TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

A hands-on introduction to Scientific Python

Markus Wallerberger

Department of solid state physics, TU Wien
Co-organised by the Fachschaft Doktorat

Sponsored content

- 

- Your representation as fellow doctoral students
- Answering your questions
  - http://fsdr.at/en
  - fsdr@fsdr.at
  - Getreidemarkt 9 (every second Tuesday 17.30)
- Orientation programme, workshops, movie screenings, parties, etc.

Goals

- Using Python as calculator
- Using Python for small scripts
- Using NumPy to speed up numerics
- Using SciPy for scientific applications
- Making SciPy programs fast
- Interfacing SciPy with existing code

	Friday	Monday
14.00	Session 1: **Python Basics**	Session 3: **SciPy + Plotting**
~16.00		
~16.30	Session 2: **NumPy basics**	Session 4: **Interfacing** Theano library *
~18.30		

* YOUR SUGGESTION!

Introduction to Scientific Python

What do you expect?

What you do you want to learn?

Python Basics

Python tutorial

- not enough time for comprehensive Python introduction
- Crash course!
- Filling the gaps:
  - read the excellent docs: https://docs.python.org/
  - Try some examples: https://projecteuler.net/

Why Python?

- **Availability**
- **Free software**
- Ease of use
- Structures
- Ecosystem

Installing (Scientific) Python

- Windows + Mac OS
  - Best: download a full SciPy stack
  - Example Anaconda: http://continuum.io/downloads
  - Alternative: use MinGW or CygWin + Packages
- Linux
  - Best: use package manager to download
  - Debian + Ubuntu: type into console

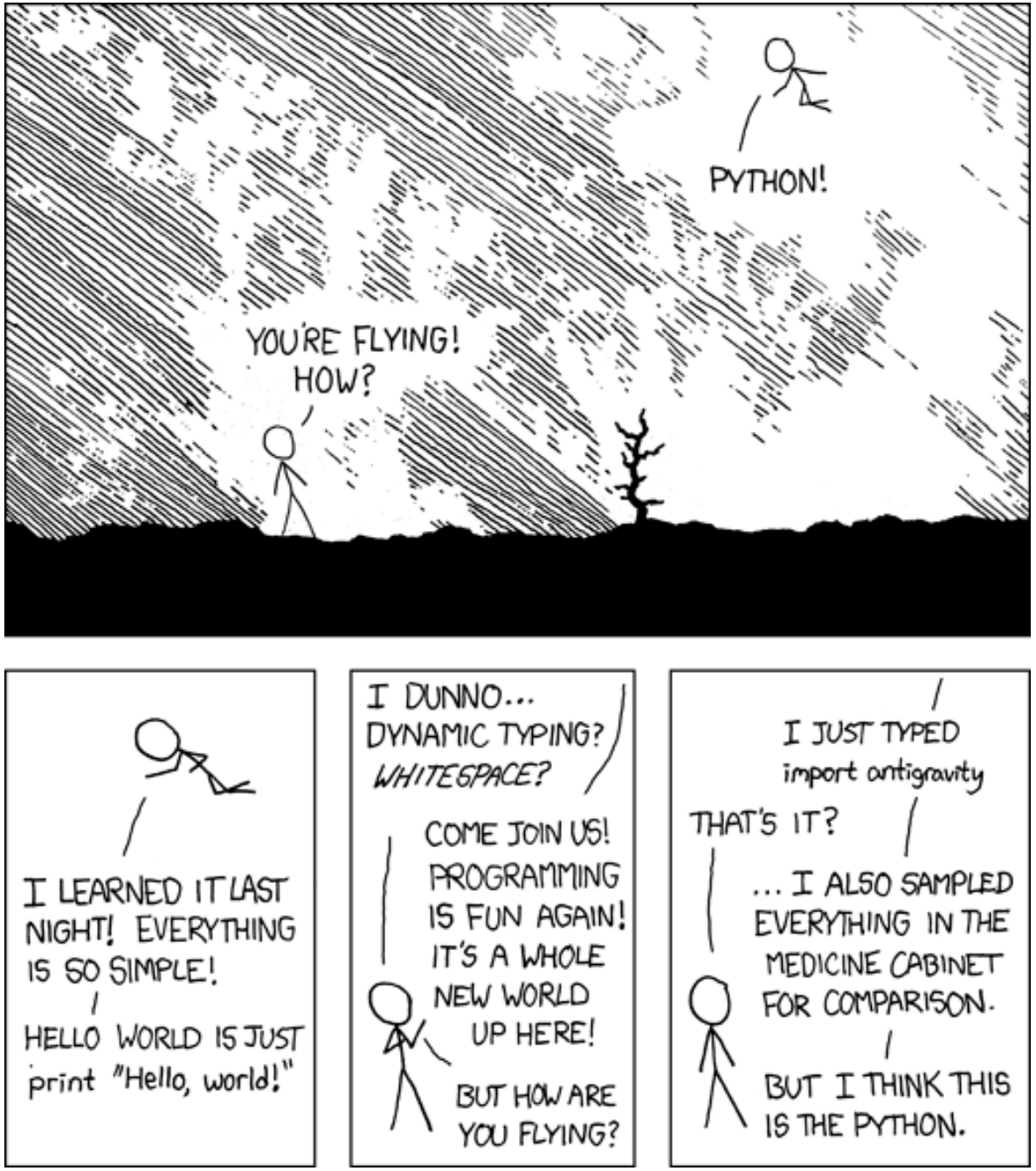
```
sudo apt-get install ipython-notebook \
                     python-scipy python-matplotlib cython
```

Python 2 or 3?

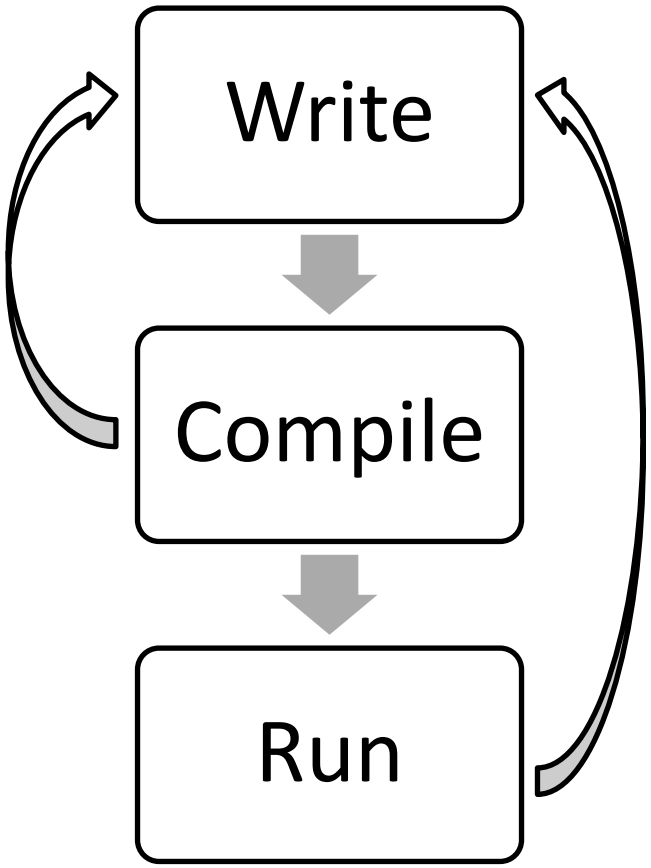
- (Some) incompatible syntax
- Python 3 is superior, but
- Python 2 is more widely available (clusters!)
- does not matter that much for SciPy
- → Python-2-with-sidegaze-to-3

Why Python?

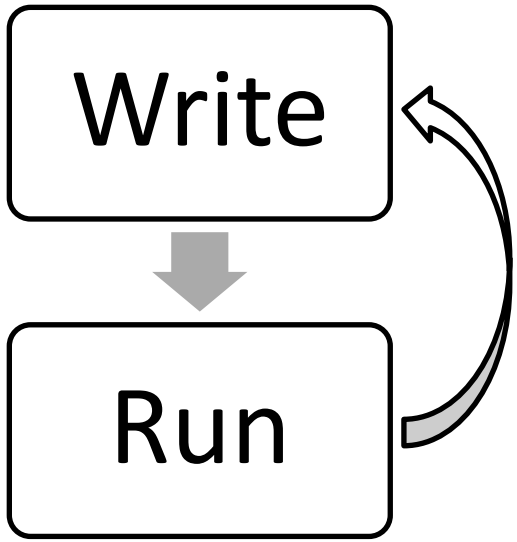
- Availability
- Free software
- **Ease of use**
- Structures
- Ecosystem

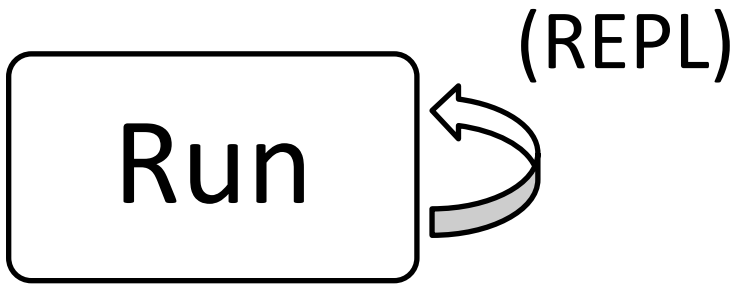


Compiled language

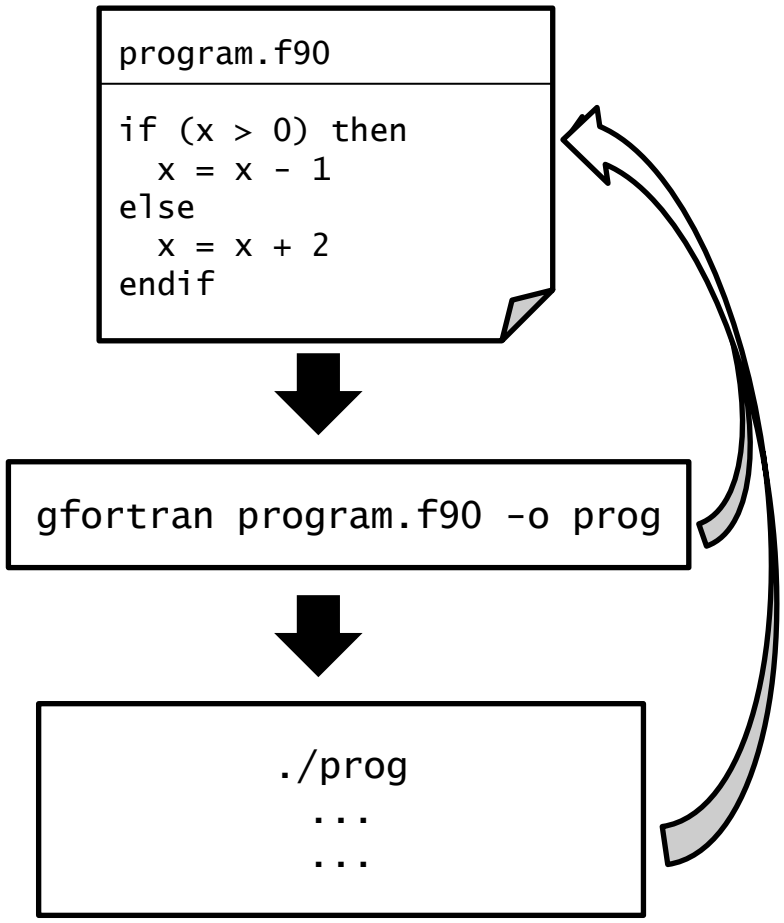


Scripting language

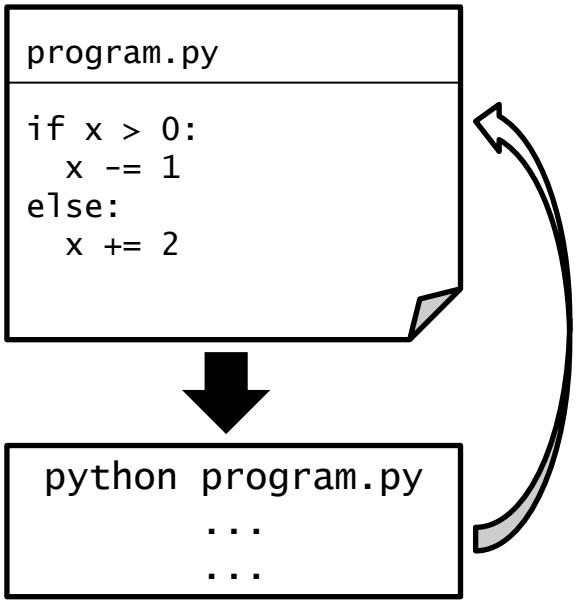


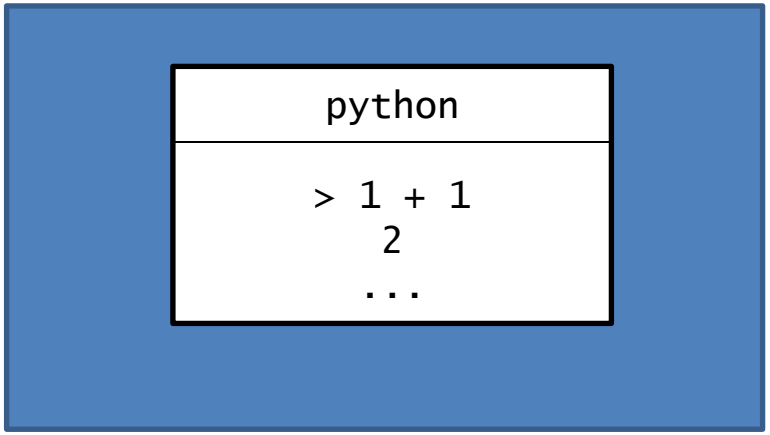


Fortran



Python



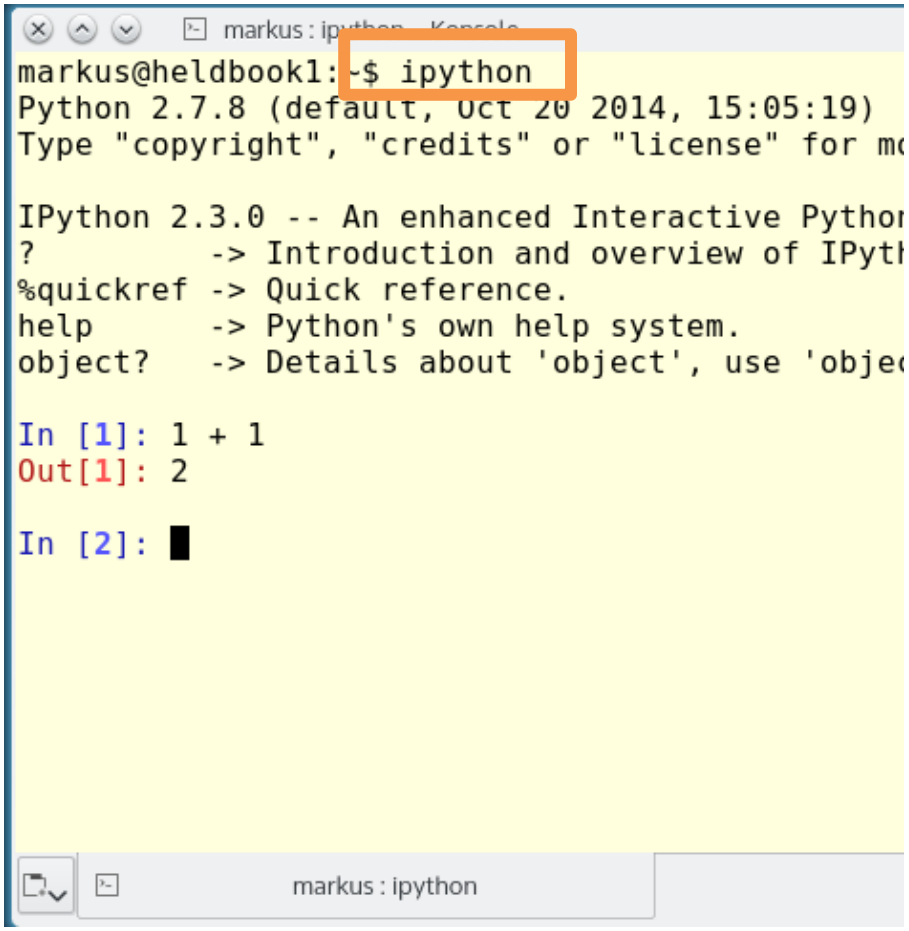


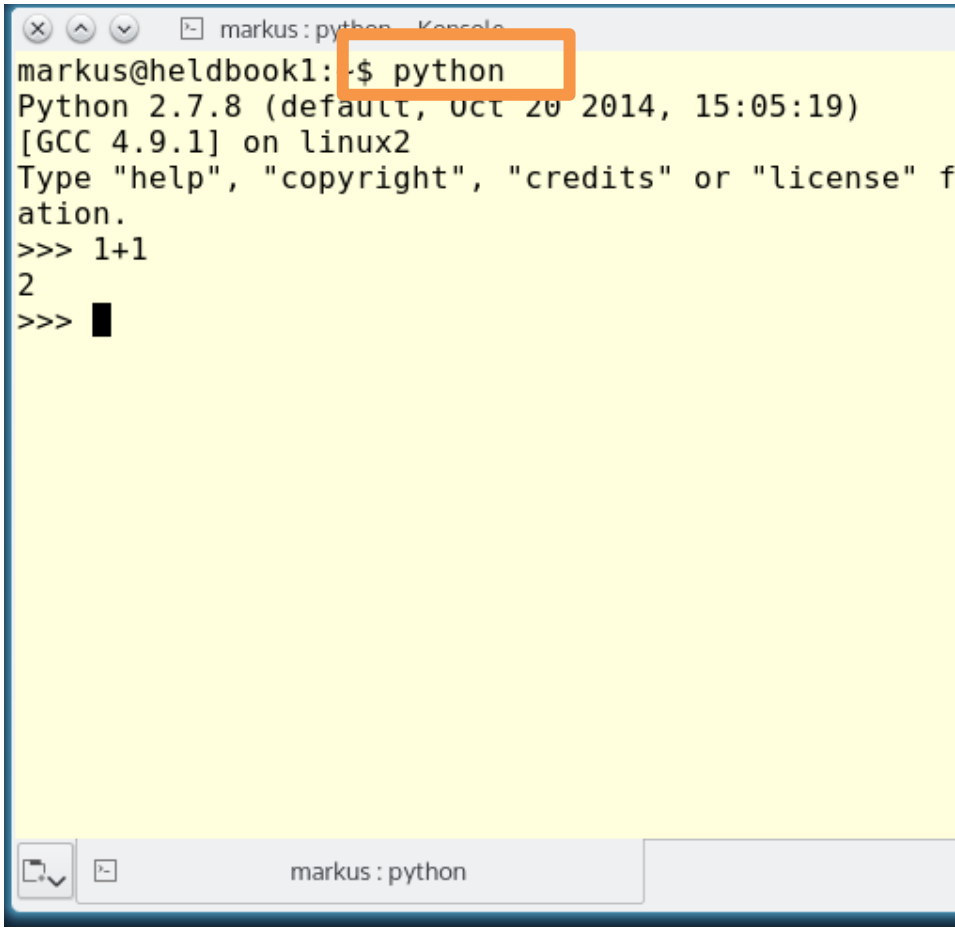
Interactive python

- **Windows**: start
  Start > Programs > Anaconda > IPython Qt Shell
- **Linux**: type python or ipython in shell
- like shell: type expression, press [Enter]
- return value is printed
- Get out: type exit() or press [Ctrl+D]

Usage (important!):

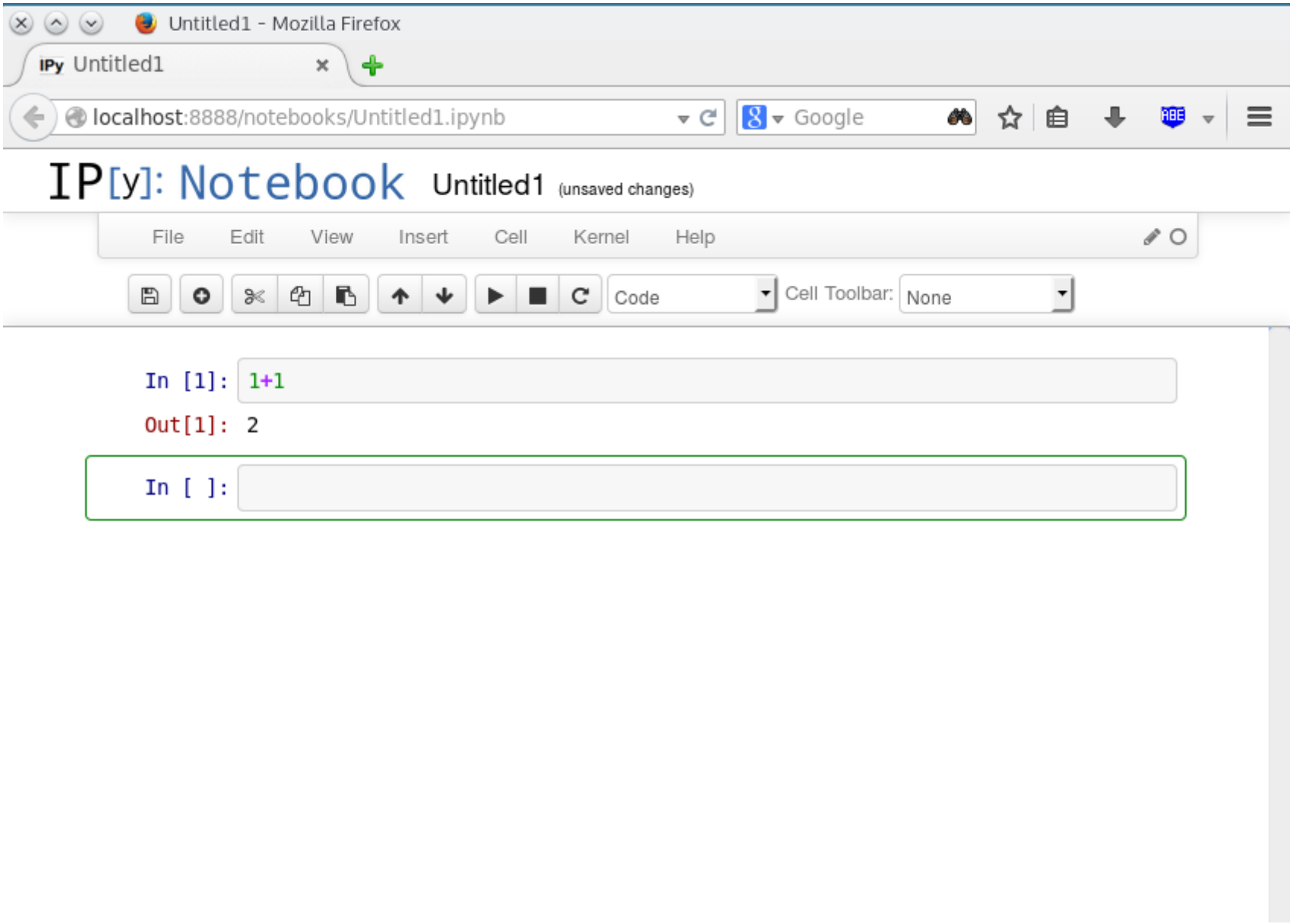
- Calculator, small tasks
- trying out stuff


markus : ipython
markus@heldbook1:~$ ipython
Python 2.7.8 (default, Oct 20 2014, 15:05:19)
Type "copyright", "credits" or "license" for mo
IPython 2.3.0 -- An enhanced Interactive Python
? -> Introduction and overview of IPyth
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'objec
In [1]: 1 + 1
Out[1]: 2
In [2]:
markus : ipython


markus : python
markus@heldbook1:~$ python
Python 2.7.8 (default, Oct 20 2014, 15:05:19)
[GCC 4.9.1] on linux2
Type "help", "copyright", "credits" or "license" f
ation.
>>> 1+1
2
>>>
markus : python

Alternative: IPython Notebook

- **Windows**: start Start>Programs>Anaconda>IPython Notebook
- **Linux**: ipython notebook in shell
- Runs in browser
- like Mathematica: type command, press [Shift+Enter]
- return value is printed
- Get out: close the browser window ☺
- Same use cases


Untitled1 - Mozilla Firefox
Untitled1
localhost:8888/notebooks/Untitled1.ipynb
Google
IP[y]: Notebook
Untitled1
(unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Help
Code
Cell Toolbar:
None
In [1]: 1+1
Out[1]: 2
In [ ]:

- 1 + 1
- 2 * 10**3
- 1/2
- 1.0/2
- 173 % 10
- variable = 3
- v*[press TAB]*
- (variable + 1)/2
- **import** math
- math.sin(math.pi/2)
- math.sin?
- math.*[press TAB]*

play around a bit

Store given sides of right-angled triangle in variables a, b;
calculate Hypothenuse

Store sides of triangle in a, b and angle gamma between them (in degrees);
calculate length of side c

Arithmetics

- Arithmetic operators like in Fortran/C:
  +, −, *, /, () , ** (power)
- Assignments:
  =, +=, −=
- Integer division in Python 2:
  1/2 → 0
- Integer divison in Python 3 (or Python 2 with future):
  from __future__ **import** division
  1/2 → 0.5
  1//2 → 0

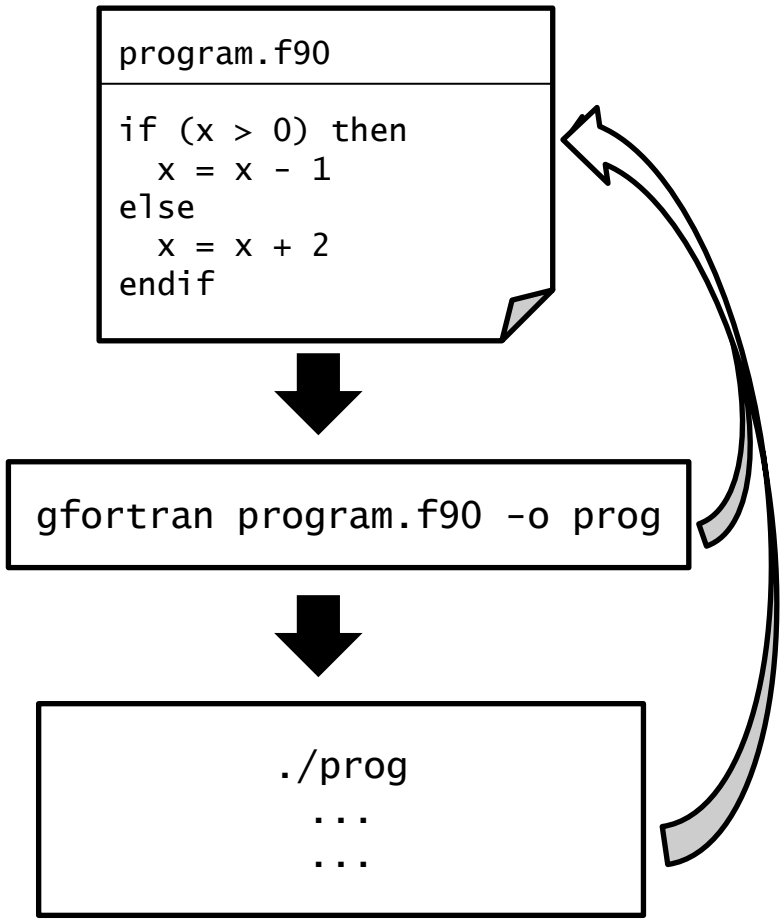
Variables

- Case-sensitive!
  - Convention: lower_case_with_underscores
- Dynamically typed
  - no type declaration needed
  - created at first assignment: x = 0
- Garbage collected
  - Objects „shared“ rather than copied (like in Java/MATLAB)
  - destroyed when unused
  - explicit deletion: **del** x

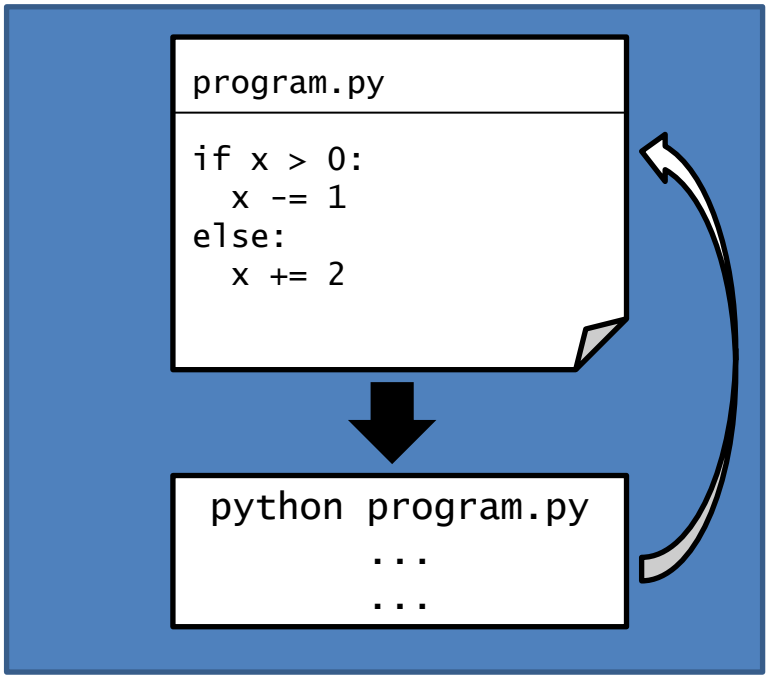
Why Python?

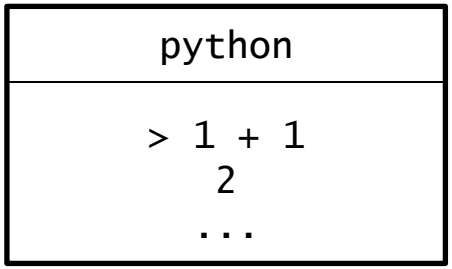
- Availability
- Free software
- Ease of use
- **Structures**
- Ecosystem

Fortran



Python





Python programme

- Bunch of statements (one per line)
- Seperate multiple statements per line by ; (only when necessary)
- Comments: # until end of line

```
x = 0      # sets x to zero
x += 1
y = x; del x
print y
```

If statements

- Leading Whitespace is significant in Python
  - indentation level marks blocks! (no "{" or „BEGIN“ or whatever)
  - typically 4 spaces per level
  - every block starts with „:“, usually no brackets for condition

```
if x == 1:
    print "x is one"
else:
    x –= 1
    print "x minus 1", x
```

```
if x != 1: print "x is not one"                    # short version
```

Conditions

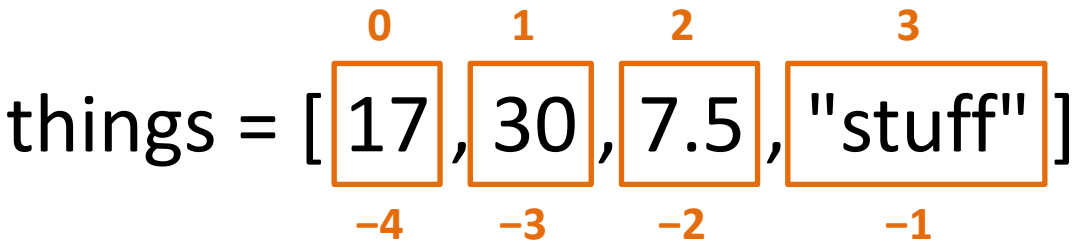
- Comparisons like in C: ==, !=, <, <=, >, >=
- ... and a bit like in Fortran: **and**, **or**, **not**
- Set relations: **in**, **not in**
- Can be stacked like in maths:
  `1 < b <= 3` is equivalent to `1 < b` **and** `b <= 3`
- Truth values: True, False

Lists

- things = [17 , 30 , 7.5 , "stuff"]

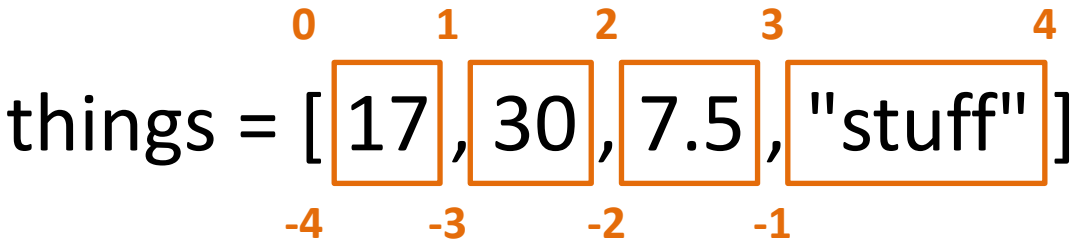
- things.append("foo")
- things.remove("stuff")
- things.pop() → "foo"
- things = [ ]

Indexing



- things[index]
- things[0] → 17
- things[−2] = 13.0

"Slicing"



- things[from (0) : to (last) : stride (1)]
- things[0:3]
- things[2:]
- things[:-2]
- things[::3]

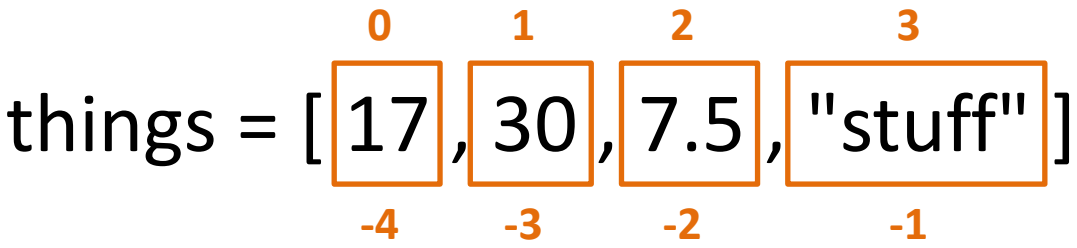
Loops

- for-loop: Block prefixed by **for ... in ... :**
  - Looping always over range of data (no C-for/FORTRAN-do)

```
for item in list:
    print item
```

- while-loop: Block prefixed by **while ... :**
  - There is no do...while

```
x = 1
while x < 10:
    print x
    x *= 2
```



- **for** e **in** things[1::2]:
  print e
- **for** i, e **in** *enumerate*(things):
  things[i] = 2 * e
- **for** index **in** *range*(10): # loop over 0 ... 10
  print index + 1
- **for** line **in** *file*("data.txt"):
  things.append(line)

List comprehensions

- Common use case: generate list from other list

```
# Generate the squares of all numbers from 1 to 9
squares = [ ]
for item in range(1, 10):
    squares.append(item ** 2)
```

- Shorthand notation:

```
squares = [ item**2 for item in range(1, 10) ]
```

- Can be nested and combined with **if** (see docs)

FORTRAN	Python
~~**integer** :: x~~ x = 0	x = 0

if ~~(~~x > 0~~) then~~
  x = x − 1
else
  x = x + 2
~~**endif**~~
	if x > 0 **:**
	  x −= 1
	else:
	  x += 2

do k = 1,10,2
  write (*,*) k
~~**enddo**~~
	for k **in** range(1,11,2):
	  print k

function sinc(x)
  ~~**real** x, sinc~~
  sinc = (sin(x)/x)**2
~~**endfunction**~~
	def sinc(x):
	  return (sin(x)/x)**2

- t = range(2,19,3)
- t.append(3.14)
- t
- t[::2]
- t.*[press TAB]*
- t.pop?
- **if** t[1] < 10:
  print "YES"
- **for** i **in** t[:5]:
  print "value", i+1

play around a bit

make a list with all odd numbers from 11 to 29, but excluding 21 and 27

sort the list [2, 4, 5, 8, 9, 12, 15] into two seperate lists of odd and even numbers

Generate all primes up to 1.000

Other datastructures

- **tuple** (unchangable list):

  t = (1, 2, 3)

  x, y = 1, 4

- **set** (sorted list without duplicates):

  x = {1, 5, 9}

  x | {2, 9}

- **dict** (e.g., string indices):

  d = {"PI": 20, "E": 2.71}

  d["PI"] = 3.14

Why Python?

- Availability
- Free software
- Ease of use
- Structures
- **Ecosystem**

Functions

- Block prefixed by **def** … (…):
  - No argument types (dynamic typing)
  - Default arguments, variable arguments possible (see docs)
  - First line may be a string explaining what the function does

```
def factorial(n):
    "return the factorial of a number"
    if n != int(n) or n < 0:
        raise ValueError()      # error handling
    if n < 2: return 1
    return n * factorial(n-1)
```

Modules

- Collection of functions in a file (e.g., **math.py**)
- Prefixed import

```
import math
print math.sin(math.pi)
```

```
import math as m
print m.sin(m.pi)
```

- Non-prefixed import

```
from math import sin, pi
print sin(pi)
```

batteries included

import ***	[***]. ...
math	sin, pi, sqrt, ...
cmath	*(complex versions)*
sys	argv, exit, stdout, stderr, ...
os	environ, chdir, listdir, ...
random	rand, randint, ...
...	...

included : http://docs.python.org/2/library/
download: https://pypi.python.org/pypi

- **import** random
- random.*[press TAB]*
- **from** random **import** rand
- rand?
- **import** sys
- **import** os
- os.path.*[press TAB]*
- **def** f(x=1):
  "does very little"
  return x + 42
- f(5)
- f()
- f?

play around a bit and read the docs

take one of the previous exercises and turn it into a function; then call that function

fill a list with fifty random numbers, each between 1 and 10

create a new directory and create files named „dat0“ until „dat100“

write a function that sends an e-mail to yourself using Python

Stuff we skipped

- Classes:

```
class Coordinate:
    def __init__(self, x, y):
        self.x = 0
        self.y = 0

    def r(self):
        return math.hypot(
                        self.x, self.y)

c = Coordinate(3, 5)
print c.x, c.y, c.r()
```

- String formatting: %
- Anonymous functions: **lambda** x: x**2
- Generators: **yield**
- Decorators: @
- Testing (...)