**Steps to Starting a New Django Project**

# Folder Structure

* **main\_project**
  + **apps**
    - \_\_init\_\_.py
    - **app\_name**
      * \_\_init\_\_.py
      * admin.py
      * apps.py
      * **migrations**
      * models.py
      * **static**
        + **app\_name**

**css**

**js**

**images**

* + - * **templates**
        + **app\_name**

index.html

* + - * tests.py
      * urls.py
      * views.py
  + db.sqlite3
  + **main\_project\_name**
    - \_\_init\_\_.py
    - settings.py
    - urls.py
  + manage.py

# Creating a Project

1. Launch virtual environment

myenvironments> call djangoEnv/scripts/activate

1. Create project

project\_folder> django-admin startproject project\_name

1. Make apps folder
2. Make apps\\_\_init\_\_.py
3. Launch first app

project\_name\apps> python ../manage.py startapp app\_name

1. Make apps\app\_name\\_\_init\_\_.py
2. Make apps\app\_name\urls.py
3. Make apps\app\_name\static\app\_name … css || js || images
4. Make apps\app\_name\templates\app\_name … index.html
5. Update project\_name\project\_name\settings.py to include app\_name:

‘apps.app\_name’,

1. Write project\_name\urls.py

from django.conf.urls import url, include

from django.contrib import admin

urlpatterns = [

url(r'^admin/', admin.site.urls),

url(r'^blogs/', include('apps.blogs\_app.urls')),

]

1. Write app\_name\urls.py

from django.conf.urls import url, include

from django.contrib import admin

from . import views

urlpatterns = [

url(r'^$', views.index),

url(r'new$', views.new\_blog),

url(r'(?P<blog\_number>[0-9]{2}$)', views.show\_blog\_num),

]

1. Write app\_name\views.py

from \_\_future\_\_ import unicode\_literals

from django.shortcuts import render, redirect

def index(request):

context = {

“email” : [email@email.com](mailto:email@email.com),

“name” : “user\_name” }

return render(request, "index.html", context)

1. Write app\_name\templates\app\_name\index.html, etc.
2. Run server:

Project\_name\> python manage.py runserver

# Setting Up Session and ORM Databases (Migration):

# Need to be in same directory as manage.py file

> python manage.py makemigrations

> python manage.py migrate

## Session Key Terms

request.POST

* + - Data from POST request

request.GET

* + - Data from GET request

request.method

* + - Returns the method/HTTP verb associated with the request

{% csrf\_token %}:

* + - Prevents cross-site request forgery (place it in a form on the HTML/template side of your project)

request.session:

* + - # It's a dictionary, so you can attach key/value pairs

## Useful session methods:

request.session['key']

* + - This will retrieve (get) the value stored in key

request.session['key'] = 'value'

* + - Set the value that will be stored by key

del request.session['key']

* + - Deletes a session key if it exists, throws a keyError if it doesn’t. Use along with try and except since it’s better to ask for forgiveness than permission

'key' in request.session

* + - Returns a boolean of whether a key is in session or not

{{ request.session.name }}

* + - Use dot notation (.) to access request.session keys from templates since square brackets ([]) aren’t allowed there

Django ORM Commands

# Creating an Object Relationship Model

# Inside models.py

from \_\_future\_\_ import unicode\_literals

from django.db import models

# Create your models here.

class Blog(models.Model):

name = models.CharField(max\_length=255)

desc = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add = True)

updated\_at = models.DateTimeField(auto\_now = True)

class Comment(models.Model):

comment = models.CharField(max\_length=255)

created\_at = models.DateTimeField(auto\_now\_add = True)

updated\_at = models.DateTimeField(auto\_now = True)

blog = models.ForeignKey(Blog, related\_name = "comments")

class Admin(models.Model):

first\_name = models.CharField(max\_length=255)

last\_name = models.CharField(max\_length=255)

email = models.CharField(max\_length=255)

blogs = models.ManyToManyField(Blog, related\_name = "admins")

created\_at = models.DateTimeField(auto\_now\_add = True)

updated\_at = models.DateTimeField(auto\_now = True)

## ORM Column Types

CharField(max\_length=integer)

TextField()

IntegerField(), BooleanField()

DateTimeField

This field can take two very useful optional parameters, auto\_now\_add=True, which adds the current date/time when object is created, and auto\_now=True, which automatically updates any time the object is modified.

ForeignKey, ManyToManyField, OneToOneField

Used to indicate a relationship between models (anything that would require a JOIN statement in SQL).

# Creating a new record

Blog.objects.create({{field1}}="{{value}}", {{field2}}="{{value}}", etc)

Alternative way of creating a record

b = Blog(name="Disney Blog", desc="Disney stuff")

b.name = "Disney Blog!"

b.desc = "Disney stuff!!!"

b.save()

# Basic Retrieval

Blog.objects.first()

retrieves the first record in the Blog table

Blog.objects.last()

retrieves the last record in the Blog table

Blog.objects.all()

retrieves all records in the Blog table

Blog.objects.count()

shows how many records are in the Blog table

# Updating the record

Once you obtain an object that has the record you want to modify, use save() to update the record

b = Blog.objects.first()

b.name = "CodingDojo Blog"

b.save()

# updates the blog record

# Deleting the record - use delete()

b = Blog.objects.get(id=1)

b.delete()

# Other methods to retrieve records

Blog.objects.get(id=1)

retrieves where id is 1; get only retrieves 1 record

Blog.object.filter(name="mike")

retrieves records where name is "mike"; returns multiple records

Blog.objects.exclude(name="mike")

opposite of filter; returns multiple records

Blog.objects.order\_by("created\_at")

orders by created\_date field

Blog.objects.order\_by("-created\_at")

reverses the order

Blog.objects.raw("SELECT \* FROM {{app\_name}}\_{{class/table name}}")

performs a raw SQL query

Blog.objects.first().comments.all()

grabs all comments from the first Blog

Blog.objects.get(id=15).comments.first()

grabs the first comment from Blog id = 15

Comment.objects.get(id=15).blog.name

returns the name of the blog where Comment id = 15 belongs to

# Setting Relationship

Comment.objects.create(blog=Blog.objects.get(id=1), comment="test")

create a new comment where the comment's blog points to Blog.objects.get(id=1)

# Conditions

Instead of just passing in the field name as a keyword argument to .get, .filter, or .exclude, pass the field name\_\_lookup type (that's a double underscore, also known as a "dunder")

Admin.objects.filter(first\_name\_\_startswith="S")

filters objects with first\_name that starts with "S"

Admin.objects.exclude(first\_name\_\_contains="E")

excludes objects where first\_name contains "E"

Admin.objects.filter(age\_\_gt=80)

filters objects with age greater than 80

# Combining queries

Queries can be chained together:

admin = Admin.objects.filter(last\_name\_\_contains="o")

.exclude(first\_name\_\_contains="o")

admin = Admin.objects.filter(age\_\_lt=70)

.filter(first\_name\_\_startswith="S")

If it's the same type of query, instead of being chained you can add multiple arguments to the function:

admin = Admin.objects.filter(age\_\_lt=70, first\_name\_\_startswith="S")

These are cases where the conditions are joined with AND, as in, all users younger than 70 AND whose first name starts with "S". If you want OR, as in users who are younger than 70 OR whose first\_name starts with "S", you can use | (the set union operator):

admin = Admin.objects.filter(age\_\_lt=70)  
|Admin.objects.filter(first\_name\_\_startswith="S")