

Scientific use of Python

TIAGO M. D. PEREIRA
RSAA



Feast of Facts, July 28, 2006



Outline

- Python? What is Python?
- Programming in Python in 5 minutes
- Scientific Tools and Libraries
- Why Python?

Python? What is Python?

What is Python?

What is Python?

“What’s Python?”

“It’s a computer programming language.”

“You mean, like DOS?”

–Some guy in a bar and Eric S. Raymond (who was wearing a conference T-shirt) at IPC7

What is Python?

What is Python?

- High level, **interpreted**, interactive, object-oriented programming language
- Very clear and readable syntax
- Powerful, modular, extensive libraries available
- Extendable, easy to use C/C++/FORTRAN modules for heavier tasks
- Cross-platform: run your code in any OS!

What is Python? History

What is Python? History

- Invented by Guido van Rossum in the early 1990's
- Name comes from Monty Python

What is Python?

What is Python?

Python's syntax succeeds in combining the mistakes of Lisp and Fortran. I do not construe that as progress.

–Larry Wall, May 12 2004

Things in Python are very clear, but are harder to find than the secrets of wizards. Things in Perl are easy to find, but look like arcane spells to invoke magic.

–Mike Meyer

Where is python used?

- Google
- BitTorrent
- Zope web engine
- Linux distributions (Red Hat, Gentoo, etc.)
- NASA
- Hubble Space Telescope

Main uses of Python

- Internet, Web & Databases
- Network Programming, Software Development
- Desktop GUIs
- Scientific

Main uses of Python

- Internet, Web & Databases
- Network Programming, Software Development
- Desktop GUIs
- Scientific
 - ★ Number crunching
 - ★ **Data Analysis**
 - ★ Scripting

Main uses of Python

- Internet, Web & Databases
- Network Programming, Software Development
- Desktop GUIs
- Scientific
 - ★ Number crunching
 - ★ **Data Analysis**
 - ★ Scripting

<http://www.python.org/about/apps/>

Python Programming Language: Technicalities

Python Programming Language: Technicalities

- **Indentation delimits blocks** – not “free” format!

Python Programming Language: Technicalities

- **Indentation delimits blocks** – not “free” format!
- Weakly typed

Python Programming Language: Technicalities

- **Indentation delimits blocks** – not “free” format!
- Weakly typed
- Multi-paradigm: several coding styles:
 - ★ object oriented programming (C++, Java)
 - ★ structured programming
 - ★ functional programming (Lisp)
 - ★ aspect oriented programming (C#, Java)

Python Programming Language: Technicalities

- **Indentation delimits blocks** – not “free” format!
- Weakly typed
- Multi-paradigm: several coding styles:
 - ★ object oriented programming (C++, Java)
 - ★ structured programming
 - ★ functional programming (Lisp)
 - ★ aspect oriented programming (C#, Java)
- Dynamically typed

Python Programming Language: Technicalities

- **Indentation delimits blocks** – not “free” format!
- Weakly typed
- Multi-paradigm: several coding styles:
 - ★ object oriented programming (C++, Java)
 - ★ structured programming
 - ★ functional programming (Lisp)
 - ★ aspect oriented programming (C#, Java)
- Dynamically typed
- Automatic memory management: garbage collector

Programming in Python in 5 minutes

Hello world

C:

```
#include <stdio.h>
main()
{
    for(;;)
    {
        printf(“Hello World!\n”);
    }
}
```

Hello world

C:

```
#include <stdio.h>
main()
{
    for(;;)
    {
        printf(“Hello World!\n”);
    }
}
```

Fortran:

```
PROGRAM HELLO
LOGICAL DONE
DO WHILE (.NOT. DONE)
WRITE(*,*) 'Hello World!'
END DO
END
```

Hello world

C:

```
#include <stdio.h>
main()
{
    for(;;)
    {
        printf(“Hello World!\n”);
    }
}
```

Fortran:

```
PROGRAM HELLO
LOGICAL DONE
DO WHILE (.NOT. DONE)
WRITE(*,*) 'Hello World!'
END DO
END
```

Python:

```
while True:
    print 'Hello World!'
```

Running Python

Running Python

- Command line:
\$ python -c "print 'Hello World'"
Hello World

Running Python

- Command line:

```
$ python -c "print 'Hello World'"  
Hello World
```

- Interactive prompt (or ipython):

```
$ python  
Python 2.4.3 (#2, Apr 27 2006, 14:43:58)  
[GCC 4.0.3 (Ubuntu 4.0.3-1ubuntu5)] on linux2  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>>
```

Running Python

- Command line:

```
$ python -c "print 'Hello World'"  
Hello World
```

- Interactive prompt (or ipython):

```
$ python  
Python 2.4.3 (#2, Apr 27 2006, 14:43:58)  
[GCC 4.0.3 (Ubuntu 4.0.3-1ubuntu5)] on linux2  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>>
```

- Run file (or as a script):

```
$ python myfile.py  
$ ./myfile.py
```

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

Python as a calculator

- Any expression in the interactive prompt is readily evaluated:

Python as a calculator

- Any expression in the interactive prompt is readily evaluated:

```
>>> 5.05e-34/34.  
1.4852941176470588e-35
```

Python as a calculator

- Any expression in the interactive prompt is readily evaluated:

```
>>> 5.05e-34/34.  
1.4852941176470588e-35
```

- Importing modules for more operations:

Python as a calculator

- Any expression in the interactive prompt is readily evaluated:

```
>>> 5.05e-34/34.  
1.4852941176470588e-35
```

- Importing modules for more operations:

```
>>> import math  
>>> math.sinh(5.5)*math.sqrt(2.)  
173.0204348218723
```

Python as a calculator

- Any expression in the interactive prompt is readily evaluated:

```
>>> 5.05e-34/34.  
1.4852941176470588e-35
```

- Importing modules for more operations:

```
>>> import math  
>>> math.sinh(5.5)*math.sqrt(2.)  
173.0204348218723
```

or

```
>>> from math import *  
>>> 5*sin(4*pi*0.07)+exp(-3.8)*log10(1.e300)  
10.563797770728627
```

Native Data Types

- Numbers (integer, floating point, long integer, complex)
- Strings (single or double quoted, triple quoted)
- Lists
- Dictionaries
- Tuples (immutable lists)
- **Arrays** (not native, added by extension modules)
- Build your own datatype!

Lists

Lists

- Consist of sequences of any datatype

```
a=[1,2,3,4,5]
```

```
b=[1.,2,'s6',['1st','2nd'],{'a':3,'b':4}]
```

Lists

- Consist of sequences of any datatype

```
a=[1,2,3,4,5]
```

```
b=[1.,2,'s6',['1st','2nd'],{'a':3,'b':4}]
```

- List slicing and manipulation:

```
>>> c=range(10) ; print c
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lists

- Consist of sequences of any datatype

```
a=[1,2,3,4,5]
```

```
b=[1.,2,'s6',['1st','2nd'],{'a':3,'b':4}]
```

- List slicing and manipulation:

```
>>> c=range(10) ; print c
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> c[0:3]
```

```
[0, 1, 2]
```

Lists

- Consist of sequences of any datatype

```
a=[1,2,3,4,5]
```

```
b=[1.,2,'s6',['1st','2nd'],{'a':3,'b':4}]
```

- List slicing and manipulation:

```
>>> c=range(10) ; print c
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> c[0:3]
```

```
[0, 1, 2]
```

```
>>> c[-3:]
```

```
[7, 8, 9]
```

Lists

- Consist of sequences of any datatype

```
a=[1,2,3,4,5]
```

```
b=[1.,2,'s6',['1st','2nd'],{'a':3,'b':4}]
```

- List slicing and manipulation:

```
>>> c=range(10) ; print c
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> c[0:3]
```

```
[0, 1, 2]
```

```
>>> c[-3:]
```

```
[7, 8, 9]
```

```
>>> c[::-1]
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Lists

- Iterate through a list (for loops always iterate in a list)

```
for item in c:           for i in range(len(c)):  
    print item           print c[i]  
(...)                  (...)
```

Lists

- Iterate through a list (for loops always iterate in a list)

```
for item in c:           for i in range(len(c)):  
    print item           print c[i]  
(...)                 (...)
```

- `>>> g = [1,2,3] ; g*2`

Lists

- Iterate through a list (for loops always iterate in a list)

```
for item in c:           for i in range(len(c)):  
    print item           print c[i]  
(...)                 (...)
```
- `>>> g = [1,2,3] ; g*2`
`[1, 2, 3, 1, 2, 3] → Warning! Lists are not arrays!`

Lists

- Iterate through a list (for loops always iterate in a list)

```
for item in c:           for i in range(len(c)):  
    print item           print c[i]  
(...)                 (...)
```
- ```
>>> g = [1,2,3] ; g*2
[1, 2, 3, 1, 2, 3] → Warning! Lists are not arrays!
```
- List methods:  

```
>>> g.append(8) ; g
[1,2,3,8]
```

# Lists

- Iterate through a list (for loops always iterate in a list)  

```
for item in c: for i in range(len(c)):
 print item print c[i]
(...) (...)
```
- ```
>>> g = [1,2,3] ; g*2  
[1, 2, 3, 1, 2, 3] → Warning! Lists are not arrays!
```
- List methods:

```
>>> g.append(8) ; g  
[1,2,3,8]  
>>> g.pop(3)  
8
```

Lists

- Iterate through a list (for loops always iterate in a list)

```
for item in c:           for i in range(len(c)):  
    print item           print c[i]  
(...)                 (...)
```
- ```
>>> g = [1,2,3] ; g*2
[1, 2, 3, 1, 2, 3] → Warning! Lists are not arrays!
```
- List methods:  

```
>>> g.append(8) ; g
[1,2,3,8]
>>> g.pop(3)
8
>>> g.insert(2,'chewbacca') ; g
[1, 2, 'chewbacca', 3]
```

## Dictionaries and tuples

- Dictionaries: lists that can be indexed by strings

## Dictionaries and tuples

- Dictionaries: lists that can be indexed by strings

```
>>> d={'name':'Vader','age':55,'lightsaber':'red'}
```

## Dictionaries and tuples

- Dictionaries: lists that can be indexed by strings

```
>>> d={'name':'Vader','age':55,'lightsaber':'red'}
>>> d['lightsaber']
'red'
```

## Dictionaries and tuples

- Dictionaries: lists that can be indexed by strings

```
>>> d={'name':'Vader','age':55,'lightsaber':'red'}
>>> d['lightsaber']
'red'
>>> d.keys()
['age', 'name', 'lightsaber']
```

## Dictionaries and tuples

- Dictionaries: lists that can be indexed by strings

```
>>> d={'name':'Vader','age':55,'lightsaber':'red'}
>>> d['lightsaber']
'red'
>>> d.keys()
['age', 'name', 'lightsaber']
```
- Tuples: immutable lists (same manipulation, but can't replace elements)

## Dictionaries and tuples

- Dictionaries: lists that can be indexed by strings

```
>>> d={'name':'Vader','age':55,'lightsaber':'red'}
>>> d['lightsaber']
'red'
>>> d.keys()
['age', 'name', 'lightsaber']
```
- Tuples: immutable lists (same manipulation, but can't replace elements)  
**Not very useful for most tasks**

# Functions

- The def statement

```
def myprogram(...):
 (...)
 return (...)
```

# Functions

- The def statement

```
def myprogram(...):
 (...)
 return (...)
```
- Sum example

```
def sum(a,b):
 return a+b
```

# Functions

- The def statement

```
def myprogram(...):
 (...)
 return (...)
```

- Sum example

```
def sum(a,b):
 return a+b
```

```
>>> sum(3,4)
```

```
7
```

## Functions

- The def statement

```
def myprogram(...):
 (...)
 return (...)
```

- Sum example

```
def sum(a,b):
 return a+b
```

```
>>> sum(3,4)
```

```
7
```

```
>>> sum('Luke ', 'Skywalker')
'Luke Skywalker'
```

```
>>> sum('Luke',55)
```

```
Traceback (most recent call last):
```

```
 File "<stdin>", line 1, in ?
```

```
 File "<stdin>", line 2, in sum
```

```
TypeError: cannot concatenate 'str' and
'int' objects
```

# Exception Handling

- If something goes wrong, react, don't explode

## Exception Handling

- If something goes wrong, react, don't explode
- Know when something can go wrong
- Controlled explosion
- Take measures: **try** and **except** commands

## Exception Handling

- The sum example:

```
>>> try:
... sum('Luke', 55)
>>> except TypeError:
... print 'The Force runs strong in you'
```

## Exception Handling

- The sum example:

```
>>> try:
... sum('Luke', 55)
>>> except TypeError:
... print 'The Force runs strong in you'
The Force runs strong in you
```

# Scientific tools and libraries

## Scientific Libraries for Python

- Standard Library (written in C; generic; quite big)
- Fast Array Libraries (Numeric, `numarray`, NumPy)
- Plotting Libraries (`plplot`, `matplotlib`, `vpython`, etc.)
- Scientific Python (SciPy)
- Astronomical tools:
  - ★ PyFITS: for handling FITS images
  - ★ PyRAF/PyMIDAS: command language for IRAF/MIDAS based in Python
  - ★ AstroLib: same as for IDL, still incomplete

# Array Libraries

- Numeric: first array package, fastest for small ( $< 20.000$ ) arrays

## Array Libraries

- Numeric: first array package, fastest for small ( $< 20.000$ ) arrays
- numarray: developed by STSI to replace IDL, fastest for big arrays ( $> 20.000$ )

## Array Libraries

- Numeric: first array package, fastest for small ( $< 20.000$ ) arrays
- numarray: developed by STSI to replace IDL, fastest for big arrays ( $> 20.000$ )
- numarray: used at the HST data pipeline (Advanced Camera for Surveys; Cosmic Origins Spectrograph)

## Array Libraries

- Numeric: first array package, fastest for small ( $< 20.000$ ) arrays
- numarray: developed by STSI to replace IDL, fastest for big arrays ( $> 20.000$ )
- numarray: used at the HST data pipeline (Advanced Camera for Surveys; Cosmic Origins Spectrograph)
- NumPy: future merger of Numeric and numarray

## Array Libraries

- Numeric: first array package, fastest for small ( $< 20.000$ ) arrays
- numarray: developed by STSI to replace IDL, fastest for big arrays ( $> 20.000$ )
- numarray: used at the HST data pipeline (Advanced Camera for Surveys; Cosmic Origins Spectrograph)
- NumPy: future merger of Numeric and numarray
- Written in C and Fortran, take advantage of fast algorithms (eg., LAPACK for linear algebra)

## Array Libraries

- Numeric: first array package, fastest for small ( $< 20.000$ ) arrays
- numarray: developed by STSI to replace IDL, fastest for big arrays ( $> 20.000$ )
- numarray: used at the HST data pipeline (Advanced Camera for Surveys; Cosmic Origins Spectrograph)
- NumPy: future merger of Numeric and numarray
- Written in C and Fortran, take advantage of fast algorithms (eg., LAPACK for linear algebra)
- More than arrays (similar capabilities to Matlab, IDL, Octave): FFT, Random Numbers, Linear Algebra, etc.

## Array Magic

```
>>> from numpy import *
```

## Array Magic

```
>>> from numpy import *
>>> a=arange(6.) ; a
array([0., 1., 2., 3., 4., 5.])
```

## Array Magic

```
>>> from numarray import *
>>> a=arange(6.) ; a
array([0., 1., 2., 3., 4., 5.])
>>> 2*a + 3.
array([3., 5., 7., 9., 11., 13.])
```

## Array Magic

```
>>> from numpy import *
>>> a=arange(6.) ; a
array([0., 1., 2., 3., 4., 5.])
>>> 2*a + 3.
array([3., 5., 7., 9., 11., 13.])
>>> cos(a/2)
array([1. , 0.87758256, 0.54030231,
 0.0707372 , -0.41614684, -0.80114362])
```

## Array Magic

```
>>> from numpy import *
>>> a=arange(6.) ; a
array([0., 1., 2., 3., 4., 5.])
>>> 2*a + 3.
array([3., 5., 7., 9., 11., 13.])
>>> cos(a/2)
array([1. , 0.87758256, 0.54030231,
 0.0707372 , -0.41614684, -0.80114362])
>>> a.reshape(3,2) ; a
array([[0., 1.],
 [2., 3.],
 [4., 5.]])
```

## Matrix Magic

```
>>> a = arange(9., shape=(3,3)) ; a
array([[0., 1., 2.],
 [3., 4., 5.],
 [6., 7., 8.]])
```

## Matrix Magic

```
>>> a = arange(9., shape=(3,3)) ; a
array([[0., 1., 2.],
 [3., 4., 5.],
 [6., 7., 8.]])

>>> b = identity(3) * 2. ; b
array([[2., 0, 0],
 [0, 2., 0],
 [0, 0, 2.]])
```

## Matrix Magic

```
>>> a = arange(9., shape=(3,3)) ; a
array([[0., 1., 2.],
 [3., 4., 5.],
 [6., 7., 8.]])

>>> b = identity(3) * 2. ; b
array([[2., 0, 0],
 [0, 2., 0],
 [0, 0, 2.]])

>>> c = matrixmultiply(a,b) ; c
array([[0., 2., 4.],
 [6., 8., 10.],
 [12., 14., 16.]])
```

## Matrix Magic 2

```
>>> from LinearAlgebra import *
```

## Matrix Magic 2

```
>>> from LinearAlgebra import *
>>> transpose(c)
array([[0., 6., 12.],
 [2., 8., 14.],
 [4., 10., 16.]])
```

## Matrix Magic 2

```
>>> from LinearAlgebra import *
>>> transpose(c)
array([[0., 6., 12.],
 [2., 8., 14.],
 [4., 10., 16.]])
>>> eigenvalues(b) # (b is identity * 2)
array([2., 2., 2.])
```

## Matrix Magic 2

```
>>> from LinearAlgebra import *
>>> transpose(c)
array([[0., 6., 12.],
 [2., 8., 14.],
 [4., 10., 16.]])
>>> eigenvalues(b) # (b is identity * 2)
array([2., 2., 2.])
>>> eigenvectors(b)[1]
array([[1., 0., 0.],
 [0., 1., 0.],
 [0., 0., 1.]])
```

**Ok, but what about graphics?**

## Ok, but what about graphics?

- **PLplot:**

- ★ Robust, high quality graphics (based on pgplot)
- ★ Mature, bindings for several languages (C, Fortran, Java, Perl, Python, etc.)
- ★ 2D & 3D

## Ok, but what about graphics?

- **PLplot:**

- ★ Robust, high quality graphics (based on pgplot)
- ★ Mature, bindings for several languages (C, Fortran, Java, Perl, Python, etc.)
- ★ 2D & 3D

- **matplotlib:**

- ★ Interactive (based on matlab)
- ★ Still in development, built for Python
- ★ 2D only

# PLplot examples

# matplotlib examples

## 3D fun: real-time

- VPython: 3D Programming for Ordinary Mortals
  - ★ Real time 3D output
  - ★ Easy to use

# Why Python?

## Why Python?

- Easy to use, easy to code, highly readable

## Why Python?

- Easy to use, easy to code, highly readable
- Open Source, community driven (huge collection of libraries, lots of documentation)

## Why Python?

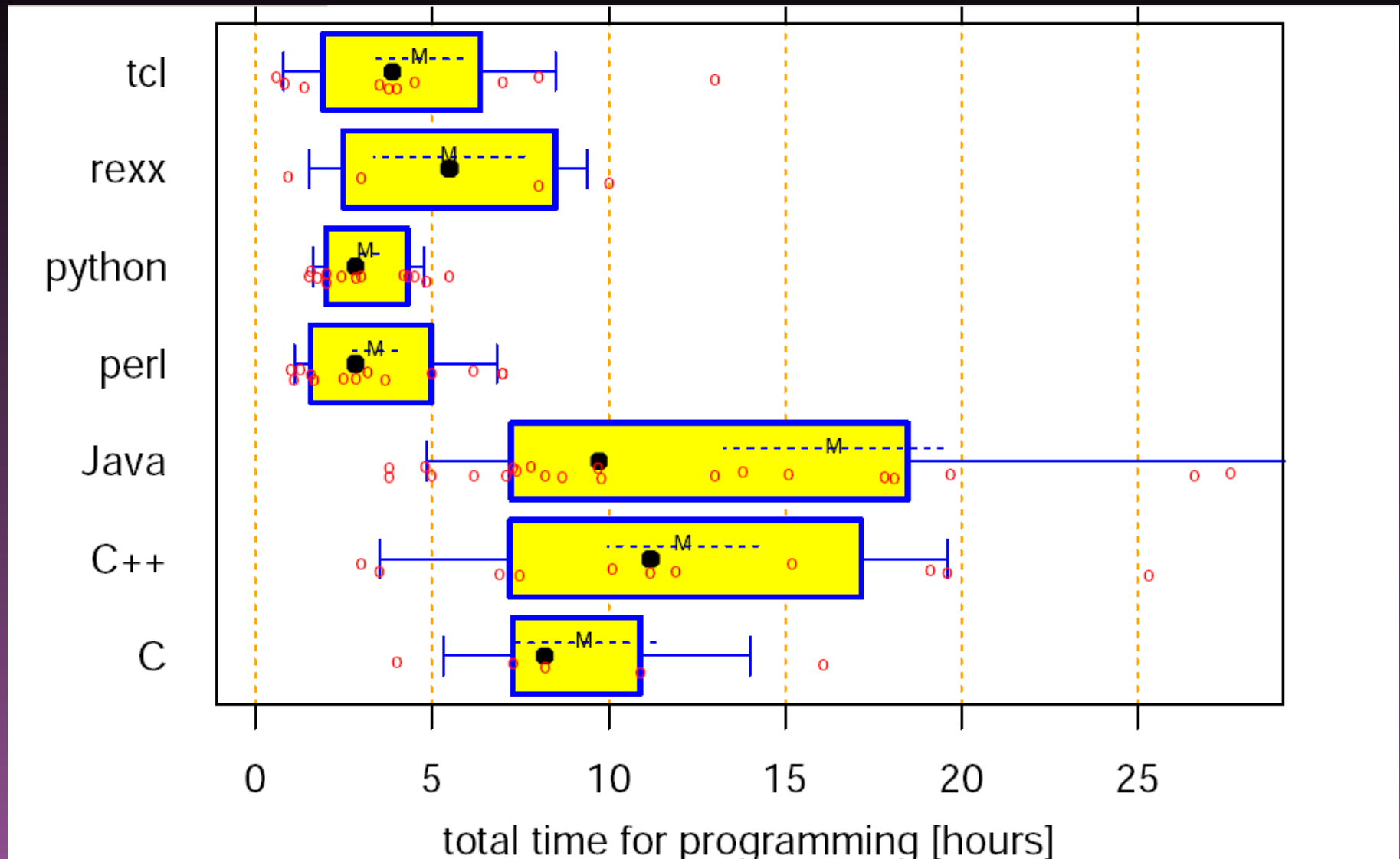
- Easy to use, easy to code, highly readable
- Open Source, community driven (huge collection of libraries, lots of documentation)
- Numeric modules bring array processing as fast as compiled languages (time critical loops C or Fortran coded)

## Why Python?

- Easy to use, easy to code, highly readable
- Open Source, community driven (huge collection of libraries, lots of documentation)
- Numeric modules bring array processing as fast as compiled languages (time critical loops C or Fortran coded)
- Truly multiplatform (runs where's a C compiler)

## Why Python?

- Easy to use, easy to code, highly readable
- Open Source, community driven (huge collection of libraries, lots of documentation)
- Numeric modules bring array processing as fast as compiled languages (time critical loops C or Fortran coded)
- Truly multiplatform (runs where's a C compiler)
- Free as in free beer!



From *Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java*. A chapter for “Advances in Computers”, Volume 58, 2002.

## Python vs. IDL

| Python                               | IDL                                           |
|--------------------------------------|-----------------------------------------------|
| Free!                                | Expensive                                     |
| Community driven                     | Dependant on 3rd party companies              |
| Use anywhere                         | Subject to license hassle                     |
| Fractionary: many redundant modules  | Complete package                              |
| Science use still starting           | Bigger code base for science                  |
| Robust and modular graphic libraries | Limited to existing graphics system           |
| Modern, simple and robust syntax     | Still inherits some shortcomings from Fortran |
| Web-based documentation              | Integrated help system                        |

# Where to start?

## Where to start?

- Python tutorial:  
<http://docs.python.org/tut/>
- Dive into python: (book)  
<http://diveintopython.org>
- Python for Science:  
<http://starship.python.net/crew/hinsen/>
- Python in Astronomy  
<http://www.astro.washington.edu/owen/AstroPy.html>

## Python according to Yoda

YODA: Code! Yes. A programmer's strength flows from code maintainability. But beware of Perl. Terse syntax... more than one way to do it... default variables. The dark side of code maintainability are they. Easily they flow, quick to join you when code you write. If once you start down the dark path, forever will it dominate your destiny, consume you it will.

LUKE: Is Perl better than Python?

YODA: No... no... no. Quicker, easier, more seductive.

LUKE: But how will I know why Python is better than Perl?

YODA: You will know. When your code you try to read six months from now.