Master in High Performance Computing

Applied Mathematics: an introduction to Scientific Computing by Numerical Analysis

Lecture 03 - LAB - Introduction to Python

Luca Heltai <luca.heltai@sissa.it>

International School for Advanced Studies (www.sissa.it)
Mathematical Analysis, Modeling, and Applications (math.sissa.it)
Theoretical and Scientific Data Science (datascience.sissa.it)
Master in High Performance Computing (www.mhpc.it)
SISSA mathLab (mathlab.sissa.it)

Before we start

- Install “anaconda” (with jupyter support):
  - https://www.anaconda.com/

  and/or

- Open and activate an account on either of
  - https://cocalc.com/
  - https://colab.research.google.com/

Why Python ?

- Writing readable code is easy
  - Natural syntax to commands
  - Indentation-consciousness forces readability
- Reusing code is easy
  - PYTHONPATH/import are easy to use
- Object-oriented programming is “easy”
  - Finally understand what all the C++/Scheme programmers are talking about!
- Close ties to C
  - NumPy allows fast matrix algebra
  - Can dump time-intensive modules in C easily
- Numerical analysis is super easy :-)

Using Python Interactively

- Directly using python
  - /usr/bin/python on all platforms

- ^D (control-D) exits

```
% python
>>> ^D
%
```

- Comments start with ‘#’

```
>>> 2+2  #Comment on the same line as text
4
>>> 7/3 #Numbers are integers by default
2
>>> x = y = z = 0 #Multiple assigns at once
>>> z
0
```

sissa mathLab innovating with mathematics

The Abdus Salam International Centre for Theoretical Physics

SISSA

Running Python Programs

- In general

```
% python myprogram.py
```

- Can also create executable scripts
  - Make file executable:

    ```
    % chmod +x myprogram.py
    ```

  - The first line of the program tells the OS how to execute it:

    ```
    #!/usr/bin/python
    ```

  - Then you can just type the script name to execute

    ```
    % myprogram.py
    ```

  - or

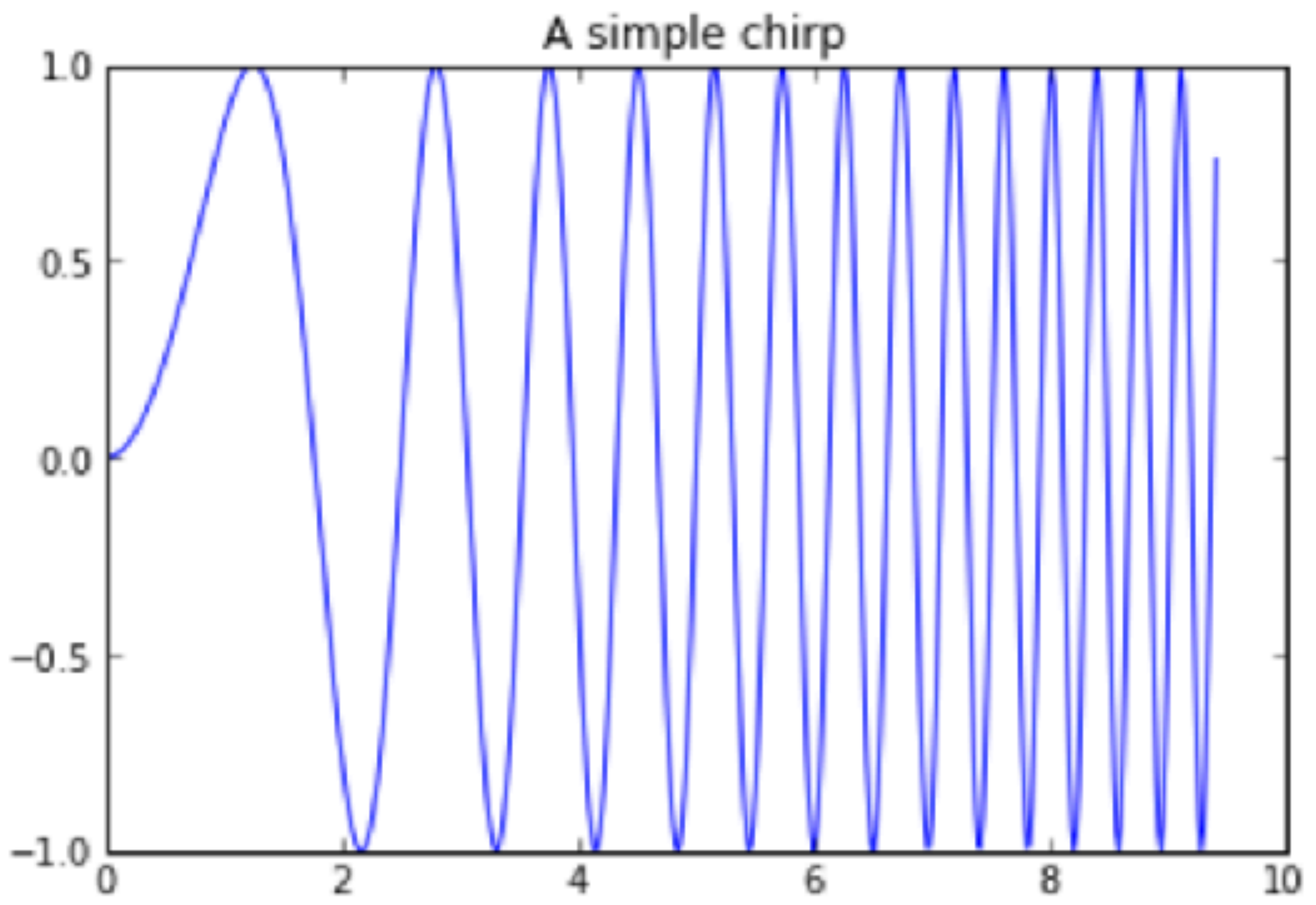
    ```
    % myprogram.py > myoutput.txt
    ```

Python notebooks

- Use “jupyter notebook” for the “next python experience”
    - Indentation
    - Font coloring
    - inline graphs
    - autocompletion
    - similar to mathematica

```
In [3]: from matplotlib.pyplot import *
        x = linspace(0, 3*pi, 500)
        plot(x, sin(x**2))
        title("A simple chirp")

Out[3]: <matplotlib.text.Text at 0x315afd0>
```



Python Data Structures

- Strings

```
MyString = "this is a string"
MyOtherString = 'this is also a string'
NewString = MyString + " " + MyOtherString +
"If you mix quotes it doesn't end the string” +
“”” If you want to go on the next line
with a string, use triple quotes. “””
```

- Integers

```
A = 1    # Normal assignment
b = 3//5 # floor division
b = 3/5  # in python 2 this is the same as the above, i.e., an integer
```

- Floats (double precision floating point numbers)

```
pi = 3.1415927
c = 3/5 # DANGER! standard division in python 3, floor division in python 2
```

Strings

```
In [1]: print('Hello, world!')
```

```
Hello, world!
```

```
In [2]: a = 'Hello, world'
        print(a)
```

```
Hello, world
```

```
In [3]: b = "Hello,"
        c = 'world!'
        d = b + " " + c
        print(d)
```

```
Hello, world!
```

```
In [4]: e = d*2
        print(e)
```

```
Hello, world!Hello, world!
```

```
In [5]: print(e, e[:], e[::], e[0:-1:1])
```

```
Hello, world!Hello, world! Hello, world!Hello, world! Hello, world!Hello,
world! Hello, world!Hello, world
```

```
In [6]: print(e[0:], e[0:-1], e[::2])
```

```
Hello, world!Hello, world! Hello, world!Hello, world Hlo ol!el,wrd
```

sissa mathLab innovating with mathematics

ICTP The Abdus Salam International Centre for Theoretical Physics

SISSA

Numbers

```
In [1]: a = 3
        b = 5

In [2]: b/a # Careful! If you use python 2, this is an int!
Out[2]: 1.6666666666666667

In [3]: a/b # Careful! If you use python 2, this is an int!
Out[3]: 0.6

In [4]: cos(a)

        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        <ipython-input-4-7cac578e0e9a> in <module>()
        ----> 1 cos(a)

        NameError: name 'cos' is not defined

In [5]: from math import cos

In [6]: cos(a)
Out[6]: -0.9899924966004454
```

mathLab innovating with mathematics

ICTP The Abdus Salam International Centre for Theoretical Physics

SISSA

import statement

- import allows a Python script to access additional modules
- Modules
  - sys: stdin, stderr, argv
  - os: system, path
  - string: split
  - re: match compile
  - math: exp, sin, sqrt, pow
  - numpy, scipy, tensorflow, etc…

import statement

```
In [1]: cos(0)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-eddb8697e1ef> in <module>()
----> 1 cos(0)

NameError: name 'cos' is not defined

In [2]: math.cos(0)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-847deae86b34> in <module>()
----> 1 math.cos(0)

NameError: name 'math' is not defined

In [3]: import math

In [4]: math.cos(0)

Out[4]: 1.0

In [5]: cos(0)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-eddb8697e1ef> in <module>()
----> 1 cos(0)

NameError: name 'cos' is not defined

In [6]: from math import cos

In [7]: cos(0)
```

Container Data Structures

- Containers hold collections of other data structures

- Lists
  - Most general sequence of objects
  - Can append, change arbitrary element, etc.

```
a = ['Hi',1,0.234]
```

- Tuples
  - On the fly data containers

```
atom = (atomic_symbol,x,y,z)
```

- Dictionaries
  - Text-indexed container

```
atomic_number = {'Dummy' : 0,'H' : 1,'He' : 2}
      atomic_number['He']# returns 2
```

Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam','eggs',100,1234]
>>> a[0]  # Lists start from 0, as in C
'spam'
>>> a[3]
1234
>>> a[-2] # Negative numbers index from the end
100
>>> a[:2] # ":" denotes a range
['spam','eggs']
```

Lists

```
In [1]: a = ['ciao', 5, 7.8, dir ] # A list

In [2]: type(a)
Out[2]: list

In [3]: type(a[0])
Out[3]: str

In [4]: type(a[1])
Out[4]: int

In [5]: type(a[2])
Out[5]: float

In [6]: type(a[3])
Out[6]: builtin_function_or_method

In [7]: a[3]
Out[7]: <function dir>

In [8]: dir
Out[8]: <function dir>
```

Adding to Lists

```
>>> a + ['bacon']
['spam','eggs',100,1234,'bacon']
>>> a.append('!')
['spam','eggs',100,1234,'!']
>>> 2*a
['spam','eggs',100,1234,'!','spam','eggs',100,
1234,'!']
```

Python functions

Functions are started with def

Function name and arguments

```
def my_function(my_argument):
 line1
 line2
 return some_value
```

Indentation matters!
Determines what is in the function, and when the function ends.

Return value sent back to main routine
value = my_function(5)

Functions

```
In [1]: def twice(argument):
            """
            Return twice the argument.

            A long text of documentation
            that can carry on the following line
            provided that indentation is respected. """
            return argument*2

In [2]: print(twice(2))

4

In [3]: print(twice('ciao'))

ciaociao

In [4]: twice?
```

```
Signature: twice(argument)
Docstring:
Return twice the argument.

A long text of documentation
that can carry on the following line
provided that indentation is respected.
File:      ~/latex/courses/slides-source/<ipython-input-1-2d795dccac28>
Type:      function
```

Flow Control: Looping

- for and while statements can be used to control looping in a program:

```
colors = ['red','green','yellow','blue']
for color in colors:
    print color ' is my favorite color!'
```

- or

```
i = 0
while i < 10:
    print i         # Prints 0, 1, ..., 9
    i = i + 1       # No i++ in Python
```

Flow Control

```
In [1]: a = ['Ciao', 'Hello', 1, 3.5]
```

```
In [4]: for i in a:
            print(i)
```

```
Ciao
Hello
1
3.5
```

```
In [5]: i = 0
        while i < len(a):
            print(a[i])
            i = i+1
```

```
Ciao
Hello
1
3.5
```

For and Range

- range returns a range of numbers

```
>>> range(3)
[0,1,2]
>>> range(1,3)
[1,2]
>>> range(2,5,2)
[2,4]
```

- for and range:

```
for i in range(10):
print i              # Prints 0, 1, ..., 9
```

Python output

- Two functions, print and file.write()
  - print prints to standard output, appends new line

    ```
    print("Hi There!")
    ```
  - file.write prints to file, does not automatically append a new line

    ```
    file.write("Hi There!\n")
    ```

- Formatted output similar to C printf

    ```
    file.write("%s has %d valence electrons\n" % ("C",4))
    ```
  - % operator puts the following tuple into the format characters
  - %s String
  - %d Integer (also %i)
  - %10.4f Float 10 characters wide, with 4 decimal characters

pylab / matplotlib modules

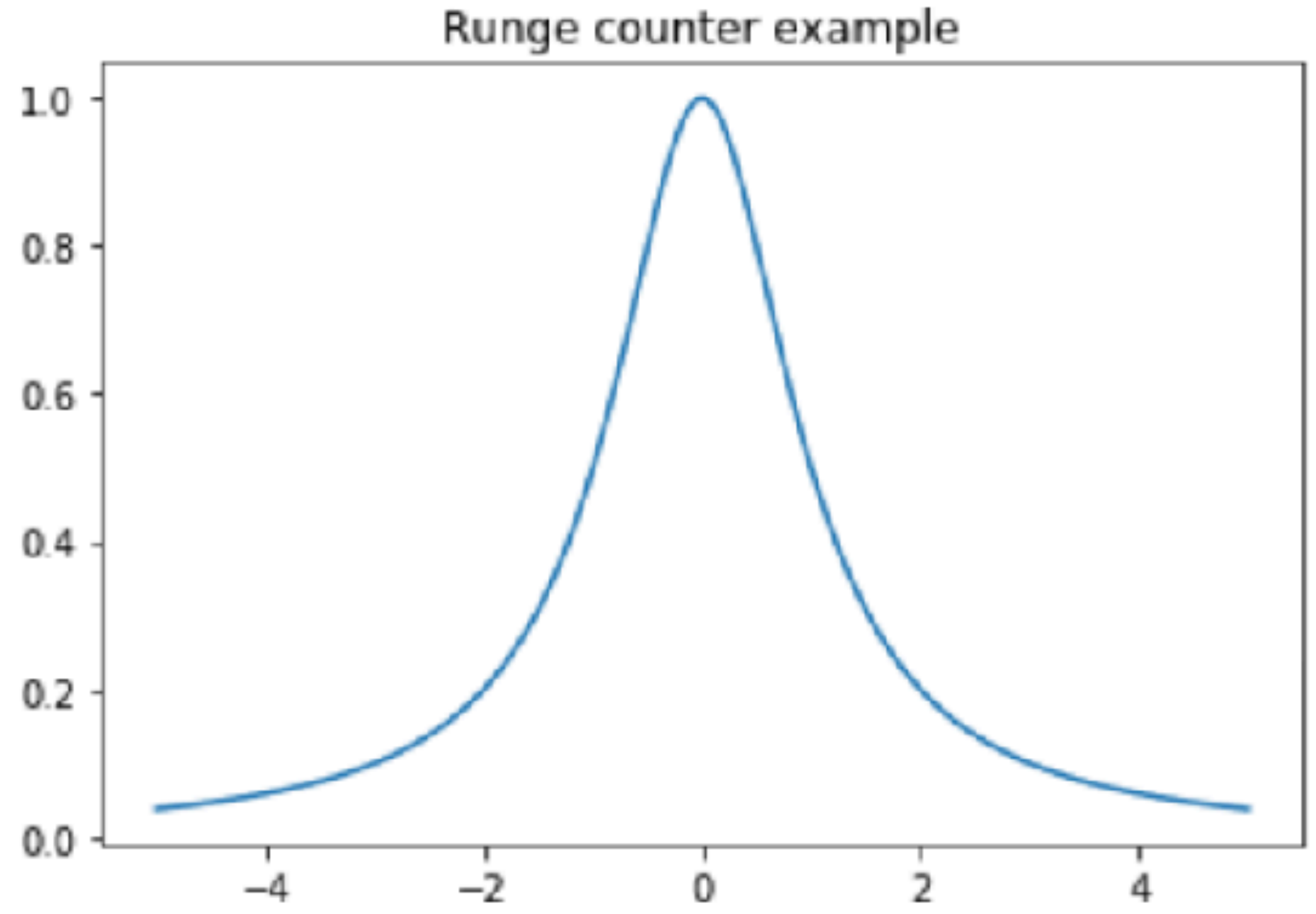
- Exter

In [1]:
```
%matplotlib inline
from numpy import *
from pylab import *

# the above three lines are the same as writing
# %pylab inline
```

In [2]:
```
x = linspace(-5,5,1025)
y = 1/(1+x**2) # Pythonic way to elevate to a power

_ = plot(x,y) # assign to "_" to avoid getting "<matplotlib.text.Text at 0x115a867f0>"
_ = title('Runge counter example')
```



Importing and $PYTHONPATH

- Environmental variable PYTHONPATH
  - Search list of modules to import

```
% setenv PYTHONPATH .:/ul/rpm/python
```

- Import previously written modules:

```
from readers import xyzread
geo = xyzread("h2o.xyz")
for atom in geo:
   symbol, x, y, z = atom # break apart tuple
   print symbol, x, y, z
```

- or

```
import readers
geo = readers.xyzread("h2o.xyz")
for atom in geo:
  symbol, x, y, z = atom # break apart tuple
  print symbol, x, y, z
```

References

- Web Pages
  - http://www.python.org Python Web Site, lots of documentation
  - https://www.cs.put.poznan.pl/csobaniec/software/python/py-qrc.html Python 3 Quick Reference
- Books
  - *Think Python (open source book on python)*
    https://mksaad.files.wordpress.com/2019/04/thinkpython2.pdf