

# Web Application Development with Python L-4

## Training Program

### Written Question and Answer



#### Qs21. What is a QuerySet in Django?

A QuerySet is a collection of database queries that can be used to retrieve, filter, and manipulate data from the database.

```
views.py

# Get all records
Book.objects.all()

# Filter records
Book.objects.filter(author='J.K. Rowling')

# Get single record
Book.objects.get(id=1)

# Order records
Book.objects.order_by('title')
```

#### Qs22. How do you pass data to a template?

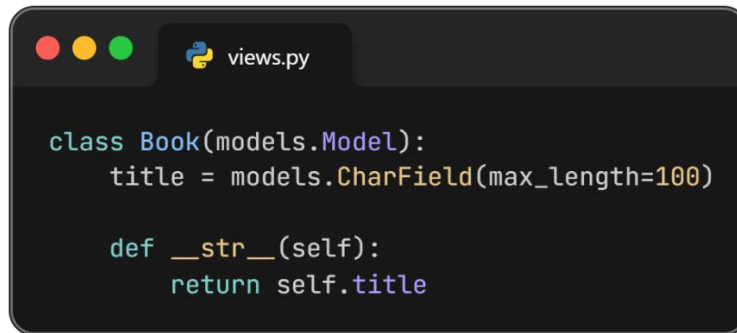
Through the context dictionary in a view we can pass data to a template:

```
views.py

def book_list(request):
    books = Book.objects.all()
    context = {
        'books': books
    }
    return render(request, 'book_list.html', context)
```

#### Qs23. What is \_\_str\_\_() method in models?

The `__str__()` method in Django models is a special Python method that defines the string representation of an object. When Django needs to display the object in a string context (like in the Django Admin or templates) it defines the human-readable string representation of a model instance.



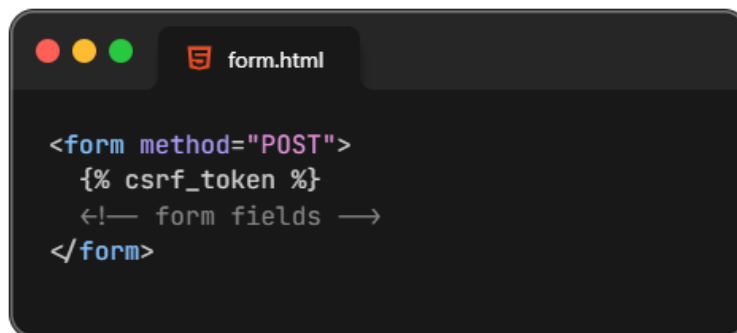
```
class Book(models.Model):
    title = models.CharField(max_length=100)

    def __str__(self):
        return self.title
```

#### Qs24. What is csrf\_token?

---

In Django, csrf\_token is a security mechanism used to protect web applications from Cross-Site Request Forgery (CSRF) attacks. CSRF attacks trick authenticated users into performing unintended actions on a website.



```
<form method="POST">
    {% csrf_token %}
    <!-- form fields -->
</form>
```

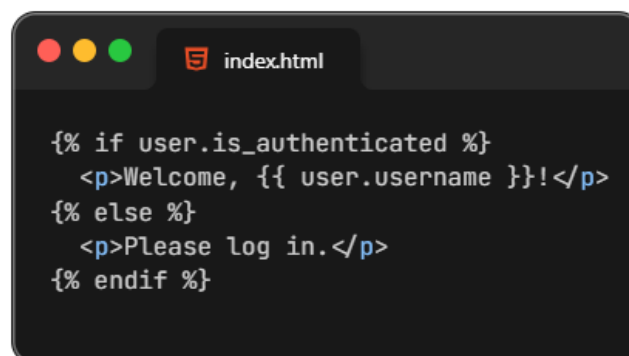
#### Qs25. What are template tags?

---

In Django, template tags are special syntax elements enclosed within `{% %}` that allow you to add logic and control flow within your templates, similar to programming constructs, but specifically for the template language. They enable tasks like looping, conditional logic, and other operations that go beyond simply displaying data.

##### ◇ Types of Template Tags:

- **Control flow:** `{% if %}`, `{% for %}`, `{% elif %}`, `{% else %}`
- **Inclusion:** `{% include %}`, `{% extends %}`, `{% block %}`
- **Utility:** `{% csrf_token %}`, `{% url %}`, `{% static %}`



```
{% if user.is_authenticated %}
    <p>Welcome, {{ user.username }}!</p>
{% else %}
    <p>Please log in.</p>
{% endif %}
```

#### Qs26. What is the difference between null=True and blank=True??

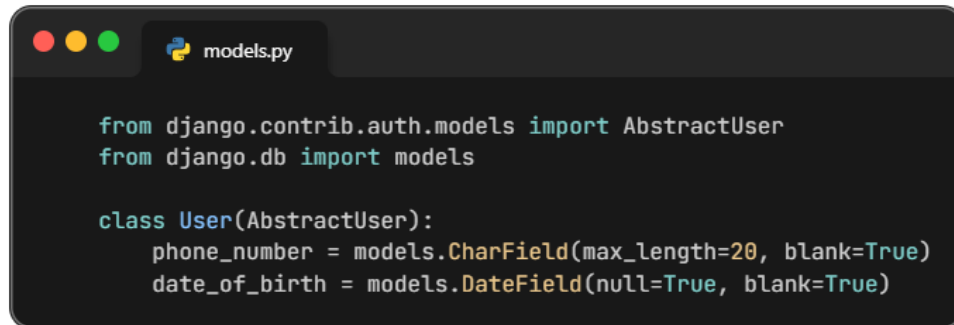
---

- **null = True** sets DB column to NULL.
- **blank = True** allows form field to be empty

## Qs27. How to create custom user model?

---

To create a custom user model in Django, we should extend **AbstractUser** or **AbstractBaseUser** and define our model within an app. Then, specify this new model in your **settings.py** file using the **AUTH\_USER\_MODEL** setting.

A screenshot of a code editor window titled 'models.py'. The code defines a custom Django user model named 'User' that inherits from 'AbstractUser'. It includes two fields: 'phone\_number' and 'date\_of\_birth'.

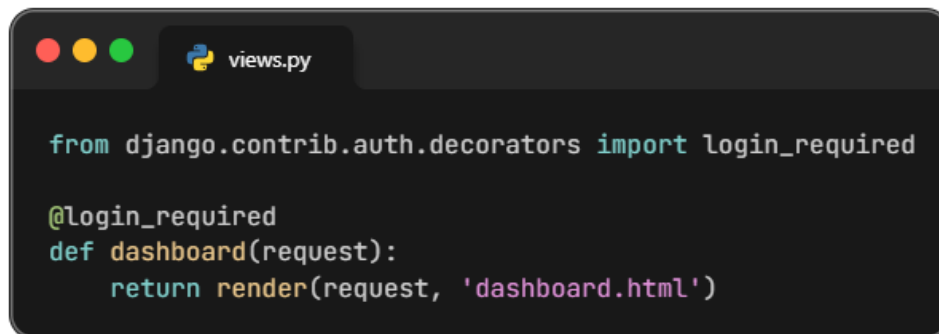
```
from django.contrib.auth.models import AbstractUser
from django.db import models

class User(AbstractUser):
    phone_number = models.CharField(max_length=20, blank=True)
    date_of_birth = models.DateField(null=True, blank=True)
```

## Qs28. What is @login\_required?

---

**@login\_required** is a decorator in Django that restricts view access to only authenticated users. If an unauthenticated user tries to access view, they will be redirected to the login page specified in Django's settings.

A screenshot of a code editor window titled 'views.py'. The code imports the '@login\_required' decorator and defines a 'dashboard' view function that renders 'dashboard.html'.from django.contrib.auth.decorators import login\_required

@login\_required
def dashboard(request):
 return render(request, 'dashboard.html')

## Qs29. Explain ForeignKey, OneToOneField, and ManyToManyField??

---

### ◆ ForeignKey

A ForeignKey represents a many-to-one relationship. This means many records in one model can be related to one record in another model.

**Example use case:** Many books written by one author.

### ◆ OneToOneField

A OneToOneField represents a one-to-one relationship. This means one record in a model is related to only one record in another model.

**Example use case:** Each user has one profile.

### ◆ ManyToManyField

A ManyToManyField represents a many-to-many relationship. This means multiple records in one model can be related to multiple records in another model.

**Example use case:** Students enrolled in multiple courses, and each course has multiple students.

## Qs30. What is the difference between path() and re\_path() functions in Django URL patterns?

---

In Django, you can filter a queryset against the current user by using the user attribute of the request object.



---

### Qs31. What does CRUD stand for?

CRUD stands for **Create, Read, Update, and Delete** — the four basic operations used to manage data in a database.

---

### Qs32. How do you perform Read operation in Django?

Reading data (retrieving objects) is done using Django QuerySets:

- `Model.objects.all()` – to get all records
- `Model.objects.get(id=1)` – to get a single object
- `Model.objects.filter(status='active')` – to get filtered objects

---

### Qs33. Explain the concept of middleware in Django.

Middleware in Django is a framework of hooks that process requests and responses globally across the application. When a request comes in, it passes through a series of middleware components before reaching the view. After the view processes the request and returns a response, that response again passes through the middleware before being sent to the client.

This mechanism allows middleware to perform tasks such as user authentication, session handling, logging, request/response modification, and security checks like CSRF protection. Middleware components are added to the **MIDDLEWARE** list in the **settings.py** file, and Django processes them in the order they appear.

---

### Qs34. What is the difference between `create()` and `create_user()`

- **`create()`** is a generic method used to create any model instance. It saves the data as-is without special handling. It does not hash passwords, so it's not secure for creating user accounts.
- **`create_user()`** is a specialized method provided by Django's `UserManager` for creating `User` objects, specifically designed for the `User` model or custom user models. It automatically handles important user-related setup like password hashing, setting `is_active=True`, and more.