Python and the P-Sim Simulation Tool

Petr Grillinger
University of West Bohemia
DSS Research Group

22. 3. 2004

Basic Features of Python

- Interpreted, interactive OO language
- Based on the ABC language
- Developed since 1991 by *Guido van Rossum*
- Intended for SW component integration:
  - Native interface to C/C++/Java
  - Embedding into most common languages
  - Wide platform support, portable
- And computer science teaching (like ABC)

Motivation (FSF)

To say I was astonished would have been positively wallowing in understatement. It's remarkable enough when implementations of simple techniques work exactly as expected the first time; but my first meta class hack in a new language, six days from a cold standing start? Even if we stipulate that I am a fairly talented hacker, this is an amazing testament to Python's clarity and elegance of design.

Eric Raymond, Author of The Cathedral & The Bazaar

- Learning speed – small, compact, C-like syntax
- Readable code – unambiguous, non-cryptic
- High productivity – rich libraries, expressive, GC

Motivation (Commercial)

- *Web-Lite* – E-commerce server SW, 1996 bought by Microsoft: in C++ and gradually switched to Python
- Benefits:
  - Code lines count lower by 90%
  - More stable (longer up-time) – no memory leaks
  - Faster development – no compile-link cycle
- Issues:
  - No type information in interface (naming convention)
  - Terrible thread implementation (no real parallelism)
  - Little development tool support (at the time)

Case study from Sixth Python Conference, Greg Stein

Data Types

- Explicit types are used seldom
  - Implicit type conversion
  - The proper type is clear from context
- Variable type is completely dynamic
- No declarations – first assignment creates
- Available types:
  - Primitive (int, long, boolean, …)
  - Sequences (list, tuple, dictionary)
  - Classes
- All types can be used as base class for inheritance

Example 1 – Primitive Types

```
count = 10
long_num = 958585858585858585
count += long_num

str1 = 'normal string'
str2 = “another normal string“
str3 = """long string that spans multiple lines,
        all white-space is preserved.“““
print str1 + str2 + str3

some_float = 111.11 + 1e10

print “count=%s, float=%5.2f: %s“ \
      % (count, some_float, str3)
```

Example 2 - Sequences

```
tuple1 = (1, 3, 'auto', 1.1)
list1 = [None, None, None, 'fourth']
dictionary1 = {'a':1, 'b':2, None:0, '':0}
mix_all = [(1, 2), (0, 0), None, {0:1, 1:0}]

for i in (0, 1, 2, 3, 4):
for elem in tuple1:
for i,elem in enumerate(tuple1):
for key in dictionary1:
for key,value in dictionary1.iteritems():

power2 = [i**2 for i in range(10)]
```

Classes and Exceptions

- Single rooted class hierarchy *(object)*
- All methods are virtual (dynamic name lookup)
- No *„public/protected/private“* modifiers, but:
  - Names '_xxx' are like *protected*
  - Names '__yyy' are like *private*
- Multiple inheritance, meta-classes, introspection
- Operator overloading

Exceptions are descendants of *Exception*

- *try-except-finally-else* mechanism

Example 3 – Classes

```
from exceptions import StandardError
from time import time

class Time_Stamp_Error(StandardError):
    """Exception that remembers the time.
    The time is stored in '_time' as float number.
    """

    def __init__(self, description=None):
        StandardError.__init__(self, description)
        self._store_time()

    def _store_time(self):
        self._time = time()

    def get_time(self):
        return self._time
```

Example 4 – Object's Life

```
from exceptions import *
from time import time
import psim

print t         # error
t = time()      # ok
del t
print t         # error

def set_global_t(value):
    global t
    t = value

set_global_t(psim.Scheduler())
print t         # ok

raise StandardError, “End of example”
```

Some Available modules

- Internet:
  - `cgi, Cookie, email, htmllib, imaplib`
- Interpret access, code evaluation:
  - `gc, compiler, profile, inspect, parser, pydoc, pdb, dis`
- General modules:
  - `gzip, logging, math, pickle, re, random, zipfile, sched, thread, xmllib, gdbm, bsddb, string, socket`
- Operating systém services:
  - `os, msvcrt, posix, tty,_winreg`
- User interface:
  - `tkinter, EasyDialogs, gl, curses`

The P-Sim Simulation Tool

- What changed since C-Sim
- What does it offer
- How to use it

Basic Features of P-Sim

- Mostly the same as C-Sim or J-Sim:
  - Objects: *process, queue, scheduler, semaphore*
  - Operations: *hold, passivate, activate, cancel*
  - Thread based implementation
- New features (improvements):
  - Multiple simulation worlds
  - Configurable multi-level logging system
  - Self-test of module functionality
- Only 852 lines (498 without comments)
  - C-Sim (thread version) 3496 lines (2756)
  - J-Sim (without GUI) 3902 lines (2615)

Available Classes of psim Module

- Exceptions:
  - `PSim_Error`
  - `Process_Error`
  - `Scheduler_Error`
- Support (optional) classes:
  - `Queue`
  - `Message`
  - `Semaphore`
- Core classes:
  - `Process`
  - `Scheduler`

Example 1 – A Process

```
from psim import Process, Scheduler
from random import expovariate, uniform

class Source(Process):
    def __init__(self, rate, queue):
        Process.__init__(self, scheduler, 'source')
        self.rate = rate
        self.queue = queue

    def _life(self):
        while not self._terminated:
            trans = Transaction(self.time())
            self.queue.append(trans)
            if self.queue.server.idle():
                self.queue.server.activate()
            self._hold(expovariate(self.rate))
```

Example 2 – Run the Simulation

```
import random
import psim

random.seed(1333)
scheduler = psim.Scheduler()

queues = 16 * [psim.Queue()]
processes = []
for i,queue in enumerate(queues):
    proc = Source(scheduler, 1000000, queue)
    proc.activate()
    processes.append(proc)

step = scheduler.step
while step():
    pass
```

Example 3 – Unit Self-Test (1/2)

```
class PSim_Test(unittest.TestCase):

    def test_core(self):
        """Test P-Sim Core"""

        scheduler = Scheduler()
        p1 = Process_Passivate(scheduler)
        p2 = Process_Hold(scheduler, p1)
        p1.activate()
        self.assert_(scheduler.step())
        self.assert_(scheduler.step())
        self.assert_(not scheduler.run())
        self.assert_(not scheduler.run())
        self.assert_(not scheduler.step())
        self.assertEqual(scheduler.time(), 1000)
```

Example 3 – Unit Self-Test (2/2)

```
if __name__ == '__main__':
    print 'Testing P-Sim module'
    suite = unittest.TestSuite()
    suite.addTest(unittest.makeSuite(PSim_Test))
    unittest.TextTestRunner().run(suite)
```

What Shall Come Next?

- Alternative process switching (generators):
  - More effective, easier implementation
  - Similar restrictions to long-jmp C-Sim
  - No return to scheduler after every switch
- Alternative message delivery method
  - With/without queue
  - Blocking/non-blocking receive and delivery
- Evaluate a possible native C implementation
  - Efficient
  - Possibly compatible with C-Sim

Useful On-Line Links

- http://www.python.org, *Python's Home page*
- http://ibiblio.org/obp/thinkCS/python.php, David Beazley, University of Chicago, *Python in science and teaching.*
- http://www.lyra.org/greg/CaseStudy, Greg Stein, *Python project case study*.
- http://www.linuxjournal.com/article.php?sid=3882, Eric S. Raymond, *Why use Python.*
- http://www.python.org/doc/essays/comparisons.html, Comparison of Python to other languages.

- http://www.p-sim.zcu.cz, *P-Sim home page.*