Python 2 and Python 3
Ottawa Canada Linux Users Group
April 5, 2011, Algonquin College

Ian Ward
http://genericconsulting.ca/
http://excess.org/
Python 2 and Python 3

- Language Basics
- Significant Libraries
- Development of Python Itself
- Alternative Implementations
- Examples
What is Python?

Python is an interpreted, general-purpose high-level programming language whose design philosophy emphasizes code readability

http://en.wikipedia.org/wiki/Python_(programming_language)

• 20 years old
• imperative, functional, OO w/multiple inheritance
• duck, dynamic, strong typing
Uncluttered Appearance

- ~30 keywords
- ~150 buitins (classes, functions, constants)

Less Punctuation

- Many operators are keywords: and, or, not, is, in
- Ending statements with ; is optional
- No end-block keyword or symbol
Documentation

http://docs.python.org/

- every version, every module
- easy to read
- reStructuredText (.rst)
- Sphinx
  
  http://sphinx.pocoo.org/
Being Pythonic

Zen of Python:

```python
>>> import this
```

http://www.python.org/dev/peps/pep-0020/
Python Popularity

def greet(name):
    print 'hello', name

greet('Jack')
greet('Jill')
We're Not the Only One

- ABC
- CoffeeScript
- Curry
- F# (if #light "off" is not specified)
- Haskell (only for where, let, do, or of clauses when braces are omitted)
- ISWIM, the abstract language that introduced the rule
- Python
- YAML

Prescience of Donald Knuth

The Future

It seems clear that languages somewhat different from those in existence today would enhance the preparation of structured programs. We will perhaps eventually be writing only small modules which are identified by name as they are used to build larger ones, so that devices like indentation, rather than delimiters, might become feasible for expressing local structure in the source language.

Configure Your Editor

![Editor Preferences Window]

- Tab Stops
  - Tab width: 4
  - Insert spaces instead of tabs

- Automatic Indentation
  - Enable automatic indentation

- File Saving
  - Create a backup copy of files before saving
  - Autosave files every 10 minutes

---

Python 2 and Python 3  Ian Ward  http://excess.org
Automated Clean Up

In the python source/installation:

```bash
Tools/scripts/reindent.py
```

On Debian or Ubuntu:

```bash
apt-get install python-examples
/usr/share/doc/python*/examples/Tools/scripts/reindent.py
```
Power of Import

http://xkcd.com/353/
Python for Science

NumPy and SciPy

- N-dimensional Array manipulations
- statistics
- optimization
- numerical integration
- linear algebra
- Fourier transforms
- signal processing
- image processing
- ODE solvers
- sophisticated random number capabilities
SciPy Optimization

http://www.scipy.org/

http://excess.org
Python for Games

Stackless

- Microthreads, Channels, Scheduling

http://www.stackless.com/

http://www.eveonline.com/
Python for the 'Net

Twisted

An event-driven networking engine

http://twistedmatrix.com/

- Servers and clients for HTTP, NNTP, IMAP, SSH, IRC, FTP and others
- With support for TCP, UDP, SSL/TLS, multicast and Unix sockets
Python Development

- BDFL
- Mailing Lists
- PEPs
  - rationale, explanation, related work
  - influenced Tcl, Erlang
Python 3

a.k.a. Python 3000, py3k

PEP 3000: "guidelines for Python 3000 development" - April 2006

http://www.python.org/dev/peps/pep-3000/

- intentionally break backwards compatibility
- clean up many long-standing language warts
- tools and plan for migration of external code from 2.x to 3.x
The Plan

- expect a 5 year transition (we're 2 years in now)
- all new development in 3.x
- continue to support 2.x and backport some features
- moratorium on language changes (just expired)
## External Project Support

<table>
<thead>
<tr>
<th>Project</th>
<th>Python 3 support</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumPy</td>
<td>YES</td>
</tr>
<tr>
<td>SciPy</td>
<td>no</td>
</tr>
<tr>
<td>Stackless</td>
<td>YES</td>
</tr>
<tr>
<td>Twisted</td>
<td>no</td>
</tr>
<tr>
<td>Django</td>
<td>no (promised this year)</td>
</tr>
<tr>
<td>PIL</td>
<td>no</td>
</tr>
<tr>
<td>Jython</td>
<td>no</td>
</tr>
<tr>
<td>Iron Python</td>
<td>no</td>
</tr>
<tr>
<td>PyPy</td>
<td>no</td>
</tr>
</tbody>
</table>

http://getpython3.net/

http://pypi.python.org/pypi?:action=browse&c=533
Alternative Implementations

Jython

- compiles to Java bytecode
- use Java libraries
- use JVM's garbage collector and JIT

Iron Python

- run code on the .Net CLI
- use .Net libraries

Shed Skin

Cython
PyPy

- 8 years old
- fast without code modification (~3.4x faster)
- optimize memory usage
- sandboxing
- stackless mode
- frontends for Python, JavaScript, Prolog, Smalltalk
  https://lwn.net/Articles/436970/
- backends for x86 32+64 bit, CLI (.Net), JVM (Java)
  http://pypy.org/
PyPy Compared to CPython

http://speed.pypy.org/

Plot 1: The above plot represents PyPy trunk (with JIT) benchmark times normalized to CPython. Smaller is better.
Strings

```
a = "these are the same"  # (8-bit in 2.x, unicode in 3.x)
b = 'these are the same'
c = b'binary data'
d = u'unicode'
e = r'c:\strings\with\backslashes'
f = r'\.\+\s(\w+)'

# ur'' and br'' work too

g = """Triple-quoted strings
may span multiple lines
and include the newlines ^^ they contain""
```
Common String Operations

```python
>>> a
'these are the same'
>>> len(a)
18
>>> a[0]
't'
>>> a[-1]
'e'
>>> a[6:9]
'are'
>>> a.startswith('the')
True
>>> 'he' in a
True
>>> a.count('he')
2
```
Lists

```python
>>> i = a.split()
>>> i
['these', 'are', 'the', 'same']
>>> len(i)
4
>>> i[0]
'these'
>>> i[-1]
'same'
>>> i[1:3]
['are', 'the']
>>> 'he' in i
False
>>> i.count('the')
1
```
def greet(family_name, given_name=None, salutation=None):
    print "Hello",
    if salutation:
        print salutation,
    if given_name:
        print given_name,
    print family_name

greet('Walker')
greet('Doe', 'John')
greet('Smith', salutation='Mr.')

Hello Walker
Hello John Doe
Hello Mr. Smith
Truthiness

Not Truthy:

- False
- None
- 0
- ''
- []
- ()
- {}
- set([])

Almost everything else is Truthy
Print Statement

```
print 'Hello, world!'

Hello, world!

print 'one and', 'two and',
print 'three and four'

one and two and three and four
```
Print Elsewhere

```python
import sys
print >>sys.stderr, 'Error, world!'
```
It Gets Worse

input()

```python
>>> raw_input('Your name? ')
Your name? Arthur
'Arthur'
>>> input('Eight times two? ')
Eight times two? 8*2
16
```

`raw_input()` becomes `input()` in 3.x
Print Function

```python
from __future__ import print_function  # the default in 3.x

print('Hello, world!')

print('one and', 'two and', sep=' ', end=' ')
print('three and four')

import sys
print('Error, world!', file=sys.stderr)
```
Numbers

x = 42
y = 10000000000000000000  # int and long merged in 3.x
z = 0x42ab  # hex
a = 0o775  # octal, 2.x supports C-style octal, eg. 0775
b = 0b1011  # binary

p = 2.934
q = 3+1j  # complex

import decimal
m = decimal.Decimal("3.14")  # exact decimal values

import fractions
n = fractions.Fraction(1, 3)  # exact rational values
Math

```python
>>> 1 / 2  # floor division in 2.x, normal division in 3.x
0
>>> 1 // 2
0
>>> 1.0 / 2
0.5
>>> abs(-3)
3
>>> round(0.375, 2)
0.38
>>> sum([29, 36, 12, 98])
175
```

Note: Floor division always rounds towards -Inf
String Formatting

```python
>>> count = 3
>>> product = 'coconuts'
>>> s = '%d %s remaining.' % (count, product)
>>> print s
3 coconuts remaining.
>>> t = '%04d %-10s remaining.' % (count, product)
>>> print t
0003 coconuts remaining.
```
Dictionaries

```python
>>> d = {'count': 3, 'product': 'coconuts'}
>>> len(d)
2
>>> d['product']
'coconuts'
>>> d['count']
3
>>> 'count' in d
True
>>> 3 in d
False
```

Or if you like to have arguments:

```python
>>> d = dict(count=3, product='coconuts')
```
Positional String Formatting

```python
>>> d = {'count': 3, 'product': 'coconuts'}
>>> s = '%(count)d %(product)s remaining.' % d
>>> print s
3 coconuts remaining.
>>> t = '%(count)04d %(product)-10s remaining.' % d
>>> print t
0003 coconuts       remaining.
```
"Advanced" String Formatting

```python
>>> print '{count} {product} remaining'.format(count=3, product='coconuts')
3 coconuts remaining.

>>> print '{count:<04d} {product:>10s} remaining'.format(count=3, product='coconuts')
0003 coconuts   remaining.
```

http://www.python.org/dev/peps/pep-3101/
Variable Arguments

def greet(*args, **kwargs):
    print 'args is', repr(args)
    print 'kwargs is', repr(kwargs)

greet('Walker')
greet('Doe', 'John')
greet('Smith', salutation='Mr.')

args is ('Walker',)
kwargs is {}
args is ('Doe', 'John')
kwargs is {}
args is ('Smith',)
kwargs is {'salutation': 'Mr.'}
Scoping

1. Local / function scope
2. Enclosing function(s)
3. Global / module scope (the .py file)
4. Builtins

What matters is where you assign to the variable. These are all assignments:

```python
a = 1
import cheese
def quest():
    pass
class FavouriteColour(object):
    pass
```
def fetchez_la_vache(v):
    """
    Return a vache-launching function.
    """
    def launch():
        v.moo = True
        v.crush_knight()
    return launch
Statements and Expressions

Statements:

```python
a = 1
if a:
    #...
assert a
print a  # except in 3.x
```

Expressions:

```python
a == 1
2 / 4
a is b
9 < x < 13
"the answer" if 42 else "not the answer"
lambda z: z ** 2
```
While Loop

```python
parents, babies = (1, 1)
while babies < 100:
    print 'This generation has %d babies' % babies
    parents, babies = (babies, parents + babies)
```

This generation has 1 babies
This generation has 2 babies
This generation has 3 babies
This generation has 5 babies
This generation has 8 babies
This generation has 13 babies
This generation has 21 babies
This generation has 34 babies
This generation has 55 babies
This generation has 89 babies
Tuple Assignment

\[
a, b = 1, 2 \\
a, b = b, a \\
a, (b, c) = (1, (2, (3, 4)))
\]

Tuple unpacking in 3.x:

\[
*a, b = 1, 2, 3 \\
*a, (b, *c) = (1, 2, (3, 4, 5))
\]

Same code in 2.x:

\[
z = 1, 2, 3 \\
a, b = z[\mathsf{:-1}], z[\mathsf{-1}]
\]

\[
x = (1, 2, (3, 4, 5)) \\
a, b, c = x[\mathsf{:-1}], x[\mathsf{-1}][\mathsf{0}], x[\mathsf{-1}][\mathsf{1:}]
\]
For Loop

```python
>>> people = ['john', 'pat', 'gary', 'michael']
>>> for i, name in enumerate(people):
...     print "iteration %d -> %s" % (i, name)
iteration 0 -> john
iteration 1 -> pat
iteration 2 -> gary
iteration 3 -> michael

>>> list(enumerate(people))
[(0, 'john'), (1, 'pat'), (2, 'gary'), (3, 'michael')]

>>> enumerate(people)
<enumerate object at 0x7f502799aeb0>
```
Generator Expression

```python
prices = {'apple': 0.40, 'banana': 0.50}
my_purchase = {'apple': 1, 'banana': 6}

fruit_prices = (prices[fruit] * my_purchase[fruit] for fruit in my_purchase)

grocery_bill = sum(fruit_prices)
print 'I owe the grocer $%.2f' % grocery_bill

I owe the grocer $3.40
```
Generator Functions

def fibonacci(top=None):
    a, b = 1, 1
    while top is None or a < top:
        yield a
        a, b = b, a+b

for n in fibonacci(20):
    print n

1
1
2
3
5
8
13
import sys

try:
    total = sum(int(arg) for arg in sys.argv[1:])
    print 'sum =', total
except ValueError:
    print 'Please supply integer arguments'
class LineItem(object):
    
    """
    An item on my shopping list
    """

    def __init__(self, fruit, price, qty):
        self.fruit = fruit
        self.price = price
        self.qty = qty

    def total(self):
        return self.qty * self.price

    def __str__(self):
        return "%d x %s at %.2f each" % (self.qty, self.fruit, self.price)
An Instance

```python
>>> item = LineItem('orange', 0.75, 3)
>>> item.fruit
'orange'
>>> item.total()
2.25
>>> item.qty = 5
>>> item.total()
3.75
>>> print item
5 x orange at 0.75 each
```
Many Instances

```python
>>> cart = [LineItem(*columns) for columns in [
...     ('apple', 0.40, 1),
...     ('banana', 0.50, 6),
...     ('orange', 0.75, 3),]]

>>> for item in cart:
...     print item
1 x apple at 0.40 each
6 x banana at 0.50 each
3 x orange at 0.75 each

>>> print sum(item.total() for item in cart)
5.65
```
Further Reading

- sets
- context managers
- function decorators
- class decorators
- special methods
- metaclasses
- descriptor protocol
- raising exceptions
- try, finally