Flying with Python

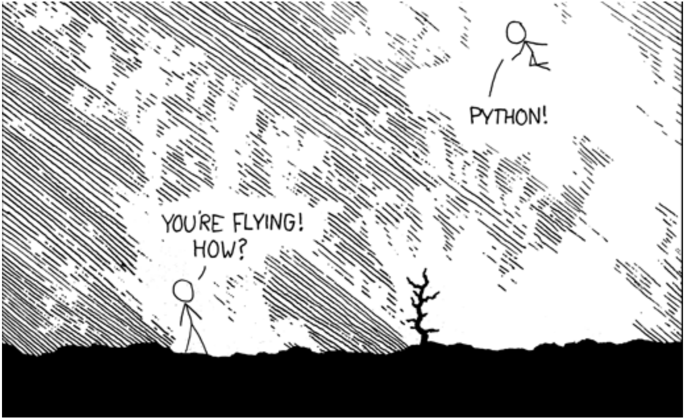
Petr Zemek

Brno University of Technology, Faculty of Information Technology
Božetěchova 2, 612 00 Brno, CZ
http://www.fit.vutbr.cz/∼izemek

Principles of Programming Languages, 2013-03-06

Motto

“Python makes you fly.”



`http://xkcd.com/353/`

Outline

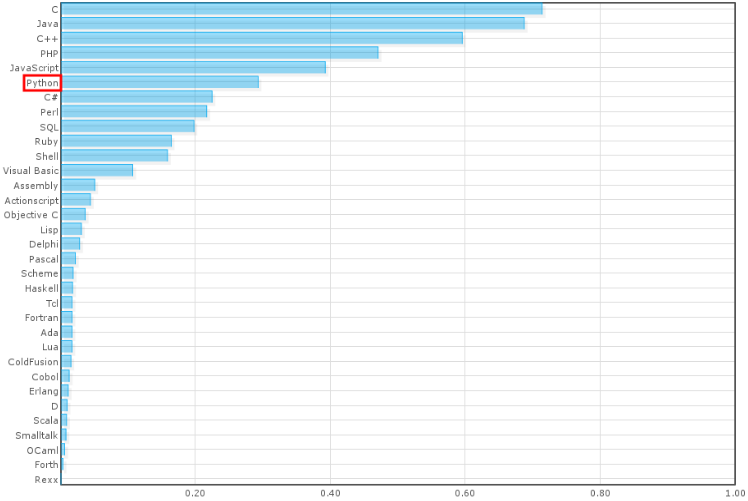
- **Introduction**
- **Language Essentials**
- **Some Cool Language Features**
- **Examples**
- **Concluding Remarks**
- **Demo**

Acknowledgements

Based on the following presentations:

- A Gentle Introduction to Python, M. J. Fromberger
- Introduction to Python, G. Griffin
- Python Programming – Introduction to Python, F. A. Nielsen
- Introduction to Python, H. Boley

Why Python?



`http://www.langpop.com/`

What is Python?

- general-purpose high-level programming language
- design philosophy emphasizes code readability
- multiparadigm (procedural, object-oriented, functional)
- compiled to bytecode and then interpreted in a virtual machine
- everything is an object
- dynamically typed (duck typing)
- portable (CPython, Jython, IronPython)
- highly extensible
- automatic memory management (garbage collector)
- free (as in “free speech”)

A Glimpse at Python History

- invented in the beginning of 1990s by Guido van Rossum

- the name Python stems from “Monty Python’s Flying Circus”
- intended to be a scripting language on Amoeba OS
- influenced by several languages, like ABC, Lisp, and Modula-3
- current versions:
  - Python 2.7.3 (April 2012)
  - Python 3.3.0 (September 2012)

A Glimpse at Python History

- invented in the beginning of 1990s by Guido van Rossum

- the name Python stems from “Monty Python’s Flying Circus”
- intended to be a scripting language on Amoeba OS
- influenced by several languages, like ABC, Lisp, and Modula-3
- current versions:
  - Python 2.7.3 (April 2012)
  - Python 3.3.0 (September 2012)

Python’s Design

- clean, minimal syntax: “executable pseudocode”
- implemented in C and is generally C-like
- uses indentation to delimit blocks
- supports both procedural and object-oriented programming
- uses a small set of powerful built-in data types
- supports generic programming via dynamic binding rather than templating

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Built-In Primitive Data Types

- bool

```
True, False
```

- integer

```
-590, 0, 17821223734857348538746273464545
```

- floating-point

```
0.125, 1e200, inf
```

- complex

```
3 + 4j
```

- string

```
'single quotes'
"double quotes"
"""triple quotes for
multiline strings"""
```

Built-In Collection Types

- list

```
[1, 2, 'a dog', 4.5]
```

- tuple

```
('id', False)
```

- set

```
{0, [], (), True}
```

- dictionary

```
{'key 1': 'value 1', 2: 3, 4: []}
```

Variables

Variables are just like in other programming languages, however:

- they do not have to be declared
- they keep references to objects

```
a = [3, 1, 2]
b = a
b.sort()
print(a) # [1, 2, 3]
```

Operators

arithmetic	+, -, *, /, //, %, **
comparison	<, >, ==, !=, <=, >=
bitwise	<<, >>, |, &, ^, ~
logical	and, or, not
assignment	=, -=, +=, *=, /=, //=, %=, **=
other	in, is

Functions

```python
def add(a, b):
    """This function returns a + b."""
    return a + b

a = add(1, 2)
```

- first-class objects
- default arguments
- variable length argument lists

Flow Control (I)

if conditional execution of a code block

```
if x > 10:
    x = 10
elif x < 5:
    x = foo(x)
else:
    print('error')
```

for traversing items in a collection

```
for i in [1, 2, 3, 4, 5]:
    print(i)
```

while repeated execution of a code block based on a boolean condition

```
while x > 0:
    print(x)
    x -= 1
```

Flow Control (II)

try/catch/finally exception handling

```
f = None
try:
    f = open('aFileName')
    f.write(data)
except IOError:
    print('Unable to open/write file')
except:    # catch all exceptions
    print('Unexpected error')
else:      # if no exceptions are raised
    print('File write completed successfully')
finally:   # clean-up actions, always executed
    if f:
        f.close()
```

Classes

```
class myint(int): # Inheritance from int
    def __init__(self, integer):
        """Constructor."""
        self.integer = integer

    def __add__(self, integer):
        """Overloaded operator '+'."""
        if self.integer == 2 and integer == 2:
            return 5
        else:
            return self.integer + integer

a = myint(2)
print(a+2)  # 5
print(2+a)  # 4
```

- multiple inheritance
- no private methods, everything is public

Packages, Modules and Imports

```
# Import a single module
import time

# Import more modules
import os, sys, re

# Import just one name from the email module
from email import message_from_file

# Import and rename
from urllib2 import urlopen as uop

# Import everything from the given module
from os.path import *
```

- packages (for structuring modules)

Some Cool Language Features (I)

- named string formatting

```
print("The %(foo)s is %(bar)i." %
    {'foo': 'answer', 'bar': 42})
```

Some Cool Language Features (I)

- named string formatting

```
print("The %(foo)s is %(bar)i." %
    {'foo': 'answer', 'bar': 42})
```

- anonymous functions (aka *lambda functions*)

```
sortedList = sort(list, lambda x, y: x > y)
```

Some Cool Language Features (I)

- named string formatting

```
print("The %(foo)s is %(bar)i." %
    {'foo': 'answer', 'bar': 42})
```

- anonymous functions (aka *lambda functions*)

```
sortedList = sort(list, lambda x, y: x > y)
```

- list comprehensions

```
[x**2 for x in range(10)] # [0, 1, 4, 9, 16, ..., 81]
```

Some Cool Language Features (I)

- named string formatting

```
print("The %(foo)s is %(bar)i." %
    {'foo': 'answer', 'bar': 42})
```

- anonymous functions (aka *lambda functions*)

```
sortedList = sort(list, lambda x, y: x > y)
```

- list comprehensions

```
[x**2 for x in range(10)] # [0, 1, 4, 9, 16, ..., 81]
```

- list indexing and slicing

```
a = [1, 2, 3, 4, 5]
print(a[-1]) # 5
print(a[1:4]) # [2, 3, 4]
print(a[2:]) # [3, 4, 5]
print(a[:3]) # [1, 2, 3]
print(a[0:4:2]) # [1, 3]
```

Some Cool Language Features (II)

- conditional expressions

```
a = 1 if x else 2
```

Some Cool Language Features (II)

- conditional expressions

```
a = 1 if x else 2
```

- `eval()` and `exec()`

```
a = eval('1 + 3') # a = 4
exec('b = [1, 2, 3]') # b = [1, 2, 3]
```

Some Cool Language Features (II)

- conditional expressions

```
a = 1 if x else 2
```

- `eval()` and `exec()`

```
a = eval('1 + 3') # a = 4
exec('b = [1, 2, 3]') # b = [1, 2, 3]
```

- duck typing

```
def iterate(col):
    for i in col:
        print(i)

iterate([1, 2, 3])
iterate(('a', 'b', 'c'))
```

Some Cool Language Features (III)

- various syntactical tidbits

```
if 1 < a < 5:
    # ...
```

Some Cool Language Features (III)

- various syntactical tidbits

```
if 1 < a < 5:
    # ...
```

- generators

```
def permute(lst):
    """A really simple permutation generator."""
    if len(lst) < 2:
        yield lst[:]
    else:
        for p in permute(lst[1:]):
            for x in range(len(p) + 1):
                yield p[:x] + [lst[0]] + p[x:]

# Prints all permutations of [1, 2, 3]
for perm in permute([1, 2, 3]):
    print(x)
```

Some Cool Language Features (IV)

- built-in functions for functional programming
  - map

```python
map(lambda s: s.upper(), ['sentence', 'fragment'])
# ['SENTENCE', 'FRAGMENT']
```

Some Cool Language Features (IV)

- built-in functions for functional programming
  - map

```
map(lambda s: s.upper(), ['sentence', 'fragment'])
# ['SENTENCE', 'FRAGMENT']
```

  - filter

```
filter(lambda x: (x % 2) == 0, range(10))
# [0, 2, 4, 6, 8]
```

Some Cool Language Features (IV)

- built-in functions for functional programming
  - map

```
map(lambda s: s.upper(), ['sentence', 'fragment'])
# ['SENTENCE', 'FRAGMENT']
```

  - filter

```
filter(lambda x: (x % 2) == 0, range(10))
# [0, 2, 4, 6, 8]
```

  - enumerate

```
for i, s in enumerate(['sub', 'verb', 'obj']):
    print(i, ':', s)
# 0 : sub
# 1 : verb
# 2 : obj
```

Example 1: File Processing

The following code counts the number of lines in the given file.

```python
f = open('file.txt')
k = 0
for line in f:
    k += 1
print(k)
```

Example 1: File Processing

The following code counts the number of lines in the given file.

```python
f = open('file.txt')
k = 0
for line in f:
    k += 1
print(k)
```

Another solution (on a single line).

```python
print(len([line for line in open('file.txt')]))
```

Example 2: Downloading a Web Page

The following code downloads and prints the given web page.

```python
from urllib.request import urlopen

url = 'http://en.wikipedia.org/wiki/Python'
page = urlopen(url).read()
print(page)
```

Standard Library

- string services (`string`, `re`, `codecs`)
- data types (`datetime`, `calendar`, `queue`, `array`)
- numeric and math modules (`math`, `random`, `functools`)
- OS, file, and directory access (`os`, `tempfile`, `argparse`)
- data persistence (`pickle`, `shelve`)
- data compression (`gzip`, `zipfile`, `tarfile`)
- cryptographic services (`hashlib`, `hmac`)
- Internet data handling and services (`urrlib`, `json`, `cgi`)
- processing tools (`html`, `xml`)
- development tools (`pydoc`, `unittest`)
- ...

Other Useful Libraries and Projects

- `django` (web framework)
- `sqlalchemy` (database toolkit)
- `pygtk`, `pyqt`, `wxpython` (graphical user interface)
- `numpy` (scientific computing)
- `antlr` (language parsing)
- `scons` (software construction tool)
- ...

Advantages of Python

- clean and simple syntax
- easy to parse (and also to learn)
- powerful built-in types
- elegant and flexible module system
- user-defined types using classes
- excellent standard library
- reflection

Disadvantages of Python

- not very fast on computationally intensive operations
- Global Interpreter Lock (GIL)
- (?) lack of variable declarations and type safety
- (?) standardization
- (?) language processor cares at a syntactic level
- (?) not that concise (not a lot of fiddly little close-in operators, a la Perl, C, etc.)

Where to Look for Further Information?

- Python Programming Language – Official Website
  `http://www.python.org/`
- The Python 3 Tutorial
  `http://docs.python.org/release/3.2/tutorial/`
- Python Entry on Wikipedia
  `http://en.wikipedia.org/wiki/Python_(programming_language)`
- Dive into Python 3
  `http://diveintopython3.org/`
- Programming in Python 3 (2nd Edition)
  `http://www.qtrac.eu/py3book.html`

Demo

We show the following:

- creation of a script that obtains email addresses from a file
- writing so-called *unit tests* for the script

Demo

We show the following:

- creation of a script that obtains email addresses from a file
- writing so-called *unit tests* for the script

Source code of the script and tests:

```
http://www.fit.vutbr.cz/~izemek/IPPe/2013/getemails.py
http://www.fit.vutbr.cz/~izemek/IPPe/2013/tester.py
```

The *thank you* slide.