python 3

Armin Ronacher
http://lucumr.pocoo.org/ // armin.ronacher@active-4.com // http://twitter.com/mitsuhiko

About Me

- using Python since version 2.2
- not-a-committer-with-commit-rights!?
- Part of the Pocoo Team: *Jinja, Werkzeug, Sphinx, Zine, Flask*

A little bit of History

1991

Python Appears

- Exceptions
- Multiple Inheritance
- C inspired IO system

1995

Python 1.5

- Regular Expressions
- Exceptions are Classes
- Built in Package Support
- It can be embedded!

2000

Python 2.0!

Spoiler: this is where it gets interesting

- Unicode Support
- Augmented assignments (+=, -= etc.)
- List Comprehensions
- Garbage Collector
- Python Enhancement Proposals!

Footnote: some of this was in 1.6 already

2004

Python 2.4!

Spoiler: unicode becomes interesting

- Source Encoding
- Boolean Type (True/False)
- sets
- reverse iteration
- generator expressions

Footnote: some of this was in 2.3 already

Not all is Fine

String Coercion

It's bytes — it's charpoints

```
>>> a = "Peter"
>>> b = u"Bärbel"
>>> "Hallo " + a
'Hallo Peter'
>>> "Hallo " + b
u'Hallo B\xe4rbel'
>>> print _
Hallo Bärbel
```

String Coercion [fail]

It's an exception

```
>>> a = u"Peter"
>>> b = "Bärbel"
Traceback (most recent call last):
  ...
UnicodeDecodeError: gibberish
```

Print is a Statement

why is that?

```
>>> print "Hello", 42
Hello 42
>>> print("Hello", 42)
('Hello', 42)
>>> x = print
Traceback (most recent call last):
  ...
SyntaxError: invalid syntax
```

O HAI — IM TEH ITERATOR

... and I want my place

```
>>> a = {'foo': 1, 'bar': 2}
>>> a.keys()
['foo', 'bar']
>>> a.iterkeys()
<dictionary-keyiterator object ...>
```

Exceptions

let's do some guessing

```
raise a
raise a, b, c
raise ((a, b), c), d
except A:
except A, B:
except (A, B), C:
```

Exceptions improved

that should clear it up:

```
raise a
raise b.with_traceback(c)
raise ((a, b), c), d
except A:
except A as B:
except (A, B) as C:
```

New Stuff in 3.x

non-local names

Re-bind variables from outer scopes:

```
def make_counter(initial=0):
    val = initial
    def inc():
        nonlocal val
        rv = val
        val += 1
        return rv
    return inc
```

New Literals

Set Literal

```
>>> a = {1, 2, 3}
>>> b = set()
```

Footnote: there is no literal for an empty set

Extended Unpacking

Unpack in variables

```
>>> first, second, *rest = unpack()
```

Footnote: The rest is a list with all the unassigned items

Abstract Base Classes

Duck Typing Improved

```
>>> class MyIterator(object):
...     def __next__(self):
...         raise StopIteration()
...     def __iter__(self):
...         return self
...
>>> from collections import Iterator
>>> isinstance(MyIterator(), Iterator)
True
```

Footnote: Implicit subclasses, awesome!

Annotations!

Add type information to your signatures

```
from numbers import Number

def add(a: Number, b: Number) -> Number
    return a + b
```

Footnote: These are unchecked, Sphinx can use them

Goodies

print a function, super with magic

```
>>> a = print
>>> a(42)
42
>>> class TagSet(set):
...     def add(self, tag):
...         super().add(tag.lower())
...
>>> type(input())
Hello World!
<type ‘str’>
```

Brrr: How does super() work? You don’t wanna know

New Land

Backwards Incompatible

be courageous

- Break up with the Past
- Fix the issues
- Help the transition with automation

Side By Side

how can we do that?

- Python 3.x and Python 2.x are developed side by side
- Python 2.x slowly adopts features that come from 3.x or make the automated conversion easier

Automation

2to3 for the rescue

- A tool to convert from Python 2.x sources to 3.x
- Operates on the parsing-time information only.
- Not 100% correct, but close

There Be Dragons

Step by Step #1

Library or Application?

- 2to3 caters for both
- But the experience differs
- generally: application easier

Step by Step #2

Unicode or Bytes?

- did your API use bytes?
- did it use unicode?
- what should it use now?

Step by Step #3

Are you sure?

- Beware of the stdlib
- things unicodified along the way
- explicit encoding!

Step by Step #4

Adapt to new idioms

- str.format instead of %
- keys() and not iterkeys(), beware on custom classes
- abstract base classes
- the new IO system

Step by Step #5

Port your Tests

- Make sure the tests run on 3.x
- Get rid of your doctests. **Now**!
- make sure 2to3 only has to correct syntax, not semantics. Runtime checks.


2to3 will byte you

Unicode Breakage

Affects many 2.x libraries

- class implements `__unicode__`
- and `__str__` calls into `__unicode__`
- on 3.x: RuntimeError
- Can be fixed with a custom fixer

Preprocessing

If everything else fails …

- write a preprocessor
- monkey-patch yourself in

```
if sys.version_info >= (3, 0):
    from preprocessor import refactor_string
    from lib2to3.refactor import RefactoringTool
    RefactoringTool.refactor_string = refactor_string
```

Footnote: that's what SQLAlchemy does

Changing Protocols

What if you depend on something?

- WSGI changes and is broken right now
- web applications are an unrealistic target for the time being.

Rejoice: WSGI 1.1 is coming around

Fixing with Fixers

What are Fixers?

2to3 does *not* work at runtime

- 2to3 is guessing
- has certain assumptions of your code
- fixers operate on a parse tree
- fixers can refactor and modify code

Anatomy of a Fixer

A fixer to rename functions

```
from lib2to3 import fixer_base
from lib2to3.fixer_util import Name, BlankLine

class FixAltUnicode(fixer_base.BaseFix):
    PATTERN = """
    func=funcdef< 'def' name='__unicode__'
                  parameters< '(' NAME ')' > any+ >
    """

    def transform(self, node, results):
        name = results['name']
        name.replace(Name('__str__', prefix=name.prefix))
```

Running custom Fixers

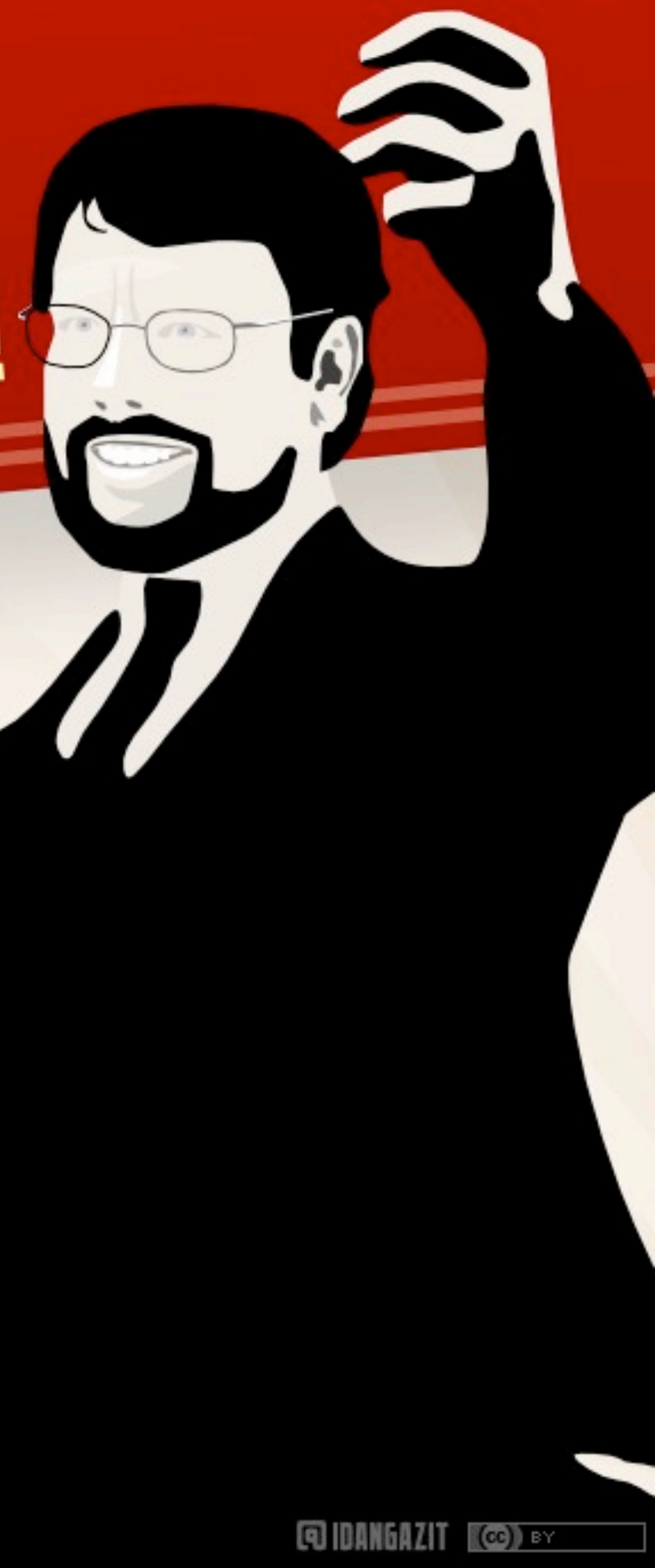
use distribute!

- distribute replaces setuptools
- has builtin 2to3 support
- *the* distribution system for 3.x

distribute: http://pypi.python.org/pypi/distribute

ATTENTION PYTHON COMRADES

NEW ORDERS FROM THE MINISTRY OF PACKAGING!



SONS AND DAUGHTERS OF THE GLORIOUS

PEOPLE'S PYTHONIC REPUBLIC

USE **DISTRIBUTE** AND **PIP**

THAT IS ALL.

THIS MESSAGE HAS BEEN APPROVED BY @JEZDEZ, HIGH CHANCELLOR OF PACKAGING

Hook into distribute

add this to your setup.py

```
extra = {}
if sys.version_info >= (3, 0):
    extra.update(
        use_2to3=True,
        use_2to3_fixers=['custom_fixers']
    )

setup(..., **extra)
```

distribute: http://pypi.python.org/pypi/distribute


GOODNEWS


Jinja

Porting Jinja2

What is it?

- Jinja2 — a template engine
- Codebase: 10KLOC
- In 2.x unicode based

Porting Jinja2

Porting Experience?

- # custom fixers: 3
- preprocessing? nope
- passing tests? all! (except for doctests in the documentation)

Porting Jinja2

An Example Fixer

```
from lib2to3 import fixer_base, pytree
from lib2to3.fixer_util import Name, BlankLine, Name, Attr, ArgList

class FixBrokenReraising(fixer_base.BaseFix):
    PATTERN = """
    raise_stmt< 'raise' any ',' val=any ',' tb=any >
    """

    # run before the broken 2to3 checker with the same goal
    # tries to rewrite it with a rule that does not work out for jinja
    run_order = 1

    def transform(self, node, results):
        tb = results['tb'].clone()
        tb.prefix = ''
        with_tb = Attr(results['val'].clone(), Name('with_traceback')) + \
                  [ArgList([tb])]
        new = pytree.Node(self.syms.simple_stmt, [Name('raise')] + with_tb)
        new.prefix = node.prefix
        return new
```

Recipe

follow these steps for enlightenment

1) port doctests to unittest or your test suite
2) decide on your API for Python 3
3) check if 2to3 works on the tests
4) make the tests work
5) run 2to3 on your own code
6) add runtime fixes or custom fixers

Unladen Swallow

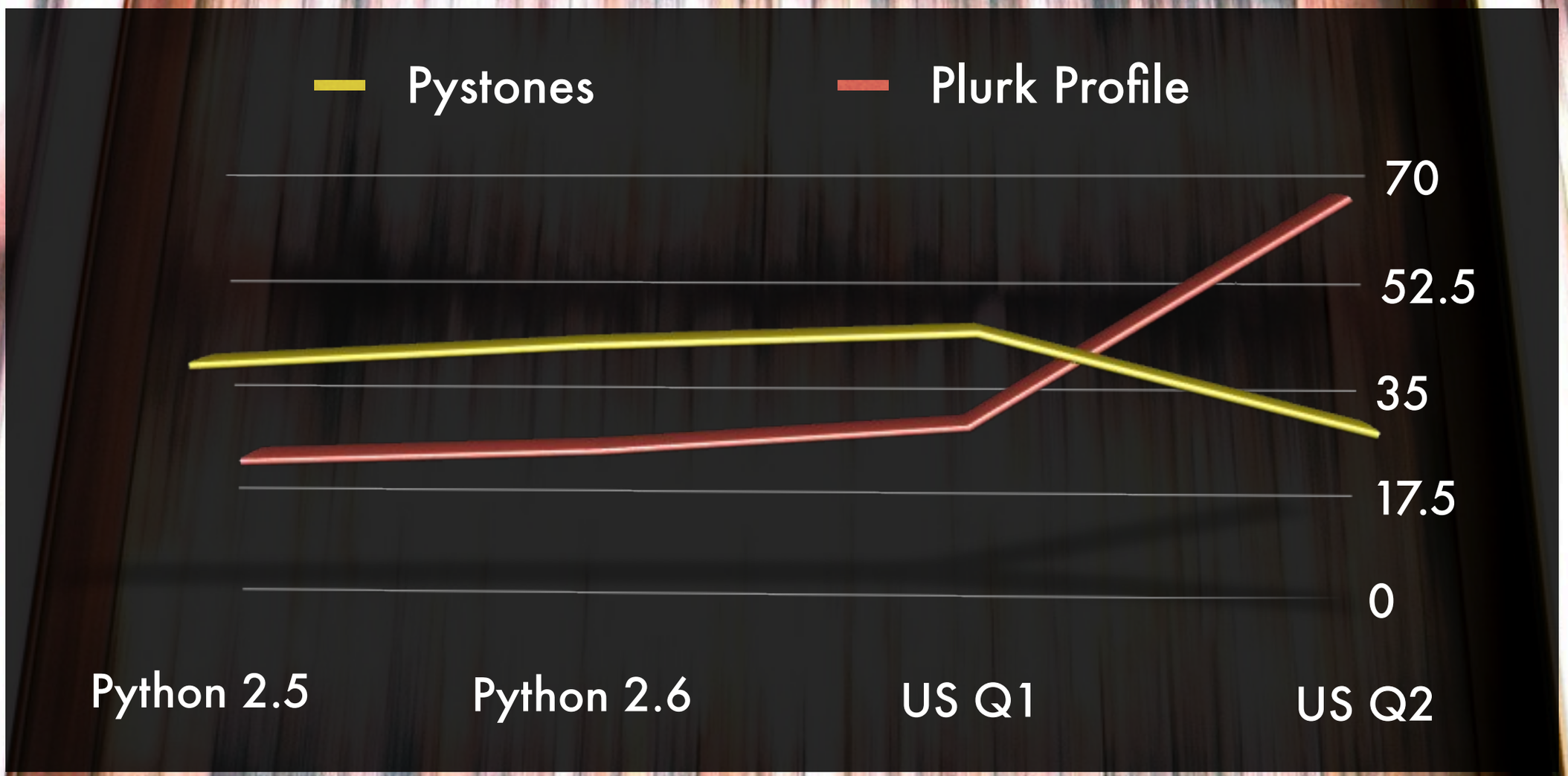
Unladen Swallow

Making Python Fast

1) LLVM JIT based
2) a development branch of Python
3) currently targeting 2.x, but major changes will land in 3.x

Numbers

Current Status



Current Status

Not very impressive

1) It's not there yet
2) There will be a development branch
3) Python is hard to optimize, will take some time

Ported Libraries

Ported Libraries

Web Related Libraries

- Jinja2
- Mako
- lxml
- SQLAlchemy
- httplib2

Ported Libraries

General Purpose

- blinker
- Pygments
- PyYAML
- tc [*Tokio Cabinet*]

Ported Libraries

Database / GUI

- sqlite3 [*part of stdlib*]
- py-postgresql / pg8000-py3
- PyQt

In The Pipeline

WSGI

- Werkzeug
- CherryPy
- mod_wsgi

Moving Target: WSGI 1.1 spec probably based on mod_wsgi for Python

Any Questions?

Legal