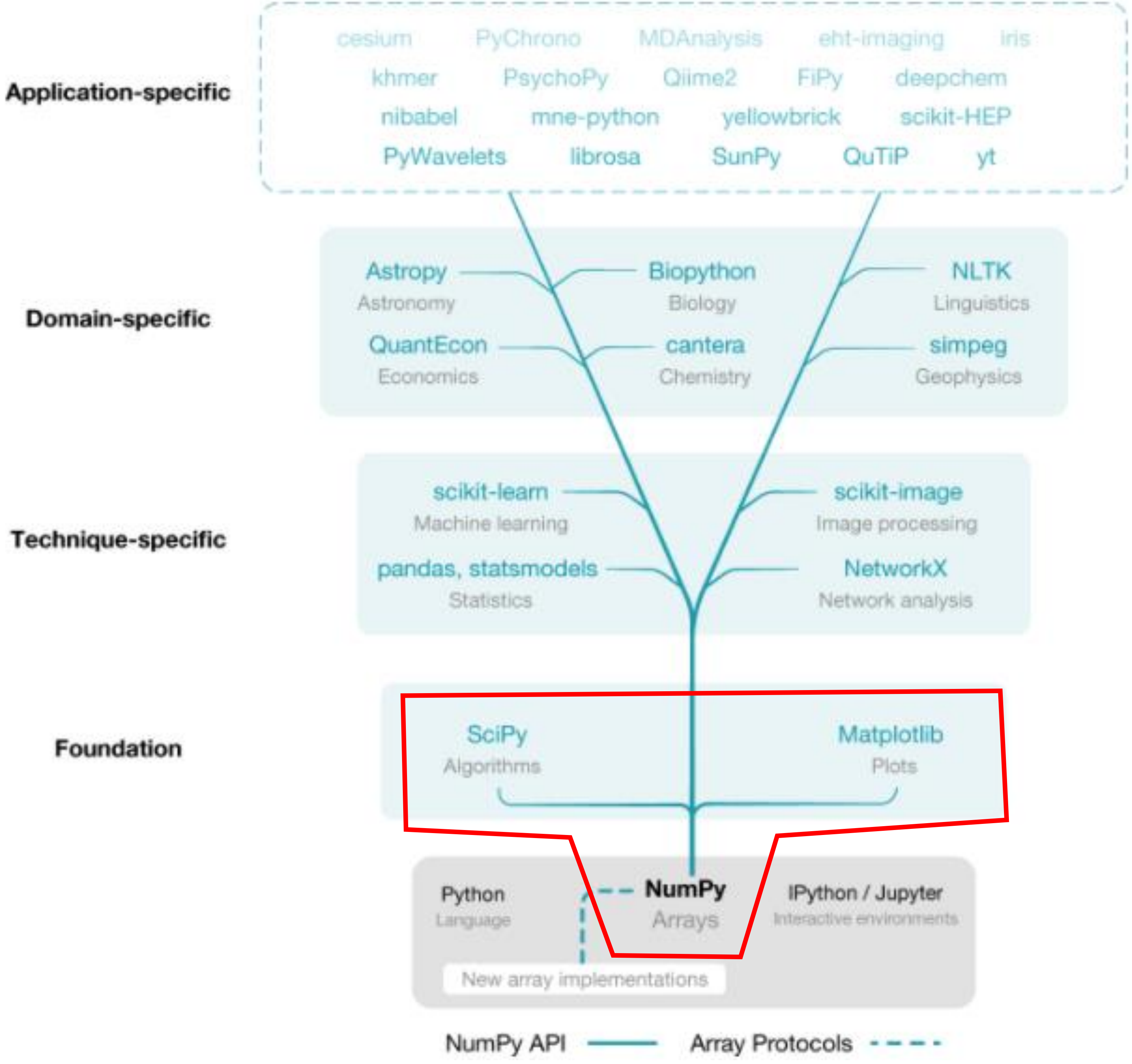
Today

- A few slides on general python stuff
- Packages: numpy, matplotlib, and scipy
  - Arrays, indexing, slicing, matrix operations
  - Plotting data, contour plot, subplots, quiver plots
  - Fitting data

Modules


Application-specific
cesium
PyChrono
MDAnalysis
eht-imaging
iris
khmer
PsychoPy
Qiime2
FiPy
deepchem
nibabel
mne-python
yellowbrick
scikit-HEP
PyWavelets
librosa
SunPy
QuTiP
yt
Domain-specific
Astropy
Astronomy
Biopython
Biology
NLTK
Linguistics
QuantEcon
Economics
cantera
Chemistry
simpeg
Geophysics
Technique-specific
scikit-learn
Machine learning
scikit-image
Image processing
pandas, statsmodels
Statistics
NetworkX
Network analysis
Foundation
SciPy
Algorithms
Matplotlib
Plots
Python
Language
NumPy
Arrays
IPython / Jupyter
Interactive environments
New array implementations
NumPy API
Array Protocols

Modules


Application-specific
cesium PyChrono MDAnalysis eht-imaging iris
khmer PsychoPy Qiime2 FiPy deepchem
nibabel mne-python yellowbrick scikit-HEP
PyWavelets librosa SunPy QuTiP yt
Domain-specific
Astropy
Astronomy
Biopython
Biology
NLTK
Linguistics
QuantEcon
Economics
cantera
Chemistry
simpeg
Geophysics
Technique-specific
scikit-learn
Machine learning
scikit-image
Image processing
pandas, statsmodels
Statistics
NetworkX
Network analysis
Foundation
SciPy
Algorithms
Matplotlib
Plots
Python
Language
NumPy
Arrays
IPython / Jupyter
Interactive environments
New array implementations
NumPy API
Array Protocols

Comments in your code

- Put comments in your code to explain what the code does:
  - helps others to understand your code
  - helps you to understand your own code, for example:
    - when you haven't looked at it for a while
    - if you are trying to track down errors

- Comments: type a # before the text
- Python will skip this text when running a cell

Comments in your code

Good example

```
def retrieve_weight(a,b,x):
    '''
    a,b, and x can be floats or numpy arrays with equal size.
    returns the fraction between a and b where x is.
    weights are used for linear interpolation
    '''
    return (x-a)/(b-a)
```

Bad example

```
def lin_to_grid(data):
    lat,lon,day,labels = data['lat'].to_numpy(),data['lon'].to_numpy(),data['day'].to_numpy(),data['labels'].to_numpy()
    gridded_labels = np.full((180,360,len(np.unique(day))),-1.)
    gridded_labels[lat.astype(int)+90,lon.astype(int)+180,day.astype(int)-np.min(day).astype(int)] =labels
    return gridded_labels
```

Comments in your code

```python
def lin_to_grid(data):
    """
    Convert the 'labels' column of a pandas DataFrame from linear to gridded,
    using the 'lat', 'lon' and 'day' columns. Gridsize is 1x1 degree.

    Parameters
    ----------
    data : pandas.DataFrame
        Dataframe with columns 'lat', 'lon', 'day' and 'labels'.

    Returns
    -------
    gridded_labels : numpy.ndarray
        Gridded data with shape (180, 360, len(np.unique(day))). The first
        dimension is the latitude, the second dimension is the longitude and the
        third dimension is the day of the year. The values of the gridded data are
        the 'labels' column of the input data.
    """
    lat,lon,day,labels = data['lat'].to_numpy(),data['lon'].to_numpy(),data['day'].to_numpy(),data['labels'].to_numpy()
    # Create an empty array with the correct shape
    gridded_labels = np.full((180,360,len(np.unique(day))),-1.)
    # Fill the array with the labels from the input data. We convert lat and lon to indices by adding 90 and 180
    gridded_labels[lat.astype(int)+90,lon.astype(int)+180,day.astype(int)-np.min(day).astype(int)] = labels
    return gridded_labels
```

Dealing with errors

At some point, you will get error messages.. Don't panic, here is what you should do:

- Try to understand the error written below your cell and in which line it occurs. Often the most relevant information of an error message is at the bottom of the message.
- If it's not exactly clear where the error occurs: simplify your code and add the other parts piece by piece. This building up process is generally considered good practice while coding.
- Do a search online: if you run into a problem, it is very likely that someone else experienced the same before you.
- Look at the package documentation

Don't ask for my help if you haven't looked the error up online!

Online resources

- You don’t always need to reinvent the wheel. There is a lot of code online, code in these workshops, code in assignments, etc. you can (re)use.

But (!):

- Always try to understand what the code does and how it works, and if it’s correct.
- Give credit if required (copyright).

Want to practice more?

- Datacamp course
  - https://www.datacamp.com/courses/intro-to-python-for-data-science
- W3schools course
  - https://www.w3schools.com/python/python_intro.asp

Notebooks for today

- Module 1a – NumPy
- Module 1a – Matplotlib
- Module 1a – SciPy

On Blackboard (course content ACCP) and GitHub!

First read the code in the cells, and then run them. See if you understand the output. Exercises are at the bottom.

Appendix

Importing Modules

- Most of useful functionalities of Python come from so-called packages or libraries (most already come with Anaconda).

- To use a library/package:
  1. import the package into your code

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

ALWAYS start your notebook with this!

Otherwise you have to type `matplotlib.pyplot` everytime you use it

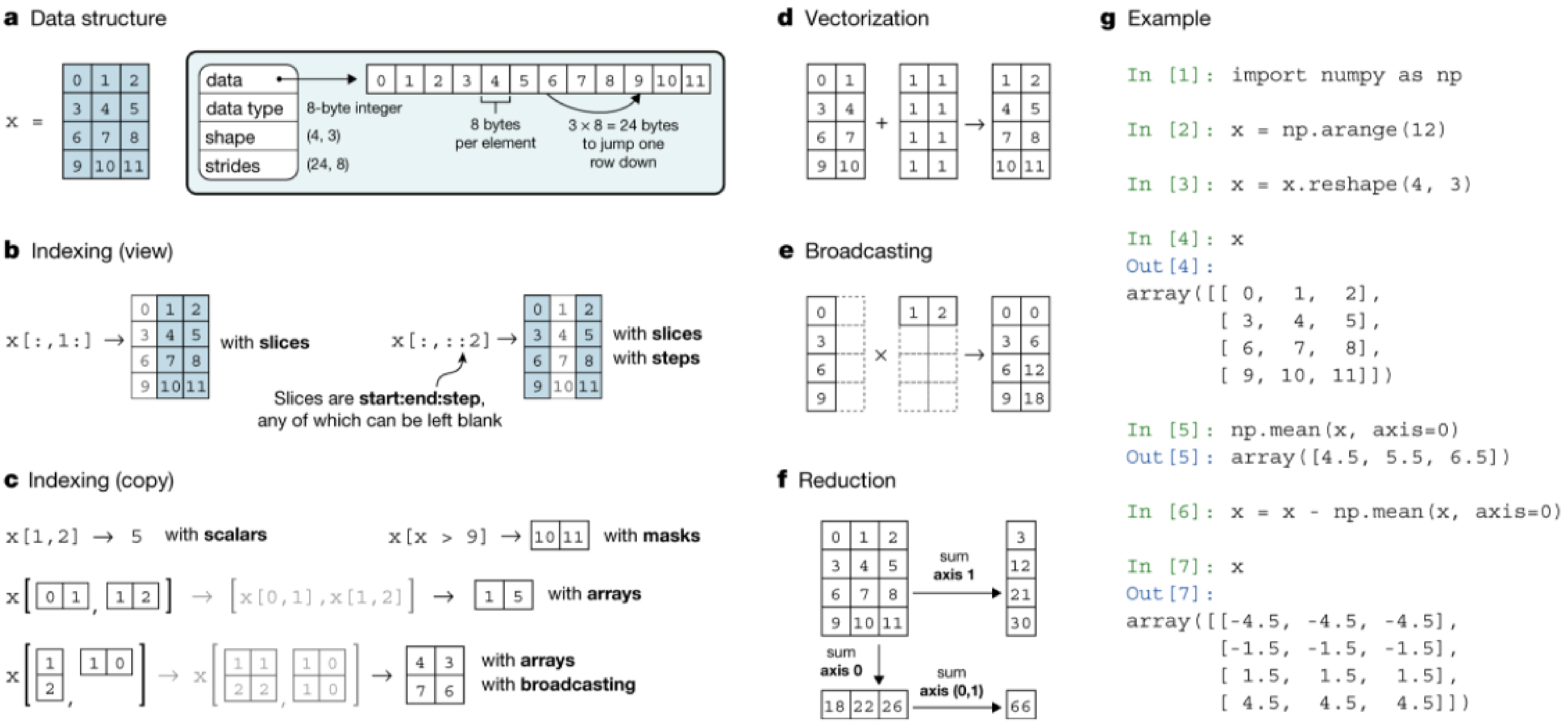
  2. use functions from the package by typing:

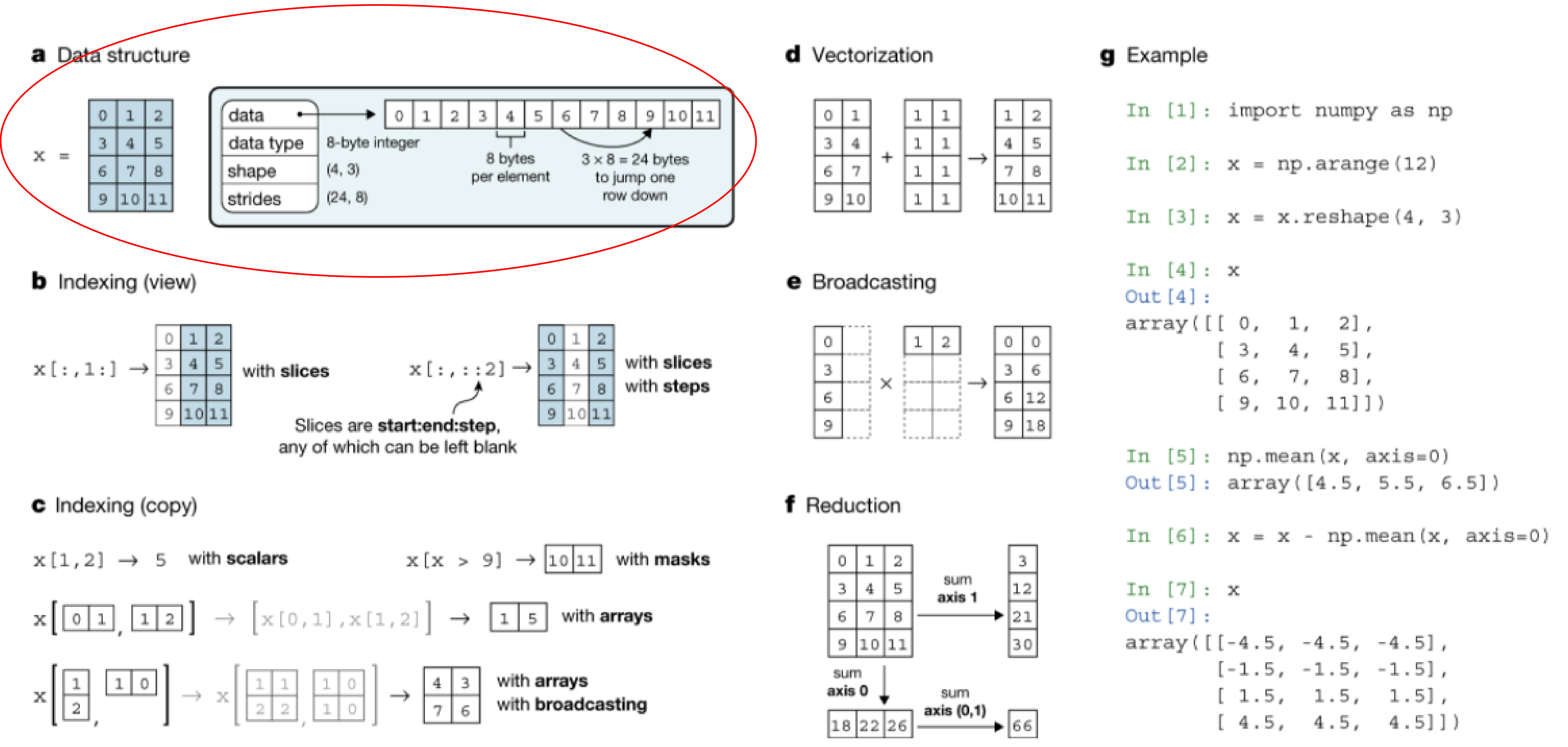
```
package.function_name

plt.function_name
np.function_name
```

NumPy

- Core is `ndarray` object: n-dimensional arrays of homogeneous data types

- All kinds of built-in operations for these data types
  → efficient way of dealing with large datasets


a Data structure
x =
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5 with scalars
x[x > 9] → with masks
x[0,1],x[1,2]
with arrays
with arrays with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)
g Example
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)
In [4]: x
Out[4]:
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]])
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
In [6]: x = x - np.mean(x, axis=0)
In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
[-1.5, -1.5, -1.5],
[1.5, 1.5, 1.5],
[4.5, 4.5, 4.5]])


a Data structure
x =
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices
with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5 with scalars
x[x > 9] → with masks
with arrays
x[0,1],x[1,2]
with arrays
with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)
g Example
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)
In [4]: x
Out[4]:
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]])
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
In [6]: x = x - np.mean(x, axis=0)
In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
[-1.5, -1.5, -1.5],
[1.5, 1.5, 1.5],
[4.5, 4.5, 4.5]])

Vectors and arrays

- `np.linspace(start,stop,number)`: creates a vector from `start` to `stop` of `number` linearly spaced numbers.

- `np.array([list])`: creates a NumPy array from a list.

- `np.arange(start, stop, step)`: creates a vector from `start` to `stop` with stepsize `step`.

Vectors and arrays

- `np.zeros(n)` = array full of zeros
- `np.ones(n)` = array full of ones
- `np.full(n,value)` = array of full with value `value`

Vectors and arrays

- `np.zeros(n)` = array full of zeros
- `np.ones(n)` = array full of ones
- `np.full(n,value)` = array of full with value `value`

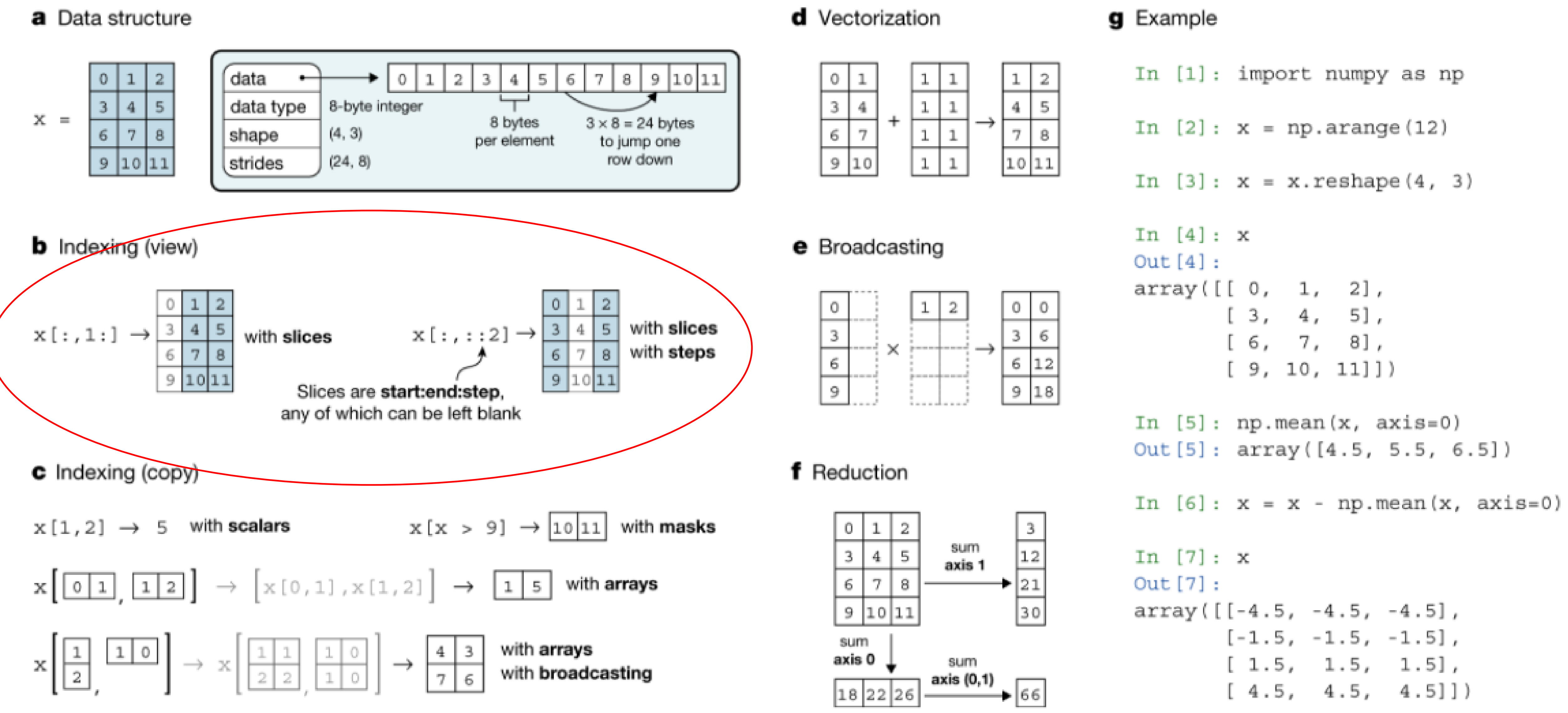
- `n` can be multidimensional: `c = np.zeros((9,9))`

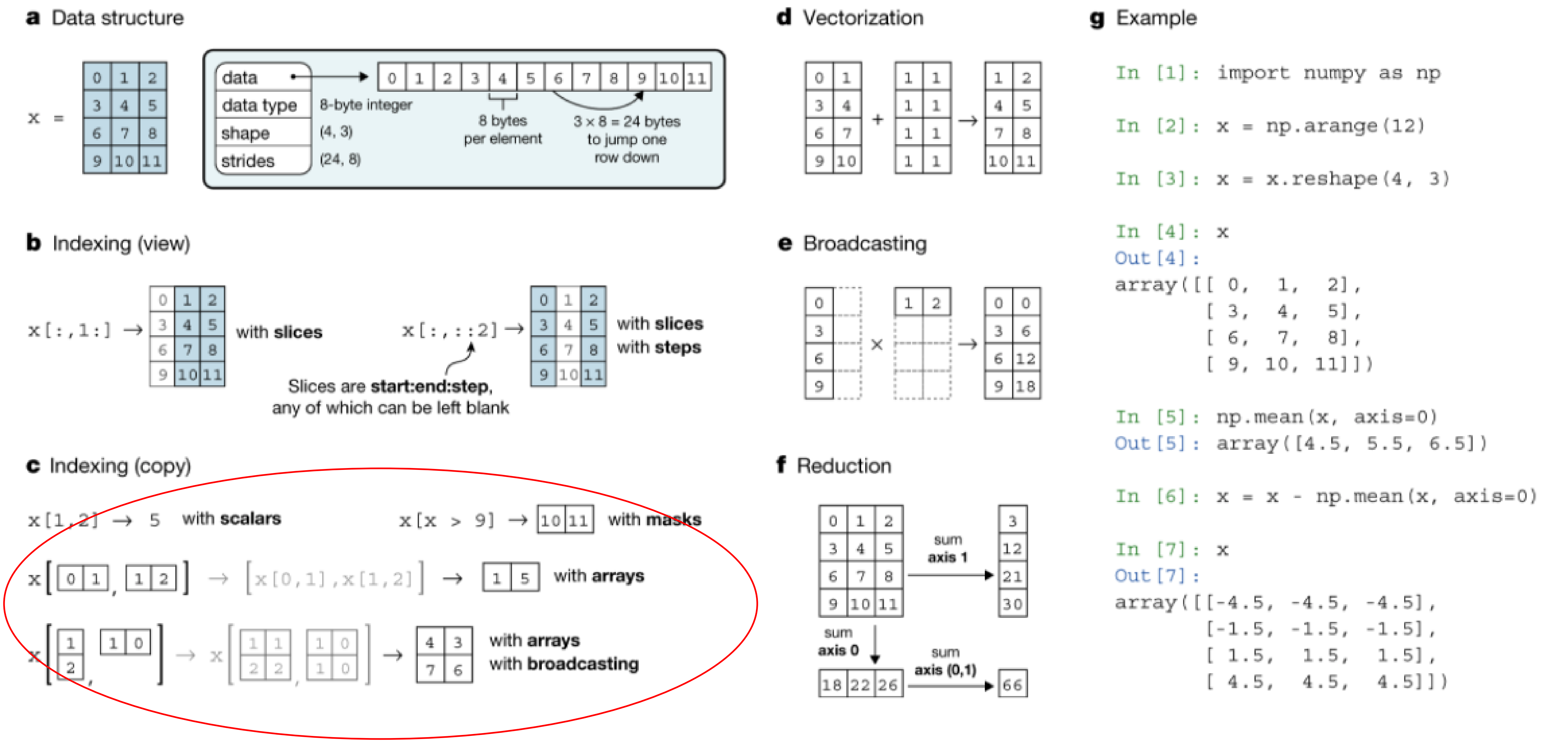
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0.,
0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 0.,
0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 0.], [0.,
0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0.,
0., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 0.,
0., 0., 0., 0., 0.]])
```

Vectors and arrays

- `np.zeros(n)` = array full of zeros
- `np.ones(n)` = array full of ones
- `np.full(n,value)` = array of full with value `value`

- `n` can be multidimensional: `c = np.zeros((9,9))`
- `c.shape = (9,9)`


a Data structure
x =
0 1 2
3 4 5
6 7 8
9 10 11
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
0 1 2 3 4 5 6 7 8 9 10 11
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices
with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5 with scalars
x[x > 9] → 10 11 with masks
x[[0 1], [1 2]] → [x[0,1], x[1,2]] → 1 5 with arrays
with arrays
with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)
18 22 26
66
g Example
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)
In [4]: x
Out[4]:
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]])
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
In [6]: x = x - np.mean(x, axis=0)
In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
[-1.5, -1.5, -1.5],
[1.5, 1.5, 1.5],
[4.5, 4.5, 4.5]])



a Data structure
x =
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5 with scalars
x[x > 9] → with masks
with arrays
x[0,1],x[1,2]
with arrays with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)
g Example
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)
In [4]: x
Out[4]:
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]])
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
In [6]: x = x - np.mean(x, axis=0)
In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
[-1.5, -1.5, -1.5],
[1.5, 1.5, 1.5],
[4.5, 4.5, 4.5]])

Indexing

- Zero-based indexing!

In [10]:
```
# two dimensional grids
x = np.linspace(-2*np.pi, 2*np.pi, 10)
y = np.linspace(-np.pi, np.pi, 5)
xx, yy = np.meshgrid(x, y)
xx.shape, yy.shape
```
Out[10]:
```
((5, 10), (5, 10))
```

In [12]:
```
# get some individual elements of xx
xx[0,0], xx[-1,-1], xx[3,-5]
```
Out[12]:
```
(-6.283185307179586, 6.283185307179586, 0.6981317007977319)
```

Indexing

- Zero-based indexing!

In [13]:
```
# get some whole rows and columns
xx[0,:].shape, xx[:,-1].shape
```
Out[13]:
```
((10,), (5,))
```

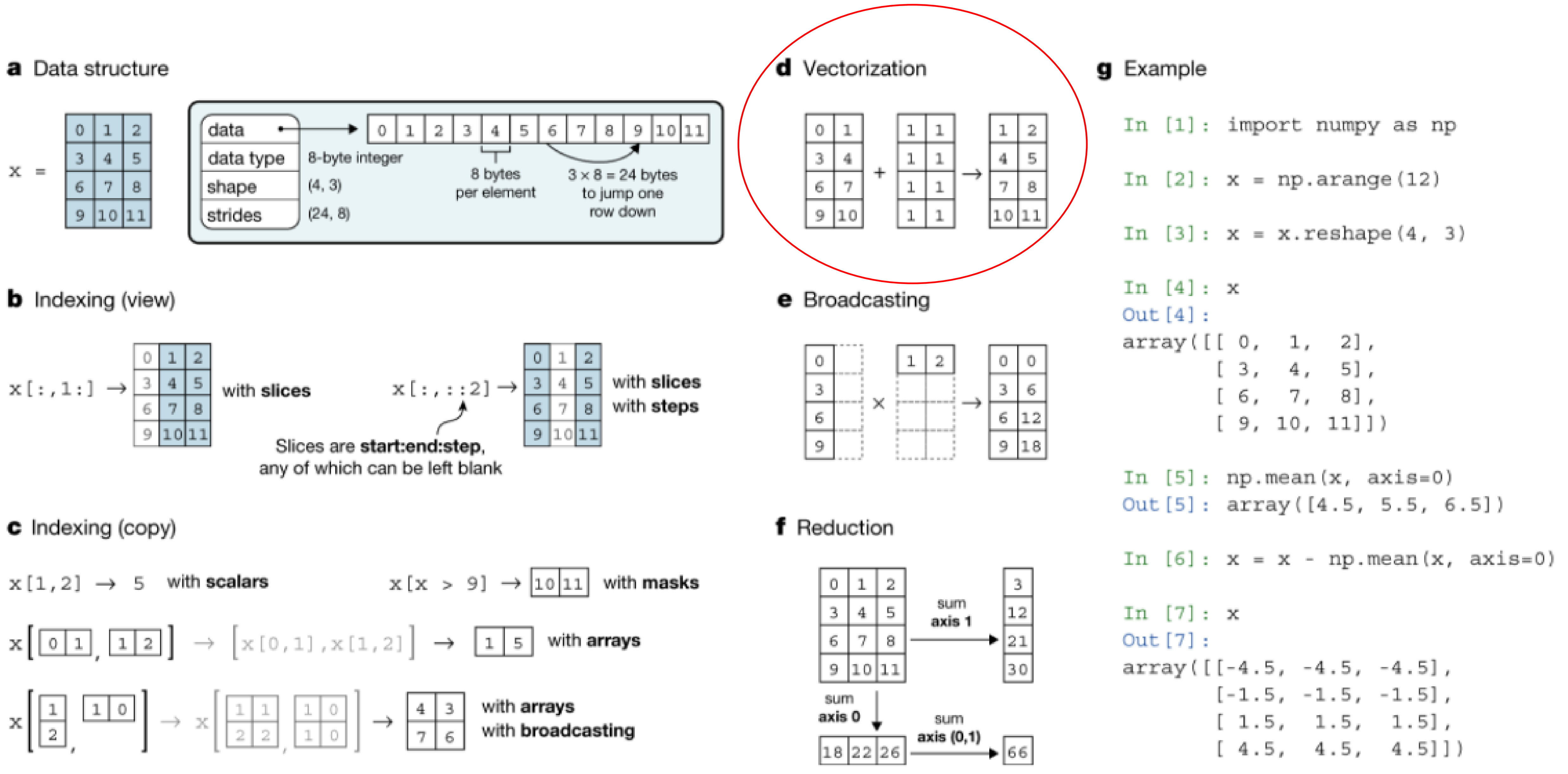
In [15]:
```
# get some ranges, this is again left-inclusive, right-exclusive
print(xx[2:5,3:4].shape)
xx[2:5,3:4]
```
```
(3, 1)
```
Out[15]:
```
array([[-2.0943951],
       [-2.0943951],
       [-2.0943951]])
```

Indexing

- Zero-based indexing!

```
In [16]:
# use a boolean array as an index
idx = xx<0
yy[idx]
idx
```

```
Out[16]: array([[ True,  True,  True,  True,  True, False, False, False, False,
         False],
        [ True,  True,  True,  True,  True, False, False, False, False,
         False],
        [ True,  True,  True,  True,  True, False, False, False, False,
         False],
        [ True,  True,  True,  True,  True, False, False, False, False,
         False],
        [ True,  True,  True,  True,  True, False, False, False, False,
         False]])
```


a Data structure
x =
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices
with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5 with scalars
x[x > 9] → with masks
x[0,1],x[1,2]
with arrays
with arrays
with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)
g Example
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)
In [4]: x
Out[4]:
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]])
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
In [6]: x = x - np.mean(x, axis=0)
In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
[-1.5, -1.5, -1.5],
[1.5, 1.5, 1.5],
[4.5, 4.5, 4.5]])

Vectorization

- **Vectorization** (in Python context) = applying operations to whole arrays instead of individual elements

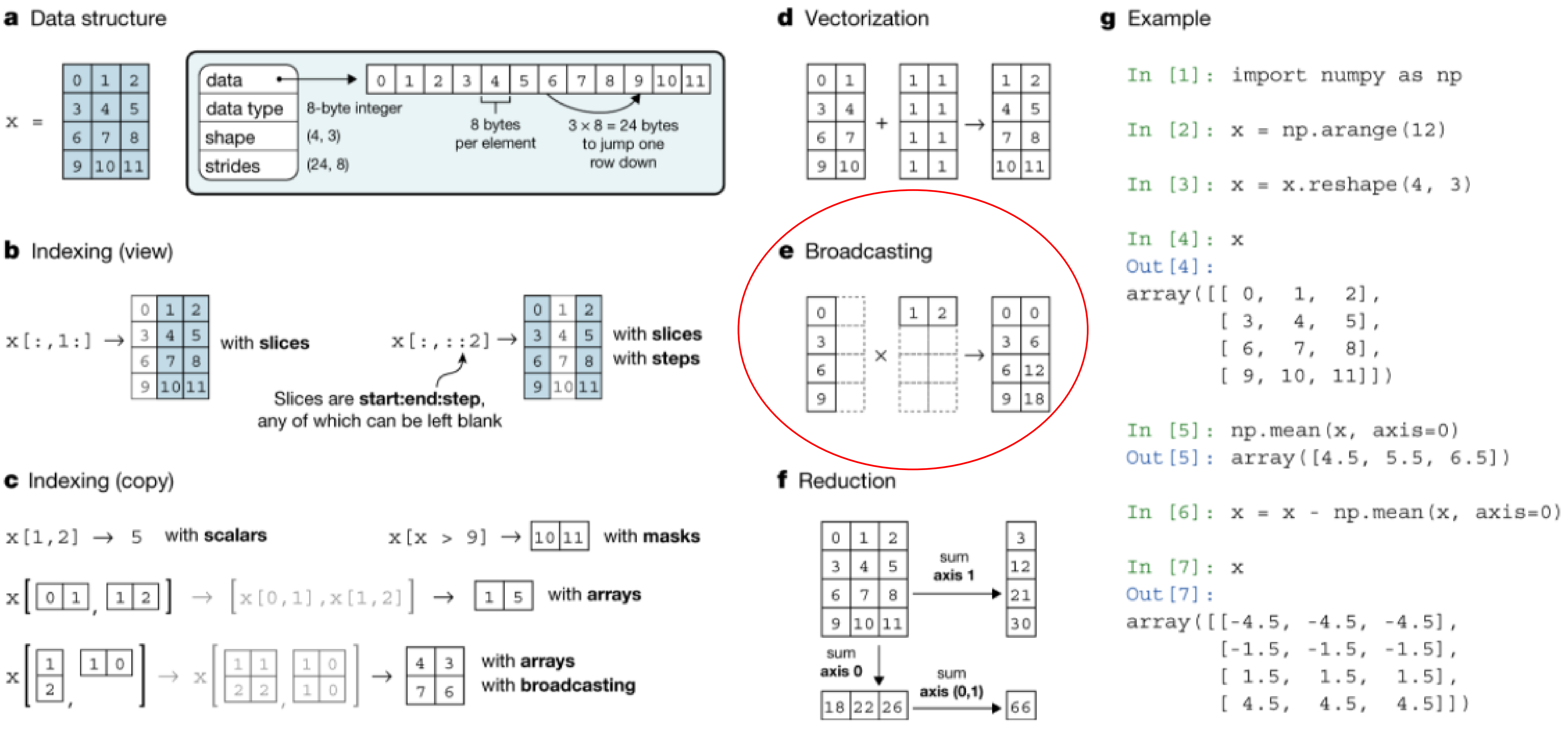
Vectorization

- **Vectorization** (in Python context) = applying operations to whole arrays instead of individual elements
- A lot more efficient than loops!

Vectorization

- **Vectorization** (in Python context) = applying operations to whole arrays instead of individual elements
- A lot more efficient than loops!

- `np.log(xx)`
- `np.sin(xx)`
- `np.cos(xx)`
- `np.exp(xx)`
- `np.pi`


a Data structure
x =
0 1 2
3 4 5
6 7 8
9 10 11
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
0 1 2 3 4 5 6 7 8 9 10 11
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices
with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5 with scalars
x[x > 9] → 10 11 with masks
x[0 1, 1 2] → [x[0,1],x[1,2]] → 1 5 with arrays
with arrays
with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)
18 22 26
66
g Example
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)
In [4]: x
Out[4]:
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]])
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
In [6]: x = x - np.mean(x, axis=0)
In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
[-1.5, -1.5, -1.5],
[1.5, 1.5, 1.5],
[4.5, 4.5, 4.5]])

Broadcasting

- **Broadcasting** (in Python context) = operations on arrays with different shapes

Broadcasting

- **Broadcasting** (in Python context) = operations on arrays with different shapes
- Dimensions are compatible when:
  - they have the same length
  - one of them is 1

→ pay attention to the shape of your arrays!

Broadcasting

- **Broadcasting** (in Python context) = operations on arrays with different shapes
- Dimensions are compatible when:
  - they have the same length
  - one of them is 1

  → pay attention to the shape of your arrays!
- `F = np.zeros((5,10))`
- `X = np.linspace(0,2*np.pi,10)`

What are their shapes?

Broadcasting

- **Broadcasting** (in Python context) = operations on arrays with different shapes
- Dimensions are compatible when:
  - they have the same length
  - one of them is 1

  → pay attention to the shape of your arrays!
- `F = np.zeros((5,10))` → `(5,10)` 5 rows, 10 columns
- `X = np.linspace(0,2*np.pi,10)` → `(10,)` 10 rows, 1 column

Broadcasting

- **Broadcasting** (in Python context) = operations on arrays with different shapes
- Dimensions are compatible when:
  - they have the same length
  - one of them is 1

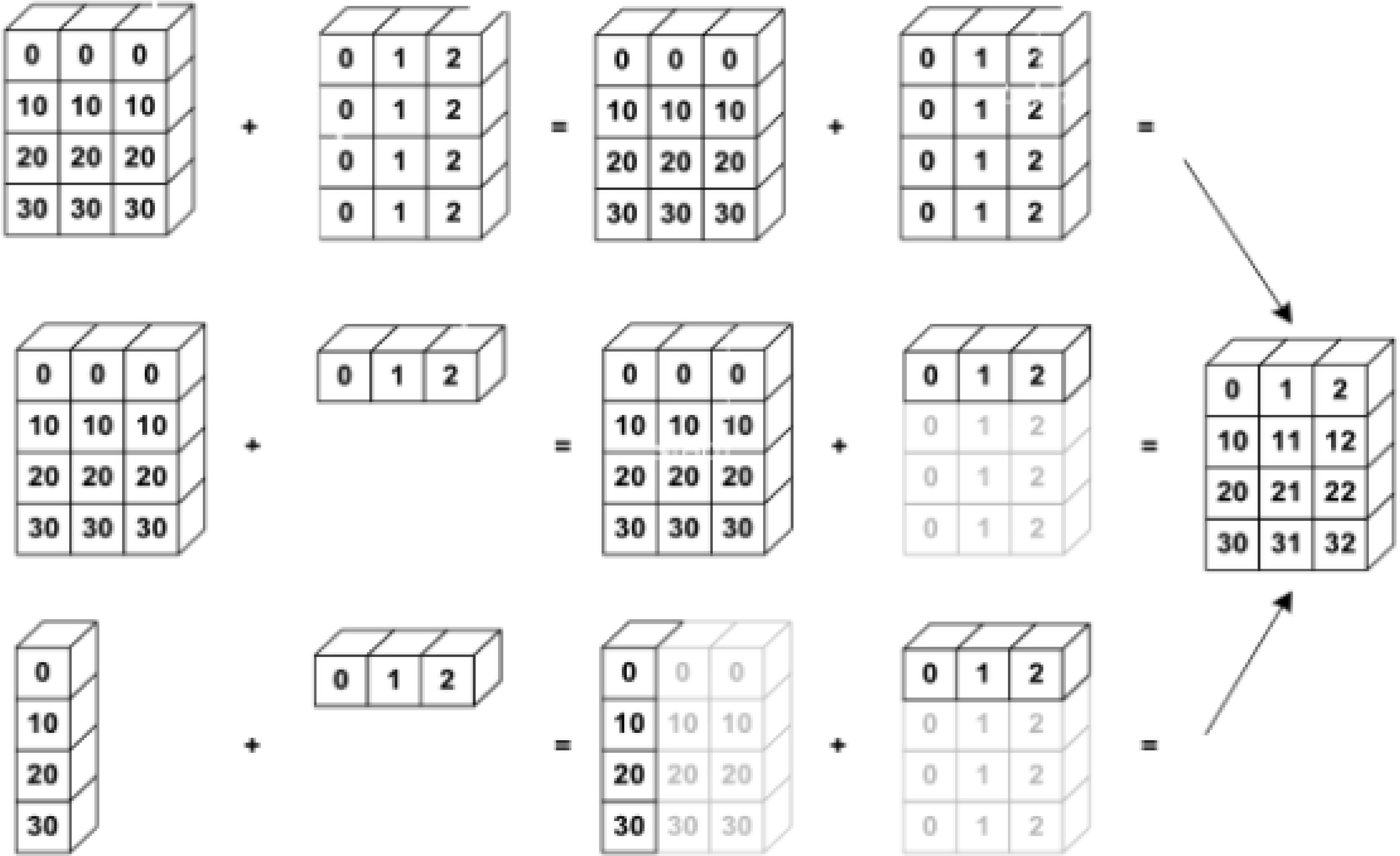
  → pay attention to the shape of your arrays!
- `F = np.zeros((5,10))` → `(5,10)`
- `X = np.linspace(0,2*np.pi,10)` → `(10,)`
- `D = F + X`

Broadcasting

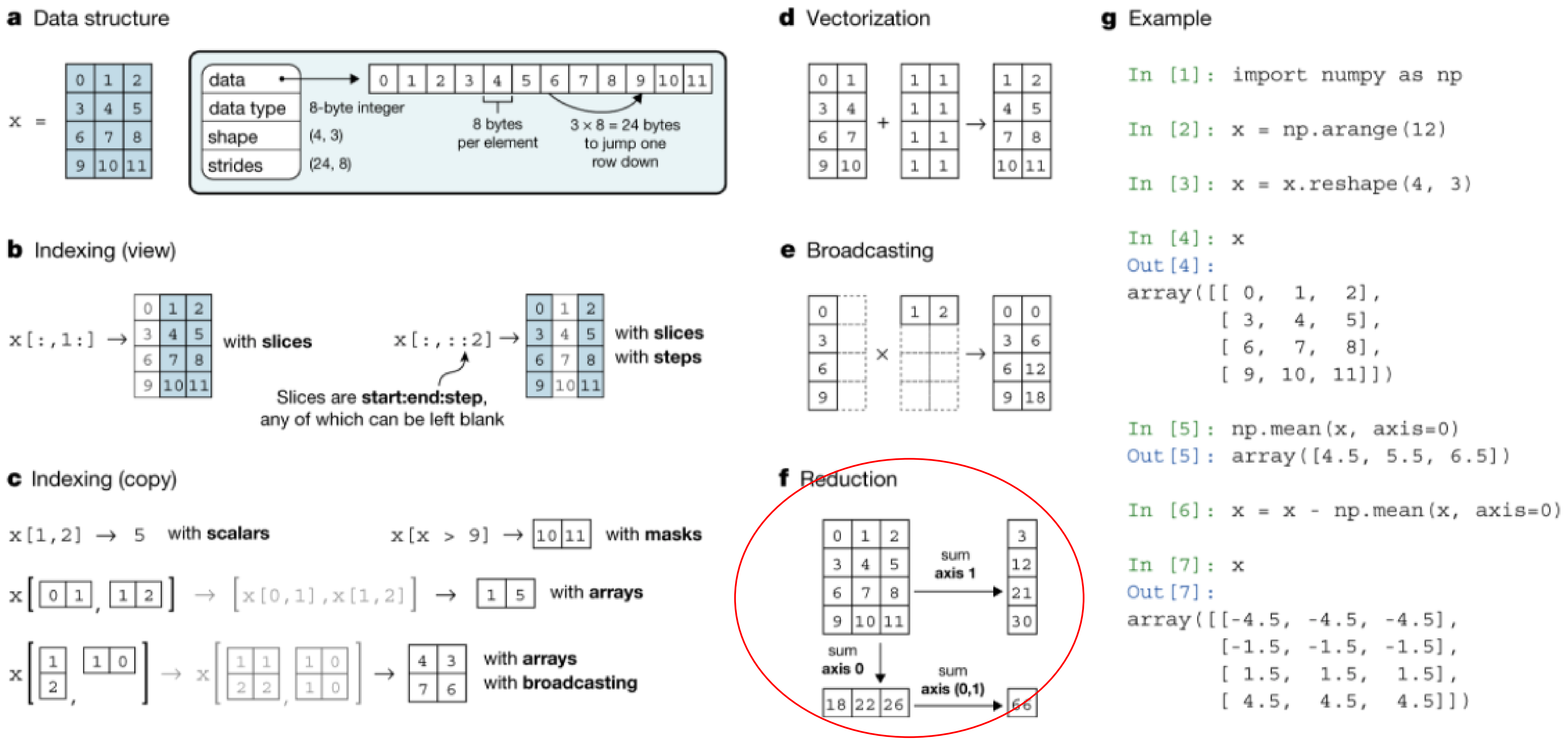
- **Broadcasting** (in Python context) = operations on arrays with different shapes
- Dimensions are compatible when:
  - they have the same length
  - one of them is 1

  → pay attention to the shape of your arrays!
- `F = np.zeros((5,10))` → `(5,10)`
- `X = np.linspace(0,2*np.pi,10)` → `(10,)`
- `D = F + X`
- `G = F * X`

What if `X` had shape `(5,)`?



0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 1 2
10 11 12
20 21 22
30 31 32
0
10
20
30
+
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=


a Data structure
x =
data
data type
shape
strides
8-byte integer
(4, 3)
(24, 8)
8 bytes per element
3 × 8 = 24 bytes to jump one row down
b Indexing (view)
x[:,1:] →
with slices
x[:,::2] →
with slices
with steps
Slices are start:end:step, any of which can be left blank
c Indexing (copy)
x[1,2] → 5 with scalars
x[x > 9] → with masks
x[0,1],x[1,2]
with arrays
with arrays
with broadcasting
d Vectorization
e Broadcasting
f Reduction
sum axis 1
sum axis 0
sum axis (0,1)
g Example
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)
In [4]: x
Out[4]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8],
       [9, 10, 11]])
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
In [6]: x = x - np.mean(x, axis=0)
In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
       [-1.5, -1.5, -1.5],
       [1.5, 1.5, 1.5],
       [4.5, 4.5, 4.5]])

Reduction

- **Reduction** (in Python context) = operations that collapse one or more dimension

Reduction

- **Reduction** (in Python context) = operations that collapse one or more dimension
- `X.sum()`
- `X.mean()`
- `X.std()`
- `X.max()`
- `X.min()`

Reduction

- **Reduction** (in Python context) = operations that collapse one or more dimension
- `X.sum()`
- `X.mean()`
- `X.std()`
- `X.max()`
- `X.min()`
- If an array has more than 1 dimension, you can also choose over which dimension to perform the computation

Matplotlib

- Library for visualizing and plotting data

Matplotlib

- Library for visualizing and plotting data
- Lineplots, scatterplots and contourplots

Plotting

Example of plotting a cosine wave:

- Combination of libraries `pyplot` and `numpy`
- Create an array *x* of 20 equally-spaced numbers between 0 and 2π:

```
x = np.linspace(0, 2*np.pi, 20)
```

- Use function plot:

```
plt.plot(x, np.cos(x))
plt.xlabel('x')
plt.ylabel('y')
plt.title('y = cos(x)')
```

Plotting

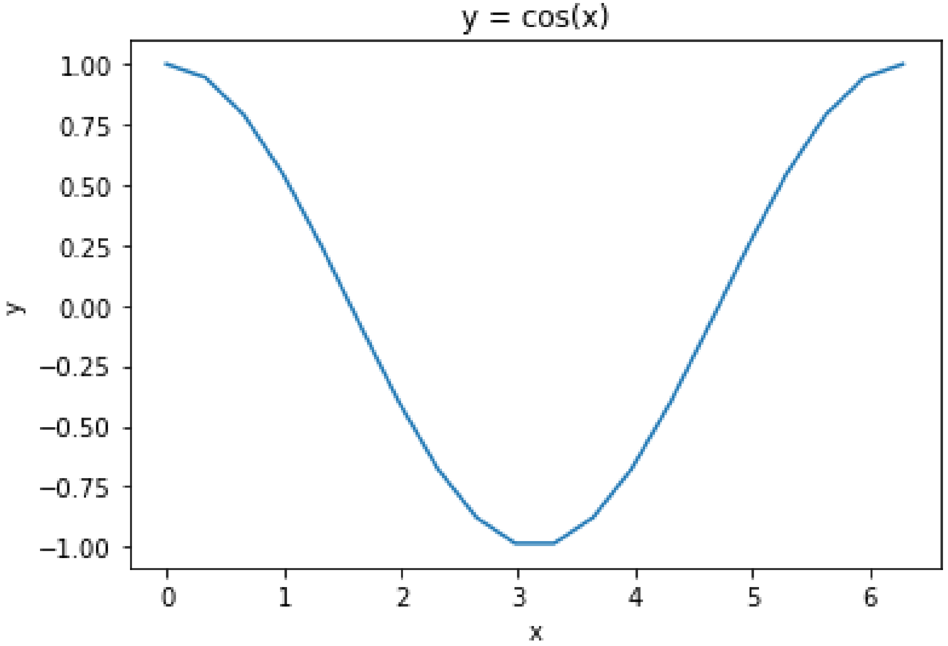
Example of plotting a cosine wave:

- Combination of libraries `pyplot` and `numpy`
- Create an array *x* of 20 equally-spaced numbers between 0 and 2π:

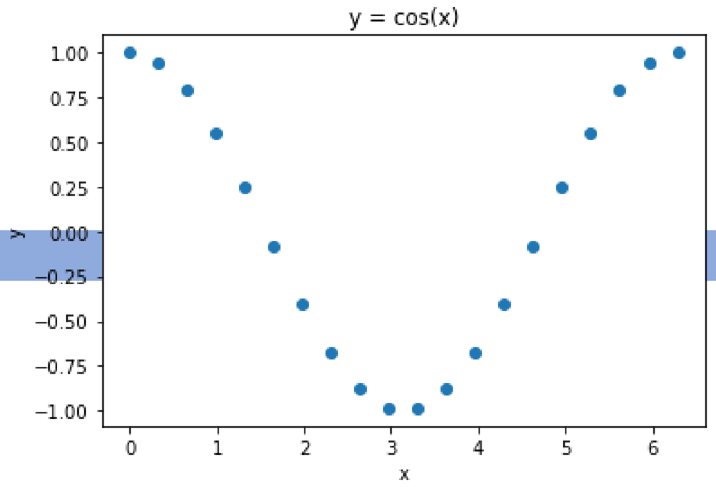
```
x = np.linspace(0, 2*np.pi, 20)
```

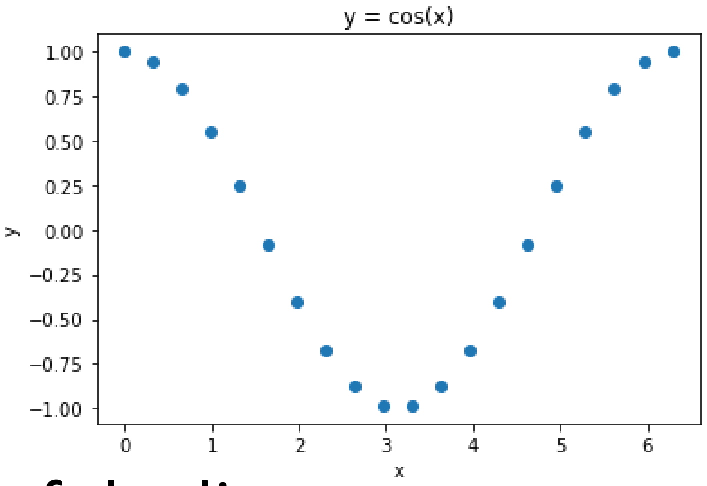
- Use function plot:

```
plt.plot(x, np.cos(x))
plt.xlabel('x')
plt.ylabel('y')
plt.title('y = cos(x)')
```



Plotting





- The previous code plotted a solid line.
  We can also only plot the points (x, cos(x)) with the code below:

```
plt.plot(x, np.cos(x), 'o')
plt.scatter(x,np.cos(x))
```

- By typing `help(plt.plot)`

you can obtain more information:

- how to change the colour or the linewidth of the lines
- how to prescribe the limits on the axes
- add a legend and title to the plot.

Contourplots

Often we want to plot **two-dimensional fields**

→ function `plt.contour()`

Contourplots

Often we want to plot **two-dimensional fields**

→ function `plt.contour()`

```
x = np.linspace(0, 10, 1000)
y = np.linspace(0, 10, 1000)
xx, yy = np.meshgrid(x,y)
```

Create an *x* and *y*-array, each has a length of 1000

Create a 2D grid from the arrays

Contourplots

Often we want to plot **two-dimensional fields**

→ function `plt.contour()`

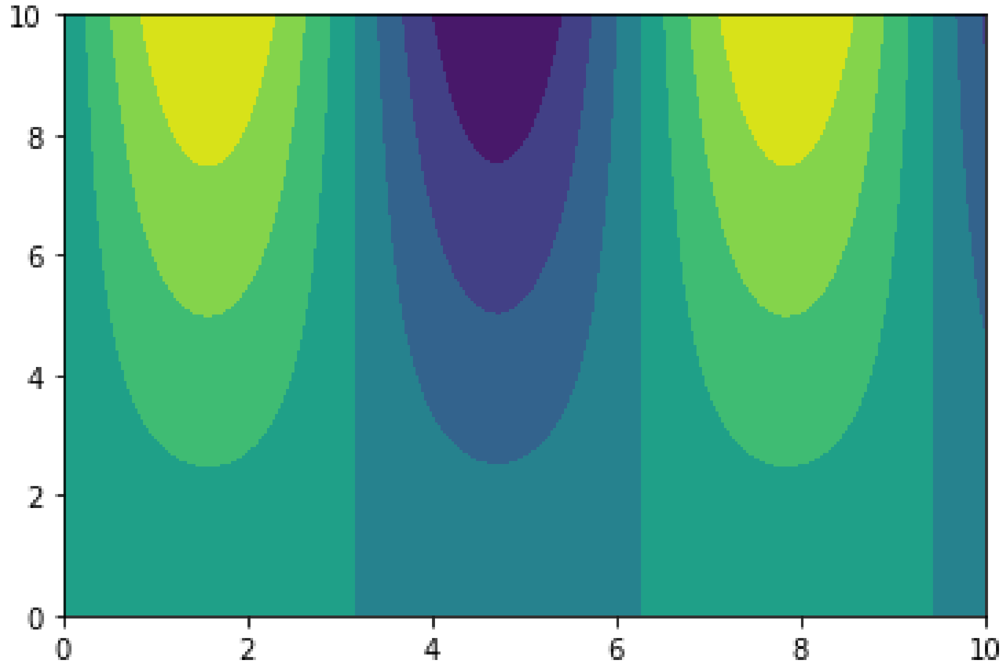
```
x = np.linspace(0, 10, 1000)
y = np.linspace(0, 10, 1000)
xx, yy = np.meshgrid(x,y)

z = np.sin(xx) * yy


plt.contourf(x,y,z)
```

Contourplots

Often we want to plot **two-dimensional fields**

→ function `plt.contour()`

```
x = np.linspace(0, 10, 1000)
y = np.linspace(0, 10, 1000)
xx, yy = np.meshgrid(x,y)

z = np.sin(xx) * yy


plt.contourf(x,y,z)
```



More plotting!

Interactive figures = figures than can be zoomed in and rotated

To achieve that, start you Notebook with:

```
%matplotlib notebook
```

If you do this, you have to tell Python each time a new figure starts, otherwise they will overlap.
So for each new figure, write:

```
plt.figure()
…
plt.show()
```

SciPy

SciPy = scientific computing package

SciPy

SciPy = scientific computing package

- Integrating
- Interpolating
- Curvefitting and optimizing
- Statistics
- Fourier transforms
- …

SciPy

SciPy = scientific computing package

- Integrating
- Interpolating
- Curvefitting and optimizing
- Statistics
- Fourier transforms
- …

Examples of interpolating and curvefitting in the (short) tutorial