Intermediate to Python

Rick Copeland
@rick446

Day 1 Agenda

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

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

Context Managers

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

Day 2 Agenda

- Testing in Python
- Introducing logging
- Numerical and scientific Python: NumPy and SciPy
- Pandas and data integration

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)

Logging module

- Why log?
- Loggers, Handlers, and Formatters
- Built-in Logging Handlers
- Logging Configuration: manual, dict, and file

NumPy and SciPy

- NumPy arrays & ufuncs
- Vector and matrix math in NumPy
- SciPy vector functions
- SciPy weave for C/C++ acceleration

Pandas and Data Integration

- Data frames
- Aggregation and grouping of data
- Reshaping, transforming, and cleaning data
- Scraping data: web API, parsing html and XML, JSON

Day 3 Agenda

- Multithreading and multiprocessing
- Network programming
- ~~Introduction to web application development using Django~~
- Introduction to mini-apps using Flask

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)

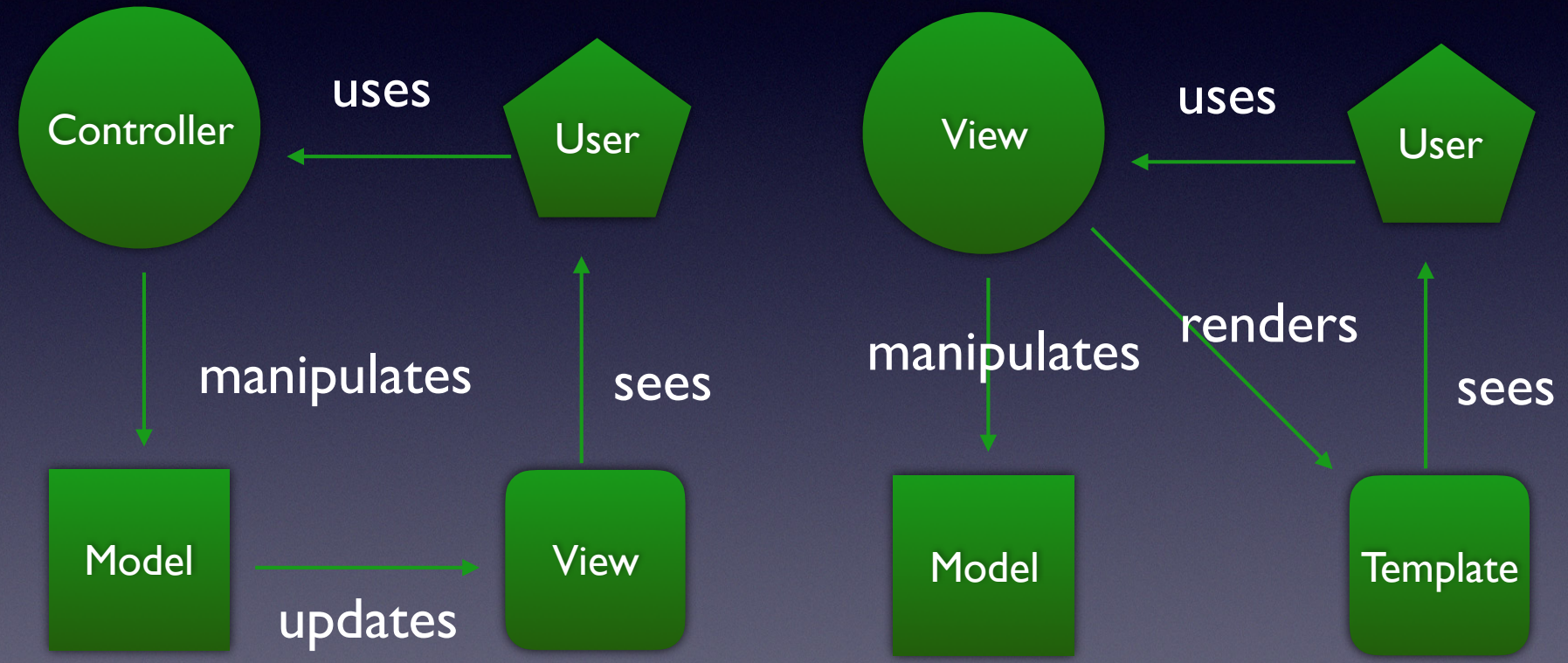
Some fun with Flask

- Key/value store (rest_api.py)
- SSQ (super-simple Queue)
- GraphIT (use Pandas to graph arbitrary CSVs)

Introduction to Django

- Model-view-template versus model-view-controller
- Models
- Admin
- URLs and Views
- Templates

Model-View-Template



Model-View-Controller

Model-View-Template

Step by Step Repo

https://bitbucket.org/rick446/django-step-by-step

```
git clone https://bitbucket.org/rick446/django-step-by-step.git
```

Getting Started

```
$ mkdir 012-Django
$ cd 012-Django
$ django-admin startproject IntermediatePython .
$ find .
.
./IntermediatePython
./IntermediatePython/__init__.py
./IntermediatePython/settings.py
./IntermediatePython/urls.py
./IntermediatePython/wsgi.py
./manage.py
```

Create a database

```
$ python manage.py migrate

Operations to perform:
  Apply all migrations: admin, contenttypes, auth, sessions
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
```

Run the server

```
$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
April 13, 2016 - 15:44:02
Django version 1.9.5, using settings 'IntermediatePython.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.  Applying auth.
0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
```

Create an app

```
$ python manage.py startapp blog
```

- Update settings.py to include new app
- Create a model

Update the database

```
$ python manage.py makemigrations blog
Migrations for 'blog':
  0001_initial.py:
    - Create model Post

$ python manage.py migrate blog
Operations to perform:
  Apply all migrations: blog
Running migrations:
  Rendering model states... DONE
  Applying blog.0001_initial... OK
```

Using the Admin

- Update admin.py
- python manage.py createsuperuser
- python manage.py runserver
- visit http://localhost:8000/admin/
- Play with the admin interface

URLs and Views

- Update urls.py and blog/urls.py
- Add a simple view

Templates

```
$ mkdir -p blog/templates/blog
$ touch blog/templates/blog/post_list.html
```

- Run server, check that we don't have an error
- Populate template with test data

Django ORM and Querying

```
$ python manage.py shell
Python 2.7.11 (default, Mar  2 2016, 12:07:11)
Type "copyright", "credits" or "license" for more information.

IPython 4.1.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra
details.

In [1]: from blog.models import Post

In [2]: Post.objects.all()
Out[2]: [<Post: My Title>, <Post: Second Post>]
```

Digression: Django Extensions and Python Notebook

```
$ pip install django-extensions
…
$ python manage.py shell_plus --notebook
```

Updating our view and template

- Update views.py
- Update post_list.html

Updating our view and template

- Update views.py
- Update post_list.html

Finishing touches

- Using Django template extensions
- Building a post_detail view
- Adding forms

What's Next

- Django ORM and query language
- Django admin customization
- More complex Django templates
- Authentication and Authorization