

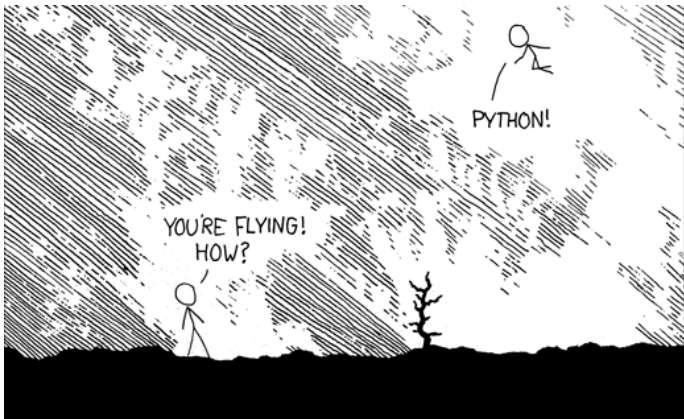
Flying with Python

Petr Zemek

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



“Python makes you fly.”



<http://xkcd.com/353/>



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

Acknowledgements

This presentation is partially 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



- 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”)

- 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 2.7.2 (June 2011)
 - Python 3.2.2 (September 2011)

- 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 foo(x):  
    if x == 0:  
        bar()  
        baz()  
    else:  
        qux(x)  
        foo(x - 1)
```



- 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"""
```



- list

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

- tuple

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

- set

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

- dictionary

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




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]
```



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



```
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



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
```



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()
```



```
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



```
# Import a single module  
import dnd  
  
# 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 utils import *
```

- packages (for structuring modules)



- string formatting

```
'{0}.) {1}'.format(5, 'John') # 5.) John
```




- string formatting

```
'{0}.) {1}'.format(5, 'John') # 5.) John
```

- anonymous (lambda) functions

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



- string formatting

```
'{0}.) {1}'.format(5, 'John') # 5.) John
```

- anonymous (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]
```



- string formatting

```
'{0}.) {1}'.format(5, 'John') # 5.) John
```

- anonymous (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]
```



- conditional expressions

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



- 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]
```



- 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'))
```



- various syntactical tidbits

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



- 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)
```




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

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



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

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

Another solution (on one line).

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



The following code downloads the given web page.

```
from urllib import urlopen

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



- **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** (`urllib`, `json`, `cgi`)
- **processing tools** (`html`, `xml`)
- **development tools** (`pydoc`, `unittest`)
- ...



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



- 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



- 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.)



- **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_\(programming_language\)](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>

The *thank you* slide.