Introduction to Python

Grant Griffin
Iowegian International Corporation
http://www.iowegian.com

Topics

- What is Python?
- Why Python?
- How does it compare to other languages?
- Examples
- How to get started using Python
- Python for C/C++ programmers

What is Python?

- Created by Guido van Rossum in 1989
- Free/open scripting language:
  - Interpreted
  - Dynamically typed
- Both procedural and object-oriented
- Semantically similar to C/C++ and Perl
- Can easily be extended or embedded
  - Broad license (not GPL, but GPL compatible)
- General use, but suited to DSP system design:
  - Built-in support for complex numbers (!)
  - Support for vectors and matrices and numeric functions via extensions

Why Use a Scripting Language?

- Favors efficiency of programming over efficiency of execution
  - Eliminates need to manage memory
  - Quick change/rebuild cycle
- Useful for common auxiliary programming tasks:
  - File munging
  - Data analysis
  - Pseudo-shell language
  - Test aids
  - Easy cross-platform GUI programming
- Conclusion: *Every professional programmer should master a general-use scripting language*

Why Python?

- Free/open
- Has a clear, clean syntax
- Works as you expect
- “Batteries Included”
- Has lots of extensions and a strong community
- Highly portable
- Scales well
- Mature and stable
- Well documented

The (Abbreviated) Zen of Python

Beautiful is better than ugly. *(Compare to Perl, Ruby, and TCL)*
Explicit is better than implicit. *(Compare to Perl)*
Simple is better than complex. *(Compare to C++)*
Complex is better than complicated. *(Compare ot C++)*
Readability counts. *(Compare to Perl)*
Special cases aren't special enough to break the rules.
Although practicality beats purity. *(Compare to Matlab)*
Errors should never pass silently. *(Compare to Matlab)*
There should be one-- and preferably only one --obvious way to do it. *(Compare to Perl)*
Although that way may not be obvious at first unless you're Dutch. *(Compare to Ruby)*
Namespaces are one honking great idea -- let's do more of those! *(Compare to Matlab and C)*

Python Compared to C/C++

- Compared to C:
  - “High-level”: no need for memory management
  - Dynamically typed
  - Uses a small set of general-use data types
  - Uses dynamic binding (references) instead of pointers
  - Uses import rather than include
- Compared to C++
  - Simplified system of object classes
  - Does generic programming via binding rather than templates
  - “Batteries Included”

Python Compared to Other Scripting Languages

- Compared to Perl:
  - Semantically similar, but beautiful rather than ugly
  - "TSBOAPOOOWTDI" rather than "TMTOWTDI" (*ick*)
- Compared to TCL
  - Much more readable
  - Uses TCL's TkInter system as standard GUI
- Compared to Matlab:
  - **Free/open**
  - **General use**
  - Indexing is zero-based rather than "horrible"
  - Has strong support for namespaces and object-oriented programming
  - Supports scientific programming via extension modules

Python's Design

- Clean, minimal syntax: "executable pseudo code"
- Implemented in C and is generally C-like
- Uses indentation to delimit blocks
- Supports both procedural and object-oriented programming
- Uses a small set of powerful data types: float, int, list, tuple, dictionary (aka hash)
- Supports generic programming via dynamic binding rather than templating

Is Python a Real Programming Language?

- What about?
  - Lack of variable declarations and type safety
  - Execution speed (compilation)
  - Standardization
  - Use in large systems
- Python isn't a *systems* language but it's useful for nearly everything else

Interactive Example: Variables

```
ActivePython 2.5.2.2 (ActiveState Software Inc.) based on
Python 2.5.2 (r252:60911, Mar 27 2008, 17:57:18) [MSC v.1310 32
   bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more
   information.
>>> x = [1, 2, 3]              # x is a list
>>> x
[1, 2, 3]

>>> 7*14
98

>>> x=32;                      # x now is an integer
>>> x=32
>>> print x
32
```

Interactive Example: Dictionaries and Exceptions

```
>>> days_in_month = {'January':31, 'February':28, 'March':31}

>>> days_in_month['March']
31

>>> days_in_month['December']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'December'

>>> try:
...     days = days_in_month['December']
... except KeyError:
...     print 'Unknown month'
...
Unknown month
>>>
```

Interactive Example: Complex Numbers

```
>>> x = 1 + 1j
>>> y = 1 - 1j
>>> x * y
(2+0j)
>>> x + y
(2+0j)
>>> x - y
2j
>>>
>>> type(x)
<type 'complex'>
```

Example: Fibonacci Function

```
def fibo(n):
  "Returns the first n Fibonacci numbers"
  x = [1, 1]
  while len(x) < n:
          x.append(x[-1] + x[-2])
  return x[:n]
```

Interactive Example: An Empty Class

```
>>> class C:
...     pass            # do nothing
...
>>> c = C()
>>> c.x = 1
>>> c.y = 2
>>> print 'c.x=%i c.y=%i' % (c.x, c.y)
c.x=1 c.y=2
```

Example: A Simple Class

File simple_class.py:

```
    class Simple:

        def __init__(self, x, y):
            self.x = x
            self.y = y

        def __str__(self):
            return 'x=%i, y=%i' % (self.x, self.y)

    if __name__ == '__main__':
        simple_instance = Simple(2, 3)
        print simple_instance
```

…

```
J:\>simple_class.py
x=2, y=3
```

Which Python?

- Python has always been fully backwards compatible through version 2.x
- Version 3.x deliberately breaks compatibility to clean up minor things Guido didn't like
- A tool to automatically convert 2.x code to 3.x is provided
- Version 2.x is still fully supported.
  - Version 2.6.5 is current
  - More 2.x versions will be released
- Conclusion: You can learn/use either 2.x or 3.x.

Which Distribution?

- The canonical version is at python.org.
- “ActivePython” includes an IDE and a nice help system
- Python(x, y) at http://www.pythonxy.com is a bundle of Python 2.6 with lots of scientific and numeric extensions
- Others: SAGE, IronPython (.NET), Jython, Cython

Getting Started With Python

- Download and install Python
- **Go through the tutorial that's built into the Python help system**
- Read "Learning Python" or "Dive Into Python" (available free online at http://diveintopython.org/)
- Start writing Python
- Learn Regular Expressions
- Read the Python newsgroup, comp.lang.python

Python for C Programmers

- Use indentation for blocks instead of {}
- Use “#” for end-of-line comments. (No block comments)
- Semi-colons are optional
- Variables:
  - Declare variables implicitly via assignment:
    - `x = 3  # x is an integer`
    - `x = 'string'  # x is a string`
    - `x = []  # x is a list`
  - Don’t use “const”, “static”, etc.
  - Don’t use pointers (not needed with dynamic binding)
  - Can change a variable’s type on-the-fly, as above

More Python for C Programmers

- Use “import” rather than “#include”
- Use “def” to define a function
- Use “%” operator to format strings ala sprintf
- Use strings instead of characters
- Use += and -= instead of ++ and --
- “for” iterates over a sequence, e.g. “for x in y”
  - use “while” to do C-style for and do/while loops

Python for C++ Programmers

- Use “__init__” for (optional) constructors
- Use explicit “self” rather than implicit “this”
- Constructors of base classes aren’t called automatically: call them yourself if you want to
- Don’t worry about privacy:
  - "If you have something that you don't want anyone to know, maybe you shouldn't be doing it in the first place." Eric Schmidt, CEO of Google
- All class functions are virtual
- Override operators via special function names, e.g. use “__gt__” rather than “operator>“

Summary

- What is Python?
  - A free/open, general-use object-oriented scripting language
- Why Python?
  - It's the Swiss Army Knife of programming languages
- How does it compare to other languages?
  - As good as or better in almost every way, though not a systems language
- How can you get started using it?
  - Go through the tutorial, read a book or two, practice

Final Thought

A day without Python
is like a day without sunshine