Python introduction

Life is better without braces

Robert Wojciechowicz

Tieto Poland Sp. z o.o.

Python introduction, 2015

Try it in a Python interactive interpreter :-)

```
>>> from __future__ import braces
  File "<stdin>", line 1
SyntaxError: not a chance
```

Outline

- Where is used Python?
- What kind of language is Python?
- Versions/Implementations
- Interpreter
- Syntax
- Data types
- Classes
- Modules
- Exceptions
- Idioms
- Gotchas

Where is used Python?

Web services

On-line games

Cloud storage

Applications

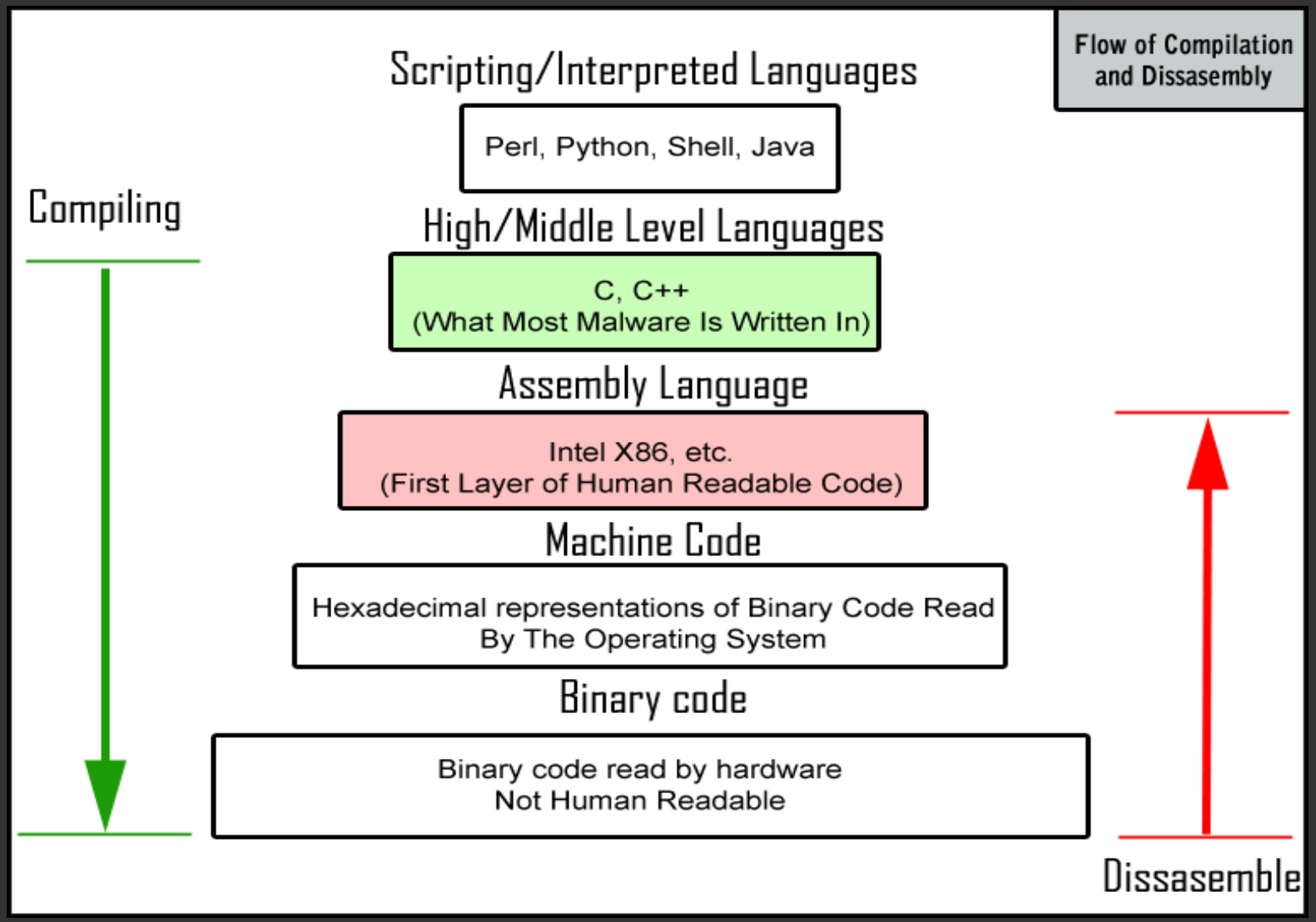
Software tools

What kind of language is Python?

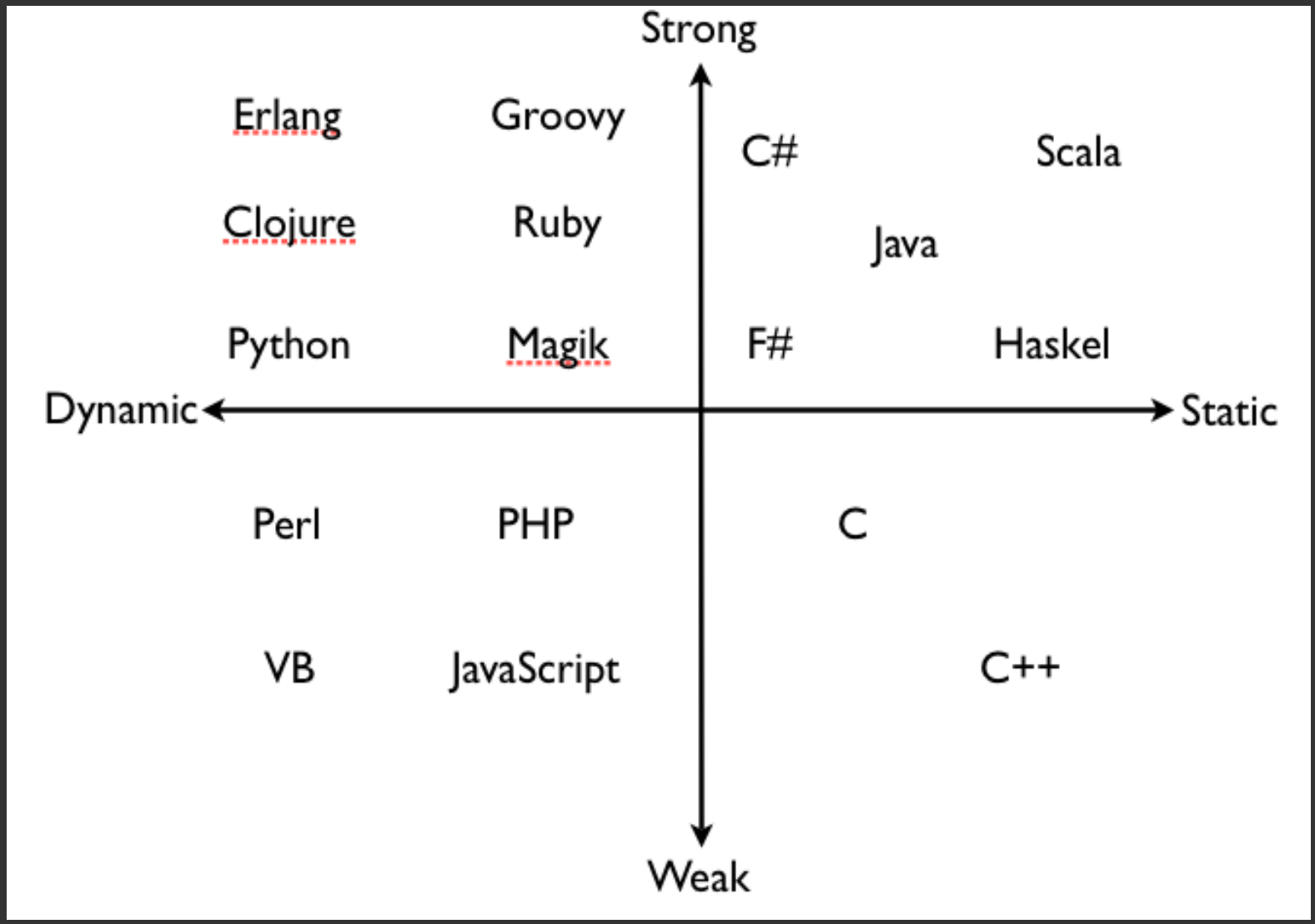
- Open source general-purpose language
- Object Oriented, Procedural, Functional
- Great interactive environment
- Rich and versatile standard library (batteries included)
- Easy to interface with C/C++ (Cython, SWIG)

- Homepage: http://www.python.org
- Documentation: https://docs.python.org
- Free book: http://www.diveintopython.net

Interpreted or compiled?



Type system (dynamic but strong)



Versions/Implementations

- versions
  - 2.x (2.7 final)
  - 3.x (currently 3.4)
- implementations
  - CPython (C, reference/standard)
  - Jython (JVM, currently compatible with CPython 2.5)
  - IronPython (.NET, currently compatible with CPython 2.7)
  - PyPy (Python, currently compatible with CPython 2.7)
  - Stackless (C, CPython branch, microthreads, no stack)

Interpreter

```
$ python
Python 2.7.6 (default, Mar 11 2014, 06:23:12)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license
>>> the_world_is_flat = 1
>>> if the_world_is_flat:
...     print "Be careful not to fall off!"
...
Be careful not to fall off!
>>> quit
Use quit() or Ctrl-D (i.e. EOF) to exit
>>> quit()
```

Zen

```
>>> import this
The Zen of Python, by Tim Peters

[...]
Explicit is better than implicit.
[...]
Readability counts.
[...]
There should be one-- and preferably
only one --obvious way to do it.
```

Running script

```
#!/usr/bin/python
# -*- coding: iso-8859-2 -*-

print 'Witaj, świecie'
```

```
$ python hello.py
Witaj, świecie

$ chmod +x hello.py
$ ./hello.py
Witaj, świecie
```

Syntax

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
>>> a, b = 0, 1
>>> while b < 10:
...       print b,
...       a, b = b, a+b
...
1 1 2 3 5 8

>>> s = 'single line'
>>> s2 = '''multiple
... line'''
>>> print s2
multiple
line
>>> repr(s2)
"'multiple\\nline'"
```

Readability counts

Programs must be written for people to read, and only incidentally for machines to execute.

– Abelson & Sussman, “Structure and Interpretation of Computer Programs”

You can write perl in any language

```
'@'.join(['.'.join(['' .join(
[chr(((ord(c)-ord('a')+13)%26)
+ord('a')) for c in w[::-1]])
for w in p.split('.')[::-1]])
for p in
'zbp.bgrvg@mpvjbuprvpwbj.gerobe'.split('@')
[::-1]])
```

Coding conventions - tabs vs spaces

```
class·Test(object):$
····def·meth1(self):$
»       »       print·'meth1'$
$
Test().meth1()$
```

```
$ python -tt test.py
[...]
TabError: inconsistent use of tabs and spaces
in indentation
$ python -m tabnanny -v test.py
'test.py': *** Line 3: trouble in tab city! ***
offending line: "\t\tprint 'meth1'\n"
indent not greater e.g. at tab sizes 1, 2
```

Variables

- Variables are names, not containers
  - Everything is an object
  - Everything is a reference
  - Variables are neither
- Everything that holds anything, holds references
- Variables refer to objects
  - Namespaces map names to objects

Execution model - bindings

- ```
  >>> variable = 3
  >>> variable = 'hello'
  ```

- So hasn't variable just changed type?
  ```

Execution model - bindings

- ```
  >>> variable = 3
  >>> variable = 'hello'
  ```

- So hasn't variable just changed type?
- Of course not, variable isn't an object at all - it's a name
- ```
  >>> type(3), id(3)
  (<type 'int'>, 26703752)
  >>> type('hello'), id('hello')
  (<type 'str'>, 140531845285568)
  ```

Scopes

- Global

```
>>> y = 1
>>> globals()['y']
1
```

- Local

```
>>> def f():
...     x = 1
...     print locals()
...
>>> f()
{'x': 1}
```

- __builtin__

```
>>> import __builtin__
>>> dir(__builtin__)[-3:]
['vars', 'xrange', 'zip']
```

Function scope

- Definition is visible in any contained block...
- ...unless a contained block introduces a different binding for the name

```
x = 1
def g():
    print x
    x = 2
```

Control Flow

```
if x < 0:
    print 'Negative'
elif x == 0:
    print 'Zero'
else:
    print 'Positive'

switch = {0: lambda: 'Zero',
          1: lambda: 'One'}
x = 1
print switch[x]()

for x in ['a', 'b', 'c']:
    print x,

for x in range(5):
    print x,
```

Data types

- Numbers: int, long, float, complex
- Booleans: True/False
- Sequences
  - immutable: string, unicode, tuple
  - mutable: list, bytearray
- Sets: set, frozenset
- Mappings: dictionary
- Functions
- Classes
  - Classic classes
  - New-style classes
- Modules

Data types - numbers

```
>>> x, y = 3, 2.0
>>> type(x), type(y)
(<type 'int'>, <type 'float'>)
>>> x / y
1.5
>>> x // y
1.0
>>> import sys
>>> n = sys.maxint
>>> n, type(n)
(2147483647, <type 'int'>)
>>> n + 1, type(n + 1)
(2147483648L, <type 'long'>)
>>> n ** 4
21267647892944572736998860269687930881L
```

Data types - sequences

```
>>> s, u = 'string', u'unicode'
>>> type(s), type(u)
(<type 'str'>, <type 'unicode'>)
>>> id(s)
3072504768L
>>> s2 = s.replace('s', 'x')
>>> s, id(s), s2, id(s2)
('string', 3072504768L, 'xtring', 3072504992L)
>>>
>>> lst, tpl = [1, 2], (1,)
>>> lst, type(lst), tpl, type(tpl)
([1, 2], <type 'list'>, (1,), <type 'tuple'>)
>>> lst2 = list('abc')
>>> lst2, type(lst2)
(['a', 'b', 'c'], <type 'list'>)
```

Data types - mappings/sets

```
>>> s, f = {'a', 'b'}, frozenset(['a', 'b'])
>>> s, f
(set(['a', 'b']), frozenset(['a', 'b']))
>>> s.add('c')
>>> s
set(['a', 'c', 'b'])
>>> f.add('c')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has
  no attribute 'add'
>>> d1, d2 = {'a': 1}, dict(a=1)
>>> d1, type(d1)
({'a': 1}, <type 'dict'>)
>>> d2, type(d2)
({'a': 1}, <type 'dict'>)
```

Data types - functions

```
>>> def is_palindrom(s):
...     '''Return True if input sequence `s`
...        is palindrom'''
...     return s == s[::-1]
>>> is_palindrom('kajak')
True
>>> is_palindrom('rower')
False
>>> is_palindrom(s=['p', 'o', 't', 'o', 'p'])
True
>>>
>>> is_palindrom.__name__
'is_palindrom'
>>> is_palindrom.__doc__
'Return True if input sequence `s`\n       is p
```

Classes

- Classic classes

```
>>> class Classic: pass
...
>>> classicobj = Classic()
>>> type(classicobj)
<type 'instance'>
>>> dir(classicobj)
['__doc__', '__module__']
```

- New-style classes

```
>>> class NewStyle(object): pass
...
>>> obj = NewStyle()
>>> type(obj)
<class '__main__.NewStyle'>
>>> dir(obj)
['__class__', '__delattr__', '__dict__',
...
```

Classes - operator overloading

- There are many special methods / hooks which can be overloaded
- Numeric type:
  __add__, __sub__, __mul__ etc.
- Container type:
  __len__, __getitem__, __iter__ etc.
- Callable:
  __call__
- Attribute access:
  __getattr__, __setattr__, __delattr__

Classes - basic but useful customization

- object.__init__(self)
  Instance initialization
- object.__str__(self)
  Called by *str* function and *print* statement to compute “informal” string representaion of an object
- object.__repr__(self)
  Called by the *repr* function to compute the “official” string representation of an object

Classes - adapter design pattern

```
class FTPAdapter(object):
    def __init__(self, ftpserver):
        self._ftpserver = ftpserver

    def run(self):
        self._ftpserver.start()

    def shutdown(self):
        self._ftpserver.stop()

    def __getattr__(self, attr):
        '''Everything else is delegated
           to the ftpserver'''
        return getattr(self._ftpserver, attr)
```

Modules

- Module is a file with suffix “.py” containing Python definitions
- Compiled module is a file with suffix “.pyc” (or “.pyo” when -O option is used)
- Module search path
  - Directory containing the input script
  - PYTHONPATH when set (the same syntax as shell PATH)
  - sys.path initialized depending on above settings and installation default paths (e.g. /usr/lib/python2.7/)

Customizing search path: PYTHONPATH

```
$ ls hello.py*
ls: nie ma dostepu do hello.py*: Nie ma takiego
$ python -c "import hello"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named hello
$ ls test/hello.py*
test/hello.py  test/hello.pyc
$ PYTHONPATH=test python -c "import hello"
hello, World !!!
```

Customizing search path: sys.path

```
$ cat test_module.py
import hello
$ python test_module.py
Traceback (most recent call last):
  File "test_module.py", line 1, in <module>
    import hello
ImportError: No module named hello
$ vi test_module.py
$ cat test_module.py
import sys
sys.path.append('test')
import hello
$ python test_module.py
hello, World !!!
```

Packages

```
oms
|-- __init__.py
|-- common
|    |-- __init__.py
|    `-- utils.py
|-- fm
|    |-- __init__.py
|    |-- fmadapter.py
|    `-- fmuigate.py
`-- pm
     |-- __init__.py
     |-- meahandler.py
     `-- pmfilefetcher.py
```

Packages (cont.)

- Package is a subdirectory with __init__.py file (possibly empty)
- Avoid using: from package import *
- If you really need use __all__ variable

```
__all__ = ['FMAdapter']
```

- Relative imports

```
from ..common import utils
```

- Dynamic import using __import__ function

```
oms = __import__('oms')
```

Handling Exceptions

```
try:
    raise Exception('spam', 'eggs')
except Exception as inst:
    print type(inst) # the exception type
    print inst.args  # arguments
    print inst       # __str__ allows prinitng
    x, y = inst      # __getitem__ allows unpac
    print 'x =', x
    print 'y =', y

>>>
<type 'exceptions.Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

Handling Exceptions - example

```
import sys
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as (errno, strerror):
    print "I/O error({0}): {1}".format(errno,
                                       strerror
except ValueError as ex:
    print "Convertion error: {0}".format(ex)
except:
    print "Unexpected error:", \
          sys.exc_info()[0]
    raise

>>>
I/O error(2): No such file or directory
```

Custom Exceptions

```
class MyError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)

try:
    raise MyError(2*2)
except MyError as e:
    print 'My exception:', e.value


>>>
My exception: 4
```

IDIOMS

Idioms

- Swapping

```
b, a = a, b
```

- Unpacking

```
lst = ['John', 'Cleese']
firstname, surname = lst
```

- Reversing sequence

```
'python'[::-1]
```

- C-like printf

```
def printf(msg, *args):
    print msg % args
```

Idioms (2)

- Interpreter last expression result in _

```
>>> 1024 * 1024
1048576
>>> x = _
>>> x
1048576
```

- building dictionaries

```
>>> firstname = ['John', 'Michael']
>>> surname = ['Cleese', 'Palin']
>>> dict(zip(firstname, surname))
{'John': 'Cleese', 'Michael': 'Palin'}
```

- indexing collections

```
>>> items = 'zero one two'.split()
>>> list(enumerate(items))
[(0, 'zero'), (1, 'one'), (2, 'two')]
>>> for index, item in enumerate(items):
```

Idioms (3)

- Script vs module

```
if __name__ == '__main__':
```

- EAFP (Easier to Ask for Forgiveness than Permission)

```
try:
    return mapping[key]
except KeyError:
    return None
```

```
# NO
isinstance(x, str)
# YES
str(x)
```

- LBYL (Look Before You Leap)

```
if key in mapping: return mapping[key]
```

Comparisons

- Comparison

```
x = 20
# NO
if x > 10 and x <= 20:
# YES
if 10 < x <= 20:
```

- Object type comparisons
  - Yes: if isinstance(obj, int):
  - No: if type(obj) is type(1):
- Empty sequences are false
  - Yes: if not seq:
  - No: if len(seq) == 0:

Batteries included

- Don’t reinvent the wheel

```
# YES
os.path.join(dname, fname)
# NO
dname + '/' + fname
```

-
```
$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

GOTCHAS

One-element tuple creation

```
>>> x = (1)
>>> x
1
>>> type(x)
<type 'int'>
>>> x = 1,
>>> x
(1,)
>>> type(x)
<type 'tuple'>
>>> x = (1,)
>>> x
(1,)
>>> type(x)
<type 'tuple'>
```

Sorting in place

```
>>> lst = [4, 3, 1]
>>> newlst = lst.sort()
>>> newlst
>>> type(newlst)
<type 'NoneType'>
>>> lst
[1, 3, 4]
```

```
>>> lst = [4, 3, 1]
>>> newlst = sorted(lst)
>>> newlst
[1, 3, 4]
>>> type(newlst)
<type 'list'>
>>> lst
[4, 3, 1]
```

Function default parameter

- Mutable object as default parameter value

```
def f(a, lst=[]):
    lst.append(a)
    return lst

>>> f(1)
[1]
>>> f(2)
[1, 2]
>>> f(3)
[1, 2, 3]
```

```
def f(a, lst=None):
    if lst is None:
        lst = []
    lst.append(a)
    return lst

>>> f(1)
[1]
>>> f(2)
[2]
>>> f(3)
[3]
```

Gotchas (cont.)

- Scope and variables

```
x = 1
def g():
    print x
    x = 2
```

- * operator copies references, not copies of objects

```
# NO
[[0] * 3] * 3
# YES
[[0 for _ in range(3)] for _ in range(3)]
```

```
>>> a = [[0] * 3] * 3
>>> a[0][0] = 1
>>> a
[[1, 0, 0], [1, 0, 0], [1, 0, 0]]
```

Finish

Thank you for your attention!