Introducing Jython
Jython is everywhere

- Enterprise apps and containers like WebLogic, Websphere, N1, Bison, others
- Solutions like Grinder, PushToTest
- Cloud computing stacks like 10gen
- Supported & used by NetBeans for Python
Hello, World
(NetBeans)
def f(n):
    if n == 0:
        return 1
    else:
        return n * f(n - 1)

if __name__ == '__main__':
    from sys import argv
    if len(argv) > 1:
        print f(int(argv[1]))
    else:
        print f(5)
Popularity Reasons

- Implementation of Python
  - Widely known, great books & docs
- Found a niche in scripting
- Plays well with others, especially Java!
- Tracked the C implementation through various releases...
“The reports of my death are greatly exaggerated.”

– Mark Twain
Reality Check

Was touch & go
- Active community, mailing lists
- Patches

2.2 released last year

2.5 in beta now, with release 2009 Q1!
Java Ecosystem

- Managing PDFs with iText, PDFBox, JPod
- Excel, thru Apache POI support of OLE
- Rules engines
- Tuple spaces and extensions
- Persistent objects - Terracotta

POI is currently looking for someone to maintain Word support

However, at least 4 commercial libraries in Java to support MS Word

rather useful if you are doing your own version of google docs

at least 10 different opensource PDF rendering tools, plus many commercial ones - eight advertised under google "java pdf"

and much more functionality
Biggest Benefit

- don’t rewrite what is good out there in the Java ecosystem
Ideal Combination

In Jython:
- Django app framework
  - Models, templates, forms
- Business logic - as complex as necessary

In Java:
- Heavy-lifting, performant infrastructure
Jython in the Enterprise

- Standardized deployment options
- Django, other WSGI apps run in standard servlet containers using modjy:
  - Tomcat, GlassFish, Jetty, others
- IDE integration
Jython, quickly

- Implementation of Python
- Runs on the Java Virtual Machine
- Compiled to Java bytecode
- Readily interoperates with Java, and vice versa
Jython is just another Python

- Past: more concern with scripting Java
- Now: Python compatibility
  - 12 test cases/6147 skip/fail (99.8% pass)
  - Major apps, libraries similarly running
  - Django, SymPy; soon, Twisted, TurboGears
- Future: better Java interop, 2.7 and 3.x dev
Python

- Strongly typed (via dynamic typing)
- Everything is an object in one namespace
- Meaningful whitespace
- One way to do things
- No magic
- Clear, concise code

Objects - examples of objects include functions; and objects can be treated as functions if they implement the __call__ interface

Namespaces - So no difference between functions/methods, modules, variables, etc., although there is a difference in the scoping of a name

imports, classes, defs, variable assignment can all introduce names, potentially via renaming, among other less common approaches (manipulation of globals, etc.)

Magic - can bring pleasure - think of monkeypatching in Ruby, but it also tends to bring about surprise

Python - rather be absolutely easy to follow by being explicit and well-defined
Compilation

- Not interpreted!
- Everything is compiled to bytecode
- Additional indirection due to wrapping frames (closures/coroutines) and objects
Python Community Support

- Extremely supportive
- But pragmatic
  - Deployment as applets, WARs
  - Access to great libraries like Lucene
  - Use heavy lifting infrastructure like Hadoop (see research.freebase.org)
Small Language, Powerful Libraries

- Easy to fit into your brain
- Get over the relative indentation
- Highly dynamic
- Batteries built-in - libs w/ high-level APIs
  - XML, Unicode, database
- Staggering number of useful libraries
Consistency

- Example: iteration where it makes sense
- Everything is an object
- Single, lexically scoped namespace
  - Aliasing
  - No true globals (except builtins)
Correctness

- Integers don’t overflow
  - Instead box int -OR- BigInteger (mostly transparent)

- No loose typing - it’s **dynamic** typing
  - Not allowed to add an int with a string, without explicit conversion
Java Integration

- Import Java packages to use just like Python
- Can alias as desired into Python namespace
- Implement interface for Java to callback
- Missing feature: single method interfaces should just map to `__call__`
Simple Java code is probably Jython code if you just remove the type declarations, braces, and semicolons:

Although there is probably a more concise way of doing it
PdfReader reader = new PdfReader(filename);
PdfStamper stamper = new PdfStamper(reader, out);
AcroFields form = stamper.getAcroFields();
form.setField(key, value);
stamper.close();
iText in Jython

from com.lowagie.text.pdf import PdfStamper, PdfReader

reader = PdfReader(filename)
stamper = PdfStamper(reader, out)
form = stamper.getAcroFields()
form.setField(key, value)
stamper.close()

So our Java code looks a lot like Jython! except this fragment is runnable as-is, assuming we have defined filename, out, key, value beforehand

Could do some wrapping; the real question is what do you want to do here

The dynamic aspect of Jython makes it easy to postpone that decision until we have found evidence of a pattern of use that justifies any further wrapping
Embedding Jython

Beyond scope to cover in depth:

- Directly embed Python runtime
- JSR 223 scripting engine support
- WAR file deployment
- OSGI? Spring dynamic beans? others?
- Replacement of jythonc

Just too varied to describe
JSR 223 currently broken on the beta
Development Strategy for Dynamic Languages

- Try something out in the Python console
- Connect together in a script
- Restructure into apps and packages, with tests
- Refactor over time, using test suite
- Package for distribution
Fibonacci

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n - 2) + fib(n - 1)
from memoizers import memoize

@memoize
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n - 2) + fib(n - 1)
Aside:

Choices for memoized

- Simple memoized (recipe 325205)
- Length limited (O(1)) LRU (recipe 498110)
- Use Django’s cache mechanism
  - Local cache, memcached, Terracotta?
Fibonacci, scripted

```python
from optparse import OptionParser
import logging
try:
    from fib_memoize import fib
except ImportError:
    logging.warn("Could not import optimized fibonacci fn")
    from fib import fib

def main():
    parser = OptionParser(
        usage="usage: %prog [options] Generates Fibonacci sequence")
    parser.add_option("--start", type="int", default=0,
                       help="Starting item in sequence (inclusive)")
    parser.add_option("--end", type="int", default=100,
                       help="Ending item in sequence (exclusive)")
    options, args = parser.parse_args()
    logging.basicConfig(level=logging.WARNING)

    for n in xrange(options.start, options.end):
        print n, fib(n)

if __name__ == '__main__':
```

Factorial, tested, with contracts
Is that all?
from java.net import *
from com.vmware.vim25.mo import *

si = ServiceInstance(
    URL("https://10.17.218.174/sdk"),
    "root", "password", True)
rootFolder = si.getRootFolder()
vms = InventoryNavigator(rootFolder).
    searchManagedEntities("VirtualMachine")

for vm in vms:
    print "VM ", vm.getName()

si.getServerConnection().logout()
Objects & Classes

So far been using objects and classes, but not defining them

In fact everything is an object in Python, including functions (and methods!)

Protocol:

- `__init__`, sometimes `__new__`
- `__iter__`, `__str__`, `__call__`, many others
Various CPython (Python implementation in C) internals depend on C modules

Not necessarily equivalent to Java

Example: expat vs SAX

- For Jython: uses the org.xml.sax package

Easier to write an expat stand-in than convert all the modules dependent on it!
Wrapping:
SAX as expat
Jython Internals,

- os package uses common code with JRuby: core and posix - Java code for wrapping
- _ast pulls in Antlr generated nodes
- pkgutil needs access to custom classloader, org.python.core.BytecodeLoader
- etc

should you?
can be quite powerful, but many packages are tagged public that are not intended for such consumption. caveat emptor.
Packaging, Distribution

- Packaging via distutils OR setuptools
  - Script defines setup, requirements, tests

- Distribution of opensource
  - Upload packages to the Cheeseshop
  - Or point to it
  - easy_install
Higher-Level Coding:

- Task: sum *available* work hours for interval

- Simple to state, not so easy to program:
  - Summer time (DST) transitions
  - Holidays, weekends, leap years
Parsing

```python
>>> parse("Thu, 25 Sep 2003 10:49:41 -0300")
datetime.datetime(2003, 9, 25, 10, 49, 41,
tzinfo=tzoffset(None, -10800))
```

- relativedelta - similar to what Joda time can do with intervals
- Recurrence rules - defines a calendar of datetime events
from dateutil import rrule

def workhours(start, end,
              start_hour=7, end_hour=20,
              days_off=None, exclude_holidays=None):

    if days_off is None:
        days_off = 5, 6
    workdays = [x for x in range(7) if x not in days_off]
    cal = rrule.rruleset()
    cal.rrule(rrule.rrule(
        rrule.DAILY, dtstart=start, until=end,
        byweekday=workdays,
        byhour=(start_hour,end_hour)))

    if exclude_holidays:
        exclude_holidays(cal)
    return cal
from pairwise import pairwise
from itertools import ifilter

def total_workhours(workhours, start_hour=7):
    def deltas(pairs):
        for start, end in pairs:
            delta = end - start
            yield delta.days * 24. + \
                  delta.seconds/3600. + \
                  delta.microseconds/
                  (3600.*1e6)

    return sum(deltas(ifilter(
            lambda pair: pair[0].hour ==
            start_hour,
            pairwise(workhours))))
from itertools import tee, izip

def pairwise(iterable):
    """s -> (s0,s1), (s1,s2), (s2, s3), ...""
    a, b = tee(iterable)
    try:
        b.next()
    except StopIteration:
        pass
    return izip(a, b)
we can also add in holidays; that’s an exercise I leave to the audience

Trying it Out

```python
>>> from workhours import workhours, total_workhours
>>> from datetime import date
>>> from dateutil.relativedelta import relativedelta
>>> end = date.today()
>>> start = end - relativedelta(weeks=3, months=1)
>>> print total_workhours(workhours(start, end))  # 468.0
```
Django
Models

- Domain Specific Language to describe database

- But in Python
  - “Dutch pseudocode”, not English
  - Model can incorporate Python as needed

- Synchronized to database (syncdb) and validated at startup
good example of Django - completely written in Django itself, although some special support in the model code to hook in validators

this is changing with the newadmin
Forms, Templates, etc.

- **URLs to dispatch to views**
- **Views**
  - Use templates, forms
  - Forms simply can build on models to validate, use user input in some
  - Or interact with them in arbitrary ways, as might be needed in a workflow system
Neo4j

- Graph database for Java
  - High performance, fully transactional
  - Data is modeled as Nodes, Relationships in between Nodes, properties on both Nodes and Relationships

- Embedded

- Small core with modular extensions

Supports everything that one can demand from a transactional system: ACID, recovery, two phase commit...
Neo4j Benefits

- Whiteboard friendly - implement exactly as drawn - no O/R mismatch
- Rapid schema evolution
- Performance and performance scalability
  - Traversals instead of Joins
Neo4j Modeling

- Implement objects by wrapping a Neo4j primitive (Node/Relationship)
- Delegate all state to that primitive
- Automatic wiring for Java, Python, JRuby...
  - Declarative programming
  - Qi4j
Django Demo
Deployment Jujitsu
Deployment Jujitsu

- If the problem is J2EE...
Deployment Jujitsu

- If the problem is J2EE...
- we can still run Django on that environment
Deployment Jujitsu

- If the problem is J2EE...
- we can still run Django on that environment
- Quite beyond the world of Pet Shop
Deployment Options

- Java app servers - show some horizontal scaling; also management - sys admins know how to do this!

- Hardened - it runs on the JVM
vs Google App Engine

- It’s Django 1.0

- We have Unicode, internationalization, ORM, admin, shell, everything!

- And we can use Java analogues:
  - BigTable <=> HBase, Hypertable
  - MapReduce <=> Hadoop
Low Level Concurrency in Jython threading

- Jython threads map 1-to-1 to Java threads
- Locks, condition variables, semaphores: same thing
- Queues can’t map because join (wait until all items processed) requires an additional condition variable
Concurrency

- Jython dict is a Java ConcurrentHashMap
- Synchronization is programmer’s responsibility
- Can readily use higher-level APIs

No Global Interpreter Lock!
Concurrency

- Jython dict is a Java ConcurrentHashMap
- Synchronization is programmer’s responsibility
- Can readily use higher-level APIs

- **No** Global Interpreter Lock!
Recursive Generators

- Are they practical?
  - Sometimes. But they are certainly fun
  - Best to get the algorithm right first!

...and as we will see, there is a direct mapping from recursive generators to the fork-join framework in Java 7.
Recursive Permutations

# based on cookbook recipe 190465

def permute(items, n=None):
    if n is None: n = len(items)
    if n==0: yield [] # base case yields
    else:
        for i in xrange(len(items)):
            for subp in permute(items[:i]+items[i+1:],n-1):
                yield [items[i]]+subp # and iterate to yield
Graph Traversal

- Problem: Traverse a graph of edges, nodes
- Workhorse algorithm
- Non-trivial parallelization
- Cool recursive generator solution!

DFS is a workhorse for topological sort, SCC or to determine the reporting structure of an organization or for a directory tree (Active Directory), who gets what in terms of profiles, etc.

We're going to look at the construction of a spanning tree.
Example Graph
def spanning(G, v, parent, marked):
    marked.add(v)
    yield v

    for w in G.get(v, ()):  
        if w not in marked:
            parent[w] = v
            for u in spanning(G, w, parent, marked):
                yield u
Concurrency and Jython

- Strong support for threading
- Java memory model
- ConcurrentHashMap for dict, __dict__
- Can readily access java.util.concurrent functionality - Futures, Execution Service
Parallelizing?

- Can we search the graph in parallel?
- Yes, but first, let’s create a useful abstraction
Fork

- Given a generator, runs it **effectively** in a separate thread
- Recursively
- But can be done efficiently too!
Serial version

```python
def spanning(G, v, parent, marked=None):
    if marked is None: marked = {}

    if v not in marked:
        marked[v] = True
        yield v

    workers = []

    for w in G.get(v, ()):  # Fork
        if w not in marked:
            marked[w] = True
            yield w
            parent[w] = v
            workers.append(spanning(G, w, parent, marked))

    for worker in workers:
        for u in worker:
            yield u

    # Implicit Join
```

Fork

Implicit Join
Fork using Java’s FJ FW

- Java Fork-Join Framework
  - Thread pool
  - “Work stealing”

- Designed to work with recursive algorithms!
class fork(object):
    def __init__(self, f):
        self.task = ForkContainer(f)
        self.task.fork()
        self.joined = None

    def __iter__(self):
        if self.joined is None:
            self.task.join()
            self.joined = iter(self.task.result)
        return self.joined

    def next(self):
        return self.__iter__().next()

    def __repr__(self):
        if self.joined is None:
            return "fork(forked)"
        else:
            return "fork(waiting)"
class ForkContainer(fj.RecursiveAction):
    def __init__(self, f):
        self.f = f

    def compute(self):
        self.result = list(self.f)

    def __iter__(self):
        return iter(self.result)
def forkjoin_pool(f, num_threads=None):
    if num_threads is None:
        num_threads = \
        java.lang.Runtime.getRuntime().\
        availableProcessors() + 1
    mfj = ForkContainer(f)
    fj_pool = fj.ForkJoinPool(num_threads)
    fj_pool.invoke(mfj)
    return mfj
def spanning(G, v, parent, plies=2, ply=0, marked=None):
    if marked is None: marked = {}  

    if not marked.setifabsent(v, True):
        yield v
    workers = []

    for w in G.get(v, ()): 
       if not marked.setifabsent(w, True):
            yield w
            parent[w] = v
            if (ply % plies == 0):
                workers.append(fork(
                    spanning(G, w, parent, plies, ply+1, marked)))
            else:
                workers.append(spanning(G, w, parent, plies, ply+1, marked))

    for worker in workers:
        for u in worker:
            yield u
Thanks!