Python for Astronomers

Reinhold Schaaf & Manuel Metz

Argelander-Institut für Astronomie
der Universität Bonn

Lingua franca for astromoners

- Easy to learn but yet powerful
- Fast and easy solutions for everyday problems
- Allows development of large and sophisticated software packages
- Several astronomical program packages implemented using Python
- Excellent tool for scientists

Python in Astronomy

Programs and libraries implemented (mainly) in Python:

- BoA (Bolometer Analysis Package)
- APECS (APEX control software)
- Astro-WISE (widefield imaging system)
- PyFITS (access to FITS files)
- ...

Python in Astronomy

Wrappers for existing programs and libraries:

- CasaPy (Casa)
- PYGILDAS (GILDAS)
- ParselTongue (AIPS)
- PyRAF (IRAF)
- PyMIDAS (MIDAS)
- PyIMSL (IMSL)
- .....

Fortran, C (and C++)

- Working horses of science
- zillions of LOC existing
- optimized for runtime performance
- no reason to replace them (at this time)
- rather cooperate (swig, f2py)

but...

- not really general purpose
- relatively primitive datatypes
- manual memory management
- slow edit/compile/test cycle

IDL, Matlab, Mathematica

- Extremely popular these days
- Interactive, great visualization, good libraries

but...

- Not really general purpose
- Not really ready for large scale programs
- Not free, even expensive

Perl, Ruby, shell scripts

- Perl very strong at text manipulation
- Ruby seems to be popular in scientific community in Japan
- Shell scripts... you will not want to do shell scripting any more once you know Python (unless forced)!

This course - audience

- AIfA / MPI: 20/21
- from master student to senior staff members
- most have at least basic experience in one or more computer languages
- 16 are interested in course project

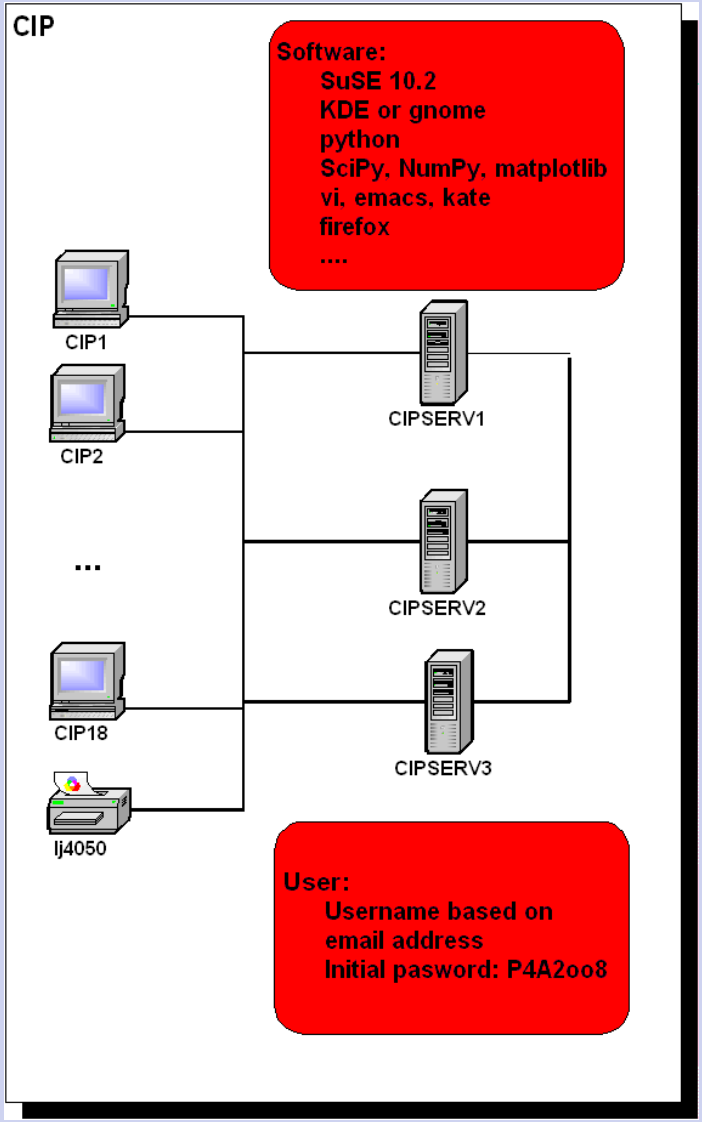
This course – program

- Give you what you need for everyday problems asap
- First half of course: Basic language elements, some modules from standard library, elements of NumPy, SciPy and matplotlib
- Second half of course: Advanced topics from Python (e.g. exceptions, classes), from NumPy, SciPy and matplotlib
- Course project will start after first half

This course - setup

- Show, tell & try approach during classes
- Exercises during classes
- No assignments between classes...
- ...but you should use what you have!
- Course project to be defined

This course – CIP room



- Terminal client below your desk
- Select server according to your row
- Select KDE or gnome
- Please switch off screen and terminal client at end of class!

Exercises

- Login to server (Password: P4A2oo8)
- Change your password!!!
- Take a look at the website of the course www.astro.uni-bonn.de/~rschaaf/Python2008
- Explore the official Python website www.python.org
- Determine the installed python version and locate the documentation for this version at www.python.org

Numbers, variables, math and while

- Python's numerical types
- Operators for numerical types
- Variables, variable names, and assignment to variables
- The math module from the Standard Library
- Loops with the while statement

Integers

```
>>> 2
>>> 0
>>> -4711
>>> 07, 022          # Octal literals
>>> 0x9, 0xa, 0XF # Hex literals
>>> 17 + 4           # expression
>>> 0xa - 2
>>> 23 ** (2+3)
>>> 7 / 2, 7 / -2 # Int division
```

Floats

```
>>> 2.3
>>> -4.
>>> 0.1, .1
>>> 2.99E10, 6.62607e-27, -1e10
>>> 1.7 + .4
>>> 17. + 4
>>> 7./2., 7./2, 7/2.
```

Arithmetic operators

```
>>> 1+.4
>>> 2-3
>>> 25*-17
>>> 2.**100
>>> 2/3, 2/-3    # Int division
>>> 2./3         # Float division
>>> 2.//3, 2.//3 # Floor division
>>> 7%3, 7%-3    # Modulus
>>> 7.%3, 7.2%2.5 # Modulus
>>> # (n//m)*m + (n%m) = n
```

Bitwise operators

```
>>> 17 & 5     # Bitwise AND
>>> 17 | 5     # Bitwise OR
>>> 17 ^ 5     # Bitwise EXOR
>>> ~17         # Bitwise complement
>>> 17 << 3    # Bitshift left
>>> 0x11 >> 2 # Bitshift right
```

Long integers, conversion

```
>>> 2**1000

>>> 2L, 3l

>>>1111111111111111111111111111111111111111
  11111111111111111

>>> 2**1001    2<<1000   # - has higher

>>> 2**1001    (2<<1000) # precedence than <<

>>> float(2), float(2**1000)

>>> float(2**2000)

>>> int(2.3), int(-2.3)

>>> int(2**1000), long(2)
```

Complex numbers

```
>>> 2.+3j, 2-3J # complex literals
>>> j           # will not work
>>> 1J          # but this will
>>> complex(1,2)
>>> # Watch operator precedence:
>>> 1+1j*2, (1+1j)*2
>>> (2.+3j).real, (2+3j).imag
```

Variables

```
>>> x = 2       # Assign variable
>>> x           # Display
>>> x + 3       # Use variable
>>> y = x + 3 # New variable
>>> x = x + 1 # Assign new value
>>> x += 1      # Shorthand; no x++
>>> x = 12.3 + 98.7j # Change type
>>> x **= 2j
```

Some stunts

```
>>> x, y = 2, 3
>>> x, y = y, x
>>> x = y = z = 0
```

Names

```
>>> xy, Xy = 2, 3 # Case sensitive
>>> 9x = 2 # Must begin w. letter
>>> x9 = 2 # ok
>>> _x = 2 # ok, but special
>>> if = 2 # must not be keyword
```

List of reserved keywords

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
None
as        with
```

Built-in math functions

```
>>> abs(-2.)
>>> abs(1+1j)
>>> max(1,2,3,4)
>>> min(1,2,3,4)
>>> hex(17), oct(-5)
>>> round(1.23456, 2)
```

The math module

```
>>> import math # math module
>>> math.<TAB>  # Explore with tab
>>> # Print doc strings:
>>> print math.__doc__
>>> print math.sinh.__doc__
>>> # Use module:
>>> math.sinh(math.pi/2)
>>> math.e**(-1j*math.pi)
>>> math.pi = 3   # Abuse! No way back!
```

Comparisons

```
>>> x = 2     # Assignment
>>> x == 2    # Testing equality
>>> x != 0
>>> x < 1
>>> x >= 2.
>>> 1 < x <= 3 # (1<x) and (x<=3)
>>> 3 < x <= 1
>>> 1 < x >= 0
```

The while statement

```
>>> x = 0
>>> while x<=10:     # Bool. expr.
...     print x, x**2 # Indentation
...     x += 1
... <RETURN>
>>> print x     # Unindented again
```

The while statement

```
>>> while x:   # Arithmetic expr.
...   print x
...   x -= 1
... <RETURN>

>>> x = 1
>>> while x:
...   print x
... <RETURN> # Abort with <Ctrl>-c
```

Nested while statements

```
>>> i = j = 0
>>> while i<=5:
...     j = 0          # outer loop
...     while j<=i:
...         print j-i, # inner loop
...         j += 1     # inner loop
...     i += 1         # outer loop
...     print          # outer loop
... <RETURN>
```

Indentation

- Top level must not be indented
- It does not matter how much blanks you use, but:
- Uniform within each block
- Better avoid tabs
- Most people use 4 blanks per level

Exercises

- Determine the smallest positive float of our Python installation by using a while-loop
- Determine the numerical accuracy of the installation. Add smaller and smaller numbers to 1. and compare the result with 1.
- Determine the largest positive float of the installation. Import numpy and compare larger and larger numbers with numpy.inf