ABILITÀ INFORMATICHE

Link moodle: https://moodle2.units.it/course/view.php?id=14606

Python functions

A **function** is a block of code which corresponds to a set of instructions and only runs when it is called.

Main **aim** of functions: split the script in logical blocks.

Syntax of functions:

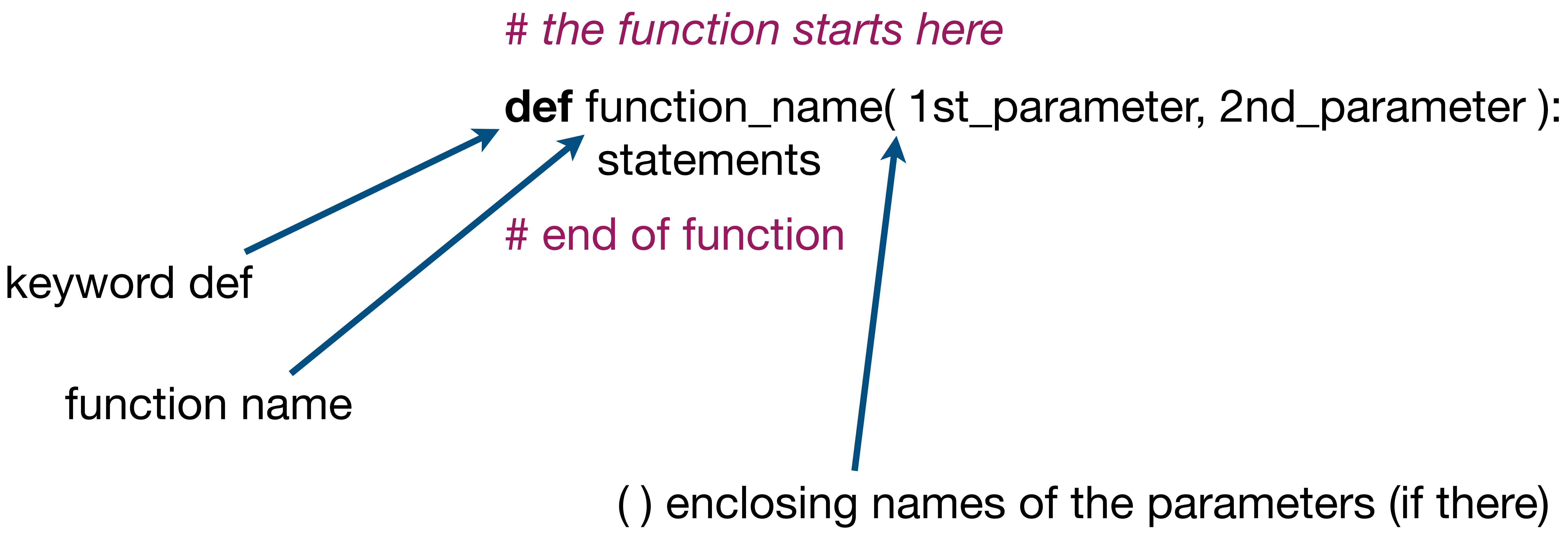
```
# the function starts here

def function_name( 1st_parameter, 2nd_parameter ):
    statements

# end of function
```

Python functions

A **function** is a block of code which corresponds to a set of instructions and only runs when it is called.

Main **aim** of functions: split the script in logical blocks.

Syntax of functions:



```
# the function starts here
def function_name( 1st_parameter, 2nd_parameter ):
    statements
# end of function
```

keyword def

function name

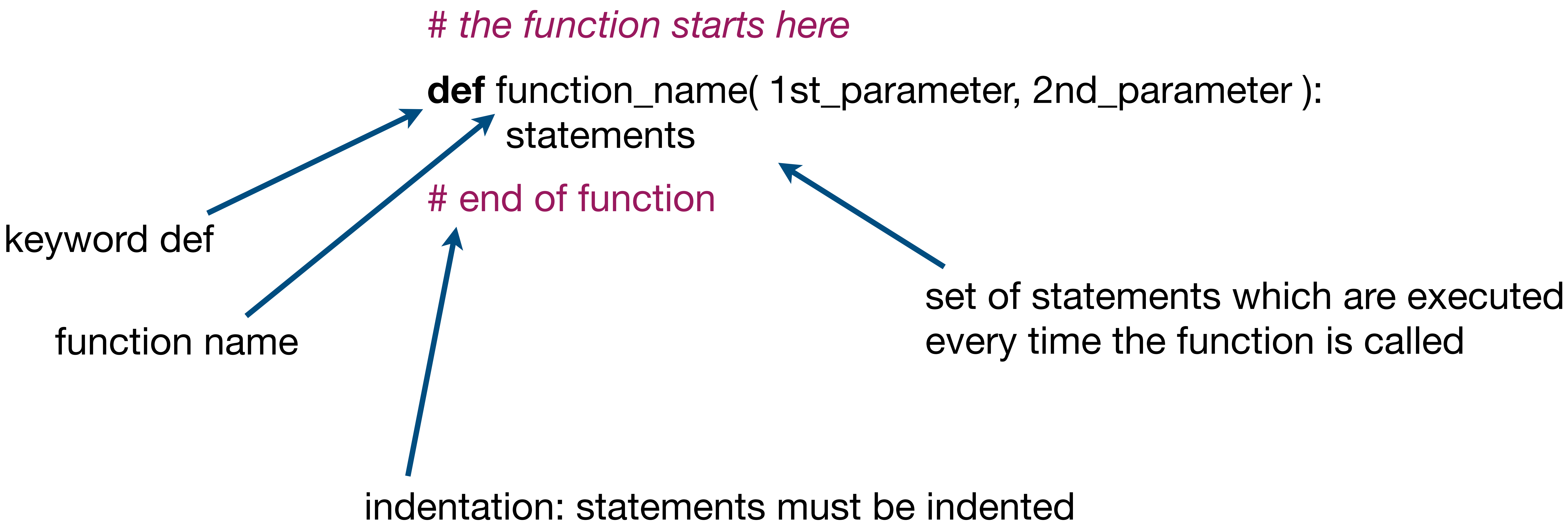
( ) enclosing names of the parameters (if there)

Python functions

A **function** is a block of code which corresponds to a set of instructions and only runs when it is called.

Main **aim** of functions: split the script in logical blocks.

Syntax of functions:



```
# the function starts here
def function_name( 1st_parameter, 2nd_parameter ):
    statements
# end of function
```

keyword def

function name

set of statements which are executed
every time the function is called

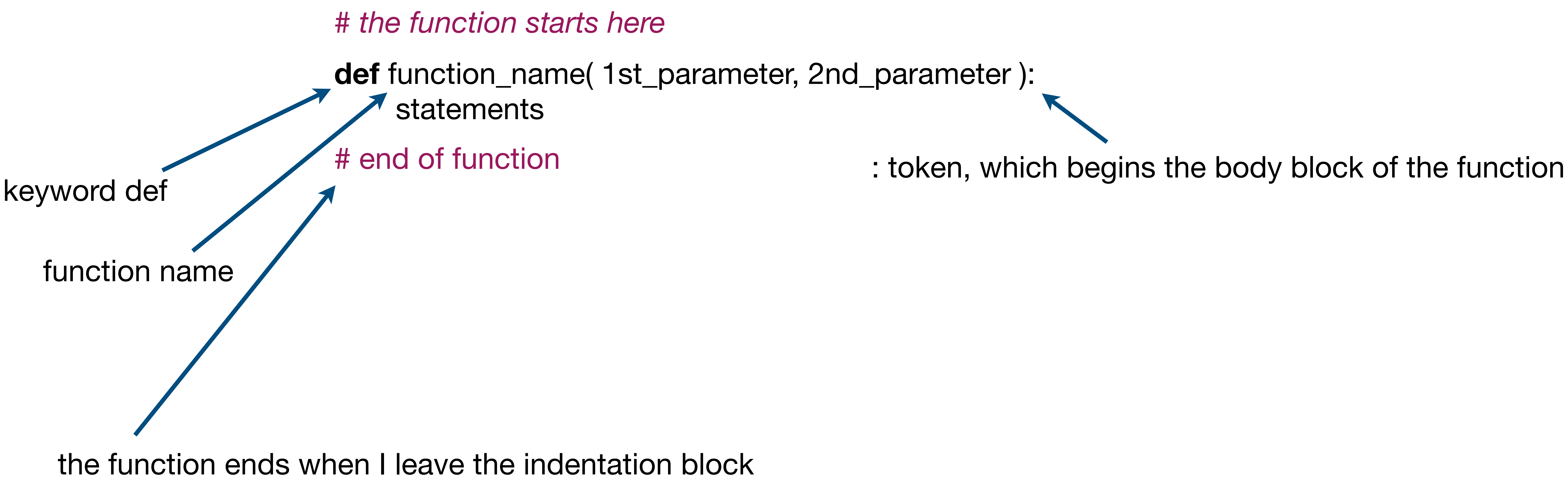
indentation: statements must be indented

Python functions

A **function** is a block of code which corresponds to a set of instructions and only runs when it is called.

Main **aim** of functions: split the script in logical blocks.

Syntax of functions:



```
# the function starts here
def function_name( 1st_parameter, 2nd_parameter ):
    statements
# end of function
```

keyword def

function name

: token, which begins the body block of the function

the function ends when I leave the indentation block

Python functions

A script can have as many functions as the user wants.

Functions can be called only after they’ve been defined (i.e., below).

Functions can call other functions.

Names of the function arguments are independent of those outside the function.

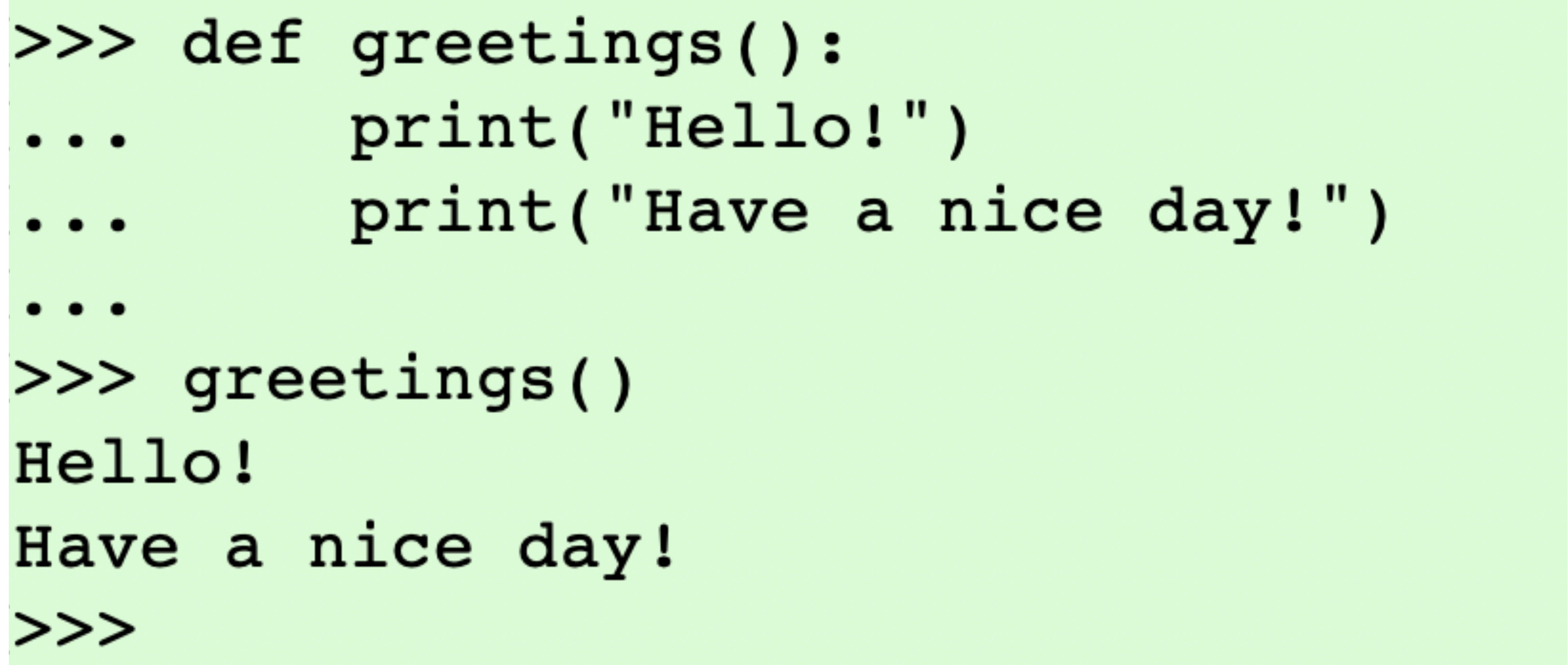
Functions help the script readability.

Python functions

Functions can be broadly split into two subgroups:

void functions and **functions returning values**

Examples of void functions:



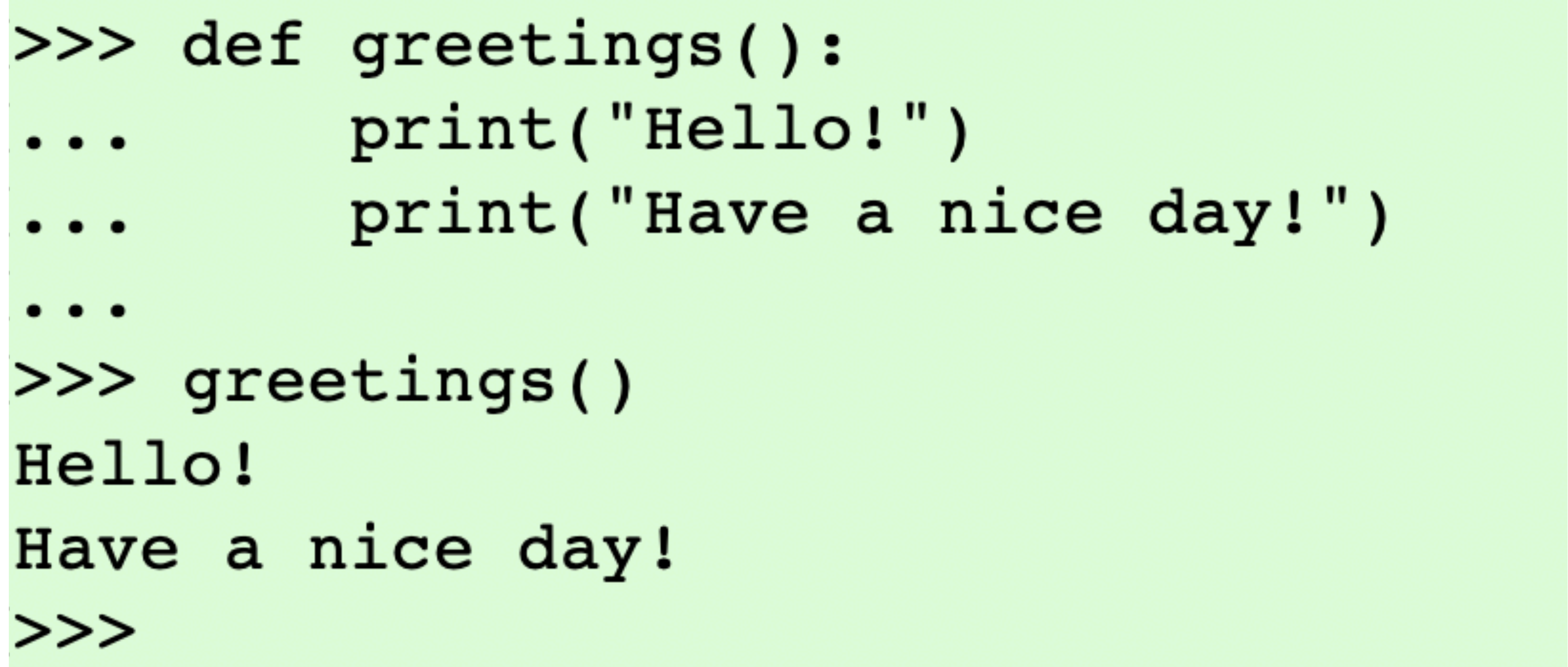
```
>>> def greetings():
...     print("Hello!")
...     print("Have a nice day!")
...
>>> greetings()
Hello!
Have a nice day!
>>>
```

Python functions

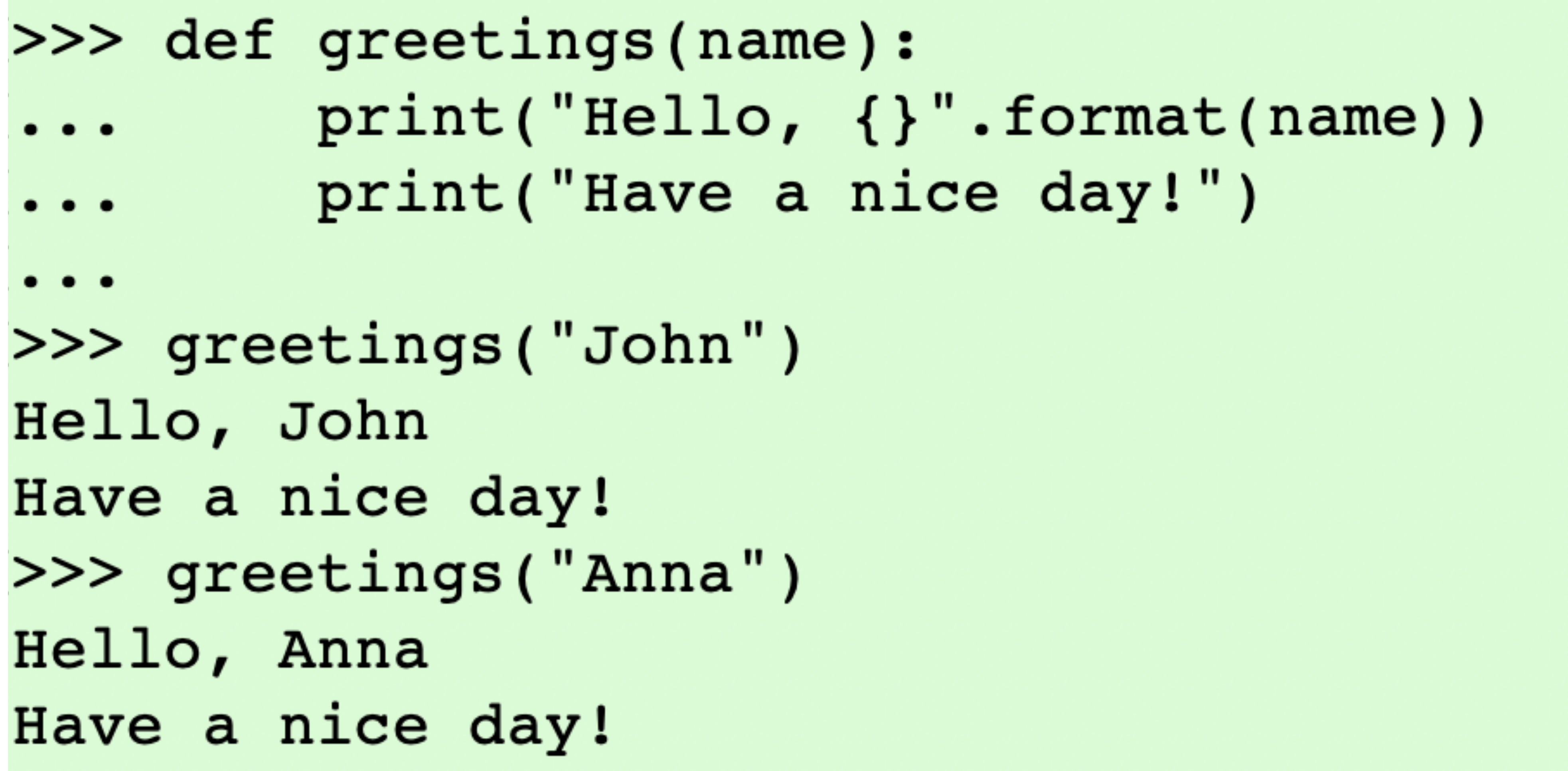
Functions can be broadly split into two subgroups:

void functions and **functions returning values**

Examples of void functions:



```
>>> def greetings():
...     print("Hello!")
...     print("Have a nice day!")
...
>>> greetings()
Hello!
Have a nice day!
>>>
```



```
>>> def greetings(name):
...     print("Hello, {}".format(name))
...     print("Have a nice day!")
...
>>> greetings("John")
Hello, John
Have a nice day!
>>> greetings("Anna")
Hello, Anna
Have a nice day!
```

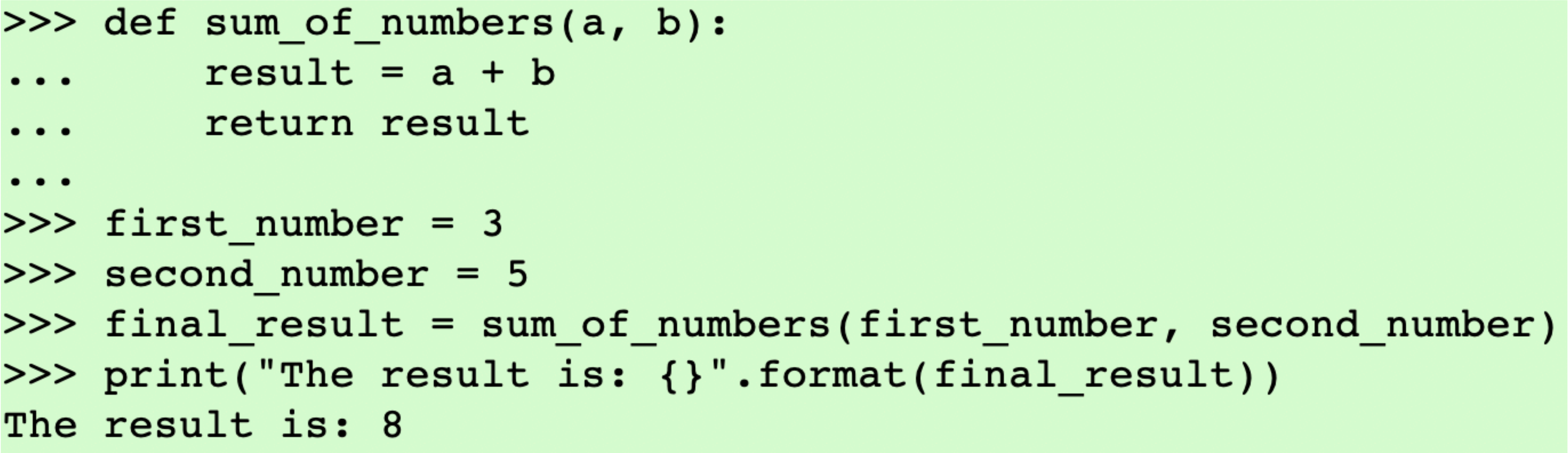
Void functions perform an action but do not return any computed value to the caller (they actually return None).

Python functions

Functions can be broadly split into two subgroups:

void functions and **functions returning values**

Example of a function returning a value:



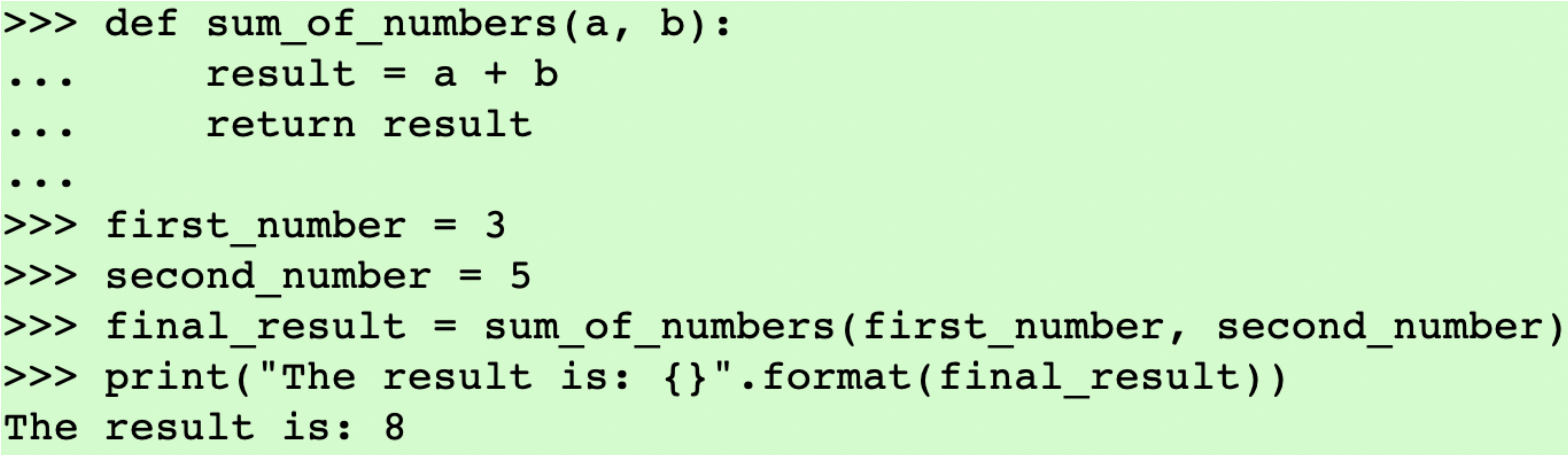
```
>>> def sum_of_numbers(a, b):
...     result = a + b
...     return result
...
>>> first_number = 3
>>> second_number = 5
>>> final_result = sum_of_numbers(first_number, second_number)
>>> print("The result is: {}".format(final_result))
The result is: 8
```

Python functions

Functions can be broadly split into two subgroups:

void functions and **functions returning values**

Example of a function returning a value:



```
>>> def sum_of_numbers(a, b):
...     result = a + b
...     return result
...
>>> first_number = 3
>>> second_number = 5
>>> final_result = sum_of_numbers(first_number, second_number)
>>> print("The result is: {}".format(final_result))
The result is: 8
```

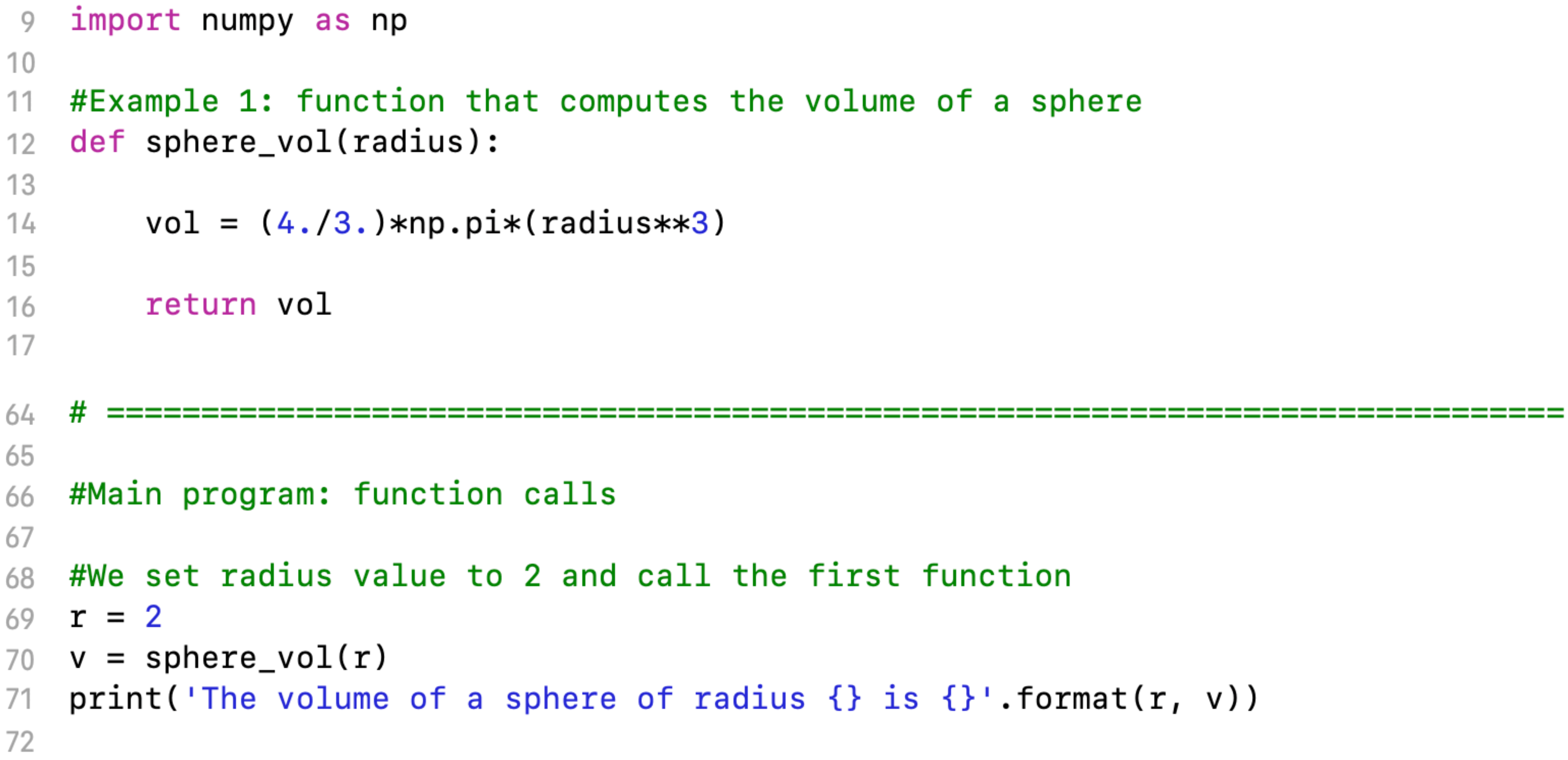
function header
temporary variable
return temporary variable

Variables created within functions only exist within them

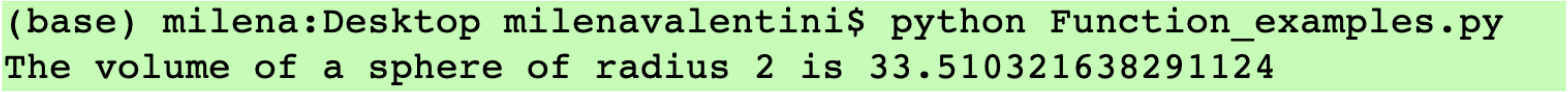
Function arguments are **local variables**, too

Python functions

Some examples.

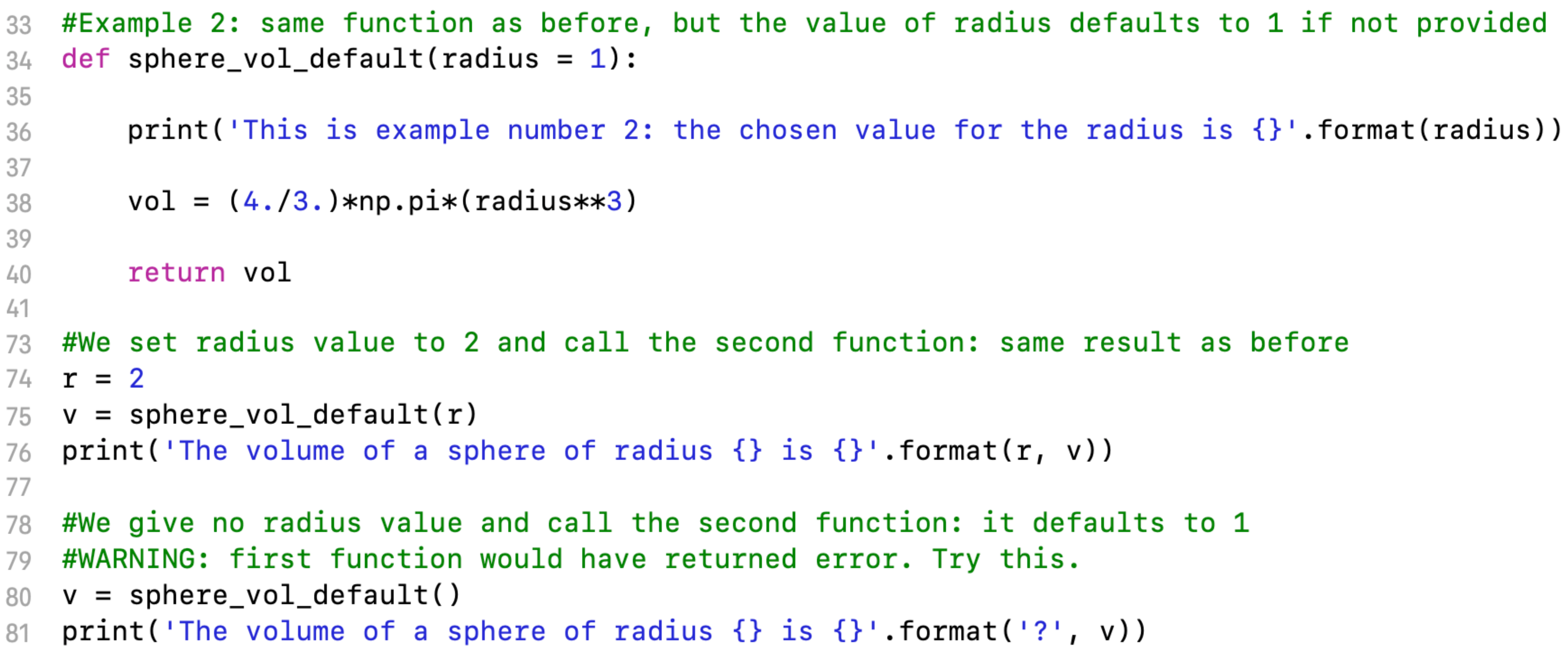


```python
import numpy as np

#Example 1: function that computes the volume of a sphere
def sphere_vol(radius):

    vol = (4./3.)*np.pi*(radius**3)

    return vol


# ==========================================================================

#Main program: function calls

#We set radius value to 2 and call the first function
r = 2
v = sphere_vol(r)
print('The volume of a sphere of radius {} is {}'.format(r, v))

```

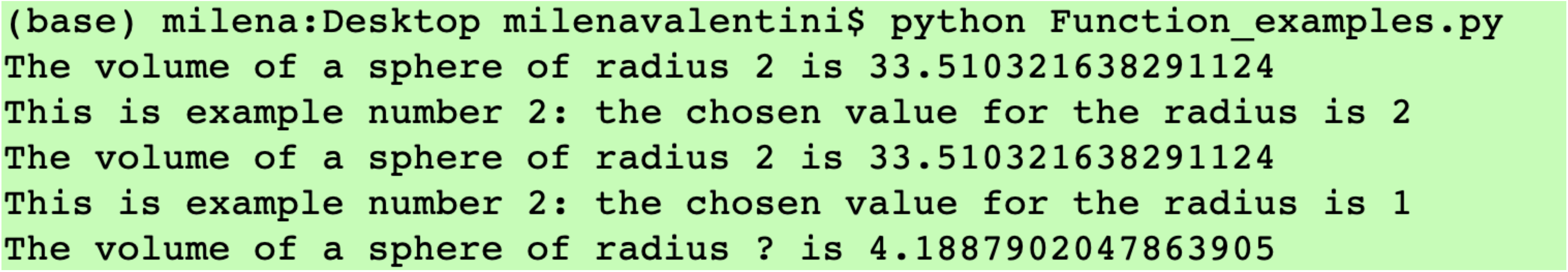


```
(base) milena:Desktop milenavalentini$ python Function_examples.py
The volume of a sphere of radius 2 is 33.510321638291124
```

Python functions

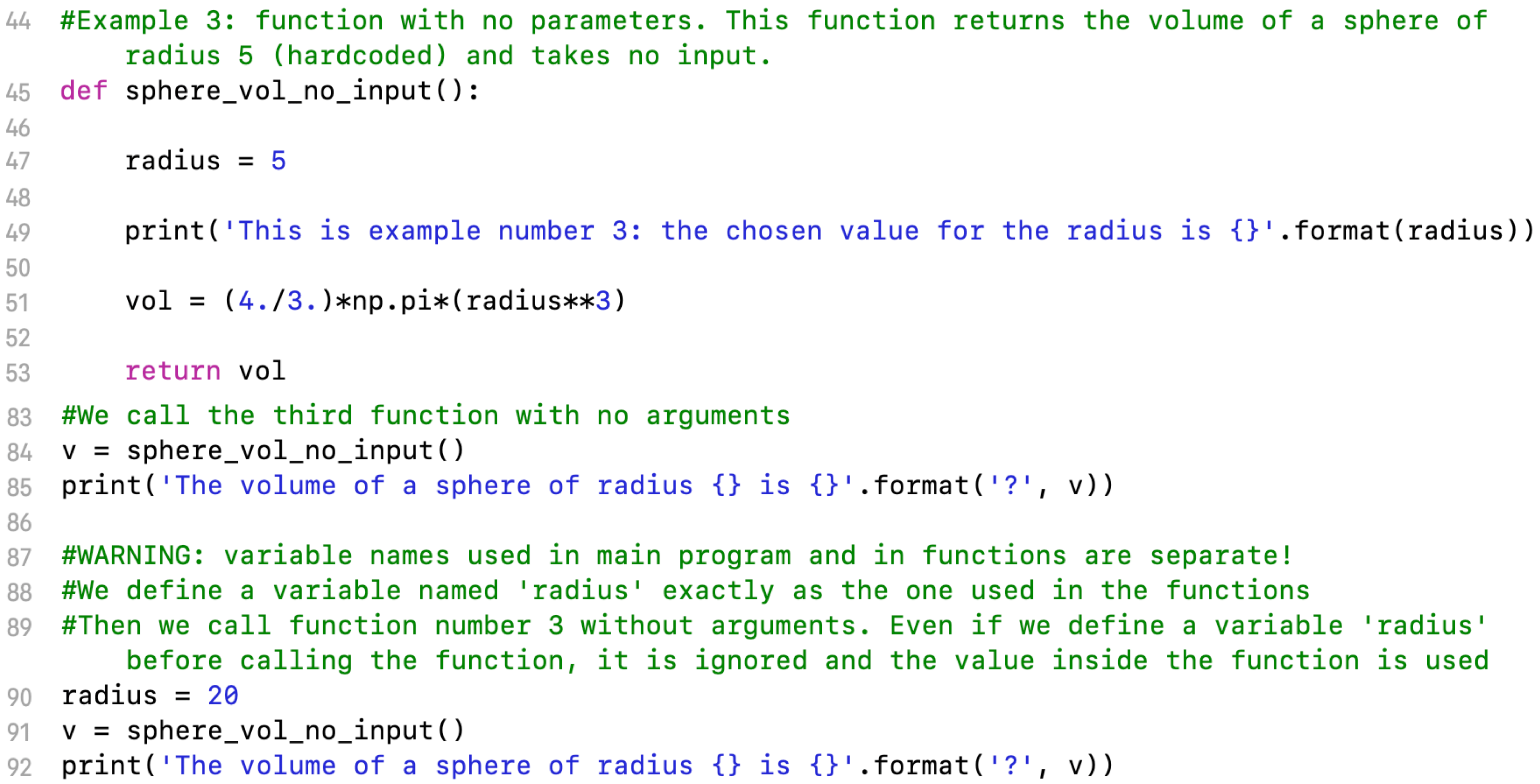


```python
#Example 2: same function as before, but the value of radius defaults to 1 if not provided
def sphere_vol_default(radius = 1):

    print('This is example number 2: the chosen value for the radius is {}'.format(radius))

    vol = (4./3.)*np.pi*(radius**3)

    return vol

#We set radius value to 2 and call the second function: same result as before
r = 2
v = sphere_vol_default(r)
print('The volume of a sphere of radius {} is {}'.format(r, v))

#We give no radius value and call the second function: it defaults to 1
#WARNING: first function would have returned error. Try this.
v = sphere_vol_default()
print('The volume of a sphere of radius {} is {}'.format('?', v))
```

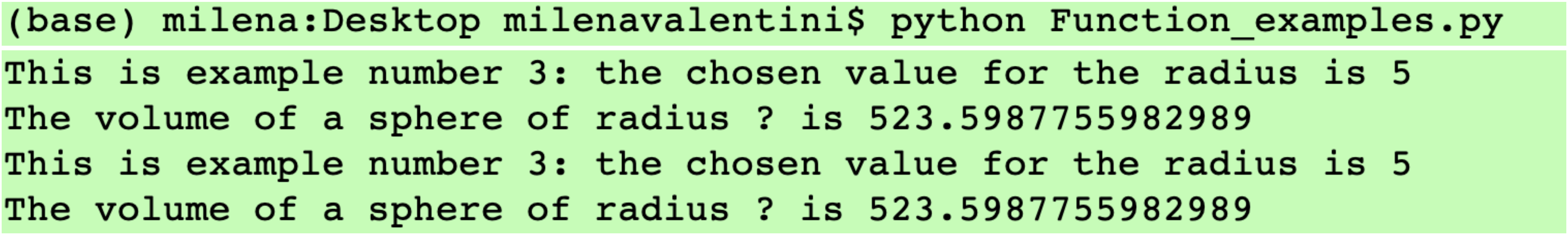


```
(base) milena:Desktop milenavalentini$ python Function_examples.py
The volume of a sphere of radius 2 is 33.510321638291124
This is example number 2: the chosen value for the radius is 2
The volume of a sphere of radius 2 is 33.510321638291124
This is example number 2: the chosen value for the radius is 1
The volume of a sphere of radius ? is 4.1887902047863905
```

Python functions

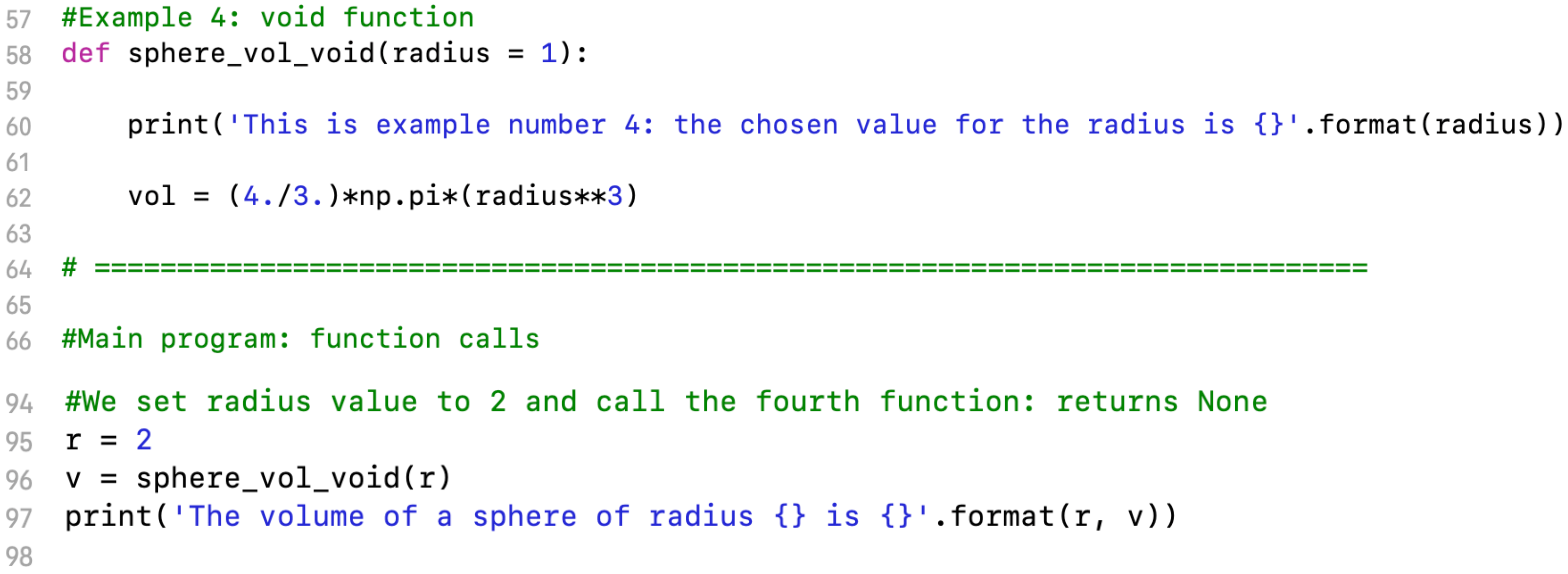


```python
#Example 3: function with no parameters. This function returns the volume of a sphere of radius 5 (hardcoded) and takes no input.
def sphere_vol_no_input():

    radius = 5

    print('This is example number 3: the chosen value for the radius is {}'.format(radius))

    vol = (4./3.)*np.pi*(radius**3)

    return vol
#We call the third function with no arguments
v = sphere_vol_no_input()
print('The volume of a sphere of radius {} is {}'.format('?', v))

#WARNING: variable names used in main program and in functions are separate!
#We define a variable named 'radius' exactly as the one used in the functions
#Then we call function number 3 without arguments. Even if we define a variable 'radius' before calling the function, it is ignored and the value inside the function is used
radius = 20
v = sphere_vol_no_input()
print('The volume of a sphere of radius {} is {}'.format('?', v))
```

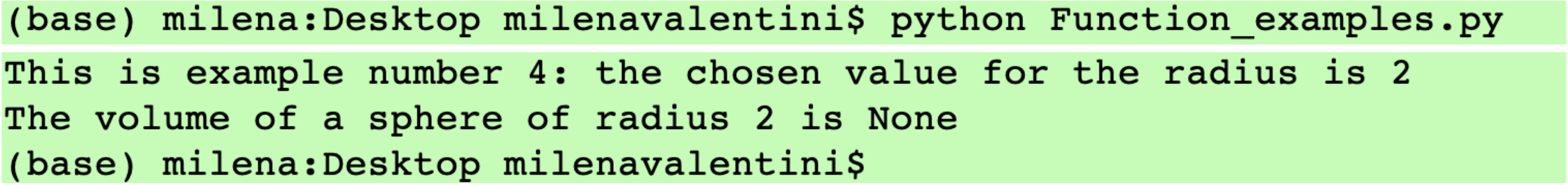


```
(base) milena:Desktop milenavalentini$ python Function_examples.py
This is example number 3: the chosen value for the radius is 5
The volume of a sphere of radius ? is 523.5987755982989
This is example number 3: the chosen value for the radius is 5
The volume of a sphere of radius ? is 523.5987755982989
```

Python functions



```python
#Example 4: void function
def sphere_vol_void(radius = 1):

    print('This is example number 4: the chosen value for the radius is {}'.format(radius))

    vol = (4./3.)*np.pi*(radius**3)

# ============================================================================

#Main program: function calls

#We set radius value to 2 and call the fourth function: returns None
r = 2
v = sphere_vol_void(r)
print('The volume of a sphere of radius {} is {}'.format(r, v))

```



```
(base) milena:Desktop milenavalentini$ python Function_examples.py
This is example number 4: the chosen value for the radius is 2
The volume of a sphere of radius 2 is None
(base) milena:Desktop milenavalentini$
```

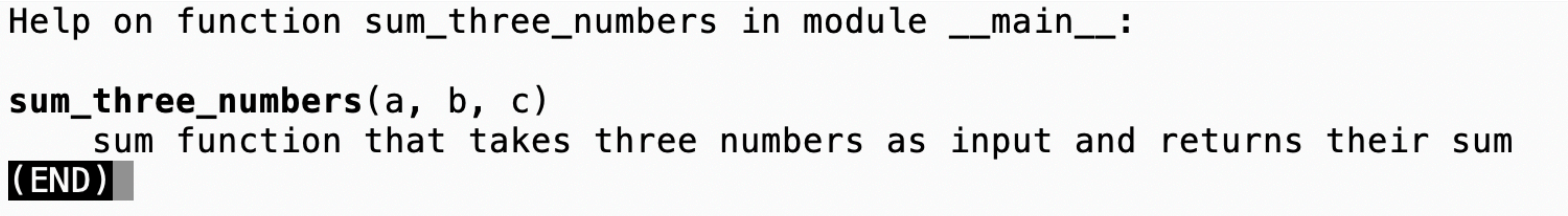
Python functions

The “““ Test here to document ””” string within a function is called **docstring**.

Placed at the very top of the function body, it acts as a documentation on the function.

This string gets printed out when you call help( ) on the function.

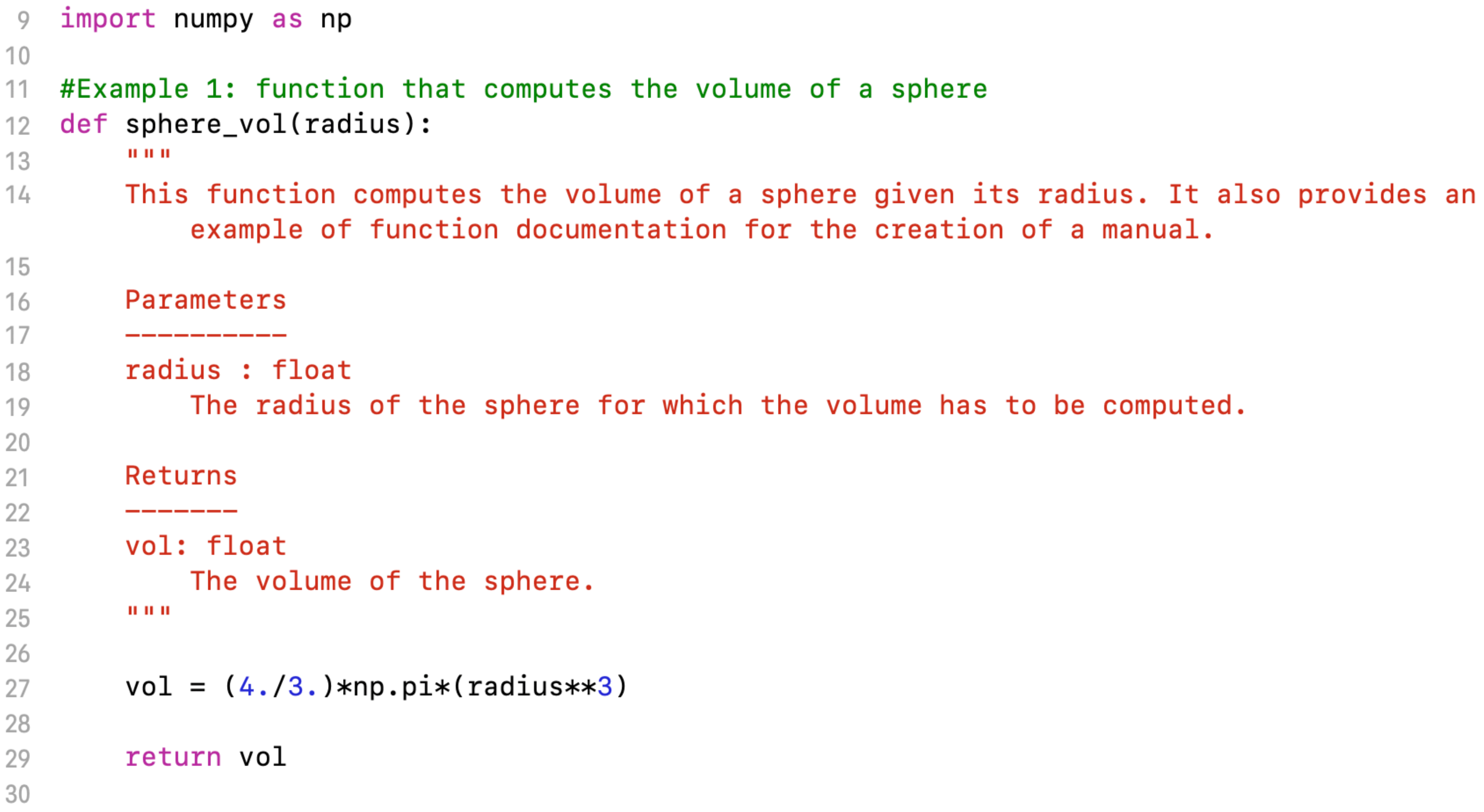
```
>>> def sum_three_numbers(a, b, c):
...     """sum function that takes three numbers as input and returns their sum"""
...     result = a + b + c
...     return result
...
>>> help(sum_three_numbers)
```



```
Help on function sum_three_numbers in module __main__:

sum_three_numbers(a, b, c)
    sum function that takes three numbers as input and returns their sum
(END)
```

Python functions

Example.



```python
import numpy as np

#Example 1: function that computes the volume of a sphere
def sphere_vol(radius):
    """
    This function computes the volume of a sphere given its radius. It also provides an example of function documentation for the creation of a manual.

    Parameters
    ----------
    radius : float
        The radius of the sphere for which the volume has to be computed.

    Returns
    -------
    vol: float
        The volume of the sphere.
    """

    vol = (4./3.)*np.pi*(radius**3)

    return vol

```

Python functions

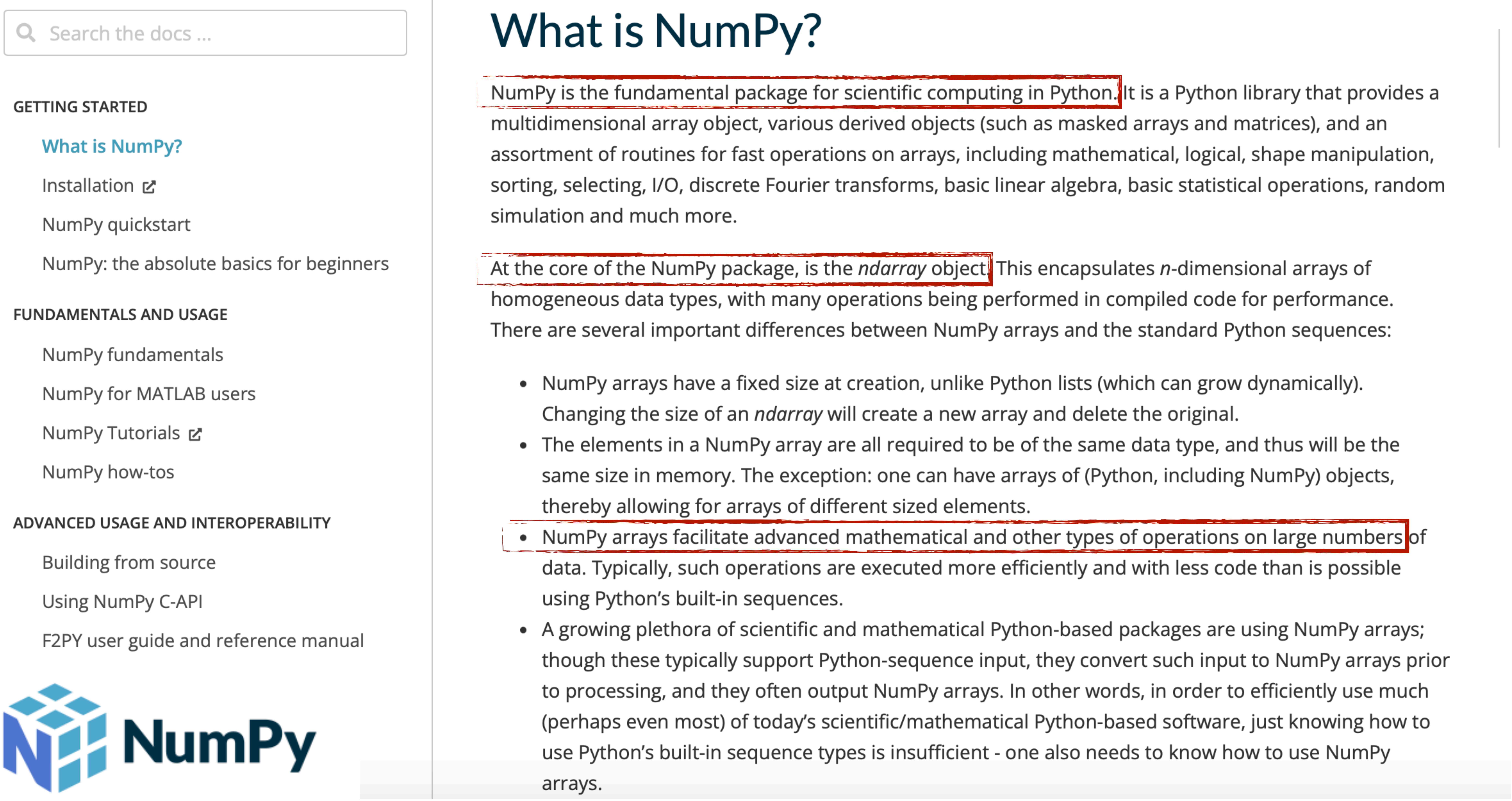
Exercises.

1. Write a function that computes the volume of a cylinder given radius and height. Make it so that radius and height default to 1 if they are not given.

2. Write another function that prints a sentence (‘Hello World’) and takes no input. Make it a void function.

3. Write one function that for each radius of a list of at least four radii computes diameter, circumference and area (via a single call to function).
   Then, use the function and print a sentence like
   “Radius xx has: diameter yy, circumference zz, and area ww”.

Python: numpy



What is NumPy?

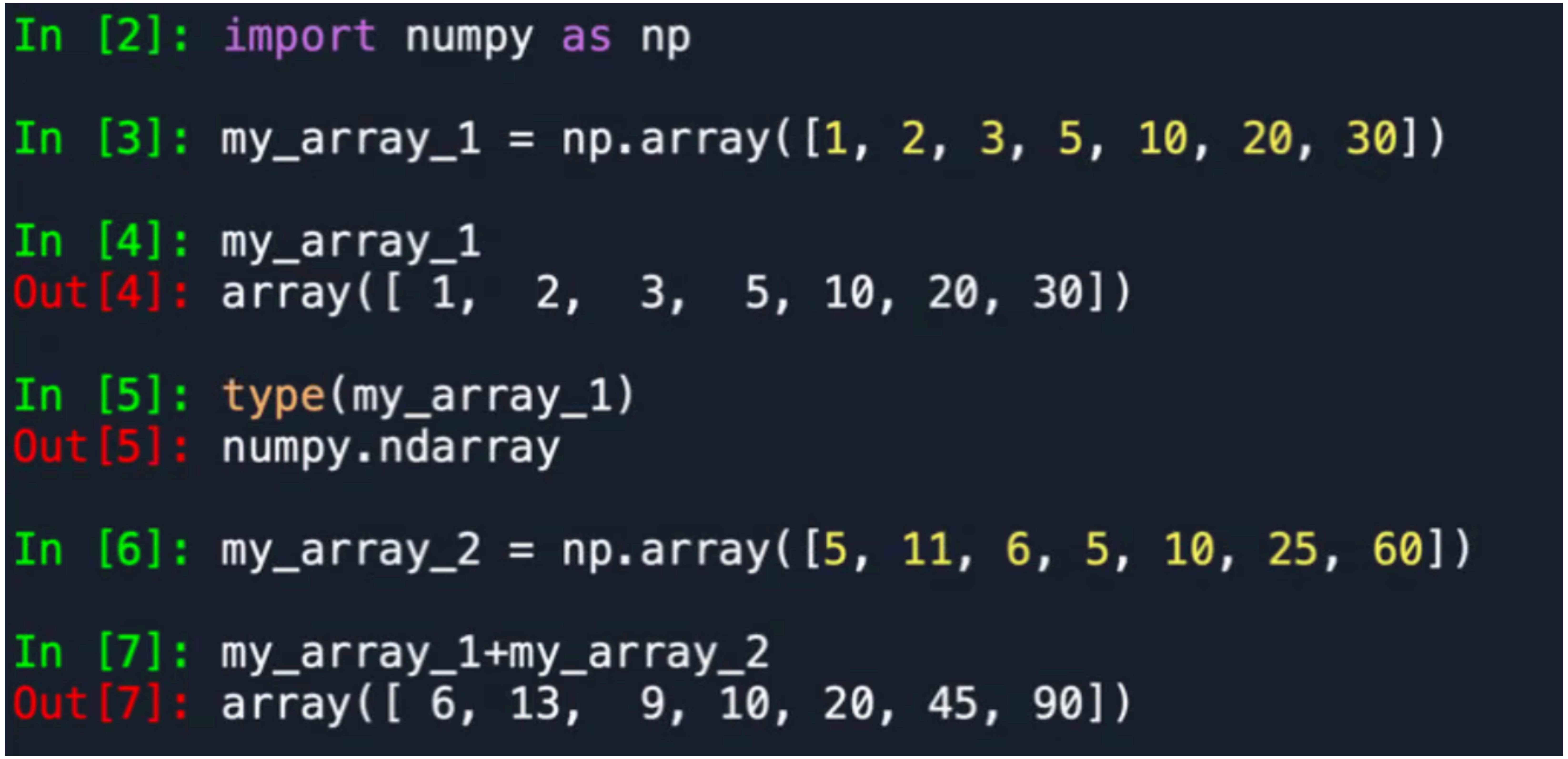
NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

Python: numpy

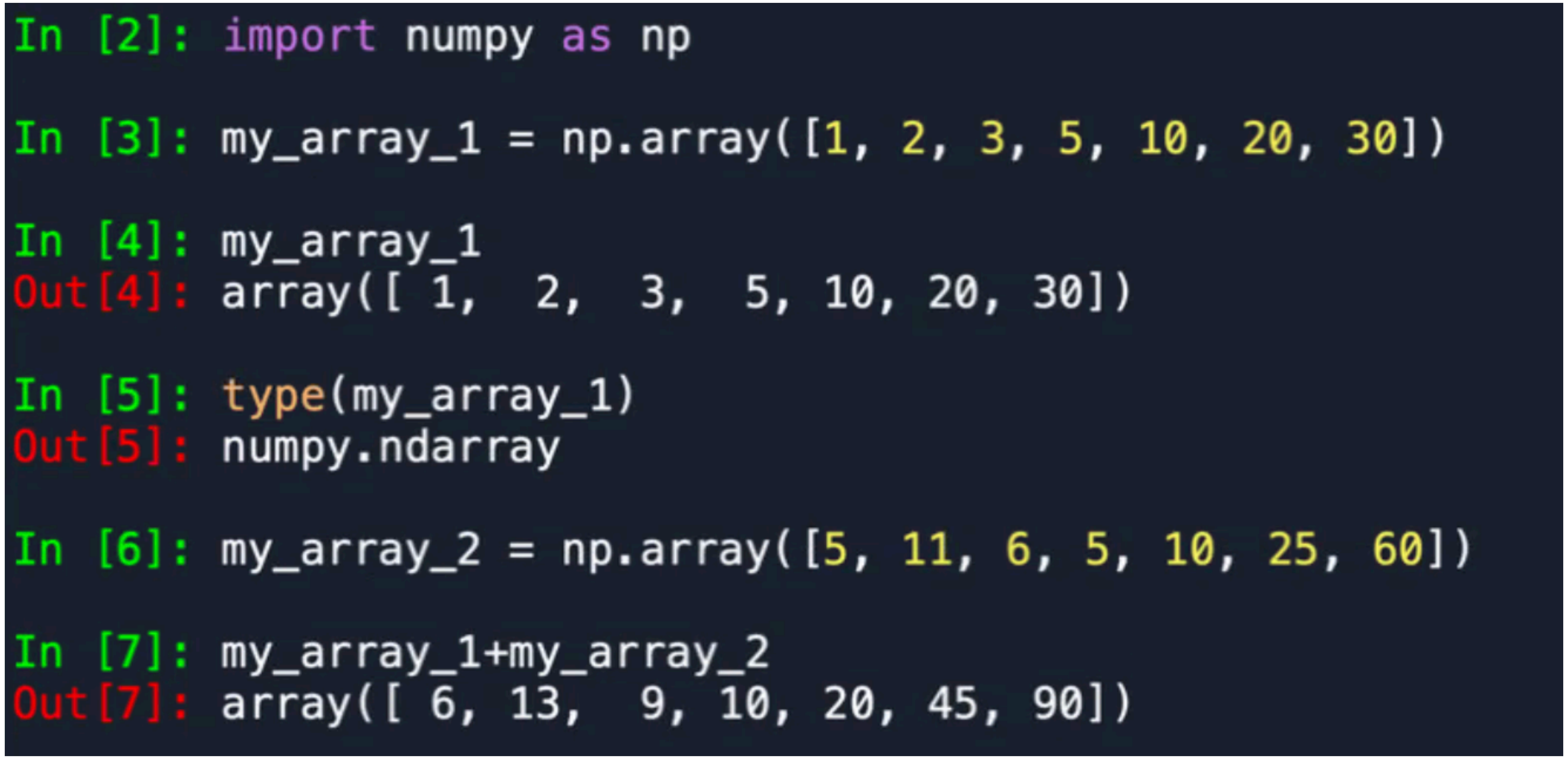
The N-dim array (ndarray) object and operations with arrays



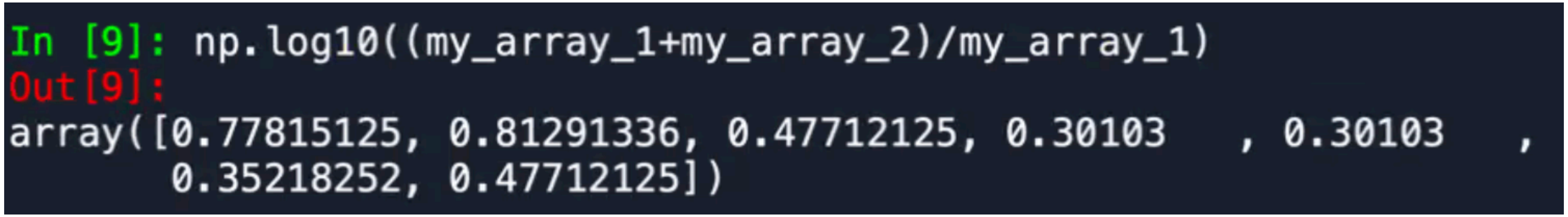
```
In [2]: import numpy as np

In [3]: my_array_1 = np.array([1, 2, 3, 5, 10, 20, 30])

In [4]: my_array_1
Out[4]: array([ 1,  2,  3,  5, 10, 20, 30])

In [5]: type(my_array_1)
Out[5]: numpy.ndarray

In [6]: my_array_2 = np.array([5, 11, 6, 5, 10, 25, 60])

In [7]: my_array_1+my_array_2
Out[7]: array([ 6, 13,  9, 10, 20, 45, 90])
```

Python: numpy

The N-dim array (ndarray) object and operations with arrays



```
In [2]: import numpy as np

In [3]: my_array_1 = np.array([1, 2, 3, 5, 10, 20, 30])

In [4]: my_array_1
Out[4]: array([ 1,  2,  3,  5, 10, 20, 30])

In [5]: type(my_array_1)
Out[5]: numpy.ndarray

In [6]: my_array_2 = np.array([5, 11, 6, 5, 10, 25, 60])

In [7]: my_array_1+my_array_2
Out[7]: array([ 6, 13,  9, 10, 20, 45, 90])
```

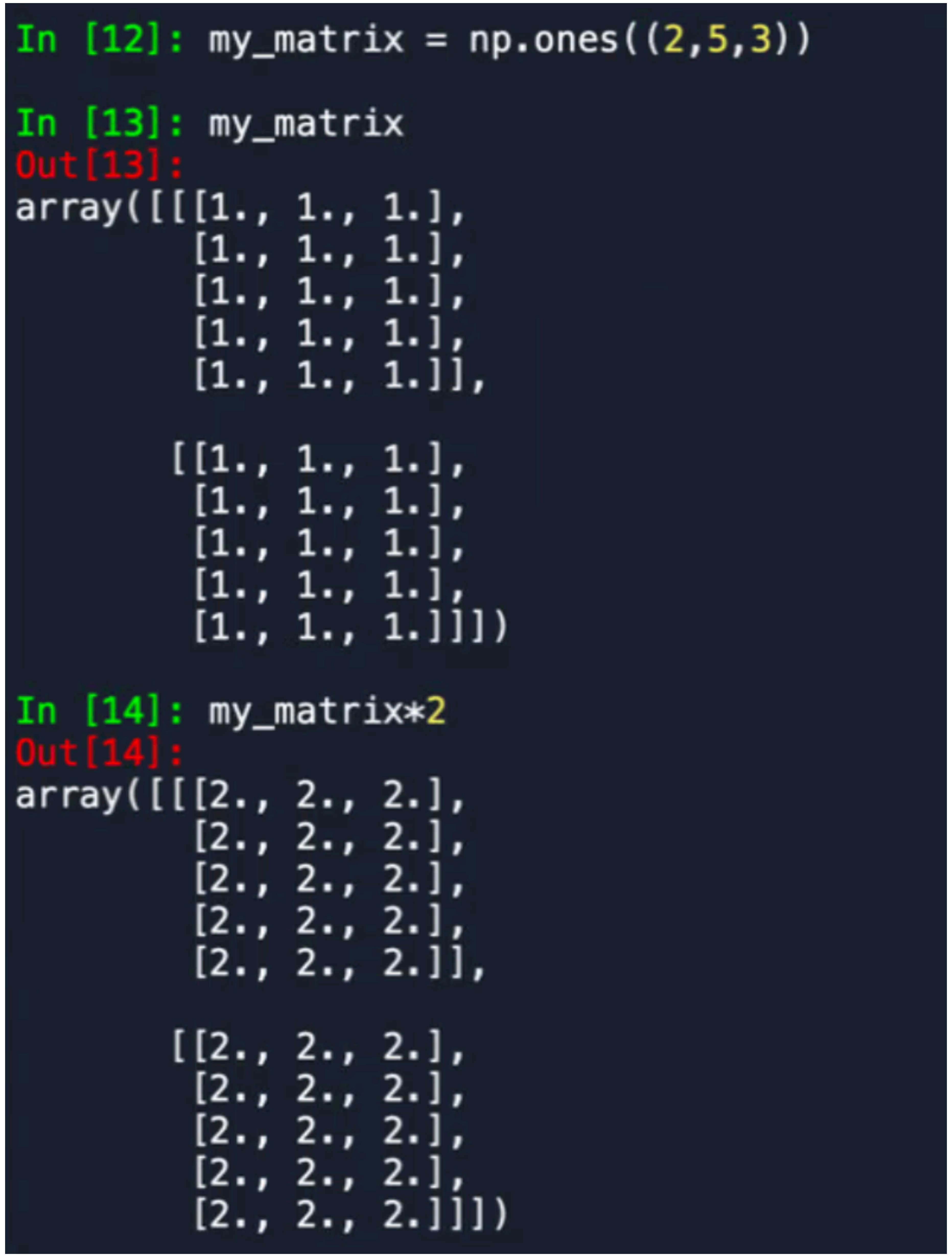


```
In [9]: np.log10((my_array_1+my_array_2)/my_array_1)
Out[9]:
array([0.77815125, 0.81291336, 0.47712125, 0.30103   , 0.30103   ,
       0.35218252, 0.47712125])
```

Python: numpy

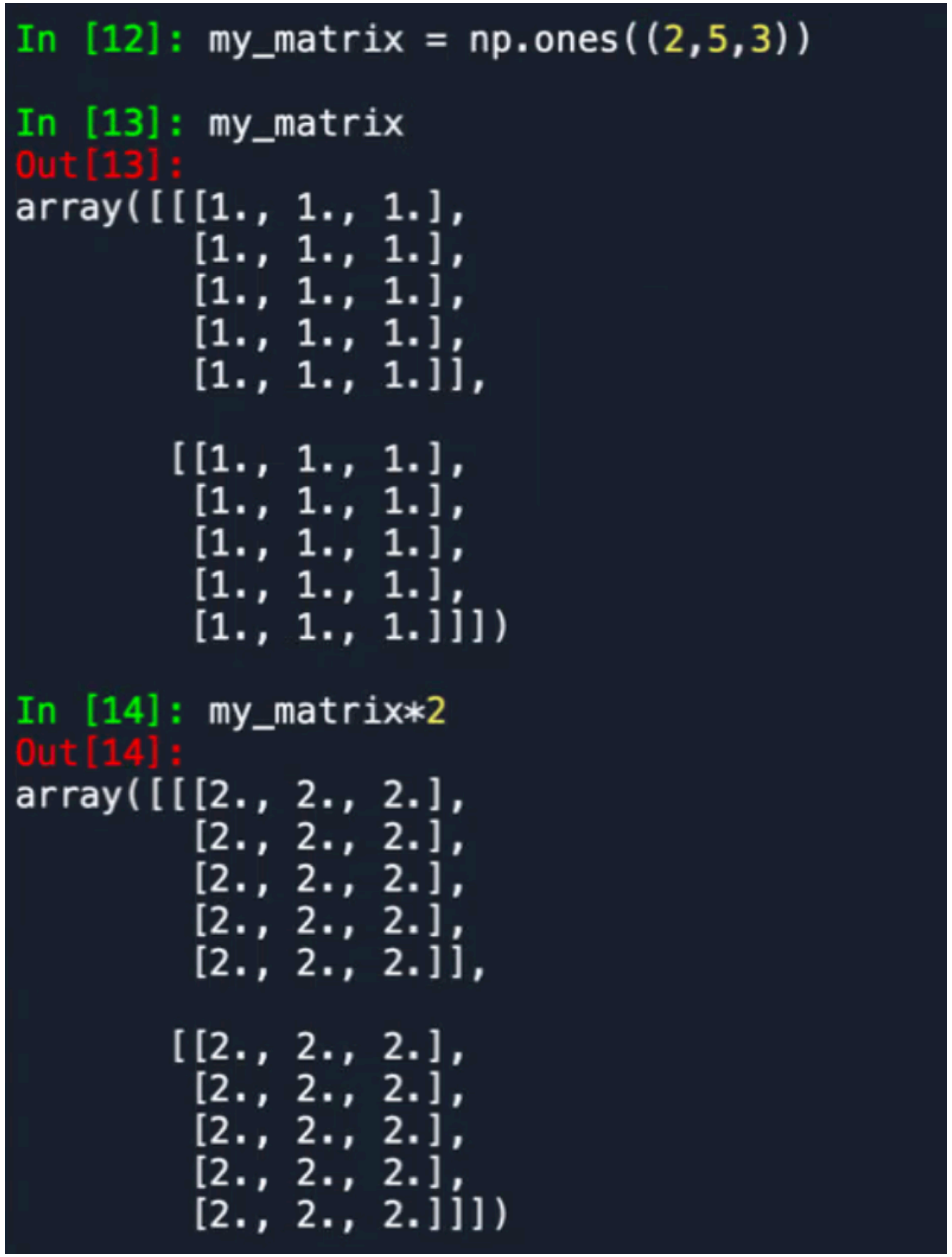
How to create an N-dim matrix

numpy.ones creates an array and fills it with 1.



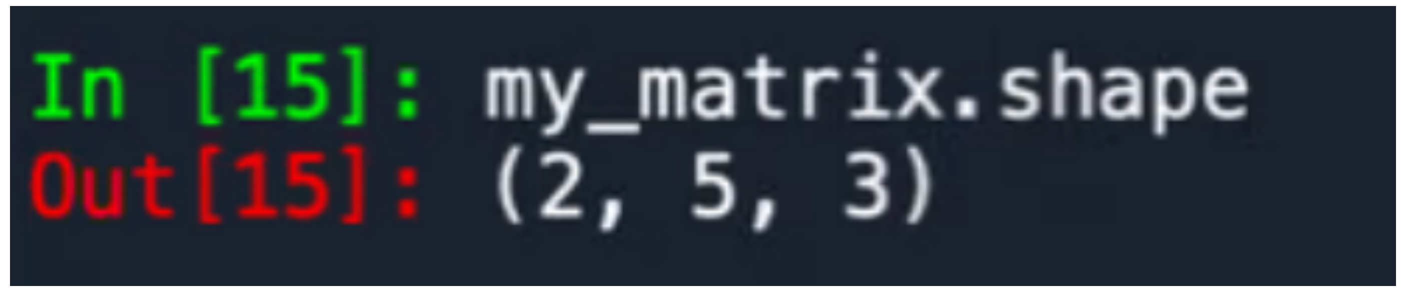
```
In [12]: my_matrix = np.ones((2,5,3))

In [13]: my_matrix
Out[13]:
array([[[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]])

In [14]: my_matrix*2
Out[14]:
array([[[2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.]],

       [[2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.]]])
```

Python: numpy



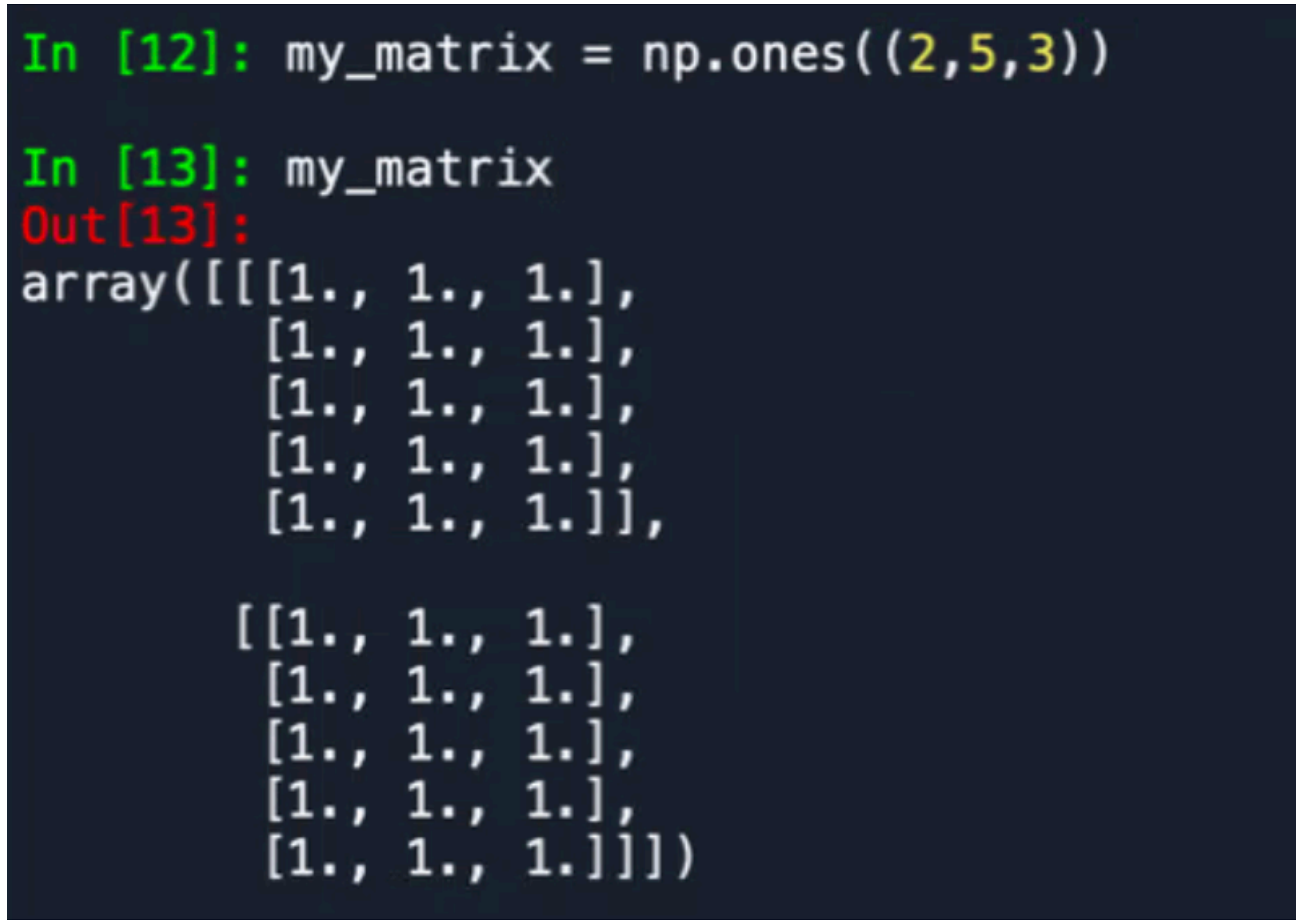
```
In [12]: my_matrix = np.ones((2,5,3))

In [13]: my_matrix
Out[13]:
array([[[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]])

In [14]: my_matrix*2
Out[14]:
array([[[2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.]],

       [[2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.],
        [2., 2., 2.]]])
```

What are the dimensions of the matrix?

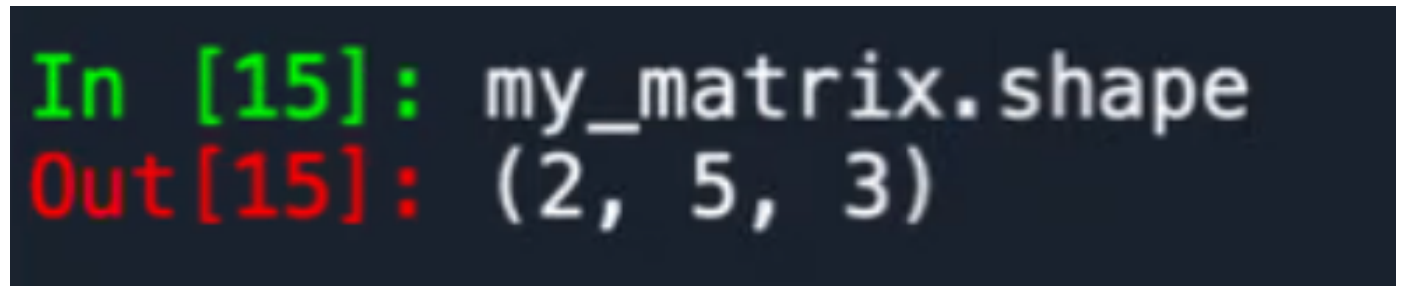


```
In [15]: my_matrix.shape
Out[15]: (2, 5, 3)
```

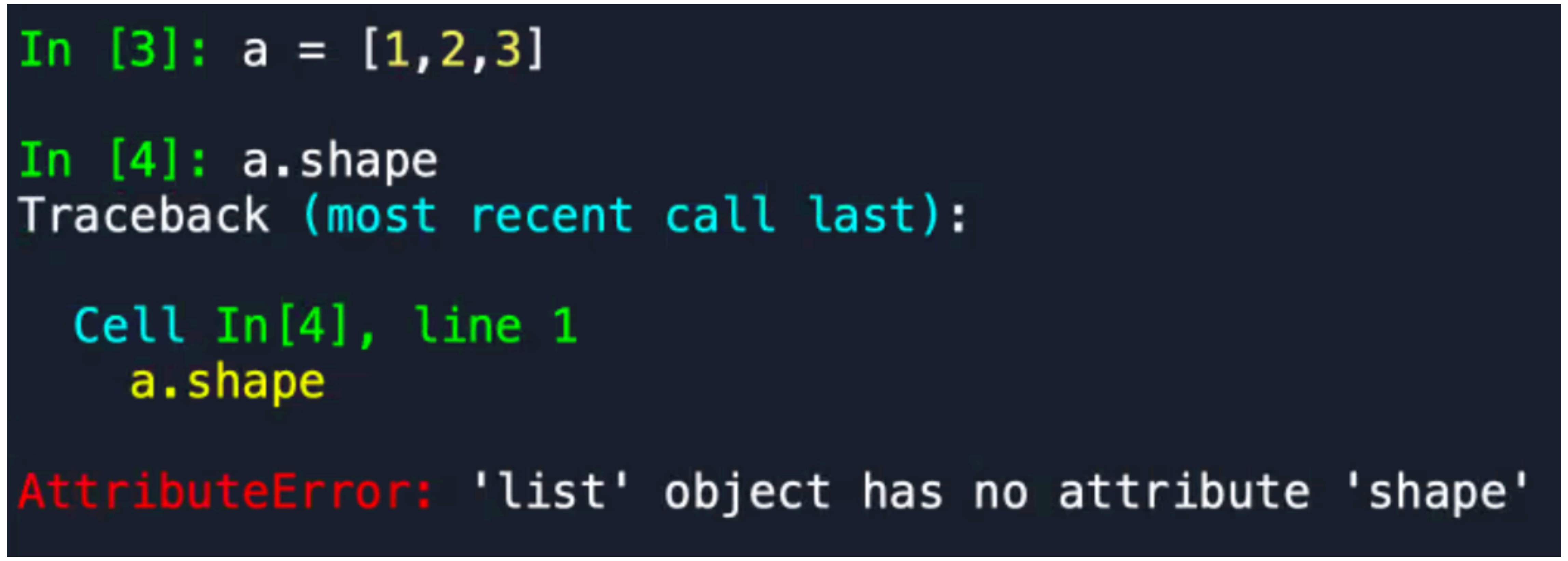
Python: numpy



```
In [12]: my_matrix = np.ones((2,5,3))

In [13]: my_matrix
Out[13]:
array([[[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]],

       [[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]])
```



```
In [15]: my_matrix.shape
Out[15]: (2, 5, 3)
```

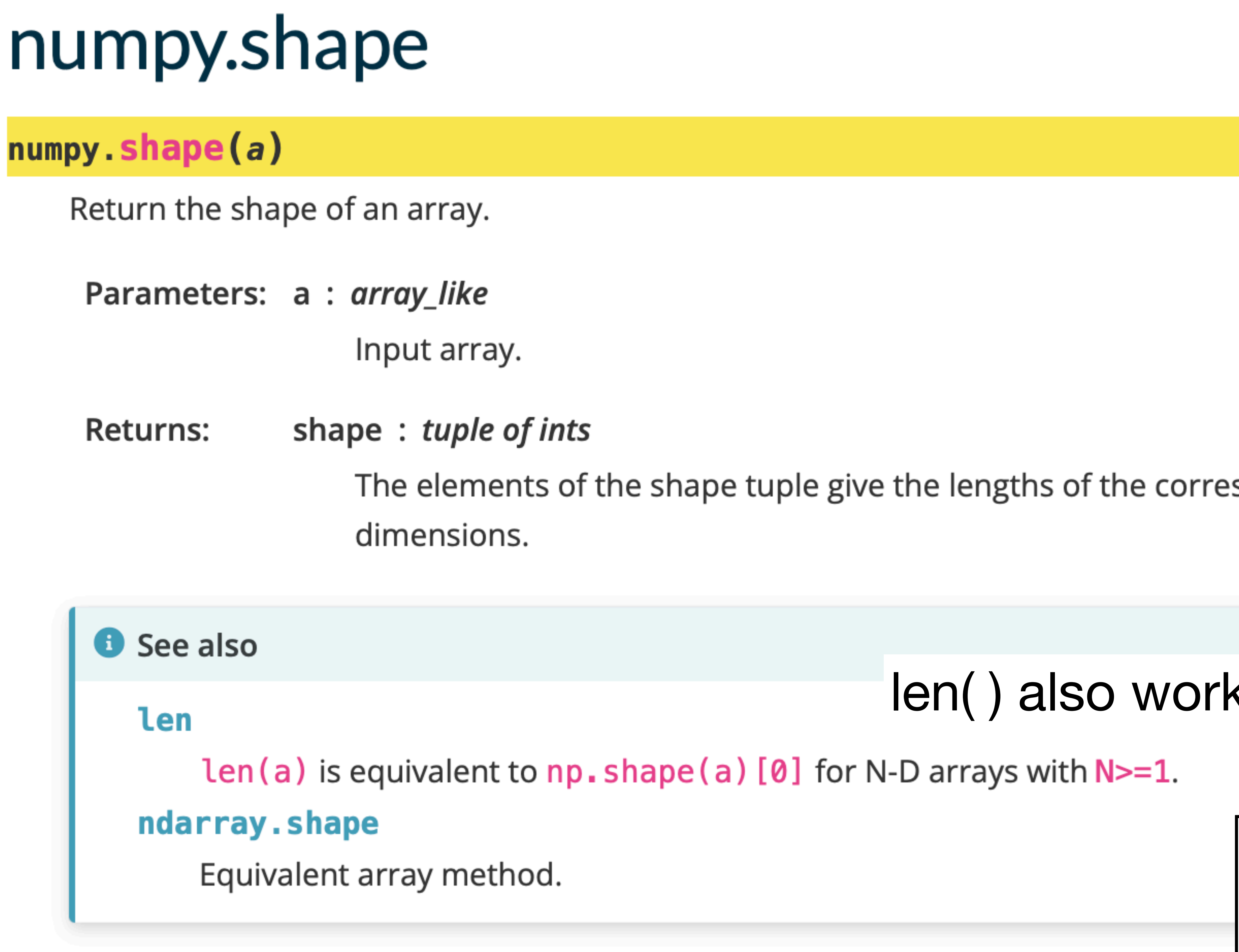


```
In [3]: a = [1,2,3]

In [4]: a.shape
Traceback (most recent call last):

  Cell In[4], line 1
    a.shape

AttributeError: 'list' object has no attribute 'shape'
```

The shape attribute only works with arrays

An attribute is a ~feature of the data structure
that you can access with the .
(if the method is present for that data structure)

Python: numpy



numpy.shape

`numpy.shape(a)` [source]

Return the shape of an array.

Parameters: **a** : *array_like*

Input array.

Returns: **shape** : *tuple of ints*

The elements of the shape tuple give the lengths of the corresponding array dimensions.

See also

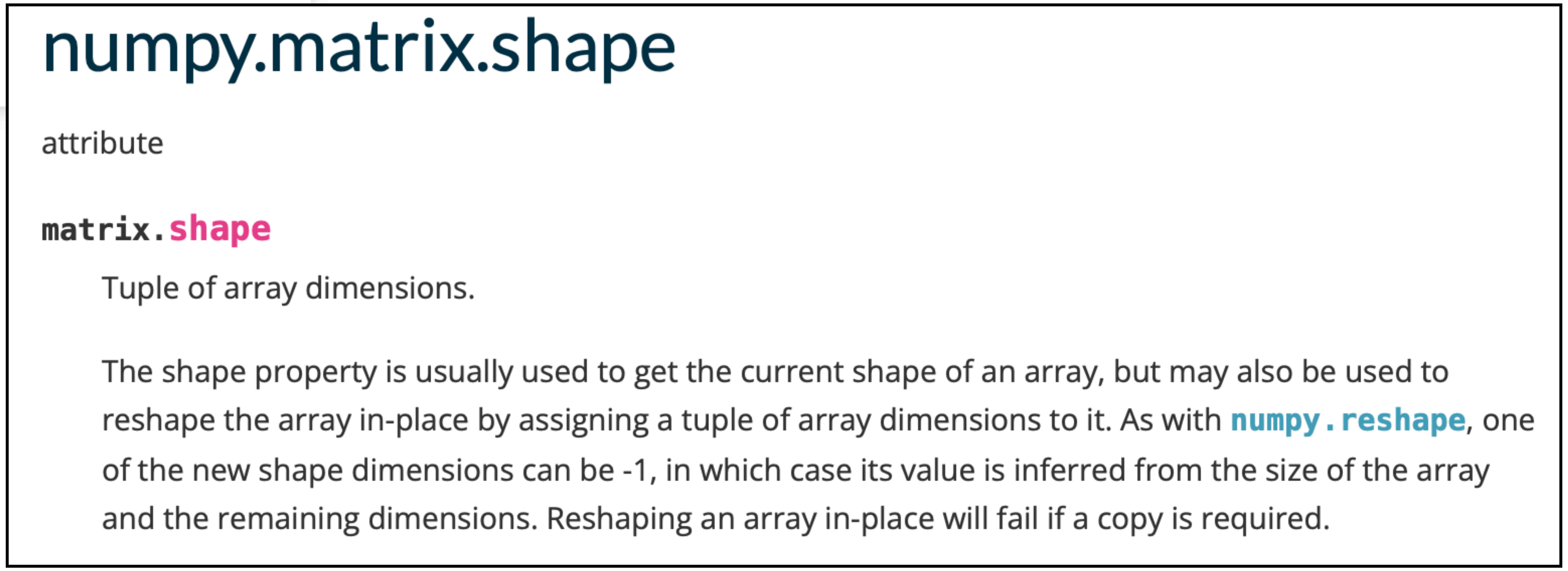
`len`

`len(a)` is equivalent to `np.shape(a)[0]` for N-D arrays with `N>=1`.

`ndarray.shape`

Equivalent array method.

len( ) also works with e.g. lists



numpy.matrix.shape

attribute

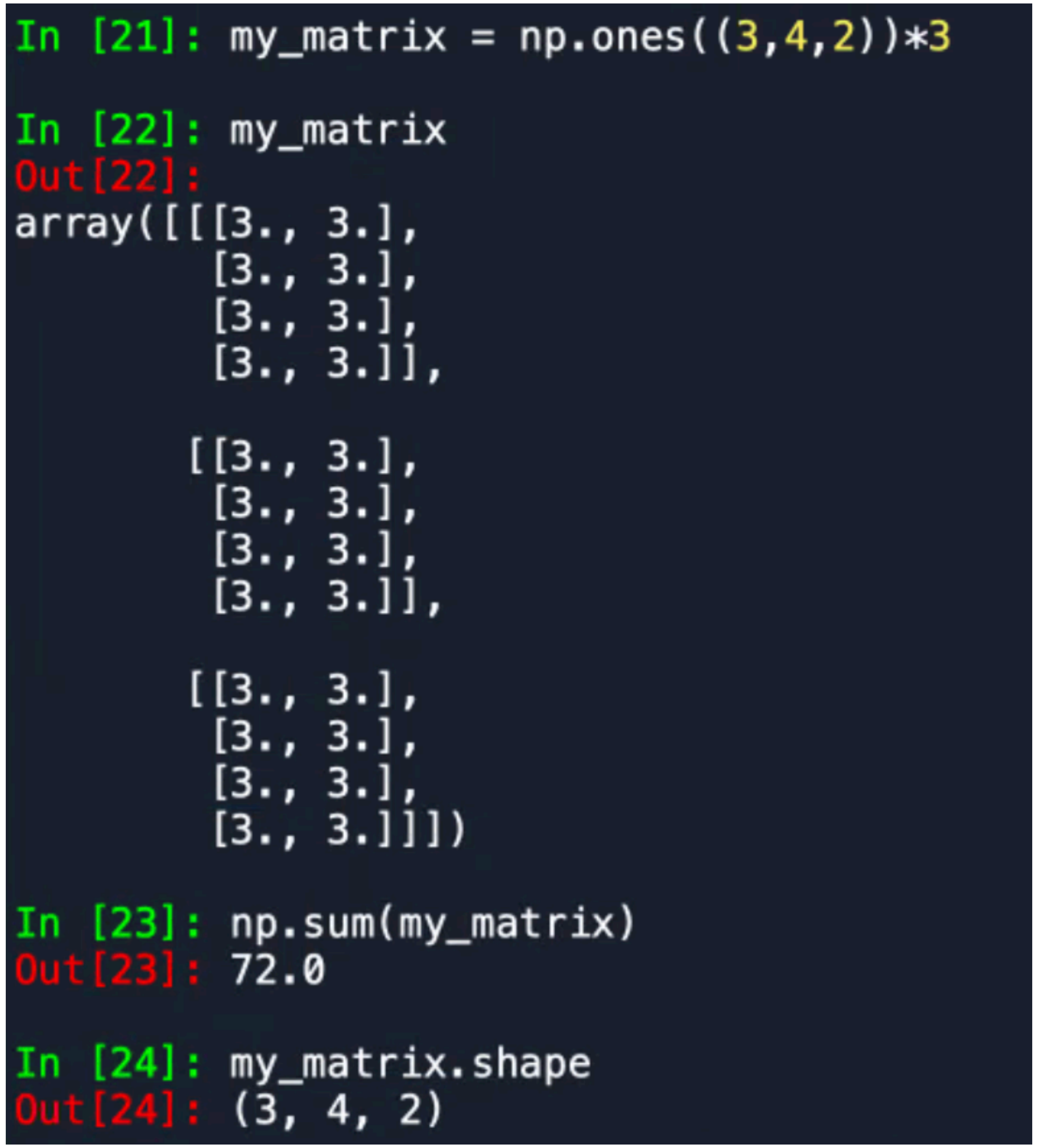
`matrix.shape`

Tuple of array dimensions.

The shape property is usually used to get the current shape of an array, but may also be used to reshape the array in-place by assigning a tuple of array dimensions to it. As with `numpy.reshape`, one of the new shape dimensions can be -1, in which case its value is inferred from the size of the array and the remaining dimensions. Reshaping an array in-place will fail if a copy is required.

Python: numpy

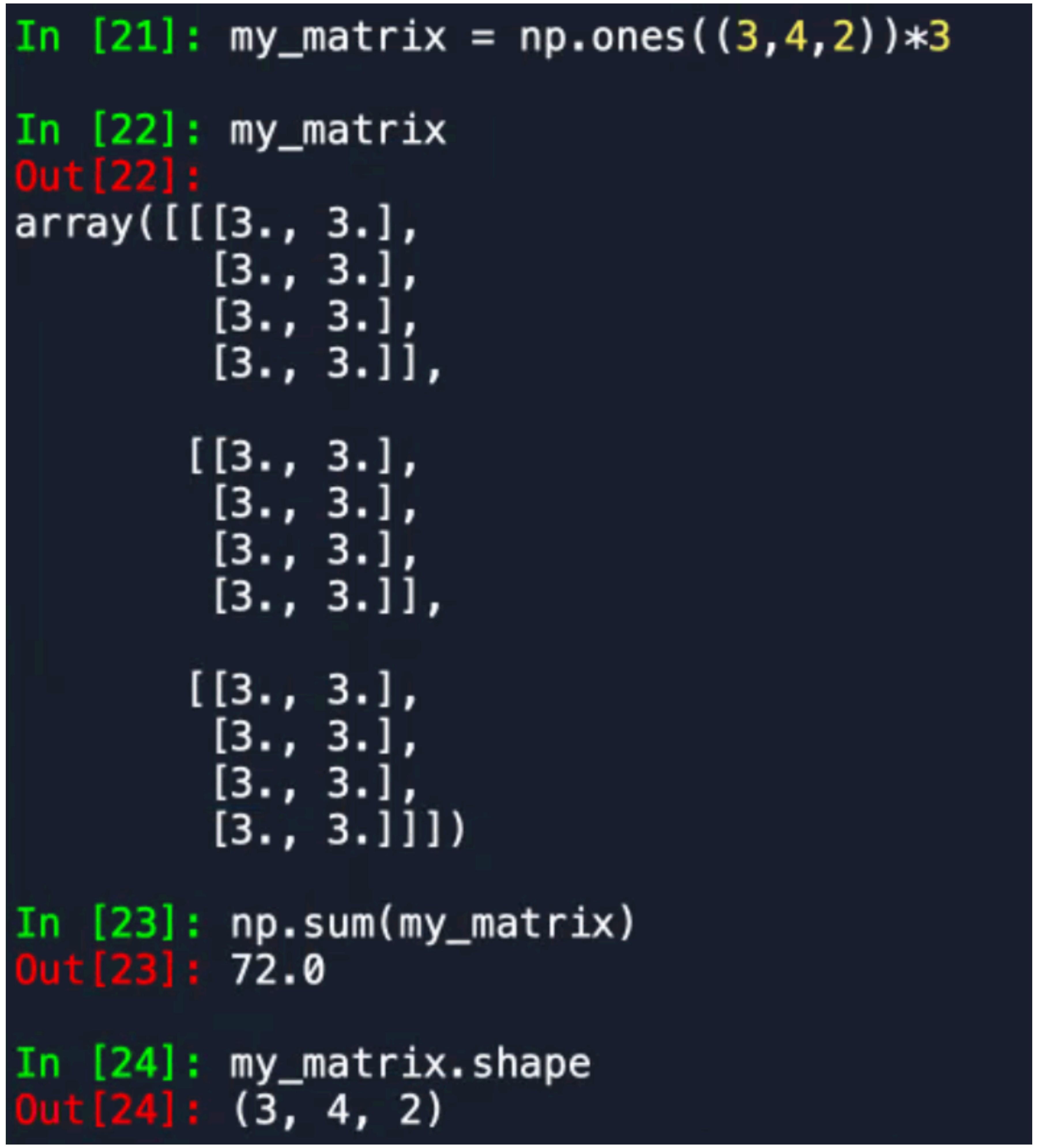
Operations with matrices



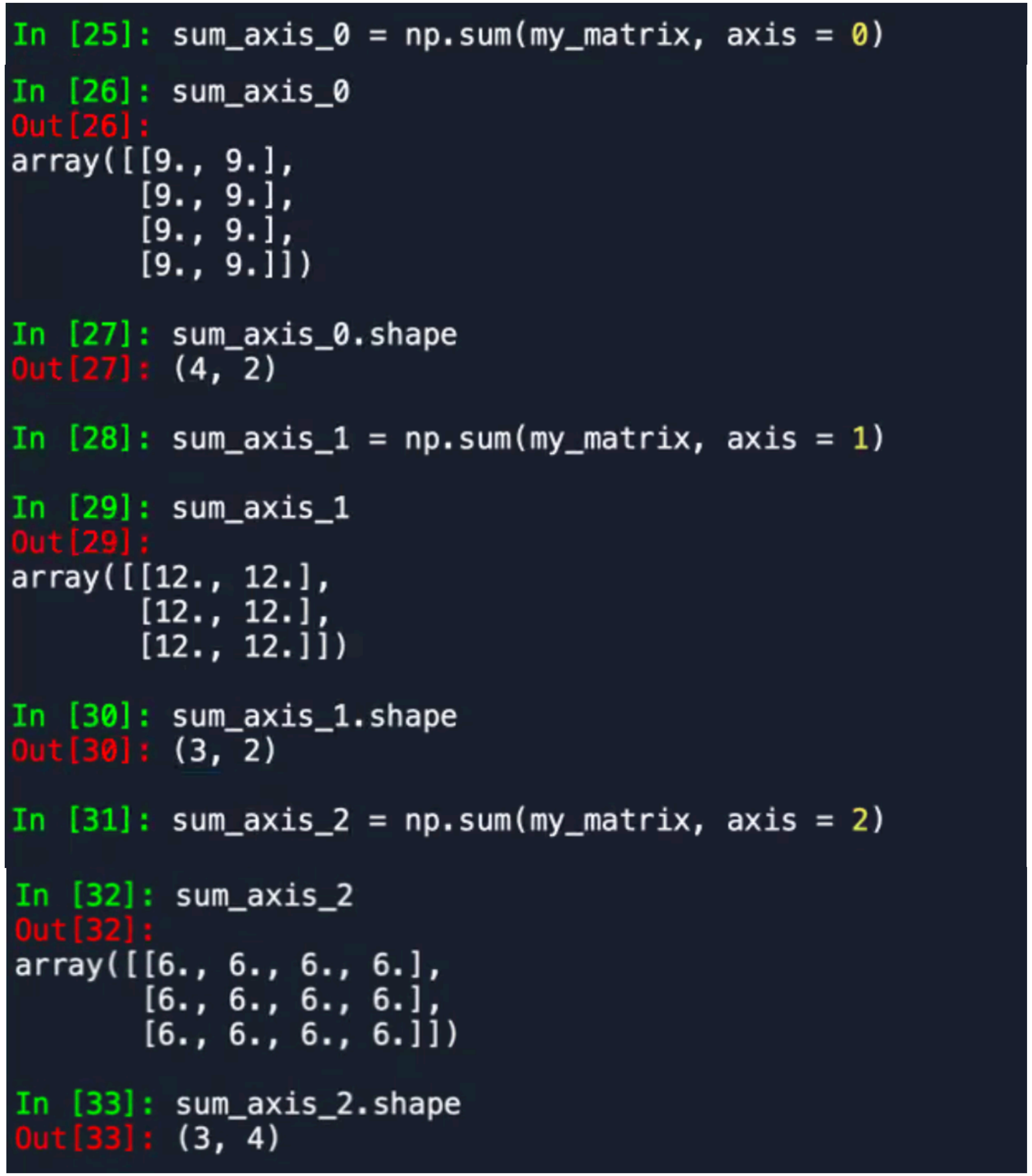
```
In [21]: my_matrix = np.ones((3,4,2))*3

In [22]: my_matrix
Out[22]:
array([[[3., 3.],
        [3., 3.],
        [3., 3.],
        [3., 3.]],

       [[3., 3.],
        [3., 3.],
        [3., 3.],
        [3., 3.]],

       [[3., 3.],
        [3., 3.],
        [3., 3.],
        [3., 3.]]])

In [23]: np.sum(my_matrix)
Out[23]: 72.0

In [24]: my_matrix.shape
Out[24]: (3, 4, 2)
```

Python: numpy

Operations with matrices



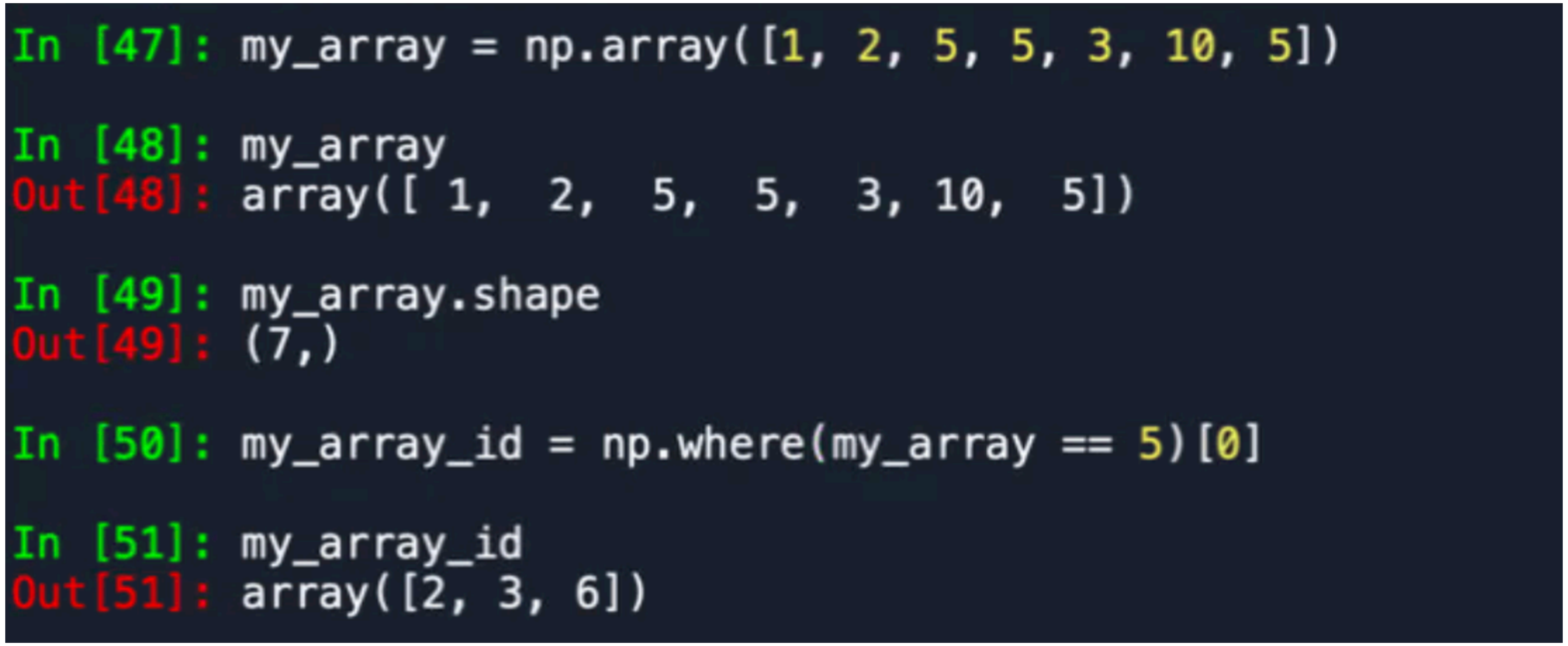
```
In [21]: my_matrix = np.ones((3,4,2))*3

In [22]: my_matrix
Out[22]:
array([[[3., 3.],
        [3., 3.],
        [3., 3.],
        [3., 3.]],

       [[3., 3.],
        [3., 3.],
        [3., 3.],
        [3., 3.]],

       [[3., 3.],
        [3., 3.],
        [3., 3.],
        [3., 3.]]])

In [23]: np.sum(my_matrix)
Out[23]: 72.0

In [24]: my_matrix.shape
Out[24]: (3, 4, 2)
```



```
In [25]: sum_axis_0 = np.sum(my_matrix, axis = 0)

In [26]: sum_axis_0
Out[26]:
array([[9., 9.],
       [9., 9.],
       [9., 9.],
       [9., 9.]])

In [27]: sum_axis_0.shape
Out[27]: (4, 2)

In [28]: sum_axis_1 = np.sum(my_matrix, axis = 1)

In [29]: sum_axis_1
Out[29]:
array([[12., 12.],
       [12., 12.],
       [12., 12.]])

In [30]: sum_axis_1.shape
Out[30]: (3, 2)

In [31]: sum_axis_2 = np.sum(my_matrix, axis = 2)

In [32]: sum_axis_2
Out[32]:
array([[6., 6., 6., 6.],
       [6., 6., 6., 6.],
       [6., 6., 6., 6.]])

In [33]: sum_axis_2.shape
Out[33]: (3, 4)
```

Python: numpy

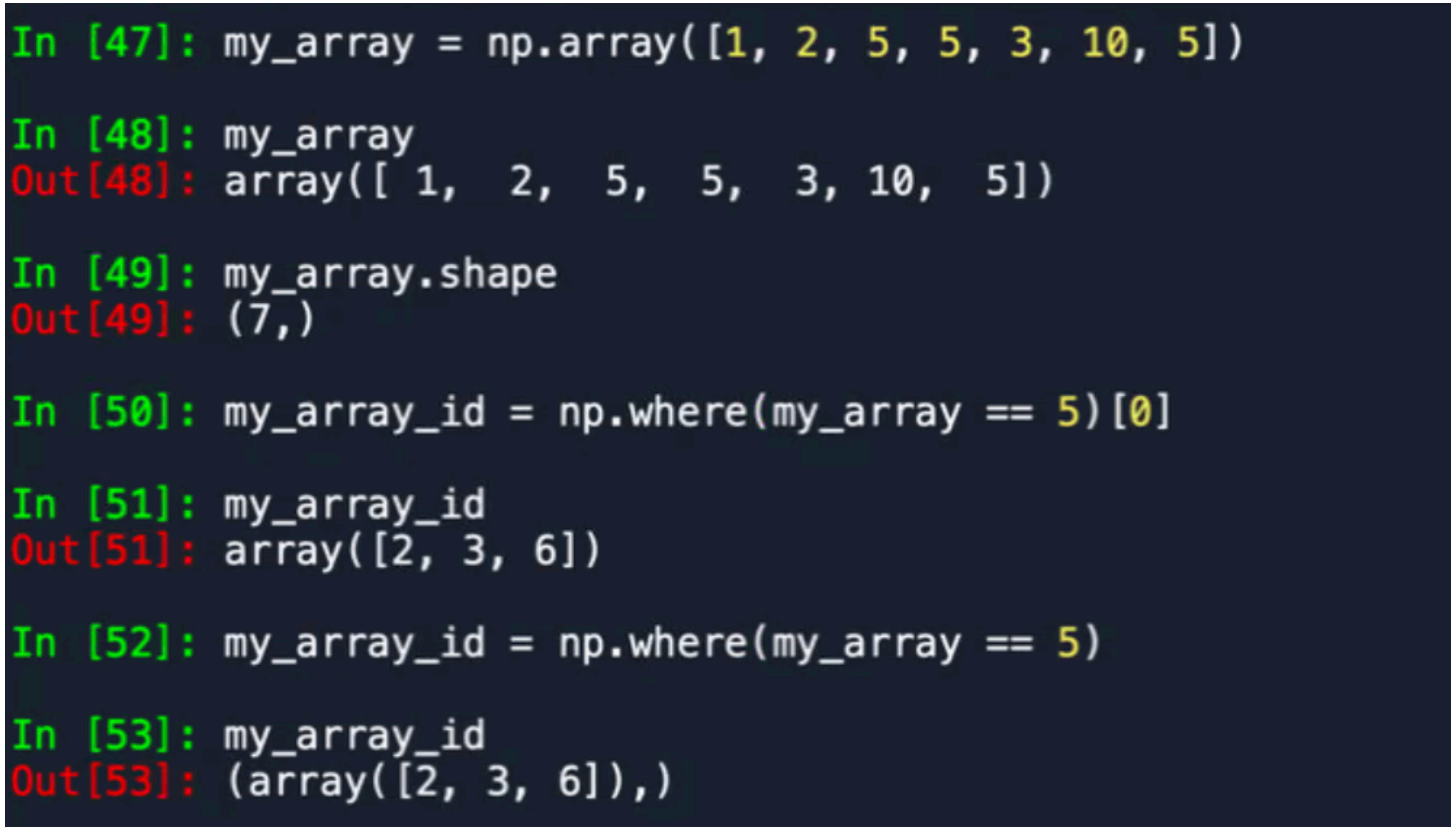
numpy.where



```
In [47]: my_array = np.array([1, 2, 5, 5, 3, 10, 5])

In [48]: my_array
Out[48]: array([ 1,  2,  5,  5,  3, 10,  5])

In [49]: my_array.shape
Out[49]: (7,)

In [50]: my_array_id = np.where(my_array == 5)[0]

In [51]: my_array_id
Out[51]: array([2, 3, 6])
```

Python: numpy

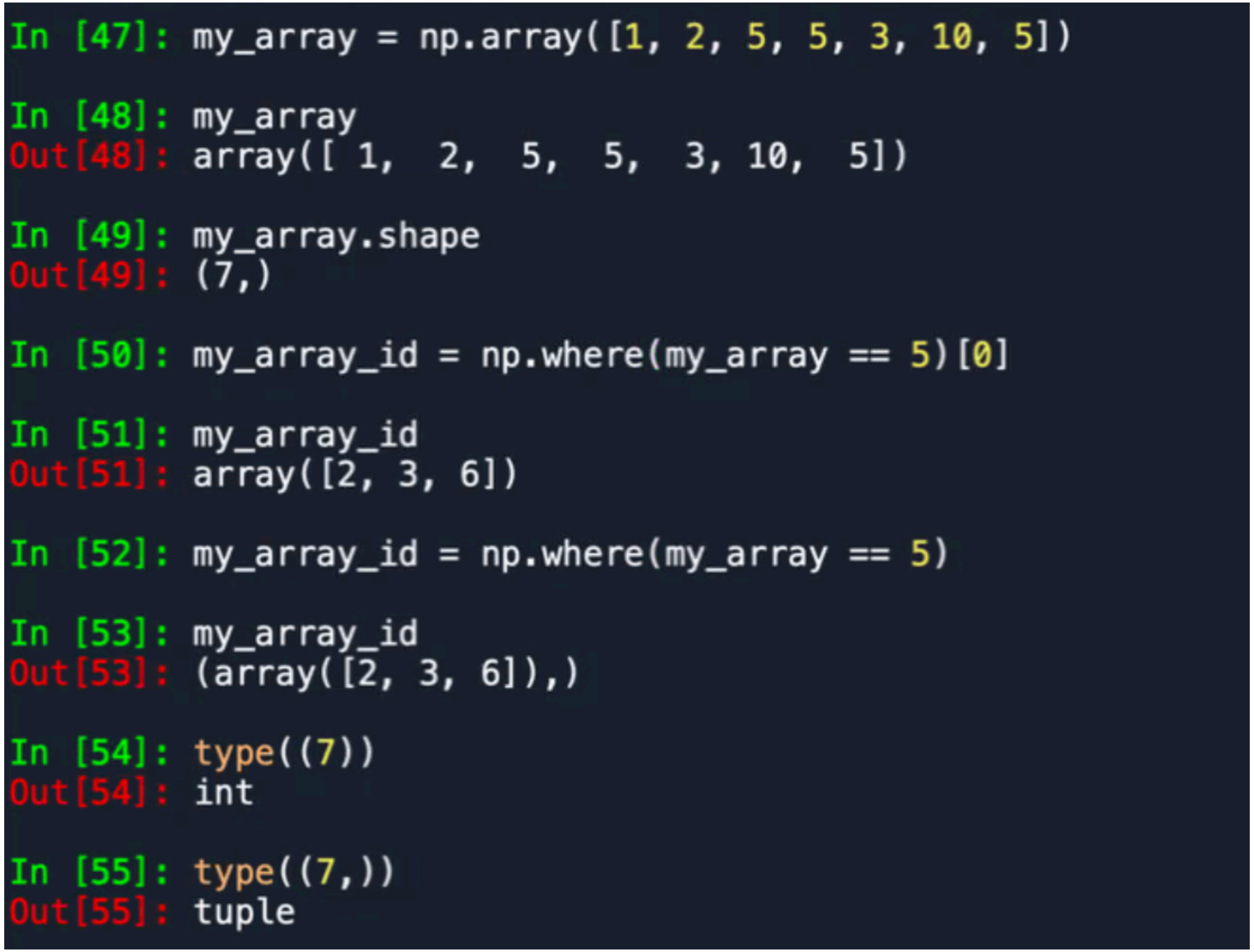
numpy.where



```
In [47]: my_array = np.array([1, 2, 5, 5, 3, 10, 5])

In [48]: my_array
Out[48]: array([ 1,  2,  5,  5,  3, 10,  5])

In [49]: my_array.shape
Out[49]: (7,)

In [50]: my_array_id = np.where(my_array == 5)[0]

In [51]: my_array_id
Out[51]: array([2, 3, 6])

In [52]: my_array_id = np.where(my_array == 5)

In [53]: my_array_id
Out[53]: (array([2, 3, 6]),)
```

Python: numpy

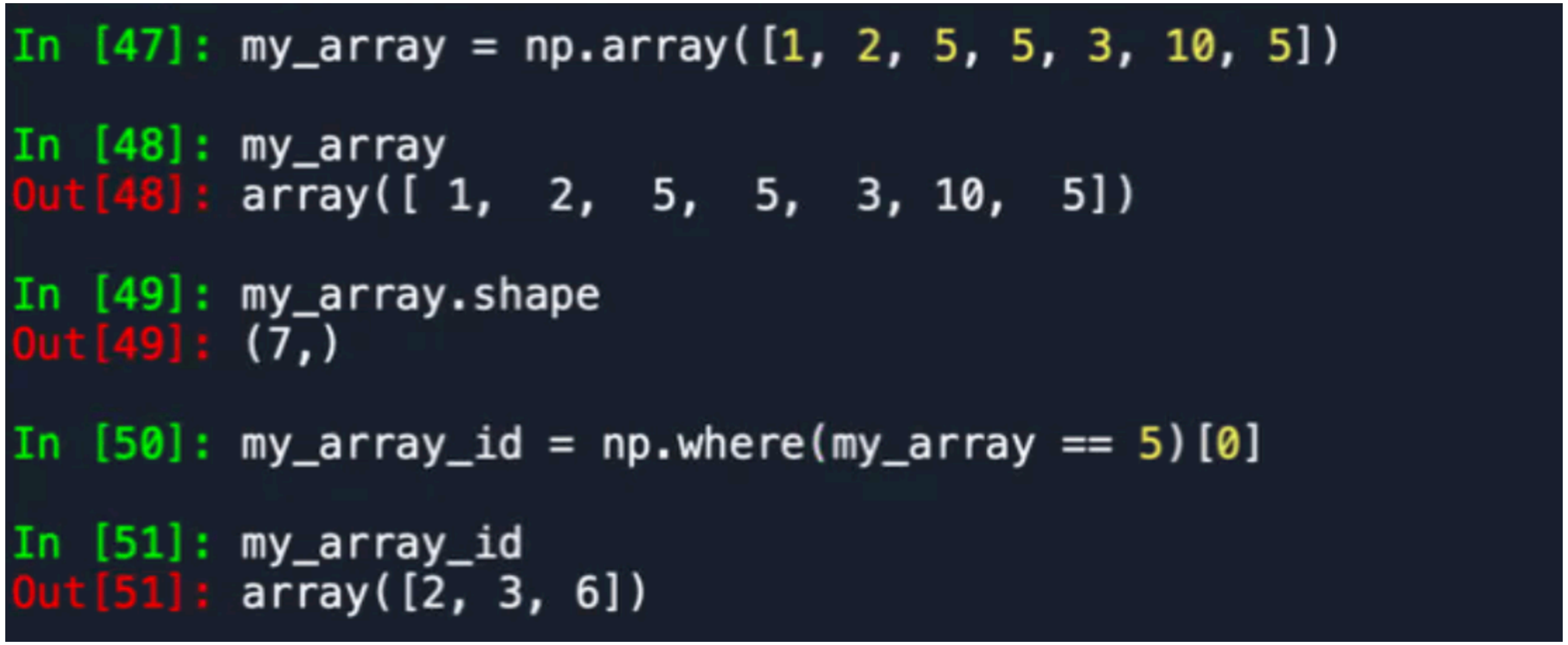
numpy.where



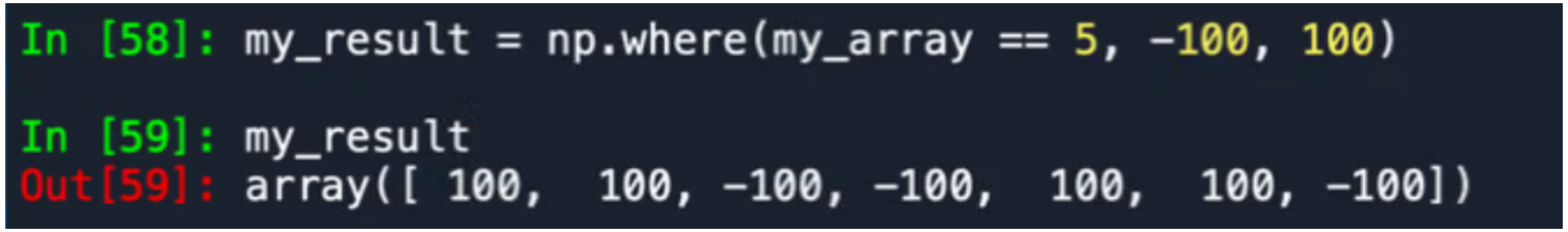
```
In [47]: my_array = np.array([1, 2, 5, 5, 3, 10, 5])

In [48]: my_array
Out[48]: array([ 1,  2,  5,  5,  3, 10,  5])

In [49]: my_array.shape
Out[49]: (7,)

In [50]: my_array_id = np.where(my_array == 5)[0]

In [51]: my_array_id
Out[51]: array([2, 3, 6])

In [52]: my_array_id = np.where(my_array == 5)

In [53]: my_array_id
Out[53]: (array([2, 3, 6]),)

In [54]: type((7))
Out[54]: int

In [55]: type((7,))
Out[55]: tuple
```

Python: numpy arrays

numpy.where



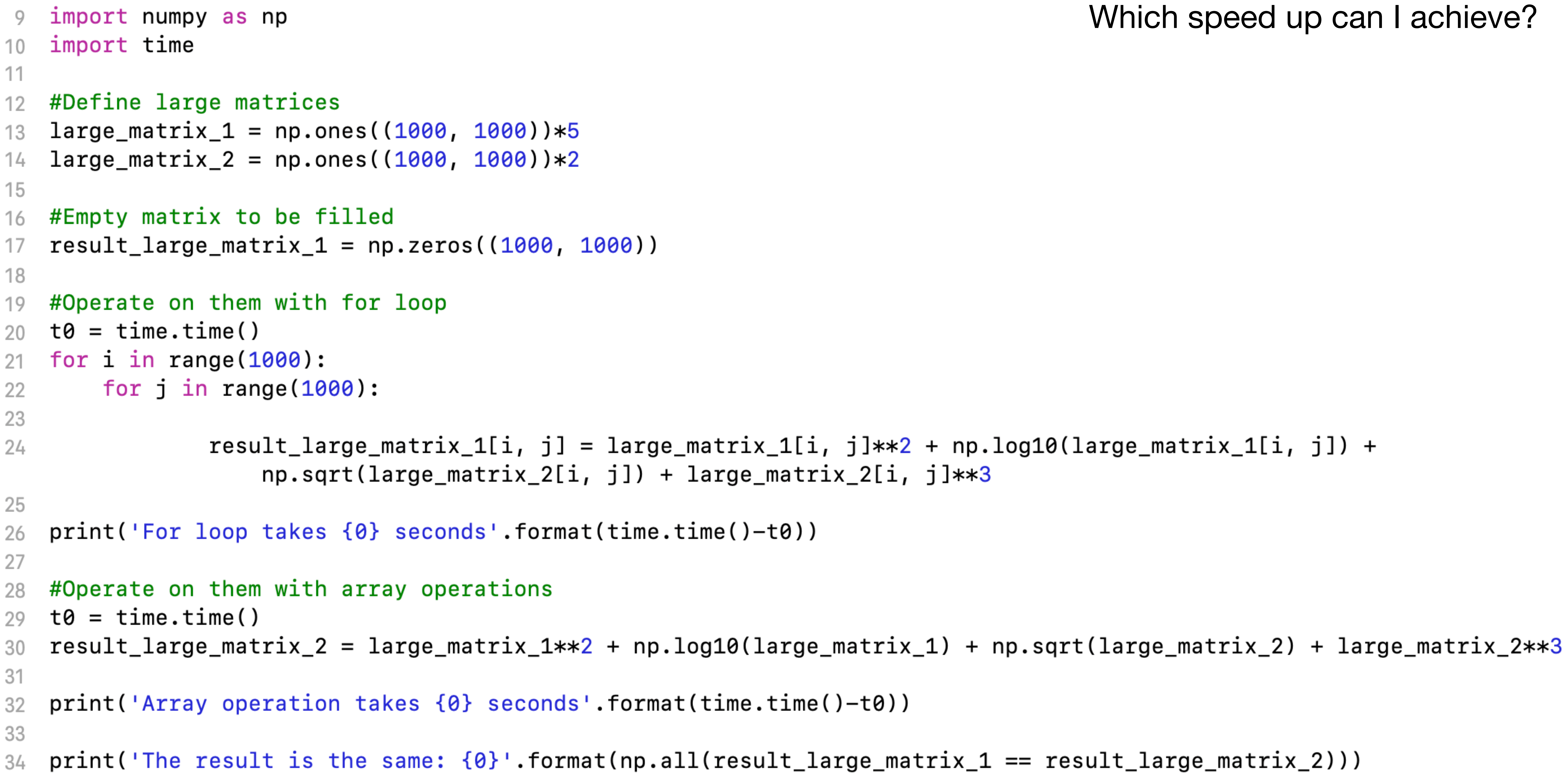
```
In [47]: my_array = np.array([1, 2, 5, 5, 3, 10, 5])

In [48]: my_array
Out[48]: array([ 1,  2,  5,  5,  3, 10,  5])

In [49]: my_array.shape
Out[49]: (7,)

In [50]: my_array_id = np.where(my_array == 5)[0]

In [51]: my_array_id
Out[51]: array([2, 3, 6])
```



```
In [58]: my_result = np.where(my_array == 5, -100, 100)

In [59]: my_result
Out[59]: array([ 100,  100, -100, -100,  100,  100, -100])
```

Python: numpy

Are array operations convenient?
Which speed up can I achieve?

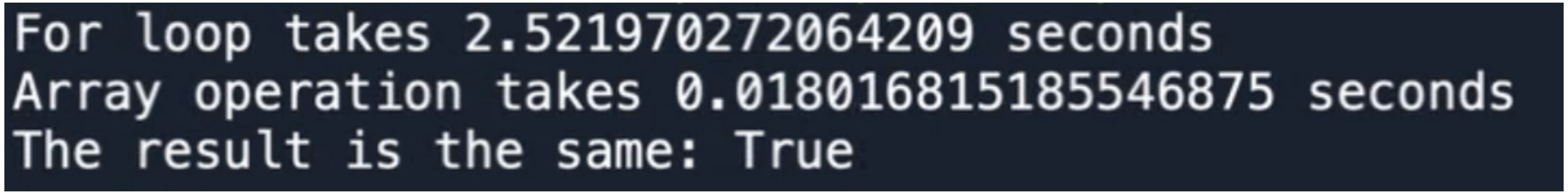


```python
import numpy as np
import time

#Define large matrices
large_matrix_1 = np.ones((1000, 1000))*5
large_matrix_2 = np.ones((1000, 1000))*2

#Empty matrix to be filled
result_large_matrix_1 = np.zeros((1000, 1000))

#Operate on them with for loop
t0 = time.time()
for i in range(1000):
    for j in range(1000):

            result_large_matrix_1[i, j] = large_matrix_1[i, j]**2 + np.log10(large_matrix_1[i, j]) + np.sqrt(large_matrix_2[i, j]) + large_matrix_2[i, j]**3

print('For loop takes {0} seconds'.format(time.time()-t0))

#Operate on them with array operations
t0 = time.time()
result_large_matrix_2 = large_matrix_1**2 + np.log10(large_matrix_1) + np.sqrt(large_matrix_2) + large_matrix_2**3

print('Array operation takes {0} seconds'.format(time.time()-t0))

print('The result is the same: {0}'.format(np.all(result_large_matrix_1 == result_large_matrix_2)))
```

Python: numpy

Are array operations convenient?
Which speed up can I achieve?

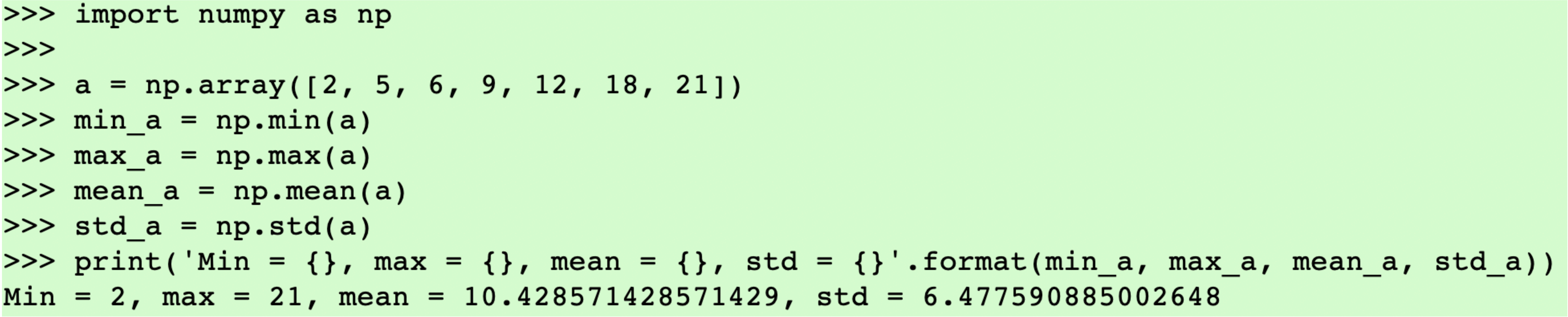
```python
import numpy as np
import time

#Define large matrices
large_matrix_1 = np.ones((1000, 1000))*5
large_matrix_2 = np.ones((1000, 1000))*2

#Empty matrix to be filled
result_large_matrix_1 = np.zeros((1000, 1000))

#Operate on them with for loop
t0 = time.time()
for i in range(1000):
    for j in range(1000):

            result_large_matrix_1[i, j] = large_matrix_1[i, j]**2 + np.log10(large_matrix_1[i, j]) + np.sqrt(large_matrix_2[i, j]) + large_matrix_2[i, j]**3

print('For loop takes {0} seconds'.format(time.time()-t0))

#Operate on them with array operations
t0 = time.time()
result_large_matrix_2 = large_matrix_1**2 + np.log10(large_matrix_1) + np.sqrt(large_matrix_2) + large_matrix_2**3

print('Array operation takes {0} seconds'.format(time.time()-t0))

print('The result is the same: {0}'.format(np.all(result_large_matrix_1 == result_large_matrix_2)))
```



```
For loop takes 2.52197027206420 9 seconds
Array operation takes 0.018016815185546875 seconds
The result is the same: True
```

Python: numpy

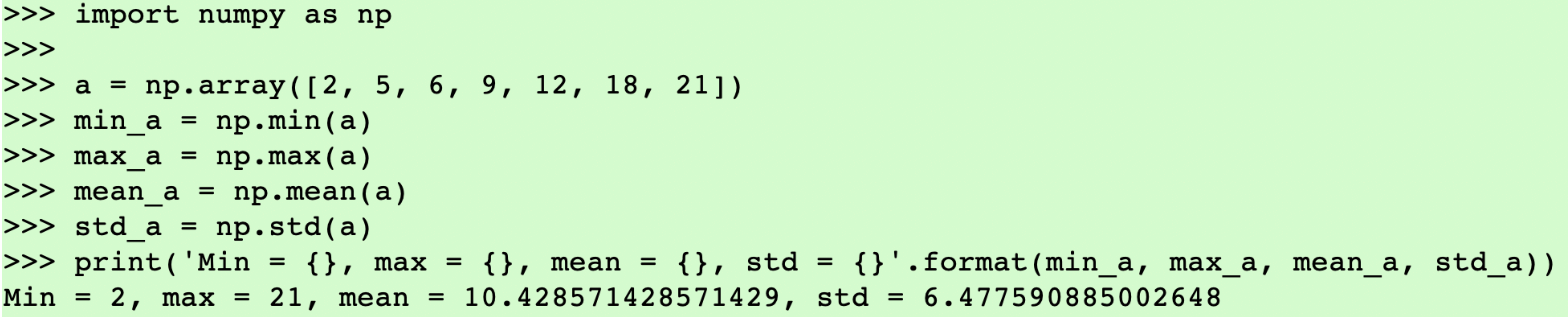
Other relevant features of numpy



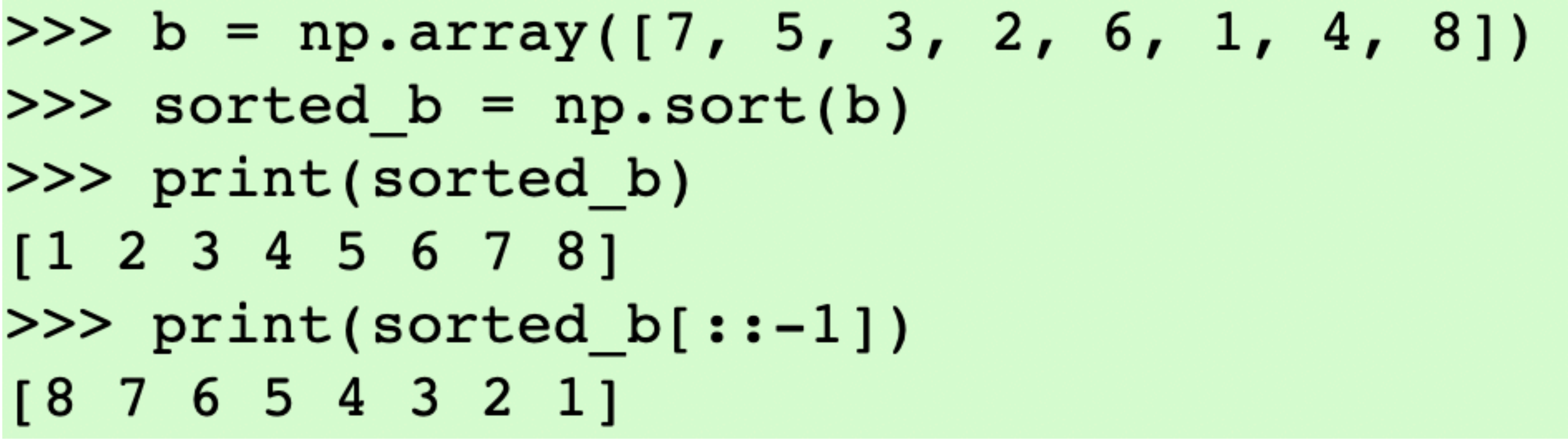
```
>>> import numpy as np
>>> 
>>> a = np.array([2, 5, 6, 9, 12, 18, 21])
>>> min_a = np.min(a)
>>> max_a = np.max(a)
>>> mean_a = np.mean(a)
>>> std_a = np.std(a)
>>> print('Min = {}, max = {}, mean = {}, std = {}'.format(min_a, max_a, mean_a, std_a))
Min = 2, max = 21, mean = 10.428571428571429, std = 6.477590885002648
```

Python: numpy

Other relevant features of numpy



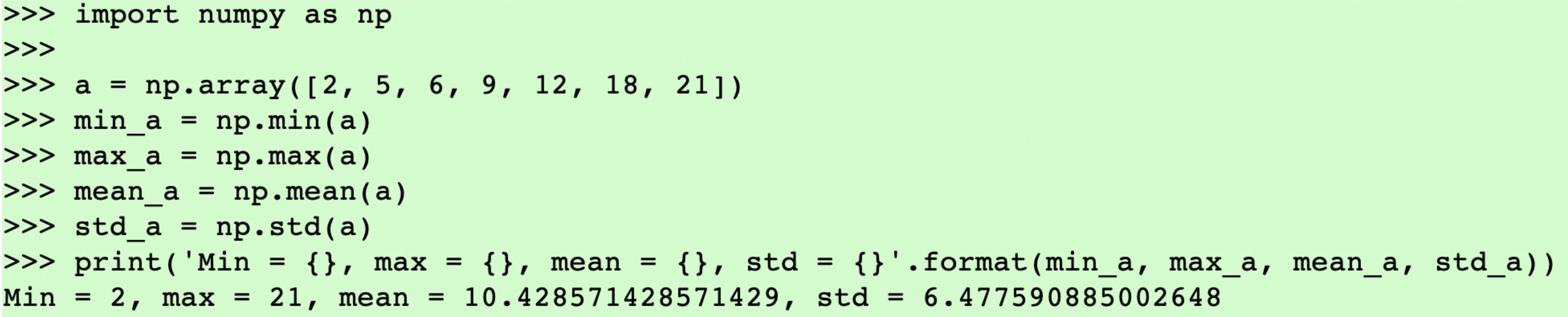
```
>>> import numpy as np
>>> 
>>> a = np.array([2, 5, 6, 9, 12, 18, 21])
>>> min_a = np.min(a)
>>> max_a = np.max(a)
>>> mean_a = np.mean(a)
>>> std_a = np.std(a)
>>> print('Min = {}, max = {}, mean = {}, std = {}'.format(min_a, max_a, mean_a, std_a))
Min = 2, max = 21, mean = 10.428571428571429, std = 6.477590885002648
```



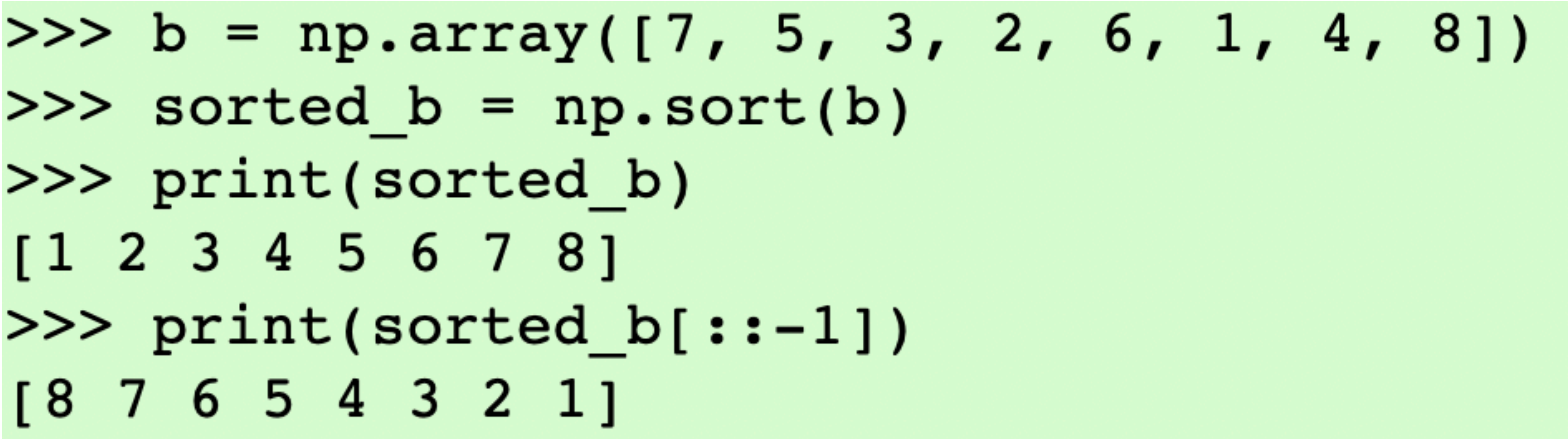
```
>>> b = np.array([7, 5, 3, 2, 6, 1, 4, 8])
>>> sorted_b = np.sort(b)
>>> print(sorted_b)
[1 2 3 4 5 6 7 8]
>>> print(sorted_b[::-1])
[8 7 6 5 4 3 2 1]
```

Python: numpy

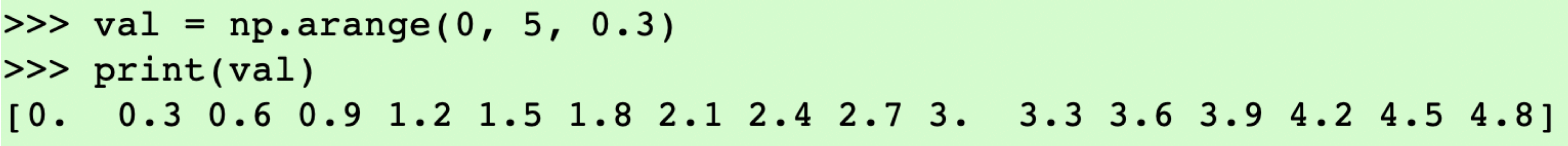
Other relevant features of numpy



```
>>> import numpy as np
>>>
>>> a = np.array([2, 5, 6, 9, 12, 18, 21])
>>> min_a = np.min(a)
>>> max_a = np.max(a)
>>> mean_a = np.mean(a)
>>> std_a = np.std(a)
>>> print('Min = {}, max = {}, mean = {}, std = {}'.format(min_a, max_a, mean_a, std_a))
Min = 2, max = 21, mean = 10.428571428571429, std = 6.477590885002648
```



```
>>> b = np.array([7, 5, 3, 2, 6, 1, 4, 8])
>>> sorted_b = np.sort(b)
>>> print(sorted_b)
[1 2 3 4 5 6 7 8]
>>> print(sorted_b[::-1])
[8 7 6 5 4 3 2 1]
```



```
>>> val = np.arange(0, 5, 0.3)
>>> print(val)
[0.  0.3 0.6 0.9 1.2 1.5 1.8 2.1 2.4 2.7 3.  3.3 3.6 3.9 4.2 4.5 4.8]
```

Python: exercise

Exercise 1.

Create two arrays (for instance with numpy.arange).
Let’s call them array_1 and array_2.
Put some zeros in at least 10 entries of array_2.

Goal: compute array_1 / array_2

Intermediate steps to perform:

I. — use numpy.where to identify the zeros in array_2, and then compute array_1 / array_2 only when the zero entries are not involved.

II. — Use numpy.where to replace the zeros in array_2 with 0.001, and then compute array_1 / array_2 for all the entries.

III. — Try to compute array_1 / array_2 (with array_1 and array_2 being the original arrays without modifications) and print the indices of the resulting array where the division by zero occurs (use numpy.isinf).

Python: libraries

matplotlib

The Astropy Project

SciPy

pandas

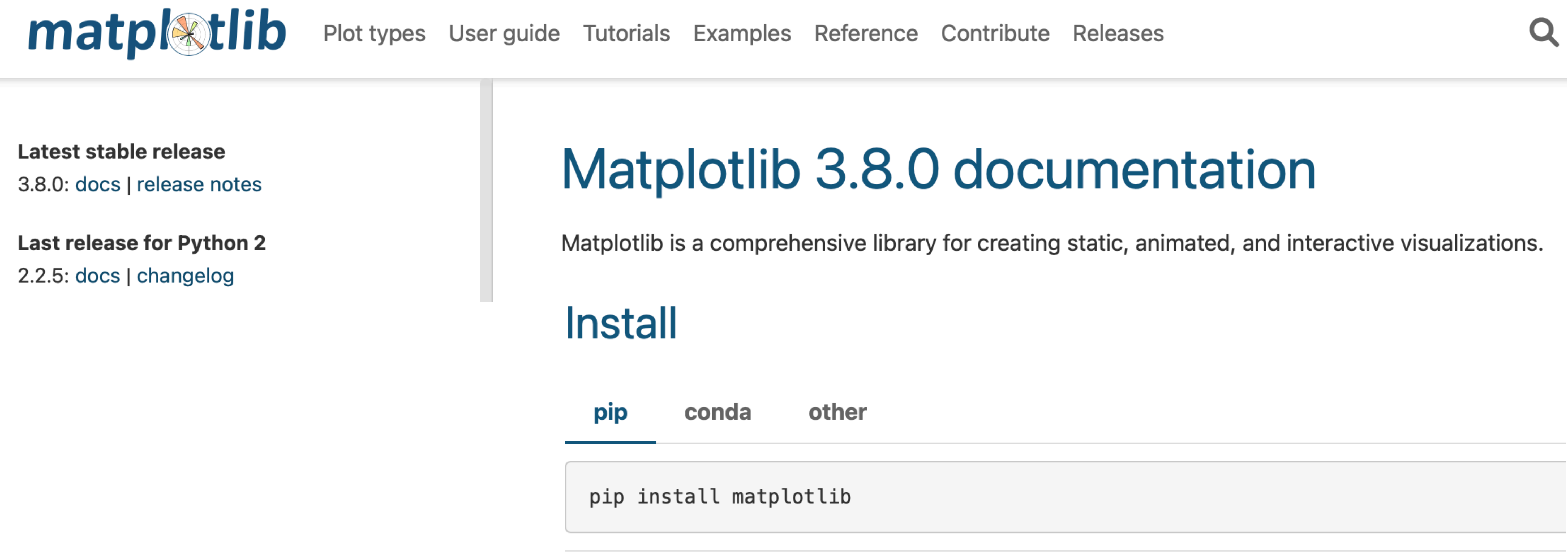
A Python library is a collection of codes and modules.

It contains bunches of code that can be used repeatedly in different programs for specific operations.

We use libraries so that we don't need to write the code again in our program that is already available.

PyTorch

Python: libraries



Latest stable release
3.8.0: docs | release notes

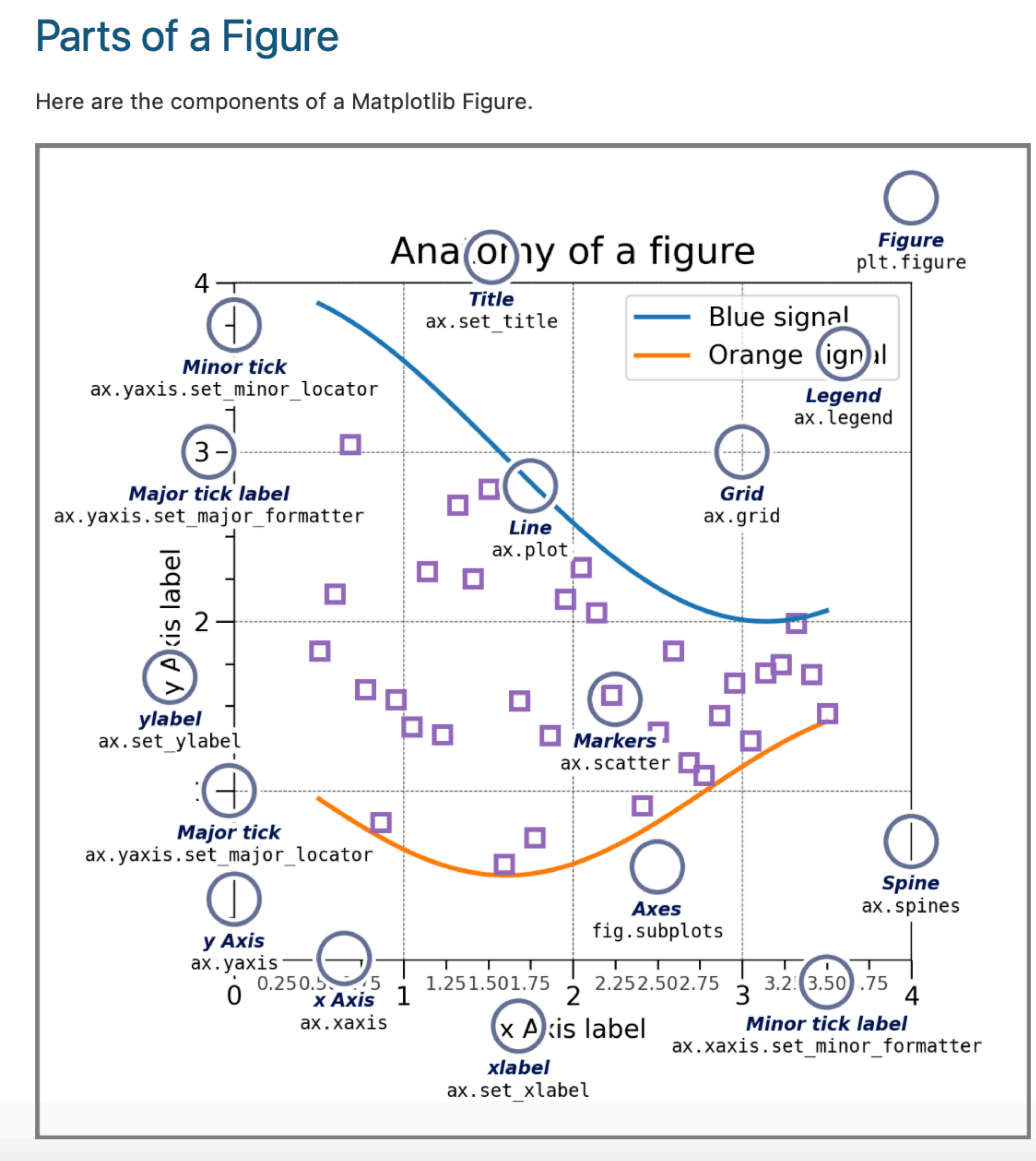
Last release for Python 2
2.2.5: docs | changelog

Matplotlib 3.8.0 documentation

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations.

Install

```
pip install matplotlib
```



Parts of a Figure

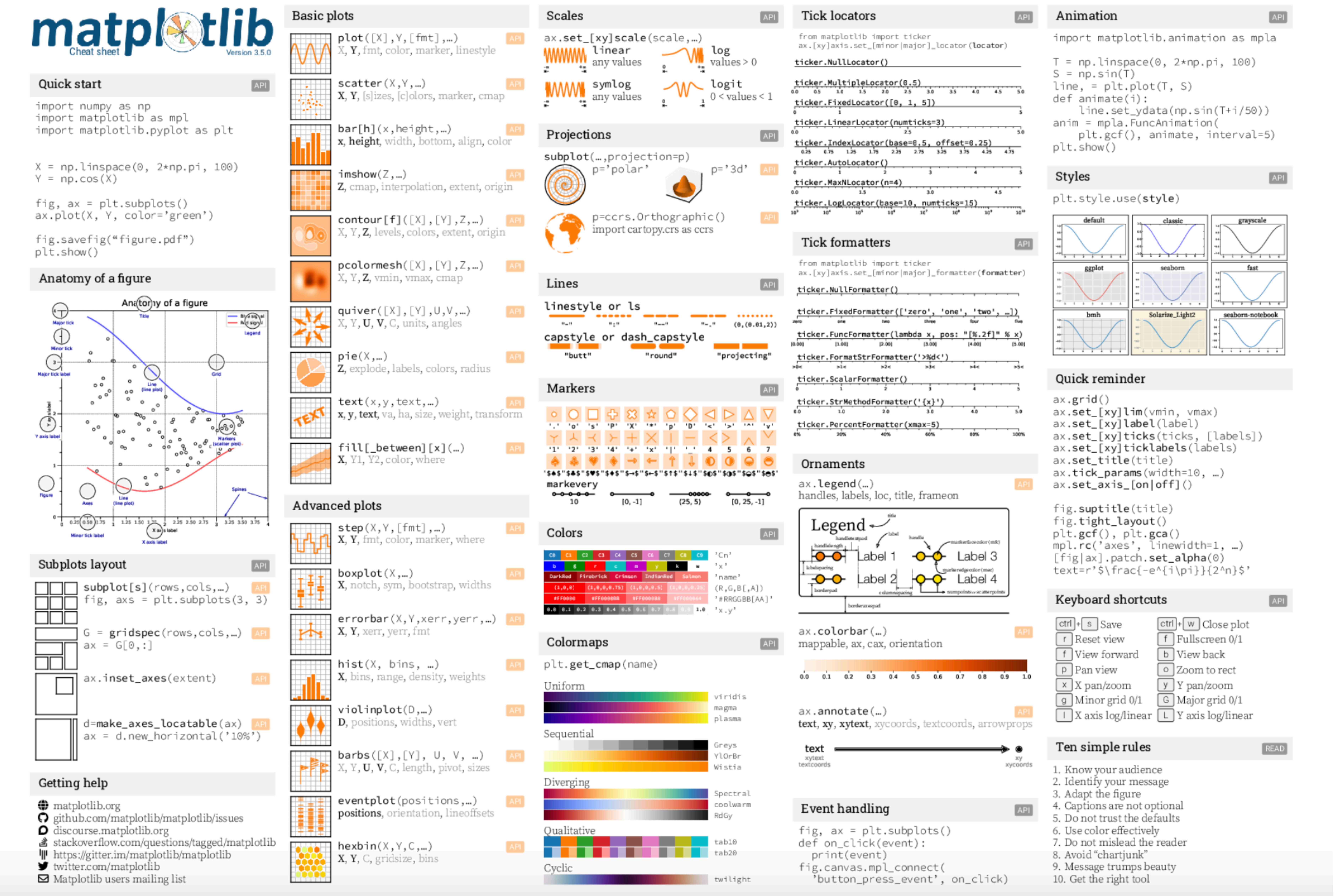
Here are the components of a Matplotlib Figure.

A Python library is a collection of codes and modules.

It contains bunches of code that can be used repeatedly in different programs for specific operations.

We use libraries so that we don't need to write the code again in our program that is already available.

Python: libraries



matplotlib

Cheat sheet Version 3.5.0

Quick start

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


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
plt.show()
```

Anatomy of a figure

Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Grid
Legend
Blue signal
Red signal
Line (line plot)
Y axis label
Markers (scatter plot)
Figure
Axes
Line (line plot)
Spines
Minor tick label
X axis label

Subplots layout

```
subplot[s](rows,cols,…)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

`plot([X],Y,[fmt],…)`
X, Y, fmt, color, marker, linestyle

`scatter(X,Y,…)`
X, Y, [s]izes, [c]olors, marker, cmap

`bar[h](x,height,…)`
x, height, width, bottom, align, color

`imshow(Z,…)`
Z, cmap, interpolation, extent, origin

`contour[f]([X],[Y],Z,…)`
X, Y, Z, levels, colors, extent, origin

`pcolormesh([X],[Y],Z,…)`
X, Y, Z, vmin, vmax, cmap

`quiver([X],[Y],U,V,…)`
X, Y, U, V, C, units, angles

`pie(X,…)`
Z, explode, labels, colors, radius

`text(x,y,text,…)`
x, y, text, va, ha, size, weight, transform

`fill[_between][x](…)`
X, Y1, Y2, color, where

Advanced plots

`step(X,Y,[fmt],…)`
X, Y, fmt, color, marker, where

`boxplot(X,…)`
X, notch, sym, bootstrap, widths

`errorbar(X,Y,xerr,yerr,…)`
X, Y, xerr, yerr, fmt

`hist(X, bins, …)`
X, bins, range, density, weights

`violinplot(D,…)`
D, positions, widths, vert

`barbs([X],[Y], U, V, …)`
X, Y, U, V, C, length, pivot, sizes

`eventplot(positions,…)`
positions, orientation, lineoffsets

`hexbin(X,Y,C,…)`
X, Y, C, gridsize, bins

Scales

`ax.set_[xy]scale(scale,…)`

linear
any values

log
values > 0

symlog
any values

logit
0 < values < 1

Projections

`subplot(…,projection=p)`

p='polar'

p='3d'

p=ccrs.Orthographic()
import cartopy.crs as ccrs

Lines

`linestyle or ls`

"-" ":" "--" "-." (0,(0.01,2))

`capstyle or dash_capstyle`

"butt" "round" "projecting"

Markers

'.' 'o' 's' 'P' 'X' '*' 'p' 'D' '<' '>' '^' 'v'

'1' '2' '3' '4' '+' 'x' '|' '_' 4 5 6 7

'$♠$' '$♣$' '$♥$' '$♦$' '$→$' '$←$' '$↑$' '$↓$' '$◐$' '$◑$' '$◒$' '$◓$'

markevery

10 [0, -1] (25, 5) [0, 25, -1]

Colors

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 'Cn'

b g r c m y k w 'x'

DarkRed Firebrick Crimson IndianRed Salmon 'name'

(1,0,0) (1,0,0,0.75) (1,0,0,0.5) (1,0,0,0.25) (R,G,B[,A])

#FF0000 #FF0000BB #FF000088 #FF000044 '#RRGGBB[AA]'

0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 'x.y'

Colormaps

`plt.get_cmap(name)`

Uniform
viridis
magma
plasma

Sequential
Greys
YlOrBr
Wistia

Diverging
Spectral
coolwarm
RdGy

Qualitative
tab10
tab20

Cyclic
twilight

Tick locators

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```

```
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```

```
ticker.NullFormatter()
ticker.FixedFormatter(['zero', 'one', 'two', …])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

`ax.legend(…)`
handles, labels, loc, title, frameon

`ax.colorbar(…)`
mappable, ax, cax, orientation

`ax.annotate(…)`
text, xy, xytext, xycoords, textcoords, arrowprops

Event handling

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

`plt.style.use(style)`

Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

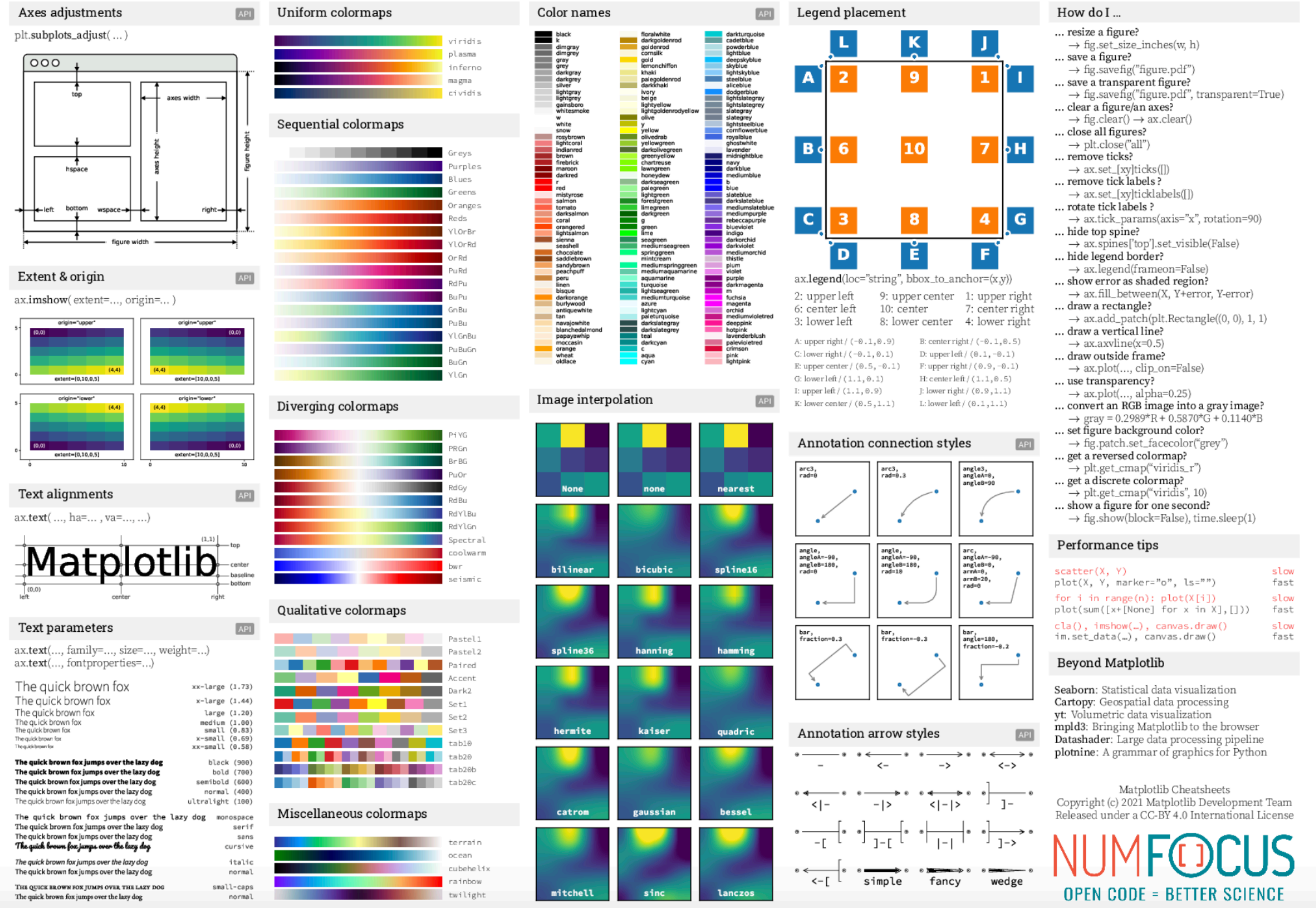
Keyboard shortcuts

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

Ten simple rules

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

Python: libraries



Axes adjustments

plt.subplots_adjust(...)

Extent & origin

ax.imshow(extent=..., origin=...)

Text alignments

ax.text(..., ha=... , va=..., ...)

Text parameters

ax.text(..., family=..., size=..., weight=...)
ax.text(..., fontproperties=...)

```
The quick brown fox                               xx-large (1.73)
The quick brown fox                                x-large (1.44)
The quick brown fox                                  large (1.20)
The quick brown fox                                 medium (1.00)
The quick brown fox                                  small (0.83)
The quick brown fox                                x-small (0.69)
The quick brown fox                               xx-small (0.58)

The quick brown fox jumps over the lazy dog        black (900)
The quick brown fox jumps over the lazy dog         bold (700)
The quick brown fox jumps over the lazy dog     semibold (600)
The quick brown fox jumps over the lazy dog       normal (400)
The quick brown fox jumps over the lazy dog   ultralight (100)

The quick brown fox jumps over the lazy dog      monospace
The quick brown fox jumps over the lazy dog          serif
The quick brown fox jumps over the lazy dog           sans
The quick brown fox jumps over the lazy dog        cursive

The quick brown fox jumps over the lazy dog         italic
The quick brown fox jumps over the lazy dog         normal

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG     small-caps
The quick brown fox jumps over the lazy dog         normal
```

Uniform colormaps

Sequential colormaps

Diverging colormaps

Qualitative colormaps

Miscellaneous colormaps

Color names

Image interpolation

Legend placement

ax.legend(loc="string", bbox_to_anchor=(x,y))

2: upper left	9: upper center	1: upper right
6: center left	10: center	7: center right
3: lower left	8: lower center	4: lower right

A: upper right / (-0.1,0.9)	B: center right / (-0.1,0.5)
C: lower right / (-0.1,0.1)	D: upper left / (0.1,-0.1)
E: upper center / (0.5,-0.1)	F: upper right / (0.9,-0.1)
G: lower left / (1.1,0.1)	H: center left / (1.1,0.5)
I: upper left / (1.1,0.9)	J: lower right / (0.9,1.1)
K: lower center / (0.5,1.1)	L: lower left / (0.1,1.1)

Annotation connection styles

Annotation arrow styles

How do I ...

... resize a figure?
→ fig.set_size_inches(w, h)
... save a figure?
→ fig.savefig("figure.pdf")
... save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
... clear a figure/an axes?
→ fig.clear() → ax.clear()
... close all figures?
→ plt.close("all")
... remove ticks?
→ ax.set_[xy]ticks([])
... remove tick labels ?
→ ax.set_[xy]ticklabels([])
... rotate tick labels ?
→ ax.tick_params(axis="x", rotation=90)
... hide top spine?
→ ax.spines['top'].set_visible(False)
... hide legend border?
→ ax.legend(frameon=False)
... show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
... draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0), 1, 1)
... draw a vertical line?
→ ax.axvline(x=0.5)
... draw outside frame?
→ ax.plot(..., clip_on=False)
... use transparency?
→ ax.plot(..., alpha=0.25)
... convert an RGB image into a gray image?
→ gray = 0.2989*R + 0.5870*G + 0.1140*B
... set figure background color?
→ fig.patch.set_facecolor("grey")
... get a reversed colormap?
→ plt.get_cmap("viridis_r")
... get a discrete colormap?
→ plt.get_cmap("viridis", 10)
... show a figure for one second?
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                                slow
plot(X, Y, marker="o", ls="")                fast
for i in range(n): plot(X[i])                slow
plot(sum([x+[None] for x in X],[]))          fast
cla(), imshow(…), canvas.draw()              slow
im.set_data(…), canvas.draw()                fast
```

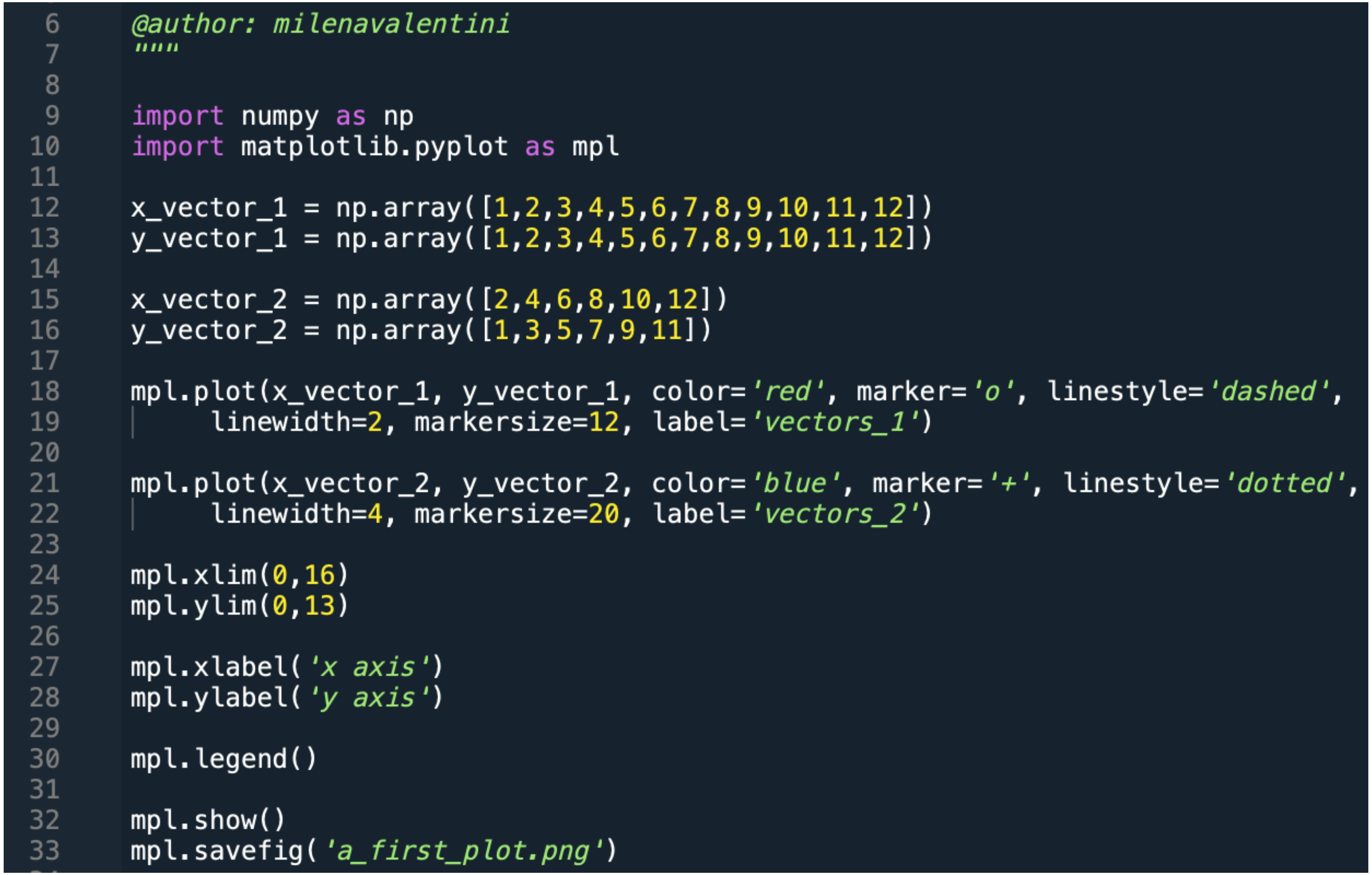
Beyond Matplotlib

Seaborn: Statistical data visualization
Cartopy: Geospatial data processing
yt: Volumetric data visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A grammar of graphics for Python

Matplotlib Cheatsheets

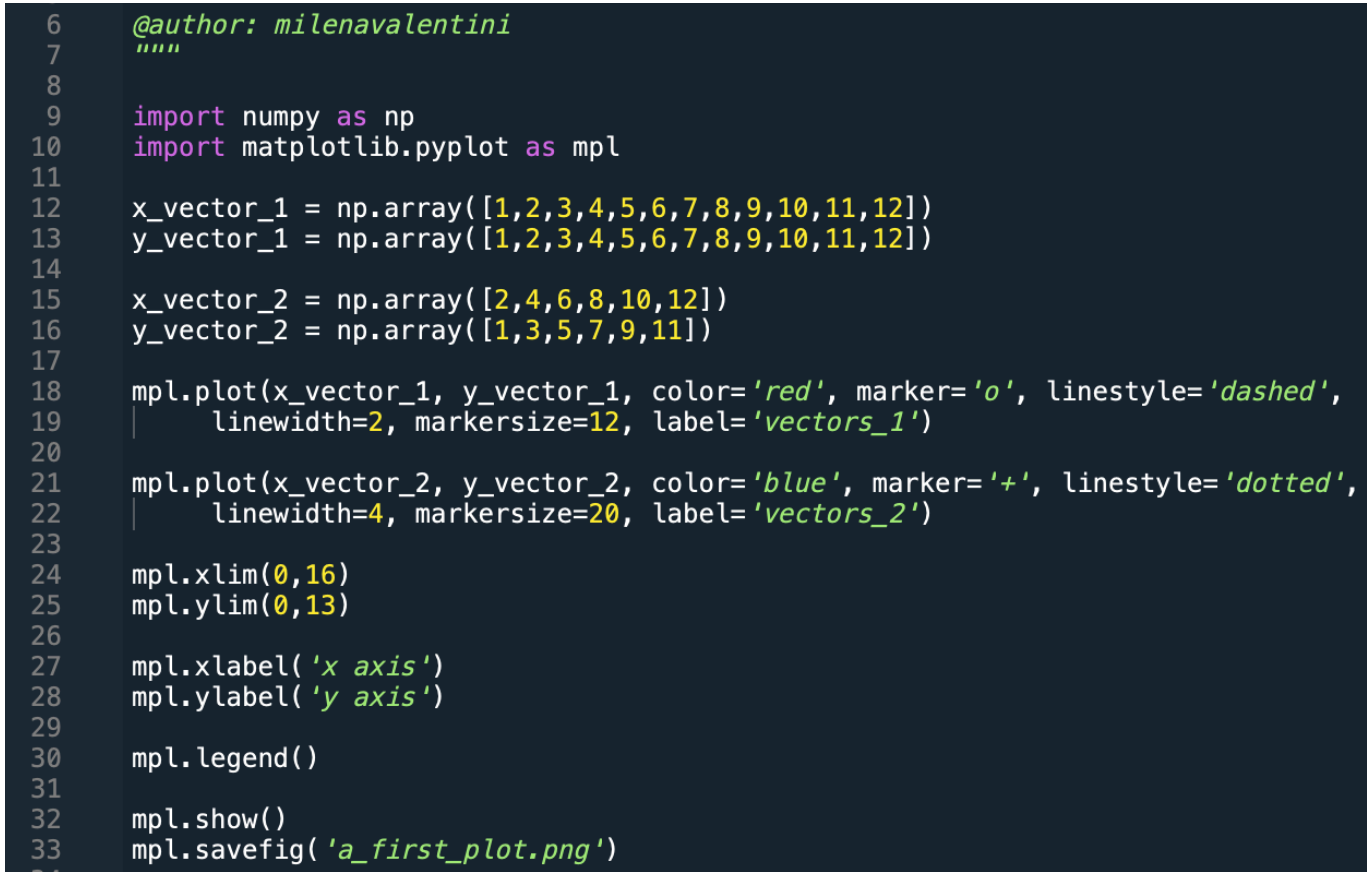
NUMFOCUS
OPEN CODE = BETTER SCIENCE

Python: plotting examples

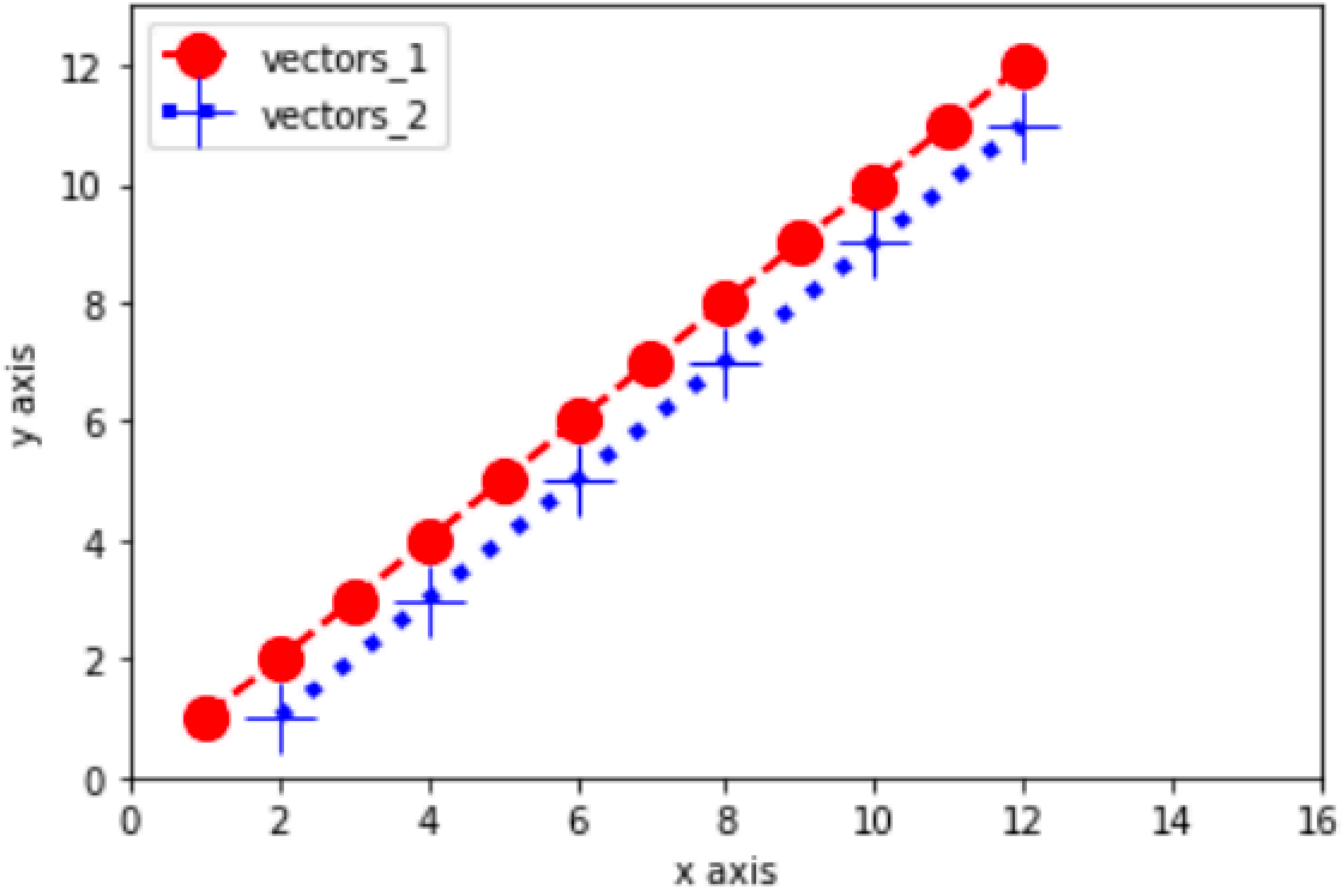


```python
@author: milenavalentini
"""


import numpy as np
import matplotlib.pyplot as mpl

x_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
y_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])

x_vector_2 = np.array([2,4,6,8,10,12])
y_vector_2 = np.array([1,3,5,7,9,11])

mpl.plot(x_vector_1, y_vector_1, color='red', marker='o', linestyle='dashed',
     linewidth=2, markersize=12, label='vectors_1')

mpl.plot(x_vector_2, y_vector_2, color='blue', marker='+', linestyle='dotted',
     linewidth=4, markersize=20, label='vectors_2')

mpl.xlim(0,16)
mpl.ylim(0,13)

mpl.xlabel(‘x axis’)
mpl.ylabel(‘y axis’)

mpl.legend()

mpl.show()
mpl.savefig('a_first_plot.png')
```

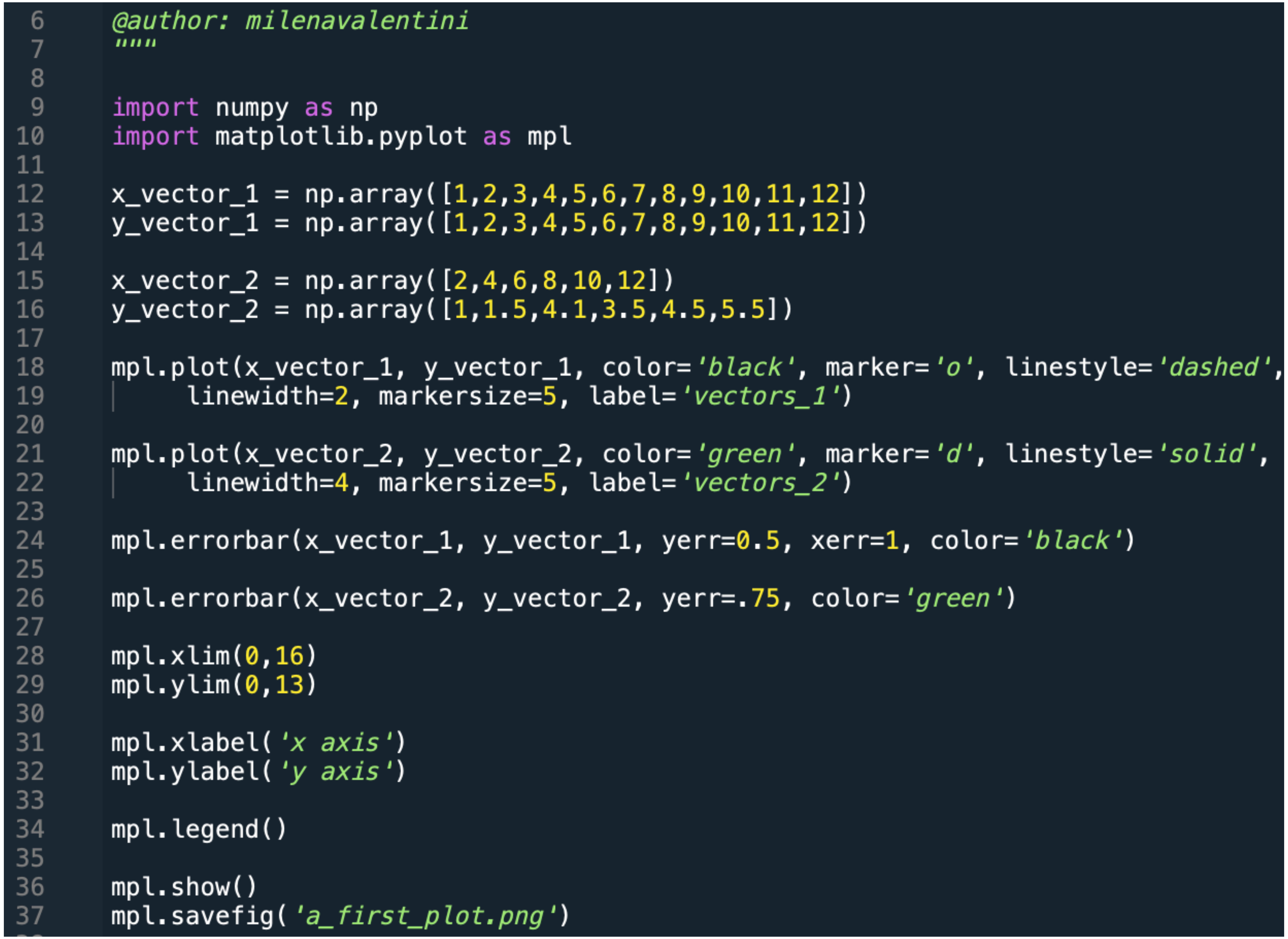
Python: plotting examples



```
@author: milenavalentini
"""

import numpy as np
import matplotlib.pyplot as mpl

x_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
y_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])

x_vector_2 = np.array([2,4,6,8,10,12])
y_vector_2 = np.array([1,3,5,7,9,11])

mpl.plot(x_vector_1, y_vector_1, color='red', marker='o', linestyle='dashed',
     linewidth=2, markersize=12, label='vectors_1')

mpl.plot(x_vector_2, y_vector_2, color='blue', marker='+', linestyle='dotted',
     linewidth=4, markersize=20, label='vectors_2')

mpl.xlim(0,16)
mpl.ylim(0,13)

mpl.xlabel('x axis')
mpl.ylabel('y axis')

mpl.legend()

mpl.show()
mpl.savefig('a_first_plot.png')
```

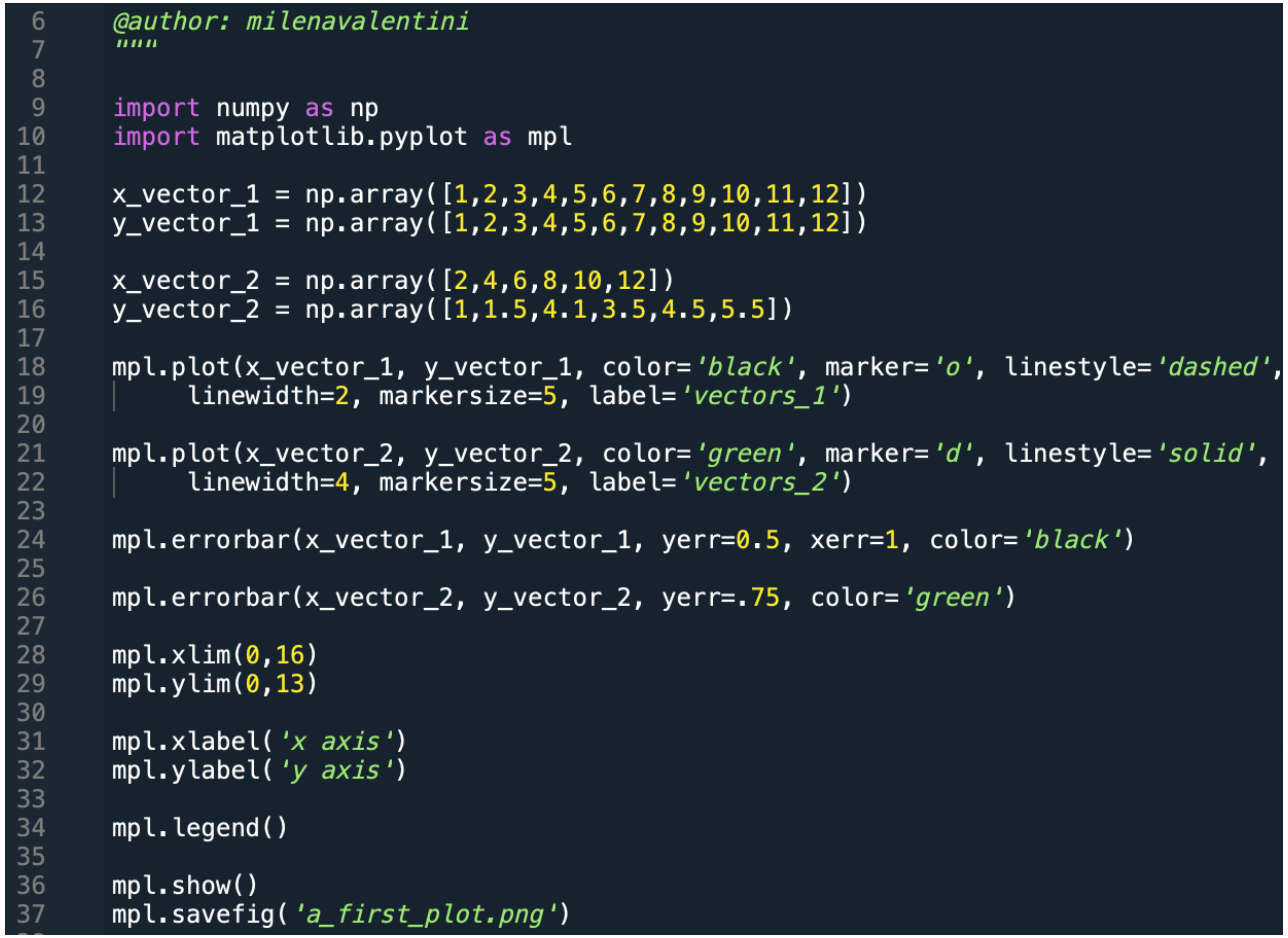


Python: plotting examples

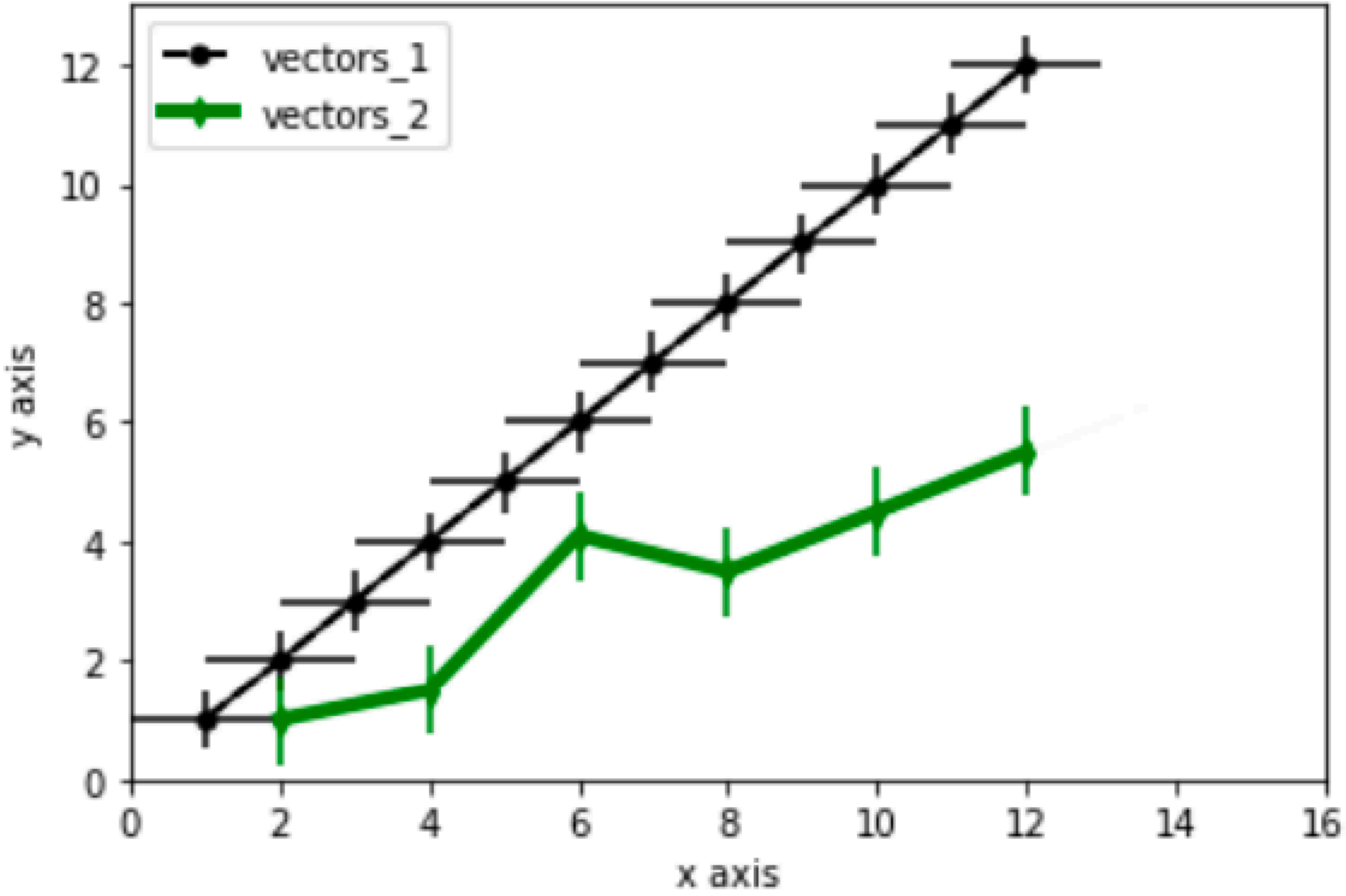


```
@author: milenavalentini
"""

import numpy as np
import matplotlib.pyplot as mpl

x_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
y_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])

x_vector_2 = np.array([2,4,6,8,10,12])
y_vector_2 = np.array([1,1.5,4.1,3.5,4.5,5.5])

mpl.plot(x_vector_1, y_vector_1, color='black', marker='o', linestyle='dashed',
     linewidth=2, markersize=5, label='vectors_1')

mpl.plot(x_vector_2, y_vector_2, color='green', marker='d', linestyle='solid',
     linewidth=4, markersize=5, label='vectors_2')

mpl.errorbar(x_vector_1, y_vector_1, yerr=0.5, xerr=1, color='black')

mpl.errorbar(x_vector_2, y_vector_2, yerr=.75, color='green')

mpl.xlim(0,16)
mpl.ylim(0,13)

mpl.xlabel('x axis')
mpl.ylabel('y axis')

mpl.legend()

mpl.show()
mpl.savefig('a_first_plot.png')
```

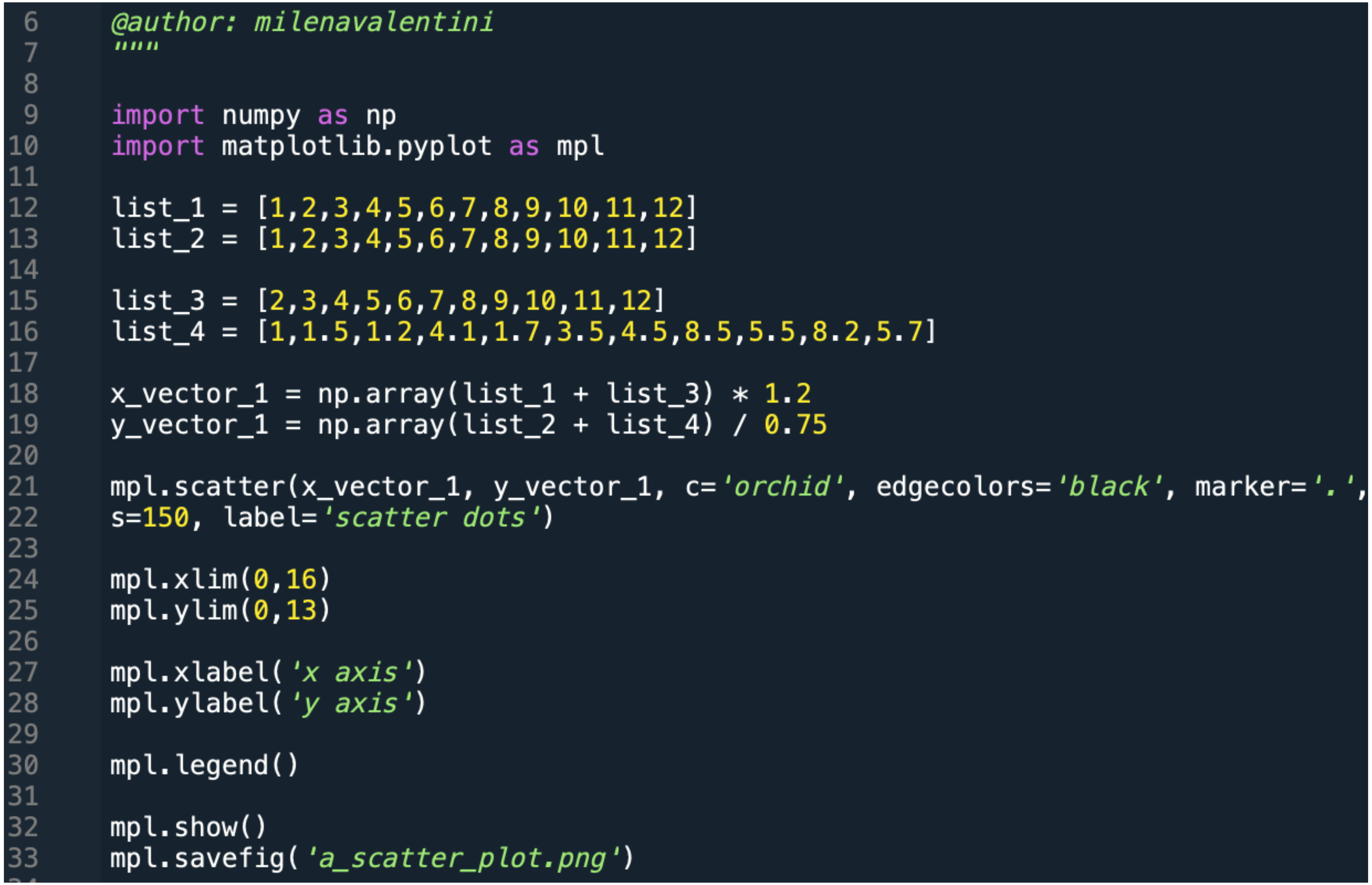
Python: plotting examples



```python
@author: milenavalentini
"""

import numpy as np
import matplotlib.pyplot as mpl

x_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
y_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])

x_vector_2 = np.array([2,4,6,8,10,12])
y_vector_2 = np.array([1,1.5,4.1,3.5,4.5,5.5])

mpl.plot(x_vector_1, y_vector_1, color='black', marker='o', linestyle='dashed',
     linewidth=2, markersize=5, label='vectors_1')

mpl.plot(x_vector_2, y_vector_2, color='green', marker='d', linestyle='solid',
     linewidth=4, markersize=5, label='vectors_2')

mpl.errorbar(x_vector_1, y_vector_1, yerr=0.5, xerr=1, color='black')

mpl.errorbar(x_vector_2, y_vector_2, yerr=.75, color='green')

mpl.xlim(0,16)
mpl.ylim(0,13)

mpl.xlabel('x axis')
mpl.ylabel('y axis')

mpl.legend()

mpl.show()
mpl.savefig('a_first_plot.png')
```

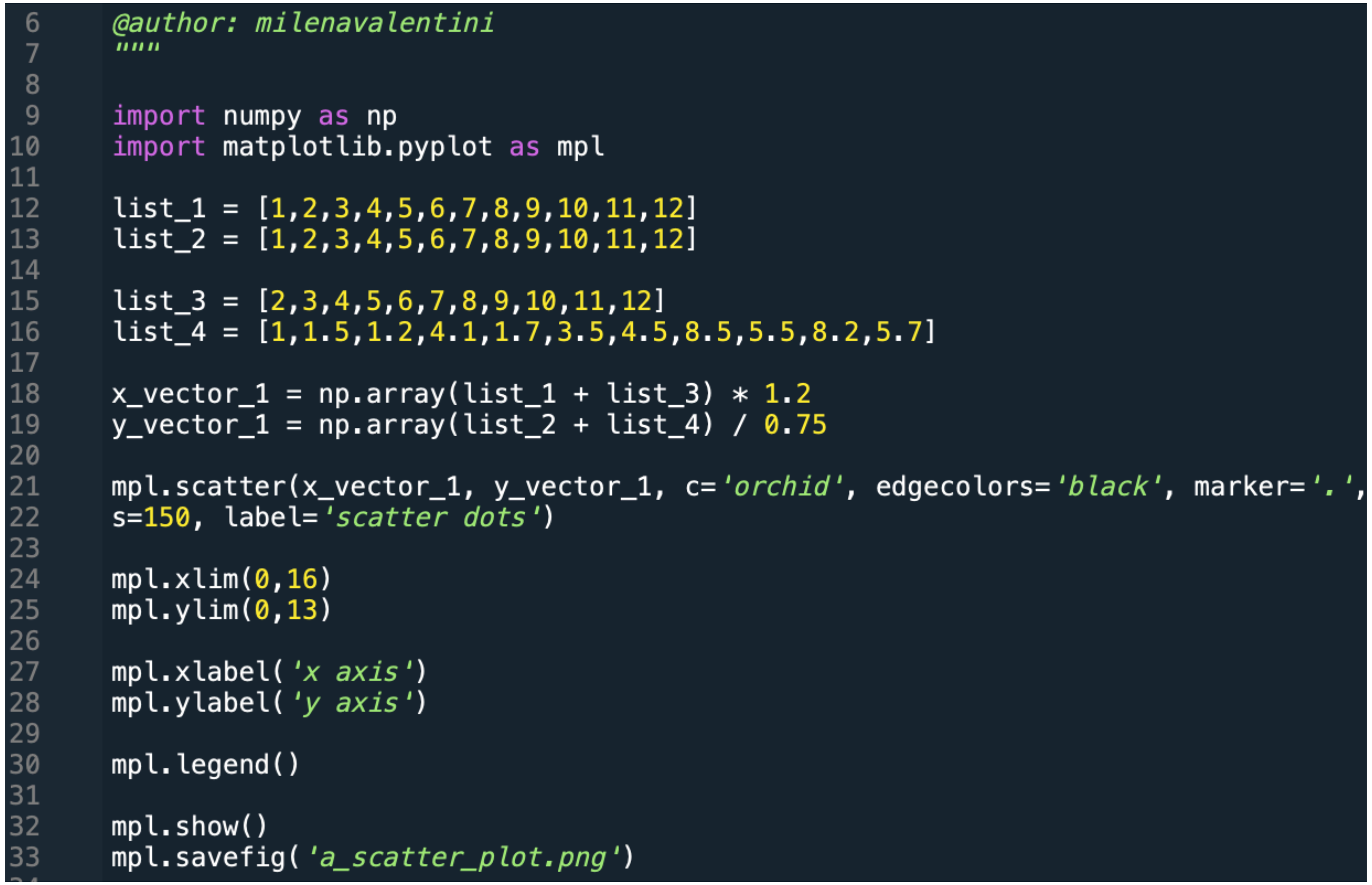


Python: plotting examples

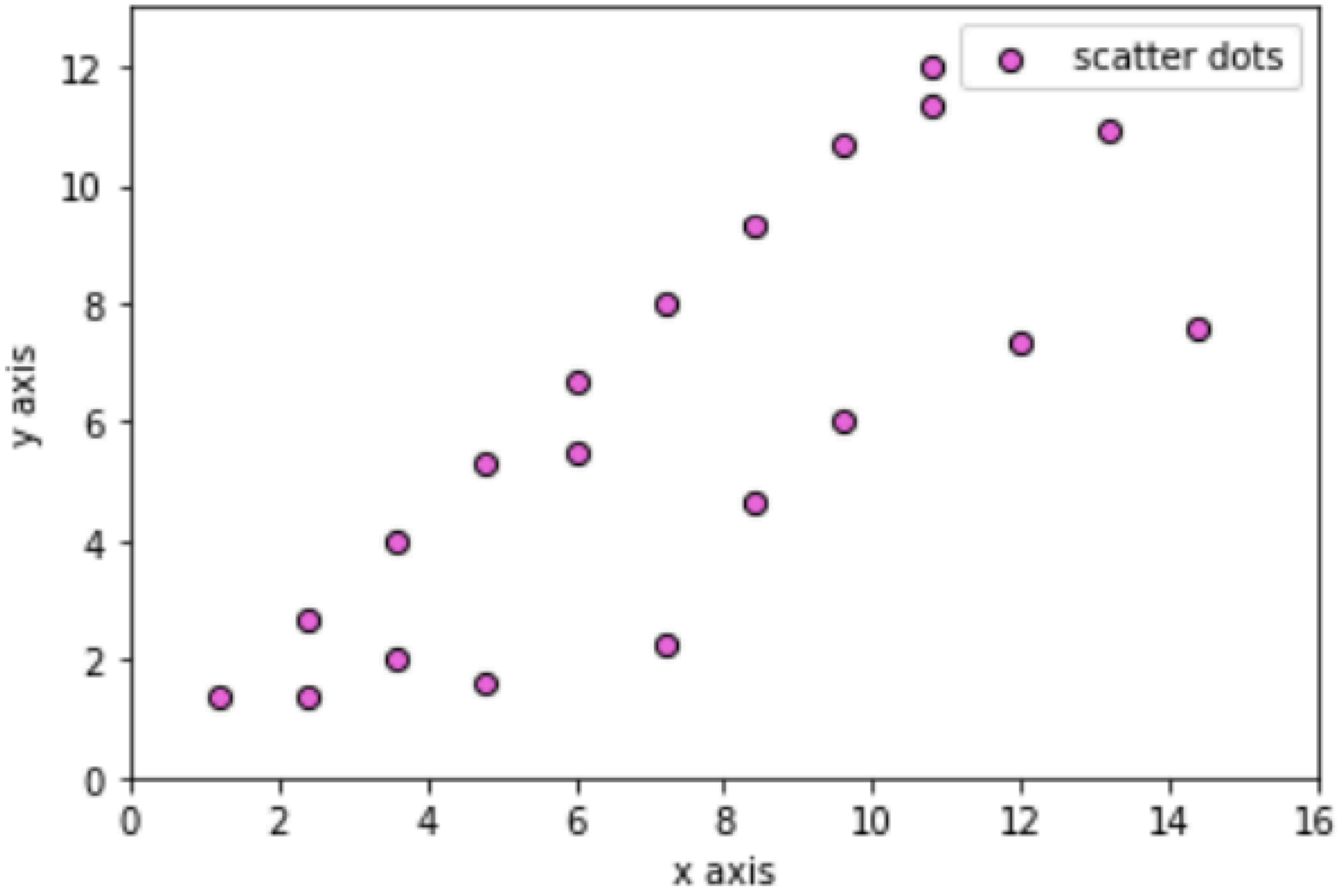


```
@author: milenavalentini
"""


import numpy as np
import matplotlib.pyplot as mpl

list_1 = [1,2,3,4,5,6,7,8,9,10,11,12]
list_2 = [1,2,3,4,5,6,7,8,9,10,11,12]

list_3 = [2,3,4,5,6,7,8,9,10,11,12]
list_4 = [1,1.5,1.2,4.1,1.7,3.5,4.5,8.5,5.5,8.2,5.7]

x_vector_1 = np.array(list_1 + list_3) * 1.2
y_vector_1 = np.array(list_2 + list_4) / 0.75

mpl.scatter(x_vector_1, y_vector_1, c='orchid', edgecolors='black', marker='.',
s=150, label='scatter dots')

mpl.xlim(0,16)
mpl.ylim(0,13)

mpl.xlabel('x axis')
mpl.ylabel('y axis')

mpl.legend()

mpl.show()
mpl.savefig('a_scatter_plot.png')
```

Python: plotting examples



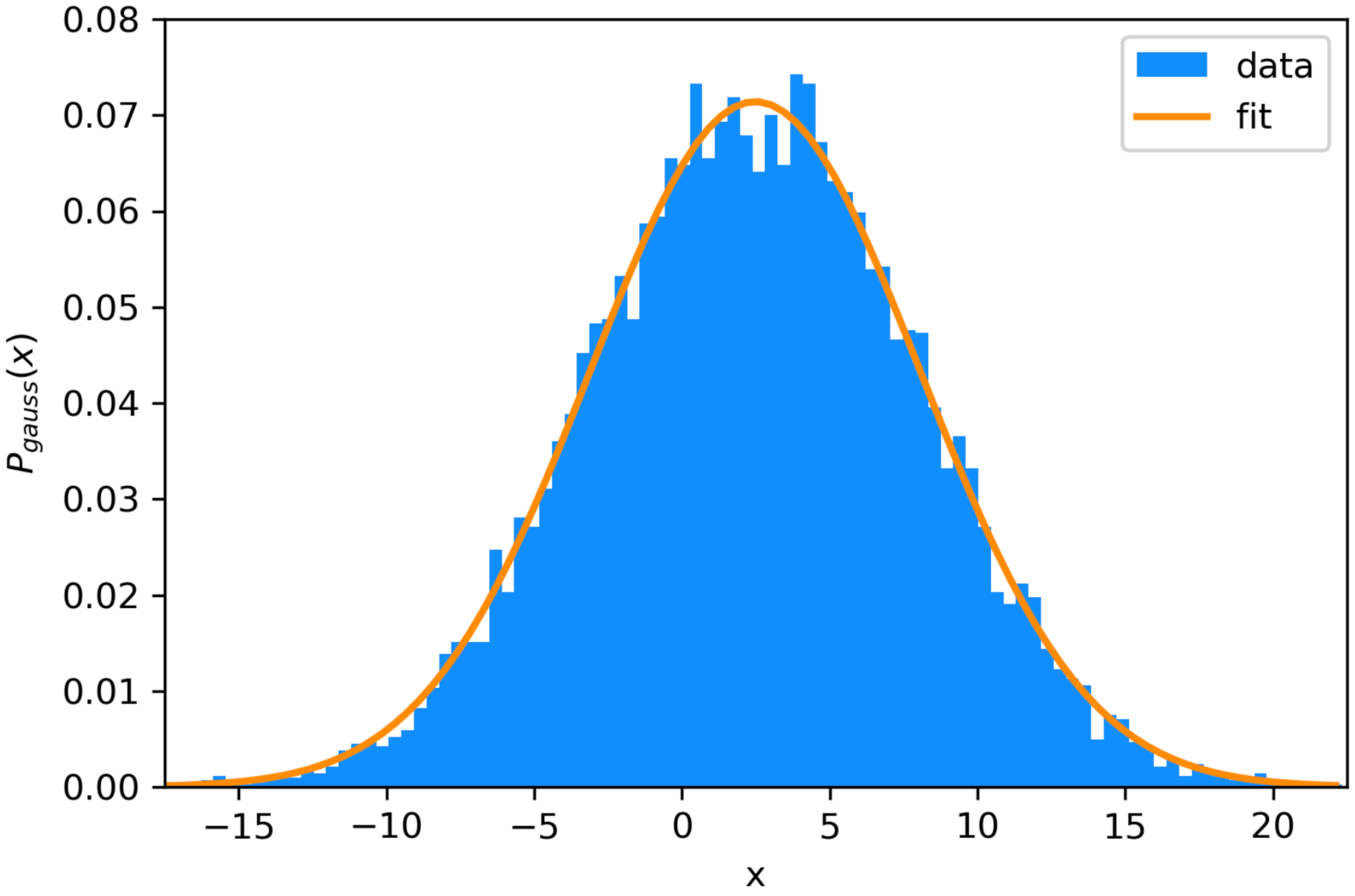
```
@author: milenavalentini
"""

import numpy as np
import matplotlib.pyplot as mpl

list_1 = [1,2,3,4,5,6,7,8,9,10,11,12]
list_2 = [1,2,3,4,5,6,7,8,9,10,11,12]

list_3 = [2,3,4,5,6,7,8,9,10,11,12]
list_4 = [1,1.5,1.2,4.1,1.7,3.5,4.5,8.5,5.5,8.2,5.7]

x_vector_1 = np.array(list_1 + list_3) * 1.2
y_vector_1 = np.array(list_2 + list_4) / 0.75

mpl.scatter(x_vector_1, y_vector_1, c='orchid', edgecolors='black', marker='.',
s=150, label='scatter dots')

mpl.xlim(0,16)
mpl.ylim(0,13)

mpl.xlabel('x axis')
mpl.ylabel('y axis')

mpl.legend()

mpl.show()
mpl.savefig('a_scatter_plot.png')
```

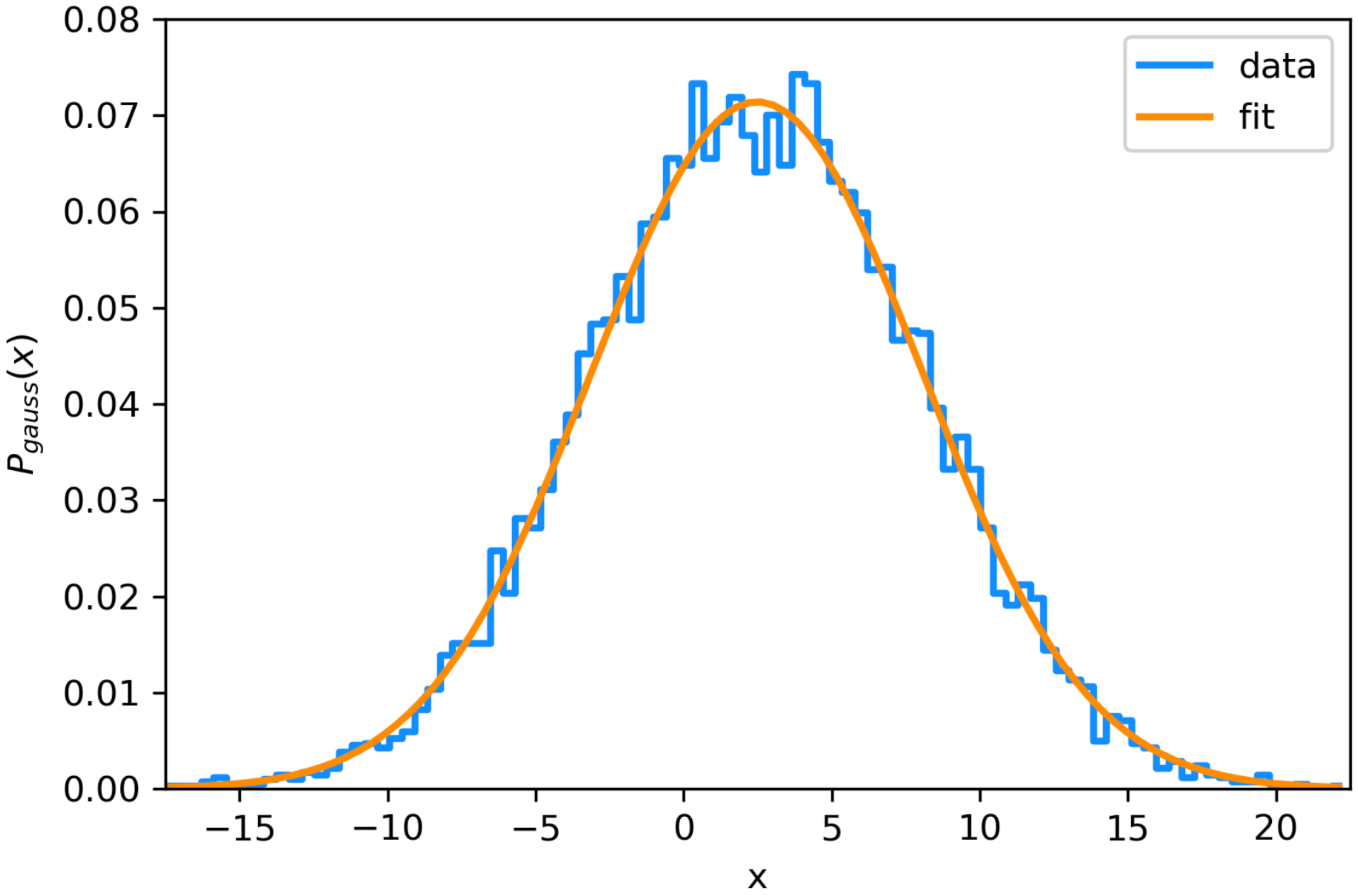


Python: plotting examples

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as sopt

#We generate random data, drawn from a gaussian centered at 2.5, standard deviation of 5.5
original_mu = 2.5
original_sigma = 5.5
my_data = np.random.normal(original_mu, original_sigma, 10000)

#We create an histogram of these data
#We do not use weights, but we ask for the probability density function to be returned. The integral of this
curve will be 1.
#We use 100 bins, we could have defined bins ourselves
gauss_hist, bin_edges = np.histogram(my_data, bins = 100, density = True)

…

#Fit curve to data using scipy.optimize
#We need to define a function to use for the fit. In our case, we fit a gaussian
#I place the function here for example, but this should really go at the top of the script
#This function returns the probability density of a Gaussian curve, i.e. consistent with our 'observed'
values (our histogram)
def gauss_for_fit(xvals, mu, sigma):

    p_of_x = (1./np.sqrt(2*np.pi*sigma**2))*np.exp(-(((xvals-mu)**2)/(2*sigma**2)))

    return p_of_x

#We perform the fit: scipy.optimize is only one of the many methods to do this!
fit_params, fit_covariance_matrix = sopt.curve_fit(gauss_for_fit, bin_centers, gauss_hist)
bestfit_mu, bestfit_sigma = fit_params

…
```

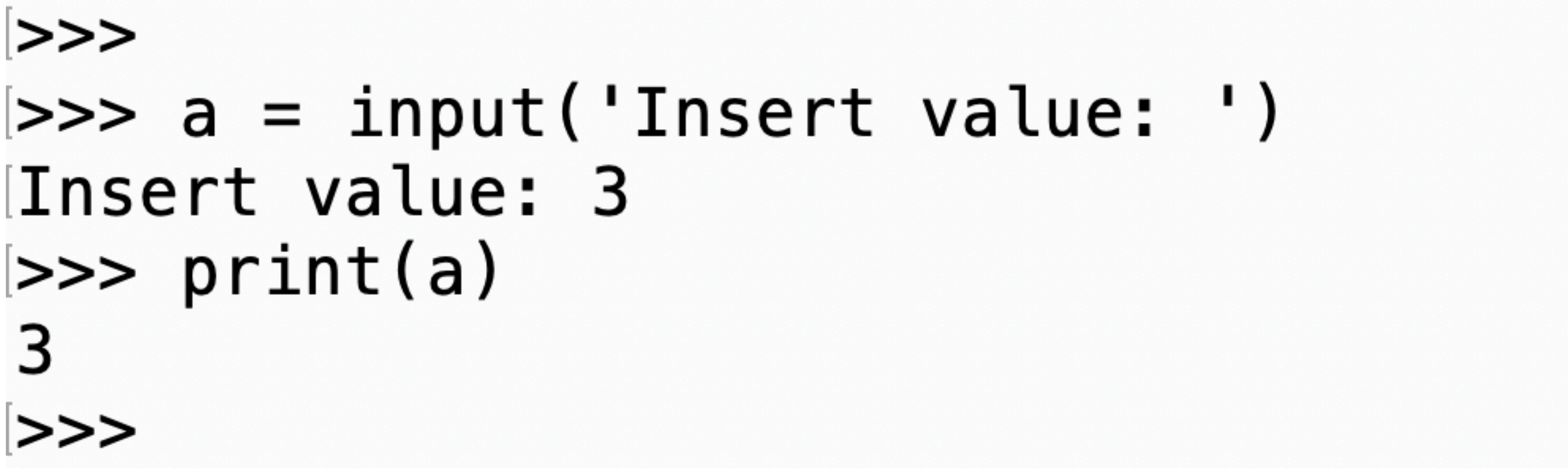
Python: plotting examples





Python: input/output

Input from the keyboard



```
>>>
>>> a = input('Insert value: ')
Insert value: 3
>>> print(a)
3
>>>
```

Python: input/output

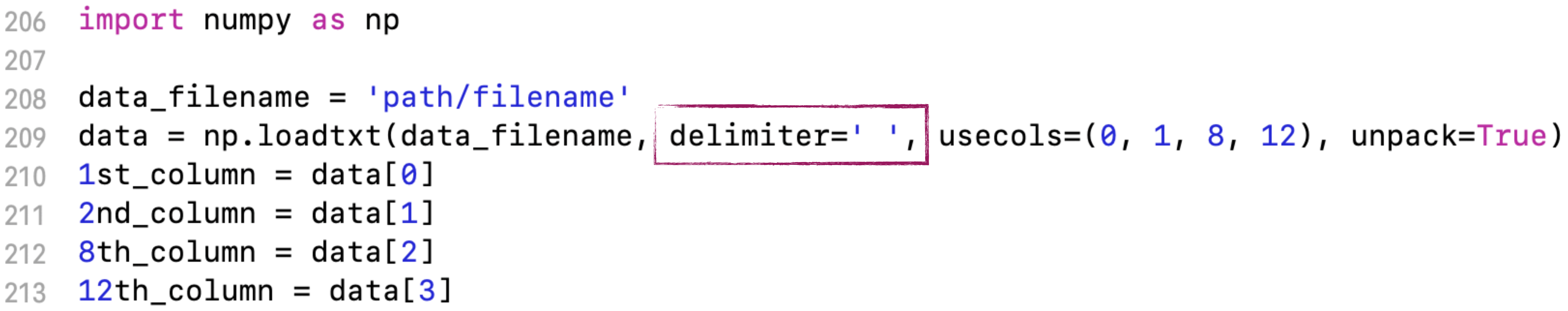
```
#Inizio dal file contenente le proprieta' delle parent particles
filename = ‘my_file.dat'
# load data from the file my_file.dat, whose header is as follows:
# M(M_Sun) rho(gr/cm3) T(K) …
mass = []
density = []
temperature = []
…

with open(filename, 'r') as ppf:
    header = ppf.readline() # skip the first line of the file (header)
    for line in ppf:
        line = line.strip()
        columns = line.split()

        mass.append(float(columns[0]))
        density.append(float(columns[1]))
        temperature.append(float(columns[2]))
        …
#Transform lists into arrays
mass_array = np.array(mass)
density_array = np.array(density)
…
```

How to read from a file

Python: input/output

How to read from a file

with numpy.loadtxt



```python
import numpy as np

data_filename = 'path/filename'
data = np.loadtxt(data_filename, delimiter=' ', usecols=(0, 1, 8, 12), unpack=True)
1st_column = data[0]
2nd_column = data[1]
8th_column = data[2]
12th_column = data[3]
```

and/or other keywords, e.g. comments=‘#’,
skiprows=1

Python: input/output

How to write a file

```
# here I open my file
datafile = "/path/filename.txt"
datafile_id = open(datafile, 'w+')

array1 = …
array2 = …
array3 = x_s[ids]
array4 = …
array5 = …
…
array13 = …
…

data = np.array([array1, array2, array3, array4, array5, …, array13, …])
data = data.T
# here I transpose my data, so to have it in columns

np.savetxt(datafile_id, data, fmt=[‘%d’,’%e’,’%e','%e','%e', …,’%e’, ...], header='ID
                    d_sun (pc)    x_star(pc)    y_star(pc)   z_star(pc)   …   [M/H]    …’)

# here the ascii file is populated

datafile_id.close()
# close the file
```