EXAMINATION PREPARATION

Basic terms and concepts

Define, compare and discuss the following terms and concepts:

a. HTML 4.01 and HTML5 (and XHTML)

b. CSS
   (i) Ways of using CSS in an HTML document
   (ii) Advantages of using CSS in Web application development

c. HTML attributes: action, method, id, class, . . .

d. JavaScript

e. Required elements in an HTML 4.01 strict document

f. Defining standards of the World Wide Web

g. Components of the standard three-tier Web application architecture

h. When to use methods GET and POST

i. Other HTTP methods and when to use them

j. Difference between HTTP and HTTPS

k. Application requirements provided by HTTPS

l. Cookies

m. Methods of maintaining state in Web applications

n. Input validation
   (i) When on client, when on server?

o. Security threats
   (i) Cross-site scripting
   (ii) Cross-site request forgery
   (iii) SQL query injection

p. One-way (hash) function

q. Required meta-data for (uploaded) images

r. DOM, XML, XPath, JSON

s. Ajax: JavaScript, DOM, XMLHttpRequest

t. Ajax frameworks: Dojo Toolkit, JQuery, Prototype; details of jQuery
u. PHP, ASP.Net, Java EE, Django, Rails
v. (Java EE frameworks: Struts 2, Hibernate, App Fuse)
w. Design patterns: MVC, MVT
x. Traditional Web Services: SOAP, WSDL, UDDI
y. RESTful Web Services: examples, concepts, properties
z. Horizontal and vertical scaling

Framework comparison and selection

Components of a good framework for Web application development?
Properties of and relationships between:

a. PHP (Smarty)
b. ASP.Net
c. [Java EE (Struts 2, Hibernate, Ant, IntelliJ, Play!)]
d. Django
e. Ruby on Rails

For example, which of the above frameworks is most suitable to an individual developer, and which is most suitable for a large organisation, and why?
For example, which pair of the above frameworks is most similar and why?
For example, of PHP, ASP.Net, Java EE and Django, which two pairs are most similar to each other, and why?
(Briefly describe the purpose of Struts 2, Hibernate, Ant and IntelliJ in Java EE Web development.)
Briefly describe three important criteria in choosing a framework for Web development in a small to medium-sized organisation.

Basic (Django) programming

Relationships between:

a. Projects
b. Apps
c. URL patterns
d. Models

e. Views

f. Templates

Performing standard tasks:

a. Using (Python) lists and dictionaries

b. Defining models (\_unicode\_, get_absolute_url())

c. Using the admin interface

d. Processing form data

e. Rendering a template

f. Redirecting to a URL

g. Reusing the same URL to render a form and process a form

h. Advantages and disadvantages of doing this

i. Using generic views

j. Using django.contrib.auth

k. Using (Python) decorators

l. Avoiding URL patterns in views and templates: why? how?

m. Using contributed libraries

User interface issues, e.g., 3 ways of presenting an HTML page with header, 3 columns and footers.

Model design and querying

Describe criteria for good model (database) designs
Identify faults in bad model (database) designs
Construct good model (database) designs
How to express database queries in:

a. SQL

b. Django API

Consider the following database design for an IMDB-like movie database application.
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=80)
    birthday = models.DateField()
    gender = models.CharField(max_length=1)

class Movie(models.Model):
    title = models.CharField(max_length=80)
    director = models.ForeignKey(Person)
    year = models.IntegerField()
    cast = models.ManyToManyField(Person, through="Role")

class Role(models.Model):
    movie = models.ForeignKey(Movie)
    actor = models.ForeignKey(Person)
    role = models.CharField(max_length=80)
    # character played by actor in movie

What assumptions does this model make?

Write Django statements using the database API that express the following queries. For example, the statement

movies = Movie.objects.all()

expresses the query “What are all the movies”.

a. What is the name of the director who directed the movie “Raiders of the Lost Ark”?

director_name =
    Movie.objects.get(title="Raiders of the Lost Ark").
    director.name

b. What are the titles of all movies directed by “Steven Spielberg”?

director = Person.objects.get(name="Steven Spielberg")
movie_titles =
    [movie.title for movie in
    Movie.objects.filter(director=director)]

c. What are the names of all female actors who starred in movies containing the character “Indiana Jones”? 

roles = Role.objects.filter(role="Indiana Jones").values()
actors =
    Person.objects.filter(id__in=[ role["actor"] for role in roles ],
    gender="f")
names = [ actor["name"] for actor in actors.values() ]

Continued next page . . .
d. Find the titles of all movies made between 2001 and 2005 (inclusive).

e. Find the names of all actors who performed in the movie “Star Wars”.

f. Find all actors who starred in “Star Wars”, listed in order of increasing age.

Suppose you are designing a Web application that allows different registered users to contribute timestamped articles to a single Web log (or blog). All readers of the blog may add one or more comments to each article.

Consider the following database design for such an application.

```python
from django.db import models

class Article(models.Model):
    author_name = models.CharField(max_length=80)
    author_email = models.EmailField()
    article_title = models.CharField(max_length=160)
    article_text = models.TextField()
    article_date = models.DateTimeField()
    comment_author = models.CharField(max_length=80)
    comment_text = models.TextField()
```

Give two main reasons why this is a bad database design.

Give a better database design for this application.

What is an atomic database transaction? Briefly describe one situation in which something can go wrong if transactions are not executed atomically. Briefly describe how to ensure transactions are executed atomically in Django.

**Application design**

**Question 1**

Suppose you have to design a **content management system** for an online newspaper. Such a system should operate as follows.

**Readers** may view current, published articles and photos. Readers may also comment on published articles, and such comments may be read by any reader.

**Journalists** (who may also be readers) may login and submit articles for publication. To submit an article, a journalist has to provide the text of the article and one or more related photos. All submitted articles are retained indefinitely.

**Editors** (who may also be journalists or readers) may login and review submitted (but not yet published) articles. An editor may reject or accept a submitted article. If the editor accepts an article, he or she adds a headline, edits the text of the article, and accepts one or more of the related photos. The editor then publishes the edited article and the accepted photos. Published articles are current until they are a week old.
a. The data model: Describe all database models required, the meaning of the fields, and any constraints required.

b. The URL patterns: Describe the set of URL patterns required and the (Django) view each pattern is associated with. The views may be either user-defined or (Django) library views.

c. The views: Describe the key (server-side) views that will be required, the transactions that they implement, and the models they access and update. Distinguish between views that return HTML pages and views that are executed only for their effect.

d. The transition diagram: Describe the transitions the application makes between scripts, where each node of the diagram is associated with a URL pattern whose view either generates an HTML page or updates the database. Include all possible (directed) transitions between nodes in the diagram and indicate what data if any is transferred on each transition.

Provide separate entry nodes for readers, journalists and editors.
For simplicity, do not provide nodes and transitions associated with user login.
Do not show nodes for HTML templates.

Clearly state any assumptions you make to resolve possible ambiguities.

Question 2

Suppose you are asked to design an online, Web-based, auction system (like eBay). The system should operate as follows.

Articles advertised for sale have a category, a description, a current highest price, and a closing date for the auction. Categories are strings such as “books”, “movies”, “electronics”, ….

Vendors (i.e., sellers) may login and advertise articles for sale. To advertise an article for sale, a vendor must provide the article’s category, description, starting price and the auction’s closing date.

Purchasers may login and bid for an advertised article until the auction’s closing date. Each bid must be strictly higher than the previous highest bid for the article.

Users may browse the articles currently advertised for sale, i.e., the articles whose auctions have not yet closed, by category. Users may also search for articles by words in their description. Users may also register if they have not already done so and login so they can act as vendors or purchasers.

a. The data model: Describe all database models required, the meaning of the fields, and any constraints required.

b. The URL patterns: Describe the set of URL patterns required and the (Django) view each pattern is associated with. The views may be either user-defined or (Django) library views.

c. The views: Describe the key (server-side) views that will be required, the transactions that they implement, and the models they access and update. Distinguish between views that return HTML pages and views that are executed only for their effect.
d. The transition diagram: Describe the transitions the application makes between scripts, where each node of the diagram is associated with a URL pattern whose view either generates an HTML page or updates the database.

Question 3

Describe informally how you would design an application to provide a simple online community server. Such a server (e.g., Yahoo! Groups) manages groups, messages and members. Each group is concerned with a particular topic, has a summary describing its purpose, is created by some member, and contains a simple (chronological) sequence of messages on that topic.

Users may browse the list of groups and search the list of groups by topic. Users may also enter and leave groups, browse and search the sequence of messages in a group, and read individual messages.

Users may become members by registering. Members may login. After logging in, members may do everything arbitrary users can do, and may also create new groups and write new messages in a group.

Members may store personal information about themselves, may view personal information of other members, may send private messages to other members, may subscribe and unsubscribe from one or more groups, and can browse just those groups to which they are currently subscribed for convenient access.

For this question, you only need to consider the most important fields of each entity, but you should consider all relevant constraints (e.g., you can only unsubscribe from a group if you are currently subscribed to it).

Your description should include the following aspects of the implementation:

a. The data model: Describe all database models required, the meaning of the fields, and any constraints required.

b. The URL patterns: Describe the set of URL patterns required and the (Django) view each pattern is associated with. The views may be either user-defined or (Django) library views.

c. The views: Describe the key (server-side) views that will be required, the transactions that they implement, and the models they access and update. Distinguish between views that return HTML pages and views that are executed only for their effect.

d. The transition diagram: Describe the transitions the application makes between scripts, where each node of the diagram is associated with a URL pattern whose view either generates an HTML page or updates the database.

Outline solution to Question 1

Database design. Here is one possible design. It assumes a single model Employee for journalists and editors with a category field that describes their role (and permissions).
from django.db import models

class Employee(models.Model):
    name = models.CharField(max_length=80)
    username = models.CharField(max_length=80)
    password = models.CharField(max_length=80)
    category = models.CharField(max_length=1)

class Article(models.Model):
    author = models.ForeignKey(Employee)
    headline = models.CharField(max_length=80)
    text = models.TextField()
    photos = models.ManyToManyField(Photo)
    submission_date = models.DateTimeField(default=datetime.datetime.now)
    publication_date = models.DateTimeField(default=datetime.datetime.now)
    status = models.CharField(max_length=1)
    editor = models.ForeignKey(Employee)

class Photo(models.Model):
    image = models.ImageField(upload_to='images', blank=True,
       height_field='height', width_field='width')
    height = models.IntegerField(blank=True, null=True)
    width = models.IntegerField(blank=True, null=True)

class Comment(models.Model):
    article = models.ForeignKey(Article)
    author = models.CharField(max_length=80)
    text = models.TextField()

This database design defines its own user class (Employee). This requires extra work but avoids having to extend the library class User with a profile containing the user’s category. Similarly, the design defines its own comment class, for clarity. It allows the same photo to be used in different articles, which seems no more difficult than restricting each photo to occur in a single article. Solutions that clearly explained how to use the library classes User and Comment would be fine.

Field category in class Employee has values “j” (journalist) or “e” (editor). You need a different representation if you want to allow the same user to be both a journalist and an employee.

It’s necessary to have both submission and publication date fields in class Article. Field status in class Article has values ”s” (submitted), ”a” (accepted) or ”r” (rejected).

URL pattern design.

articles/ - list current (accepted) article headlines
articles/\d+/ - display detail of article with given id
articles/\d+/comment/ - add a comment to the article with given id
submit/ - submit an article for publication (requires journalist to be logged in)
submitted/ - list submitted articles (requires editor to be logged in)
submitted/d+ - display and edit submitted article with given id (ditto)
submitted/d+/accept/ - accept submitted article with given id (ditto)
submitted/d+/reject/ - reject submitted article with given id (ditto)
login/ - login as journalist or editor (requires no-one to be logged in)
logout/ - logout (requires user to be logged in)

It’s important to ensure that every requirement in the specification is associated with some URL, that each URL is associated with a well-defined task (view), and that the set of URLs is somehow clear and consistent. This design assumes articles are identified by positive integers (not by slugs).

Views. Only a few view descriptions are shown.

```python
def article_list(request):
    """
    Displays a list of all current, published article headlines.
    Each headline is a link to the article details, including comments. The view accesses the models Article and does not update the database. It renders an HTML template.
    """

def article_detail(request, article_id):
    """
    Displays the details of the article with the given article_id.
    The details include any photos and comments, and provides a form for the reader to add new comments. The view accesses the models Article and Photo. It renders an HTML template.
    """

def article_comment(request, article_id):
    """
    Adds a posted comment to the article with the given id.
    The view updates the model Comment. It redirects back to the URL pattern /articles/article_id/.
    """
```

Transition diagram. Not provided.

Other topics

a. Briefly describe two main differences between HTML and XML.
   Briefly describe two main differences between the XML query languages XPATH and XSLT.
b. What is the difference between a news feed and a news aggregator. 
   Briefly describe (a) Atom and (b) AtomPub.

c. Name three main components of industry-standard (“big”) Web Services, and briefly 
   describe the purpose of each component.

d. Briefly describe the main concepts behind RESTful Web Services.
   Give two examples of existing RESTful Web Services.