Intro to Python and Django

Ken Cochrane

Welcome

- Introduction
- Python tutorial
- Django tutorial
- Questions

Warning

- These were slides I made for a presentation
- Viewed by themselves without the talk might make some of the slides less clear.

Intro

- About Ken
- Django Maine
- Sponsors
- Food
- Raffle

Ken Cochrane

- Twitter: @KenCochrane
- Blog: http://KenCochrane.net
- Work: http://dotCloud.com
- Using python/django for about 5 years

DjangoMaine

- Started over a year ago
- Meets once a month
- http://www.DjangoMaine.com

Sponsors

- Base36 - Providing the food + Swag
- dotCloud - Swag for raffle

Questions

- How many people have used python before?

- How many people have used Django before?

Python

Python

- Created 1991
- Guido van Rossum
- http://python.org
- Current versions: 2.7.3, 3.3.0

What is python?

- General purpose, high-level scripting language
- Clear syntax and is expressive
- Batteries included
- Supports object oriented and functional programming

Implementations

- CPython (c) (most common)
- PyPy (python)
- Jython (Java)
- IronPython (.NET)

Python 2 or 3?

- Python 2 is the status quo. (use 2.7.x)
- Python 3.x is the present and the future
- If you can use 3, do it. If not use 2.7.x with 3 in mind.

Interpreters

- **python** (comes standard)
- **Ipython** (http://ipython.org)
- **Bpython** (http://bpython-interpreter.org)

Code Formatting

- Formatting matters
- No useless curly braces
- No tabs, all spaces
- use 4 spaces instead of 1 tab
- Tip: Turn on hidden symbols in editor

Editors

- TextMate (OS X)
- SublimeText
- TextPad, notepad++(windows)
- Vi, Emacs
- many many more

IDEs

- PyCharm
- Komodo
- Wing IDE
- many more

Hello Name

Simple python example

```
>>> def hello(name):
...     print("Hello {0}".format(name))
...
>>> hello('ken')
Hello ken
```

Common DataTypes

- String
- int
- float
- long
- boolean
- dict
- list
- set

Data type Examples

```
>>> string_variable = "string"
>>> int_variable = 24
>>> float_variable = 2.4
>>> list_variable = [1, 2, 3, 4]
>>> dict_variable = {'a':1, 2:'b', 'c':'c'}
>>> tuple_variable = (1, 2)
>>> boolean_variable = True
```

String

```
>>> a = "String Sentence!"
>>> a
'String Sentence!'
>>> a.upper()
'STRING SENTENCE!'
>>> a.lower()
'string sentence!'
>>> a.split(" ")
['String', 'Sentence!']
```

String cont.

```
>>> a.strip()
'String Sentence!'
>>> a.replace('!','?')
'String Sentence?'
>>> a.startswith('Str')
True
>>> len(a)
16
>>> b = " other sentence"
>>> a + b
'String Sentence! other sentence'
```

String formatting

```
>>> "Cat in the %s" % ('hat')
'Cat in the hat'

>>> "Cat in the {0}".format("hat")
'Cat in the hat'

>>> "Cat in the {0} knows nothing about {1}".format("Hat", "That")
'Cat in the Hat knows nothing about That'

>>> "Cat in the {thing}".format(thing='hat')
'Cat in the hat'

>>> "Cat in the {thing}, where is my {thing}".format(thing='hat')
'Cat in the hat, where is my hat'
```

Numbers and Math

- + plus
- - minus
- / slash
- * asterisk
- % percent
- < less-than
- > greater-than
- <= less-than or equal
- >= greater-than or equal
- == equal
- != not equal

Math Examples

```
>>> a = 5
>>> a
5
>>> a + 5
10
>>> a - 6
-1
>>> a = a + 5
>>> a
10
>>> a += 1
```

```
>>> a
11
>>> a / 2
5
>>> a / 2.0
5.5
>>> 5 % 2
1
>>> 5 / 2
2
```

LISTS

```
>>> l = [1, 2, 3]
>>> l.append(4)
>>> l
[1, 2, 3, 4]
>>> l.pop()
4
>>> l
[1, 2, 3]
>>> l.reverse()
>>> l
[3, 2, 1]
>>> l.count(1)
1
```

```
>>> l.remove(2)
>>> l
[3, 1]
>>> l.sort()
>>> l
[1, 3]
>>> l2 = ['a', 'b']
>>> l + l2
[1, 3, 'a', 'b']
```

Dictionaries

```
>>> d = {'a':1, 'b':2, 3:3, 'c':'c'}
>>> d
{'a': 1, 3: 3, 'c': 'c', 'b': 2}
>>> 'a' in d
True
>>> 'z' in d
False
>>> d['d'] = 'd'
>>> d
{'a': 1, 3: 3, 'c': 'c', 'b': 2, 'd': 'd'}
>>> e = {'e':'e'}
>>> d.update(e)
>>> d
{'a': 1, 'c': 'c', 3: 3, 'e': 'e', 'd': 'd', 'b': 2}
```

Conditional Logic

- < less-than
- > greater-than
- <= less-than or equal
- >= greater-than or equal
- == equal
- != not equal
- “is” object identity
- “is not” negated object identity

If statements

Example of a simple if else statement

```
>>> a = 8
>>> if a > 10:
...     print("a is bigger than 10")
... elif a > 5:
...     print("a is bigger than 5")
... else:
...     print("a is <= 5")
...
a is bigger than 5
```

For Loop

```
# loop 1 to 3
>>> for x in range(1, 4):
...     print(x)
1
2
3
>>> a = [1, 2, 3]
>>> for x in a:
...     print(x)
1
2
3
```

While Loop

```
>>> count = 0
>>> while count < 4:
...     print(count)
...     count += 1
0
1
2
3
```

Functions

```
>>> def hello(name='fish'):
...     print("hello {0}".format(name))
...
...
>>> hello()
hello fish
>>> hello("ken")
hello ken
>>> hello(name="kenny")
hello kenny
```

Functions cont.

```
>>> def hello(first, last='Smith'):
...     print("hello {0} {1}".format(first, last))
...
>>> hello("ken", "cochrane")
hello ken cochrane
>>> hello("ken")
hello ken Smith

>>> hello("ken", last="cochrane")
hello ken cochrane

>>> hello()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: hello() takes at least 1 argument (0 given)

>>> hello(last="cochrane", "ken")
  File "<input>", line 1
SyntaxError: non-keyword arg after keyword arg
```

Functions Cont. 2

```
>>> def adder(a, b):
...     return a + b
...
>>> adder(1, 2)
3
>>> def joiner(a, b):
...     return a, b
...
>>> joiner(1, 2)
(1, 2)
```

Classes

```
>>> class myclass(object):
...
...     def __init__(self, first, last):
...         self.first = first
...         self.last = last
...         self.status = "NEW"
...
...     def __repr__(self):
...         return u"{0} class".format(self.full_name)
...
...     def change_status(self, status):
...         self.status = status
...
...     @property
...     def full_name(self):
...         return “{0} {1}”.format(self.first, self.last)
...
```

Classes Cont.

```
>>> my = myclass("joe", "smith")
>>> my
joe smith class

>>> my.first
'joe'

>>> my.last
'smith'

>>> my.full_name
'joe smith'

>>> my.status
'NEW'

>>> my.change_status("OLD")
>>> my.status
'OLD'
```

Exceptions

```
>>> def div(a, b):
...     try:
...         return a / b
...     except ZeroDivisionError as exp:
...         return 0
...
>>> div(2,1)
2
>>> div(2,0)
0
```

Exceptions Cont.

```
>>> def div(a, b):
...     try:
...         return a / b
...     except ZeroDivisionError as exc:
...         raise Exception("Wha? Div by zero yo!")
...
>>> div(2,0)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "<input>", line 5, in div
Exception: Wha? Div by zero yo!
```

Python resources

That was a very quick overview, look at these resources to learn more about python

- http://python.org
- http://LearnPythonTheHardway.org
- http://www.CodeCademy.com

django

Django Tutorial

Django Tutorial

- Guided tour using the Official Django tutorial
- https://docs.djangoproject.com/en/1.4/intro/tutorial01/
- There isn't enough time to go over everything, so I skipped some parts.

Before we get started

- Django Intro
- Installing python packages
- Install Django

What is django?

- Django is a High-level Python web framework that encourages rapid development and clean, pragmatic design.
- MVT - Model , View, Template

Who uses Django?

- Disqus
- Instagram
- Pinterest
- Mozilla
- Rdio
- Open Stack

Batteries included

- ORM
- Templates
- Admin site
- Authentication/ Authorization
- I18N
- Forms
- Validators
- Caching
- Comments
- GeoDjango
- CSRF protection
- Sites
- testing
- Dev server
- Elegant URL design

Installing Django

- Today's Requirements:
  - Python 2.6.x+ (ideally 2.7.x)
  - distribute or setup tools
  - pip
  - virtualenv
  - virtualenv wrapper

Pip + Distribute

- Tools for installing and managing python packages
- Pip requires Distribute

Install Distribute

```
$ curl -O http://python-distribute.org/
distribute_setup.py

$ python distribute_setup.py
```

Or if you have easy_install you can run this:

```
$ easy_install distribute
```

Install PIP

```
$ curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py

$ [sudo] python get-pip.py
```

Virtualenv

- Python Virtual Environments make it easier for you to manage multiple projects' dependencies on your machine at once
- http://www.virtualenv.org

Install virtualenv

```
$ [sudo] pip install virtualenv
```

Virtualenv wrapper

- Dealing with all of your virtualenv's can become a pain.
- Install Virtualenv Wrapper to help manage them.
- http://virtualenvwrapper.readthedocs.org

Install Virtualenv Wrapper

```
$ pip install virtualenvwrapper
```

Edit your shell startup script to include

```
source /usr/local/bin/virtualenvwrapper.sh
```

Create a virtualenv

```
$ mkvirtualenv tutorial

# it should automatically switch, but if not run
$ workon tutorial
```

install Django

```
$ pip install django==1.4.3
```

DJANGO-ADMIN.PY

- django-admin.py is Django’s command-line utility for admin tasks
- Use it to start new projects
- https://docs.djangoproject.com/en/1.4/ref/django-admin/

Create a new project

Create a projects directory somewhere on your computer

```
$ mkdir -p ~/projects

$ cd ~/projects

$ django-admin.py startproject mysite
```

Project directory

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Verify project

- Let’s make sure your django project is configured correctly.
- Best to do this now before you make any changes.
- $ python manage.py runserver
- Should have started with no errors.
- I’ll come back to runserver a little later

SETTINGS.PY

- This is where you configure your Django application
- Lots of sane defaults
- Many more advanced ways of configuring settings, avoid for now

Settings.py Cont.

- Common things you need to change
    - DEBUG
    - ADMINS
    - DATABASES
    - TIME_ZONE
    - MEDIA_ROOT and STATIC_ROOT
    - INSTALLED_APPS
    - MIDDLEWARE
    - LOGGING

Settings.py Cont. 2

- For simplicity we will mostly keep the settings the way they are for now.
- Let's use SQLite for a database
- Change DATABASES.default.ENGINE to 'django.db.backends.sqlite3'
- Change DATABASES.default.NAME to 'tutorial.db'

Syncdb Command

- The syncdb command looks at the INSTALLED_APPS setting and creates the needed database tables for the apps listed
- Creates superuser if one doesn't exist.
- $ python manage.py syncdb

Add an App

- A Django project consists of Django apps
- $ python manage.py startapp polls

POLLS APP LAYOUT

```
polls/
    __init__.py
    models.py
    tests.py
    views.py
```

Django Models

- A Model is a normal python file, usually named models.py
- Contains the essential fields and behaviors of the data you're storing.
- Generally, each model maps to a single database table

Django Models Cont.

- Each model is a Python class that subclasses django.db.models.Model
- Each attribute of the model represents a database field.
- With all of this, Django gives you an automatically-generated database-access API.
- Creates the initial database tables for you.

Model Fields

- AutoField
- BigIntegerField
- BooleanField
- CharField
- CommaSeperatedIntegerField
- DateField
- DateTimeField
- DecimalField
- EmailField
- FileField
- FilePathField
- FloatField
- ImageField
- IntegerField
- IPAddressField
- GenericIPAddressField
- NullBooleanField
- PositiveIntegerField
- SlugField
- SmallIntegerField
- TextField
- TimeField
- URLField

Model field options

- null
- blank
- choices
- db_column
- db_index
- db_tablespace
- default
- editable
- error_messages
- help_text
- primary_key
- unique
- unique_for_date
- unique_for_month
- unique_for_year
- verbose_name
- validators

Model Relationships

- Foreign Key
- Many To Many
- One to One

Polls/models.py

Add the following to your polls/models.py

```
from django.db import models

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
```

Activating models

- Adding that small bit of code allows Django to do the following:
  - Builds the “Create table” sql statement
  - Creates a python db api for accessing those db tables

Activating Models Cont.

- In order for Django to know about our new app, we need to add our app to our INSTALLED_APPS setting in our Settings.py

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    # 'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
    'polls', # <---- We added it here
)
```

Generated Model SQL

- If you want to know what SQL will be run when you kick off syncdb, you can use the “sql” command
- $ python manage.py sql polls

```
BEGIN;
CREATE TABLE "polls_poll" (
    "id" serial NOT NULL PRIMARY KEY,
    "question" varchar(200) NOT NULL,
    "pub_date" timestamp with time zone NOT NULL
);
CREATE TABLE "polls_choice" (
    "id" serial NOT NULL PRIMARY KEY,
    "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id") DEFERRABLE INITIALLY
DEFERRED,
    "choice" varchar(200) NOT NULL,
    "votes" integer NOT NULL
);
COMMIT;
```

Other Helpful commands

- **validate** - Checks for any errors in your models
- **sqlcustom** - Outputs any custom sql statements that are defined for the app
- **sqlclear** - Outputs the necessary drop table statements for the app.
- **sqlindexes** - Outputs the create index statements
- **sqlall** - Outputs all sql (creates, custom, index, etc) for these models

Running syncdb

- Run syncdb to create the new tables for your polls app
- $ python manage.py syncdb

Django Shell

- One of the cooler features of Django is the shell
- $ python manage.py shell
- Starts an interactive python shell that lets you play around with your app form the command line.
- Install Bpython or Ipython for more features

Fire up the shell

- $ python manage.py shell

```
# Import the model classes we just wrote.
>>> from polls.models import Poll, Choice
>>> Poll.objects.all()
[]

# No polls are in the system yet.
>>> from django.utils import timezone
>>> p = Poll(question="What's new?", pub_date=timezone.now())

# Save the object into the database. You have to call save()
explicitly.
>>> p.save()

>>> p.id
1
```

Shell cont.

```
# Access database columns via Python attributes.
>>> p.question
"What's new?"

>>> p.pub_date
datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=<UTC>)

# Change values by changing the attributes, then calling save().
>>> p.question = "What's up?"
>>> p.save()

# objects.all() displays all the polls in the database.
>>> Poll.objects.all()
[<Poll: Poll object>]

# count all of objects in database
>>> Poll.objects.count()
1
```

Models Cont.

- Human readable object names

```
class Poll(models.Model):
    # ...
    def __unicode__(self):
        return self.question

class Choice(models.Model):
    # ...
    def __unicode__(self):
        return self.choice
```

Custom model methods

- You can add your own methods to make your live easier.

```
import datetime
from django.utils import timezone
# ...
class Poll(models.Model):
    # ...
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - 
datetime.timedelta(days=1)
```

Django Admin site

- Another powerful part of Django is the automatic admin interface.
- It reads the metadata in your model to provide a powerful, production ready interface
- It is disabled by default

ACTIVATING DJANGO ADMIN

- settings.py
  - Uncomment “django.contrib.admin” in INSTALLED_APPS setting.
- urls.py
  - Uncomment the 3 lines shown in the comments, to be for admin.
- Run syncdb to create table

URLS.PY

- The urls.py is a way to map application urls to django application views.
- We will go into more detail in a little bit

Activating Django Admin Cont.

- mysite/urls.py when finished

```
from django.conf.urls import patterns, include, url

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', '{{ project_name }}.views.home', name='home'),
    # url(r'^{{ project_name }}/', include('{{ project_name }}.foo.urls')),

    # Uncomment the admin/doc line below to enable admin documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    url(r'^admin/', include(admin.site.urls)),
)
```

Running django admin

- $ python manage.py runserver
- open browser, point to
  - http://localhost:8000
- Login with your admin user/password you created during first 'syncdb'

Configuring Django Admin

- Django admin needs to know about your application and it's models before they will work in the admin
- You do this by creating a <app_name>/admin.py file and configure your models

POLL/ADMIN.PY

- There are lots of different ways to configure your models to work in the admin, but we will just use the normal way for now.
- Create a polls/admin.py and add the following

```
from polls.models import Poll, Choice

from django.contrib import admin

admin.site.register(Poll)
admin.site.register(Choice)
```

Django Admin advanced

- There are a lot of different ways to customize the django admin, we don’t have time today to go into all of them.
- Follow the tutorial online to find out more.
- https://docs.djangoproject.com/en/1.4/intro/tutorial02/

Views

- 3 types of views
  - Generic views
  - Function based views
  - Class based views

Function based views

- The view is where the glue code lives. It ties your request, model, and template together.
- Here is a very simple view

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world.")
```

Views Cont.

Add this to your mysite/views.py

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the index.")

def detail(request, poll_id):
    return HttpResponse("You're looking at poll %s." % poll_id)

def results(request, poll_id):
    return HttpResponse("results of poll %s." % poll_id)

def vote(request, poll_id):
    return HttpResponse("You're voting on poll %s." % poll_id)
```

URLS.PY

- The urls.py allows you to decouple your business logic with your urls. You can change the url later on without changing the code.
- Uses regular expressions to match urls to views
- Allows you to create nice looking urls

MYSITE/URLS.PY

Change the mysite/urls.py to look like this

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^polls/$', 'polls.views.index'),
    url(r'^polls/(?P<poll_id>\d+)/$', 'polls.views.detail'),
    url(r'^polls/(?P<poll_id>\d+)/results/$',
'polls.views.results'),
    url(r'^polls/(?P<poll_id>\d+)/vote/$', 'polls.views.vote'),
    url(r'^admin/', include(admin.site.urls)),
)
```

URLS TO VIEWS

Notice how the variable in the url matches the parameter in the view

```
  url(r'^polls/(?P<poll_id>\d+)/$', 'polls.views.detail'),
```

```
def detail(request, poll_id):
    return HttpResponse("Poll %s." % poll_id)
```

More views

- Those views didn't do much, lets do some more.
- Change the index view to this.

```
from polls.models import Poll
from django.http import HttpResponse

def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    output = ', '.join([p.question for p in latest_poll_list])
    return HttpResponse(output)
```

Django Templates

- Designed to strike a balance between power and ease.
- Clear line between logic and presentation, no code allowed
- Has a set of built in tags and filter
- Ability to write custom tags and filters
- Block based to support template inheritance

Common template tags

- block
- for
- if
- include
- url
- extends

Common template filters

- date
- default
- linebreaksbr
- pluralize
- safe
- truncatewords
- upper
- wordwrap
- wordcount
- yesno
- length

Template example

```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock title %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock content %}
```

Lets add templates to our view

- Our poll view is pretty basic, lets add a template to make it better.
- create a ‘templates/polls’ directory under polls app.
- create a file called index.html

INDEX.HTML

Put this in your index.html

```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li><a href="/polls/{{ poll.id }}/">{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

INDEX VIEW

Now that we have a template we need to change our view, to use it.

```
from django.template import Context, loader
from polls.models import Poll
from django.http import HttpResponse

def index(request):

    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]

    t = loader.get_template('polls/index.html')

    c = Context({
        'latest_poll_list': latest_poll_list,
    })

    return HttpResponse(t.render(c))
```

Django Forms

- Forms are the glue that helps convert HTML form data into something useful for views
- Also provides validation
- 2 types of Django Forms
  - Model forms
  - Regular forms

Forms

- The Form classes look a lot like Models. They have fields with attributes, and widgets.
- Widgets tell django what type of HTML field to map it too.
- Django can create HTML forms for you from a form object.

Form example

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField()
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

Using form in a view Example

```python
from django.shortcuts import render
from django.http import HttpResponseRedirect

def contact(request):
    if request.method == 'POST': # If the form has been submitted...
        form = ContactForm(request.POST) # A form bound to the POST data
        if form.is_valid(): # All validation rules pass
            # Process the data in form.cleaned_data
            # ...
            return HttpResponseRedirect('/thanks/') # Redirect after POST
    else:
        form = ContactForm() # An unbound form

    return render(request, 'contact.html', {
        'form': form,
    })
```

Display a form in a template example

```
<form action="/contact/" method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <input type="submit" value="Submit" />
</form>
```

Becomes

```
<form action="/contact/" method="post">
   <p><label for="id_subject">Subject:</label>
      <input id="id_subject" type="text" name="subject" maxlength="100" /></p>
   <p><label for="id_message">Message:</label>
      <input type="text" name="message" id="id_message" /></p>
   <p><label for="id_sender">Sender:</label>
      <input type="text" name="sender" id="id_sender" /></p>
   <p><label for="id_cc_myself">Cc myself:</label>
      <input type="checkbox" name="cc_myself" id="id_cc_myself" /></p>
   <input type="submit" value="Submit" />
</form>
```

Model Forms

- Model Forms are just like regular forms, but they make it easier to get the form data into models.
- Instead of having to move the data from the form to the model, it is done automatically, and saved to the db when save() is called.

Model form example

```python
from django.db import models
from django.forms import ModelForm, Textarea

class Author(models.Model):
    name = models.CharField(max_length=100)
    title = models.CharField(max_length=3)
    birth_date = models.DateField(blank=True, null=True)

    def __unicode__(self):
        return self.name

class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = ('name', 'title', 'birth_date')
        widgets = {
            'name': Textarea(attrs={'cols': 80, 'rows': 20}),
        }
```

Free Django Hosting

- If you want a place you can deploy your application for free, and play around with it.
- Check out : http://www.dotCloud.com
- Django **Tutorial**: http://docs.dotcloud.com/tutorials/python/django/

Django Resources

- DjangoProject.com
- DjangoBook.com
- Two Scoops of Django ebook
- gettingstartedwithdjango.com

Too much to cover so little time

- This was just a quick overview of Django and Python, there is way, way more then this, and I apologize if I skimmed over a part you really wanted to know more about.

Thank you

- If this was helpful, or if you have any suggestions on how to make it better, please contact me on twitter (@kencochrane) or send me an email kencochrane@gmail.com