Programming, bridging course
2009
## Course outline

Python, week 44-45, Lennart Svensson  
C, week 46-47, Olle Eriksson  
C++ or FORTRAN  
  week 48-51  
  C++: Olle Eriksson, Lennart Svensson  
  FORTRAN: Per Wahlund

<table>
<thead>
<tr>
<th>week</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
<th>48</th>
<th>49</th>
<th>50</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>nr of lectures</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
General information

Homepage
http://www.it.uu.se/edu/course/homepage/progbrygg/ht09/

Teachers
Olle Eriksson, olle.eriksson@it.uu.se
Lennart Svensson, lennart.svensson@cb.uu.se
Per Wahlund, per.wahlund@it.uu.se
About the course

• Few lectures
• Focus on programming practice
• Slides on the homepage
• No course book. Using online resources instead.
Online Python resources

http://www.python.org/
http://docs.python.org/
http://en.wikipedia.org/wiki/Python_(programming_language)

• Short tutorial:
  http://www.poromenos.org/tutorials/python
Books

If you want to go deeper:

- Python Essential Reference, D. Beazley, 2009
  - manual-like, few examples, concise
- Dive Into Python, M. Pilgrim, 2004
  - popular but a little old
- Core Python Programming, W. Chun, 2006
- Python programmering, E. Lindblad, 2006
  - in Swedish, easy to read
Assignments

• Compulsory
• 1 in Python
• 3 in C
• Probably 2 in C++
• Hand in when the corresponding part of the course ends
Examination

• Complete the assignments for passing grade
• For grades 4 & 5 you have to write the exam
Python

- General purpose high-level programming language
- Designed by Dutch programmer Guido Van Rossum (a.k.a. Python's Benevolent Dictator for Life) who released the first version in 1991 (Rossum now works at Google, which claims that 15% of their code base is in Python)
- There is an active community that contributes to Python's development
Lecture outline

• Example program
• Some general remarks about Python
• Python:
  – Basic statements
  – Code partitioning
  – Fundamental data types
Python example

- All computer languages intros start with a program that prints "Hello, World!" to the screen
- Engineering extension: read a number, compute its sine value, and print out
- The script, called hw.py, should be run like this:
  `python hw.py 3.4`
- or just (Unix)
  `./hw.py 3.4`
- Output:
  `Hello, World! sin(3.4)=-0.255541102027`
Purpose of this script

Demonstrate

- how to get input from the command line
- how to call a math function like sin(x)
- how to work with variables
- how to print text and numbers
The code

• File hw.py:

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

# load system and math module:
import sys, math

# extract the 1st command-line argument:
r = float(sys.argv[1])
s = math.sin(r)
print "Hello, World! sin(" + str(r) + ")=" + str(s)
```

• Make the file executable (on Unix):

`chmod a+rx hw.py`
• The first line specifies the interpreter of the script (here the first python program in your path)

```python
python hw.py 1.4 # first line is not treated as comment
./hw.py 1.4      # first line is used to specify an interpreter
```

• Even simple scripts must load modules:

```python
import sys, math
```

• Numbers and strings are two different types:

```python
r = sys.argv[1] # r is string
s = math.sin(float(r))
# sin expects number, not string r
# s becomes a floating-point number
```
Alternative print statements

- Desired output:
  Hello, World! sin(3.4)=-0.255541102027

- String concatenation:
  print "Hello, World! sin(" + str(r) + ")=" + str(s)

- Base string plus arguments:
  print "Hello, World! sin(%g)=%g" % (r,s)
  (as printf in C)
printf format strings

%d : integer
%5d : integer in a field of width 5 chars
%-5d : integer in a field of width 5 chars, but adjusted to the left
%e : float variable in scientific notation
%11.3e : float variable in scientific notation, with 3 decimals, field of width 11 chars
%5.1f : float variable in fixed decimal notation, with one decimal, field of width 5 chars
%.3f : float variable in fixed decimal form, with three decimals, field of min. width
%s : string
Programming languages

**Application:** batch, script, applications, web, database, numerical, education...

**Paradigms:** functional, object-oriented...

**Other differentiations:** dynamic/static type checking, expressiveness, speed, compiled/interpreted, high-/lowlevel ...

http://en.wikipedia.org/wiki/Programming_language
Important aspects outside language

Available libraries
Programming tools
Supported platforms
Integration with other components
Documentation
Community
PR
Compiled vs interpreted languages

In some way the code we write must be translated to machine code.

Alternative 1:

When you've finished programming you run a compiler to do the translation (compiling), and a resulting binary file containing the machine code is acquired. Examples: C, Fortran.

Alternative 2:

An interpretator do the translation (interpretation) when the program is executed. You say you run your script code directly. Examples: Python, Perl, Ruby.

Not entirely true, and the two alternatives are converging.
Are scripts compiled?

- Are Python scripts compiled?
- Yes and no, depending on how you see it
- Python first compiles the script into bytecode
- The bytecode is then interpreted
- No linking with libraries; libraries are imported dynamically when needed
- It appears as there is no compilation
- Quick development: just edit the script and run!
- (no time-consuming compilation and linking)
- Extensive error checking at run time
Python benefits

• Concise
• Readable
• Portable
• Many language elements
• Extensive standard library
• Easy to integrate with C/C++
Python drawbacks

• Execution speed
• Multi-threading not very good

Google supports the development of Python and is focusing particularly on these issues.
Python popularity

Open source commits according to ohloh.net
History of Python

• 1991, First source code release to alt.sources
• 1994, v1.0
• 2000, v2.0, garbage collection
• 2008, v3.0, many redundant commands are removed, increased coherency, broke backward compatibility
Python tasks

• Internet applications, for example via HTTP, FTP, SMTP and CGI.

• Control program for components written in low level languages, glue together building blocks.

• Portable programs

• Database programming. Database server is the limiting factor, not the client.

Not suitable: time critical modules, safety critical systems
Python interpreter

- You can also run the interpreter without a program file, and instead use Python in command line mode:

  ```python
  >>> print "Hello World!"
  Hello World!
  >>> import math
  >>> 73 - math.sqrt(52)
  65.78897449072024
  >>> _ - 2
  63.78897449072024
  ```

- `_` stores the last result (not available in scripts)
Basic statements in Python

• Control statements - controlling execution flow
  • Conditional: if, elif, else
    - Determines whether a code block should be executed or not.
  • Loop: for, while
    - Iterates over a list
    - Executes a code block many times.
if a < b:
    print "a is smaller than b"
elif a == b:
    print "a is equal to b"
else:
    print "a is greater than b"

• No parenthesis needed in the conditional statement.

• No braces to denote code blocks. It's specified by the indentation.

• Multiple choices are handled with 'elif', there isn't any 'switch' as in some languages, as redundant features are avoided.
For-loops

- Executes a code block many times for each iterated element

```
for x in range(3):
    print "x: " + str(x)
```

```
> python test.py
x: 0
x: 1
x: 2
```

For-loops

- With an extra code line in the loop:

```python
import math
for x in range(3):
    print "x: " + str(x),
    print "sin(x): " + str(math.sin(x))
```

```
> python test.py
x: 0   sin(x): 0.0
x: 1   sin(x): 0.841470984808
x: 2   sin(x): 0.909297426826
>
```
For-loop formats

- For-loop in Python:

  ```python
  for i in range(start, stop, inc):
    ...
  for j in range(stop):
    ...
  ```

- generates:

  $$i = start, start+inc, start+2*inc, \ldots, stop-1$$

  $$j = 0, 1, 2, \ldots, stop-1$$
Nested for-loops

• For-loops can be inside other for-loops in any number of steps

```python
for x in range(2):
    for y in range(2):
        for z in range(2):
            print x, y, z
```

> python test.py
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
>
Code partitioning

• Plain script
• Functions
• Classes
• Modules
import math

def myfunc(y):
    if y >= 0.0:
        return y*5*math.exp(-y)
    else:
        return 0.0

# alternative way of calling module functions
# (gives more math-like syntax in this example):
from math import *
def myfunc(y):
    if y >= 0.0:
        return y*5*exp(-y)
    else:
        return 0.0
A module is corresponding to a file

Example:

```python
# file: div.py
def divide(a, b):
    q = a/b
    r = a - q*b
    return(q, r)
```

to use in another file:

```python
import div
a, b = div.divide(c, d)
```
Classes

- Classes bundle a set of objects and a set of functions.

```python
class Message:
    def __init__(self, aString):
        self.text = aString
    def printIt(self):
        print self.text
```

- More about classes in a later lecture
Data type classes

Numeric types: int, long, float, complex, bool
Sequence types: str, unicode, list, tuple, xrange
Mapping types: dict
Set types: set, frozen set
None type: type(None)

Each class share the basic means of accessing elements.
Numeric types

- Booleans
- Integers, 32-bit signed
- Long integers
- Floating-point numbers, 64-bit, IEEE 754
- Complex numbers

Implicitly defined by how the number is written.
Strings

• Creation, no explicit typing

```python
>>> a = "Hello World"
```

• Indexing

```python
>>> a[:5]
'Hello'
>>> a[6:]
'World'
>>> a[4:7]
'o W'
```  

• Concatenation

```python
>>> a + " adding some text"
'Hello World adding some text'
```
Strings

- Single- and double-quoted strings work in the same way

  ``` python
  s1 = "some string with a number %g" % r
  s2 = 'some string with a number %g' % r # = s1
  ```

- Triple-quoted strings can be multi line with embedded newlines:

  ```python
text = """
large portions of a text
can be conveniently placed
inside triple-quoted strings
(newlines are preserved)"""
  ```

- Raw strings, where backslash is backslash:

  ```python
  s3 = r’\(\s+\.\d+\)’
  # with ordinary string (must quote backslash):
  s3 = ‘\(\s+\.\d+\)’
  ```
Two other sequences

- **List:**
  \[
  a = [1, 2, 3, "four"]
  \]

- **Tuple:**
  \[
  b = (1, 2, 3, "four")
  \]

- Lists are slower but dynamic, tuples are faster but static.

- Indexed in the same way as strings.
Important libraries

- NumPy - multi-dimensional arrays and functions related to these
- SciPy - built using NumPy, integration, optimization, linear algebra, statistics
- matplotlib - histograms, power spectra, bar charts, errorcharts, scatterplots
Assignment

- Calculating the largest eigenvalue and the corresponding eigenvector
- Test different ways of doing matrix multiplication
- Create a graphical user interface (GUI) for the algorithm
- Due Nov. 9
Python Editors/IDEs

IDLE coming with the installation

Free: Emacs, Wing 101 (Windows)

Commercial: Wing, Komodo
Tips

- Python command prompt good for learning and checking small things.
- When you work on something you want to save in the end, I suggest coding directly into script files. Good code should be saved, and it forces you to think deeper about your problem and not just trying different things out randomly.
Exploring objects

- id(), type(), dir(), help()

```python
>>> p = 4
>>> id(p)
30222660
>>> type(p)
<type 'int'>
>>> dir(p)
['__abs__', '__add__', ...]
>>> help(p)
Help on int object:
...