Python

Tutorial

Jan Pöschko

Graz University of Technology

March 22, 2010

Why Python?

Python is:

- very readable
- easy to learn
- interpreted & interactive – like a UNIX shell, only better
- object-oriented – but not religious about it
- slower than C, but it is easy to integrate C, Fortran, Java

"Batteries included"

The Zen of Python

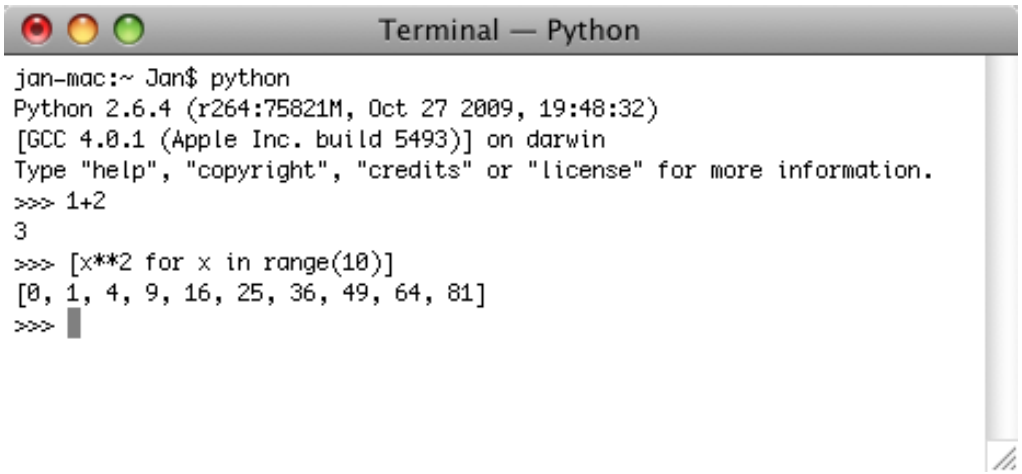
- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one – and preferably only one – obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea – let's do more of those!

Installing Python

- included in most distributions of Linux and Mac OS X
- downloadable from `http://www.python.org/download/`
- libraries:
  - NumPy: `http://sourceforge.net/projects/numpy/`
  - matplotlib: `http://sourceforge.net/projects/matplotlib/files/matplotlib/`
  - NetworkX: `http://networkx.lanl.gov/install.html`
- editors:
  - Eclipse
  - emacs, vim
  - . . . (anything that knows how to handle tabs and spaces)

Invoking Python

- Interactive mode: Open a console/terminal window and run `python` (fancier alternative: `ipython`)



- Executing files: `python myfile.py`

Language basics

```
n = 42                  # integer
x = 3.14159             # float
x = "Hello world!"      # string
```

- no variable declarations
- dynamically typed

String delimeters:

```
s = "Hello world!"
s = 'Hello world!'
s = """Hello
world!"""               # multi-line string
```

Booleans

- `True` is true, `False` is false
- 0, `""` and `None` are false, (almost) everything else is true

```
not A
A and B
A or B
```

Comparisons: ==, !=, <, <=, >, >=

```
2 == 2
1 < 2 <= 3
1 == 2 and "3" or "4"        # = "4"
```

Lists

```
L = [1, 2, 3]
L[0]                    # = 1
L[1:3]                  # = [2, 3]
L[:-1]                  # = [1, 2]
L.append(4)             # -> [1, 2, 3, 4]
L += [5, 6]             # -> [1, 2, 3, 4, 5, 6]
del L[5]                # -> [1, 2, 3, 4, 5]
len(L)                  # = 5
L.reverse()             # -> [5, 4, 3, 2, 1]
L.sort()                # -> [1, 2, 3, 4, 5]
```

Variables only hold references to lists (like to everything else):

```
M = L
M[2] = "A"              # -> [1, 2, "A", 4, 5]
L[2]                    # -> "A"
```

Dictionaries and sets

Dictionaries:

```
D = {"Mozart": 1756, "Schubert": 1797}
D["Mozart"]                 # = 1756
D.keys()                    # = ['Schubert', 'Mozart']
D.values()                  # = [1797, 1756]
D.update({"Beethoven": 1770})
D["Einstein"] = 1879
"Einstein" in D             # = True
len(D)                      # = 4
D.get("Newton", "unknown")  # = 'unknown'
```

Sets:

```
s = set([1, "hello", "world"])
s.add(3)
2 in s                      # = False
```

Control flow

```
if n == 0:
    print "n is 0"
elif n > 0:
    print "n is positive"
else:
    print "n is negative"
```

- Indentation matters!
- Don't mix tabs and spaces – configure your editor appropriately!

```
while n > 0:
    n -= 1
```

```
for n in [1, 2, 3, 4]:
    print n
```

Idioms

Iterate over a dictionary and format strings:

```
D = {"Mozart": 1756, "Schubert": 1797}
for name, year in D.iteritems():
    print "%s was born in %d" % (name, year)
```

List comprehensions:

```
quad = [x**2 for x in range(8)]
    # = [0, 1, 4, 9, 16, 25, 36, 49]
even = [x**2 for x in range(8) if x % 2 == 0]
    # = [0, 4, 16, 36]
```

Files

```
values = [[1, 0], [1, 1]]

# Join values by ";" and lines by newlines
csvtext = '\n'.join(';'.join(str(value) \
    for value in line) for line in values)

# Write the text into a file
csvfile = open('file.csv', 'w')
csvfile.write(csvtext)
csvfile.close()
```

Functions: $\mathrm{fac}(n) = n! := \prod_{k=1}^{n} k$

```
def fac(n):
    if n == 1:
        return 1
    else:
        return n * fac(n - 1)
```

```
def fac(n):
    result = 1
    for k in range(1, n + 1):
        result *= k
    return result
```

```
def fac(n):
    return reduce(lambda a, b: a * b,
        range(1, n + 1), 1)
```

Functions: Parameters

Parameters point to values (like all variables):

```
def change(a, b):
    a = b[0] = 0
a, b = 1, [1, 1]
change(a, b)          # -> a = 1, b = [0, 1]
```

Extra arguments (passed in as list):

```
def sum(*values):
    result = 0
    for value in values:
        result += value
    return result
sum(1, 2, 5)                  # = 8
```

Keyword arguments (passed in as dictionary): `**kwargs`

Generators

```
def subsets(list):
    if not list:
        yield []
        return
    for next in subsets(list[1:]):
        yield [list[0]] + next
        yield next

for set in subsets([1, 2, 3]):
    print set
```

Classes and objects

```
class Animal:
    def __init__(self, name):
        self.name = name
    def say_hello(self):
        print "I'm %s" % self.name

class Dog(Animal):
    def __init__(self, name, owner):
        super(Dog, self).__init__(name)
        self.owner = owner
    def say_hello(self):
        print "Hello %s" % self.owner

dog = Dog("Charly", "Jan")
dog.say_hello()
```

Modules

Any Python file (e.g. `mymodule.py`) is a module and can be imported:

```
import mymodule

mymodule.myfunction()
```

```
from mymodule import myfunction
from mymodule import *    # rather discouraged

myfunction()
```

File-system directories can be used to create "namespaces", if they contain a file `__init__.py` (which may contain initialization code):

```
import mydir.mymodule
from mydir.mymodule import myfunction
```

main

Variable `__name__` contains the name of the current module. It equals `"__main__"` when the script is run directly from the command-line.
Common idiom:

```
def main():
    print "Hello world!"

if __name__ == "__main__":
    main()
```

More advanced concepts

- exceptions
- multiple inheritance
- operator overloading
- metaclasses
- inline documentation and test code
- extensive “runtime” information

Regular expressions

`re` is a module for handling regular expressions.

```
import re

re.findall('[A-Za-z]+', 'Hello world!')
# = ['Hello', 'world']
```

*	0 or more	`findall(pattern, string)`
+	1 or more	`match(pattern, string)`
?	0 or 1	`sub(pattern, repl, string)`
[...]	certain characters	`split(pattern, string)`
[^...]	characters excluded	...

Pre-compiling regular expressions:

```
WORD_RE = re.compile('[A-Za-z]+')
WORD_RE.findall('Hello world!')
```

URLs

`urllib2` is a module for opening URLs.

```
from urllib2 import urlopen

urlfile = urlopen("http://www.google.com")
content = urlfile.read()
  # = '<!doctype html><html>[...]</script>'
```

Basic HTTP authentication:

```
import urllib2
auth_handler = urllib2.HTTPBasicAuthHandler()
auth_handler.add_password(realm='the realm',
     uri='http://www.example.com',
     user='usr', passwd='pwd')
opener = urllib2.build_opener(auth_handler)
urllib2.install_opener(opener)
urlopen('http://www.example.com/restricted')
```

XML documents

`xml.dom` is a module for parsing XML documents and accessing them using the Document Object Model.

```
from xml.dom.minidom import parseString

content = """<tag attr="1">
  <sub>text</sub></tag>"""
dom = parseString(content)
dom.childNodes[0].getAttribute('attr') # = '1'
subs = dom.getElementsByTagName('sub')
subs[0].childNodes[0].nodeValue      # = 'text'
```

JSON data

`json` is a module for parsing and exporting JSON data snippets (included in Python 2.6; use package `simplejson` in 2.5).

```
import json

json_data = '{"list": [1, 2], "key": "value"}'
json.loads(json_data)
```

NetworkX

`networkx` is a module for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

```
import networkx as nx

G = nx.Graph()
G.add_node(1)
G.add_edge(1, 2)
G.add_edge(2, 3)
```

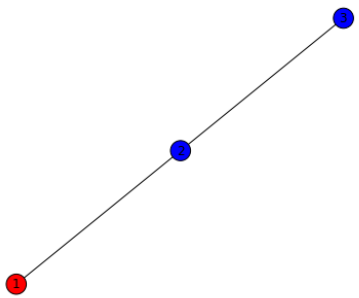
Plot graphs:

```
import matplotlib.pyplot as plt

plt.figure()
nx.draw(G)
plt.savefig('graph.png')
```

NetworkX: Customized visualization

```
plt.figure()
plt.axis('off')
pos = nx.spring_layout(G, iterations=50)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_nodes(G, pos, [1],
    node_color='r')
nx.draw_networkx_nodes(G, pos, [2, 3],
    node_color='b')
nx.draw_networkx_labels(G, pos)
plt.savefig('graph.png', dpi=72)
```



Further reading

- Official Python tutorial: `http://docs.python.org/tutorial/`
- Python tutorial at the University of Toronto: `http://www.cs.toronto.edu/~gpenn/csc401/401_python_web/`
- Python 2.6 Quick Reference: `http://rgruet.free.fr/PQR26/PQR2.6.html`
- `http://docs.python.org/library/re.html`
- `http://docs.python.org/library/urllib2.html`
- `http://docs.python.org/library/xml.dom.html`
- `http://code.google.com/p/simplejson/`
- NetworkX: `http://networkx.lanl.gov/contents.html`