Python Introduction

Contact: weber@informatik.uni-hamburg.de
WTM / Informatik Raum F-233
Tel. 42883-2537

Sources: www.csc.villanova.edu/~nlp/python1.ppt
www.cs.umbc.edu/pub/www/courses/graduate/631/Fall2002/Python.ppt
...

Python Features

- interpreted language
- clean syntax, object oriented, powerful extensions
- useful built-in types (lists, dictionaries)

  → for symbolic AI processing
- easy matrix algebra (module `numpy`)

  → for statistical AI processing
- easy to program GUIs
- easy to produce HTML content

Edit and Run a Program

- IDLE integrated development environment (IDE)
  for Linux and commercial operating systems
- `ipython` interactive shell for Linux, understands
  python and ''magic'' commands:
  `%run file.py` loads and runs a file
  `%hist` prints history of input
- `python file.py` interpretes the file
- `file.py` runs the file, but first make it executable,
  and write inside like: `#!/usr/bin/python`

A Sample of Code ...

```
x = 4 - 3       # comment: integer difference
y = "Hello"

if x == 0 or y == "Hello":
      x = x + 1
      y = y + " World"  # concatenate string

print x
print y
```

Enough to Understand the Code

- assignment uses = and comparison uses ==
- + -- * / % compute numbers as expected
- use + for string concatenation
- use % for string formatting
- logical operators are words (and, or, not),
  but not symbols (&&, ||, !)
- first assignment to a variable will create it
- Python assigns the variable types

Basic Data Types

- integers (default for numbers)

```
z = 5 / 2   # answer is 2, integer division
```

- floats

```
x = 3.456
```

- strings

Can use "" or ‘’ to specify. `"abc"` `‘abc’` (same)

Unmatched quotes can occur in the string: `"matt’s"`

Use triple double-quotes for multi-line strings or strings which contain both ’ and " inside: `"""a‘b"c"""`

Whitespace and Indentation

- use a newline to end a line of code

  (use \ when must go to next line prematurely)

- use consistent indentation to mark blocks of code

Comments

- start comments with # – the rest of line is ignored
- can include a "documentation string" as the first line of any new function or class

  – the development environment, debugger, and other tools use it; good style to include one

```
def my_function(x, y):
  """This is the docstring. This function
     does blah blah blah."""
# The code would go here ...
```

Python and Data Types

- Python determines types automatically: "Dynamic Typing"
- But Python is not casual about types, it enforces them thereafter: "Strong Typing"

E.g., you can't just append an integer to a string.

```
x = "the answer is "     # x is string
y = 23                   # y is integer
print x + y              # Python complains:
```

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

Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores, e.g.:

  `bob  Bob  _bob  _2_bob_  bob_2  BoB`

- There are some reserved words:

and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

Multiple Assignment

```
x, y, z = 1, 2, 3
y
```

```
2
```

Many String Operations

- built-in formatting on string data type, e.g.

```
"hello".upper()
 str.upper("hello")          # same

 'HELLO'



"abc;;  456  ".split(";")

 ['abc', '', '  456  ']
```

Printing to Screen

- use % string operator to format output text

```
print "%s xyz %d"  %  ("abc", 34)
                # one single (tuple) object given
                # after the formtting operator %
```

```
abc xyz 34
```

```
print "abc",  "xyz", 34      # does same as above
```

```
# print statement replaced by print() function
in Python 3.0
```

File Reading and Writing

```
fobj = open("infile.txt", "r")

line = fobj.readline()

rest = fobj.read()

fobj.close()
```

```
fobj = open("outfile.txt", "w")

fobj.write("values: %d, %.6f\n" % (4, 1.234))

fobj.close()
```

Module Import: NumPy

```
import numpy

A = numpy.zeros((2,3))

array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

```
from numpy import random

B = random.uniform(0.0,1.0,(2,3))

array([[ 0.05718388,  0.77822063,  0.85527822],
       [ 0.85458464,  0.98040686,  0.9404561 ]])
```

Pitfall: Inconsistent Modules

```
import numpy
import random
n, r = numpy.zeros(15), numpy.zeros(15)
for i in range (15):
    n[i] = numpy.random.randint(0,2)    # 0 <= n[i] < 2
    r[i] = random.randint(0,2)          # 0 <= r[i] < 3 !
print 'n =', n, '\nr =', r
```

```
n = [ 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1.]
r = [ 1. 1. 0. 0. 0. 0. 0. 2. 0. 1. 1. 2. 2. 2. 1.]
```

Pitfall: Inconsistent Vector Handling

```
import numpy, math


a = numpy.ones(2)
numpy.exp(a)                    # numpy can handle vectors
array([2.718, 2.718])

math.exp(a)                     # math expects scalar
TypeError: only length-1 arrays can be converted to Python scalars

map(math.exp, a)      # map: apply function to iterable
[2.718, 2.718]
```

Assignments are by Reference

- `b = a` does not make a copy; it's the same object

e.g.:

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

b = a

a.append(7)

print b
```

[1,2,3,7]

but:

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

c = a[:]

a.append(7)

print c
```

[1,2,3]

```
import numpy

a = [1,2,3]

d = \

numpy.array(a)

a.append(7)

print d
```

[1,2,3]

Pitfall: NumPy Arrays ≠ Lists

```
A=numpy.array([[1,2],
[3,4]])

A[0,1]
  2


A[0,:]      # 0th row
  array([1,2])
A[:,0]      # 0th column
  array([1,3])
```

```
A=[[1,2],[3,4]]

A[0,1]
  TypeError

A[0][1]
  2

A[0][:]
  [1,2]

A[:][0]
  [1,2]   !!
```

Fast and Slow Python

```
import numpy

I = numpy.ones(10000)
W = numpy.ones((400,10000))
S = numpy.zeros(400)

for i in range(10):
  S += numpy.dot(W, I)
```

ran in 0.8 seconds

```
import numpy

I = numpy.ones(10000)
W = numpy.ones((400,10000))
S = numpy.zeros(400)

for i in range(10):
  for i in range (400):
    for j in range (10000):
      S[i] += W[i][j]*I[j]
```

ran in 49 seconds

Neural Networks with Numpy

```
import numpy

I = numpy.ones(10000)
W = numpy.ones((400,10000))


for i in range(10):
  S = numpy.dot(W, I)
```

use vectors (rank 1 arrays)

for neural layer activations

use matrices (rank 2 arrays)

for connection weights

scalar product activates a

neural layer from its input

Class

```
class stack:
    def __init__(self):
        self.items = []
    def push(self, x):
        self.items.append(x)
    def pop(self):
        x = self.items[-1]
        del self.items[-1]
        return x
    def empty(self):
        return len(self.items) == 0
```

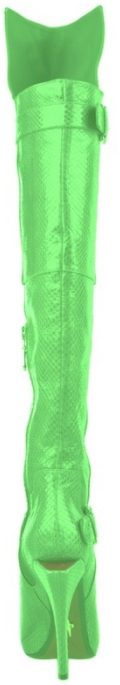
Use like:

```
t = stack()
print t.empty()
 True
t.push("hello")
print t.empty()
 False
t.pop()
print t.empty()
 True
```

Links

- http://python.org
- http://wiki.python.org/moin/BeginnersGuide/Programmers
- http://www.diveintopython.net
- http://www.rexx.com/~dkuhlman/python_book_01.html
- http://rgruet.free.fr
- http://ipython.org
- http://openbook.galileocomputing.de/python

The End



... additional topics ...

Embedding C in Python (option 1)

```
import scipy.weave

a, b = 1, 2

c = scipy.weave.inline('return_val = a + b;\

printf("a=%d b=%d\\n", a, b);', ['a','b'])

print "c = %d" % c
```

```
a=1 b=2

c = 3
```

Embedding C in Python (option 2)

array.c

```
#include <stdlib.h>

void square(int nsize,
            double *v){
  int i;
  for(i=0;i<nsize;i++){
    v[i] = v[i]*v[i];
  }
}
```

call.py

```
import ctypes

MyLib = ctypes.CDLL('./array.so')
myArray = (5*ctypes.c_double)()
for i in range(5):
    myArray[i] = i
myLib.square(5,myArray)
for i in range(5):
    print(myArray[i])
```

gcc -fPIC -c array.c -o array.o
gcc -shared -Wl,-soname, -o array.so -fPIC array.o

```
npArray = numpy.frombuffer(myArray)
```
... obtain a proper numpy array

python call.py

```
0.0
1.0
4.0
9.0
16.0
```

slde content thanks to Mehmet Suzen

Embedding Python in C

```
void main () {
  Py_Initialize();
  PyRun_SimpleString("k = \"hello\"");
  PyRun_SimpleString("print k");
  Py_Finalize();
}

gcc embedPythonInC.c -lpython2.6
./a.out
hello
```

http://www.rexx.com/~dkuhlman/python_book_01.html#extending-and-embedding-python