Upgrade your Python

Interesting new Idioms

Who am I

- Armin Ronacher / @mitsuhiko
- Founding member of the Pocoo Team
- Working on Flask, Jinja2, Werkzeug, Sphinx and more

Talk Focus

- Focus on Python 2.5 and newer
- Also have a look at features we can look forward when using Python 3

Python 2.5

- The new Python 2.3

Python 2.6

- Class decorators
- Abstract base classes
- New string formatting
- builtin with-statement
- Compile from AST

Python 2.7

- Dictionary views on Dictionaries (!?)
- New IO system
- Multiple arguments to with
- future imports
  - `print` as function
  - map/filter return iterables etc.
  - new string literals

Class Decorators

Why?

- More explicit alternative for metaclasses
  - can patch and replace
  - can be combined with metaclasses and other decorators

Plugin Interfaces

```
class Macro(object):
    macros = {}

    def __init__(self, arguments):
        self.arguments = arguments

    def render(self):
        raise NotImplementedError()

    @staticmethod
    def register(name):
        def decorator(cls):
            macros[name] = cls
            return cls
        return decorator

    @staticmethod
    def by_name(name):
        return Macro.macros.get(name)
```

A Plugin

```
from thatwiki import Macro

@Macro.register('RecentChanges')
class RecentChangesMacro(Macro):

    def render(self):
        return 'render all changes'
```

Heavy Functions

```
@app.route('/users/')
class Users(Controller):

    def get(self):
        return the list of users

    def post(self):
        return create a new user instead


@app.route('/users/<id:user_id>', methods=['GET'])
def show_user(request, user_id):
    return show the user
```

Creating Instances

```
def to_dict(thing):
    return dict((k, v) for k, v in thing.__dict__.iteritems()
                if not k.startswith('_'))


@to_dict
class Settings(object):
    DEBUG = True
    APPLICATION_NAME = 'Testing'
    SUBTITLE = 'Python is a cool thing'
```

The Funny Descriptor

The Empty Descriptor

Non-Data

You all used it

```
>>> class Foo(object):
...  def foo(self):
...   pass
...
>>> Foo.foo.__get__
<method-wrapper '__get__' of instancemethod object at
0x1004551e0>


>>> hasattr(Foo.foo, '__set__')
False
>>> hasattr(Foo.foo, '__delete__')
False
```

Caching Things

```
>>> request = Request(environ)
# nothing happened so far

>>> request.args
MultiDict({‘foo’: u’bar’})
# the request arguments were now parsed and stored

>>> request.args
MultiDict({‘foo’: u’bar’})
# this returns the very same object as above but no
# function is called any more.  Magic?
```

It's a monkeypatch

```
_missing = object()

class cached_property(object):

    def __init__(self, func):
        self.func = func
        self.__name__ = func.__name__
        self.__doc__ = func.__doc__
        self.__module__ = func.__module__

    def __get__(self, obj, type=None):
        if obj is None:
            return self
        value = obj.__dict__.get(self.__name__, _missing)
        if value is _missing:
            value = self.func(obj)
            obj.__dict__[self.__name__] = value
        return value
```

JFTR

```
$ python -mtimeit -s 'from werkzeug import Request; \
  r = Request.from_values("?foo=bar")' 'r.args'
10000000 loops, best of 3: 0.0629 usec per loop

$ python -mtimeit -s 'from werkzeug import Request; \
  r = Request.from_values("?foo=bar")' 'int()'
10000000 loops, best of 3: 0.101 usec per loop
```

Mixins

Multiple Inheritance

- Python has Multiple Inheritance
- Multiple Inheritance is not a bad thing
- It does interfaces and mixin classes

Real World

```
class Request(BaseRequest, AcceptMixin, ETagRequestMixin,
              UserAgentMixin, AuthorizationMixin,
              CommonRequestDescriptorsMixin):
    pass


class Response(BaseResponse, ETagResponseMixin,
               ResponseStreamMixin,
               CommonResponseDescriptorsMixin,
               WWWAuthenticateMixin):
    pass
```

I'm serious

```
class Mapping(Sized, Iterable, Container):
    ...

class Set(Sized, Iterable, Container):
    ...

class Sequence(Sized, Iterable, Container):
    ...
```

Dead serious

```
class OrderedDict(MutableMapping)
 |  Dictionary that remembers insertion order
 |
 |
 |  Method resolution order:
 |      OrderedDict
 |      MutableMapping
 |      Mapping
 |      Sized
 |      Iterable
 |      Container
 |      object
```

Okay, I cheated

```
class OrderedDict(dict, MutableMapping)
 |  Dictionary that remembers insertion order
 |
 |  Method resolution order:
 |      OrderedDict
 |      dict
 |      MutableMapping
 |      Mapping
 |      Sized
 |      Iterable
 |      Container
 |      object
```

Anyways

```
class AcceptMixin(object):

    @cached_property
    def accept_mimetypes(self):
        return parse_accept_header(
            self.environ.get('HTTP_ACCEPT'), MIMEAccept)

    @cached_property
    def accept_charsets(self):
        return parse_accept_header(
            self.environ.get('HTTP_ACCEPT_CHARSET'),
            CharsetAccept)
```

Abstract Base Classes

Not just inheritance

```
>>> from collections import Iterator
>>> class Foo(object):
...  def __iter__(self):
...   return self
...  def next(self):
...   return 42
...
>>> foo = Foo()
>>> isinstance(foo, Iterator)
True
>>> foo.next()
42
>>> foo.next()
42
```

But inheritance too

```
from collections import Mapping

class Headers(Mapping):

    def __init__(self, headers):
        self._headers = headers

    def __getitem__(self, key):
        ikey = key.lower()
        for key, value in self._headers:
            if key.lower() == ikey:
                return value
        raise KeyError(key)

    def __len__(self):
        return len(self._headers)

    def __iter__(self):
        return (key for key, value in self._headers)
```

And it's pretty sweet

```
>>> headers = Headers([('Content-Type', 'text/html')])
>>> headers['Content-type']
'text/html'
>>> headers.items()
[('Content-Type', 'text/html')]
>>> headers.values()
['text/html']
>>> list(headers)
['Content-Type']
```

New Rules

```
callable(x)           ->  isinstance(x, Callable)
tryexcept(hash(x))    ->  isinstance(x, Hashable)
tryexcept(iter(x))    ->  isinstance(x, Iterable)
tryexcept(len(x))     ->  isinstance(x, Sized)
tryexcept(hasattr(x, '__contains__'))
                      ->  isinstance(x, Container)

                      ->  isinstance(x, Mapping)
                          isinstance(x, Set)
                          isinstance(x, Sequence)
                          isinstance(x, MutableMapping)
                          isinstance(x, MutableSet)
                          isinstance(x, MutableSequence)
```

New String Formatting

Basic Formatting

```
>>> 'Hello {0}!'.format('World')
'Hello World!'

>>> 'Hello {0} {1}!'.format('Mr', 'World')
'Hello Mr World!'

>>> 'Hello {1}, {0}!'.format('Mr', 'World')
'Hello World, Mr!'

>>> 'Hello {name}!'.format(name='World')
'Hello World!'
```

This time . . . useful

```
>>> from datetime import datetime
>>> 'It\'s {0:%H:%M}'.format(datetime.today())
"It's 09:22"

>>> from urlparse import urlparse
>>> url = urlparse('http://pocoo.org/')
>>> '{0.netloc} [{0.scheme}]'.format(url)
'pocoo.org [http]'
```

My Suggestions

- Start using this for i18n. Why? Positions can be overridden in the translated string.
- Expose format strings instead of these printf thingies if possible.
- Provide `__format__` for your classes

Need 2.4/2.5 compat?

- We got you covered:
- http://bit.ly/stringfmt

`with` Statements

What has `with` ever done for us?

- Nicer interface for stack operations
- Guaranteed code execution on exit
- Ability to suppress tracebacks in a block

What hasn't it?

- It's not a Ruby block
- it's executed once, and you cannot control how (besides doing state changes in advance)

What has it really done?

- People are lazy
- I know I didn't close my files properly in small scripts and I'm pedantic…
- More correct applications / scripts
- Start of a good trend

Exhibit A

```
texture = Texture.from_file('textures/grass.png')
with texture:
    draw_all_quads()


transformation = Scale(1.5, 1.5, 1.5)
with transformation:
     render_the_scene()
```

So much nicer

```
glPushMatrix()
glRotate3f(45.0, 1, 0, 0)
glScalef(0.5, 0.5, 0.5)
glBindTexture(texture_id)
draw_my_object()
glBindTexture(0)
glPopMatrix()
```

```
with Matrix(), \
     Rotation(45.0, 1, 0, 0), \
     Scale(0.5, 0.5, 0.5), \
     texture:
    draw_my_object()
```

Exhibit B

```
with test_request_context():
    # setup a fake request context for testing purposes
    # for the duration of this block here.
```

Exhibit C

```
with my_log_handler:
     # everything that is logged here, is handled by
     # “my_log_handler”
     warning(‘This is pretty nifty’)
```

Exhibit D

```
    with pool.connection() as con:
        # get a connection from the pool and do something
        # with it here.  When everything works without
        # exception we commit, otherwise we roll back.
        # either way the connection goes back to the pool.
```

Exhibit E

```
with capture_stderr() as captured:
    execute code that might write to stderr
    assert captured.getvalue() == expected output
```

Nifty Tricks

- with block can catch down exceptions
- Combine with custom exceptions to do extra meta magic
- Not that I have found any use cases for that …

Things not to do

- Please don’t abuse with for XML/HTML generation
- Don’t use bytecode hacks to force Python to execute the body multiple times.

Little Things

Uniquifying Sequences

```
>>> list(OrderedDict.fromkeys([1, 1, 1, 2, 3, 3, 4, 5, 6]))
[1, 2, 3, 4, 5, 6]


>>> OrderedDict([(1, 2), (1, 3), (4, 2)]).items()
[(1, 3), (4, 2)]
```

Count Items #1

```
>>> from collections import Counter
>>> Counter('aaaaabc')
Counter({'a': 5, 'c': 1, 'b': 1})
>>> dict(Counter('aaaaabc'))
{'a': 5, 'c': 1, 'b': 1}
>>> dict(Counter([1, 1, 2, 3, 3, 4]))
{1: 2, 2: 1, 3: 2, 4: 1}
```

Count Items #2

```
>>> from collections import defaultdict
>>> d = defaultdict(int)
>>> d['foo'] += 42
>>> d['foo'] += 1
>>> d
defaultdict(<type 'int'>, {'foo': 43})
```

Enumerate with Index

```
>>> dict(enumerate(['hello', 'world'], 1))
{1: 'hello', 2: 'world'}
```

any() and all()

```
def has_header(headers, key):
    return any(k.lower() == key.lower()
               for k, v in headers)


def ensure_type(type, iterable):
    assert all(isinstance(obj, type) for obj in iterable)
```

Think Outside the Box

```
from itertools import izip, repeat

def batch(iterable, n):
    return izip(*repeat(iter(iterable), n))
```

Enter the Box

```
>>> def debug(*args):
...  print args
...
>>> debug(*repeat(iter([1, 2, 3, 4]), 2))
(<listiterator object at 0x100491e50>,
 <listiterator object at 0x100491e50>)


>>> iterator = iter([1, 2, 3, 4])
>>> zip(iterator, iterator)
[(1, 2), (3, 4)]
```

New Comprehensions

```
>>> {v: k for k, v in {'foo': 'bar'}.iteritems()}
{'bar': 'foo'}

>>> {x.lower() for x in [‘Content-Type’, ...]}
{‘content-type’, ...}
```

Upgrade your Tuples

```
>>> from collections import namedtuple
>>> Token = namedtuple('Token', ['type', 'value', 'lineno'])
>>> tok = Token('string', "Hello World!", 42)
>>> tok
Token(type='string', value='Hello World!', lineno=42)
```

Catching Exceptions

```
try:
    ...
except:
    ...
```

```
try:
    ...
except Exception:
    ...
```

Going Meta (the AST)

What's the AST?

- AST == Abstract Syntax Tree
- Let Python parse itself and show you what it looks like
- Modify the tree and compile it back

Playing with Source

```
>>> import ast
>>> node = ast.parse('def say_hello(name): '
...     print "Hello %s!" % name')
>>> node.body[0].body[0].values[0].left.s
'Hello %s!'
>>> node.body[0].body[0].values[0].left.s = 'Goodbye %s!''
>>> exec compile(node, '<stdin>', 'exec')
>>> say_hello('pycon')
Goodbye pycon!
```

Literal Eval

```
>>> import ast
>>> ast.literal_eval('[42, 23, "testing"]')
[42, 23, 'testing']

>>> ast.literal_eval('[42, 23, eval("1 + 2")]')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: malformed string
```

WHY?!

- Proper DSLs
- Implementation independent way to do generate executable Python code
- Helpful for things embedding Python such as template engines, PyFlakes etc.
- Python syntax in configuration files.

Also . . .

- Really horrible hacks:
- http://bitbucket.org/birkenfeld/karnickel

On the other hand . . .

- Might be a viable alternative to py.test's assertion re-evaluation thing
- So actually, less of a hack

Magic is the word

```
from karnickel import macro

@macro
def assign(variable, value):
    variable = value



from horriblemagic.__macros__ import assign

def testing():
    assign(variable_name, 42)
    return variable_name
```

Magic is the word

```
from karnickel import macro

@macro
def assign(variable, value):
    variable = value



from horriblemagic.__macros__ import assign

def testing():
    variable_name = 42
    return variable_name
```

Remember

- It’s fun until someone’s hurt.
- So don’t do it

Other things to avoid

- PyPy / Unleaden Swallow are upcoming
- So stop doing sys._getframe() in performance critical code

Python 3

```
def counter(initial=0):
    value = initial - 1
    def count():
        nonlocal value
        value += 1
        return value
    return count
```

Python 3

```
>>> a, *b = [1, 2, 3, 4]
>>> a
1
>>> b
[2, 3, 4]

>>> a, *b, c = [1, 2, 3, 4]
>>> a
1
>>> b
[2, 3]
>>> c
4
```

?

Go on, ask :-)

Slides will be at http://lucumr.pocoo.org/