Intermediate Python

Rick Copeland
@rick446

http://bit.ly/nutanix-intermediate-python

Agenda

Day 1

- Review of Python syntax
- Advanced data types and functional programming
- Generators and Iterators
- Context Managers

Day 2

- Context Managers
- Testing in Python
- Python Profiling
- Multithreading and multiprocessing
- Network programming

Day 1 Agenda

- Review of Python syntax
- Advanced data types and functional programming
- Generators and Iterators

Review of Python Syntax

- Python functions
- Builtin data structures: list, dict, tuple, set, string
- Basic control structures
- Classes and Exceptions

Advanced Data Types and Functional Programming

- Collections module: namedtuple, defaultdict, ordereddict, deque
- Functional programming: map/filter/reduce, lambda, operator module
- Functional closures
- Decorators

Generators and Iterators

- Loop comprehensions
- Writing generators with yield
- The iterator protocol
- Generator expressions
- The itertools module

Day 2 Agenda

- Context Managers
- Profiling and Performance
- Testing in Python
- Multithreading and multiprocessing
- Network programming

Context Managers

- Use cases: nested operations
  - file: open/close
  - mutex: lock/unlock
  - xml: <tag> ... </tag>
- Old way: “try:... finally:...”
- New way: “with:...”

Profiling and Performance

- Micro-benchmarks with timeit
- cProfile and PStats
- Instrumenting high-performance code

Testing

- Unit versus integration tests
- Test-driven development
- Unit testing with unittest
- Using nose to discover tests
- Using coverage
- Mocking complex objects for better unit testing

Unit Tests

- Test isolated functionality (i.e. a single function or method)
- Test one specific code path
- Must be *fast*
- Interactions with other services are stubbed or mocked

Integration Tests

- Test is focused on the interaction between multiple units
- May be slower than unit tests
- May interact with other services

Functional Tests

- Often a subset of integration tests
- Focus is on testing the *function* rather than the *implementation* of a module

Test-Driven Development

- Expected functionality is described by a failing (ideally unit-)test
- Code is updated to make test pass (and to not make any existing tests fail)

Threading

- Global interpreter lock (GIL)
- Threads & Timers
- Locks & Semaphores
- Conditions & Events

Threading: the GIL

- Only one *Python* thread active at a time
- C libraries can release the GIL
  - I/O libraries, NumPy, etc.
- Python threads are *real OS threads*
  - "Interesting" behavior on multicore systems

Threads and Timers

- threading.Thread
  - target - Python function to call
  - args, kwargs - arguments to function
  - can also subclass & override run()
- threading.Timer
  - Simple subclass that sleeps and then runs its target

Threading Exercise

- Write a function print_time() that logs the current time each second
- Write a program that starts the print_time() function in a thread, sleeps for 10s, and then exits (use setDaemon())

Thread synchronization

- Lock & RLock (mutual exclusion)
- Semaphore (atomic counter)
- Condition
- Event
- Queue

Threading Exercise

- Write a log() function that prints a message atomically *without* using the logging module

Multiprocessing

- Based on Threading
- No GIL
- Requires “module” programming, even in main script

Multiprocess Synchronization

- Lock, Condition, Semaphore, Event
- Queue & Pipe
- Shared Memory

Multiprocessing Exercise

- Write a function print_time() that logs the current time each second
- Write a program that starts the print_time() function in a process, sleeps for 10s, and then exits (use terminate())

Network Programming

- Review of network programming concepts and protocol layers
- Fetching web resources with urllib/urllib2
- Sending email with smtplib
- sockets for low(er) level programming
- Creating a simple JSON-REST client

Network Layers (OSI)

OSI	Protocol	Address
Application	?	?
Presentation	?	?
Session	?	?
Transport	TCP / UDP	Port
Network	IP	IP
Data Link	802.x	MAC
Physical	DSL	?

- Most application programming done against the TCP layer
- Need IP address and port to build a client or a server

Socket Programming (TCP)

Client

- "connect" a socket to a port
- communicate over the connected socket

Server

- "bind" socket to a port & "listen"
- "accept" a connection, yielding a new socket
- communicate new socket

Socket Programming (UDP)

- Connectionless
- Should bind to *some* port
- No separate “connected” socket
- “sendto” IP addr/port
- “recvfrom” (specifies IP addr/port)