Introduction to the Python Programming Language

Scott Snyder

June 4, 1999

And now for something completely different.

The Life of Python

- "an interpreted, object-oriented, high-level programming language with dynamic semantics."
- Features:
  - Very clear and coherent syntax and overall design.
  - Support for object-oriented programming.
  - Powerful collection of built-in types.
  - Easy to extend/embed with C/C++.
  - Large library of services and extensions, including interfaces to many GUI toolkits (Tk, MFC, wxwindows, KDE, Gnome, etc.).
  - Portable across many platforms (Unix, VMS, Mac, VxWorks, NT, etc.).
  - Freely available in source form.
- First developed 1990 by Guido van Rossum at CWI, Amsterdam.
- Now a large community of users.

Python Applications

- Much faster to develop in Python than in languages like C++.
- Scales well from small scripts to medium-to-large applications (100's of source files).
- *Not* recommended for compute-intensive or time-critical applications.
  - But can be used as high-level 'glue' code in such applications.
- Being used extensively in the online system for control and user interface tasks.
  - Python interfaces exist for EPICS and d0me.
- Example:

```
# Recursive factorial computation.
def fac (x):
    if x > 1:
        return x * fac (x-1)
    return 1
```

Running Python

- Setup (on DØ systems):

  ```
  setup python
  ```

  Currently, this sets up 1.5.1. The newer 1.5.2 is installed on some systems; use

  ```
  setup python v1_5_2
  ```

- Running the interpreter:

  python

  Use Control-D (EOF) to exit.

- From Emacs:
  - Python-mode is included with xemacs.
  - From a python buffer:
    * 'C-c !' starts an interactive interpreter.
    * 'C-c C-c' sends the current buffer to the interpreter.

- Idle
  - Python IDE in development.
  - Requires Python 1.5.2.
  - `$PYTHON_DIR/src/Python/Tools/idle/idle.py`

Simple Examples

- Numbers

```
>>> 2+3*4   # Integers.
14
>>> a=2     # Assignment to a variable.
>>> a**10
1024
>>> a**100  # This gets an overflow error.
Traceback (innermost last):
  File "<stdin>", line 1, in ?
OverflowError: integer pow()
>>> 2L**100  # Long integers.
1267650600228229401496703205376L
>>> 3.4 / 5.6  # Floats.
0.607142857143
>>> abs (-10)  # A built-in function.
10
>>> (2 + 2.1j)**4  # Complex numbers.
(-70.3919-6.888j)
```

Simple Examples

- Strings

```
# String constants and concatenation
>>> print "It’s " + ’"Monty Python\’s Flying Circus"’
It’s "Monty Python’s Flying Circus"
>>> ’Ni! ’ * 4  # Repetition.
’Ni! Ni! Ni! Ni! ’
>>> spam = ’spam’
>>> spam[1]     # Slicing.
’p’
>>> spam[-1]
’m’
>>> spam[1:3]
’pa’
>>> spam[1:]
’pam’
>>> len (spam)     # len built-in
4
>>> ’This is an ex-%s!’ % ’parrot’   # Formatting.
’This is an ex-parrot!’
```

Simple Examples

- Lists

```
>>> l1 = []   # Empty list
>>> l2 = [1, 'dimsdale', ['gumby', 'McTeagle']]
>>> l2[1]   # Indexing.
'dimsdale'
>>> l2[1] = 'spiny norman'  # Assignment.
>>> l2
[1, 'spiny norman', ['gumby', 'McTeagle']]
>>> l3 = ['spam'] * 4 + ['egg']  # Concatencation.
>>> l3.append ('spam')   # Concatenation.
>>> l3
['spam', 'spam', 'spam', 'spam', 'egg', 'spam']
>>> l3[3:5]    # Slicing.
['spam', 'egg']
>>> l3[3:5] = ['tomato']  # Slice assignment.
>>> l3
['spam', 'spam', 'spam', 'tomato', 'spam']
>>> len (l3)   # Built-in len() function.
5
>>> del l3[-2]  # Removing an element.
>>> l3
['spam', 'spam', 'spam', 'spam']
>>> l4 = range (5)  # Building a list of ints.
>>> l4
[0, 1, 2, 3, 4]
>>> 3 in l4,  10 in l3  # Membership testing.
(1, 0)
```

Simple Examples

- Dictionaries

```
>>> table = {'Perl':    'Larry Wall',
...          'Tcl':     'John Ousterhout',
...          'Python':  'Guido van Rossum' }
>>> table['Python']
'Guido van Rossum'
>>> table['C'] = ['Brian Kernighan', 'Dennis Richie']
>>> table['C']
['Brian Kernighan', 'Dennis Richie']
```

- Files

```
>>> f = open ('/usr/dict/words')
>>> for i in range (4): print f.readline(),
a
AAA
AAAS
aardvark
>>> words = f.readlines()
>>> len (words)
25482
>>> f.close()
>>> words[1000]
'annoyance\012'
>>> print words[15000],
migrant
>>> words.index('python\n')
18654
```

Interlude: Python isn't C++ Assignment

- C++:
  - Variables name regions of memory.
  - Assignment is a *copying* operation — the source is copied to the destination.

    ```
    list<int> a;
    list<int> b = a;
    ```

    sets `b` to a *copy* of `a`.

- Python:
  - Assignment is a *naming* operation — it assigns a new name to an object.

    ```
    >>> l1 = [1, 2, 3]
    >>> l2 = l1
    ```

    Now `l1` and `l2` name the *same* object:

    ```
    >>> l2
    [1, 2, 3]
    >>> l2[2] = 10
    >>> l1
    [1, 2, 10]
    ```

Simple Statements

- Assignment.

```
spam = 'SPAM'              #     basic form
spam, ham = 'yum', 'YUM'   #     tuple assignment
spam = ham = 'lunch'       #     multi-target
```

- Expressions.

```
spam(egs, ham)                #   function calls
spam.ham(eggs)                #   method calls
spam                          #   print interactive
spam < ham and ham != eggs    #   compound expr's
spam < ham < eggs             #   range tests
```

- Print.

```
print spam, ham   # print objects to stdout
print spam, ham,  # don't add linefeed
```

Control structures

- If

```
if x == 'bunny':
    print 'hello little bunny'
elif x == 'bugs':
    print "what's up doc?"
else:
    print 'Run away!  Run away!...'
```

- While

```
x = y / 2
while x > 1:
    if y % x == 0:                    # remainder
        print y, 'has factor', x
        break                         # skip else
    x = x-1
else:                                 # normal exit
    print y, 'is prime'
```

- For

```
>>> for x in ["spam", "eggs", "spam"]: print x,
spam eggs spam
>>> for i in range(5): print 'A shrubbery!'
A shrubbery!
A shrubbery!
A shrubbery!
A shrubbery!
A shrubbery!
```

Syntax Notes

- Comments introduced with '`#`'.

- Block structuring is encoded by indentation.
  - No chance for the lexical structure to be inconsistent with the logical structure.
  - Smart editors (such as emacs's python-mode or Idle) help with this.

- Lines may be continued with a backslash:

  ```
  if woman.weight() == \
     duck.weight():
      print "She's a witch"
  ```

  But the backslash can be omitted if there's a pending open delimeter:

  ```
  if (woman.weight() ==
      duck.weight()):
      print "She's a witch"
  ```

Function Definitions

- Example:

```
def fibo (lim, a=0, b=1):
    while b < lim:
        print b,
        a, b = b, a+b
    return b

>>> fibo(100)
1 1 2 3 5 8 13 21 34 55 89
144
>>> fibo(100, 2, 5)
5 7 12 19 31 50 81
131
```

- Note that `def` is an executable statement, and functions are objects.

```
def inc (x): return x+1
def double (x): return x*2
def compose (a, b):
    def tmp (x, a=a, b=b):
        return a (b (x))
    return tmp
>>> f = compose (inc, double)
>>> f (3)
7
>>> compose (double, inc) (3)
8
```

Interlude: Scoping, or How Not To Be Seen

- User identifiers are looked up in the *local* scope and the *global* scope. If a name is on the left side of an assignment, it is put in local scope.

```
>>> x = 1   # In global scope
>>> def foo(): print x   # Found in global scope
>>> foo ()
1
>>> def bar():
...     x = 2      # Assigned in local scope
...     print x
>>> bar()
2
>>> x
1
```

- A name in the global scope can be assigned by using the `global` statement:

```
>>> def fee():
...     global x  # Look in global scope for x
...     x = 2     # Assigned in global scope
...     print x
>>> fee()
2
>>> x
2
```

Interlude: Scoping

- But note this common error:

```
>>> def fum():
...     print  x  # Looks up x in local scope
...     x = 4     # (Due to this assignment)
>>> fum ()
Traceback (innermost last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 2, in fum
NameError: x
```

- Scopes do not nest.

```
>>> def f1 ():
...     x = 'albatross'
...     def f2 (): print x # Finds x in global scope
...     return f2
>>> x
2
>>> f=f1()
>>> f()
2
```

- Standard solution:

```
>>> def f1 ():
...     x = 'albatross'
...     def f2 (x=x): print x
...     return f2
```

Library Modules

- Library code is contained in *modules*.
- Each module has its own namespace.
- Use `import` to gain access to the module:

```
>>> import math
>>> math.sqrt (4)
2.0
>>> math.log (10)
2.30258509299
```

- Use `from ... import ...` to import names from module into the current namespace.

```
>>> from math import log
>>> log (10)
2.30258509299
```

- Write your own modules by putting python definitions in a file ending with `.py`.
  - `import` will search `PYTHONPATH` for module files.
  - `.py` files will be automatically compiled to `.pyc` files. You shouldn't need to touch the `.pyc` files.

Object-Oriented Programming

- Classes created with the `class` statement.

- Methods always take the instance as the first argument (conventionally named `self`).

```
class Myclass:
    def meth (self):
       print 'Hello world'

>>> c = Myclass()  # Create a class instance.
>>> c.meth()       # Call the method.
Hello world
```

- For instance variables, '`self`' acts like a namespace.

```
class Myclass:
    def meth (self, x):
        self.var = x     # Instance variable.
        self.__priv = x  # Private instance variable.

>>> c = Myclass()
>>> c.meth(3)
>>> c.var
3
```

With inheritance and constructors

```
class Actor:
    def __init__ (self, name):  # Constructor.
        self.__name = name
    def line (self, line):
        print '%s says: %s' % (self.__name, line)

class Clerk (Actor): # Inherits from Actor.
    def __init__ (self):  # Constructor.
        Actor.__init__ (self, 'Wensleydale')
    def reply (self, cheese):
        if cheese == 'Camembert':
            self.line ("It's a bit runny.")
        else:
            self.line ("We're right out of %s today."
                       % cheese)

class Customer (Actor): # Inherits from Actor.
    def __init__ (self, clerk):  # Constructor.
        Actor.__init__ (self, 'Customer')
        self.__clerk = clerk
    def ask (self, cheese):
        self.line ('Do you have %s?' % cheese)
        self.__clerk.reply (cheese)

def skit ():
    cust = Customer (Clerk ())
    for cheese in ['red Leicester', 'Tilsit',
                   'Camembert', 'Caerphilly']:
        cust.ask (cheese)
```

A Useful Application

- Copy an object given by URL to a local file.

```
#!/usr/bin/env python
# Standard trick to find interpreter in PATH.

import sys
import urlparse
import string
from urllib import FancyURLopener

# Arg list available in sys.argv.
if len (sys.argv) < 2:
    print "Usage: %s url [file]" % sys.argv[0]
    sys.exit (1)

url = sys.argv[1]

if len (sys.argv) > 2:
    file = sys.argv[2]
else:
    # If only one arg was given, pull of the trailing
    # component of the URL.
    path = urlparse.urlparse (url) [2]
    file = string.split (path, '/') [-1]

# Copy URL into FILE.
opener = FancyURLopener ()
opener.addheader ('Accept', '*/*')
opener.retrieve (url, file)
```

Tkinter

- Tkinter is a python interface to the Tk toolkit.
- Can only give a brief example here.

```
from Tkinter import *

top = Tk()

def b1_cmd ():
    print 'Ooh, that tickles!'
    return

b1 = Button (top, text = 'Push me', command = b1_cmd)
b1.pack ()
b2 = Button (top, text = 'Quit', command = top.quit)
b2.pack ()

top.mainloop ()
```

Further Information

- Python web site: `http://www.python.org/`
  - Standard documentation set.
  - Tutorials.
  - Essays, white papers.
  - Contributed software.
- Books:
  - David Ascher and Mark Lutz, **Learning Python**, O'Reilly, 1999.
  - Aaron Watters, Guido van Rossum, and James Ahlstrom, **Internet Programming with Python**, Henry Holt, 1996.
  - Mark Lutz, **Programming Python**, O'Reilly, 1996.
- DØ packages:
  - `thread_util_wrappers`
  - `d0me_wrappers`

NOBODY expects the Spanish Inquisition!

Questions?