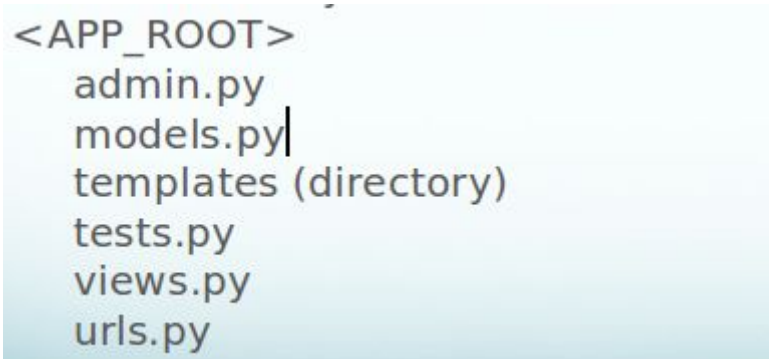
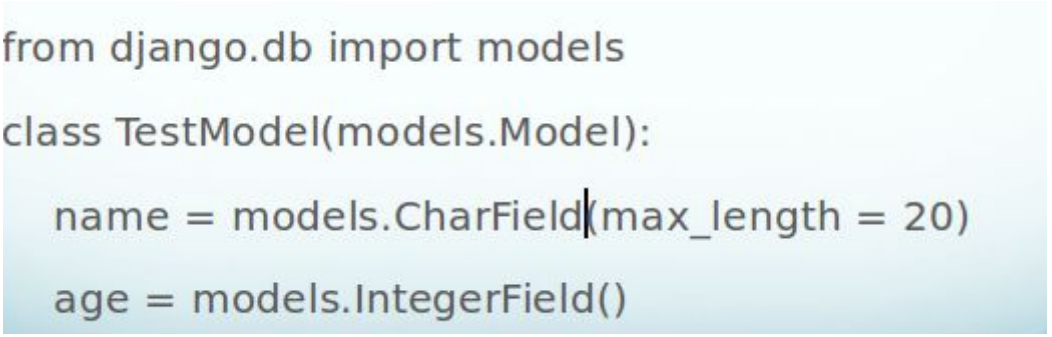
Django

1. The web framework for perfectionists with deadlines
2. MVC
3. Flexible template language that can be used to generate HTML, CSV or any other format
4. Includes ORM that supports many databases – Postgresql, MySQL, Oracle, SQLite
5. Lots of extras included – middleware, csrf protections, sessions, caching, authentication
6. Django Concepts/Best Practices
    a. DRY Principle – “Don’t Repeat Yourself”
    b. Fat models, thin views
    c. Keep logic in templates to a minimum
    d. Use small, reusable “apps” (app = python module with models, views, templates, test)
7. Django Project Layout
    a.
```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```
8. Settings.py
    a. Defines settings used by a Django application
    b. Referenced by wsgi.py to bootstrap the project loading
    c.
```
DEBUG = True
TEMPLATE_DEBUG = True
ALLOWED_HOSTS = []
# Application definition
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
)
```
9. Django Apps

a. Reusable modules
b. django-admin.py startapp <app_name>
c. Creates stub layout:
    i. 

10. Django Models
    a. How to
        i. Defined in models.py
        ii. Typically inherit from django.db.models.Model
        iii. 

        iv. Field parameters(null=True, blank = True etc)
        v. Relationships defined through special field types:
            1. models.OneToOneField(model)
            2. models.ForeignKey(model)
            3. models.ManyToManyField(model)
        vi. Need Nulls in a Boolean Field? Use models.NullBooleanField()
        vii. Set Default value with “default”: - count = models.IntegerField(default = 0)
        viii. Use a inner Meta class to define additional options
    b. Model Methods
        i. model.save(self, *args, **kwargs)
        ii. model.delete(self, *args, **kwargs)
        iii. model.get_absolute_url(self)
        iv. model.__str__(self) [Python 3]
        v. Override with super(ModelClass, self).save(*args, **kwargs)
    c. Activating a Model
        i. Add the app to INSTALLED_APPS in settings.py
        ii. Migrations
            1. Makemigrations
            2. Migrate
    d. Selecting Objects

i. Models include a default manager called objects
ii. Manager methods allow selecting all or some instances
    1. Question.objects.all()
    2. Question.objects.get(pk = 1)
    3. Question.objects.filter(created_date__lt = '2014-01-01')
    4. All of the above return a queryset

11. Function vs. Class Views
    a. Django allows two styles of views – functions or class based views
    b. Functions – take a request object as the first parameter and must return a response object
    c. Class based views – allow CRUD operations with minimal code. Can inherit from multiple generic view classes (i.e. Mixins)
    d. Sample – Viewing a List of Questions
        i.
```
from .models import Question
from django.shortcuts import render_to_response

def question_list(request):
    questions = Question.objects.all()
    return render_to_response('question_list.html', {
                                'questions':questions
})
```
    e. Quick CRUD Operations with Generic Views
        i. ListView
        ii. UpdateView
        iii. CreateView
        iv. If Model is specified, automagically creates a matching ModelForm
        v. Form will save the Model if data passes validation
        vi. Override form_valid() method to provide custom logic (i.e sending email or setting additional fields)
    f. Sample – As Class Based View
        i.
```
from .models import Question
from django.views.generic import ListView

class QuestionList(ListView):
    model = Question
    context_object_name = 'questions'
```

12. Django Templates
    a. variables = {{variable_name}}
    b. template tags = {%tag%}
    c. Flexible – can be used to render html, text, csv, email, you name it!

d. Dot notation – template engine attempts to resolve by looking for matching attributes, hashes and methods
e. Question List Template
i.

```
<!doctype html>
<html lang=en>
<head>
    <meta charset=utf-8>
    <title>List of Questions</title>
</head>
<body>
{%if questions%}
<ul>
    {%for q in questions%}
    <li>{{q.question_text}}</li>
    {%endfor%}
</ul>
{%else%}
<p>No questions have been defined</p>
{%endif%}
</body>
</html>
```

13. Urls.py
    a. Defines routes to send urls to various views
    b. Can use regular expressions
    c. Extract parameters from a url and pass to the view as a named parameter:
        i. r('^question/(?P<question_id>\d+)/$','views.question_detail')
    d. Extensible – urls.py can include additional url files from apps:
        i. r('^question/',include(question.urls))
    e.

```
from django.conf.urls import patterns
from views import QuestionListView

   urlpatterns = patterns('',
      (r'^questions/$','views.QuestionList.as_view())

)
```

14. Forms in Django
    a. django.forms provides a class to build HTML forms and validation.
    b.

```
from django import forms

class EditQuestionForm(forms.Form):
   question_text = forms.CharField(max_length =
200)
```

c. ModelForms
    i. Automatically generate a form from a model.
    ii. Handles saving a bound model
    iii. Can specify fields to be included or excluded in the form
    iv.
```
from django.forms import ModelForm
from .models import Question

class QuestionForm(ModelForm):
    class Meta:
        model = Question
        fields = ['question_text']
```
    v. Using a ModelForm
        1. Create an instance of an empty form - form = QuestionForm()
        2. Pass the form into the template and use the form methods to render the form
            a. Form.as_p
            b. Form.as_ul etc..

15. Request & Response
    a. Request object encapsulate the request and provide access to a number of attributes and methods for accessing cookies, sessions, the logged in user object, meta data (i.e environment variables),
    b. Response objects are returned to the browser. Can set content type, content length, response does not have to return HTML or a rendered template
    c. Special response types allow for common functionality
        i. HttpResponseRedirect
        ii. Http404
        iii. StreamingHttpResponse
16. Django Extras
    a. CRSF Middleware – enabled by default. Include template tag in all forms: {%csrf_token%}
    b. Authentication
    c. Caching
    d. Sessions
    e. Messages
    f. Email
    g. Logging
17. Authentication
    a. Django's out of the box Auth system uses database authentication.
    b. If using the Authentication middleware and context_processors the current user is available to code as request.user and {{user}} is defined in all templates
18. Auth Decorators
    a. Login_required

    b. @login_required
        def function_view(request):
    c. @user_passes_test(lambda u: u.is_staff)
    d. Has_perms
19. Sending Email
    a. django.core.mail includes functions and classes for handling email
    b. Set EMAIL_HOST in settings.py to outgoing mail server
    c. Import send_mail for simple mail:
        send_mail(subject, message, from, to_emails)
    d. Use django.template.render_to_string to format a message using a template
    e. Use EmailMultiAlternatives to create a text message and attach a html version as well.

Other areas to cover in detail:

1. Django ORM
2. Admin customization
3. Authentication
4. User model
5. Media and static files management
6. Template tags

Assignments:

Set 1

1. Create a poll app and list the questions and choices. The user can select the choice and on submit it will display the statistics.
2. Create a website with below features
    a. Users can register into the system
    b. Users can login into the system
    c. Users can update their profile

Set 2:

1. In the created project(user registration) create a new app posts and implement below features:
    a. Users should be able to post texts
        i. Use Ajax
    b. Other users should able to view all the posts created by all users based on the date of creation.
        i. Use twitter style pagination
    c. User can like(dislike a liked post) a post and each post should show number of likes

Set 3: Time: 1 day

1. Implement social login- using facebook and google - Use allauth

Set 4: Time - 2 days

1. Create Rest API's for below features - use django rest framework
    a. Login(Use token authentication)
    b. Create post
    c. List posts
    d. Like post

Set 5: Time 1 days

1. Add an option to upload image along with a post creation
2. Implement search posts feature
3. On like of a post the owner should get an email notification