Migrate Python from 2.X to 3.X
PHILIPPE BOULANGER

C++ & PYTHON DEVELOPER

6,5 years in CAD&Numeric simulations
3 years in embedded programming
11,5 years in finance
20,5 years in C++
19,5 years in Python

@Pythonicien
CONCERNS
2017

- Instagram migrated major part of its code to Python 3

September 2018

- Dropbox announced the end of its migration to Python 3 (they began in 2015)!
Support for Python 2.7 will stop soon
- January the 1st 2020

Some libraries are no more compliant
- Django, numpy (2019), etc.

Python 4
- Will arrive in the next few years (2023 ?)
▪ Asynchronous programming (asyncio)

▪ Consistency
  - Return generator instead of containers
  - Functional programming
### DIFFERENCES (1/3)

<table>
<thead>
<tr>
<th></th>
<th>Python 2</th>
<th>Python 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print</code></td>
<td><code>print 'blabla'</code></td>
<td><code>print('blabla')</code></td>
</tr>
</tbody>
</table>

---

---

---
### DIFFERENCES (1/3)

<table>
<thead>
<tr>
<th></th>
<th>Python 2</th>
<th>Python 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>print</td>
<td><code>print 'blabla'</code></td>
<td><code>print('blabla')</code></td>
</tr>
<tr>
<td>raise</td>
<td><code>raise IOError, 'file error'</code></td>
<td><code>raise IOError('file error')</code></td>
</tr>
</tbody>
</table>

---

---

---
# DIFFERENCES (1/3)

<table>
<thead>
<tr>
<th></th>
<th>Python 2</th>
<th>Python 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>print</strong></td>
<td><code>print 'blabla'</code></td>
<td><code>print('blabla')</code></td>
</tr>
<tr>
<td><strong>raise</strong></td>
<td><code>raise IOError, 'file error'</code></td>
<td><code>raise IOError('file error')</code></td>
</tr>
<tr>
<td><strong>long</strong></td>
<td><code>long(myvar)</code></td>
<td><code>int(myvar)</code></td>
</tr>
<tr>
<td></td>
<td><code>5/2 = 2</code></td>
<td><code>5/2 = 2.5</code></td>
</tr>
<tr>
<td></td>
<td><code>5//2 = 2</code></td>
<td><code>5//2 = 2</code></td>
</tr>
</tbody>
</table>
## DIFFERENCES (2/3)

<table>
<thead>
<tr>
<th>string</th>
<th>Python 2</th>
<th>Python 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>unicode</td>
<td>str</td>
<td>bytes</td>
</tr>
<tr>
<td>str</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## DIFFERENCES (2/3)

<table>
<thead>
<tr>
<th></th>
<th>Python 2</th>
<th>Python 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>string</strong></td>
<td><strong>unicode</strong></td>
<td><strong>str</strong></td>
</tr>
<tr>
<td></td>
<td><strong>str</strong></td>
<td><strong>bytes</strong></td>
</tr>
<tr>
<td><strong>dict, map, zip</strong></td>
<td><strong>dict.items(): list</strong></td>
<td><strong>dict.items(): dict_items</strong></td>
</tr>
<tr>
<td></td>
<td><strong>dict.keys()[0]</strong></td>
<td><strong>list(dict.keys())[0]</strong></td>
</tr>
<tr>
<td></td>
<td><strong>dict.iteritems()</strong></td>
<td><strong>dict.items()</strong></td>
</tr>
</tbody>
</table>
Differences (3/3)

Unicode Management

```
u'toto'
b'titi'

instruction unicode()
method __unicode__()
StringIO/BytesIO
```
FINANCIAL ASPECT

- **Migration costs?**
  - Heavy costs at short term
  - Few costs at long term

- **Costs to keep Python 2?**
  - No immediate costs
  - Heavy cost at middle/long term
NON-REGRESSION TESTS
HOW TO VALIDATE THE MIGRATION?

- Is the migration a success?
- Are the performances as good as the 2.X version?
- What is the coverage of the tests?
- We need indicators!
▪ Use a unit test for a small part of code testing (As a function).

▪ Utopic goal: have unit tests for all API.
Functional tests are more complex because several API are linked but cover a real service or functionality.

Objective: cover most of the functionalities as possible. Having a tool to measure code coverage will be useful.
The need to automate tests is increasing with program size.

A functional test could be:
- A chain of API calls
- GUI actions (use of UFT/QTP)
You need to validate that migration keep the application performances: algorithms used in libraries could be replaced between versions, some conflicts between libraries could appear...

Load tests?
Knowing the number of lines of codes tested when all the tests are using (unit, functional or performance)

https://coverage.readthedocs.io/en/coverage-4.5.1a
According to my experience, with less than 60% of covered code, the chance of having hidden bugs is very important.

A good target is 80% of covered code.
- Test the GUI
  - Either manual
  - Or use tool like UFT (previous name: QTP)
  - ...

- Allow to automate test as if it was done by a user
A developer tests the code from a way which corresponds to the implementation he done, a real user tests according to its habits

- Update GUI controls, click... raise events and code execution and the order of calls can change the behavior
- human add random part inside tests
PERIMETER?
WHAT MODULES ARE LOADED? (1/2)

- Standard modules in Python?
- Modules which were developed in intern?
- What are the external modules?
How to determine the list of dynamically loaded modules... Available since Python 2.3.

```python
import re, itertools

try:
    import baconhameggs
except ImportError:
    pass

try:
    import guido.python.ham
except ImportError:
    pass

from modulefinder import ModuleFinder

finder = ModuleFinder()
finder.run_script('bacon.py')

print 'Loaded modules:
for name, mod in finder.modules.iteritems():
    print '%s: ' % name,
    print ','.join(mod.globalnames.keys()[:3])

print '-'*50
print 'Modules not imported:
print '\n'.join(finder.badmodules.iterkeys())
```
AND THEN: PROBLEMS?...

- Is there module:
  - with ended support or unmigrated?
  - with modified API?
  - licencing changed?
  - library name changed?
Non-Python Modules

- Problems with C/C++ written modules
  - C++ compiler migration
  - porting C++ libraries
  - tools problems (swig...)
  - licences, etc...
MIGRATION METHODOLOGY
- «Divide to reign»: it will be better to migrate small groups of files to minimize interactions.

- Create bundles in using module dependencies (have a graph should be useful), internal or external module...
PORTING EXTERNAL CODE (1/2)

- External code has no dependency with house-made code: start with them will be a good idea.

- Take count of tools:
  - Compiler
  - Integration tools in Python: swig, boost.python, etc.
  - External libraries
Library was ported or not?
API changed?
Licensing changed?
Is there constraints according to the versions of different libraries?
Is source code available?
- from __future__ import division
  - PEP 238: Changing the Division Operator

- from __future__ import print_function
  - PEP 3105: Make print a function
from __future__ import absolute_import
    - PEP 328: Imports: Multi-Line and Absolute/Relative

from __future__ import unicode_literals
    - PEP 3112: Bytes literals in Python 3000
ADD PYTHON 3.X CHANGES INSIDE PYTHON 2.X CODE (3/3)

- Six: six.readthedoc.io

### Python 2

```python
from urllib2 import urlopen

my_url = 'http://myurl.net'

try:
    x = urlopen(my_url).read()
    print(x)
except Exception, e:
    raise IOError, 'Error 404'
```

### Python 3

```python
from urllib.request import urlopen

my_url = 'http://myurl.net'

try:
    x = urlopen(my_url).read()
    print(x)
except Exception as e:
    raise IOError('Error 404')
```

### Six to add 3.X features in 2.X code

```python
from six.urllib.request import urlopen

my_url = 'http://myurl.net'

try:
    x = urlopen(my_url).read()
    print(x)
except Exception as e:
    raise IOError('Error 404')
```
2to3

```
from urllib.request import urlopen
my_url = "http://pythonprogramming.net"
try:
x = urlopen[my_url].read()
- print x
- except Exception, e:
+ raise IOError, "Error 404"
+ print(x)
+ except Exception as e:
  + raise IOError("Error 404")
```
2to6

- Based on 2to3
- For compilancy between 2 and 3
- Add __future__, six
REFACTORING
```python
def nb_file_listdir(path, ext):
    nb = 0
    for name in.listdir(path):
        fname = F"{path}\{name}"
        if.isdir(fname):
            nb += nb_file_listdir(fname, ext)
        else:
            r, e = splitext(name)
            if e.lower() == ext:
                nb += 1
    return nb

print(nb_file_listdir(PATH, ".py"))
```

```python
PATH = "C:\\Tools\\Anaconda3"
def nb_file_scandir(path, ext):
    nb = 0
    for entry in scandir(path):
        if entry.is_dir():
            nb += nb_file_scandir(entry.path, ext)
        else:
            r, e = splitext(entry.name)
            if e.lower() == ext:
                nb += 1
    return nb

print(nb_file_scandir(PATH, ".py"))
```

---

**Fonction/Module**

<table>
<thead>
<tr>
<th>Fonction/Module</th>
<th>Appel</th>
<th>Durée totale</th>
<th>Durée locale</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_file_listdir</td>
<td>18214</td>
<td>7.14 sec</td>
<td>212.04 ms</td>
</tr>
<tr>
<td>nb_file_scandir</td>
<td>18214</td>
<td>1.54 sec</td>
<td>572.59 ms</td>
</tr>
</tbody>
</table>

Use generators

```python
def frange(a, b, n):
    h = (b - a) / n
    for i in range(n+1):
        yield a + i * h
for x in frange(0, 1, 10):
    print(x)
```

```python
a = 0
b = 1
n = 10
h = (b - a) / n
for x in (a + i * h for i in range(n+1)):
    print(x)
```
Comprehension containers

```python
A = [ 1, 2, 3, 4 ]
B = {}
for x in A:
    B[ str(x) ] = x
```
String format

name = "Toto"

"My name is %s" % ( name ) # since 1.x

"My name is {}".format( name ) # since 2.0

f"My name is {name}" # since 3.6
REFACTORIZATION (5/6)

- JIT compiler: numba

```python
def fib1(n):
    if n < 2:
        return n
    return fib1(n - 1) + fib1(n - 2)
print(fib1(35))

from numba import jit
@jit
def fib2(n):
    if n < 2:
        return n
    return fib2(n - 1) + fib2(n - 2)
print(fib2(35))
```

<table>
<thead>
<tr>
<th>Fonction/Module</th>
<th>Durée totale</th>
<th>Durée locale</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib1</td>
<td>5.52 sec</td>
<td>5.52 sec</td>
</tr>
<tr>
<td>_find_and_load</td>
<td>353.08 ms</td>
<td>2.87 ms</td>
</tr>
<tr>
<td>_handle_fromlist</td>
<td>342.76 ms</td>
<td>1.54 ms</td>
</tr>
<tr>
<td>_compile_for_args</td>
<td>119.95 ms</td>
<td>7.76 us</td>
</tr>
<tr>
<td>fib2</td>
<td>69.17 ms</td>
<td>69.17 ms</td>
</tr>
<tr>
<td>jit</td>
<td>23.36 ms</td>
<td>10.93 us</td>
</tr>
</tbody>
</table>
Cache strategy

```python
from functools import lru_cache as cache

@cache(maxsize=None)
def fib(n):
    if n<2:
        return n
    return fib(n-1) + fib(n-2)

x = [fib(i) for i in range(35)]
print(x)
```
TO CONCLUDE
AND NOW...

- Migrations are like children: each of them is different

- Split in steps...
  - After each step, TEST!!!!

- You will have difficulties but keep hope.
CONTACT
Philippe BOULANGER
Python Expertise Manager
Philippe.boulanger@invivoo.com

www.invivoo.com
www.blog.invivoo.com
www.xcomponent.com

PARIS
13, Rue de l’abreuvoir
92400 Courbevoie

BORDEAUX
Rue Lucien Faure
33000 Bordeaux

LONDRES
Landsdowne House / City Forum
250 City Road – London EC1V 2PU