A harmony in 3 parts

- What is Python?
  - Gentle introduction to the language

- What is Google App Engine?
  - Introduction to cloud computing
  - App Engine product/feature overview

- Google App Engine: Live Experience
  - Tutorial/workshop
  - Live & hands-on with the platform
What is Python?
An Introduction

"Perl is worse than Python because people wanted it worse."
-- Larry Wall (14 Oct 1998 15:46:10 -0700, Perl Users mailing list)

"Life is better without braces."
-- Bruce Eckel, author of Thinking in C++, Thinking in Java

"Python is an excellent language[, and makes] sensible compromises."
-- Peter Norvig (Google), author of Artificial Intelligence

Python is Fun!
(I've used it at lots of places!)
I'm here to give you an idea of what it is!
(I've written a lot about it!)
I've taught it at lots of places!
(companies, schools, etc.)
Instructor Background

- Python: 14+ consecutive years as software developer
  - Original Yahoo!Mail team member
  - Experience: networking, middleware+server-side engineer
  - Now Developer Advocate at Google

- Book Author (or Co-author):
  - *Core Python Programming* ([2009,] 2007, 2001)
  - *Python Web Development with Django* (2009)

- Teacher since 1983
  - BASIC programming to 1st, 4th-6th graders
  - Undergraduate math tutoring at UC Berkeley
  - Algebra, Geometry, SAT Prep to 6th-12th graders at CCC
  - C, Unix, Python for UCSB & UCSC Extension, FHDA
  - CSTA member

(c) 1998-2011 CyberWeb Consulting.
All rights reserved.
About You

• SW/HW Engineer/Lead
• Sys Admin/IS/IT/Ops
• Web/Flash Developer
• QA/Testing/Automation
• Scientist/Mathematician
• Toolsmith, Hobbyist
• Release Engineer/SCM
• Artist/Designer/UE/UI/UX
• Student/Teacher

• Hopefully familiar with one other high-level language:
  • Java
  • C, C++, C#
  • PHP, JavaScript
  • (Visual) Basic
  • Perl, Tcl, Lisp
  • Ruby, etc.

• Django, TurboGears/Pylons, Pyramid, Plone, Trac, Mailman, App Engine

Why are you here? You…

• Have heard good word-of-mouth
• Came via Django, App Engine, Plone, etc.
• Discovered Google, Yahoo!, et al. use it
• Already know but want formal training
• Were forced by your boss

• Safari Books Online: Top 5, Apr 2009
  1. iPhone
  2. Java
  3. Python
  4. C#
  5. PHP
For New Programmers

- Programming should be easy enough to teach like reading & writing (CP4E)
  
- What you need to be good
  - Some math background
    - Algebra, physics (problem solving)
  - Buddy who already is a developer
  - Even better if they know Python
  - Tinkerer, curious, itch-scratcher
About this Talk

● **Goal:** Introduce as much Python as one can in a single hour

● **Seminar Topics**
  ● Language Introduction
  ● Python Object Types
  ● Loops and Conditionals
  ● Files, Functions, Modules
  ● Object-Oriented Programming
  ● Errors and Exception Handling
  ● Miscellaneous

Background

● Guido van Rossum: began late '89, released early '91
  ● Inherits from ABC, Modula, C/C++, LISP, ALGOL, Perl, Java, etc.
  ● Named after Monty Python, not the snake 😎

● Cross-platform (Mac, PC, *ix; only need a C compiler)
  ● Alternate implementations: Jython, IronPython, PyPy, Stackless, etc.

● Philosophy, Concepts and Syntax
  ● Robust: Enough "batteries included" to get job done
  ● Simple: Clean, easy-to-read, easier than VB?!?
  ● Modular: Plug-n-play, use only what you need
  ● Extensible: Can extend language to meet your needs
  ● Intuitive: "Python fits your brain."
  ● OOP: Object-oriented when you need it (not req'd)
  ● "Pythonic": "There's only one way to do it...."
Why (not) Python?

- **Advantages**
  - Simple Syntax
  - Rapid Development
  - High-level data structures
  - Object-Oriented
  - Exception Handling
  - Functional Programming
  - Memory Management
  - Extensible (C/C++/Java)
  - Many libraries: NW, DB, GUI, MT, XML, RE, OS/FS, Math, Web; plus 3rd-party
  - Grassroots community

- **Disadvantages**
  - Generally slower than compiled languages
  - Idiosyncratic idioms
  - Obscure syntax? Naaah!!
  - No marketing force… mostly word-of-mouth
  - No world domination… yet! (Happening slowly.)

http://tiobe.com/index.php/content/paperinfo/tpci

Built for Non-Programmers

- **Educational language syntax**

- **Works well for school-aged children**
  - Syntax, memory management, data structures, object-oriented programming… such issues get in way of learning concepts
  - C++ and Java deter interest and students
  - Python: retention, morale, understanding
  - Probably the "best" 1st language

- **Imagine what it means for seasoned programming professionals?**
Some Newbie Resources

● Published Books
  ● *Hello World! Computer Programming for Kids & Other Beginners* (Sande, Sande: 2009)
  ● *Python Programming for the Absolute Beginner* (Dawson: 2010)
  ● *Learning with Python: How to Think Like a Computer Scientist* (Downey, Elkner, Meyers: 2002)
  ● *Head First Programming* (Griffiths, Barry: 2009)
  ● *Learn Python the Hard Way* (Shaw, 2010)

● Online Books, Tutorials, Environments, etc.
  ● *How to Think Like a Computer Scientist* (Downey, Elkner, Meyers)
  ● *Learning to Program* (Gauld)
  ● LiveWires Python course
  ● *A Byte of Python* (Swaroop)
  ● *Instant Hacking: Learning to Program with Python* (Hetland)
  ● *Snake Wrangling for Kids* (Briggs)
  ● *Computer Programming is Fun!* (Handy)
  ● *Karel the Robot* clones, Guido van Robot, RUR-PLE

Interactive Interpreter

● Running Python's interpreter from the default IDE: IDLE

```
Python Shell
IDLE 1.1.1
>>> >>> >>>
>>> print 'Hello World!'
Hello World!
```

● Starting from the cmd-line; can also run scripts (.py extension)

```
$ python   # or C:\> python
Python 2.6.2 (r262:71600, May 12 2009, 23:46:27)
[GCC 4.0.1 {Apple Inc. build 5465}] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> >>> >>>
>>> print 'Hello World!'
Hello World!
```

(c) 1998-2011 CyberWeb Consulting.
All rights reserved.
What You Just Saw

- >>> ← this is the Python prompt
  - Enter any Python command after it
- Use `print` to display output to users
  - Use `print()` in Python 3
- To exit interactive interpreter
  - Ctrl-D from IDLE or *ix command-line
  - Ctrl-Z from a DOS/command shell
- Integrated Development Environment: developer tools
- Interactive interpreter: testing, debugging, hacking, experimenting regardless of IDE use or otherwise

Getting Python on your Computer

- MacOS, Unix: already installed
- Win, source, and other downloads
  - http://python.org or http://corepython.com
- Other IDEs
  - SPE (pictured)
  - PythonWin
  - Eclipse & PyDev
  - ActiveState
  - Wingware
  - IPython
Common Beginner Gotchas

- Code delimited by indentation not braces \{ \}
  ```python
  if x > 10:
      return "'x' is greater than 10"
  else:
      return "'x' is less than 10"
  ```
- Similarly, no extraneous characters ( ; , $, …)
- No switch-case, private class members, ++/--,
  static type checking, anonymous blocks, etc.
- True division: $1/2 \equiv 0.5$ (not 0)
- Freaky-looking floats: $1.1 \rightarrow 1.10000000001$
- Single element tuple needs comma: (None,)

Objects

- Allocated on assignment: Dynamic/Duck Typing
- Additional refs (aka aliases) similar to pointers
- Call by Reference, Call by Value? Neither. Both.
- Memory Management: Reference Counting

Standard Types
- Numbers (3-8)
- Strings (2-3)
- Lists
- Tuples
- Dictionaries
- Sets (2)

Other Types
- None
- Files
- Functions/Methods
- Modules
- Type/Classes
- miscellaneous
Variables and Expressions

```python
>>> 4 + 6 * 5          # math ops like other langs
34
>>> a = 4 + 6 ** 2    # no declaration needed
>>> a
40
>>> a = 'Python'      # auto garbage collection
>>> b = 'is cool'
>>> a + b             # ops can be overloaded
'Pythonis cool'
>>> a = a + ' ' + b   # reassignment no problem
>>> a
'Python is cool'
>>>                        # useful interactive tool
```

Numbers

- **Integers (no size limit except for VM)**
  - -680, 0o237, 0xDEADBEEF, 64-0x41, 0b110

- **Floating point real numbers (IEEE-754 C double)**
  - -97.65, -3.14159, -6e2, 0.1

- **Complex numbers**
  - **Composed of real and imaginary parts (floats)**
  - 11.65-5.55J, 4J, -3e2+8.73J, 0.1-2.3e4J

- **Also long, bool, Decimal, Fraction, Rational, etc.**
- **Other modules:** math, cmath, random, operator
Standard Operators

+    -    *    /    //    %    **
<<    >>    &    |    ^    ~
==    >=    <=    <    >    !=
is    is not    and    or    not

• Grouping expressions in ( ) okay as usual
• ** means exponentiation, % means modulus/remainder
• / means true division in 3.x and classic division in 2.x
• Use // for standard integer floor division
• Assignment using single equals (=)
• Augmented assignment +=, -=, *=, etc. (no ++ though)

Strings

' '  " "   ''' '''

• Strings are sequences of characters (single/double quotes)
• Format operator (%) for printf()-like functionality
• Triple quotes allow special characters like newlines

>>> s = 'Python'
>>> s * 2
'PythonPython'
>>> s = s * 2
>>> s
'PythonPython'
>>> s[4:6]
'on'
>>> s[-1]
'n'
>>> s[-4:-1]
'tho'
>>> s[:6]
'Python'

>>> '%s is number %d' % (s[:6], 1)
's is number 1

>>> hi = '''hi
there'''
>>> hi = 'hi
there'
>>> print hi
hi
there

(c) 1998-2011 CyberWeb Consulting.
All rights reserved.
Lists [ ] and Tuples ( )

- Lists are ordered sequences of arbitrary objects
- Mutable (values can be updated)
- Can have lists of lists, i.e., multidimensional indexing
- Tuples similar but immutable (no value updates allowed)
- "List comprehensions" allows for quick logical building of lists

Common list methods:
- `list.sort()` # "sort" list contents in-place
- `list.reverse()` # reverse a list in-place
- `list.append()` # append item to list
- `list.remove/pop()` # remove item(s) from list
- `list.extend()` # extend a list with another one
- `list.count()` # return number of item occurrences
- `list.index()` # lowest index where item is found
- `list.insert()` # insert item(s) in list

List Operations

```python
>>> m = ['Core', 'Programming', 9, 2006]
>>> m.append('Prentice Hall')
>>> m.insert(1, 'Pytho')
>>> m
['Core', 'Pytho', 'Programming', 9, 2006, 'Prentice Hall']
>>> m[1] = 'Python'
>>> m.pop(3)
9
>>> m
['Core', 'Python', 'Programming', 2006, 'Prentice Hall']
>>> m.sort()
>>> m
[2006, 'Core', 'Prentice Hall', 'Programming', 'Python']
```

```python
>>> [i*3 for i in range(20) if i % 2 == 0]
[0, 6, 12, 18, 24, 30, 36, 42, 48, 54]
>>> f = open('myFile', 'r')
>>> data = [line.strip() for line in f]
>>> f.close()
```
Dictionaries \{ \}

- Dictionaries are Python's only mapping type
  - Mutable, resizable hash tables
  - Mappings of keys to values
  - Keys are scalar (usually strings or numbers)
  - Values are arbitrary Python objects
  - No key collisions allowed
  - Similar to Java HashMaps and Perl hashes/associative arrays

Common dictionary methods:
- `d.keys()`  
  - iterable: keys of `d`
- `d.values()`  
  - iterable: values of `d`
- `d.items()`  
  - list of key-value pairs
- `d.get()`  
  - return key's value (or default)
- `d.pop()`  
  - remove item from `d` and return
- `d.update()`  
  - merge contents from another dict

Dictionary Operations

```python
>>> d = {'title': 'Core Python Programming', 'year': 2007}
>>> d
d
{'year': 2007, 'title': 'Core Python Programming'}
>>> 'year' in d
True
>>> 'pub' in d
False
>>> d.get('pub', 'N/A')  
# KeyError if d['pub']
'N/A'
>>> d['pub'] = 'Prentice Hall'
>>> d.get('pub', 'N/A')  
# no KeyError for d['pub'] now
'Prentice Hall'
>>> for eachKey in d:
print eachKey, ':', d[eachKey]
year : 2007
pub : Prentice Hall
title : Core Python Programming
```
if-elif-else Statements

● Conditional statements are what you expect

```python
# prompt, get, and check user input
data = raw_input("Enter 'y' or 'n': ").lower()

if data[0] == 'y':
    print "You typed 'y'."  # 'y' key
elif data[0] == 'n':
    print "You typed 'n'."  # 'n' key
else:
    print 'invalid key!'  # other key
```

● Ternary Operator (aka Conditional Expressions: C ? T : F)

```python
smaller = x if x < y else y  # T if C else F
```

Loops

● Python has while and for loops – while loops are "normal"
● for loops more like shell foreach
● Iterate over a sequence rather than as a conditional
● range() was created to "simulate" a traditional for

```python
aList = [123, 'xyz', 45.67]
```
Files

- Open a file and get back a file object
  
  \[
  f = \text{open}(\text{file\_name}, \text{access\_mode})
  \]

- Most commonly-used file methods
  
  - `f.close()` Close file
  - `f.read()` Read bytes from file
  - `f.readlines()` Read all lines into an iterable
  - `f.write()` Write a string to file

- Example of displaying a text file to the screen
  
  ```python
  fp = open('data.txt', 'r')  # open file, get file object
  for eachLine in fp:  # display one line at a time
    print eachLine,
  fp.close()  # close file
  ```

Functions

- Function declarations created with `def` statement
- Support for default and variable-length arguments
- Support for variety of invocation styles

```python
def foo(x):
  # create foo()
  print 'Hello %s!' % x

>>> foo('Guido')  # call foo()
Hello Guido!
```

- Functional Programming elements:
  - List comprehensions and generator expressions
  - Currying and partial function application
  - Statically-nested: Inner functions and closures
  - Anonymous Functions (lambda)
Importing Modules & Attributes

- Importing a module using import statement
  ```python
  import module_name
  ```
  ```python
  import string
  num = string.atoi('123')
  ```

- Importing module attributes using from-import statement
  ```python
  from module_name import module_element
  ```
  ```python
  from string import atoi
  num = atoi('123')
  ```

- Packages: allow for organizing modules using the file system

## Standard Library Sampler (“B.I.”)##

<table>
<thead>
<tr>
<th>Module Name(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys</td>
<td>System data, processing, and functionality</td>
</tr>
<tr>
<td>os and os.path</td>
<td>Operating and file system interface</td>
</tr>
<tr>
<td>re, csv</td>
<td>Regex and CSV text processing</td>
</tr>
<tr>
<td>time, datetime, calendar</td>
<td>Date and time constants and functions</td>
</tr>
<tr>
<td>socket, SocketServer</td>
<td>Socket interface &amp; server classes (TCP, UDP)</td>
</tr>
<tr>
<td>sqlite3</td>
<td>API for SQLite databases</td>
</tr>
<tr>
<td>subprocess</td>
<td>External process management</td>
</tr>
<tr>
<td>email</td>
<td>Email/MIME construction and parsing package</td>
</tr>
<tr>
<td>Tkinter</td>
<td>Python/Tk GUI toolkit interface</td>
</tr>
<tr>
<td>threading, multiprocessing</td>
<td>High-level threads/multiprocess APIs</td>
</tr>
<tr>
<td>pickle, cPickle, shelve</td>
<td>Serialize Python objects</td>
</tr>
<tr>
<td>{c,}math, random, fractions,...</td>
<td>Various math/numeric processing</td>
</tr>
<tr>
<td>gzip, bz2, zipfile, tarfile</td>
<td>Data compression and archive files</td>
</tr>
<tr>
<td>{ftp, pop, url, http, smtp,*}lib</td>
<td>Various Internet client libraries</td>
</tr>
<tr>
<td>xml.sax, xml.dom, xml.etree</td>
<td>SAX parsing, DOM tree mgmt, ElementTree API</td>
</tr>
</tbody>
</table>

(c) 1998-2011 CyberWeb Consulting. All rights reserved.
Object-Oriented Programming

- "Constructor"/Initializer is `__init__()`; "self" is "this"
- Class instantiation via function interface (rather than "new")
- Instance attrs, multiple inheritance; no overloading nor private

```python
class MyClass(object):
    ...  def __init__(self, data=2):
        ...      self.info = data
    ...  def times(self, x):
        ...      return "%d * %d is %d" % (self.info, x, self.info * x)

>>> inst = MyClass(21)
>>> inst.info
21
>>> print inst.times(3)
21 * 3 is 63
```

Exceptions and try-except

- Exception handling via try-except statement

```python
try:
    # statements to monitor
except (ErrorType1, ErrorType2,....) as e:
    # code to exec if exception occurs

try:
    fp = open('data.txt', 'r')
except IOError as e:
    print 'file open error:', e
return False
```

- Throw exceptions with `raise`; there is also a `finally`
Programmer Tools

- Debugger
  - pdb

- Profilers
  - profile
  - hotshot
  - cProfile

- Tracer/Tracker
  - trace

- Timer
  - timeit

- Logger
  - logging

- Testing
  - unittest
  - doctest
  - (external) nose
  - (external) py.test

Python 2 vs. Python 3

- The What and the Why
  - Fix early design flaws
  - Some new features, many small improvements
  - Plan: develop (remainder of) 2.x and 3.x together
  - Provide transition tools (2to3, 2.6+)

- Key Updates (no major syntax changes)
  - print, exec changed to functions
  - True division: 1/2 == 0.5
  - Performance enhancements (more iterators)
  - Type consolidation (integers, classes, obj comps)
  - Strings: Unicode default; bytes/bytearray types

- Python 3 article on InformIT

(c) 1998-2011 CyberWeb Consulting.
All rights reserved.
Where do you go from here?

• The tip of the iceberg…
  • System Administration scripting tools and automation
  • Networked/Internet Client-Server systems
  • Graphical User Interface (GUI) design and development
  • Web frameworks and rich Internet applications (RIA)
  • Scientific/numeric/mathematics-centric development
  • Database programming, Object-Relational Mappers (ORMs)
  • XML, JSON, CSV and other forms of text processing
  • Java with Jython, .NET/Mono with IronPython, Win32 COM
  • Graphics/imaging/multimedia/visual art processing
  • Multithreaded and higher-performance applications
  • Embedded system design (hardware as well as Python)
  • QA/Testing automated test tools, suites, and frameworks
  • Rapid application prototyping (all application areas)

Additional Resources

• Published Books
  • Quick Python Book (Ceder, 2010)
  • Core Python Programming (Chun, 2007 [2009 special])
  • Python Fundamentals LiveLessons DVD (Chun, 2009)
  • Beginning Python (Hetland, 2008)
  • Dive into Python (Pilgrim, 2004)
  • Python Essential Reference (Beazley, 2009)
  • Python in a Nutshell (Martelli, 2006)
  • Python Cookbook (Martelli, Ravenscroft, Ascher, 2005)

• Online Books, Tutorials, Environments, etc.
  • Python Quick Reference Guide http://rgruet.free.fr#QuickRef
  • worldwide Python Conferences http://www.pycon.org
  • Core Python Programming http://corepython.com
  • comp.lang.python newsgroup http://groups.google.com
  • Python FAQ http://python.org/doc/faq
  • PyPI/Cheeseshop repository http://python.org/pypi
The Zen of Python (or import this by Tim Peters)
1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one — and preferably only one — obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than right now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea — let's do more of those!

(c) 1998-2011 CyberWeb Consulting.
All rights reserved.

Q&A, Contact Info, Events

• wescpy@gmail.com (Python)
• wesc-api@google.com (Google App Engine, etc.)
• @wescpy
• http://cyberwebconsulting.com
• http://corepython.com

● Google I/O: 2011 May 9-11, San Francisco
  ● Python@Google
  ● http://google.com/io

  ● Writing Books using Python
  & Open Source Software
  ● Python 3: The Next Generation
  ● Using Python in the Medical Industry
  ● Python 103: MMMM (Methods, Memory Model, Mutability)
  ● What is Google App Engine?
  ● http://europython.eu

● ACM CSTA CS&IT: 2011 Jul 11-13, NYC
  ● Python, Cloud Computing, Google App Engine workshop
  ● http://www.csitsymposium.org

● OSCON: 2011 Jul 25-29, Portland
  ● Google App Engine workshop
  ● Python 3: The Next Generation
  ● http://oscon.com

● 3-day public Python training course:
  2011 Oct 18-20, San Francisco
  ● Intro+Intermediate Python
  ● http://cyberwebconsulting.com

(c) 1998-2011 CyberWeb Consulting.
All rights reserved.
What is... Google App Engine?

Wesley Chun
Developer Advocate, Google
http://code.google.com/appengine
@wescpy / @Google / @App_Engine

Cloud Computing
What is it (besides being buzzword-compliant)?
What is Cloud Computing?

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

National Institute of Standards and Technology (NIST) Forum May 2010
http://csrc.nist.gov/groups/SNS/cloud-computing/forum-workshop_may2010.html

Cloud Computing concept... it's been around longer than you think

"In 1984, John Gage and Bill Joy of Sun Microsystems were credited with saying, "The network is the computer," one of the most memorable slogans of the information age. This concept, based on the tenet of making computing resources available to all users irrespective of location, as long as they are connected to the network, forms the basis of the Internet as we know it, and foretold the advent of cloud computing."

Garry Metcalf, Analysys Mason, Sep 2009
http://www.analysysmason.com/about-us/newsletter/Previous-news-articles/Now-the-network-really-is-the-computer
The Benefits of Cloud Computing

- **Reduced Cost**
  - Incremental payment based on usage
- **Elastic Storage**
  - Data storage can easily grow with business
- **High Automation**
  - Platform updates/patches updated automatically
- **Flexibility**
  - Resources can be changed on-demand
- **More Mobility**
  - Universal Accessibility
- **Shift towards Innovation**
  - Less maintenance, can focus on product innovation

*Bottom line: More focus on business!*

---

The shift towards Cloud Computing

*"By 2012, 20% of Global 2000 enterprises will be using public cloud services, up from under 5% in 2009."*

Yefim Natis - Gartner Cloud Computing Analyst, AADI Dec 2009
The Cloud Pyramid

Cloud Computing Defined

SaaS

PaaS

IaaS

Source: Gartner AADI Summit Dec 2009

What is App Engine?

- App Engine is a platform
- You build & test your app
- Then upload your app to Google
- App Engine runs everything
- No need to worry about machines, network, storage, scalability, etc.
Challenges building web apps

What keeps you up at night?
DIY Hosting means hidden costs

- Idle capacity
- Software patches & upgrades
- License fees
- Lots of maintenance
- Traffic & utilization forecasting
- Upgrades
Easy to start
Easy to scale
Easy to maintain

We do the dirty work...

Google App Engine

“We wear pagers so you don’t have to”
The Components of Google App Engine

- Components
  - Language Runtimes
  - SDK
  - Web-based Admin Console
  - Scalable Infrastructure
Components

Scalable Infrastructure

Linux
GFS
Bigtable
Hardware

Python, Java
& Go Runtimes

Batteries
Included

Alternative API/Runtimes also available (more later)
Getting the job done

What are programming languages?
Various dialects that let humans give instructions to computers

- Some are easier than others
- Some are special-purpose
- Adults: Java, C/C++, Python. Ruby, PHP
- Kids: Scratch, Alice, Python, BASIC

The king of ease-of-use

- Extremely rapid development
- Very low barrier of entry
- Simple yet robust syntax
- Rich library of packages/modules
- App Engine’s first language API
The king of enterprise development

- Ubiquitous in Enterprise computing
- Adheres to Java servlet standard
- Rich library of packages/modules
- Eclipse Plug-in support
- Alternative language support

The new kid on the block

- Best of both worlds
  - The complexity & power of a statically-type language
  - The benefits of dynamically-typed languages
  - Built-in concurrency
  - Compiled but w/garbage collection & run-time reflection
- Flexible alternative to Python & Java
### Java compliant Servlet Standard

#### Ensuring Portability

<table>
<thead>
<tr>
<th>Java standards</th>
<th>JSR-164</th>
<th>JSR-220, JSR-243</th>
<th>Java SE</th>
<th>JSR-919</th>
<th>JSR-107</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Servlet</td>
<td>JDO / JPA</td>
<td>java.net.URL</td>
<td>javax.mail</td>
<td>javax.cache</td>
<td></td>
</tr>
</tbody>
</table>

- **Web App Container**
- **Datastore API**
- **URL Fetch**
- **Mail API**
- **Memcache**

#### Low level App Engine APIs

### Java Plug-n-Play

#### Google Plugin for Eclipse
Extended Language support through JVM

- Java
- Scala
- JRuby (Ruby)
- Groovy
- Quercus (PHP)
- Rhino (JavaScript)
- Jython (Python)

Components
App Engine Dashboard

App Engine Health History
Components

SDK

- Run Locally
- Easy Deploy
- Manage Versions
- APIs

Case Studies
Serving our Users
App Engine growth

2008 2009 2010 2011

App Engine Launch
Python
Datastore
Memcache
logs export

Batch write/read
Java
DB Import
cron

Task Queues
XMPP
Incoming email

Blobstore
AppStats
cursors
Mapper

Multi-tenant
Instance Console
Always On
hi-perf imag
10 min tasks

Hi-Replcation
Datastore
Channel API
Files API
Remote API
Prosp Search

How many ______ ? By the numbers

100,000+
Active developers
per month

2008 2009 2010 2011
200,000+
Active apps per week

1,500,000,000+
Page views per day
Developers who know that App Engine…

…scales for social web and mobile apps

Social networking at scale
…and grows with you and your app

Social networking at scale

>62M Users

• 3.6MM DAUs on FB
• 1.9MM DAUs on MS
• Add Orkut, Bebo, Hi5, Friendster, Hyves, Ning...

Gigya…

gigya Socialize
… scales for event-based websites

Official Royal Wedding Website hosted on App Engine

On Wedding day, served:
- Up to 2000 requests per second
- 15 million pageviews
- 5.6 million visitors

http://goo.gl/F1SGc
…scales for enterprise-targeted cloud apps

Not all apps user-facing or web-based!!

- Need backend server processing? Want to build your own?
- Go cloud with App Engine!
- No UI needed for app to talk to App Engine, just need HTTP or XMPP
- Great place for user info e.g., high scores, contacts, badges, etc.
- Better UI: move user data off phone & make universally available
…scales for web and mobile gaming

Features and Futures
Services and APIs now... and coming soon
Rich set of App Engine services/APIs

- Memcache
- Datastore
- URL Fetch
- Mail
- XMPP
- Task Queue
- Images
- Blobstore
- User Service

In Review: 4+ years of New Features & Continuous Innovation

<table>
<thead>
<tr>
<th>Year</th>
<th>Feature Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td><strong>App Engine with Python runtime launch</strong>, Django 0.96, Memcache API, Images API, Logs export, Batch write/delete, HTTPS support, System status dashboard page, Detailed Quota page in Admin Console</td>
</tr>
<tr>
<td>2009</td>
<td><strong>Java runtime launch</strong>, Billing, Remote API &amp; Shell, Bulkloader, Cron jobs, Key-only queries, Task Queues, Django 1.0 &amp; 1.1, XMPP API, Receiving Email, Blobstore API</td>
</tr>
<tr>
<td>2010</td>
<td>Datastore cursors, Async URLfetch, Appstats profiling, DoS filter, Eventual consistency, OpenID, OAuth, Namespaces, high-performance image service, Instances console, Datastore Admin with bulk entity delete &amp; copy, Channel API(?)</td>
</tr>
<tr>
<td>2011</td>
<td><strong>Go runtime launch</strong>, High Replication datastore, Bulk entity copy b/w apps, 10-minute URLfetch, Backends (long-term servers), Pull Queues, Geolocation headers, WebP images, ProtoRPC, SDK HRD, Channel presence, Django 1.2(?)</td>
</tr>
</tbody>
</table>
Fees & Usage

- Three classes of service
  - Free
  - Paid ($9 app/month)
  - Premium ($500/month, unlimited apps)

- Quota
  - Current preview/beta quotas (now till late 2011)
    - [http://code.google.com/appengine/docs/quotas.html](http://code.google.com/appengine/docs/quotas.html)
  - Official product pricing/quotas (late 2011)
    - [http://www.google.com/enterprise/appengine/appengine_pricing.html](http://www.google.com/enterprise/appengine/appengine_pricing.html)

- 99.95% SLA for paid service classes
- Operational support for Premium service class

---

App Engine Roadmap

- App Engine out-of-preview/official support
- SSL access on non-appspot.com domains
- Improved datastore import/export backup/restore
- MapReduce
- Full-text Search over Datastore
- Python 2.7
- Improved monitoring and alerting
- Raise request/response size limits for some APIs
- [See code.google.com/appengine/docs/roadmap.html](http://code.google.com/appengine/docs/roadmap.html)
What is “vendor lock-in?”

- What is it?
  - Systems which inherently make it very difficult or impossible to migrate (data and/or logic) to other similar/competitive systems

- Applies to App Engine?
  - Yes & no.
  - Don’t get something for nothing.
  - Goal: take advantage of Google’s infrastructure
  - Price: need to write against Google APIs. Make sense?

Fighting “lock-in” with App Engine

- “Must use” Google APIs
  - Not necessarily

- Web frameworks
  - App Engine comes with webapp
  - But can use Django, web2py, Tipfy, Bottle

- Datastore API
  - Can use Django ORM (with django-nonrel)

- Datastore bulkloader
  - Easily up- or download your data

- Open source backend systems
  - AppScale (UCSB)
  - TyphoonAE
Google Apps Integration
App Engine apps in your Apps domain

Google Apps + your apps

Our Google Apps

Your custom applications

Google's scalable serving architecture
Google Apps integration

Add your app to your Apps domain

Google Apps

Add Domain

You have requested that the "exco8190" service be added to your domain

Please accept the Google App Engine terms and conditions to continue.

Google App Engine terms and conditions

You can manage your app via the Google Admin console at https://admin.google.com. For more information on using your app, please see the documentation at the following URLs:

* [App Engine developer’s guide](https://developers.google.com/appengine)
* [Documentation for your app](https://yourapp.com)

Additional links:

* [Help Center](https://support.google.com)
* [Feedback](https://www.google.com/feedback)

Privacy policy

By accepting "Accept", you agree to the terms and conditions for this service at any time on the service settings page.
Your app now part of your Apps suite

Getting Started

http://code.google.com/appengine (download, docs, etc.)
http://code.google.com/p/googleappengine (issues, wiki, etc.)
http://code.google.com/eclipse (Eclipse plug-in)
http://appengine.google.com (login, app management)
http://googleappengine.blogspot.com (blog)
App Engine online resources

http://code.google.com/appengine – downloads, docs, forums, FAQ, etc.

http://appengine.google.com – managing your applications here

http://googleappengine.blogspot.com – our blog

Also check http://stackoverflow.com

Hello World

Linux, MacOS, etc. command-line:

$ dev_appserver.py helloworld # run dev svr
$ appcfg.py update helloworld # deploy live

Windows GUI (also avail for Mac):

![App Engine Launcher](image)
Project Contents

- **app.yaml** – main configuration file
- **index.yaml** – automatically generated to index your data
- **main.py** – your main application "controller" code goes here

```yaml
application: new-project-template
version: 1
runtime: python
api_version: 1
handlers:
  - url: .*
    script: main.py
```

Hello World
Hello World

from google.appengine.ext import webapp
from google.appengine.ext.webapp import util

class MainHandler(webapp.RequestHandler):
  def get(self):
    self.response.out.write('Hello world!')

application = webapp.WSGIApplication([
  ('', MainHandler),
], debug=True)

util.run_wsgi_app(application)

Testing Your Install

$ cd google_appengine
~/Desktop/google_appengine/helloworld
$ dev_appserver.py helloworld
INFO  2009-03-04 17:51:22,354 __init__.py
  .
  .
  .

(optionally use the launcher for Macs & PCs)
Hello World

By default, your apps hosted at:
http://APP-ID.appspot.com
Create an App Id

Modify 'app.yaml'

```yaml
application: helloworld
version: 1
runtime: python
api_version: 1

handlers:
  - url: .*
    script: main.py
```
Upload the default application

$ cd google_appengine
~/Desktop/google_appengine/helloworld
$ appcfg.py update helloworld
Scanning files on local disk.
Initiating update.
Email: ...

Check it out
Summary: Why App Engine?

- Launch quickly
  - Familiar development tools
  - Fast app uploads & version management
- Innovate continuously
  - Many APIs enable continuous addition of new features
- Serve reliably
  - Google infrastructure, administration console
- Scale seamlessly
  - Elastic usage
- Distribute widely
  - Web, mobile app backend, Google Apps Marketplace

Thank You
Questions?

wesc+api@google.com
@wescpy / @app_engine
Your First Google App Engine App

```python
from google.appengine.ext import webapp
from google.appengine.ext.webapp import util

class MainPage(webapp.RequestHandler):
    def get(self):
        self.response.out.write('Hello world!')

application = webapp.WSGIApplication([('/', MainPage),], debug=True)

util.run_wsgi_app(application)
```
Putting main code in a function

```python
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app

class MainPage(webapp.RequestHandler):
    def get(self):
        self.response.out.write('Hello World!')

application = webapp.WSGIApplication([('/', MainPage),], debug=True)

def main():
    run_wsgi_app(application)

if __name__ == '__main__':
    main()
```

Add HTML

```python
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app

class MainPage(webapp.RequestHandler):
    def get(self):
        self.response.out.write('<h1>Hello World!</h1>')
```
Add HTML form

```python
class MainPage(webapp.RequestHandler):
    def get(self):
        self.response.out.write('<h1>Hello World!</h1>''
        self.response.out.write('''
        <html>
        <body>
        <form action="/sign" method=post>
        <input type=text name=content>
        <input type=submit value="Sign Guestbook">
        </form>
        </body>
        </html>
        ''')
```

Add signing handler

```python
from cgi import escape
from google.appengine.ext import webapp

class GuestBook(webapp.RequestHandler):
    def post(self):
        self.response.out.write("<html><body><h1>You wrote:</h1><html>$s</html>" % escape(self.request.get('content')))

application = webapp.WSGIApplication([
    ('/', MainPage),
    ('/sign', GuestBook),
], debug=True)```
Storing data

```python
from google.appengine.ext import db, webapp

class Greeting(db.Model):
    content = db.StringProperty(multiline=True)
    date = db.DateTimeProperty(auto_now_add=True)

class MainPage(webapp.RequestHandler):
    def get(self):
        greetings = Greeting.all()
        for greeting in greetings:
            self.response.out.write('<li> %s' % escape(greeting.content))

    def post(self):
        greeting = Greeting()
        greeting.content = self.request.get('content')
        greeting.put()
```

Adding users (and authors)

```python
from cgi import escape
from google.appengine.api import users

class Greeting(db.Model):
    author = db.UserProperty()
    content = db.StringProperty(multiline=True)
    date = db.DateTimeProperty(auto_now_add=True)

class MainPage(webapp.RequestHandler):
    def get(self):
        user = users.get_current_user()
        if user:
            self.response.out.write('<html><body>Hello %s!</body></html>' % user.nickname())
        else:
            self.redirect(users.create_login_url(self.request.uri))

    def post(self):
        greeting = Greeting()
        greeting.author = user
        greeting.content = self.request.get('content')
        greeting.put()
```
from os import path
from google.appengine.ext.webapp.template import render

class MainPage(webapp.RequestHandler):
  def get(self):
    user = users.get_current_user()
    greetings = Greeting.all()
    context = {
      'user': user,
      'greetings': greetings,
      'login': users.create_login_url(self.request.uri),
      'logout': users.create_logout_url(self.request.uri),
    }
    tmpl = path.join(path.dirname(__file__), 'index.html')
    self.response.out.write(render(tmpl, context))

Adding an HTML template (index.html)

<html><body>Hello {{ user.nickname }}
<h2>My Guestbook </h2><ol>
{% for greeting in greetings %}
<li>
  {% if greeting.author %}
  {{ greeting.author.nickname }}
  {% else %}
  <i>anonymous</i>
  {% endif %}
  {{ greeting.content|escape }}
{% endfor %}
</ol><hr>
<form action="/sign" method=post>
  <textarea name=content rows=3 cols=60></textarea>
  <input type="submit" value="Sign Guestbook">
</form></body></html>