Python: A great programming toolkit

Asokan Pichai
Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

10, August 2009

Acknowledgements

This program is conducted by
IIT, Bombay
through CDEEP
as part of the open source initiatives
under the aegis of
National Mission on Education through ICT,
Ministry of HRD.

Outline

About the Workshop

Day 1, Session 1	Sat 09:30–11:00
Day 1, Session 2	Sat 11:15–12:45
Day 1, Session 3	Sat 13:45–15:15
Day 1, Session 4	Sat 15:30–17:00
Day 2, Session 1	Sun 09:30–11:00
Day 2, Session 2	Sun 11:15–12:45
Day 2, Session 3	Sun 13:45–15:15
Day 2, Session 4	Sun 15:30–17:00

About the Workshop

Goal of the workshop

Successful participants will be able to use python as their scripting and problem solving language.

Workshop Audience

Aimed at Engg., Mathematics and Science teachers but should serve a similar purpose for others.

Focus of the workshop

Focus on basic numerics and plotting

Checklist

python

Type python at the command line. Do you see version 2.5 or later?

IPython

Is IPython available?

Editor

Which editor? scite, vim, emacs, ...

Session 1

- Introduction and motivation
- Using the interpreter(s)
- Basic data types: int, float, string
- Basic data structures: list
- Basic console IO: `raw_input()`, `print`
- Basic control flow: `if`, `while`
- Problem set 1
- Functions $\longrightarrow$ Problem set 2
- lists, `for` $\longrightarrow$ Problem set 3
- IO, Modules $\longrightarrow$ Problem sets 4,5, . . .

Introduction

- Creator and BDFL: Guido van Rossum
- December 1989
- “Python” as in Monty Python’s Flying Circus
- 2.6.x
- PSF license (like BSD: no strings attached)
- Highly cross platform
- Nokia series 60!
- Philosophy: Simple and complete by design

Resources

- Part of many GNU/Linux distributions
- Web: `http://www.python.org`
- Doc: `http://www.python.org/doc`
- Free Tutorials:
  - Official Python tutorial: `http://docs.python.org/tut/tut.html`
  - Byte of Python: `http://www.byteofpython.info/`
  - Dive into Python: `http://diveintopython.org/`

Why Python?

- Readable and easy to use
- High level, interpreted, modular, OO
- Much faster development cycle
- Powerful interactive environment
- Rapid application development
- Rich standard library and modules
- Interfaces well with C++, C and FORTRAN
- More than a math package $\Rightarrow$ some extra work compared to math packages

Use cases

- NASA: Space Shuttle Mission Design
- AstraZeneca: Collaborative Drug Discovery
- ForecastWatch.com: Helps Meteorologists
- Industrial Light & Magic: Runs on Python
- Zope: Commercial grade Toolkit
- Plone: Professional high feature CMS
- RedHat: install scripts, sys-admin tools
- Django: A great web application framework
- Google: A strong python shop

To sum up, python is. . .

- dynamically typed, interpreted $\longrightarrow$ rapid testing/prototyping
- powerful, very high level
- has full introspection
- Did we mention powerful?

But . . .

may be wanting in performance. specialised resources such as SWIG, Cython are available

15 m

Outline

At the prompt, type the following

```
>>> print 'Hello Python'
>>> print 3124 * 126789
>>> 1786 % 12
>>> 3124 * 126789
>>> a = 3124 * 126789
>>> big = 12345678901234567890 ** 3
>>> verybig = big * big * big * big
>>> 12345**6, 12345**67, 12345**678
```

At the prompt, type the following

```
>>> s = 'Hello '
>>> p = 'World'
>>> s + p
>>> s * 12
>>> s * s
>>> s + p * 12, (s + p)* 12
>>> s * 12 + p * 12
>>> 12 * s
```

At the prompt, type the following

```
>>> 17/2
>>> 17/2.0
>>> 17.0/2
>>> 17.0/8.5
>>> int(17/2.0)
>>> float(17/2)
>>> str(17/2.0)
>>> round( 7.5 )
```

Mini exercise

Round a float to the nearest integer, using `int()`?

Midi exercises

- What does this do?
- `round(amount * 10) /10.0`

More exercises?

Round sums

How to round a number to the nearest 5 paise?

Remember $17.23 \rightarrow 17.25$,

while $17.22 \rightarrow 17.20$

How to round a number to the nearest 20 paise?

A question of good style

```
amount = 12.68
denom = 0.05
nCoins = round(amount/denom)
rAmount = nCoins * denom
```

Style Rule #1

Naming is 80% of programming

A question of good style

```
amount = 12.68
denom = 0.05
nCoins = round(amount/denom)
rAmount = nCoins * denom
```

Style Rule #1

Naming is 80% of programming

Odds and ends

- Case sensitive
- Dynamically typed $\Rightarrow$ need not specify a type

```
a = 1
a = 1.1
a = "Now I am a string!"
```

- Comments:

```
a = 1   # In-line comments
# Comment in a line to itself.
a = "# This is not a comment!"
```

Outline

Basic types

- Numbers: float, int, long, complex
- Strings
- Boolean

Also to be discussed later

tuples, lists, dictionaries, functions, objects. . .

Numbers

```
>>> a = 1 # Int.
>>> l = 1000000L # Long
>>> e = 1.01325e5 # float
>>> f = 3.14159 # float
>>> c = 1+1j # Complex!
>>> print f*c/a
(3.14159+3.14159j)
>>> print c.real, c.imag
1.0 1.0
>>> abs(c)
1.4142135623730951
>>> abs( 8 - 9.5 )
1.5
```

Boolean

```
>>> t = True
>>> f = not t
False
>>> f or t
True
>>> f and t
False
```

Try:

NOT True
not TRUE

Relational and logical operators

```
>>> a, b, c = -1, 0, 1
>>> a == b
False
>>> a <= b
True
>>> a + b != c
True
>>> a < b < c
True
>>> c >= a + b
True
```

Strings

```
s = 'this is a string'
s = 'This one has "quotes" inside!'
s = "I have 'single-quotes' inside!"
l = "A string spanning many lines\
one more line\
yet another"
t = """A triple quoted string does
not need to be escaped at the end and
"can have nested quotes" etc."""
```

More Strings

```
>>> w = "hello"
>>> print w[0] + w[2] + w[-1]
hlo
>>> len(w) # guess what
5
>>> s = u'Unicode strings!'
>>> # Raw strings (note the leading 'r'
... r_s = r'A string $\alpha \nu$'

>>> w[0] = 'H' # Can't do that!
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item
```

More Strings

```
>>> w = "hello"
>>> print w[0] + w[2] + w[-1]
hlo
>>> len(w) # guess what
5
>>> s = u'Unicode strings!'
>>> # Raw strings (note the leading 'r'
... r_s = r'A string $\alpha \nu$'

>>> w[0] = 'H' # Can't do that!
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item
```

Let us switch to IPython

Why?

Better help (and a lot more)

Tab completion

?

.?

object.function?

More on strings

```
In [1]: a = 'hello world'
In [2]: a.startswith('hell')
Out[2]: True
In [3]: a.endswith('ld')
Out[3]: True
In [4]: a.upper()
Out[4]: 'HELLO WORLD'
In [5]: a.upper().lower()
Out[5]: 'hello world'
```

Still with strings

```
In [6]: a.split()
Out[6]: ['hello', 'world']
In [7]: ''.join(['a', 'b', 'c'])
Out[7]: 'abc'
In [8] 'd' in ''.join( 'a', 'b', 'c')
Out[8]: False
```

Try:

```
a.split( 'o' )
'x'.join( a.split( 'o' ) )
```

Surprise! strings!!

```
In [11]: x, y = 1, 1.2
In [12]: 'x is %s, y is %s' %(x, y)
Out[12]: 'x is 1, y is 1.234'
```

Try:

```
'x is %d, y is %f' %(x, y)
'x is %3d, y is %4.2f' %(x, y)
```

docs.python.org/lib/typesseq-strings.html

Interlude

A classic problem

How to interchange values of two variables? Please note that the type of either variable is unknown and it is not necessary that both be of the same type even!

60 m

Outline

Control flow constructs

- `if/elif/else` : branching
- `while` : looping
- `for` : iterating
- `break, continue` : modify loop
- `pass` : syntactic filler

Basic conditional flow

```
In [21]: a = 7
In [22]: b = 8
In [23]: if a > b:
    ....:     print 'Hello'
    ....: else:
    ....:     print 'World'
    ....:
    ....:
World
```

Let us switch to creating a file

Creating python files

- aka scripts
- use your editor
- Note that white space is the way to specify blocks!
- extension `.py`
- run with `python hello.py` at the command line
- in IPython...

If...elif...else example

```
x = int(raw_input("Enter an integer:"))
if x < 0:
      print 'Be positive!'
elif x == 0:
      print 'Zero'
elif x == 1:
      print 'Single'
else:
      print 'More'
```

Simple IO

Console Input

`raw_input()` waits for user input.
Prompt string is optional.
All keystrokes are Strings!
`int()` converts string to int.

Console output

`print` is straight forward. Major point to remember is the distinction between `print x` and `print x,`

Basic looping

```
# Fibonacci series:
# the sum of two elements
# defines the next
a, b = 0, 1
while b < 10:
    print b,
    a, b = b, a + b
```

```
1 1 2 3 5 8
```

Recall it is easy to write infinite loops with **while**

80 m

Problem set 1

- All the problems can be solved using `if` and `while`

Problem 1.1

Write a program that displays all three digit numbers that are equal to the sum of the cubes of their digits. That is, print numbers *abc* that have the property $abc = a^3 + b^3 + c^3$
These are called *Armstrong* numbers.

Problem 1.2 - Collatz sequence

1. Start with an arbitrary (positive) integer.
2. If the number is even, divide by 2; if the number is odd multiply by 3 and add 1.
3. Repeat the procedure with the new number.
4. It appears that for all starting values there is a cycle of 4, 2, 1 at which the procedure loops.

Write a program that accepts the starting value and prints out the Collatz sequence.

Problem 1.3 - Kaprekar's constant

1. Take a four digit number–with at least two digits different.
2. Arrange the digits in ascending and descending order, giving A and D respectively.
3. Leave leading zeros in A!
4. Subtract A from D.
5. With the result, repeat from step 2.

Write a program to accept a 4-digit number and display the progression to Kaprekar's constant.

Problem 1.4

Write a program that prints the following pyramid on the screen.

```
1
2   2
3   3   3
4   4   4   4
```

The number of lines must be obtained from the user as input.

When can your code fail?

Problem 1.4

Write a program that prints the following pyramid on the screen.

```
1
2   2
3   3   3
4   4   4   4
```

The number of lines must be obtained from the user as input.

When can your code fail? 105 m

Outline

Functions: examples

```
def signum( r ):
    """returns 0 if r is zero
    -1 if r is negative
    +1 if r is positive"""
    if r < 0:
        return -1
    elif r > 0:
        return 1
    else:
        return 0
```

Functions: examples

```
def pad( n, size ):
    """pads integer n with spaces
    into a string of length size
    """
    SPACE = ' '
    s = str( n )
    padSize = size - len( s )
    return padSize * SPACE + s
```

What about %3d?

Functions: examples

```
def pad( n, size ):
    """pads integer n with spaces
    into a string of length size
    """
    SPACE = ' '
    s = str( n )
    padSize = size - len( s )
    return padSize * SPACE + s
```

What about %3d?

What does this function do?

```
def what( n ):
    if n < 0: n = -n
    while n > 0:
        if n % 2 == 1:
            return False
        n /= 10
    return True
```

What does this function do?

```
def what( n ):
    i = 1
    while i * i < n:
        i += 1
    return i * i == n, i
```

What does this function do?

```
def what( n, x ):
    z = 1.0
    if n < 0:
        x = 1.0 / x
        n = -n
    while n > 0:
        if n % 2 == 1:
            z *= x
        n /= 2
        x *= x
    return z
```

Before writing a function

- Builtin functions for various and sundry
- `abs, any, all, len, max, min`
- `pow, range, sum, type`
- Refer here: `http://docs.python.org/library/functions.html`

120 m

Problem set 2

The focus is on writing functions and calling them.

Problem 2.1

Write a function to return the gcd of two numbers.

Problem 2.2

A pythagorean triad (a, b, c) has the property $a^2 + b^2 = c^2$.

By primitive we mean triads that do not 'depend' on others. For example, (4,3,5) is a variant of (3,4,5) and hence is not primitive. And (10,24,26) is easily derived from (5,12,13) and should not be displayed by our program.

Write a program to print primitive pythagorean triads. The program should generate all triads with a, b values in the range 0—100

Problem 2.3

Write a program that generates a list of all four digit numbers that have all their digits even and are perfect squares.
For example, the output should include 6400 but not 8100 (one digit is odd) or 4248 (not a perfect square).

Problem 2.4

The aliquot of a number is defined as: the sum of the *proper* divisors of the number. For example, the aliquot(12) = 1 + 2 + 3 + 4 + 6 = 16.
Write a function that returns the aliquot number of a given number.

Problem 2.5

A pair of numbers (a, b) is said to be amicable if the aliquot number of a is b and the aliquot number of b is a.
Example: `220, 284`
Write a program that prints all five digit amicable pairs. 150 m

Outline

List creation and indexing

```
>>> a = [] # An empty list.
>>> a = [1, 2, 3, 4] # More useful.
>>> len(a)
4
>>> a[0] + a[1] + a[2] + a[-1]
10
```

- Indices start with ?
- Negative indices indicate ?

List: slices

- Slicing is a basic operation
- `list[initial:final:step]`
- The step is optional

```
>>> a[1:3] # A slice.
[2, 3]
>>> a[1:-1]
[2, 3, 4]
>>> a[1:] == a[1:-1]
False
```

Explain last result

List: more slices

```
>>> a[0:-1:2] # Notice the step!
[1, 3]
>>> a[::2]
[1, 3]
>>> a[-1::-1]
```

What do you think the last one will do?

Note: Strings also use same indexing and slicing.

List: examples

```
>>> a = [1, 2, 3, 4]
>>> a[:2]
[1, 3]
>>> a[0:-1:2]
[1, 3]
```

Lists are mutable (unlike strings)

```
>>> a[1] = 20
>>> a
[1, 20, 3, 4]
```

List: examples

```
>>> a = [1, 2, 3, 4]
>>> a[:2]
[1, 3]
>>> a[0:-1:2]
[1, 3]
```

Lists are mutable (unlike strings)

```
>>> a[1] = 20
>>> a
[1, 20, 3, 4]
```

Lists are mutable and heterogenous

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]
>>> a[0:2] = [1, 12] # Replace items
>>> a
[1, 12, 123, 1234]
>>> a[0:2] = [] # Remove items
>>> a.append( 12345 )
>>> a
[123, 1234, 12345]
```

List methods

```
>>> a = ['spam', 'eggs', 1, 12]
>>> a.reverse() # in situ
>>> a
[12, 1, 'eggs', 'spam']
>>> a.append(['x', 1])
>>> a
[12, 1, 'eggs', 'spam', ['x', 1]]
>>> a.extend([1,2]) # Extend the list.
>>> a.remove( 'spam' )
>>> a
[12, 1, 'eggs', ['x', 1], 1, 2]
```

List containership

```
>>> a = ['cat', 'dog', 'rat', 'croc']
>>> 'dog' in a
True
>>> 'snake' in a
False
>>> 'snake' not in a
True
>>> 'ell' in 'hello world'
True
```

Tuples: immutable

```
>>> t = (0, 1, 2)
>>> print t[0], t[1], t[2], t[-1]
0 1 2 2
>>> t[0] = 1
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item
           assignment
```

- Multiple return values are actually a tuple.
- Exchange is tuple (un)packing

range() function

```
>>> range(7)
[0, 1, 2, 3, 4, 5, 6]
>>> range( 3, 9)
[3, 4, 5, 6, 7, 8]
>>> range( 4, 17, 3)
[4, 7, 10, 13, 16]
>>> range( 5, 1, -1)
[5, 4, 3, 2]
>>> range( 8, 12, -1)
[]
```

for...range(...) idiom

```
In [83]: for i in range(5):
    ....:     print i, i * i
    ....:
    ....:
0 0
1 1
2 4
3 9
4 16
```

for: the list companion

```
In [84]: a = ['a', 'b', 'c']
In [85]: for x in a:
    ....:     print x, chr( ord(x) + 10 )
    ....:
a   k
b   l
c   m
```

Iterating over the list and not the index + reference

what if you want the index?

`for`: the list companion

```
In [89]: for p, ch in enumerate( a ):
    ....:     print p, ch
    ....:
    ....:
0 a
1 b
2 c
```

Try: `print enumerate(a)` 170 m

Problem set 3

As you can guess, idea is to use `for` !

Problem 3.1

Which of the earlier problems is simpler when we use `for` instead of `while` ?

Problem 3.2

Given an empty chessboard and one Bishop placed in any square, say (r, c), generate the list of all squares the Bishop could move to.

Problem 3.3

Given two real numbers `a, b`, and an integer `N`, write a function named `linspace( a, b, N)` that returns an ordered list of `N` points starting with `a` and ending in `b` and equally spaced.
For example, `linspace(0, 5, 11)`, should return,

```
[ 0.0 ,  0.5,  1.0 ,  1.5,  2.0 ,  2.5,
  3.0 ,  3.5,  4.0 ,  4.5,  5.0 ]
```

Problem 3.4a (optional)

Use the `linspace` function and generate a list of N tuples of the form

$[(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_N, f(x_N))]$

for the following functions,

- `f(x) = sin(x)`
- `f(x) = sin(x) + sin(10*x).`

Problem 3.4b (optional)

Using the tuples generated earlier, determine the intervals where the roots of the functions lie.

185 m

Outline

Simple tokenizing and parsing

```
s = """The quick brown fox jumped
      over the lazy dog"""
for word in s.split():
    print word.capitalize()
```

Problem 4.1

Given a string like, “1, 3-7, 12, 15, 18-21”, produce the list

```
[1,3,4,5,6,7,12,15,18,19,20,21]
```

File handling

```
>>> f = open('/path/to/file_name')
>>> data = f.read() # Read entire file.
>>> line = f.readline() # Read one line
>>> f.close() # close the file.
```

Writing files

```
>>> f = open('/path/to/file_name', 'w')
>>> f.write('hello world\n')
>>> f.close()
```

- Everything read or written is a string

Try `file?` for more help

File and for

```
>>> f = open('/path/to/file_name')
>>> for line in f:
...         print line
...
```

Problem 4.2

The given file has lakhs of records in the form:
`RGN;ID;NAME;MARK1;...;MARK5;TOTAL;PFW`
Some entries may be empty. Read the data from this file and print the name of the student with the maximum total marks.

Problem 4.3

For the same data file compute the average marks in different subjects, the student with the maximum mark in each subject and also the standard deviation of the marks. Do this efficiently.

205 m

Outline

Modules

```
>>> sqrt(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> import math
>>> math.sqrt(2)
1.4142135623730951
```

Modules

- The `import` keyword "loads" a module
- One can also use:

```
>>> from math import sqrt
>>> from math import *
```

- What is the difference?
- Use the later only in interactive mode

Package hierarchies

```
>>> from os.path import exists
```

Modules: Standard library

- Very powerful, “Batteries included”
- Some standard modules:
  - Math: `math`, `random`
  - Internet access: `urllib2`, `smtplib`
  - System, Command line arguments: `sys`
  - Operating system interface: `os`
  - Regular expressions: `re`
  - Compression: `gzip`, `zipfile`, and `tarfile`
  - And a whole lot more!
- Check out the Python Library reference: `http://docs.python.org/library/`

Modules of special interest

`numpy`	Efficient, powerful numeric arrays
`matplotlib`	Easy, interactive, 2D plotting
`scipy`	statistics, optimization, integration, linear algebra, Fourier transforms, signal and image processing, genetic algorithms, ODE solvers, special functions, and more
Mayavi	Easy, interactive, 3D plotting

Creating your own modules

- Define variables, functions and classes in a file with a `.py` extension
- This file becomes a module!
- Accessible when in the current directory
- Use `cd` in IPython to change directory
- Naming your module

Modules: example

```
# --- arith.py ---
def gcd(a, b):
    if a%b == 0: return b
    return gcd(b, a%b)
def lcm(a, b):
    return a*b/gcd(a, b)
# ------------------
>>> import arith
>>> arith.gcd(26, 65)
13
>>> arith.lcm(26, 65)
130
```

Problem 5.1

Put all the functions you have written so far as part of the problems into one module called `iitb.py` and use this module from IPython.

225 m

Did we meet the goal?

1. Agenda
2. Agenda
3. Overview
4. Python
   - Getting Started
   - Data types
   - Control flow
   - Functions
   - Lists
   - IO
   - Modules

Tomorrow

- Plotting: 2D, 3D
- NumPy, SciPy
- Dictionary, Set
- Debugging
- Testing
- ...

11:30–13:00 Discussion of answers to problems OPTIONAL