Introduction to Python

Summer Teacher Institute
2007

Fourth Watch Software LC

Overview

- Why I am here
- Why Johnny can't Code
  - http://www.salon.com/tech/feature/2006/09/14/basic/
- Why Python?
- Technical Resources
- Usage “HowTo” Resources
- What is it “good” for?
- What Python is not good for
- Characteristics of Python
- Brief look at Python

Overview

- First, there is absolutely no way I can teach you anything meaningful about Python in a few minutes.
- My goal is to suggest meaningful reasons you should look at Python as a vehicle to encourage and enthuse students in the use of a computer to “scratch their itch” by programming their own solution.
- It is a BIG mistake to “optimize early” since you don't know what needs optimization

Overview

- Using languages like C and C++, etc, are examples of optimizing early:
  - Usual “excuse” is “speed” or “efficiency”.
  - Code is reduced to “machine instructions” so is “faster” than using a “vitrual machine” of some kind.
- Using languages like Java and C#, etc
  - Both almost as much work as C or C++.
  - Memory management is somewhat easier.

Why Python?

- I got into programming back when BASIC was new (circa 1970)
- Programming was fun, BASIC was easy for easy things, ok for hard things, failed for difficult things
- Enter Pascal, C, then C++, Java, etc
- Never got into Perl as after I wrote a program and left it for a while, couldn't understand it later without working hard, awk and C were easier.

Why Python?

- After 20+ years of programming, writing code was getting too tedius. Solving even an easy problem was a lot of work.
- In the 1980's developed what was called a “fourth generation language” which would normally “just do” what you probably wanted, letting you override the details if the “normal” was not what you wanted.
- Got back into the “dynamic language” space in 1991 using MUMPS.

Why Python?

- Became a big fan of “dynamic language” environments, because:
  - Immediate feedback
  - Didn't have to write a “program” to see how something (operator, function, etc) worked
  - Didn't have to worry about managing memory as a “garbage collector” was constantly watching for things no longer needed
  - I could concentrate on the problem I wanted to solve rather than on trying to figure out why my program just “cored”.

Why Python?

- My Python journey
  - Learned of Python in mid-1990's
  - With a leader named “Guido” -- how cool!
  - Started attending the International Python Conferences in about 1999 (IPC8)
  - Have been an active member of the PyCon organizers for several years
  - Have been impressed with the leadership of the Python community
  - Active development of the language, but stability as well

Why Python?

- My Observations about the Python community:
  - Python has a large core of developers and is not dependent on Guido to survive
  - Several implementations of Python exist
    - CPython – the traditional reference
    - Jython – written in Java generates JVM
    - IronPython – Microsoft's .NET implementation
    - IPython – an enhanced interactive Python
  - Very open to new fans of Python
  - Expanding and coalescing (refining, refactoring) which is true of the code as well

Technical Resources

- Python programming language
  http://www.python.org
- SimPy simulation package
  http://simpy.sourceforge.net/index.html
- Pygame graphical interface for games
- SimPlot basic plotting package for SimPy
- SimGUI graphical interface to SimPy models
- SimulationStep allows stepping thru model event by event
- SimulationRT allows synchronization to real time

Usage “HowTo” Resources

- Example: The Bank
  http://simpy.sourceforge.net/SimPyDocs/TheBank.html
  http://simpy.sourceforge.net/SimPyDocs/TheBank2.html
- Modeling of a Fuel Fabrication Facility Using Python and SimPy (LANL)
  http://www.python.org/pycon/dc2004/papers/16/
- Introduction to the SimPy Discrete-Event Simulation Package (Professor Norm Matloff)
  http://heather.cs.ucdavis.edu/~matloff/simpy.html
- SimPy: Simulation Systems in Python
  http://www.onlamp.com/lpt/a/3257

What is it “good” for?

- Good “starter” language
  - Don't forget it is programming – can be hard
  - Dynamic/interactive environment
  - No funky syntax: `print 'hello, world'`
  - Visibility of “Code blocks”
  - No mandatory declaration of variables
  - No mandatory declaration of types
  - Just write code

What is it “good” for?

- Good “intermediate” language
  - Software may be organized in modules
  - Software may be organized in packages
  - Refactoring facilitated by code block syntax
  - Blocks defined by indentation
  - No misleading block structure due to an unseen block signal token (e.g. “{“ “}” or begin/end)
  - Larger bodies of code tends to live longer, coming back to “cold code” is not difficult (c.f. Perl, C, C++, Java, etc)
  -
-

What is it “good” for?

- Good “advanced” language
  - Well defined object hierarchy.
  - Multiple inheritance with well defined “MRO” (Method Resolution Order) so programmers are not forced to introduce artificial “interfaces” when the object model naturally would use multiple inheritance.
  - Can “decorate” (Python “borrowed” the Java “annotation” approach) functions/methods to give short-hand for complex specifications.
  - Dynamic creation & compilation of code and object structures

What is it “good” for?

- Good “expert” language
  - You can start out simple, and grow the problem to a very sophisticated package
- Significant applications are written in Python
  - YouTube
  - RedHat / SuSE installation / system admin
  - BitTorrent
  - Plone/Zope (content management systems)
  - Google – current home/sponsor of Python
- It starts out small and simple, yet is ready for “industrial strength” use.

What Python is not good for

- When the problem is computationally bound and the limit is the Python virtual machine
  - LANL/LBL/Sandia/NASA and others have done much for making numerical analysis in Python effective.
  - SWIG (Simplified Wrapper Interface Generator) makes specification of inter-language interfaces formal and automated.
  - Interfacing between Python and C libraries is well defined, so you can write an interface to your C module as needed.

Characteristics of Python

- It comes “Batteries included”
- It “fits the brain”
- Programming:
  - The way
  - Guido
  - Indented it
- Life is better without braces

Characteristics of Python

- Platform independent
- Yet, may bind to platform dependent libraries
- Many useful built-in data types
- Ability to create user data types and add operators that work on them (like operator overloading in C++)
- Well integrated with graphical libraries that work the same, **no code changes**, across platforms such as Linux, Mac OS X, and (if you must) Microsoft Windows
-

Characteristics of Python

Characteristics of Python

Brief Look at Python

```
>>> print “Hello, World!”
Hello, World
```

Brief Look at Python

```
>>> 2 + 2
4
>>> 2 ** 64
18446744073709551616L
```

Brief Look at Python

```
>>> import math
>>> math.e
2.718281828459045
>>> math.pi
3.141592653589793
>>> math.sin(math.radians(90))
1.0
```

Brief Look at Python

```
>>> def fib(n):
...     if n == 1:
...         return n
...     else:
...         return n * fib(n-1)
...
>>>
```

Brief Look at Python

```
>>> fib(1)
1
>>> fib(2)
2
>>> fib(3)
6
>>> fib(4)
24
```

Brief Look at Python

```
>>> for i in range(6):
...   print i, fib(i)
...
0 1
1 1
2 2
3 6
4 24
5 120
>>>
```

Brief Look at Python

```
>>> s = "a string"
>>> s.find("z")
-1
>>> s.find("i")
5
>>> s[5]
'i'
>>>
```

Brief Look at Python

```
>>> s = "a string"
>>> s.find("z")
-1
>>> s.find("i")
5
>>> s[5]
'i'
>>>
```

Brief Look at Python

```
>>> s = "%d squared is %d" % (
        123, 123**2)
>>> s
'123 squared is 15129'
>>>
```

Brief Look at Python

```
>>> a = "apple"
>>> p = 0.49
>>> n = 2
>>> "%d %s%s $%5.2f" % (n, a,
     n != 1 and "s" or "", n * p)
'2 apples $ 0.98'
>>> n = 1
>>> "%d %s%s $%5.2f" % (n, a,
     n != 1 and "s" or "", n * p)
'1 apple $ 0.49'
```

Brief Look at Python

```
>>> lst = [1, 2, 3, 4]
>>> lst[0]
1
>>> lst.append(5)
>>> lst
[1, 2, 3, 4, 5]
>>>
```

Conclusion

- Programming is not easy
- In fact it is hard
- Programming in C++ is roughly equivalent to requiring your students to do all long division using Roman numerals, with no concept of 0
- It can be done (so I'm told), but why?
  - If there is a good reason to require long division to be done in Roman numerals, I'd like to hear it.
- Keep programming fun
- It's tough enough out there.