Computational Physics with PYTHON

Tobias Micklitz
Linneu Holanda
Carsten Hensel

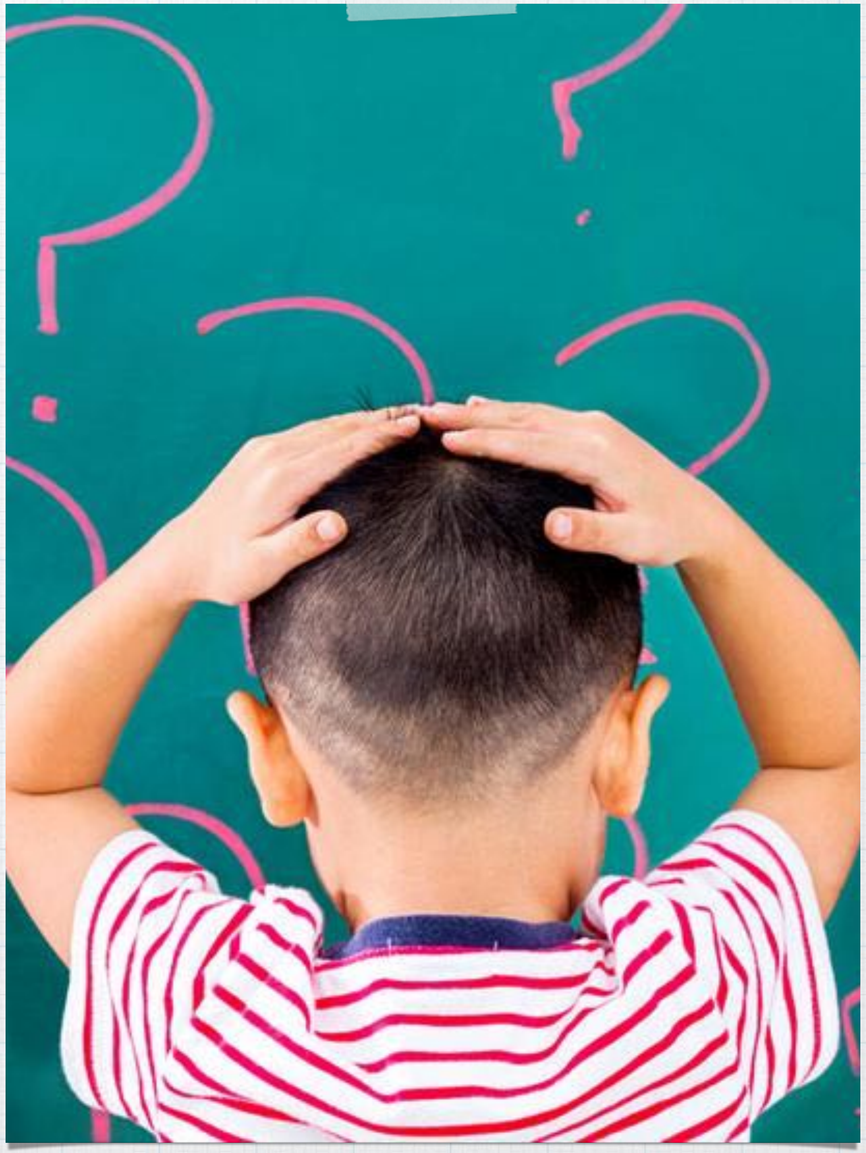
Disclaimer

* We're not a computing engineers nor computing scientists.

OUTLINE

* Why?
* Introduction to Python
* Python basics
* Differences to other programming languages
* Applications in Physics
* Summary/Outlook

Why?



Why Computational Physics?

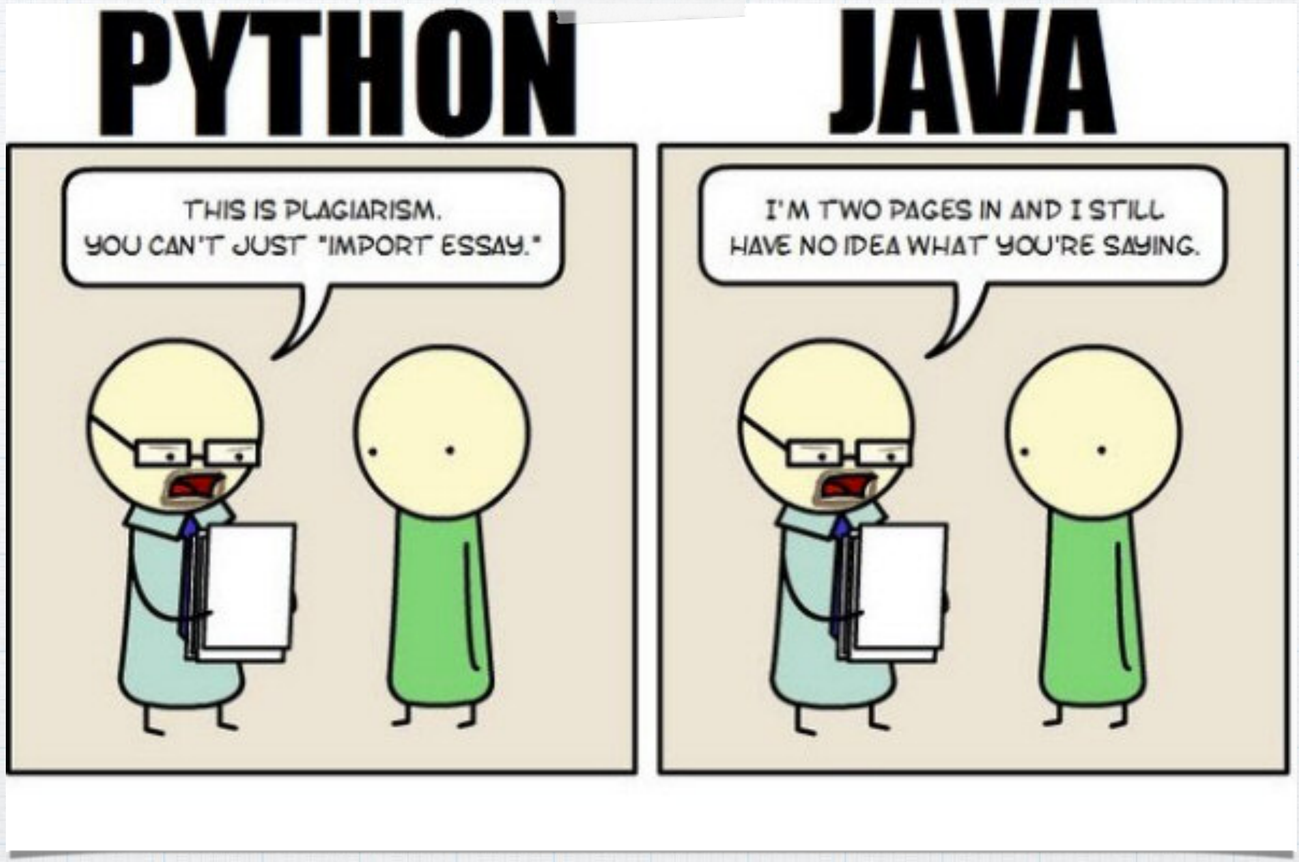
* Doing physics without computers is basically impossible nowadays:
  * From information exchange over monitoring experiments and simulations to complicated calculations.
* Computers became an integral part of physics (or research for that matter).
* Now, if we want to (have to) use computers in our daily work life we need to learn to communicate with them.

Why PYTHON?

* Python is easy to use, powerful and versatile.
* Perfect for beginners and experts alike.
* Python's readability makes it a great first programming language.
* It allows you to think like a programmer and not waste time understanding mysterious syntax.

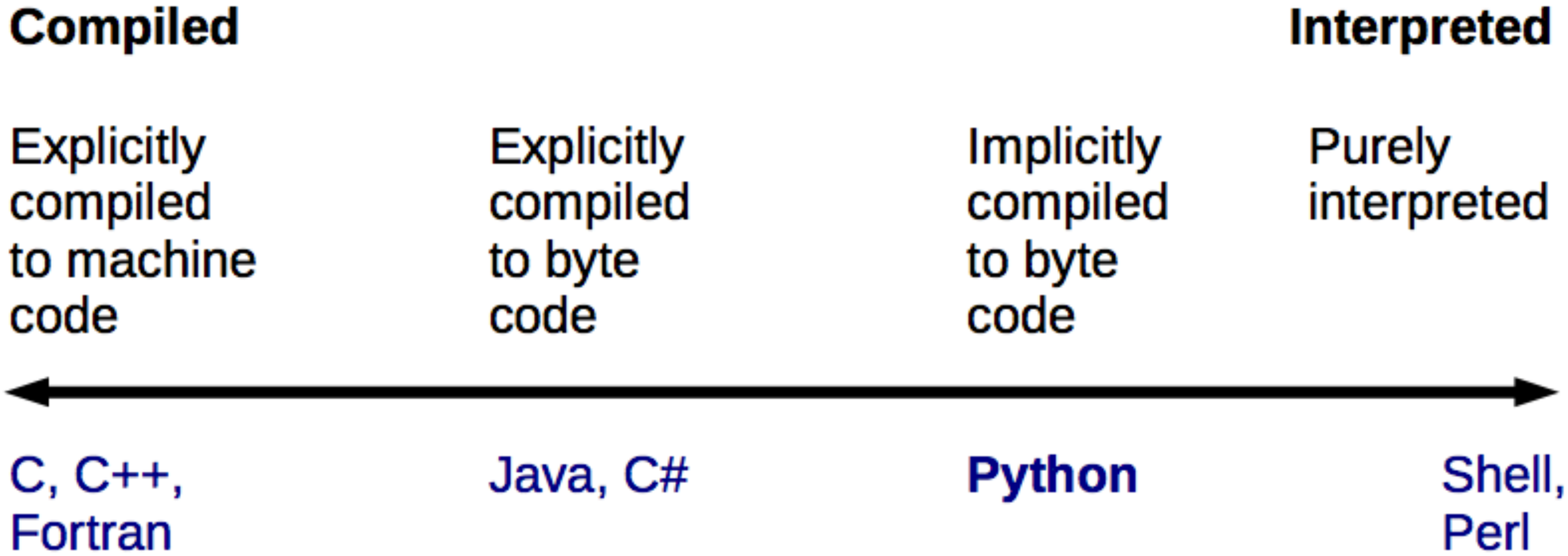
Which Programming Languages Do You Know/Use?

* C or C++
* Java
* Perl
* Scheme
* Fortran
* Python
* Matlab



Introduction

What Kind of Programming Language is PYTHON?

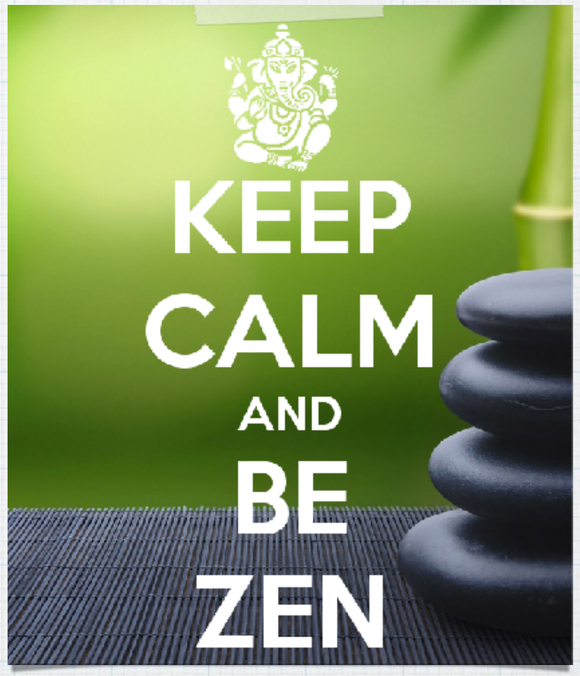


PYTHON

* open source general-purpose language
* objected oriented, procedural, functional
* easy to interface with C/ObjC/Java/Fortran
* easy-ish to interface with C++ (via SWIG)
* great interactive environment

The Zen of PYTHON

* Beautiful is better than ugly.
* Explicit is better than implicit.
* Simple is better than complex.
* Complex is better than complicated.
* Flat is better than nested.
* Readability counts.
* Special cases aren't special enough to break the rules.
* Although practicality beats purity.
* In the face of ambiguity, refuse the temptation to guess.
* There should be one -and preferably only one-obvious way to do it.
* Now is better than never.
* Although never is often better than *right* now.
* If the implementation is hard to explain, it's a bad idea.
* If the implementation is easy to explain, it may be a good idea



Python Basics

Starting to Program in Python

* The following is an overview of the language basics.
* It's not meant as a language tutorial.
* We will cover everything in detail in the next part of the lecture.
* So, lean back and relax for now!

Which Version of PYTHON?

* 'current' version is 2.7.X
* 'new' version is 3.6.X
* 2.7.X will be last stable release of PYTHON 2.
* Differences seem to be subtle for a beginners.
* If you start writing PYTHON code, you might want to stick with PYTHON 3

Running PYTHON

* We will assume that PYTHON is installed on your system.
* It comes pre-installed on Linux and Mac-OSX.
* For Windows please see the instructions (and binaries) on www.python.org
* We highly recommend the **anaconda** Python distribution.
  * Easy to install and everything you need.

Running PYTHON - The PYTHON Interpreter

* interactive interface to PYTHON:

```
Big-Bang:~ carsten$ python
Python 2.7.11 |Anaconda 4.0.0 (x86_64)| (default, Dec  6 2015, 18:57:58)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>>
```

* interactive interface to PYTHON:

```
>>> 3*(7+2)
27
>>>
```

* exit with CTRL-D

Running PYTHON - Running Programs

* execute your program like this

```
Big-Bang:~ carsten$
Big-Bang:~ carsten$ python my_python_program.py
```

* or make it executable by adding to the top of your file:

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

A Code Example

```
1
2   x = 34 - 23              # a comment
3
4   y = "Hello"              # another comment
5
6   z = 3.45
7
8   if z == 3.45 or y == "Hello":
9       x = x + 1
10      y = y + "World"     # String concat
11
12  print x
13  print y
14
```

* assignment with = and comparisons with ==
* for numbers +-*/% as expected
  * special use of + for string concatenation
  * special use of % for string formatting
* logic operators are words (and, or, not) not symbols
* basic printing command is print
* first assignment of a variable creates it
  * variable types don't need to be declared
  * Python figures out the variable type on its own

Basic Data Types

* Integers

```
z = 5 / 2   # answer is 2 (integer division)
```

* Floats

```
x = 3.456
```

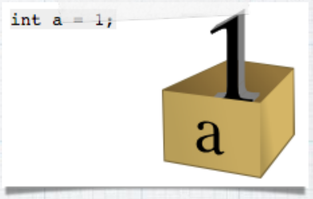
* Strings

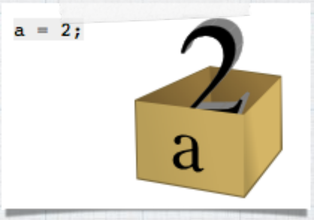
```
s = 'abc'
s = "abc"
s = """abc"""
```

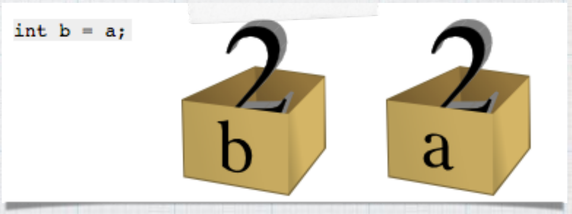
At this point we need to talk about how Python treats variable names.

Understanding Reference Semantics

* other languages have variables
* assigning to a variable puts a value into a 'box'
* box 'a' contains now value '1'
* assigning another value to the same variable replaces the contents of the box
* assigning one variable to another variable makes a copy of the value and puts it into a new box
* box 'b' is a second box with a copy of the value from box 'a'

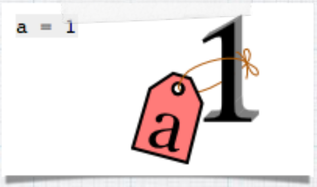






Understanding Reference Semantics

* Python has names
* in Python a name or identifier is like parcel tag
* here an integer 1 object has a tag labelled ‘a’
* reassigning moves the tag to another object
* assigning one name to another adds another tag
* here the name ‘b’ is just a second tag attached to the same object



Understanding Reference Semantics

* Does it matter?

* For simple built-in datatypes assignments behave as expected.

* However, mutable datatypes behave differently!

```
>>> a = 2
>>> b = a
>>> b = 4
>>> print b
4
>>> print a
2
>>>
```

```
>>> a = [1, 5, 10]   # this is a list
>>> b = a
>>> b.append(33)
>>> print b
[1, 5, 10, 33]
>>> print a
[1, 5, 10, 33]
```

Sequence Types: Tuples, Lists and Strings

* Tuple
  * a simple, immutable ordered sequence of items
  * items can be of mixed types, including collection types
* Strings
  * immutable
  * conceptually very much like a tuple
* List
  * mutable, ordered sequence of items of mixed types

Sequence Types: Tuples, Lists and Strings

* All three sequence types share much of the same syntax and functionality.
* key difference:
  * tuples and strings are immutable
  * lists are mutable
* examples shown here can be applied to all sequence types

Sequence Types: Definitions

* Tuples are defined using parenthesis (and commas).
* Lists are defined using square brackets (and commas).
* Strings are defined using quotes.

```
>>> tu = (3 , 'abc', 4.56, (2, 23), 'k')
>>> li = ['abc', 34, 3.1415, 23]
>>> st = 'Hello'
>>> st = "Hello"
>>> st = """Hello"""
>>>
```

Sequence Types: Accessing Members

* Individual members of a tuple, list or string can be accessed using a square bracket notation.
* Sequence types are all 0 based.

```
>>>
>>> tu = (3 , 'abc', 4.56, (2, 23), 'k')
>>> print tu[1]
abc
>>>
>>>
>>>
>>> li = ['abc', 34, 3.1415, 23]
>>> print li[1]
34
>>>
>>>
>>>
>>> st = 'Hello'
>>> print st[1]
e
>>>
```

Sequence Types: Negative Indices

* positive index: count from left, starting at 0
* negative index: count from right, starting with -1

```
>>>
>>> tu = (3 , 'abc', 4.56, (2, 23), 'k')
>>> print tu[1]
abc
>>> print tu[-1]
k
>>>
```

Sequence Types: Slicing

* You can return a copy of the container with a subset of the original members using a colon notation.

```
>>>
>>> tu = (3 , 'abc', 4.56, (2, 23), 'k')
>>> print tu[1:4]
('abc', 4.56, (2, 23))
>>> print tu[1:-1]
('abc', 4.56, (2, 23))
>>>
```

Tuples vs. Lists

* Lists are slower but more powerful than tuples.
    * Lists can be modified, and they have lots of handy operations we can perform on them (reverse, sort, count, remove, index, insert, ...)
    * Tuples are immutable and have fewer features.
* With the list() and tuple() functions lists and tuples can be converted.

One More Datatype: Dictionaries

* Dictionaries store a mapping between a of keys and a set of values.
  * Keys can be any immutable (!) type.
  * Values can be any type.
  * A single dictionary can store values of different types.
* You can define, modify, view, lookup, and delete the key-value pair in the dictionary.

Dictionary Examples

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user']
'bozo'
>>> d['pswd']
1234
>>> d['bozo']

Traceback (innermost last):
  File '<interactive input>' line 1, in ?
KeyError: bozo

>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user'] = 'clown'
>>> d
{'user':'clown', 'pswd':1234}

>>> d['id'] = 45
>>> d
{'user':'clown', 'id':45, 'pswd':1234}
```

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> del d['user']              # Remove one.
>>> d
{'p':1234, 'i':34}
>>> d.clear()                  # Remove all.
>>> d
{}


>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> d.keys()                   # List of keys.
['user', 'p', 'i']
>>> d.values()                 # List of values.
['bozo', 1234, 34]
>>> d.items()          # List of item tuples.
[('user','bozo'), ('p',1234), ('i',34)]
```

Whitespace

* Whitespace is meaningful in Python: especially indentation and placement of new lines.
* Use newline to end a line of code
* No braces { } to mark blocks of code!
* Use indentation instead

```
157   f=open(fn,'r')
158   for line in f:
159     try:
160       newline = line.rstrip('\n')
161       toks=newline.split(',')
162       if (len(toks) == 1):
163         if newline == "Spend lifts":
164             print "that's it: " + line
165               break # don't plot spend lift
166             if(len(toks)!=4) and (len(toks)!=5) and (len(toks)!=7):
167                 continue
168             if line.find("Lg0") > 0:
169                 bidType=toks[2]
170                 lift=float(toks[3])
171                 name=toks[0]
172                 updateDetails(summaryDetails, bidType + "-details-" + mydate, lift)
173                 updateDetails(flightData, name + "-lift", lift )
174                 updateDetails(flightData, name + "-liftPoints", 0.01) # fake variance for now
175                 updateDetails(flightData, name + "-date", mydate)
176                 if not name in flightKeys.keys():
177                     flightKeys[name] = 1
```

Functions

* 'def' creates a function and assigns a name
* 'return' sends a result back to the caller
* arguments are passed by assignment
* arguments and return types are not declared

```
def <name>(arg1, arg2, ..., argN):
  <statements>
  return <value>

def times(x,y):
  return x*y
```

Passing Arguments to Functions

* Arguments are passed by assignment.
* Passed arguments are assigned to local names.
* Assignment to argument names don't affect the caller.
* Changing a mutable argument may affect the caller.

```
def changer (x,y):
  x = 2                          # changes local value of x only
  y[0] = 'hi'                    # changes shared object
```

Function Gotchas

* All functions in Python have return values!
* Functions without a return, return the special value 'None'
* There is no function overloading in Python.
    * Two different functions can't have the same name, even if they have different arguments.
* Functions can be used as any other data type. They can be:
    * arguments to other functions
    * return values of functions
    * assigned to variables
    * parts of lists, tuples, etc.

Fun With Functions

```
def f(x, y):
    return x + y


def g(x, y):
    return x * y


def h(x, y):
    if y == 0:
        return 0
    else:
        return x / y


list_of_functions = [f, g, h]

a = 23
b = 9
for function in list_of_functions:
    print function(a, b)
```

Things Not Covered

* OO, classes, inheritance
* modules
* introspection
* iterators, generators, comprehensions
* standard library
* ...

Differences to Other Languages

Python vs. Java

* Python programs are usually expected to run slower.
* But they also take less time to develop.
* Python programs are usually 3-5 times shorter than equivalent Java code.

Python vs. Perl

* Both come from the same background.
* Have many similar features, but very different philosophies.
  * Perl emphasises application-oriented tasks: file scanning, regular expressions, report generating features, etc.
  * Python emphasises common programming methodologies: data structures, OO,...
* Python comes close to Perl but will not be able to beat it in its core use cases.
* However, Python has an applicability well beyond Perl.

Python vs. C++

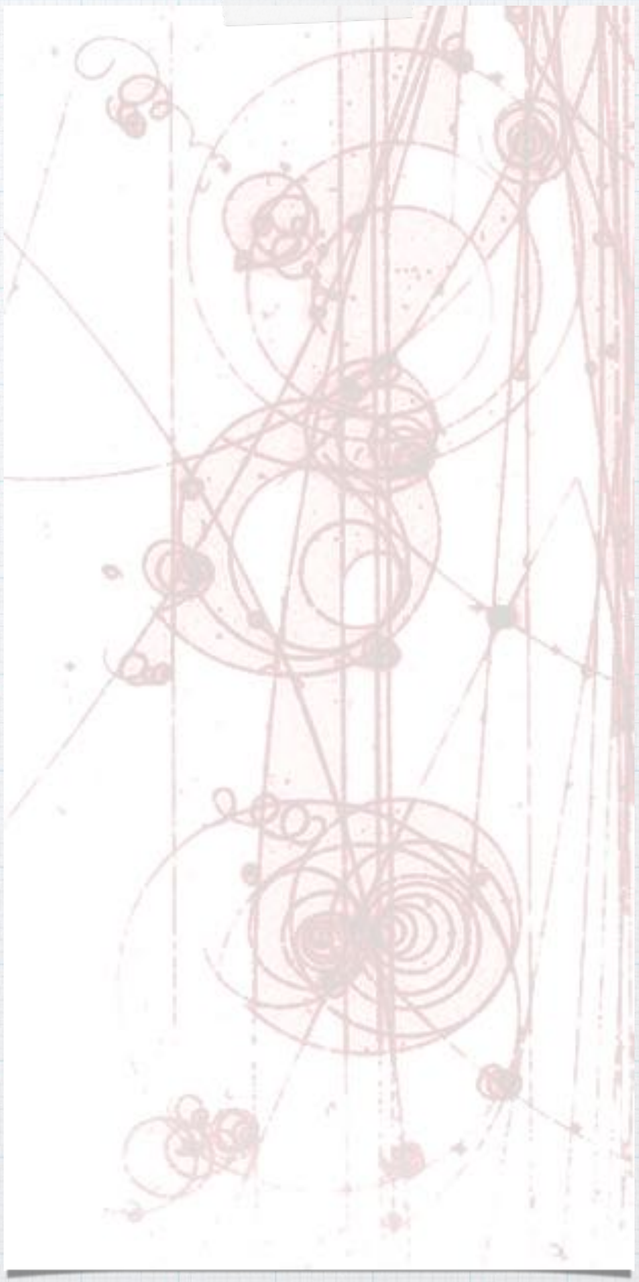
* Everything said about Java applies here as well.
* Python code usually 5-10 times shorter.
* There's saying, that one Python programmer can finish in two months what two C++ programmers can't complete in one year.
* Python shines as a glue language to combine components written in C++.

Pros and Cons

* 3 Disadvantages of Python
  * concurrency and parallelism possible but not very elegant
  * server/client programming does not really require Python
  * meta-programming (LISP) not a strong side of Python
* 3 Advantages of Python
  * time-wasting matters of style (blocks and curly braces) don't exist
  * The 'easy' way of doing something in Python is usually the correct way.
  * You can become productive with Python very quickly, even as a beginner.

Python in Physics

Is Python right for me?

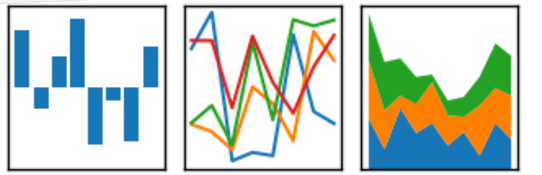


I need... to manipulate big data structures

* You might want to look into Pandas.

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



* A software library written for Python for data manipulation and analysis.
  * data alignment
  * time series functionality
  * group by, pivoting,
  * ...

PLOT	1	2	4	5	6	7	8	9	10	11	12	13	14	15
SPECIES	aq	aq	gr	mix	gr	gr	aq	mix	mix	sed	gr	gr	mix	gr
DEPTH	d	s	s	d	d	s	d	s	d	s	d	d	d	s
SLOPE	n	n	n	n	n	n	n	n	n	n	n	n	n	n
ORIENT	-	-	-	-	-	-	-	-	-	-	-	-	-	-
07/06/2013 14:47	1083.89	1022.22	1117.6	1103.12	1146.6	1055.02	1152.46	1044.65	1104.03	1050.03	1088.2	1098.95	1193.37	1054.44
08/06/2013 8:11	1078.73	1018.97	1114.06	1094.41	1138.12	1048.54	1144.73	1027	1090.59	1037.39	1081.65	1090.11	1179.23	1045.56
10/06/2013 12:51	1064.39	1015.18	1109.93	1060.64	1126.93	1048	1137.39	999.481	1056.97	1013.69	1073.29	1084.58	1144.61	1043.89
11/06/2013 14:55	1060.71	1023.85	1117.01	1057.73	1129.24	1055.56	1138.94	1001.56	1052.72	1017.9	1074.01	1086.05	1140.15	1050.56
12/06/2013 20:26	1041.57	1005.42	1091.07	1038.13	1110.73	1033.98	1119.61	981.828	1035.03	1000	1058.01	1068.36	1113.34	1027.22
13/06/2013 19:11	1064.02	1047.15	1134.1	1061	1132.72	1073.35	1146.28	1018.69	1058.03	1037.39	1080.2	1093.05	1136.05	1065.56
14/06/2013 9:50	1056.66	1037.4	1127.03	1055.92	1128.09	1065.8	1140.1	1010.38	1052.37	1030.02	1075.83	1087.89	1130.09	1058.89
14/06/2013 19:16	1067.33	1061.79	1148.84	1068.63	1140.05	1086.84	1155.17	1029.6	1064.76	1049.5	1087.11	1102.27	1142.38	1079.44
15/06/2013 10:36	1094.55	1102.98	1199.53	1099.13	1172.45	1129.99	1188.41	1074.25	1096.6	1091.63	1116.2	1133.96	1177.37	1129.44
16/06/2013 15:24	1058.87	1050.41	1144.71	1062.45	1136.57	1079.83	1153.24	1024.92	1057.32	1042.65	1081.65	1100.42	1138.28	1074.44
17/06/2013 16:44	1051.51	1040.65	1134.1	1054.47	1130.4	1069.04	1146.28	1012.46	1047.42	1032.12	1078.74	1094.53	1123.39	1064.44
18/06/2013 14:15	1058.5	1056.37	1148.84	1062.82	1140.05	1084.68	1156.71	1025.44	1056.97	1044.23	1087.11	1104.48	1132.7	1078.33
19/06/2013 15:05	1058.87	1062.33	1152.96	1062.45	1141.2	1092.77	1159.42	1025.96	1057.32	1045.81	1087.47	1106.32	1133.07	1086.67
20/06/2013 13:00	1179.18	1139.84	1252.58	1223.31	1247.3	1169.9	1250.24	1211.32	1219.75	1212.74	1179.49	1191.45	1303.18	1176.11
21/06/2013 16:42	1199.78	1158.27	1272.62	1254.54	1265.05	1186.08	1255.65	1240.39	1253.72	1241.18	1194.76	1207.67	1331.47	1196.11
25/06/2013 21:35	1160.41	1201.08	1229.59	1221.5	1231.48	1149.95	1237.49	1190.03	1215.15	1199.58	1157.3	1177.81	1303.55	1157.78
26/06/2013 20:48	1117	1137.13	1162.39	1184.46	1185.96	1089.54	1194.2	1127.73	1174.81	1145.87	1115.11	1135.07	1263.35	1095.56

I need... to deal with big arrays, matrices

* NumPy extension was written adding support for large, multi-dimensional arrays and matrices along with a large library of high-level mathematical functions to operate on these arrays.

* example: element-wise multiplication of large arrays

Python

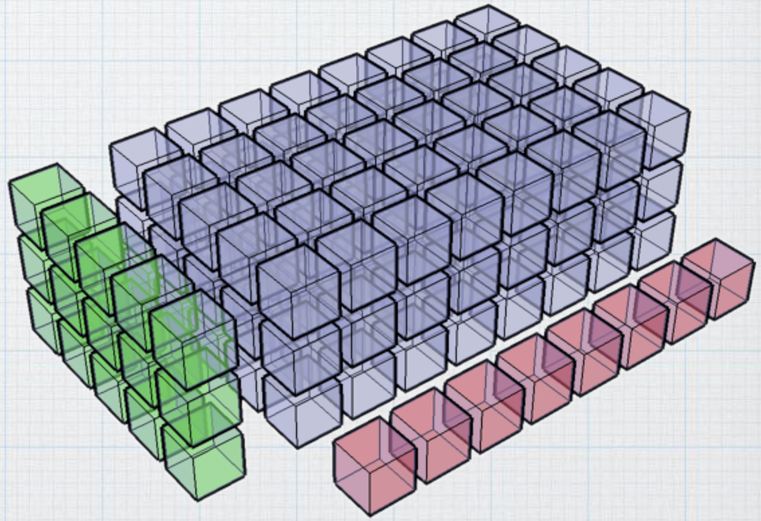
```
c = []
for i in range(len(a)):
    c.append(a[i]*b[i])
```

slow

Numpy

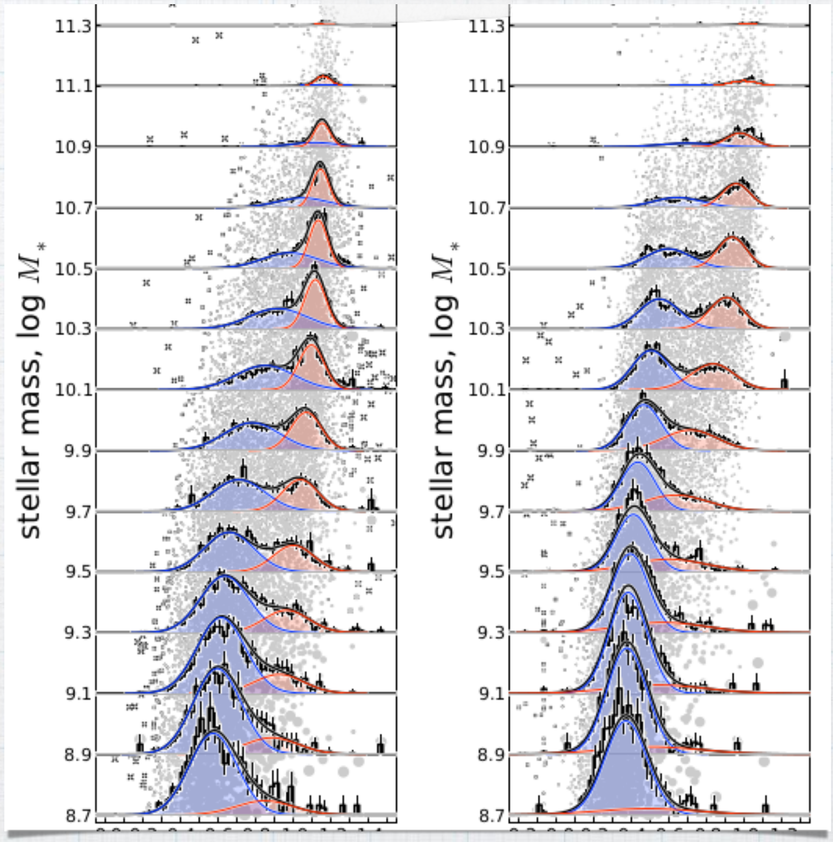
```
c = a * b
```

fast (C)



I need... to do scientific computing

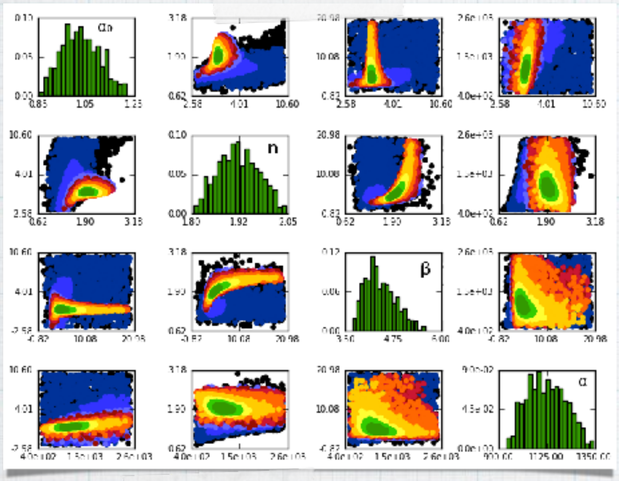
- SciPy comes with support of:
  - optimisation
  - linear algebra
  - integration
  - interpolation
  - special functions
  - FFT
  - ordinary differential equations
  - ...

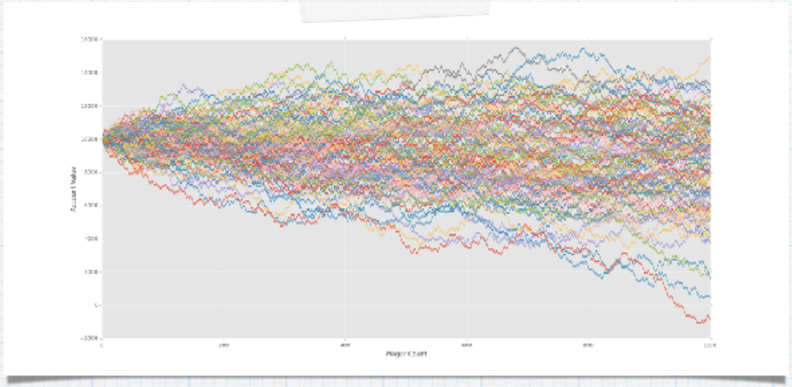


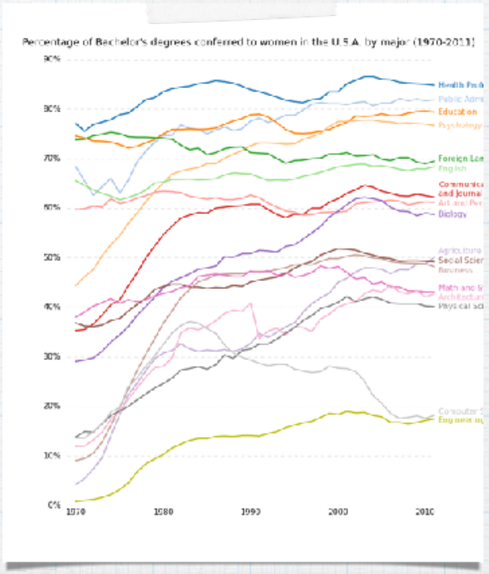
I need... to visualise data/results

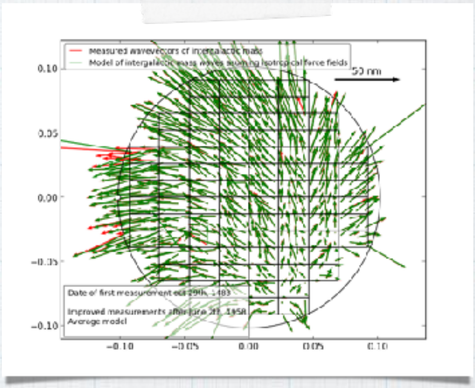
* matplotlib is a plotting library for Python

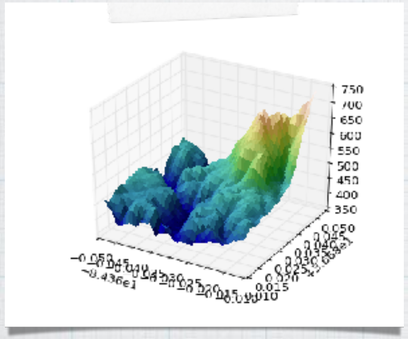
matplotlib

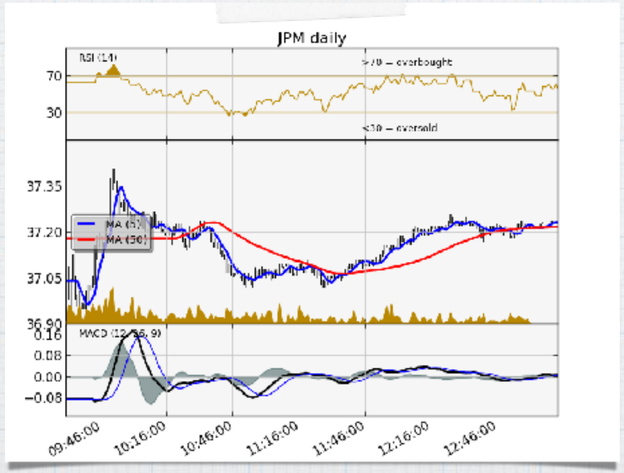


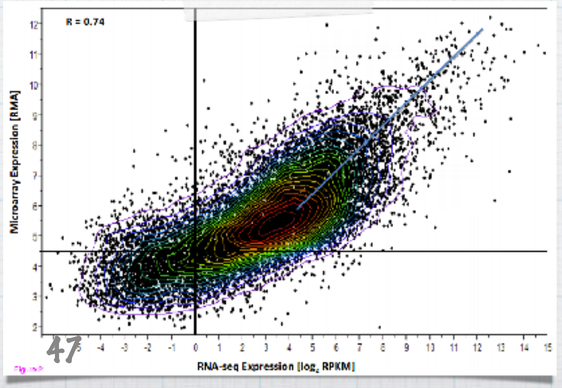


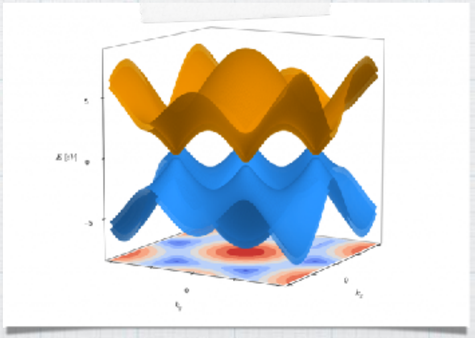












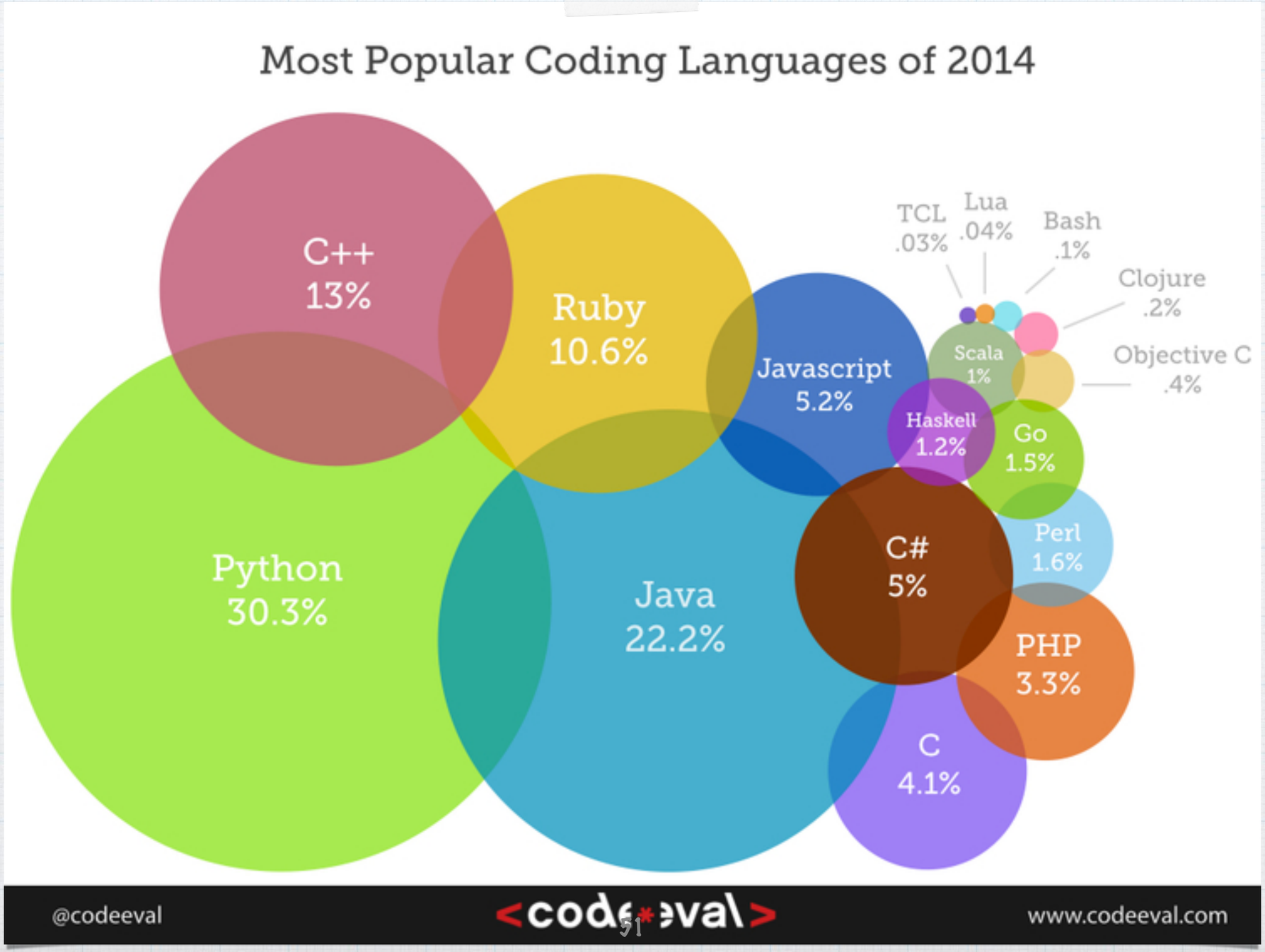
I need... ROOT

* Rootpy is a pythonic layer on Pyroot (which is a Python interface to ROOT)
  * ...does not intend to recreate ROOT or to severely alter the default behaviour of ROOT.
  * ...is not an analysis framework, but rather a library that one's analysis framework might use.
  * ...provides interface to scientific Python packages (Pandas, Numpy, SciPy, ...)

Wait... There's More

* QuTiP: simulation of dynamics of open quantum systems
* SymPy: library for symbolic mathematics
* scikit-learn: machine learning in Python
* astropy: single core package for astronomy in Python
* cosmocalc: Python version of the Cosmology Calculator
* ALPS: algorithms and libraries for physics simulations
* SunPy: solar physics
* ...

Summary


Most Popular Coding Languages of 2014
C++
13%
Ruby
10.6%
Javascript
5.2%
TCL
.03%
Lua
.04%
Bash
.1%
Clojure
.2%
Scala
1%
Objective C
.4%
Haskell
1.2%
Go
1.5%
Perl
1.6%
C#
5%
PHP
3.3%
Python
30.3%
Java
22.2%
C
4.1%
@codeeval
<codeeval>
www.codeeval.com

Python

* Python, yet another programming language
* There aren't many pathological cases (in physics) that won't allow the usage of Python.
* Quick development cycles make Python a true alternative to other programming options.
* Beginners make progress fast.

Goal of this Course

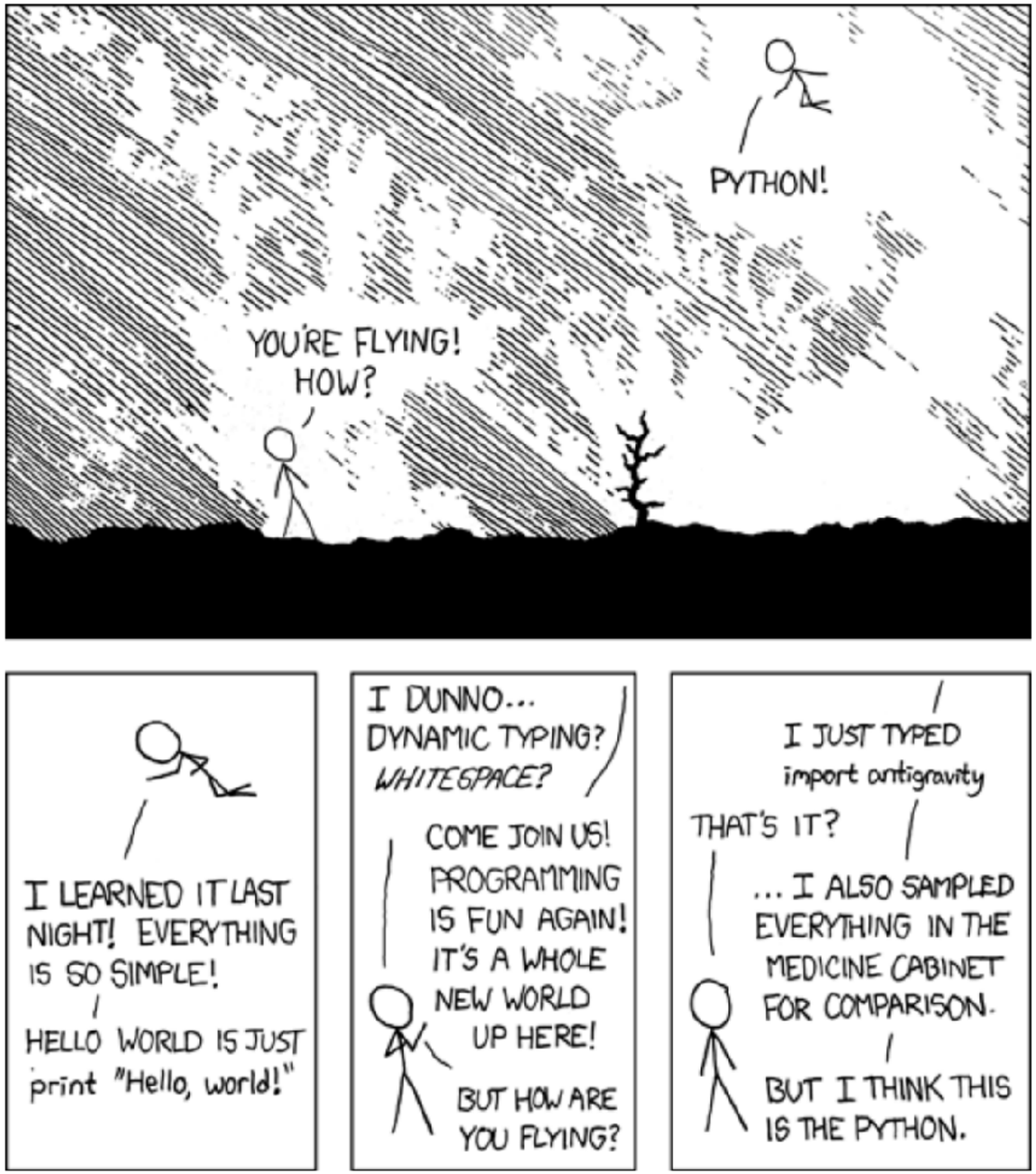
* Teach Python
* Use examples from computational physics.

The Course Itself

* It's going to be a lecture.
* But we strongly believe that active participation from your side will be greatly beneficial for you!
* Active participation: exercises, questions, discussions, presentations (?)
* Course details (slides, references, articles of interest) will be collected on our website: https://pythonatcbpf.wordpress.com
* Code examples etc. will be available in our github repository: https://github.com/CarstenHensel/PythonAtCBPF

Course Outline

* Python Basics - Variables and such
* Python Basics - Program flows and programming styles
* Differential Equations (Phase Space Portraits)
* Random Number Generators (Simulations)
* Classical and Quantum Random Walks
* Topological Phases in Condensed Matter
* Classifications (Artificial Neural Networks, Decision Trees)


PYTHON!
YOU'RE FLYING! HOW?
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.