"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***

What is Python?

An Introduction

+Wesley Chun, Principal
CyberWeb Consulting
wescpy@gmail.com :: @wescpy
cyberwebconsulting.com
goo.gl/P7yzDi

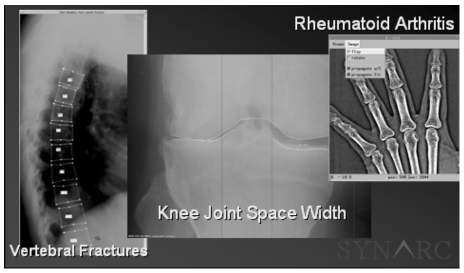
Python is Fun!

(I've used it at lots of places!)

slide

RocketMail

YAHOO! MAIL

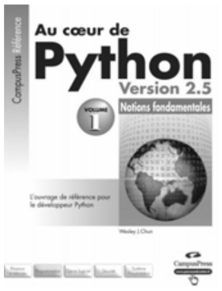


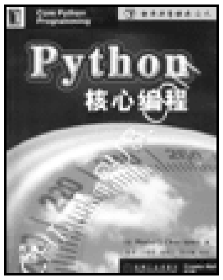
NearbyNow

I'm here to give you an idea of what it is!

(I've written a lot about it!)

Easy CROSS-PLATFORM Encryption
LINUX JOURNAL
WHY UPGRADE? Python 2.2
4.4 GHz of SCREAMING computation!
Reviewing the HOT HP x4000
Writing a functional KDE application
Learning Java by FIGHTING ROBOTS





I've taught it at lots of places!

(companies, schools, etc.)

CISCO

O'REILLY OSCON Open Source Convention JULY 20-24 2009, CA I'm Speaking

Google

PyCon Connecting The Python Community

AVAYA

UC Santa Cruz extension in Silicon Valley

O'REILLY OSCON Open Source Convention JULY 23 – 27, 2007 PORTLAND, OREGON Speaker

HITACHI

Foothill College Upgrade. Advance.

vmware

SEATTLE PUBLIC SCHOOLS

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

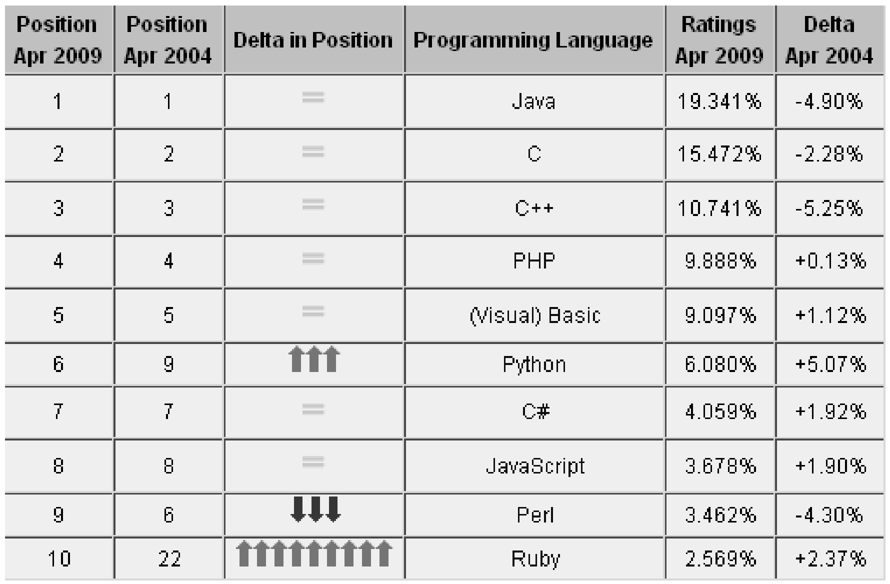
- Hopefully familiar with one other high-level language:
  - Java
  - C, C++, C#
  - PHP, JavaScript
  - (Visual) Basic
  - Perl, Tcl, Lisp
  - Ruby, etc.

- 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



Position Apr 2009	Position Apr 2004	Delta in Position	Programming Language	Ratings Apr 2009	Delta Apr 2004
1	1	=	Java	19.341%	-4.90%
2	2	=	C	15.472%	-2.28%
3	3	=	C++	10.741%	-5.25%
4	4	=	PHP	9.888%	+0.13%
5	5	=	(Visual) Basic	9.097%	+1.12%
6	9	↑↑↑	Python	6.080%	+5.07%
7	7	=	C#	4.059%	+1.92%
8	8	=	JavaScript	3.678%	+1.90%
9	6	↓↓↓	Perl	3.462%	-4.30%
10	22	↑↑↑↑↑↑↑↑↑	Ruby	2.569%	+2.37%

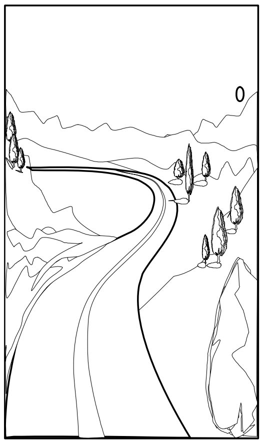
source: TIOBE Programming Community Index for April 2009 (tiobe.com)

For New Programmers

- Programming should be easy enough to teach like reading & writing (CP4E)
- Young kids: start with Scratch/Tynker
- Can start with Python: age ~10 & older
- Skills you need
  - 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 30 or 60 mins
- **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
  - ➔<u>Grassroots community</u>
- Disadvantages
  - ➔Generally slower than compiled languages
  - ➔Idiosyncratic idioms
  - ➔Obscure syntax? Naaah!!
  - ➔No marketing force… mostly word-of-mouth
  - ➔No world domination… yet! (Happening slowly.)

`www.tiobe.com/index.php/content/paperinfo/tpci`

Built for Non-Programmers

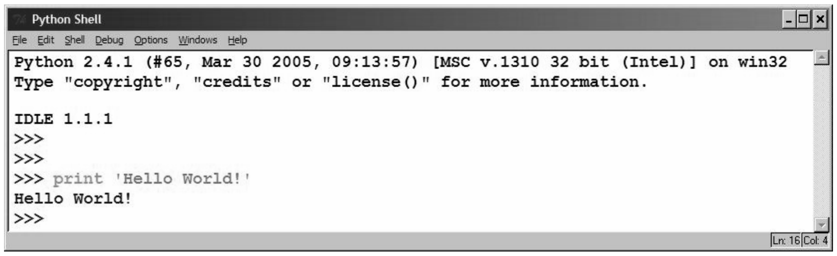
- Educational language syntax
- Works well for school-aged children
  - Syntax, memory management, data structures, object-oriented programming… such issues get in way of learning concepts
  - C++ and Java deter interest and students
  - Python: retention, morale, understanding
  - Probably the "best" 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)
  - *Invent Your Own Computer Games with Python* (Sweigart: 2010)
  - *Python Programming for the Absolute Beginner* (Dawson: 2010)
  - *Learning with Python: How to Think Like a Computer Scientist* (Downey, Elkner, Meyers: 2002)
  - *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

File Edit Shell Debug Options Windows Help

```
Python 2.4.1 (#65, Mar 30 2005, 09:13:57) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

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

Ln: 16 Col: 4

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

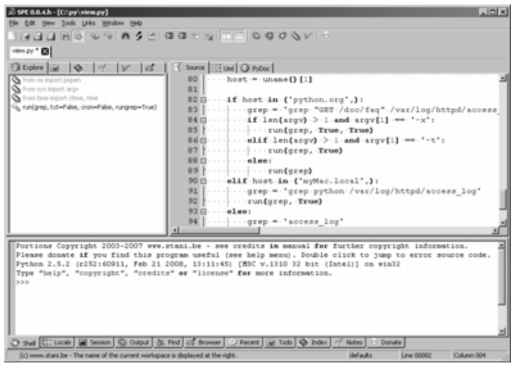
```
$ python     # or C:\> python
Python 2.6.2 (r262:71600, May 12 2009, 23:46:27)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print 'Hello World!'
Hello World!
```

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

- Unix (Linux, Mac OS): already installed
- PCs, source, and other downloads
  - python.org or corepython.com
- Other IDEs
  - SPE (pictured)
  - PythonWin
  - Eclipse + PyDev
  - Komodo
  - WingIDE
  - PyCharm


SPE 0.8.4.h - [C:\py\view.py]
File Edit View Tools Links Window Help
view.py *
Explore
Source Uml PyDoc
from os import popen
from sys import argv
from time import ctime, time
run(grep, txt=False, cron=False, rungrep=True)
Shell Locals Session Output Find Browser Recent Todo Index Notes Donate
(c) www.stani.be - The name of the current workspace is displayed at the right.
defaults Line 00082 Column 004

```
    host = uname()[1]

    if host in ('python.org',):
        grep = 'grep "GET /doc/faq" /var/log/httpd/access_
        if len(argv) > 1 and argv[1] == '-x':
            run(grep, True, True)
        elif len(argv) > 1 and argv[1] == '-t':
            run(grep, True)
        else:
            run(grep)
    elif host in ('myMac.local',):
        grep = 'grep python /var/log/httpd/access_log'
        run(grep, True)
    else:
        grep = 'access_log'
```

```
Portions Copyright 2003-2007 www.stani.be - see credits in manual for further copyright information.
Please donate if you find this program useful (see help menu). Double click to jump to error source code.
Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Common Beginner Gotchas

- Code delimited by indentation not braces { }

```
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 == 0.5` (not `0`)
- Freaky-looking floats: `1.1` → `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

```
>>> 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 is number %d' % (s[:6], 1)
>>> s * 2                   'Python is number 1'
'PythonPython'              >>> s = s[:6] + ' is cool'
>>> s = s * 2               >>> s
>>> s                       'Python is cool'
'PythonPython'              >>>
>>> s[4:6]                  >>> hi = '''hi
'on'                        there'''
>>> s[-1]                   >>> hi
'n'                         'hi\nthere'
>>> s[-4:-1]                >>> print hi
'tho'                       hi
>>> s[:6]                   there
'Python'
```

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 items in list`

List Operations

```
>>> 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']

>>> [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

```
>>> d = {'title': 'Core Python Programming', 'year': 2007}
>>> 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**

```
# 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`)**

```
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`

```
aList = [123, 'xyz', 45.67]
>>> for eachItem in aList:
...   print eachItem
123
xyz
45.67

>>> for i in range(0, 3): # [0, 1, 2]
...   print i
0
1
2
```

```
>>> i = 0
>>> while i < 3:
...     print i
...     i += 1
0
1
2
```

Files

- Open a file and get back a file object

```
f = open(file_name, 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

```
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

```
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

```
import module_name

import string
num = string.atoi('123')
```

- Importing module attributes using `from-import` statement
    - Names brought into local *namespace*

```
from module_name import module_element

from string import atoi
num = atoi('123')
```

- Packages: allow for organizing modules using the file system

Standard Library Sampler ("B.I.")

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

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

```
 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

```
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`
- Logger
  - `logging`
- Timer
  - `timeit`
- Help/Documentation
  - `pydoc`
- Testing
  - `unittest`
  - `doctest`
  - (external) `nose`
  - (external) `py.test`
  - Testing tools taxonomy `goo.gl/Fpz0Z`

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
  - http://www.informit.com/articles/article.aspx?p=1328795

Additional Resources

- Published Books
  - ***Quick Python Book*** **(Ceder, 2010)**
  - ***Core Python Programming*** **(Chun, 2006/2009)**
  - ***Python Fundamentals*** **LiveLessons DVD (Chun, 2009)**
  - ***Beginning Python*** **(Hetland, 2008)**
  - ***Dive into Python*** **(Pilgrim, 2009)**

  - ***Python Standard Library by Example*** **(Hellmann, 2011)**
  - ***Python Essential Reference*** **(Beazley, 2009)**
  - ***Python in a Nutshell*** **(Martelli, 2006)**
  - ***Python Cookbook*** **(2005 [2.x] & 2013 [3.x])**
- Other Resources

Python Reading List(s)	`goo.gl/i4u0R`
Python Quick Reference Guide	`rgruet.free.fr#QuickRef`
Worldwide Python Conferences	`www.pycon.org`
Core Python site & blog	`corepython.com & wescpy.blogspot.com`
comp.lang.python newsgroup	`groups.google.com`
PyPI/Cheeseshop repository	`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!**

Thank you!

Q&A

+Wesley Chun (plus.ly/wescpy)
@wescpy (twitter.com/wescpy)
wescpy@gmail.com

cyberwebconsulting.com
corepython.com
wescpy.blogspot.com