Lesson 2: Second steps with Python and the NumPy module

Alessio Cardillo

Department of computer science and mathematics (DΣIM),
Universitat Rovira i Virgili, Tarragona, Spain

Python Days
Dept. of physics and astronomy – University of Catania, Catania, Italy

UNIVERSITAT ROVIRA i VIRGILI

Foreword

Outline of the course/lesson

Course

1. ~~Introduction to the basics of the Python language.~~

Outline of the course/lesson

Course

1. ~~Introduction to the basics of the Python language.~~
2. More "*basics*" of Python & basics of NumPy.
3. IPython notebook & Visualization of data (Matplotlib).
4. Data analysis (Pandas).

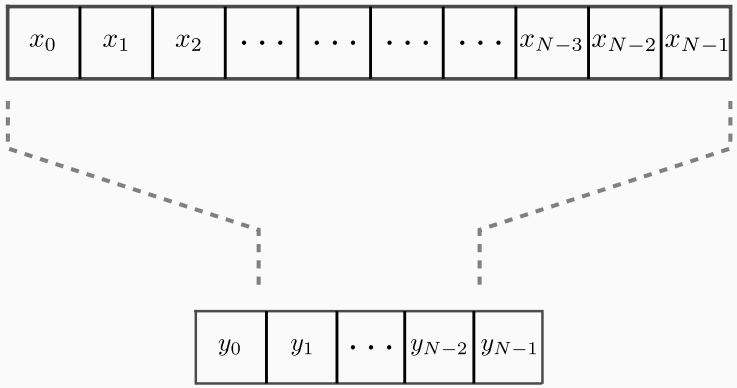
Lesson 2

- Manipulating lists
- File input/output, JSON, functions, & handling errors
- Running third party software in Python
- The NumPy module
- Hands-on session

Manipulating lists

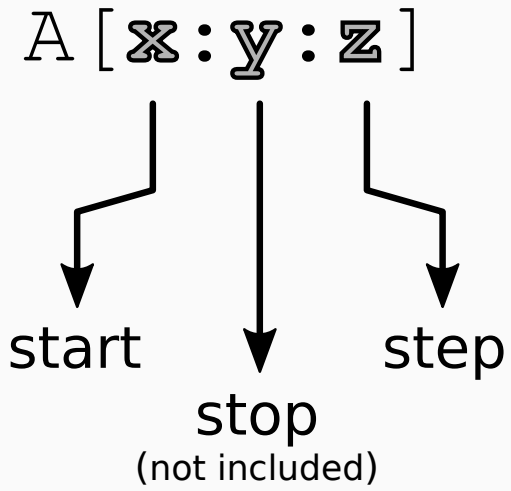
List slicing/comprehension

- By *list slicing* we refer to a way of extracting (and eventually copying) subsets (slices) of list objects (and other ordered iterables).



List slicing/comprehension

- By *list slicing* we refer to a way of extracting (and eventually copying) subsets (slices) of list objects (and other ordered iterables).
- Slicing is based on three stride indicators (even not all together).



Hands-on

```
>>> # defining a list
>>> a = list(range(10))
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> # setting a new list (slicing)
>>> b = a[2:6:2]
>>> b
[2, 4]
```

List slicing/comprehension

- By *list slicing* we refer to a way of extracting (and eventually copying) subsets (slices) of list objects (and other ordered iterables).
- Slicing is based on three stride indicators (even not all together).
- Slicing can be done both forward and backward.

Hands-on

```
>>> # defining a list
>>> a = list(range(10))
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> # striding forward
>>> b = a[2:7:2]
>>> b
[2, 4, 6]
>>> # striding backward
>>> c = a[7:2:-2]
>>> c
[7, 5, 3]
```

List slicing/comprehension

- By *list slicing* we refer to a way of extracting (and eventually copying) subsets (slices) of list objects (and other ordered iterables).
- Slicing is based on three stride indicators (even not all together).
- Slicing can be done both forward and backward.
- Note: When we do assignment through slicing the size of the two objects must be THE SAME.

Hands-on

```
>>> a = list(range(10))
>>> b = list(range(5))
>>> a[2:6:2] = b[1:3:2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: attempt to assign sequence
of size 1 to extended slice of size 2
```

List slicing/comprehension

By list comprehension we indicate any "*compact way*" to perform complex operations on lists.

Hands-on

```
>>> l = range(10)
>>> # list comprehension
>>> a = [x for x in l]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> # a more sophisticated one
>>> b = [x**2 for x in l]
>>> b
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> # you can nest multiple instances
>>> c = [[x**i for x in l[2:5]] for i in range(3)]
>>> c
[[1, 1, 1], [2, 3, 4], [4, 9, 16]]
```

File input/output, JSON, Functions, & Handling errors

Input: command line arguments

Getting input from `stdin`

- The simplest way to pass variables to a Python script via the standard input (`stdin`) involves the `sys.argv` variable. More sophisticated ways involve parsing command line options (for instance via the `getopt` or the `argparse` modules).
- The `sys.argv` variable (available through the `sys` module) works exactly as its homologous in `C` (*e.g.*, `sys.argv[0]` is the name of the program).

Hands-on

```
# basic usage of the sys.argv method
import sys

print('Nr. of args:', len(sys.argv))
print('Args List:', str(sys.argv))
```

Then, run the script as:

```
$ python3 myscript.py var1 3.5 True
```

Note

REMEMBER: Python parses arguments as strings. Hence, you need to convert them to the appropriate type!

- Check the bibliography for references.

Read/Write from/to file

Input file (example-read-io.txt)

```
This file has 3 lines
This is the second line
This is the third line
```

Opening modes	
Code	Mode
'r'	read only
'w'	write (over)
'a'	append
'r+'	read and write

Reading from file

```
>>> # opening the file object
>>> f = open('hands-on/example-read-io.txt', 'r')
>>> # reading the WHOLE file
>>> f.read()
'This file has 3 lines\nThis is the second line\nThis is the third line\n'
>>> # closing the file object
>>> f.close()
```

Tip

Be **very careful** with the `.read()` method because it reads the WHOLE file at once!

Read/Write from/to file

Python

```
>>> # opening the file inside a with statement
>>> with open('hands-on/example-read-io.txt', 'r') as f:
...     f.read()
```

```
>>> # reading one row at a time
>>> with open('hands-on/example-read-io.txt', 'r') as f:
...     for line in f:
...         print(line, end='')
...
This file has 3 lines
This is the second line
This is the third line
```

```
>>> # reading skipping the first two rows
>>> with open('hands-on/example-read-io.txt', 'r') as f:
...     for _ in range(2):
...         a = next(f)
...     for line in f:
...         print(line, end='')
...
This is the third line
```

C

```c
#include <stdio.h>

int main() {

   FILE *f;
   char buff[255];

   f = fopen("example-read-io.txt", "r");
   fscanf(f, "%s", buff);
   printf("%s\n", buff);

   fgets(buff, 255, (FILE*)f);
   printf("%s\n", buff );

   fclose(f);

   return 0;
}
```

- https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files
- https://docs.python.org/3/reference/compound_stmts.html#with

Read/Write from/to file

Writing to file

```
>>> # declaring a matrix
>>> mydata = [[1,2,3],[4,5,6],[7,8,9]]
>>> # opening a file and writing on it
>>> with open('hands-on/example-write-io.txt', 'w') as f:
...     for row in mydata:
...         for col in row:
...             f.write("%d\t" % col)
...         f.write("\n")
...
```

Read/Write from/to file

Good to know

Two very handy modules are the `tarfile` and the `zipfile`. The former makes it possible to read and write `tar` archives, including those using *gzip*, *bz2*, and *lzma* compression.
The latter, instead, can be used to read/write `.zip` files.

Hands-on

```
>>> import tarfile
>>>
>>> # loading the tarfile
>>> fname = 'tarfile-example.tar.gz'
>>> mytar = tarfile.open('hands-on/'+fname, 'r:gz')
>>>
>>> print('Printing the content of the tarfile %s \n' % fname)
>>>
>>> # iterating over the files in the tarfile
>>> for members in mytar.getmembers():
...     print(members)
...
Printing the content of the tarfile tarfile-example.tar.gz

<TarInfo 'archive-1.dat' at 0x7ff6a40ee818>
<TarInfo 'archive-2.dat' at 0x7ff6a40ee688>
<TarInfo 'archive-3.dat' at 0x7ff6a40ee750>
```

- https://docs.python.org/3/library/tarfile.html
- https://docs.python.org/3/library/zipfile.html

The JSON format

What is a JSON?

JSON (*JavaScript Object Notation*) is a data-interchange format easy for humans to read/write, and for machines to parse/generate. It is completely language independent (ideal for data-interchange), but uses conventions that are familiar to programmers of the C-family of languages (*e.g.*, C, C++, C#, Java, JavaScript, Perl), and OF COURSE Python.

- `https://www.json.org/json-en.html`

The JSON format

A JSON is based on two structures

object A collection of name/value pairs (*i.e.*, an associative array).

array An ordered list of values.

The JSON format

A JSON is based on two structures

object A collection of name/value pairs (*i.e.*, an associative array).

array An ordered list of values.

Question

What are the Python equivalents of the **object** and the **array**?

The JSON format

Example of a JSON

```
myjson = {
         'key1': 0.358,
         'key2': [1, 3, 5],
         'key3': True,
         'key4': {
                 'key41': 'mystring',
                 'key42': [[3.5, 'string'], [-7.5, 3]]
                 }
         }
```

Note

JSON **does not support** comments!

The JSON format

The json module

```
import json

# loading a json from file
with open('data.txt', 'r') as infile:
    myjson = json.load(infile)

print(myjson)

# printing a json to file
myjson = {'a': True, 'b': False}

with open('test_file-dump.json', 'w') as outfile:
    json.dump(myjson, outfile)

# pretty printing
with open('test_file-dump.json', 'w') as outfile:
    json.dump(myjson, outfile, indent=4)
```

- https://docs.python.org/3/library/json.html

Functions/modules

Functions are declared using the keyword `def`. There is no need to declare the type of the function or of its arguments (*duck typing*). All functions **must** return something. By default, the return of a function is `None`.

Hands-on

```
>>> # define universal sum
>>> def mysum(v1, v2):
...     return v1+v2
...
>>> mysum(3,5)
8
>>> mysum('Hello',' World!')
'Hello World!'
>>> mysum([1,3,5],['a','b','c'])
[1, 3, 5, 'a', 'b', 'c']
```

- https://docs.python.org/3/glossary.html#term-function

Functions/modules

Functions are declared using the keyword `def`. There is no need to declare the type of the function or of its arguments (*duck typing*). All functions **must** return something. By default, the return of a function is `None`.

Hands-on

```
>>> # factorial
>>> def factorial(n):
...     if n < 2:
...         return 1
...     else:
...         return n*factorial(n-1)
...
>>> for i in [1, 3, 5, 10, 40]:
...     print('fact(%d) = %d' %(i, factorial(i)))
...
fact(1) = 1
fact(3) = 6
fact(5) = 120
fact(10) = 3628800
fact(40) = 815915283247897734345611269596115894272000
```

- https://docs.python.org/3/glossary.html#term-function

Functions/modules

Parameters can be passed either by position or by name (reference). Parameters might have default values. We can also write functions with an arbitrary number of parameters.

Hands-on

```
>>> # define a function
>>> def mysum(v1, v2):
...     return v1+v2
...
>>> # pass by position & name
>>> mysum(1,v2=6)
7
>>> # pass only by name (notice the order)
>>> mysum(v2=6,v1=2)
8
```

- https://stackoverflow.com/questions/986006/how-do-i-pass-a-variable-by-reference

Functions/modules

Parameters can be passed either by position or by name (reference). Parameters might have default values. We can also write functions with an arbitrary number of parameters.

Hands-on

```
>>> # setting default value
>>> def mysum(v1, v2=5):
...     return v1+v2
...
>>> # passing only v1
>>> mysum(2)
7
>>> # passing v1 and v2
>>> mysum(2, 7)
9
>>> # passing v1 by name
>>> mysum(v1=2, 7)
  File "<stdin>", line 1
SyntaxError: positional argument follows
keyword argument
```

- https://stackoverflow.com/questions/986006/how-do-i-pass-a-variable-by-reference

Functions/modules

Parameters can be passed either by position or by name (reference). Parameters might have default values. We can also write functions with an arbitrary number of parameters.

Hands-on

```
>>> # defining a function
>>> def mysum(v1, *args):
...     for elem in args:
...         v1 += elem
...     return v1
...
>>> mysum(1,3)
4
>>> # declaring a list l
>>> l = [1,2,3]
>>> # passing the content of l to the function
>>> mysum(1,*l)
7
```

- https://stackoverflow.com/questions/986006/how-do-i-pass-a-variable-by-reference

Functions/modules

Parameters can be passed either by position or by name (reference). Parameters might have default values. We can also write functions with an arbitrary number of parameters.

Hands-on

```
>>> # function with arbitrary nr. of parameters
>>> def myfunc(a, **kwargs):
...     print(a)
...     print(kwargs)
...     return None # optional
...
>>> # passing parameters by name
>>> myfunc(5, x=1, y=2)
5
{'x': 1, 'y': 2}
>>> # passing parameters as dictionary
>>> c = {'x':1, 'y': 'Hello'}
>>> myfunc(5, **c)
5
{'x': 1, 'y': 'Hello'}
>>> # ERROR multiple definition
>>> c = {'x':1, 'a': 'Hello'}
>>> myfunc(5, **c)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: myfunc() got multiple values for argument 'a'
```

- https://stackoverflow.com/questions/986006/how-do-i-pass-a-variable-by-reference

Functions/modules

We can associate to each function a *doc string* explaining what the function does. We can access the doc string via the command `myfunc.__doc__` or by `help(myfunc)`. It is usually delimited with """ """.

Hands-on

```
>>> def mysum(v1, v2):
...     """This function computes the sum between
...        two elements of arbitrary type."""
...     return v1 + v2
...
>>> print(mysum.__doc__)
This function computes the sum between
two elements of arbitrary type.
>>> help(mysum)
```

- https://www.python.org/dev/peps/pep-0257/

Handling errors

- Error handling, despite being a complex task, ensures the functionality of code.
- Older and modern languages handle errors in completely different ways.
- A modern error handling system should have these characteristics:
  1. Low impact on performances (if errors do not occur).
  2. Small overhead on the code.
  3. Possibility to handle the error in an automated way.

Handling errors

Let's look at "typical" error occurring in scientific computing: the zero division error!

$$\frac{x}{0} = \infty$$

Handling errors

C

```c
#include <stdio.h>
#include <stdlib.h>

main() {

   int dividend = 20;
   int divisor = 0;
   int quotient;

   if( divisor == 0){
      fprintf(stderr, "Division by zero! Exiting...\n");
      exit(-1);
   }

   quotient = dividend / divisor;
   fprintf(stderr, "Value of quotient : %d\n", quotient );

   exit(0);
}
```

Handling errors

Python

```
>>> num = 10.
>>> denom = 0.
>>>
>>> try:
...     print('division = %.2f' %(num/denom))
... except ZeroDivisionError:
...     print('You cannot divide by zero!')
...
```

- We can handle the error by placing the code inside a so-called `try/except` block.
- We can catch either specific classes of errors or all of them.

Handling errors

Hands-on

```
try:
... # tries to execute the code here
...
except Exc0: # captures exceptions of type Exc0
... # do something in case Exc0 occurs
...
except (Exc0, Exc1): # captures exceptions of type Exc0 or Exc1
... # do something in case Exc0 or Exc1 occurs
...
except: # captures EVERY type of exceptions
... # do something in case an exception occurs
...
else: # what should be done in case no exception occurs
... # do something in case of no exception
...
finally: # what must be done regardless of what happens
... # execute this code ALWAYS
...
```

Note

We can generate an error/exception using the command `raise`.

- https://docs.python.org/3/library/exceptions.html

Running third party software

Running external software in Python

It is possible to launch/run “external code” from a Python script.
There are two distinct way of doing it:

Running external software in Python

It is possible to launch/run "external code" from a Python script.
There are two distinct way of doing it:

1. Calling directly functions (*e.g.*, of a shared C library) in the code.

Running external software in Python

It is possible to launch/run "external code" from a Python script.
There are two distinct way of doing it:

1. Calling directly functions (*e.g.*, of a shared C library) in the code.
2. Running externally the code (*e.g.*, a `bash` script).

Running external software in Python

Extension modules

Such modules can do two things that can't be done directly in Python: they can implement new built-in object types, and they can call C library functions and system calls.

the `ctypes` module

It provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python.

- `https://docs.python.org/3/extending/extending.html`
- `https://docs.python.org/3/library/ctypes.html#module-ctypes`

Running external software in Python

Running external code through Python

Let's suppose that we need to interact directly with the operative system, or that we have an old, but well-working, `bash` script that automatically backups all our important files/directories. How can we do such operations through Python?

Answer

We can use either the `os` module, or the `subprocess` one!

- `https://stackabuse.com/executing-shell-commands-with-python/`

Running external software in Python

Our bash script: myscript-bash.sh

```
#!/bin/bash

echo -e "HELLO! I AM A BASH SCRIPT\n"
echo -e "THE VALUE INSERTED IS: $1\n\n"

echo -e "IN THIS DIRECTORY WE HAVE THE FOLLOWING FILES:\n"
ls -lght

# end of script
```

```
alessio@aleurv:~/corso-python-dfa/lectures/lect2/hands-on$ bash myscript-bash.sh 7
HELLO! I AM A BASH SCRIPT

THE VALUE INSERTED IS: 7


IN THIS DIRECTORY WE HAVE THE FOLLOWING FILES:

total 4,0K
-rw-rw-r-- 1 alessio 176 nov 12 15:47 myscript-bash.sh
```

Running external software in Python

Using the os module

```
import os

os.system("bash hands-on/myscript-bash.sh 7")
```

```
HELLO! I AM A BASH SCRIPT

THE VALUE INSERTED IS: 7


IN THIS DIRECTORY WE HAVE THE FOLLOWING FILES:

total 572K
-rw-r--r-- 1 alessio 12K nov 12 16:26 lecture-2.tex
-rw-r--r-- 1 alessio 65K nov 12 16:24 lecture-2.log
-rw-r--r-- 1 alessio 439K nov 12 16:24 lecture-2.pdf
-rw-r--r-- 1 alessio 3,7K nov 12 16:24 lecture-2.nav
-rw-r--r-- 1 alessio 463 nov 12 16:24 lecture-2.toc
-rw-r--r-- 1 alessio 426 nov 12 16:24 lecture-2.out
-rw-r--r-- 1 alessio 9,6K nov 12 16:24 lecture-2.aux
drwxr-xr-x 2 alessio 20K nov 12 16:24 _minted-lecture-2
-rw-r--r-- 1 alessio 0 nov 12 16:24 lecture-2.snm
-rw-r--r-- 1 alessio 40 nov 12 16:24 lecture-2.vrb
drwxrwxr-x 2 alessio 4,0K nov 12 16:15 hands-on
0
```

Running external software in Python

Using the os module

```
import os

os.system("bash hands-on/myscript-bash.sh 7")
```

```
HELLO! I AM A BASH SCRIPT

THE VALUE INSERTED IS: 7


IN THIS DIRECTORY WE HAVE THE FOLLOWING FILES:

total 572K
-rw-r--r-- 1 alessio 12K nov 12 16:26 lecture-2.tex
-rw-r--r-- 1 alessio 65K nov 12 16:24 lecture-2.log
-rw-r--r-- 1 alessio 439K nov 12 16:24 lecture-2.pdf
-rw-r--r-- 1 alessio 3,7K nov 12 16:24 lecture-2.nav
-rw-r--r-- 1 alessio 463 nov 12 16:24 lecture-2.toc
-rw-r--r-- 1 alessio 426 nov 12 16:24 lecture-2.out
-rw-r--r-- 1 alessio 9,6K nov 12 16:24 lecture-2.aux
drwxr-xr-x 2 alessio 20K nov 12 16:24 _minted-lecture-2
-rw-r--r-- 1 alessio 0 nov 12 16:24 lecture-2.snm
-rw-r--r-- 1 alessio 40 nov 12 16:24 lecture-2.vrb
drwxrwxr-x 2 alessio 4,0K nov 12 16:15 hands-on
0
```

Using the subprocess module

```
import subprocess as subproc

out_stat = subproc.run(["bash",\
                        "hands-on/myscript-bash.sh",\
                        "7"])

print("\nThe exit code was %d" % out_stat.returncode)
```

SAME OUTPUT AS WITH os MODULE, PLUS:

```
The exit code was 0
```

Redirect of the output

```
import subprocess as subproc

out_stat = subproc.run(["bash",\
                        "hands-on/myscript-bash.sh",\
                        "7"],\
                        stdout=subprocess.DEVNULL)

print("\nThe exit code was %d" % out_stat.returncode)
```

THE OUTPUT BECOMES:

```
The exit code was 0
```

The NumPy module

Numerical Python (NumPy)

What is NumPy?

- Is a Python library used for working (mainly) with arrays.

- https://numpy.org/
- https://numpy.org/doc/stable/index.html

Numerical Python (NumPy)

What is NumPy?

- Is a Python library used for working (mainly) with arrays.
- In Python `lists` serve as `arrays`, but they are slow to process. A NumPy `array` is up to 50x faster than lists.

- https://numpy.org/
- https://numpy.org/doc/stable/index.html

Numerical Python (NumPy)

What is NumPy?

- Is a Python library used for working (mainly) with arrays.
- In Python `lists` serve as `arrays`, but they are slow to process. A NumPy `array` is up to 50x faster than lists.
- NumPy arrays (unlike lists) are stored at one continuous place in memory (locality of reference) $\Rightarrow$ very efficient access & manipulation.

- https://numpy.org/
- https://numpy.org/doc/stable/index.html

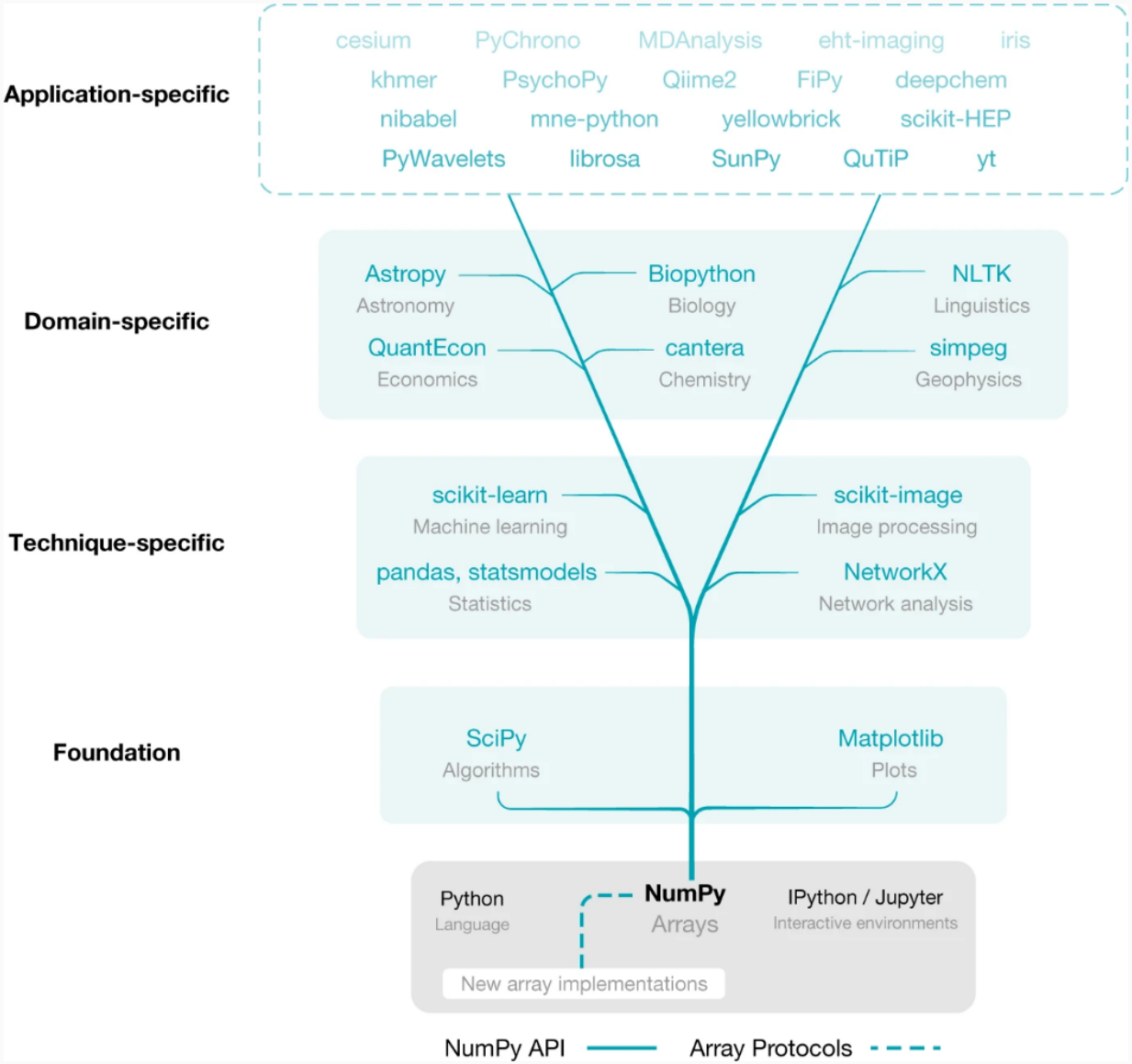
Numerical Python (NumPy)

What is NumPy?

- Is a Python library used for working (mainly) with arrays.
- In Python `lists` serve as `arrays`, but they are slow to process. A NumPy `array` is up to 50x faster than lists.
- NumPy arrays (unlike lists) are stored at one continuous place in memory (locality of reference) $\Rightarrow$ very efficient access & manipulation.
- NumPy is written partially in Python, but most of the parts requiring fast computation are written in C or C++.

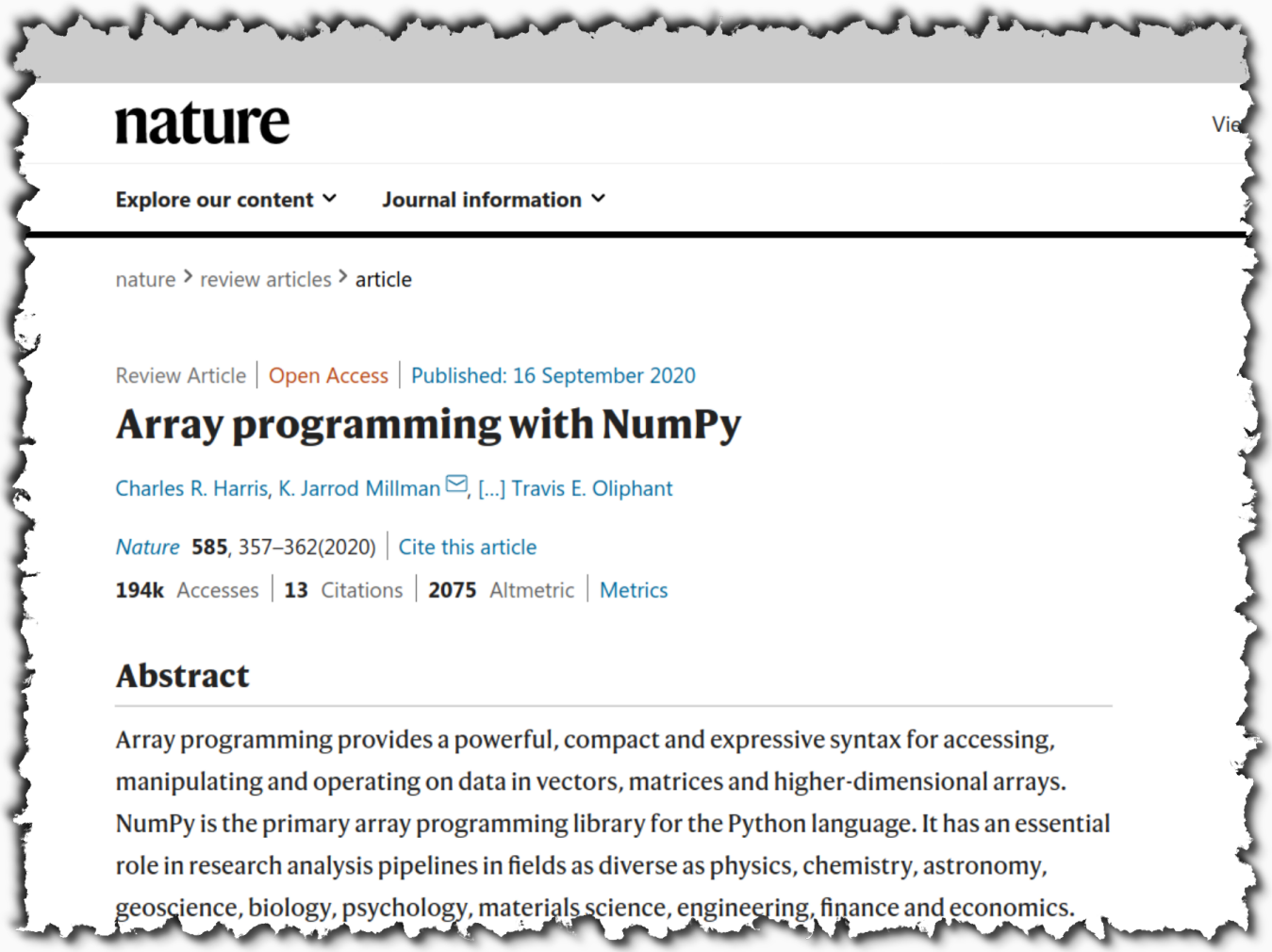
- https://numpy.org/
- https://numpy.org/doc/stable/index.html

Numerical Python (NumPy)



- https://www.nature.com/articles/s41586-020-2649-2

Numerical Python (NumPy)



nature

Explore our content ⌄ Journal information ⌄

nature › review articles › article

Review Article | Open Access | Published: 16 September 2020

Array programming with NumPy

Charles R. Harris, K. Jarrod Millman ✉, [...] Travis E. Oliphant

Nature **585**, 357–362(2020) | Cite this article

194k Accesses | **13** Citations | **2075** Altmetric | Metrics

Abstract

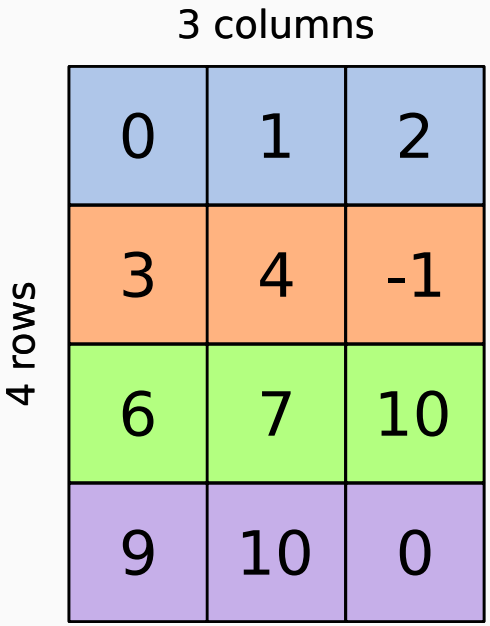
Array programming provides a powerful, compact and expressive syntax for accessing, manipulating and operating on data in vectors, matrices and higher-dimensional arrays. NumPy is the primary array programming library for the Python language. It has an essential role in research analysis pipelines in fields as diverse as physics, chemistry, astronomy, geoscience, biology, psychology, materials science, engineering, finance and economics.

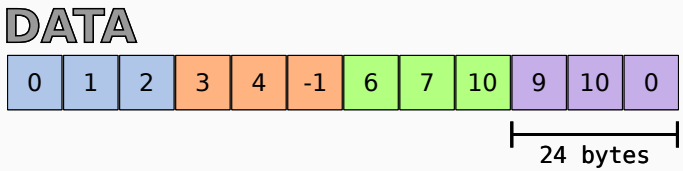
- `https://www.nature.com/articles/s41586-020-2649-2`

Arrays & Main operations

Hands-on

```
>>> import numpy as np
```





DATA TYPE
8 byte int

SHAPE
(4,3) (rows,columns)

STRIDES
(24,8) (jump row,jump col)

Arrays & Main operations

Hands-on

```
>>> import numpy as np
>>> # arrays must contain elements of the same type
>>> a = np.array([1, 3.4, 'test'])
>>> a
array(['1', '3.4', 'test'], dtype='<U32')
>>> a = np.array([1, 3.4, -5.+3j])
>>> a
array([ 1. +0.j,  3.4+0.j, -5. +3.j])
```

Note

Contrary to `lists`, NumPy arrays must contain objects of the **same type**!

Arrays & Main operations

Hands-on

```
>>> import numpy as np
>>> # setting up a list
>>> a = [1, 6, 'hello', -6.5, True]
>>> # converting to numpy array
>>> b = np.asarray(a)
>>> # printing the result
>>> a
[1, 6, 'hello', -6.5, True]
>>> type(a)
<class 'list'>
>>> b
array(['1', '6', 'hello', '-6.5', 'True'], dtype='<U21')
>>> type(b)
<class 'numpy.ndarray'>
```

Good to know

It is possible to convert a list into a NumPy array using the method `.asarray()`. It is also possible to convert a NumPy array to a list using the method `.tolist()` (or, equivalently, `list()`).

Arrays & Main operations

Slicing & masking

Slicing works like for lists, but we can apply it to multiple dimensions.

```
a[:2,::2]
```

```
a[:,1]
```

```
a[2:,2:]
```

0	1	2	3
3	4	-1	0
6	7	10	9
9	10	0	4

Slicing & masking

```
>>> # creating a 4x4 matrix
>>> a = np.array([[0,1,2,3],[3,4,-1,0],\
...               [6,7,10,9],[9,10,0,4]])
>>> a
array([[ 0,  1,  2,  3],
       [ 3,  4, -1,  0],
       [ 6,  7, 10,  9],
       [ 9, 10,  0,  4]])
>>> # returning only elements > 5
>>> a[a>5]
array([ 6,  7, 10,  9,  9, 10])
```

a[a>5]

0	1	2	3
3	4	-1	0
6	7	10	9
9	10	0	4

Arrays & Main operations

Vectorization

```
>>> a = np.array([[0,1],[3,4],[6,7],[9,10]])
>>> b = np.array([[1,1],[1,1],[1,1],[1,1]])
>>> c = a + b
>>> c
array([[ 1,  2],
       [ 4,  5],
       [ 7,  8],
       [10, 11]])
```

0	1
3	4
6	7
9	10

+

1	1
1	1
1	1
1	1

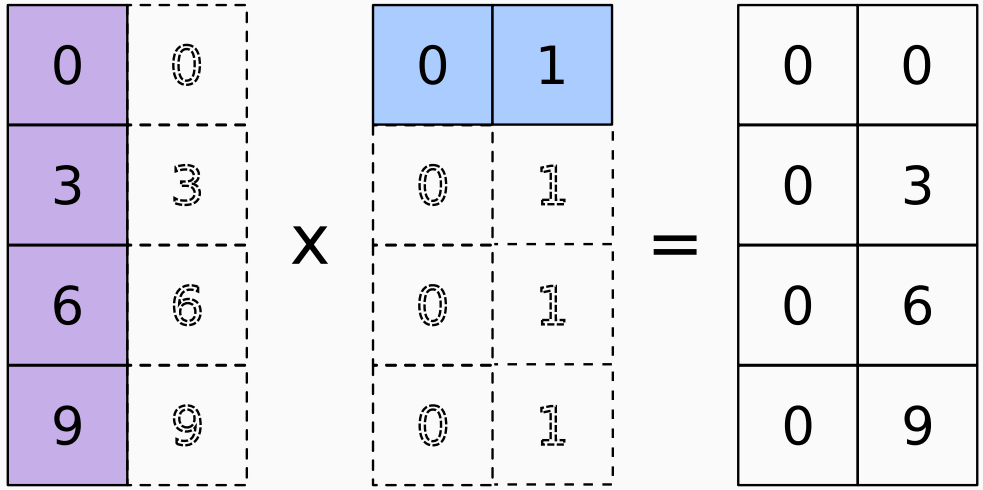
=

1	2
4	5
7	8
10	11

Arrays & Main operations

Broadcasting

```
>>> # to have a column vector
>>> # I have to set it in this way
>>> a = np.array([[0],[3],[6],[9]])
>>> b = np.array([0,1])
>>> c = a * b
>>> c
array([[0, 0],
       [0, 3],
       [0, 6],
       [0, 9]])
```



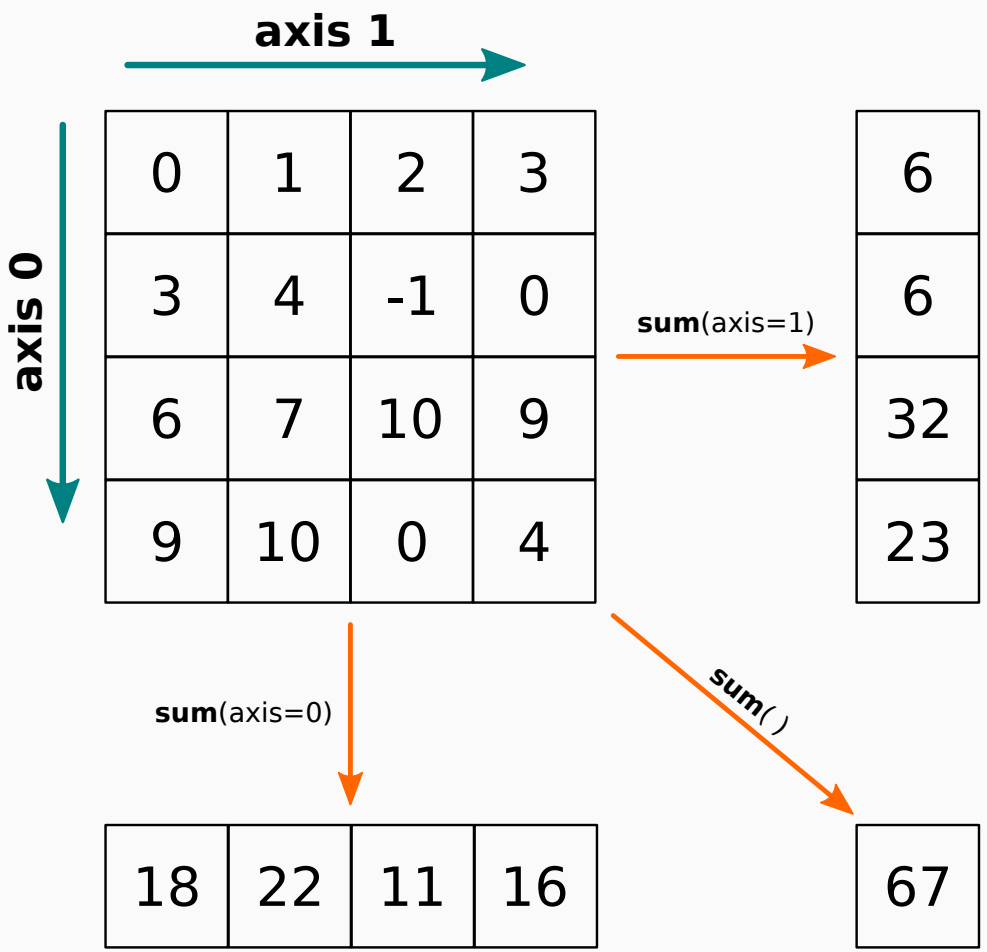
Note

There are more pythonic ways to *transpose* a row array into a column one. For instance:

```
>>> a = np.array([0,3,6,9])
>>> a = a[:, np.newaxis]
array([[0],
       [3],
       [6],
       [9]])
```

Reduction

```
>>> # define a matrix
>>> a = np.array([[0,1,2,3],[3,4,-1,0],\
...               [6,7,10,9],[9,10,0,4]])
>>> a
array([[ 0,  1,  2,  3],
       [ 3,  4, -1,  0],
       [ 6,  7, 10,  9],
       [ 9, 10,  0,  4]])
>>> # sum over axis 0
>>> s1 = a.sum(axis=0)
>>> # sum over axis 1
>>> s2 = a.sum(axis=1)
>>> # sum over axis 0 first and 1 then
>>> s3 = a.sum()
>>> # displaying results
>>> s1
array([18, 22, 11, 16])
>>> s2
array([ 6,  6, 32, 23])
>>> s3
67
```



Arrays & Main operations

Evaluating mathematical expressions with NumPy is quite easy!

Mean square error

$$\langle \varepsilon^2 \rangle = \frac{1}{N} \sum_{i=1}^{N} (X_i^\star - X_i)^2$$

$N \rightarrow$ Number of measures.

$X^\star \rightarrow$ Expected value.

$X \rightarrow$ Measure.

Arrays & Main operations

C

```c
#include <stdlib.h>
#include <stdio.h>

float myexpt[4] = {1., 3.5, -6., 0.4};
float myvals[4] = {1.05, 2.8, -4., -0.3};
float dummy, v1, v2, avg_sq_err;

int N = sizeof(myexpt)/sizeof(float); // 4
int i;

int main(){

  dummy = 0.;

  for(i=0; i<N; i++)
  {
    v1 = myexpt[i];
    v2 = myvals[i];
    dummy += (v1-v2)*(v1-v2);
  }
  avg_sq_err = (1./((float) N))*dummy;


  printf("expect \t measures\n");

  for(i=0; i<N; i++)
  {
    printf("%.2f\t%.2f\n", myexpt[i], myvals[i]);
  }

  printf("mean quadratic error = %.4f\n", avg_sq_err);

  return 0;

}
```

Arrays & Main operations

Python

```
1    myexpt = [1., 3.5, -6., 0.4]
2    myvals = [1.05, 2.8, -4., -0.3]
3
4    N = len(myexpt)
5
6    dummy = 0.
7
8    for i in range(N):
9        v1 = myexpt[i]
10       v2 = myvals[i]
11       dummy += (v1-v2)*(v1-v2)
12
13   avg_sq_err = (1./N)*dummy
14
15   print('expectations = ', myexpt)
16   print('measures = ', myvals)
17   print('mean quadratic error = %.4f'\
18         % avg_sq_err)
```

NumPy

```
1    import numpy as np
2
3    myexpt = np.array([1., 3.5, -6., 0.4])
4    myvals = np.array([1.05, 2.8, -4., -0.3])
5
6    N = np.shape(myexpt)[0]
7
8    avg_sq_err = ((1./N)
9                  * np.sum(np.square(myexpt-myvals)))
10
11   print('expectations = ', myexpt)
12   print('measures = ', myvals)
13   print('mean quadratic error = %.4f'\
14         % avg_sq_err)
```

Useful NumPy's functions

Cheat Sheets

Together with the course materials, there are also the cheat sheets for NumPy and SciPy.

Useful NumPy's functions

shape & reshape

```
>>> import numpy as np
>>> # find the shape of an array
>>> a = np.array([[1,2,3],[4,5,6]])
>>> a.shape
(2, 3)
>>> np.shape(a)
(2, 3)
>>> # reshape of an array
>>> a = np.array([1,2,3,4,5,6])
>>> a.reshape(2,3)
array([[1, 2, 3],
       [4, 5, 6]])
```

Useful NumPy's functions

zeros & ones

```
>>> import numpy as np
>>> # create an array full of zeros
>>> a = np.zeros(5)
>>> a
array([0., 0., 0., 0., 0.])
>>> # create an array full of ones
>>> a = np.ones(5)
>>> a
array([1., 1., 1., 1., 1.])
```

Note

There are two functions called `zeros_like()` and `ones_like()` which generate arrays of zeros and ones with a shape identical to another array (*e.g.*, (3,4)).

Useful NumPy's functions

arange & linspace

Both return sequences of evenly spaced numbers, but they are not the same thing.

```
>>> import numpy as np
>>> # arange(start, stop, step)
>>> # (stop not included)
>>> a = np.arange(0,10,2)
>>> a
array([0, 2, 4, 6, 8])
>>> # linspace(start, stop, nrpoints)
>>> # (stop is included)
>>> a = np.linspace(0,10,2)
>>> a
array([ 0., 10.])
```

Note

There exist two logarithmic counterparts of `linspace()` called `logspace()` and `geomspace()`.

Useful NumPy's functions

mean, std, & var

```
>>> import numpy as np
>>> # generate an array of size 10 filled of random numbers in [0,1)
>>> a = np.random.random(10)
>>> a
array([0.82191943, 0.8119978 , 0.93116978, 0.74907317, 0.35424478,
       0.41143222, 0.1770835 , 0.98988076, 0.30800019, 0.37610524])
>>> # arithmetic mean
>>> np.mean(a)
0.593090686506173
>>> # standard deviation
>>> np.std(a)
0.28068161984733503
>>> # variance
>>> np.var(a)
0.0787821717201239
```

Note

There exist similar functions to account for the presence of NaN values called `nanmean()`, `nanstd()`, and `nanvar()`.

Useful NumPy's functions

histogram

```
>>> import numpy as np
>>> # generate a histogram with three bins
>>> np.histogram([1, 2, 1], bins=[0, 1, 2, 3])
(array([0, 2, 1]), array([0, 1, 2, 3]))
>>> # you can feed multiple arrays to the histogram
>>> np.histogram([[1, 2, 1], [1, 0, 1]], bins=[0,1,2,3])
(array([1, 4, 1]), array([0, 1, 2, 3]))
>>> # compute a "kernel density estimator" (KDE)
>>> hist, bin_edges = np.histogram(np.random.randint(0,4,size=100), bins=np.linspace(0,4,10),\
...                                density=True)
>>> hist
array([0.585 , 0.    , 0.4725, 0.    , 0.63  , 0.    , 0.5625, 0.    , 0.    ])
>>> bin_edges
array([0.        , 0.44444444, 0.88888889, 1.33333333, 1.77777778,
       2.22222222, 2.66666667, 3.11111111, 3.55555556, 4.        ])
>>> hist.sum()
2.25
>>> # computing the "area under the curve"
>>> np.sum(hist * np.diff(bin_edges))
1.0
```

Warning

In previous versions of the NumPy (< 1.6), the normed histogram did not compute well the probability density. Remember: ALWAYS BENCHMARK THE CODE!

Useful NumPy's functions

savetxt

```
>>> a = np.array([[1,2,3,5],[2,4,10,1],\
...               [0,30,5,-1],[16,8,20,20]])
>>> np.savetxt('hands-on/ex_np_savetxt.dat',
...            a, fmt='%.1f', delimiter=';')

alessio@aleurv:$ less ex_np_savetxt.dat

1.0;2.0;3.0;5.0
2.0;4.0;10.0;1.0
0.0;30.0;5.0;-1.0
16.0;8.0;20.0;20.0
```

loadtxt

```
>>> b = np.loadtxt('hands-on/ex_np_savetxt.dat',
...                delimiter=';', usecols=(1,3))
>>> b
array([[ 2.,  5.],
       [ 4.,  1.],
       [30., -1.],
       [ 8., 20.]])
```

Good to know

The `loadtxt()` function is quite versatile as it allows to define **comments**, **delimiters**, **rows to skip**, and **columns to use**. Similarly, the `savetxt()` function allows to define also **formats**, **headers**, and **footers**.

Hands-on Session

Exercise 1

Task

1. Compute $\int_a^b \alpha \sin(x)\, e^{-\beta x}\, dx$ using the **Monte Carlo method**.

   $$I = \int_a^b g(x)\, dx \simeq \langle I_N \rangle .$$

   Being

   $$\langle I_N \rangle \equiv \frac{1}{N} \sum_{i=1}^{N} I_i = \frac{1}{N} \left[(b-a) \sum_{i=1}^{N} g(x_i) \right],$$

   with x_i a random number in the $[a, b]$ range.

2. Print the solution and its accuracy (in absolute value and in %) as a function of the size of the sample, N.

3. Parameters' values: $a = 0$, $b = 2\pi$, $\alpha = 2.5$, $\beta = 1$, and $N \in \{10, 100, 1000, 10000, 100000, 1000000, 10000000\}$.

Exercise 1

Random numbers in $[a, b]$ range

Use the function `numpy.random.uniform()`.

Compute the exact solution

```
from scipy import integrate

val, err = integrate.quad(myfunc, val_a, val_b)
```

Tips

- Leverage the *power* of NumPy arrays!
- Use a function to define the integrand.
- NumPy has built-in $\sin(x)$ (`numpy.sin()`) and $\exp(x)$ (`numpy.exp()`) functions as well as the π constant (`numpy.pi`).

- https://towardsdatascience.com/monte-carlo-integration-in-python-a71a209d277e
- http://people.duke.edu/~ccc14/sta-663-2016/15C_MonteCarloIntegration.html

Exercise 2

Task

1. Compute the **Shannon entropy**, S, of a text using the following relation:

$$S = -\frac{1}{\log_2 N_W} \sum_{i=1}^{N_W} p_i \log_2 p_i \, .$$

Where N_W is the number of distinct words in the text. The probability of extracting word i uniformly at random from the text, p_i is equal to:

$$p_i = \frac{N_i}{L_{\mathrm{TOT}}} \, ,$$

with N_i the number of occurrences of word i and L_{TOT} the total length of the text.

- `https://en.wikipedia.org/wiki/Entropy_(information_theory)`

Exercise 2

Task (continuing)

2. For a given text, compute the following quantities:
   - The number of distinct words, N_W.
   - The total length of the text, L_{TOT}.
   - The entropy of the text, S.
   - The number of times each word appears (stored as a dictionary).
   - The lists of the top 5% most frequent and least frequent words (two distinct lists).
3. Print the results on screen and store them in a JSON file.
4. Repeat points 1-3 for all the text files available.

Exercise 2

Tips

- Use list slicing to remove the file extension from file names.
- Not all the lines in the text are useful. Find a way to understand where the "interesting" part of the text is, and tell Python which rows to parse.
- Try to get rid of the punctuation and other "undesired" characters (*e.g.*, '–' or '\n') from words.
- NumPy has a way to compute the percentile of a set of values.
- Print the JSON using the "pretty print" style.

Split string into "words"

```
fin = open(filename, 'r')
    for line in fin:
        # splitting using space as separator
        words = line.split(" ")
```

Remove punctuation

```
import string

str_translator = str.maketrans('',
                               '',
                               string.punctuation)

line = line.translate(str_translator)
```

Bibliography i

- Homepage of the Python programming language. Available at: `https://www.python.org/`
- HTML.it, *Guida al linguaggio Python*. Available at: `https://www.html.it/guide/guida-python/`
- Homepage of the Stackoverflow website. Available at: `https://stackoverflow.com/`
- Python 3 documentation. Available at: `https://docs.python.org/3/`
- Use of the `sys.argv` variable. Available at: `https://www.pythonforbeginners.com/system/python-sys-argv`

Bibliography ii

- Tutorialspoint – Python command line arguments. Available at: https://www.tutorialspoint.com/python/python_command_line_arguments.htm
- The getopt module. Available at: https://docs.python.org/3/library/getopt.html
- The argparse module. Available at: https://docs.python.org/3/library/argparse.html#module-argparse
- The with statement. Available at: https://docs.python.org/3/reference/compound_stmts.html#with

Bibliography iii

- Reading and writing files. Available at: https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files
- The tarfile module. Available at: https://docs.python.org/3/library/tarfile.html
- The zipfile module. Available at: https://docs.python.org/3/library/zipfile.html
- The JSON format homepage. Available at: https://www.json.org/json-en.html
- The Python JSON module documentation. Available at: https://docs.python.org/3/library/json.html

Bibliography iv

- Functions. Available at: https://docs.python.org/3/glossary.html#term-function
- Stackoverflow. Passing variable by reference. Available at: https://stackoverflow.com/questions/986006/how-do-i-pass-a-variable-by-reference
- Python PEP 257: Docstring conventions. Available at: https://www.python.org/dev/peps/pep-0257/
- List of built-in exceptions. Available at: https://docs.python.org/3/library/exceptions.html
- Extending Python with C or C++. Available at: https://docs.python.org/3/extending/extending.html

Bibliography v

- ctypes A foreign function library for Python. Available at: https://docs.python.org/3/library/ctypes.html#module-ctypes
- Running bash commands in Python. Available at: https://stackabuse.com/executing-shell-commands-with-python/
- Homepage of the NumPy package. Available at: https://numpy.org/
- Documentation of the NumPy package. Available at: https://numpy.org/doc/stable/index.html

Bibliography vi

- C.H. Harris *et al.* Array programming with NumPy. Nature **585**, 357 (2020). Available at: https://www.nature.com/articles/s41586-020-2649-2
- Wikipedia – The Monte Carlo method. Available at: https://en.wikipedia.org/wiki/Monte_Carlo_method
- Numerical evaluation of integrals. Available at: https://towardsdatascience.com/monte-carlo-integration-in-python-a71a209d277e
- Monte Carlo integration in Python. Available at: http://people.duke.edu/~ccc14/sta-663-2016/15C_MonteCarloIntegration.html

Bibliography vii


Wikipedia – Entropy (information theory). Available at: https://en.wikipedia.org/wiki/Entropy_(information_theory)