deepsense.io
BIG DATA SCIENCE

Neptune dla Python 2/3

Tomasz Żołnowski

Python 2 czy 3?

deepsense.io
BIG DATA SCIENCE

Python 2 czy 3?

deepsense.io
BIG DATA SCIENCE

3

Python 2 czy 3?

deepsense.io
BIG DATA SCIENCE

- Ujednolicone interfejsy
- Uporządkowana biblioteka standardowa
- Obsługa unicode
- Nowe biblioteki
- **Koniec wsparcia dla Python 2!**

http://pythonclock.org/

Nasz problem

deepsense.io
BIG DATA SCIENCE

- Mieliśmy CLI i API dla użytkownika w python 2.7
- Potrzebne jednoczesne wsparcie dla gałęzi 2 i 3

- Co udało się zrobić?

Neptune

Nasz problem

deepsense.io
BIG DATA SCIENCE

- Środowisko dla testów
- Automatyczna konwersja
- Pozostałe poprawki i stabilizacja testów
- Guideline dla dalszego utrzymania

Neptune

Narzędzie futurize

deepsense.io
BIG DATA SCIENCE

```
$ cat hello.py
print ‘hello world’

$ futurize hello.py > 2to3.patch

$ cat 2to3.patch
--- hello.py    (original)
+++ hello.py    (refactored)
@@ -1 +1,2 @@
-print 'hello world!'
+from __future__ import print_function
+print('hello world!')

$ patch < 2to3.patch

$ cat hello.py
from __future__ import print_function
print('hello world!')
```

Składnia - print

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
print 'hello world'
```

```
# Python 2 and 3
print('hello world')
```

Składnia - print

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
print >> sys.stderr, 'hello world',
```

```
# Python 2 and 3
from __future__ import print_function

print('hello world', file=sys.stderr, end='')
```

Składnia - wyjątki

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
try:
  raise ValueError, 'message'
catch ValueError, e:
  ...
```

```
# Python 2 and 3
try:
  raise ValueError('message')
catch ValueError as e:
  ...
```

Składnia - moduły

deepsense.io
BIG DATA SCIENCE

```
mypackage/
     __init__.py
     submodule1.py
     submodule2.py
main.py
```

```
$ cat mypackage/submodule1.py
import submodule2      # Python 2 only
```

```
# Python 2 and 3
from __future__ import absolute_import
from . import submodule2
```

Liczby - definiowanie

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only

x = 123
x = 432520032744898560000L
```

```
# Python 2 and 3

x = 123
x = 43252003274489856000
```

Liczby - definiowanie

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only

mode = 0644
```

```
# Python 2 and 3

mode = 0o644
```

Liczby - dzielenie

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
assert 3 / 2 == 1
```

```
# Python 2 and 3
assert 3 // 2 == 1
```

```
# Python 2 and 3
from __future__ import division
assert 3 / 2 == 1.5
```

```
# Python 2 and 3
from future.utils import old_div
assert old_div(3, 2) == 1
```

Liczby - sprawdzanie typów

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
assert isinstance(x, (int, long))

# Python 3 only
assert isinstance(x, int)
```

```
# Python 2 and 3, option 1
from future.builtins import int
assert isinstance(x, int)
```

```
# Python 2 and 3, option 2
from past.builtins import long
assert isinstance(x, (int, long))
```

Teksty - definiowanie

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only

s1 = ‘abc’
s2 = u‘zażółć gęślą jaźń’
```

```
# Python 2 and 3

s1 = u’abc’
s2 = u‘zażółć gęślą jaźń’
```

Teksty - konwersja

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only

template = 'mystring is: %s' % unicode(mystring)
```

```
# Python 2 and 3
from future.builtins import str

template = 'mystring is: %s' % str(mystring)
```

Teksty - konwersja

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only

template = 'mystring is: %s' % unicode(mystring)
```

```
# Python 2 and 3
from future.builtins import str as text

template = 'mystring is: %s' % text(mystring)
```

Teksty - odczyt pliku

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only

with open('myfile.txt') as f:
  data = f.read()
  utf_text = data.decode('utf-8')
  ...
```

```
# Python 2 and 3
from io import open

with open('myfile.txt', encoding='utf-8') as f:
  utf_text = f.read()
  ...
```

Teksty - iteracja po bajtach

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
for bytechar in 'bytes with \xf8':
  ...

# Python 3 only
for myint in b'bytes with \xf8':
  bytechar = bytes([myint])
  ...
```

```
# Python 2 and 3
from future.builtins import bytes

for myint in bytes(b'bytes with \xf8'):
  bytechar = bytes([myint])
  ...
```

Teksty - chr jako unicode

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
assert unichr(8364) == u'€'

# Python 3 only
assert chr(8364) == u'€'
```

```
# Python 2 and 3
from future.builtins import chr

assert chr(8364) == u'€'
```

Teksty - chr jako bytes

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
assert chr(36) == b'$'
assert chr(0xf8) == b'\xf8'

# Python 3 only
assert chr(36).encode('latin-1') == b'$'
assert chr(0xf8).encode('latin-1') == b'\xf8'
```

```
# Python 2 and 3
from future.builtins import chr

assert chr(36).encode('latin-1') == b'$'
assert chr(0xf8).encode('latin-1') == b'\xf8'
```

Teksty - chr jako bytes

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
assert chr(36) == b'$'
assert chr(0xf8) == b'\xf8'

# Python 3 only
assert bytes([36]) == b'$'
assert bytes([0xf8]) == b'\xf8'
```

```
# Python 2 and 3
from future.builtins import bytes

assert bytes([36]) == b'$'
assert bytes([0xf8]) == b'\xf8'
```

Teksty - przeciążanie __str__

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only

class MyClass(object):

  def __unicode__(self):
    return u'Unicode with \u20ac'

  def __str__(self):
    return unicode(self).encode('utf-8')
```

Teksty - przeciążanie __str__

deepsense.io
BIG DATA SCIENCE

```
# Python 2 and 3
from future.utils import python_2_unicode_compatible

@python_2_unicode_compatible
class MyClass(object):

  def __str__(self):
    return u'Unicode with \u20ac'
```

Teksty - sprawdzanie typów

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
u = u'abc'
b = 'def'

assert (isinstance(u, basestring) and
        isinstance(b, basestring))
```

```
# Python 2 and 3, option 1
from past.builtins import basestring

u = u'abc'
b = b'def'

assert (isinstance(u, basestring) and
        isinstance(b, basestring))
```

Teksty - sprawdzanie typów

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
u = u'abc'
b = 'def'

assert (isinstance(u, basestring) and
        isinstance(b, basestring))
```

```
# Python 2 and 3, option 2
from future.builtins import str

u = u'abc'
b = b'def'
b_str = b.decode()

assert (isinstance(u, str) and
        isinstance(b_str, str))
```

Kontenery - range

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
mylist = range(3)
assert mylist == [0, 1, 2]

# Python 2 and 3
mylist = list(range(3))       # inefficient in Py2
assert mylist == [0, 1, 2]
```

```
# Python 2 and 3
from future.builtins import range

mylist = list(range(3))
assert mylist == [0, 1, 2]
```

Kontenery

deepsense.io
BIG DATA SCIENCE

map, zip, filter

tak samo jak range

Kontenery - iteracja po kluczach

deepsense.io
BIG DATA SCIENCE

```
d = {'alice': 7, 'bob': 8, 'charlie': 5}

# Python 2 only
for key in d.iterkeys():
  ...
```

```
# Python 2 and 3
for key in d:
  ...
```

Kontenery - klucze jako lista

deepsense.io
BIG DATA SCIENCE

```
d = {'alice': 7, 'bob': 8, 'charlie': 5}

# Python 2 only
keylist = d.keys()
assert isinstance(keylist, list)
```

```
# Python 2 and 3
keylist = list(d)
assert isinstance(keylist, list)
```

Kontenery - iteracja po wartościach

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
for value in d.itervalues()
  ...

# Python 2 and 3
for value in d.values():        # inefficient in Py2
  ...
```

```
# Python 2 and 3, option 1
from future.builtins import dict

d = dict(alice: 7, bob: 8, charlie: 5)
for value in d.values():
  ...
```

Kontenery - iteracja po wartościach

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
for value in d.itervalues()
  ...

# Python 2 and 3
for value in d.values():        # inefficient in Py2
  ...
```

```
# Python 2 and 3, option 2
from future.utils import itervalues

for value in itervalues(d):
  ...
```

Kontenery - wartości jako lista

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
values = d.values()

# Python 2 and 3
values = list(d.values())    # inefficient in Py2
```

```
# Python 2 and 3, option 1
from future.builtins import dict

d = dict(alice: 7, bob: 8, charlie: 5)
values = list(d.values())
```

Kontenery - wartości jako lista

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
values = d.values()

# Python 2 and 3
values = list(d.values())    # inefficient in Py2
```

```
# Python 2 and 3, option 2
from future.utils import listvalues

values = listvalues(d)
```

Kontenery - wartości jako lista

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
values = d.values()

# Python 2 and 3
values = list(d.values())    # inefficient in Py2
```

```
# Python 2 and 3, option 3
from future.utils import itervalues

values = list(itervalues(d))
```

Kontenery - własny iterator

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only

class Upper(object):

  def __init__(self, s):
    self._iter = iter(s)

  def __iter__(self):
    return self

  def next(self):          # Py2 style
    return self._iter.next().upper()


it = Upper('hello')
assert it.next() == 'H'  # Py2 style
assert list(it) == list('ELLO')
```

Kontenery - własny iterator

deepsense.io
BIG DATA SCIENCE

```
# Python 2 and 3, option 1
from future.builtins import object

class Upper(object):

  def __init__(self, s):
    self._iter = iter(s)

  def __iter__(self):
    return self

  def __next__(self):    # Py3 style
    return next(self._iter).upper()


it = Upper('hello')
assert next(it) == 'H'    # compatible style
assert list(it) == list('ELLO')
```

Kontenery - własny iterator

deepsense.io
BIG DATA SCIENCE

```
# Python 2 and 3, option 2
from future.utils import implements_iterator

@implements_iterator
class Upper(object):

  def __init__(self, s):
    self._iter = iter(s)

  def __iter__(self):
    return self

  def __next__(self):    # Py3 style
    return next(self._iter).upper()
```

Inne - standardowe biblioteki

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
from StringIO import StringIO
#or
from cStringIO import StringIO
```

```
# Python 2 and 3
from io import BytesIO
```

Inne - standardowe biblioteki

deepsense.io
BIG DATA SCIENCE

```
# Python 2 only
from urlparse import urlparse
from urllib import urlencode


# Python 3 only
from urllib.parse import urlparse, urlencode
```

Inne - standardowe biblioteki

deepsense.io
BIG DATA SCIENCE

```
# Python 2 and 3, option 1

try:
  from urllib.parse import urlparse, urlencode
except ImportError:
  from urlparse import urlparse
  from urllib import urlencode
```

Inne - standardowe biblioteki

deepsense.io
BIG DATA SCIENCE

```
# Python 2 and 3, option 2

from future.moves.urllib.parse import (urlparse, urlencode)
```

Inne - standardowe biblioteki

deepsense.io
BIG DATA SCIENCE

```
# Python 2 and 3, option 3

from future.standard_library import hooks

with hooks():
  from urllib.parse import urlparse, urlencode
```

Krótko o Neptune

deepsense.io
BIG DATA SCIENCE

Platforma wspierająca przeprowadzanie eksperymentów Machine Learning.

- Śledzenie kodu źródłowego, parametrów, środowiska uruchomienia.
- Śledzenie wyników - dane kanałów, wykresy, logi, podgląd obrazów.
- Uruchamianie obliczeń w chmurze, w tym na GPU.
- Porównywanie eksperymentów, definiowanie metryk, grid search.
- Praca w zespołach.

Neptune

Referencje

deepsense.io
BIG DATA SCIENCE

- http://python-future.org/compatible_idioms.html
- http://www.youtube.com/watch?v=KOqk8j11aAI
- http://www.youtube.com/watch?v=f_6vDi7ywuA
- http://www.youtube.com/watch?v=YgtL4S7Hrwo&feature=youtu.be&t=155

- http://deepsense.io/neptune/
- https://go.neptune.deepsense.io/

deepsense.io
BIG DATA SCIENCE

Pytania?

deepsense.io
BIG DATA SCIENCE

Dziękuję za uwagę

Tomasz Żołnowski
technical leader

tomasz.zolnowski@deepsense.io