Introduction to Zope

Part 1: Python

Kirk Strauser

Zope is a web application server, similar in concept to proprietary products like Cold Fusion. However, it is free software that is available under the GPL-compatible Zope Public License, which is very similar to the BSD License. Zope was designed with the specific goals of creating a powerful, secure framework for the development of robust web-based services with a minimum of effort.

However, Zope's biggest distinguishing characteristic is how closely it models the language it is written in: Python. In fact, many of its features are directly derived from its underlying Python structure. Because of that, it's difficult to truly understand or appreciate Zope without having a basic knowledge of Python. This article, the first in a two part series, is intended as a high-level introduction to the language. Next month's instalment will build upon this by demonstrating practical examples of Zope code.

Zope's biggest distinguishing characteristic is how closely it models the language it is written in: Python

Language features

Although Python has been in use since the early 1990's, it's only become relatively popular in the last few years. Many programmers view it as the spiritual successor to Perl. That is, it's an expressive, interpreted language that's equally at home in small system scripts or much larger applications. However, it has the deserved reputation of usually being easier to read and maintain than the equivalent Perl code. Python also sports an excellent object oriented approach that's much cleaner and more integral to the overall design than is Perl's. Perhaps most important, though, is a belief by the core development team in doing things the right way. It was designed from the beginning with an emphasis on practical elegance—Python strives to allow programmers to easily express their ideas in intuitive ways.

It was designed from the beginning with an emphasis on practical elegance—Python strives to allow programmers to easily express their ideas in intuitive ways

Significant whitespace

The first thing that everyone notices about Python is its use of significant whitespace. Rather than marking blocks of code with keywords such as "begin" and "end", or curly brackets a la C, Python sets them apart with indentation. Frankly, a lot of programmers hate the idea when they first see it. If you're one of them, don't be discouraged; the feeling passes quickly. It enforces the style guidelines that most good programmers would be following anyway, and soon becomes quite natural. Python is flexible regarding the use

of spaces versus tabs, as long as you consistently use the same kind and amount of whitespace to indent. Furthermore, almost all programming editors have Python modes that handle the details for you.

The standard comparison of formatting between C and Python is the "factorial" function. In C, that could be written as:

```
int factorial(int i) {
    if(i == 1) {
        return 1;
    }
    else {
        return i * factorial(i - 1);
    }
}
```

(or in one of many other common styles). A Python programmer would probably write something extremely similar to:

```
def factorial(i):
    if i == 1:
        return i
    else:
        return i * factorial(i - 1)
```

Except for the missing curly brackets, the formatting is almost identical between the two.

Interactive development

Python includes an interactive shell where you can experiment and test new code. Running the `python` command without any arguments will result in something like:

```
Python 2.3.5 (#1, Apr 27 2005, 08:55:40)
>>>
```

At this point, you can enter Python commands directly to see their effect. If you're working on a large project, you can load specific parts of it for manual testing without affecting other modules. It's equally handy for verifying that short functions will work as expected before embedding them into a larger body of code. It's difficult to convey exactly how convenient this is, and how efficient the code-experiment-code cycle can be.

Finally, the interactive prompt is an excellent place to explore objects, and the data and functions inside them. Typing dir(someobject) will return the list of objects referenced by someobject, and most of the functions in Python's core libraries contain a __doc__ attribute with usage information:

Python keywords

The whole language is built upon a short list of words: and, del, for, is, raise, assert, elif, from, lambda, return, break, else, global, not, try, class, except, if, or, while, continue, exec, import, pass, yield, def, finally, in, and print. If you've ever written a program, you probably already have an accurate idea of what most of them do.

```
>>> dir(str)
[lots of stuff, ..., 'translate', 'upper', 'zfill']
>>> print str.upper.__doc__
S.upper() -> string

Return a copy of the string S converted to uppercase.
```

Tiny core language

Python 2.3.5, the version recommended for use with the latest production release of Zope, has just 29 reserved words. Perl has quite a few more: 206 as of version 5.6.8. PHP tips the scales with up to an incredible 3972 commands and functions in the base language (although many can be added and removed at compilation time). The practical upshot is that any experienced programmer should be able to memorize the entire language in an evening. This simplicity does not reflect a lack of power though. Although most of the familiar commands are similar to their counterparts in other languages, several are significantly more flexible. The for command, as an example, will cheerfully iterate across a set of numbers, a list of strings, or the keys of a dictionary object.

Any experienced programmer should be able to memorize the entire language in an evening

Strong dynamic typing

Python is dynamically typed, which means that it executes its type checks during program execution (as opposed to C). It is also strongly typed, meaning that it won't convert data from one type to another unless you explicitly ask it to (as

opposed to Perl). The language makes great use of this flexibility by passing parameters to functions as reference instead of by value. The net effect is that you can pass almost any object to a function, and if the operations in the function make sense for that type of object, then the function will work as expected. For example, the following code defines a function that will add any two compatible values together:

```
>>> def add(a, b):
...     return a + b
...
>>> add(1, 2)  # Two integers
3
>>> add([1,2,3], [4,5,6])  # Two lists
[1, 2, 3, 4, 5, 6]
>>> add('1', 2)  # A string and an integer
TypeError: cannot concatenate 'str' and 'int' objects
```

In practice, this allows you to write generic code that can operate on any number of data types without additional modification.

Python is dynamically typed, which means that it executes its type checks during program execution (as opposed to C)

Garbage collection

Never alloc() or free() memory again. Python automatically allocates space to store your data structures and frees it when you're finished with them. This has numerous large advantages. First, it frees programmers from the low-level details that waste their time. By allowing you to concentrate on algorithms and design instead of pedantic record keeping, it gives you the freedom to spend your time where it's most useful. Second, it eliminates an entire class of efficiency and security errors. You don't have to worry about the buffer overruns or memory leaks that C programmers must carefully avoid. Finally, it's fast. While experts could possibly write optimized memory management routines for themselves, Python is much better at the task than the vast majority of average users.

Garbage collected memory management has quite a few other non-obvious benefits, with complex datatypes being near the top of the list. To compose a list of various types objects, you simply create them and put them together into such a list:

```
>>> a = 'a short string';
>>> b = [1, 2, 3]
>>> c = [a, b]
>>> c
['a short string', [1, 2, 3]]
```

Whenever the program's execution moves past the scope where these objects are defined, they simply vanish. Compare this with other languages that would require you to track an objects existence by hand and decide whether (a) you're truly finished using that object, or (b) another object still references them. This isn't the same as saying that programmers *never* have to consider memory usage—bad design is still bad design—but the penalties for not doing so are far smaller.

Object oriented and functional programming

In Python, almost everything is an object. A module is an object that contains definitions of other objects. Classes are objects that contain functions and variables. Functions themselves are objects. Since values are passed to functions by reference, this means that you can pass functions just as easily as integers, strings, or any other objects. In the example below, I define three simple functions that perform an operation on a number and return the result. Then, I define another function, which takes a number and a function to call with that number, and execute it with some sample values:

```
>>> def plusone(a): return a + 1
...
>>> def plustwo(a): return a + 2
...
>>> def timesthree(a): return a * 3
...
>>> def math(number, operation):
...           return operation(number)
...
>>> math(1, plusone)
2
>>> math(2, plustwo)
4
>>> math(3, timesthree)
9
```

You don't have to worry about the buffer overruns or memory leaks that C programmers must carefully avoid

This simple pattern is used widely in Python. For example, the list.sort function can use an optional function to compare two values in a list and order them appropriately. Various GUI toolkits work by passing functions to event handlers so that they're executed when the respective events occur. Functions can even be stored in other data structures, such as dictionaries, and retrieved as needed.

Pervasive namespaces

As mentioned above, imported modules are just another kind of object. This means that rather than bringing the functions and variables from a module into the current namespace (as C does), they remain within their named object:

```
>>> import time
>>> dir(time)
[a lot of stuff, ..., 'asctime', 'clock', ...]
>>> clockName
Error: name 'clock' is not defined
```

Thanks to this feature, you don't have to worry about conflicting names from unrelated modules. Experienced programmers should immediately appreciate the organizational advantages this brings. Novices will like the fact that they're not immediately faced with an overwhelming number of functions.

Python is one of a new generation of cross-platform programming languages. It's simple enough that new programmers can immediately start using it immediately, but it's equipped with the tools that make experts rejoice

Conclusion

Python is one of a new generation of cross-platform programming languages. It's simple enough that new programmers can immediately start using it, but equipped with the tools that make experts rejoice. It freely mixes imperative, object oriented, and functional programming so that you can choose the approach most appropriate for the task at hand. It's used by companies such as Google and websites like Wikipedia, and is quickly becoming a common choice for new application development.

I haven't forgotten about Zope. However, the features that have made it a powerful and popular application server originate in Python, and to truly "get" Zope, you must have a passing understanding of Python. In next month's column, I'll explore the ties between the two and demonstrate Zope's power by implementing several practical web application components.

Bibliography

Python Keywords (http://www.python.org/doc/2.3.5/ref/keywords.html)

Perl Functions (http://perldoc.perl.org/index-functions-by-cat.html)

PHP Quick Reference (http://www.php.net/quickref.php)

PHP 'Reserved' Words (http://us2.php.net/manual/en/reserved.php)

Copyright information

About the author

Kirk Strauser has a BSc in Computer Science from Southwest Missouri State University. He works as a network application developer for The Day Companies, and runs a small consulting firm (http://www.strausergroup.com/) that specializes in network monitoring and email filtering for a wide array of clients. He has released several programs under free software licenses, and is active on several free software support mailing lists and community websites.