Fast Track to Python

Rick Copeland
@rick446

Getting to Know One Another

- Programming language background?
- What do you day-to-day at your job?
- What is one thing you want to be able to do after this training?

Day 1 Agenda

- Introduction and installation
- Basic types and control structures
- Dictionaries and exceptions
- Built-in functions and file I/O
- Standard library: os, sys, math, pdb
- String processing and regular expressions

What are Python's Distinguishing Features?

- Clear syntax
- Dynamic typing
- Object-oriented
- First-class functions
- Full modularity
- Extensive standard and third-party libraries

Installing Python

- Using Linux?
  - It's probably already there
- Installers for Mac and Windows available at http://www.python.org/download/
- Using version 2.7.3
  - Not the latest, but still the most widely used
- I'll assume Ubuntu 12.04 for consistency

Basic Python Syntax

- Hello, Python! and functions
- Basic Types: int, float, long, complex
- Variables and Arithmetic
- Strings, Lists, and Tuples
- Basic control structures
- Dicts (hash tables)
- Exceptions

Useful builtins

- zip & enumerate - sequence manipulation
- eval - dynamic expression evaluation
- dir & help - introspection
- chr & ord - string / integer conversion
- map & filter - simple sequence operations
- sum, max, min, & len - simple aggregations
- repr - “Pythonic” string representation

Useful builtins

- str, unicode - construct a string/unicode
- int(value, base=10) - convert to int
- float(value) - convert to float
- list(sequence), tuple(sequence),
- dict(sequence), dict(a=1, b=2,...)

File I/O

- open() builtin
- different file modes
- file as a sequence (for line in file:...)

Using Python Modules

- The import statement
- sys - inspect the running process
- os - inspect the operating system
- math - floating-point math functions
- time and datetime
- files and the StringIO module

String Processing

- String interpolation
- String methods
- String templates
- Regular expressions

Package Layout

- Building your own modules
  - if __name__ == '__main__': ...
- Building your own packages
  - __init__.py

Day 2 Agenda

- Functions in Python
- Standard library: Logging module
- Object-oriented programming
- Decorators
- Generators and iterators
- Context managers

Functions in Python

- def and lambda
- Argument-passing, *args, and **kwargs
- Recursion
- Higher-order functions

Logging module

- Modeled after Log4J
- Loggers, Handlers, and Formatters
- Built-in Logging Handlers
- Logging Configuration: manual, dict, and file

Object-Oriented Programming

- Defining Python classes
- Method access, visibility, and conventions
- Exploring inheritance
- “Magic” methods

Decorators

- Decorator definition
- Useful decorators
- Building your own decorators

Generators and Iterators

- Writing generators
- The iterator protocol
- Loop comprehensions
- Generator expressions

Context Managers

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

Day 3 Agenda

- Virtual Environments & Packaging
- Testing
- Subprocesses
- Threading
- Multiprocessing

Virtual Environments and Packaging

- Create a virtualenv
- Python Package Index (PyPI)
- Installing packages
- Packaging your code for PyPI and virtualenv

Installing 3rd Party Packages

- Approach 1
  - download .tar.gz or .zip
  - Unpack the .tar.gz or .zip
  - Run “python setup.py install”
- Approach 2 (if you have pip or easy_install)
  - Run “pip *package_name*”

So where do you get pip from?

- Recommended: Use within virtualenv
  - https://raw.github.com/pypa/virtualenv/master/virtualenv.py
    - python virtualenv.py *name*
- Packaged for your OS
- https://raw.github.com/pypa/pip/master/contrib/get-pip.py
  - then “pip install virtualenv”

Activate the Virtualenv

- Linux/OS X
  - $ virtualenv *name*
  - $ source *name*/bin/activate
    - (*name*) $
- Windows
  - c:\> virtualenv *name*
  - c:\> *name*\Scripts\activate
  - (*name*) c:\>

System-wide Packages

- Some packages may already be installed
  - RPMs, DEBs, MSIs
  - Running “setup.py install” outside a virtualenv
- By default, these are *not* available to the virtualenv unless you explicitly allow it
  - virtualenv --system-site-packages *name*

Installing packages into the virtualenv

- Python Package Index
  - http://pypi.python.org/pypi
    - 24556 packages available
- Examples
  - pip install nosetests
  - pip install mock
  - pip install coverage

Packaging Your Own Code

- ProjectName/
  - setup.py
  - package/
    - __init__.py
      - module1.py
      - module2.py

setup.py commands

- sdist - Create a source distribution
- bdist_rpm, bdist_wininst, bdist_msi
- bdist_egg (setuptools only)
- register - Register the package with PyPI
- upload - Upload a version to PyPI
- develop (setuptools only)
  - pip install -e .

Exercise

- Register for an account on PyPI
- Create a setuptools-based package
- Register your package on PyPI
- Upload a new release (sdist)

Testing

- unittest
- Using nose to discover tests
- Using coverage
- Mocking complex objects for better unit testing

Exercise

- Create a test suite for your telephone directory class

Subprocesses

- call, check_call, check_output
- Popen

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())

More Modules and Builtins

- Sorting and reversing
- Accessing URLs
- Serialization and deserialization
- Filename matching
- Output formatting
- Random number generation