Django Unchained!

A primer on Django framework for python web development

The Goals

1) Cover enough Python to get kick started with the series.
2) Enable you to explore other Pythonic stuff on your own.
3) Understand how modern web applications are engineered.
4) The Django framework.

The slides and other relevant material will be available online.

About me

- Alumnus of this college, class of 2016.
- GSoC Mentor | GCI Mentor
- Former Software Engineer at Roadrunnr Inc.
- Currently, Mobile and User Experiences Consultant for SAP India.
- Reach me at sricharanized@gmail.com

Python Trivia

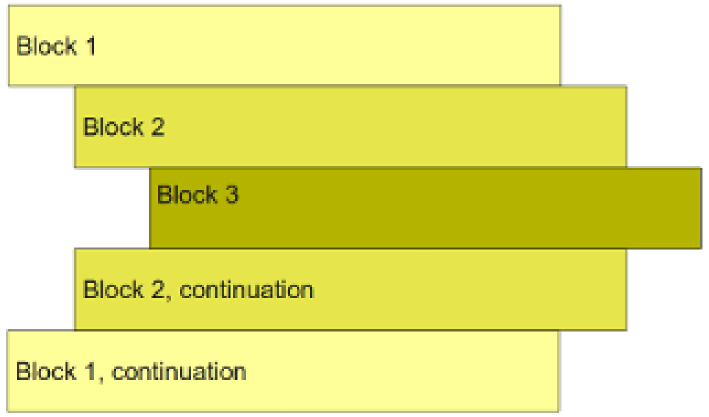
- Created by Guido Van Rossum in 1991.
- Python is an interpreted, multi-paradigm language.
- Python has **no compile-time type checking** of the source code. Python tracks the types of all values at runtime and flags code that does not make sense as it runs.
- ‘Batteries included’ in nature.

Running Python

- Interactive Interpreter
  - Just run python command in your terminal. You will be presented with an interactive interpreter.
  - Read, Eval, Print loop.
- Script Interpreter
  - Write a script in text editor, save it with .py extension. Run **python file.py** to execute.
- Console
  - Python -c command.

Python syntax and styling

- One unusual Python feature is that the whitespace indentation of a piece of code affects its meaning.
- **Everything in python is an object.**



Let’s play!

- Basic data structures.
  - Lists [ ]
  - Tuples ()
  - Dictionaries {:}
  - Sets {}
- Functions, Loops and module imports.

We’re not using the pythonic features yet. More on that later.

Running Python

- Interactive Interpreter
  - Just run python command in your terminal. You will be presented with an interactive interpreter.
  - Read, Eval, Print loop.
- Script Interpreter
  - Write a script in text editor, save it with .py extension. Run **python file.py** to execute.
- Console
  - Python -c command.

Let’s do a python primer

http://bit.do/amritapy

Diving into web applications

- How models, views, urls and templates talk to each other to render dynamic content?
- Let’s create a “sell your products online” inventory application.
- All you need is:
  - Web application architecture – Models, Controllers, Views, Templates...
  - A decent IDE/ text editor.
  - Basics of python – Loops, variables, lists, dictionaries ...

What is Django?

- A web framework. What is a web framework?
- Django as a framework comes with
  - Object Relational Mapper (ORM)
  - URL routing
  - Front-end templating
  - Form handling
  - Unit testing tools
  - A lot others...

Django is NOT

- A web server
- A single language web framework
- A collection of python modules
- A packaging tool – “Python Installer of Packages”

Make sure you have pip installed before we proceed further.

Installing Django

- http://djangoproject.com
- *pip install django==1.8*
- *Django-admin --version*

Creating our first project

- Cd to the directory you want to work with.
- *django-admin startproject firstdjango*
- Let’s explore!
- *cd firstdjango/*
- *python manage.py* – to list the available sub-commands.
- *Python manage.py runserver*

Generated files

- __init__.py – Tells django where the project folder is. Differentiate from app folders
- Manage.py – Run commands
- Firstdjango/wsgi.py – Used by the web server to run
- Firstdjango/settings.py – Configures Django
- Firstdjango/urls.py – Routes requests based on URL

Django App vs Django Project

- Within Django App is a component.
- Each App fits a specific purpose.
- Blog, Forum, Wiki, Cart, Products…
- Models.py – Data Layer, admin.py – Administrative Interface, Views.py – Control Layer, tests.py – Tests the app, migrations/ – Holds the migration files.

Our firstapp

- Cd to the project
- Python manage.py startapp firstapp
- Need to edit settings.py to add a new project.
  INSTALLED_APPS

Exploring settings.py

- INSTALLED_APPS
- TEMPLATES
- STATICFILES_DIRS
- DEBUG
- DATABASES

Inventory App – Models

- Rename firstapp to inventory
- Change the same in the settings.py
- Models create the data layer of an app
- Defines the database structure
- A model is a class inheriting from django.db.models.Model and is used to define fields as class attributes.

Inventory App.

- Store items with a title, description, and amount of stock.
- Allow administrators to create, edit, or delete items.
- Allow users to see a list of items in stock, with details.

Models.py

```
From django.db import models
class Item(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    amout = model.IntegerField()
```

Other field types

- IntegerField, DecimalField
- CharField (needs max_length), TextField, EmailField, URLField.
- FileField, ImageField.
- BooleanField, DateTimeField

Field Attribute Options

- max_length
- Null = true
- Blank = true
- Default
- choices

Migrations

- When a model is added, a migration is required.
- Migrations – Generate scripts to change the database structure.
- Adding a model
- Adding a field
- Removing a field
- Changing the attribute of a field

Running Migrations

- Python manage.py makemigrations
- Python manage.py migrate –list
- Python manage.py migrate
- db.sqlite3!

The Django Admin Site

- Admin.py

```
from django.contrib import admin
from .models import Item
class ItemAdmin(admin.ModelAdmin):
    list_display = [‘title’, ‘amount’]
admin.site.register(Item, ItemAdmin)
```

- We need to have a superuser to access admin

```
    python manage.py createsuperuser
```

The ORM queries

- Runserver
- /admin
- Add Item, Editing and deleting.
- Python Interactive Shell:

```
from inventory.models import Item
items = Item.objects.all()
item = item[0]
item.title
item.id
Item.objects.get(id=1).title
Item.objects.filter(amout=0)
Item.objects.exclude(amout=0)
Item.objects.exclude(amout=0)[0].title
```

Web Application Architecture

- URL Patterns → Views → Templates
- Models ← Views

- / → index → index.html
- /item/1/ → item_detail → item_detail.html

URL Patterns

- \d → single digit char
- \d+ → one or more digits
- ^admin/ → begins with admin/
- Suffix$ → ends with suffix
- ^$ → empty strings

Urls.py

```
From django.conf.urls import url

from inventory import views

urlpatterns = [
    url(r’^$’, views.index, name=’index’),
    url(r’^item/(?P<id>\d+)/’, views.item_detail,
    name=’item_detail’),
]
```

Views.py

- From django.http import HttpResponse

  def index(request):

      return HttpResponse(‘<p>In Index View</p>’)

  def item_detail(request, id):

      return HttpResponse(‘<p>In Item_detail with id {0}</p>’.format(id))
- Run the server

Django Views

- See the complete View file.
- Index all the items in stock with

```
items = Item.objects.exclude(amout=0)
return render(request, ‘inventory/index.html’, {
    ‘items’: items,
})
```

- Item_details gets the item instance.

Item details view

```
Try:
    item = Item.objects.get(id=id)
except Item.DoesNotExist:
    railse Http404(‘This item does not exist’)
return render(request,
‘inventory/item_detail.html’, {
    ‘item’ : item,
})
```

Templates

- Modify settings.py

  TEMPLATES → DIRS: [‘firstdjango/templates’]
- Create a directory called templates, then create an inventory directory, make index.html and item_detail.html
- Put a couple of <p> tags and see if templates are working fine.

Template Tags

- {{ variable }}
- {% tag %}
- {{ variable | filter }}

- <h3>{{ item.title }}</h3>
- {% for item in items %}
    - <li>{{ item.title }}</li>

  {% endfor %}
- {% url ‘index’ %}
- {% url ‘item_detail’ item.id %}
- <h3>{{ item.title | capfirst }}</h3>

The completed templates

- Template inheritance
- Future-proofing
- Block tag overriding

Project files are at
https://github.com/raincrash/PythonCourse

Questions