Computer Modeling of Physical Phenomena
Rebrushing your Python

Jakub Tworzydło

Institut of Theoretical Physics
phone: (022) 5532-919, room 5.19
Jakub.Tworzydlo@fuw.edu.pl

28/02/2022 Pasteura, Warszawa

Plan

Plan

1. Course organization and schedule
2. Introduction
3. Basics of Python programming
   - Variables, data types, operators
   - Loops and conditionals
4. More effective Python
   - Functions
   - Modules

Plan

1. Course organization and schedule
2. Introduction
3. Basics of Python programming
   - Variables, data types, operators
   - Loops and conditionals
4. More effective Python
   - Functions
   - Modules

Plan

1. Course organization and schedule
2. Introduction
3. Basics of Python programming
   - Variables, data types, operators
   - Loops and conditionals
4. More effective Python
   - Functions
   - Modules

Plan

Scheme of our meetings

Lecutre & lab are one unit, presence is required.
Earning credits/points – let's discuss next week.

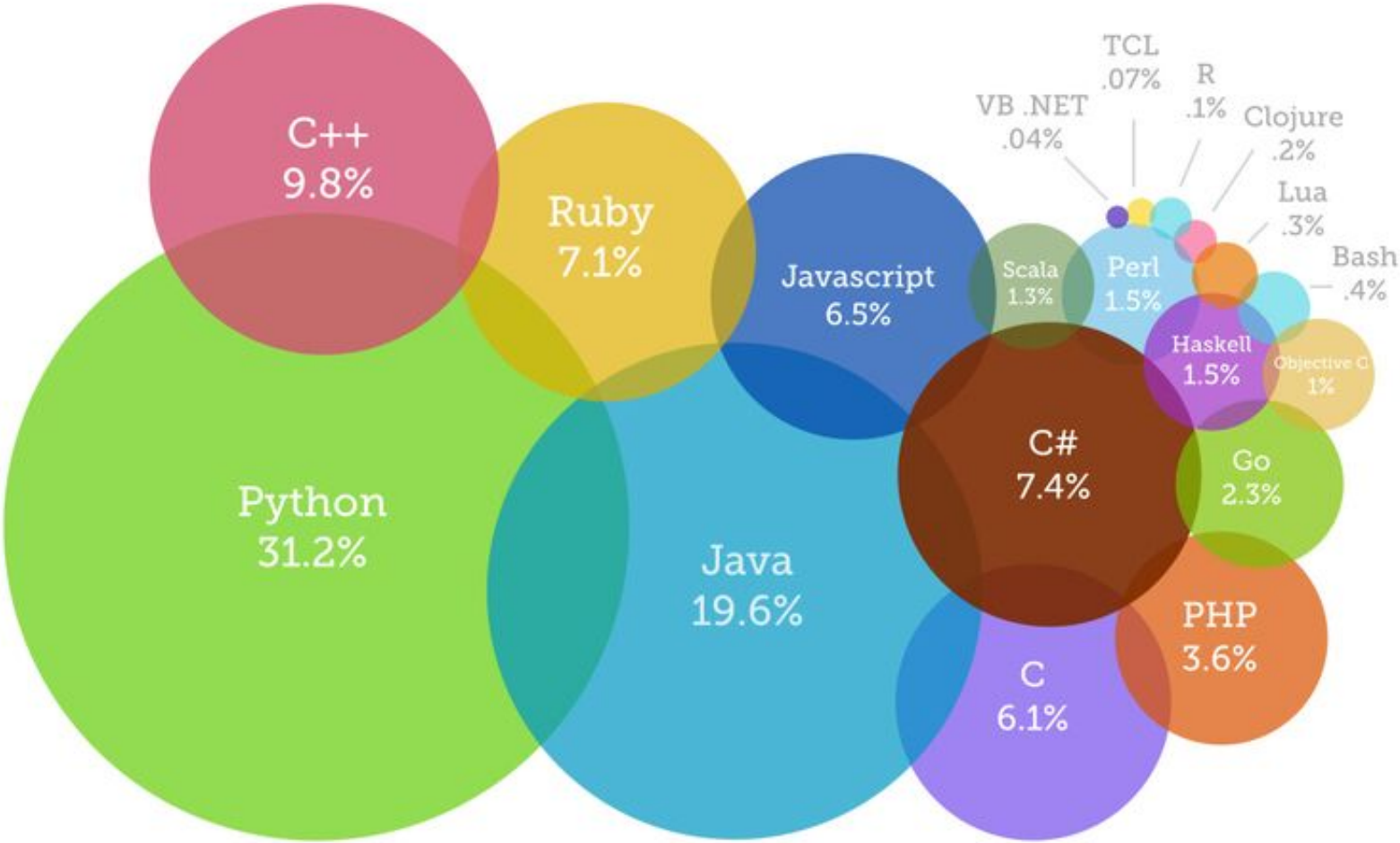
- review/seminar like lecture
- presentation: results of the previous lab
- practical intro to the curent lab
- 1 or 2 tasks to complete on your own

Course schedule

- Python intro (this week)
- Content presentation (JT, DAMF) (next week)
- First lab (next week)
- Three weeks long block of common subjects, alternating lecturer
- from our syllabus:
  “complexity, physics of biological system, quantum simulations, genetic algorithms and neural networks”

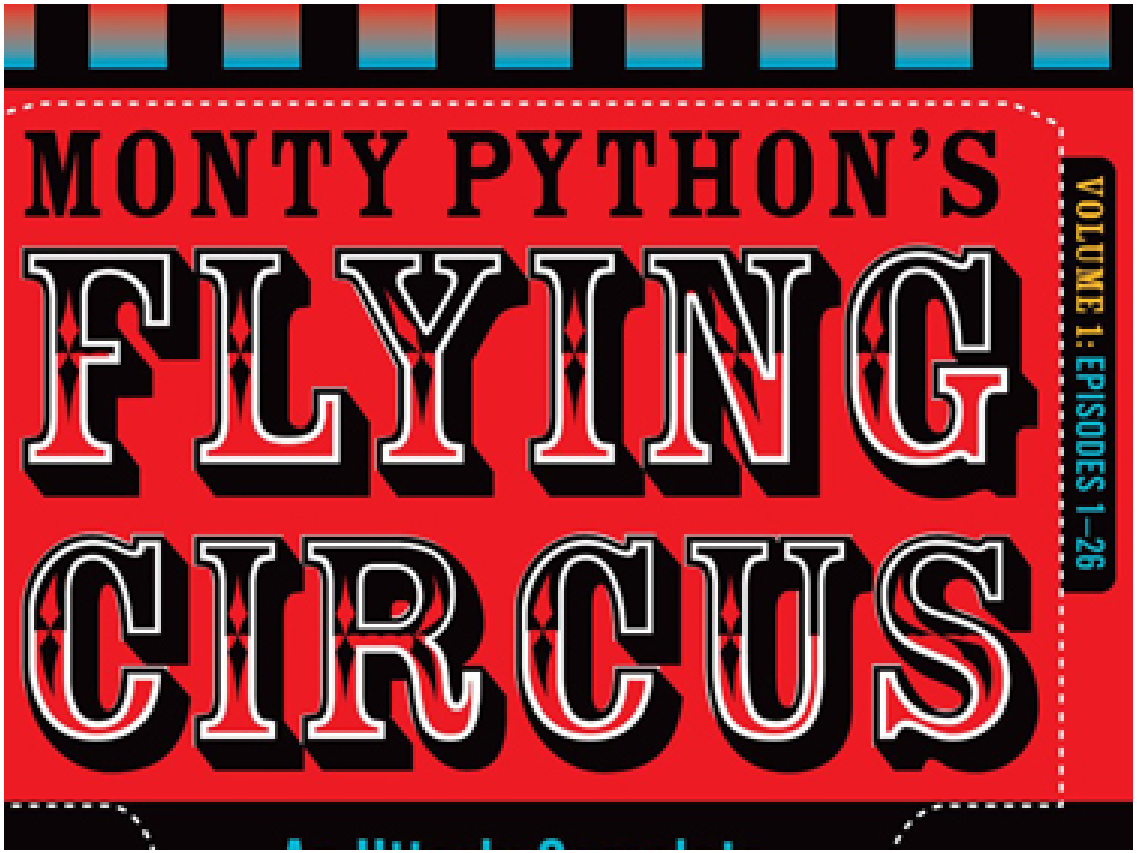
Plan

Programming



PYTHON

named after BBC show "Monty Python's Flying Circus"
... and has nothing to do with reptiles
... references to MP skits are encoureged

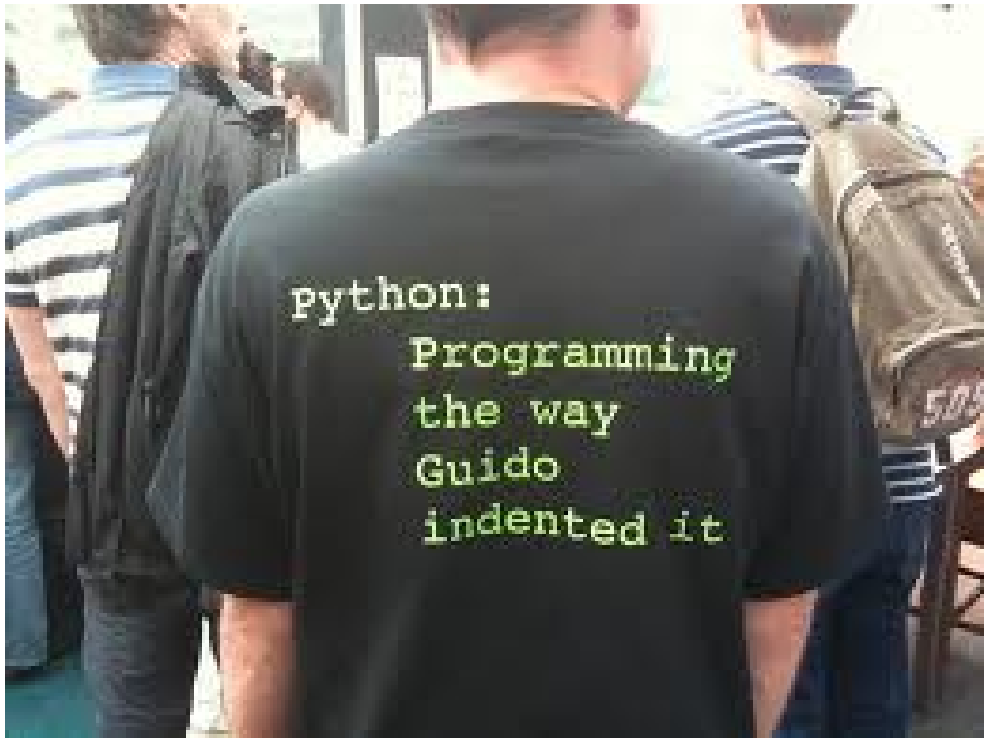


What is Python?

- interactive and interpreted (scripting language)
- is a real high-level language
- supports many data structures
- paradigms: procedural, object-oriented, even functional

What is Python?

- interactive and interpreted (scripting language)
- is a real high-level language
- supports many data structures
- paradigms: procedural, object-oriented, even functional



Python features

- clear syntax, easy to learn, readable code
- very powerful (batteries included philosophy)
  The Python Standard Library (huge),
  NumPy, SciPy, MatPlotLib (this is what we need)
- portable: can ran on many platforms (supercomputers inculding)

Python documentation

- http://docs.python.org/3.8/
  Fast access to Python documentation.
- http://docs.python.org/3.8/tutorial/
  Python Tutorial (by Guido)
- http://wiki.python.org/moin/ Python Wiki (also in Polish) – links to more guides
- Our Department: https://brain.fuw.edu.pl/edu/index.php/"Programowanie_z_Pythonem3"

Learning curve

- “A Byte of Python” by C.H.Swaroop
  good point to start (free online)
- “Python Tutorial 3.8” by Guido
  a classic, many details
- “Dive into Python”
  good for consulting when one knows something
  (free online)
- “Learning Python” by M. Lutz
  long text if one prefers a slower approach (our library)
- “Python Crash Course” by Eric Matthes (free online) eg.
  https://github.com/ValeriiaMur/Python-books-and-exercises/blob/master/python-crash-course.pdf

Some special equipment

- Numpy tutorial – tricks with arrays
  https://numpy.org/devdocs/user/quickstart.html
- Matplotlib tutorial – a way to control the plot
  https://github.com/rougier/matplotlib-tutorial
- ... and more on scientific Python
  http://scipy-lectures.github.io/

Some special equipment

- Numpy tutorial – tricks with arrays
  https://numpy.org/devdocs/user/quickstart.html
- Matplotlib tutorial – a way to control the plot
  https://github.com/rougier/matplotlib-tutorial
- ... and more on scientific Python
  http://scipy-lectures.github.io/
- $\rightarrow$ three useful cheat shits on our web page

Plan

1. Course organization and schedule
2. Introduction
3. Basics of Python programming
   - Variables, data types, operators
   - Loops and conditionals
4. More effective Python
   - Functions
   - Modules

Running Python

Different ways to use Python:

- interactive interpreter started from command line

```
>>> 2+2
4
```

- running a script

```
$ python3 skrypt.py
```

- from a graphical user interface (GUI) environment;
  basic `IDLE`, more advanced `Spyder`,
  `PyCharm` is popular on mimuw
  see also `http://wiki.python.org/moin/IntegratedDevelopmentEnvironments`
- new trend: using Jupyter notebook, running on Google Colab
- professionals: running Jupyter on a server (cluster)

Python identifiers

- identifier starts with a letter `A...Z` (or `a...z`),
  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
```

PEP 20 (Python Enhancement Proposals) – The Zen of Python
https://www.python.org/dev/peps/pep-0020/
PEP 8 – Style Guide for Python Code
https://www.python.org/dev/peps/pep-0008/

Lines and indentation

- one line one statement (unless un-readable)
- spaces and blank lines encouraged for readability
- blocks of statements are denoted by identation
- comments follow # sign up to the line end

```
condition = True

if condition:
    print( "I'm happy" )
    print( "True" )
else:
    print( "Too bad" )
    print( "False" )
```

Variables

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

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

- numerical types:

  `int` = C long int or... unlimited precision
  `float` = C double precision
  `complex` = two floats in double precision

Basic data types

immutable – to change the value we have to allocate a new object

- numbers
- string literals
- tuples – elements enclosed in `()` e.g. `(x,y,z)`

mutable – existing object can be modified

- lists – elements in square brackets: `['a','b','c']`
- dictionaries – unsorted pairs key:value contained within `{}`

Lists

- ordered sequence of any objects, array like
- elements are accessed with [] or [:] operators
- freedom to add and to remove elements

List – an example

```
>>> list_example = [ 'abcd', 786 , 2.23, 'john', 70.2]

>>> list_example.append('eva'); print( list_example )
['abcd', 786, 2.23, 'john', 70.2, 'eva']

>>> list_example.pop()
'abcd'
>>> print( list_example )
[786, 2.23, 'john', 70.2, 'eva']
```

Note that list methods `append()` and `pop()` realize a FIFO queue.

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)
- operators `[]` and `[:]` work with tuples
- applications: multiple results from a function, sets of parameters

Python philosophy

- in the algebraic expressions the type conversion is performed automatically
- operate on all types for which it makes sense (assigment `=` , arithmetic `+` , logical `==`)
- to convert between build-in types:
  `float(x), str(n), tuple(lista)`

If else elif

Similar to many other languages

```
if conditions:
    statements
```

or with else-elif construct

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

While loop

Just an example

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

For loop

Example 1

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

Example 2

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

Example 3

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

Plan

1. Course organization and schedule
2. Introduction
3. Basics of Python programming
   - Variables, data types, operators
   - Loops and conditionals
4. More effective Python
   - Functions
   - Modules

Function definition

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

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

welcome()
```

There are default values for the arguments

```
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(a=40, c=50)
```

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)
maximum__doc__
help(maximum)

maximum('ala','bob') # our code is very general!
```

Practical example of function wrapping by Anton Akhmerov
https://www.youtube.com/watch?v=EQIOyF2oots

What is a module?

- Module allows you to organize your Python code. A module is a Python object with named attributes, which you can use.
- 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, SicPy, MatPlotLib

Example how to use a module

Import a standard system module

```
import sys

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

print( '\nThe PYTHONPATH is', sys.path, '\n' )
```

The code above prints arguments given at a command line

```
$ python program.py ...
```

and then the directories searched for Phython modules

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

Our own module

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

version = "0.1"
# End of mymodule.py

-----------------------------------------------
# Filename: mymodule_demo.py
import mymodule

mymodule.hi()
print( 'Version', mymodule.version )
```

Script as a module

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

version = "0.1"

if __name__ == "__main__":
     print( hi() )

# End of mymodule.py
```

(from Guido) The file `mymodule.py` is usable as a script as well as an importable module, because the code that parses the command line only runs if the module is executed as the “main” file.

Next steps

Lab “0”: basic tutorial
There are $\approx$ 25 slides of a basic tutorial, browse through it to make sure you know all of the material.

Lab – NumPy & Matplotlib
Please follow this guide and run short scripts (as described in the text) on your own.

Task 1 simple plot with description
Task 2 linear algebra program
Task 3 histogram from radom Gaussian data