Computer modeling of physical phenomena

Lab I – Python
Why python?

- easy to learn, very clear syntax, readable code
- interpreter and a scripting language
- huge standard libraries, special libraries for scientific computing
- open software, good documentation, easy installation
- expandable (modules from C, C++ or Fortran can be added)
Why python?

Most popular coding languages of 2020

- Python: 31.2%
- C: 6.1%
- C#: 7.4%
- PHP: 2.3%
- Java: 19.6%
- Javascript: 6.5%
- Ruby: 7.1%
Freedom

I learned it last night! Everything is so simple! Hello World is just print "Hello, world!"

I dunno... dynamic typing? Whitespace?
Come join us! Programming is fun again! It's a whole new world up here!
But how are you flying?

I just typed import antigravity
That's it?
... I also sampled everything in the medicine cabinet for comparison.

But I think this is the Python.

xkcd.com
PYTHON

• named after BBC show "Monty Python’s Flying Circus"

• ... and has nothing to do with reptiles
Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus)
Python documentation

• http://docs.python.org  Fast access to Python documentation.
• http://docs.python.org/tutorial/  Python Tutorial
• http://wiki.python.org/moin/
  http://wiki.python.org/moin/PolishLanguage
Python Wiki (also in Polish) – links to more guides
and tutorials

• “A Byte of Python” by C.H. Swaroop; good point to start (free online)

• “Python Tutorial” by Guido; a classic, sometimes too much detailed

  https://bugs.python.org/file47781/Tutorial_EDIT.pdf

• “Dive into Python”; good for consulting when we know something (free online)

• “Learning Python” by M. Lutz; long text if one prefers slower approach (our library)

• The Python Cookbook: collection of examples and useful hints

  https://d.cxcore.net/Python/Python_Cookbook_3rd_Edition.pdf

• "An introduction to Python for scientific computing" by Scott Shell
Running Python

There are different ways to use Python:

• interactive interpreter started from command line
  >>> 2+2
  4

• running a script
  $ python script.py

• from a graphical user interface (GUI) environment; e.g. idle
Python identifiers

• identifier starts with a letter A...Z (or a...z),
• no @, $, and % within identifiers, only letters and digits
• case sensitive

• reserved words

and, exec, not, assert, finally, or, break, for, pass, class, from, print, continue, global, raise, def, if, return, del, import, try, elif, in, while, else, is, with, except, lambda, yield
Lines and indentation

- one line one statement
- blank lines encouraged for readability
- blocks of statements are denoted by indentation
  same indentation within a block

- condition = True
  if condition:
    print("True")
  else:
    print("False")
Comments

- comments follow # sign up to the line end
- string literals are contained within single ’, double " or triple " " " quotation marks
Variables

• the declaration happens automatically when you assign a value to a variable
• standard data types: numbers and strings

>>> a = 10.
>>> type(a)
<class 'float'>
>>> a = 1
>>> type(a)
<class 'int'>
>>> a = '1'
>>> type(a)
<class 'str'>

• numerical types:
  int = integers with unlimited precision
  float = C double precision
  complex = two floats in double precision
Lists

elements in square brackets: ['a', 'b', 'c'],

```python
>>> list = ['abcd', 786, 2.23, 'john', 70.2]
>>> list[0]
'abcd'
>>> list[1:3]
[786, 2.23]
>>> list[3:]
['john', 70.2]
>>> list.append('eve')
>>> list.append('eve')
['abcd', 786, 2.23, 'john', 70.2, 'eve']
```
Tuples

• Sequences similar to lists
  • elements of a tuple and its size can’t be changed
  • tuples can be thought of as read-only lists
  • applications: multiple results from a function, sets of parameters

```python
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')
# Following action is not valid for tuples
# tup1[0] = 100
# Let's create a new tuple as follows
tup3 = tup1 + tup2
print tup3
```
Type conversions

- to convert between built-in types you simply use the type name as a function
  \[ \text{float}(x), \text{str}(n), \text{tuple}(\text{list}) \]
- these functions return a new object representing the converted value
- in the algebraic expressions the type conversion is performed automatically (mixed types converted to a common type)
List of available operators

Python philosophy: operate on all types for which it makes sense

- assignment =
- more assignments +=, *=, /=, ...
- arithmetic: (+, -, *, **, /, //, %)
- logical: (> , < , >= , <= , == , !=)
- and also: (not, and, or)
- bit operations (&, |, ^, ~)

a=5
a+=2
a*=2
a/=7
print (a)
Similar to many other languages

```python
if conditions:
    statements
```

with else-elif construct

```python
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```
While loop

Just an example:

```python
running = True
while running:
    running = statements()
else:
    print "End of while loop"
```
Example 1

for l in "Python":
    print ("current letter ", l)

Example 2

fruits = ["banana", "apple", "mango"]
for f in fruits:
    print ("current fruit ", f)

Example 3

n = 5
for i in range(n):
    print ("current index ", i)
Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusability. We can define function in any place in the code

```python
def welcome():
    print("Hello World!")

welcome()
```

There are default values for the arguments

```python
def func(a, b=5, c=10):
    print("a = ", a, "b = ", b, "c = ", c)
    return a+b+c

func(3)
func(15, c=14)
func(c=50, a=40)
```
Documenting a function

Good practice requires documenting

def maximum(x, y):
    """Prints the bigger of two arguments"""
    if x > y:
        print (x, "is bigger")
    else:
        print (y, "is bigger")

maximum(3, 5)
print
(maximum.__doc__)
help(maximum)
maximum('ala','bob') # our code is very general!
What is a module?

- Module allows you to organize your Python code
- Simply, a module is a file consisting of Python code. A module can define functions, classes, and variables.
- Many most useful tools are just modules: NumPy, SciPy, MatPlotLib
Example on how to use a module

Import a standard system module

```python
import sys

print ('The command line arguments are:"
for i in sys.argv:
    print (i)

print ("The PYTHONPATH is", sys.path, "\n")
```

The code above prints arguments given at a command line

$ python program.py ...

and then the directories searched for Python modules

Warning: never name a program after a standard module name!!!
Our own module

```python
# Filename: mymodule.py
def hi():
    print("Hi, this is me, your module!")

version = "0.1"
# End of mymodule.py
```

```python
# Filename: mymodule_demo.py
import mymodule

mymodule.hi()
mymodule.print ('Version', mymodule.version)
```
Project “0”: a tutorial

there are ≈ 20 slides of a practical tutorial, illustrating all main constructs introduced during the lecture. The tutorial contains also simple examples of plots, special functions from SciPy, and some basic operations with arrays in NumPy. Please follow this guide interactively and then write and run short scripts (as described in the text) on your own.
First task:
  • install Python on your computer
  • make sure you can run it in our lab

Try from the shell prompt
$ python -V
we will work with 3.x version

NUMBERS

$ python

>>> 2+2  # this is a comment
>>> 2/3  # floating point division (integer division in python 2.7)
>>> h = 20.1  # assignment
>>> h        # prints the value
>>> w = 3.2
>>> w*h

>>> w, h = 5, 6.  # multiple assignment

>>> t = w, h     # composed variable
>>> t          # known as TUPLE
>>> t[0]
>>> t[1]
Numbers and strings

```python
>>> s = "12.3"  # is a string
>>> s*2        # well, it works for the string
>>> s*0.5      # but this one doesn’t

>>> float(s)*2 # this is what we mean
>>> int("2343453656") # conversion to integer type

>>> str(12.3)  # ... converting to strings
>>> str(2343453656)

>>> ss = "%.20f" % (2./3) # check what type is ss?
   # %.20f means C formatting

>>> text = "My voltameter shows %.5f V" % (2./3)
   # another example of formatting
```
Complex numbers

```python
>>> 1j # complex unit
>>> (1+2j)*3
>>> (1+2j)/(1+1j)
>>> z = 1.5+.5j
>>> abs(z) # complex module
>>> z.real
>>> z.imag
```
Module *math*

```python
>>> import math  # here we import a module
    # with lots of useful math

>>> math.factorial(70)

>>> math.factorial(70)/math.factorial(69)

>>> ( math.e**(1j*math.pi) ).imag

>>> ( math.e**(1j*math.pi) ).real

>>> help(math)  # checks the documentation
    # use q to quit

>>> help(math.tanh)
```
Module `cmath`

```python
>>> z = (1+2j)*3
>>> math.cos(z)

>>> import cmath as cm  # complex numbers module
    # we give it
    # our own short name
>>> cm.cos(z)
>>> dir(cm)  # list of available functions
>>> cm.polar(z)  # returns polar coordinates
    # radius and angle

>>> rfi = cm.polar(z)
>>> rfi[0]  # an example how to use tuple
>>> rfi[1]
>>> r, fi = cmath.polar(z)  # ... is also fine
```
Strings

```python
>>> word = "Help" + "A"  # concatenation
>>> word

>>> "<" + word*5 + ">" 

>>> word[0:2]  # accessing substrings
>>> "<" + word[0:2]*5 + ">" 

>>> word[:2] + word[3:]
>>> len(word)  # returns length
```
Scientific computing module SciPy

```python
>>> from scipy import constants as const
    # here we import a sub-module
>>> help(const)

>>> const.c    # speed of light
>>> const.h    # Plank's constant

>>> const.h/const.e**2   # quantum of resistance
    # (in Ohms)

>>> from scipy import special as sp

>>> sp.jn(2,0.345)   # Bessel function Jn_2(x)

>>> help(special)   # ... many are available
```
Scripts

Set of commands can be collected into a file *.py and executed simply by running it $python my_example.py

Script “Scientific Hello World” illustrates
— how to initialize variables from the command line
— how to print numbers and text

Create a file hello.py

```python
import sys, math  # import system module

r = float(sys.argv[1])  # reads 1-st argument
    # from the command line

s = math.sin(r)

print ("Hello, World! sin(" + str(r) + ")=" + str(s))
```

and run using  $ python hello.py 2.3
Editor

- use any editor which underlines Python syntax
- or use an integrated environment e.g. idle

$ idle &
File -> New Window  print (2+2)
Run -> Module  (or F5)
Write a simple script as below and run it

```python
# Fibonacci sequence:
# sum of two previous elements gives the new one
a, b = 0, 1  # multiple assigment
while b < 10 :  # !!! remember the colon
    print (b)  # !!! ... and indentation
    a, b = b, a+b

print "Done!"
```

Alternatively print results in a single line

```python
while b < 10:
    print (b, end=" ")  # ... no new line
```
# Create a list consisting of strings
b = ["Mary", "had", "a", "little", "lamb"]

print (b[3])
print (b[3][1:2])
print (len(b))
print (len(b[3]))

for i in range(len(b)):
    print (i, b[i], len(b[i]))

for x in b:
    print (b.index(x), x, len(x))
More list operations

Try to predict what happens in the code, then run it and check

```python
# Same list as in the previous example
b = ["Mary", "had", "a", "little", "lamb"]

print (b*2)
b.append('!')
print (b)
b.reverse()
print (b)
b.reverse()
print (b)
print (b.pop(3))
print (b)
```

`list.pop(i)` - remove the item at the given position in the list, and return it.
Functions

We rewrite out “while loop example” using a function

# Define a simple function

def fib(n=3):
    a, b = 0, 1
    while a < n:
        print (a, end=" ")
        a, b = b, a+b
    print ()

fib(2000)
fib()
Local and global variables

Python understands that local variables do exist only within a function. Run and see what happens.

```
# Filename: func.py

def f_loc(x):
    print ('x is ', x)  # print to check the value
    x = 2              # a new local variable is created
    print ('locally x is ', x)

x = 50
f_loc(x)
print ('x is still', x)
```
Local and global variables

There is a way to change a global variable within the function

# Filename: func.py

def f_glob():
    global x
    print ('x is', x)
    x = 2
    print ('x is now changed to ', x)

x = 50
f_glob()
print ('x in this case is ', x)
NumPy arrays

Run the code below, note that one-line creation of an array is more Python-like than the loop over elements.

```python
import numpy as np # short name of this module

n = 7
v = np.zeros(n) # vector is represented as numpy array

for i in range(n):
    v[i] = i/2.0 # sets elements v_i

print (v)

u = np.arange(-1.0,1.0,0.2) # range of values
print (u)

w = np.linspace(-np.pi,np.pi,6) # also some range, given number of points
print (w)
```
Plotting with Matplotlib

Plotting is easy, check it with the example below

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

x = np.linspace(-np.pi, np.pi, 101)

y1 = np.cos(x)  # Numpy math functions
y2 = np.sin(x)  # operate on whole arrays

plt.plot(x, y1, color='r', linestyle='-', linewidth=2)
# plots with a red line

plt.plot(x, y2, 'g--', lw=1)
# shorter description: thinner green dashed line

plt.show()
```