

ORAL Question and Answer

Qs1. Which file is responsible for defining the database settings for a Django project?

Ans: The settings.py file is responsible for defining the database settings for a Django project. In this file, you can find the DATABASES configuration where you specify details such as the database engine, name, user, password, host, and other settings.

Qs2. Explain the purpose of the django admin site?

Ans: The Django admin site is a built-in web-based interface that provides a powerful and customizable administrative interface for managing a Django web application.

It provides functionalities like:

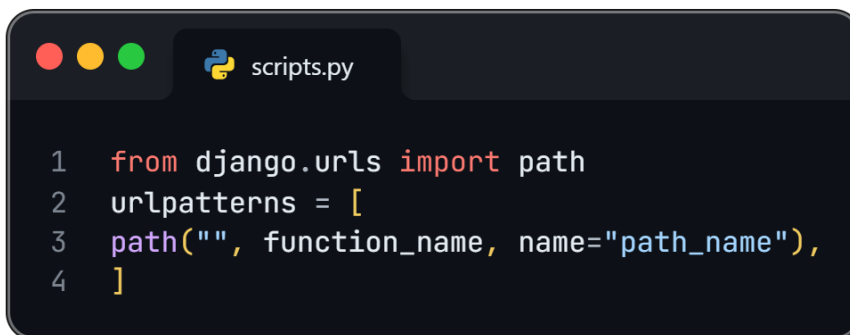
- Creating, editing, and deleting data records.
- Viewing all data objects for a specific model.
- Searching and filtering data based on various criteria.

Registering new models to be accessible in the admin interface.

Qs3. What is the purpose of a URL pattern in django?

Ans: The purpose of a URL pattern in Django is to map URLs to views. It defines the structure of the URL and specifies which view function or class-based view should handle the corresponding HTTP request.

Example:

A code editor window titled 'scripts.py' with a dark background. It contains four lines of Python code for defining Django URL patterns. The code is: 1 from django.urls import path, 2 urlpatterns = [, 3 path("", function_name, name="path_name"),, 4].

```
1 from django.urls import path
2 urlpatterns = [
3     path("", function_name, name="path_name"),
4 ]
```

Qs4. What is the difference between path() and re_path() functions in django URL patterns?

Ans:

Path = The path() function is used for simple, straightforward URL patterns that do not involve complex regular expressions.

The re_path() function is used when you need to use regular expressions to define more complex and flexible URL patterns.

Example:

A code editor window titled 'scripts.py' with a dark background. It contains six lines of Python code for defining Django URL patterns. The code is: 1 from django.urls import path, 2 urlpatterns = [, 3 path("", function_name, name="path_name"),, 4, 5 re_path(r'^example/(?P<slug>[\w-]+)/\$', dynamic_view),, 6].

```
1 from django.urls import path
2 urlpatterns = [
3     path("", function_name, name="path_name"),
4
5     re_path(r'^example/(?P<slug>[\w-]+)/$', dynamic_view),
6 ]
```

Qs5. What does the 'yesno' filter do in Django templates?

Ans: In Django templates, the yesno filter is used to display one of two specified strings based on whether a given value is true or false.

```
1  {{ value | yesno:"yes,no,maybe" }}
```

Qs6. What does the 'date' filter do in Django templates?

Ans: In Django templates, the date filter is used to format date and time values according to a specified format. It allows you to present date and time information in a human-readable way by applying a specific format to a datetime object.

Qs7. What is a Django form and what is its purpose?

Ans: In Django, a form is a Python class that defines the structure and behavior of an HTML form. The purpose of a Django form is to simplify the process of collecting and validating user input on the server side.

```
1  from django import forms
2  class ContactForm(forms.Form):
3      fields_name = forms.CharField(label='Field Label Name', widget=forms.TextInput())
4  def contact_view(request):
5      form = ContactForm()
6      return render(request, 'contact.html', {'form': form})
```

Qs8. How is a Django form submitted in a template?

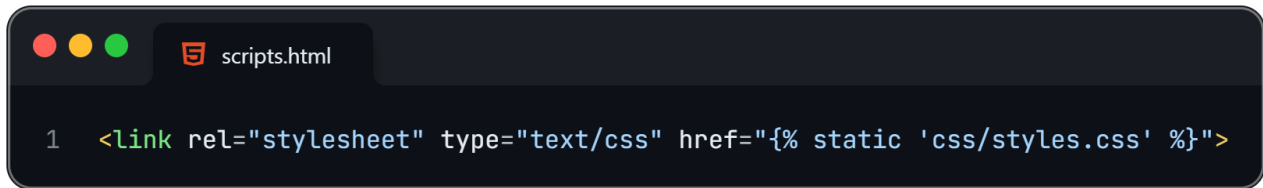
Ans: In Django, forms are typically submitted in templates using HTML <form> elements.

```
1  from django import forms
2  class ContactForm(forms.Form):
3      fields_name = forms.CharField(label='Field Label Name', max_length=50, widget=forms.TextInput())
4  def contact_view(request):
5      form = ContactForm()
6      return render(request, 'contact.html', {'form': form})
```

```
1  <!-- templates/contact.html -->
2  <form method="post" action="{% url 'contact_view' %}">
3      {% csrf_token %}
4      {{ form.as_p }}
5      <button type="submit">Submit</button>
6  </form>
```

Qs9. What is a static file URL in web development?

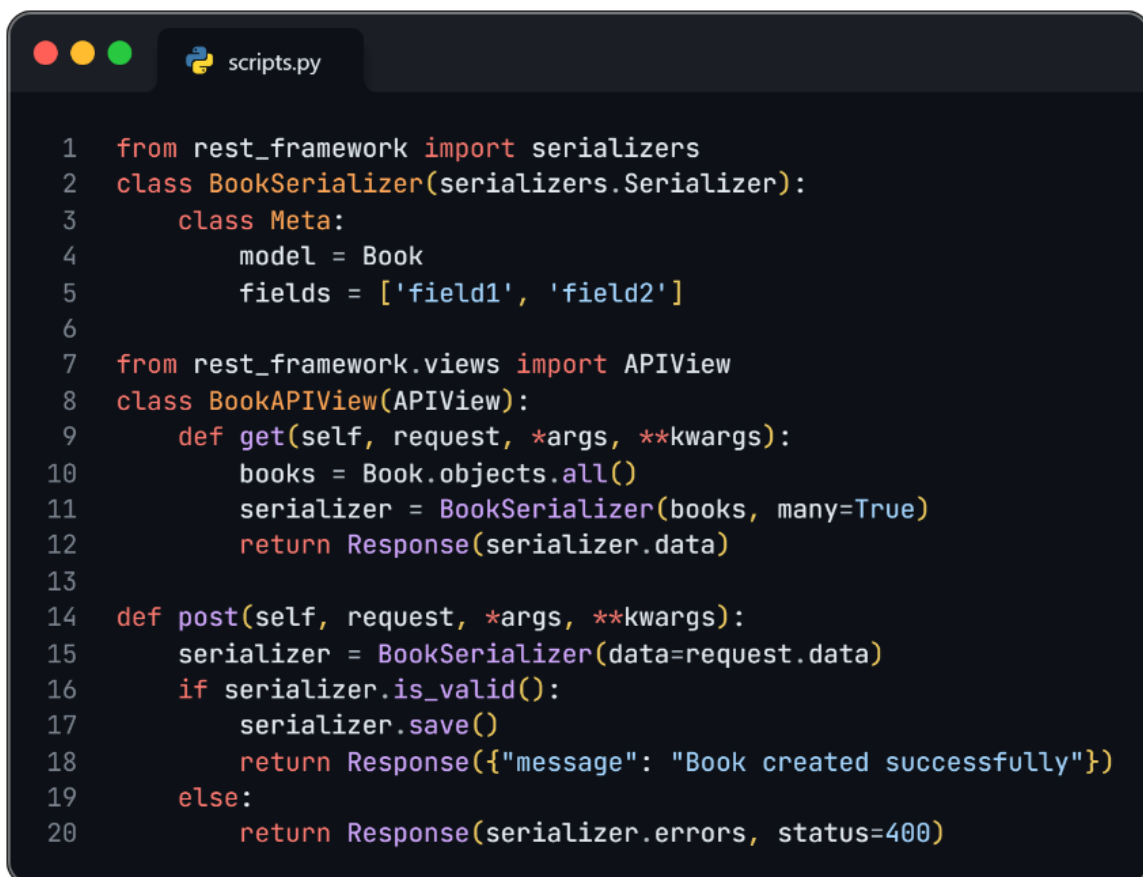
Ans: In web development, a static file URL refers to the web address (URL) used to access static files such as images, stylesheets (CSS), JavaScript files, fonts, and other assets that do not change dynamically based on user input or server-side processing.



```
1 <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
```

Qs10. What is the purpose of serializers in DRF?

Ans: In Django Rest Framework (DRF), serializers are used to converting complex data types, such as Django models or querysets, into native Python datatypes that can be easily rendered into JSON, XML, or other content types. Additionally, serializers handle the reverse process, deserializing data from incoming requests into complex Python data types.



```
1 from rest_framework import serializers
2 class BookSerializer(serializers.Serializer):
3     class Meta:
4         model = Book
5         fields = ['field1', 'field2']
6
7 from rest_framework.views import APIView
8 class BookAPIView(APIView):
9     def get(self, request, *args, **kwargs):
10         books = Book.objects.all()
11         serializer = BookSerializer(books, many=True)
12         return Response(serializer.data)
13
14     def post(self, request, *args, **kwargs):
15         serializer = BookSerializer(data=request.data)
16         if serializer.is_valid():
17             serializer.save()
18             return Response({"message": "Book created successfully"})
19         else:
20             return Response(serializer.errors, status=400)
```

Multiple Choice Questions

1. Which commands use to create a project in Django?

- A. `$ django-admin createprojectproject_name`
- B. `$ django-admin startprojectproject_name`**
- C. `$.djangostartprojectproject_name`
- D. `$ djangocreateprojectproject_name`

2. How many kinds of HTTP requests there in Django?

- A. 2
- B. 3
- C. 4**
- D. 5

3. What is a Django model in the context of web development?

- A. A template engine
- B. A database table representation**
- C. A CSS framework
- 4. A JavaScript library

4. Which Django command is used to generate the necessary database schema based on the model?

- A. `create_model`
- B. `make_model`
- C. `start_model`
- D. `makemigrations`**

5. What is the purpose of the `auto_now_add` option in a Django `DateTimeField`?

- A. Updates the field to the current timestamp whenever the object is saved
- B. Updates the field to the current timestamp every time the model is accessed
- C. Sets the field to the current timestamp when the object is created**
- D. Sets the field to a static timestamp

6. What does the default option in a Django model field represent?

- A. Sets the field as the primary key
- B. Sets the field as required
- C. Sets the default value for the field**
- D. Sets the maximum length for the field

7. What is the purpose of the `get_or_create()` method in Django models?

- A. Deletes a record if it exists, otherwise creates a new one
- B. Retrieves a record if it exists, otherwise creates a new one**
- C. Retrieves a record if it exists, otherwise returns None
- D. Deletes a record if it exists, otherwise returns None

8. What is the purpose of the `{% block %}` tag in Django templates?

- A. Defines a block that can be overridden in child templates**
- B. Imports external JavaScript files
- C. Sets the background colour of a section
- D. Defines a placeholder for images

9. How do you denote a variable in Django templates?

A. {{ variable_name }}

B. { variable_name }

C. [variable name]

D. (variable name)

10. How do you include another template within a Django template?

A. (% load %)

B. (% use %)

C. (% include %)

D. (% import %)

11. Which of the following commands creates a new virtual environment named "myenv" in Python?

A. createenvmyenv

B. python -m venv myenv

C. venvmyenv

D. virtualenvmyenv

12. Which of the following files is used to configure the virtual environment in Python?

A. config.ini

B. virtualenv.ini

C. venv.cfg

D. pyvenv.cfg

13. What is the purpose of the pip freeze command in Python?

A. It installs Python packages.

B. It lists all installed packages and their versions.

C. It upgrades the Python interpreter.

D. It uninstalls Python packages.

14. What is the purpose of the python manage.py collectstatic command in Django?

A. It compiles static files into a single directory for deployment.

B. It creates a new Django app with static file templates.

C. It collects all the media files used in the project.

D. It compiles Python code into bytecode.

15. Which of the following commands activates the virtual environment named "myenv" in Linux or macOS terminal?

A. activate myenv

B. source myenv/bin/activate

C. venv activate myenv

D. myenv/bin/activate

Write the short Answer of the following Questions (16-30)

Qs16. Which file is responsible for defining the database settings for a django project?

In a Django project, the file responsible for defining the database settings is typically named `settings.py`. This file is located within the main project directory. Inside `settings.py`, you'll find a section where you can configure various settings related to the database, including the database engine, database name, user, password, host, and port. These settings are usually specified in a dictionary named `DATABASES`. Here's a typical example:

A screenshot of a code editor window titled 'scripts.py'. The code defines the 'DATABASES' dictionary with a 'default' entry. The entry specifies the engine as 'django.db.backends.sqlite3' and the name as 'BASE_DIR / 'db.sqlite3''.

```
1 DATABASES = {  
2     'default': {  
3         'ENGINE': 'django.db.backends.sqlite3',  
4         'NAME': BASE_DIR / 'db.sqlite3',  
5     }  
6 }
```

In this example, Django is configured to use SQLite as the database engine, and the database file is `db.sqlite3`, which is located in the project's base directory. However, depending on your project requirements, you might configure Django to use other database engines like PostgreSQL, MySQL, or others, and the settings in `DATABASES` would reflect those choices.

Qs17. Where should you store your static files like CSS and JavaScripts in a Django project?

Ans: In Django Project, static files such as CSS, image, fonts and JavaScripts files are store 'static' folder in project main directory or within each app. Use `{% static %}` template tag to reference them in templates.

A screenshot of a code editor window titled 'scripts.txt'. The code lists various file paths and static file types, including 'myproject/', 'myproject/', 'myapp/', 'static/', 'image/', 'styles.css', 'script.js', 'templates/', and 'manage.py'.

```
1 myproject/  
2 |-- myproject/  
3 |-- myapp/  
4 |-- static/  
5 | |-- image/  
6 | |-- styles.css  
7 | |-- script.js  
8 |-- templates/  
9 |-- manage.py
```

Qs18. How is a URL pattern defined in django's 'urls.py' file?

Ans: In Django's `urls.py` file, URL patterns are defined using the `urlpatterns` list, where each URL pattern is represented by a call to the `path()` function or the `re_path()` function for regular expression-based patterns. Each pattern is associated with a view function that handles the corresponding URL, and optionally, a name can be provided for the pattern.

```
scripts.py
1 from django.urls import path
2 from .views import function_name
3 urlpatterns = [
4     path('', function_name, name="path_name"),
5 ]
```

Qs19. What is the purpose of the 'reverse()' function in Django with respect to URLs?

Ans: The `reverse()` function in Django is used to generate URLs dynamically by providing the view name and optionally any parameters needed by the URL pattern. This function helps in avoiding hardcoding URLs in templates or views, making the code more maintainable and flexible. It allows developers to refer to URLs by their logical names rather than their physical paths, which simplifies URL changes and improves code readability.

```
scripts.py
1 # urls.py
2 from django.urls import path
3 from .views import my_view
4 urlpatterns = [
5     path('my-url/', my_view, name='my-view'),
6 ]
7 from django.urls import reverse
8 url = reverse('my-view', args=[1, 'example'])
```

Qs20. What does the 'date' filter do in Django templates?

In Django templates, the date filter is used to format dates according to a given format string. It takes a date object or a string representing a date as input and outputs a string formatted according to the specified format. For example, if you have a date object `my_date` representing January 1, 2024, you can format it using the date filter like this

```
index.html
1 {{ my_date|date:"Y-m-d" }}
```

This will output: 2024-06-16

Qs21. What does the purpose of the 'length' filter in Django templates?

Ans: In Django templates, the length filter is used to get the length of a variable. It works with various type of variable such as string, list, querysets etc. It returns the number of items in a list, the number of characters in a string, or the number of keys in a dictionary.

```
scripts.html
1  {% with my_list=[1, 2, 3, 4, 5] %}
2  The length of the list is: {{ my_list|length }}
3  {% endwith %}
```

Qs22. How do you create a new database record (object) using Django ORM?

To create a new database record (object) using Django's ORM (Object-Relational Mapping), you typically follow these steps:

1. Import the model class representing the database table where you want to create a new record.

```
scripts.py
1  from django.db import models
2
3  class Person(models.Model):
4      first_name = models.CharField(max_length=30)
5      last_name = models.CharField(max_length=30)
6      age = models.IntegerField()
7
8      def __str__(self):
9          return f"{self.first_name} {self.last_name}"
10
```

2. Create an instance of the model class, setting the desired field values.

```
scripts.py
1  from myapp.models import Person
2
3  new_person = Person.objects.create(first_name="John", last_name="Doe", age=28)
4
```

3. Call the save() method on the instance to persist it to the database.

```
scripts.py
1  from myapp.models import Person
2
3  new_person = Person(first_name="John", last_name="Doe", age=28)
4  new_person.save()
```

Qs23. How do you perform a bulk delete operation on multiple records in Django models?

Ans: In Django, you can perform a bulk delete operation on multiple records in a model using the delete() method provided by the QuerySet.


```
scripts.py

1 from yourapp.models import YourModel
2 YourModel.objects.filter(age__lt=30).delete()
```

Qs24. How do you define a form class in django?

Ans: In Django, you can define a form class by creating a subclass of Django's `forms.Form` or `forms.ModelForm` class.

```
scripts.py

1 # forms.py
2 from django import forms
3 class YourForm(forms.Form):
4     name = forms.CharField(max_length=100)
5     age = forms.IntegerField()
6     email = forms.EmailField()
```

Qs25. How do you access form data submitted through POST method in a Django view?

Ans: In a Django view, you can access form data submitted through the POST method using the request.POST dictionary. When a form is submitted via POST, the form data is sent as part of the request, and you can retrieve it in your view.

```
scripts.py

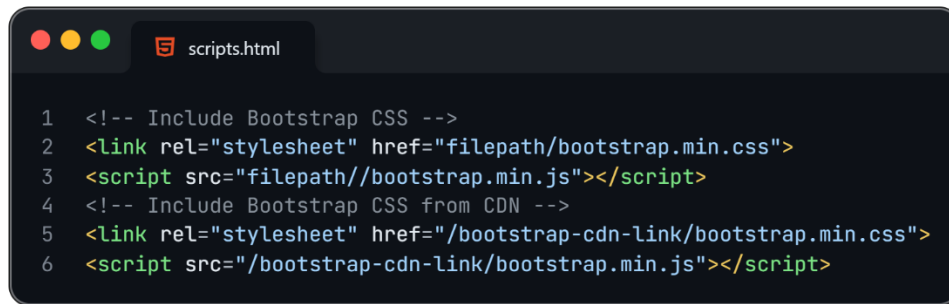
1 from django import forms
2 class YourForm(forms.Form):
3     name = forms.CharField(max_length=100)
4 def your_view(request):
5     if request.method == 'POST':
6         form = YourForm(request.POST)
7         if form.is_valid():
8             name = request.POST['name']
9
10    else:
11        form = YourForm()
12    return render(request, 'your_template.html', {'form': form})
```

Qs26. What is the role of the 'get_context()' method in custom form widgets in django?

Ans: The `get_context()` method in custom Django form widgets is used to provide extra context data for widget rendering, enhancing customization options in templates. It allows you to add information that can be accessed within the template using `field.widget.custom_data`.

Qs27. How can you include Bootstrap in your HTML file?

Ans: To include Bootstrap in your HTML, by using the Bootstrap CDN (Content Delivery Network) or download the Bootstrap files locally in your project.

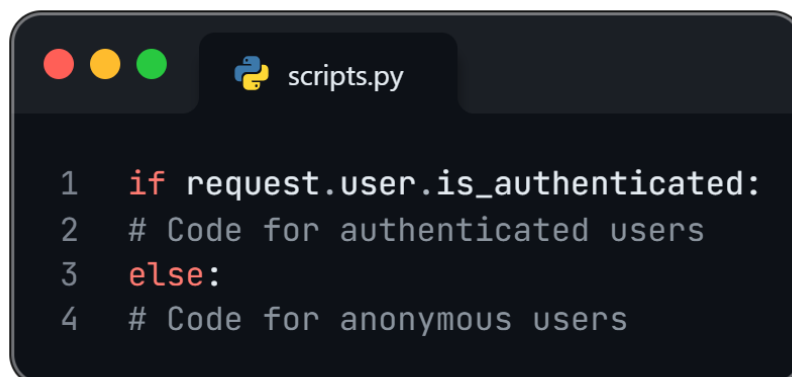


```
1 <!-- Include Bootstrap CSS -->
2 <link rel="stylesheet" href="filepath/bootstrap.min.css">
3 <script src="filepath/bootstrap.min.js"></script>
4 <!-- Include Bootstrap CSS from CDN -->
5 <link rel="stylesheet" href="/bootstrap-cdn-link/bootstrap.min.css">
6 <script src="/bootstrap-cdn-link/bootstrap.min.js"></script>
```

Qs28. What is the role of the 'is_authenticated' attribute in Django user authentication?

Ans: In Django, the `is_authenticated` attribute is a built-in attribute of the User object provided by the authentication system. This attribute is used to check whether a user is authenticated or not.

When a user is successfully authenticated, the `is_authenticated` attribute returns `True`.



```
1 if request.user.is_authenticated:
2     # Code for authenticated users
3 else:
4     # Code for anonymous users
```

Qs29. What are Bootstrap modals, and how are they implemented?

Ans: Bootstrap modals are dynamic dialog boxes in web development used to display content or forms; they are implemented using HTML, CSS, and Bootstrap's JavaScript functionality.

Qs30. Name two commonly used web servers with Django and briefly explain their roles.

Ans: Two commonly used web servers with Django are:

- **Gunicorn (Green Unicorn):** Gunicorn is a widely used WSGI (Web Server Gateway Interface) HTTP server for running Django applications. It acts as a middleman between the Django application and the web client, handling incoming HTTP requests, communicating with Django, and serving responses back to the client. Gunicorn is often used in production environments due to its simplicity, scalability, and compatibility with Django.
- **uWSGI (micro WSGI):** uWSGI is another popular WSGI HTTP server commonly used with Django. Similar to Gunicorn, uWSGI acts as a bridge between Django and the web client, handling HTTP requests and responses. It's known for its performance, flexibility, and extensive feature set. uWSGI supports various protocols, including HTTP, WebSocket, and FastCGI, making it suitable for deploying Django applications in a variety of scenarios.