Things you didn't know about Python

a presentation by Armin Ronacher
for PyCon South Africa 2012

@mitsuhiko
http://lucumr.pocoo.org/
Things you didn’t know about computers

a presentation by Armin Ronacher
for PyCon South Africa 2012

@mitsuhiko
http://lucumr.pocoo.org/
Things you didn’t know about computers and the world!!!
Everything is horrible
and nobody cares
We're doomed
[Untitled]
Motivation for this Talk

Who am I and why this talk
Armin Ronacher
Software Engineer at Fireteam
Game Middleware Provider
@mitsuhiko / @fireteamltd
We're using Python

And not just us.
Python has been popular in parts of in the gaming industry
I'm also doing Python Libraries
and help people online using them.
What we can learn from Wifi Hotspots
Starting Somewhere

Intended Login Mask
Down the Rabbit Hole

Served by Apache, PHP 5.3, Directory Listings
source code? Check!
SQL Injection? Check
Register Globals? Check
Debug Comments? Check
And Further

GPL Violation? Check
Pre generated voucher PDF? Check

Yay!
To Round it all Up

Comes with Instructions
Priorities

It's not secure if it does not have XML
A Step Back

What do Wifi Hotspots have to do with anything?
Python is not perfect

... but the criticism is very high level
“First World Problems”
Most of our problems with Python are not stopping us from using the language. It just might make it less pleasant.
Who is using Python?
Let's start with the marketing bits
Big Players

NASA, Disney, Youtube, Google, etc.
Trivia: Microsoft shipped Python in 96

Microsoft Merchant Server was written in Python in 1996
But really everybody
Python is one of the things that just shows up.
If for nothing else, then build scripts.
Trivia: Dropbox uses Python

Not just on the server, the client is also implemented in Python!
Gaming uses Python
Demonware, Agora, EA/ESN, Fireteam
Nobody got fired for choosing Python
We are the IBM of Dynamic Languages
History Lessons

A few interesting bits about the past
1991: Where all started

- Exceptions
- Multiple Inheritance
- C inspired IO system
Trivia: `string.py` was horrible

It had $O(n^2)$ upper/lower functions
Trivia: what did this do until 2.5?

```
raise ((a, b), c), d
```

answer: raises exception a with value d
Trivia: `mutex.py`

a module called mutex
not actually a mutex
survived until 2.7
1995: The Huge Jump to 1.5

- Regular Expressions
- Exceptions as classes
- Built-in package support
- Embeddable
Trivia: did you know re is 50% python?

the regular expression compiler is written in Python

You notice that when you `python -mtrace`
Trivia: why are `builtin` types lowercase?

because they used to be functions
the types where in `types.py`
types.StringType was the type of a string (camelcase)
2000: Modern Python: Python 2.0

- Unicode Support
- Augmented assignments (+= etc.)
- Garbage Collector
- PEPs
2004: Python as you know it

- File encoding cookies
- Boolean types
- Sets
- Reverse iteration
- Generator expressions
Trivia: 2.2.1 introduced True and False

... but no boolean type.
2.2.0: no true/false
2.3.0: real boolean type
Today: Evolving Language

- PyPy
- Python 3
Reasons for Popularity
key adopters and killer-apps
Really Early Adopters
Math and Scientific Community
Python's operator overloading and simple syntax was very convenient for scientific uses.
Trivia: Math Driven Syntax

\[
\text{foo}[1,\ldots,2] =\text{foo}[(1, \text{Ellipsis}, 2)]
\]
Other Factors

Python was easy to extend with C extensions and starting with distutils it was possible to distribute them
Windows!

Python has had excellent Windows support in the past unlike many other programming languages that were created in the POSIX environment.
Trivia: **Battlefield 2 used Python**

And since the engine is still used today there are free to play versions of Battlefield still using Python for scripting.
Web Development

We slowly and steadily became a proven platform for the web. Python is not the final answer there but an amazing platform to start with.
Twisted

If you wanted to do networking a few years ago Twisted was the answer
Trivia: Early XMPP Transports

Most of the XMPP to X transports were written in Python with Twisted
But Really …
It's fun!

People enjoy working with the language
I have yet to see a Wifi Hotspot Portal Page that is written in Python and sucks
**Disclaimer:** I understand that this statement is very optimistic and bad Python code exists in practice, that there are frameworks in the Python community that advocate for sloppy code, that there are widely used modules with security problems or bad general design, that there are indeed Wifi hotspot login pages that are horrible and indeed written in Python, that there are well written Wifi login pages in PHP (I don't actually believe that), that I am hugely biased and that my sample size is waaaaaaaaaaaaaaaaaay too small.
“FUN!?"

What is this?
Y U NO WORK
No Running into Walls
Descriptors

Python's most important language feature
What are Descriptors?

• `__get__`
• `__set__`
• `__delete__`
• Common descriptors: functions, properties
Trivia: Functions are Descriptors
that's what makes them methods if placed within classes
Example: Basic Descriptor Lookup

```python
>>> class Foo(object):
...     def my_function(self):
...         pass
...
>>> Foo.my_function
<unbound method Foo.my_function>
>>> Foo.__dict__['my_function']
<function my_function at 0x1002e1410>
>>> Foo.__dict__['my_function'].__get__(None, Foo)
<unbound method Foo.my_function>
>>> Foo().my_function
<bound method Foo.my_function of <__main__.Foo object at 0x1002e2710>>
>>> Foo.__dict__['my_function'].__get__(Foo(), Foo)
<bound method Foo.my_function of <__main__.Foo object at 0x1002e2750>>
```
Example: Everyday Decorators

```python
>>> class Foo(object):
...     @property
...     def foo(self):
...         return 'hello pycon'
...
>>> Foo().foo
'hhello pycon'
```
Cached Properties

```python
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
```
Example: Cached Properties

class Post(object):
    
    def __init__(self, text):
        self.text = text

    @cached_property
    def rendered_text(self):
        return markdown_to_html(self.text)
Duck Typing

“if it's not a penguin it must be a duck”
ABDT: Abstract Base Duck Typing
abstract bases for improved duck typing
Abstract Base Duck Typing

• `abc.ABCMeta` — metaclass for abstract bases
• `collections.*` — common abstract bases
Abstract Base Duck Typing

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)
Example: Abstract Base Duck Typing

```python
>>> from collections import Iterator
>>> class MyIter(object):
...    def __iter__(self):
...        return self
...    def next(self):
...        return 42
...>>> isinstance(MyIter(), Iterator)
True
```
Custom Ducks

from abc import ABCMeta, abstractmethod

class Markedup(object):
    __metaclass__ = ABCMeta

    @classmethod
def __subclasshook__(cls, C):
        if cls is Markedup:
            if hasattr(C, "__html__"):
                return True
        return NotImplemented
Example: Custom Ducks

```python
>>> class Markup(unicode):
...     def __html__(self):
...         return self
...
>>> isinstance(Markup('test'), Markedup)
True
```
Debugging Helpers
use internals to track down bugs
import sys
import __builtin__
real_import = __builtin__.__import__

def debug_import(name, locals=None, globals=None, fromlist=None, level=-1):
    glob = globals or sys._getframe(1).f_globals
    importer_name = glob and glob.get('__name__') or 'unknown'
    print '%s imports %s' % (importer_name, name)
    return real_import(name, locals, globals, fromlist, level)

__builtin__.__import__ = debug_import
Example: Tracking Imports

```python
>>> import urlparse
__main__ imports urlparse
urlparse imports collections
collections imports _abcoll
collections imports _collections
collections imports operator
collections imports keyword
collections imports sys
collections imports heapq
heapq imports itertools
heapq imports operator
heapq imports bisect
bisect imports _bisect
heapq imports _heapq
collections imports itertools
```
def print_frame_info(frame):
    print 'module: %s' % frame.f_globals.get('__name__')
    print 'filename: %s' % frame.f_code.co_filename
    print 'current line: %d' % frame.f_lineno
    loc = dict((k, v) for k, v in frame.f_locals.iteritems()
                if not k.startswith('__'))
    print 'local variables: %s' % loc
Example: Interpreter Frames

```python
>>> import sys
>>> print_frame_info(sys._getframe())
module: __main__
filename: <stdin>
current line: 1
local variables: {
    'a': 2,
    'b': 4,
    'sys': <module 'sys' (built-in)>,
    'print_frame_info': <function print_frame_info at 0x100484668>
}
```
import sys
import traceback

def dump_threads():
    for thread_id, frame in sys._current_frames().items():
        print 'Thread %d' % thread_id
        print ''.join(traceback.format_stack(frame))
Example: Dumping Threads

>>> import time, threading
>>> def foo():
...    for x in xrange(10):
...       time.sleep(1)
...
>>> threading.Thread(target=foo).start()
>>> dump_threads()
Thread #4302381056
   File "lib/python2.7/threading.py", line 483, in run
      self.__target(*self.__args, **self.__kwargs)
    File "<stdin>", line 3, in foo
       time.sleep(1)

Thread #140735295412576
   File "<stdin>", line 1, in <module>
      dump_threads()
   File "<stdin>", line 4, in dump_threads
      print ''.join(traceback.format_stack(frame)).rstrip()
Why we love Python
and why we don't use other things
Win because awesome Fail

This is how I “sell” Python
Slow Execution Monitor

• Dump stacktrace if an API request runs longer than N seconds
• Permanent background thread
• Request start -> set marker
• Request end -> remove marker
• If marker active for more than N seconds -> log stacktrace
Remote Console

• All Python code has a thread that listens for requests on redis
• Can be used to execute arbitrary Python code for debugging
• Sends results back to redis
Rich Logging

• We log into Sentry error groups
• all stacktraces on dev environments include local variables for all frames
• Also speaks $language
Sentry

Includes all information, groups automatically
Memory Leak Finding

- Walk over all objects the garbage collector can reach
- Resolve weak references
- Group by type
- Log to top grossing to graphite every second
Finding a Greenlet Leak

Easy to track down what exactly is happening, ~40 lines of code
Killer Libraries

• SQLAlchemy
• lxml
• *WSGI
• $webframework
virtualenv

• it does not matter that packaging or the import system is broken
• it could be so much worse
• virtualenv makes the big chaos into many separate small chaoses
Easy to Learn

- Language can be picked up in weeks
- It's a fun language to learn
- It's very dynamic and leaves room for (crazy) experimentation
not insane™
An Amazing Community

- Developers and people all around the world
- embraces simple and truly open licenses
- loves documentation
- … now also loves testing
screw hackernews
Worry Less & get stuff done