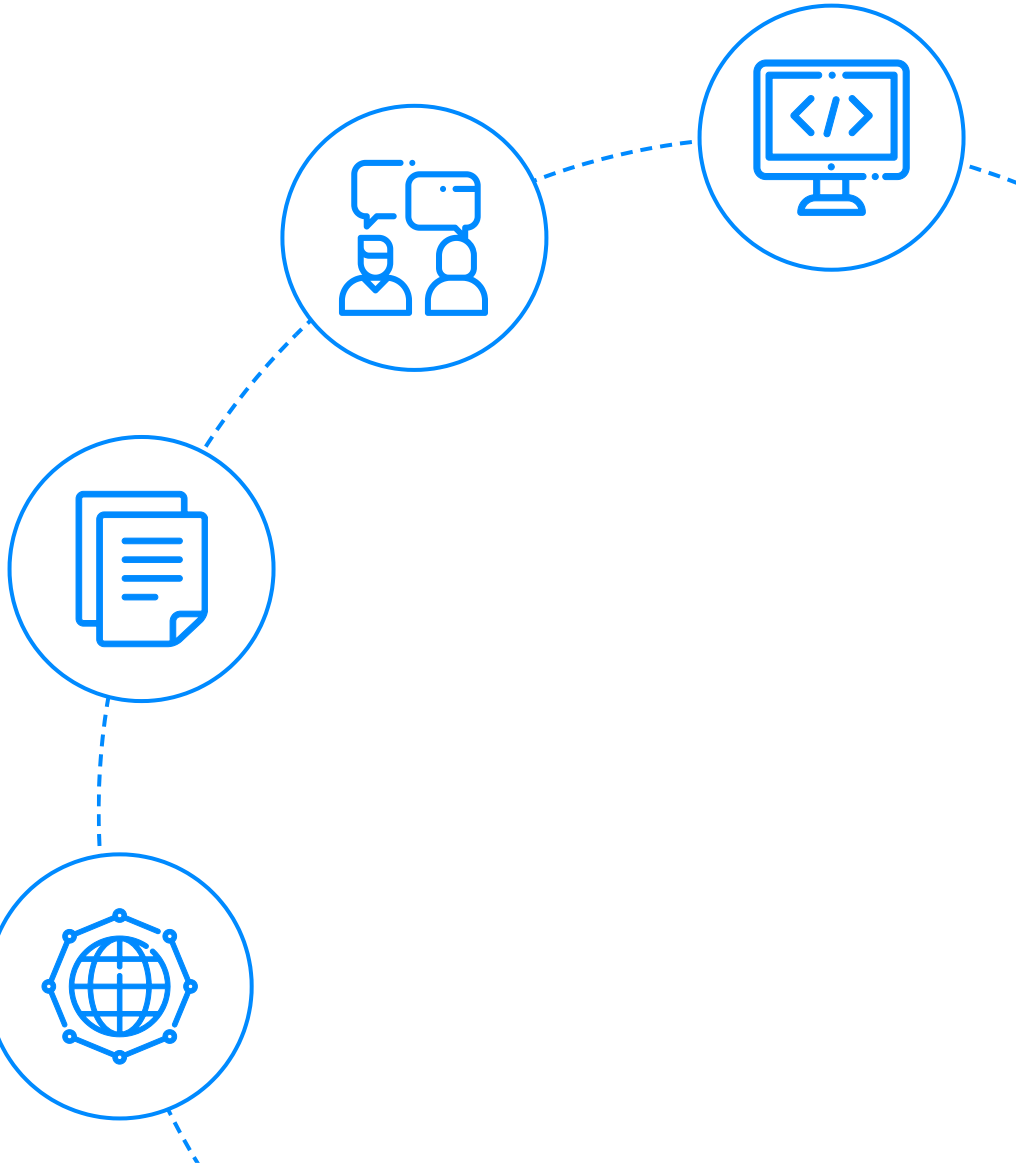
InterviewBit

Django Interview Questions



To view the live version of the page, click here.

Contents

Basic Django Interview Questions

Intermediate Django Interview Questions

Intermediate Django Interview Questions (.....Continued)

19. What databases are supported by Django?
20. What's the use of a session framework?
21. What's the use of Middleware in Django?
22. What is context in the Django?
23. What is django.shortcuts.render function?
24. What's the significance of the settings.py file?
25. How to view all items in the Model?
26. How to filter items in the Model?

Advanced Django Interview Questions

27. How to use file-based sessions?
28. What is mixin?
29. What is Django Field Class?
30. Why is permanent redirection not a good option?
31. Difference between Django OneToOneField and ForeignKey Field?
32. How can you combine multiple QuerySets in a View?
33. How to get a particular item in the Model?
34. How to obtain the SQL query from the queryset?
35. What are the ways to customize the functionality of the Django admin interface?
36. Difference between select_related and prefetch_related?

Advanced Django Interview Questions

(.....Continued)

37. Explain Q objects in Django ORM?
38. What are Django exceptions?

Let's get Started

Django (named after the Django Reinhardt) is a high-level python-based free and open-source web framework that follows the model-view-template(MVT) architectural pattern. As of now, the framework is maintained by Django Software Foundation (DSF), an independent organization based in the US and established as a 501(c)(3) non-benefit.

It was created in the fall of 2003 when the web programmers at the Lawrence Journal-World newspaper, Adrian Holovaty and Simon Willson. Began using python to build applications. Two years later, in July of 2005, it was released to the public under a BSD license. Later, in June 2008, the fledgling Django Software Foundation(DSF) took over the development of Django.

List of popular firms using the Django

PBS
Instagram
Mozilla
The Washington Times
Disqus, Bitbucket
NextDoor

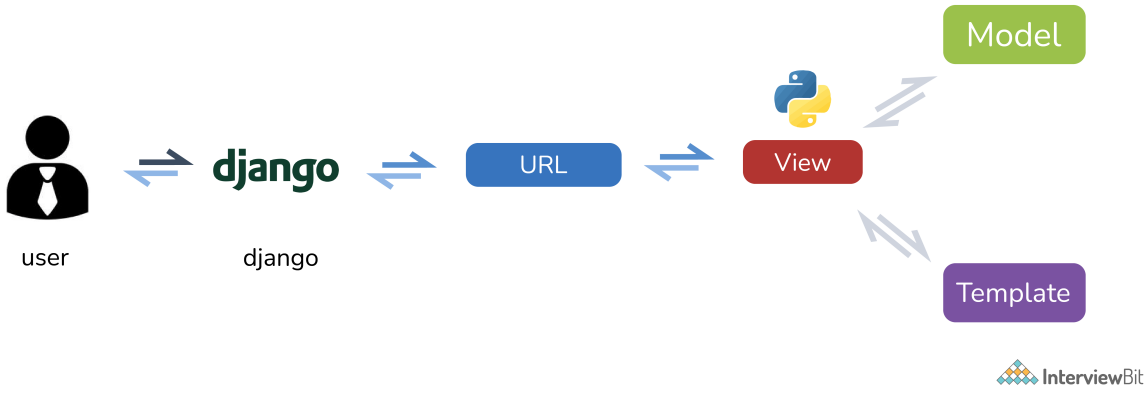
Advantages of using Django:

- Rich Ecosystem: It comes with numerous third-party apps which can be easily integrated as per the requirements of the project.
- Maturity: Django has been in use for over a decade. In the time frame, a lot of features are added and enhanced to make it a Robust framework. Apart from that, there are a large number of developers who are using Django.
- Admin panel: Django provides an admin dashboard that we can use to do basic CRUD operations over the models.
- Plugins: Allow programmers to add various features to applications and leave sufficient space for customization.
- Libraries: Due to the large development community there is an ample number of libraries for every task.
- ORM: It helps us with working with data in a more object-oriented way.

Basic Django Interview Questions

1. Explain Django Architecture?

Django follows the MVT (Model View Template) pattern which is based on the Model View Controller architecture. It's slightly different from the MVC pattern as it maintains its own conventions, so, the controller is handled by the framework itself. The template is a presentation layer. It is an HTML file mixed with Django Template Language (DTL). The developer provides the model, the view, and the template then maps it to a URL, and finally, Django serves it to the user.



2. Explain the django project directory structure?

- manage.py - A command-line utility that allows you to interact with your Django project
- __init__.py - An empty file that tells Python that the current directory should be considered as a Python package
- settings.py - Comprises the configurations of the current project like DB connections.
- urls.py - All the URLs of the project are present here
- wsgi.py - This is an entry point for your application which is used by the web servers to serve the project you have created.

3. What are models in Django?

A model in Django refers to a class that maps to a database table or database collection. Each attribute of the Django model class represents a database field. They are defined in app/models.py

Example:

```
from django.db import models
class SampleModel(models.Model):
field1 = models.CharField(max_length = 50)
field2 = models.IntegerField()
class Meta:
db_table = “sample_model”
```

Every model inherits from django.db.models.Model

Our example has 2 attributes (1 char and 1 integer field), those will be in the table fields.

The metaclass helps you set things like available permissions, singular and plural versions of the name, associated database table name, whether the model is abstract or not, etc.

To get more information about models you can refer here: https://docs.djangoproject.com/en/3.1/topics/db/models/.

4. What are templates in Django or Django template language?

Templates are an integral part of the Django MVT architecture. They generally comprise HTML, CSS, and js in which dynamic variables and information are embedded with the help of views. Some constructs are recognized and interpreted by the template engine. The main ones are variables and tags.

A template is rendered with a context. Rendering just replaces variables with their values, present in the context, and processes tags. Everything else remains as it is.

The syntax of the Django template language includes the following four constructs :

- Variables
- Tags
- Filters
- Comments

To read more about templates you can refer to this: https://docs.djangoproject.com/en/3.1/topics/templates/

5. What are views in Django?

A view function, or “view” for short, is simply a Python function that takes a web request and returns a web response. This response can be HTML contents of a web page, or a redirect, or a 404 error, or an XML document, or an image, etc.

Example:

```
from django.http import HttpResponse
def sample_function(request):
 return HttpResponse(“Welcome to Django”)
```

There are two types of views:

- **Function-Based Views:** In this, we import our view as a function.
- **Class-based Views**: It’s an object-oriented approach.

6. What is Django ORM?

This ORM (an acronym for Object Relational Mapper) enables us to interact with databases in a more pythonic way like we can avoid writing raw queries, it is possible to retrieve, save, delete and perform other operations over the database without ever writing any SQL query. It works as an abstraction layer between the models and the database.

7. Define static files and explain their uses?

Websites generally need to serve additional files such as images. Javascript or CSS. In Django, these files are referred to as “static files”, Apart from that Django provides `django.contrib.staticfiles` to manage these static files.

8. What is Django Rest Framework(DRF)?

Django Rest Framework is an open-source framework based upon Django which lets you create RESTful APIs rapidly.

9. What is django-admin and manage.py and explain its commands?

django-admin is Django's command-line utility for administrative tasks. In addition to this, a manage.py file is also automatically created in each Django project. Not only does it perform the same purpose as the django-admin but it also sets the `DJANGO_SETTINGS_MODULE` environment variable to point to the project's settings.py file.

- django-admin help - used to display usage information and a list of the commands provided by each application.
- django-admin version - used to check your Django version.
- django-admin check - used to inspect the entire Django project for common problems.
- django-admin compilemessages - Compiles .po files created by makemessages to .mo files for use with the help of built-in gettext support.
- django-admin createcachetable - Creates the cache tables for use in the database cache backend.
- django-admin dbshell - Runs the command-line client for the database engine specified in your ENGINE setting(s), with the connection parameters (USER, PASSWORD, DB_NAME, USER etc.) specified settings file.
- django-admin diffsettings - Shows the difference between the existing settings file and Django's default settings.
- django-admin dumpdata - Used to the dumpdata from the database.
- django-admin flush - Flush all values from the database and also re-executes any post-synchronization handlers specified in the code.
- django-admin inspectdb - It generates django models from the existing database tables.
- django-admin loaddata - loads the data into the database from the fixture file.
- django-admin makemessages - Used for translation purpose and it generates a message file too.
- django-admin makemigrations - Generates new migrations as per the changes detected to your models.
- django-admin migrate - Executes SQL commands after which the database state with the current set of models and migrations are synchronized.
- django-admin runserver - Starts a light-weight Web server on the local machine for development. The default server runs on port 8000 on the IP address 127.0.0.1. You can pass a custom IP address and port number explicitly if you want.
- django-admin sendtestemail - This is used to confirm email sending through Django is working by sending a test email to the recipient(s) specified.
- django-admin shell - Starts the Python interactive interpreter.
- django-admin showmigrations - Shows all migrations present in the project.
- django-admin sqlflush - Prints the SQL statements that would be executed for the flush command mentioned above.
- django-admin sqlmigrate - Prints the SQL statement for the named migration.

10. What is Jinja templating?

Jinja Templating is a very popular templating engine for Python, the latest version is Jinja2.

Some of its features are:

- Sandbox Execution - This is a sandbox (or a protected) framework for automating the testing process
- HTML Escaping - It provides automatic HTML Escaping as <, >, & characters have special values in templates and if using a regular text, these symbols can lead to XSS Attacks which Jinja deals with automatically.
- Template Inheritance
- Generates HTML templates much faster than default engine
- Easier to debug as compared to the default engine.

11. What are Django URLs?

URLs are one of the most important parts of a web application and Django provides you with an elegant way to design your own custom URLs with help of its module known as URLconf (URL Configuration). The basic functionality of this python module is to
You can design your own URLs in Django in the way you like and then map them to the python function (View function). These URLs can be static as well as dynamic. These URLs as present in the urls.py where they are matched with the equivalent view function.

Basic Syntax:

```
from django.urls import path
from . import views
urlpatterns = [
   path('data/2020/', views.data_2020),
   path('data/<int:year>/', views.data_year)
]
```

12. What is the difference between a project and an app in Django?

In simple words Project is the entire Django application and an app is a module inside the project that deals with one specific use case.
For eg, payment system(app) in the eCommerce app(Project).

13. What are different model inheritance styles in the Django?

- **Abstract Base Class Inheritance**: Used when you only need the parent class to hold information that you don't want to write for each child model.
- **Multi-Table Model Inheritance:** Used when you are subclassing an existing model and need each model to have its own table in the database.
- **Proxy Model Inheritance:** Used when you want to retain the model's field while altering the python level functioning of the model.

Intermediate Django Interview Questions

14. What are Django Signals?

Whenever there is a modification in a model, we may need to trigger some actions. Django provides an elegant way to handle these in the form of signals. The signals are the utilities that allow us to associate events with actions. We can implement these by developing a function that will run when a signal calls it.

List of built-in signals in the models:

Signals	Description
django.db.models.pre_init & django.db.models.post_init	Sent before or after a models's _init_() method is called
django.db.models.signals.pre_save & django.db.models.signals.post_save	Sent before or after a model's save() method is called
django.db.models.signals.pre_delete & django.db.models.signals.post_delete	Sent before or after a models' delete() method or queryset delete() method is called
django.db.models.signals.m2m_changed	Sent when a ManyToManyField is changed
django.core.signals.request_started & django.core.signals.request_finished	Sent when an HTTP request is started or finished

15. Explain the caching strategies in the Django?

Caching refers to the technique of storing the output results when they are processed initially so that next time when the same results are fetched again, instead of processing again those already stored results can be used, which leads to faster accessing as well us less resource utilization. Django provides us with a robust cache system that is able to store dynamic web pages so that these pages don't need to be evaluated again for each request.

Some of the caching strategies in Django are listed below:

Strategy	Description
Memcached	A memory-based cache server is the fastest and most efficient
FileSystem Caching	Values of the cache are stored as separate files in a serialized order
Local-memory Caching	This is used as the default cache strategy by Django if you haven't set anything. It is per-process as well as thread-safe.
Database Caching	Cache data will be stored in the database and works very well if you have a fast and well-indexed DB server.

16. Explain user authentication in Django?

Django comes with a built-in user authentication system, which handles objects like users, groups, user-permissions, and few cookie-based user sessions. Django User authentication not only authenticates the user but also authorizes him.

The system consists and operates on these objects:

- Users
- Permissions
- Groups
- Password Hashing System
- Forms Validation
- A pluggable backend system

17. How to configure static files?

Ensure that `django.contrib.staticfiles` is added to your `INSTALLED_APPS`

In your settings file. define STATIC_URL for ex.

```
STATIC_URL = '/static/'
```

In your Django templates, use the static template tag to create the URL for the given relative path using the configured STATICFILES_STORAGE.

```
{% load static %}
<img src="{% static 'my_sample/abcxy.jpg' %}" alt="ABC image">
```

Store your static files in a folder called `static` in your app. For example

```
my_sample/static/my_sample/abcxy.jpg
```

18. Explain Django Response lifecycle?

Whenever a request is made to a web page, Django creates an HttpRequest object that contains metadata about the request. After that Django loads the particular view, passing the HttpRequest as the first argument to the view function. Each view will be returning an HttpResponse object.
On the big picture following steps occur when a request is received by Django:

1. First of the Django settings.py file is loaded which also contain various middleware classes (`MIDDLEWARES`)
2. The middlewares are also executed in the order in which they are mentioned in the `MIDDLEWAREST`
3. From here on the request is now moved to the URL Router, who simply gets the URL path from the request and tries to map with our given URL paths in the urls.py.
4. As soon as it has mapped, it will call the equivalent view function, from where an equivalent response is generated
5. The response also passes through the response middlewares and send back to the client/browser.

19. What databases are supported by Django?

PostgreSQL and MySQL, SQLite and Oracle. Apart from these, Django also supports databases such as ODBC, Microsoft SQL Server, IBM DB2, SAP SQL Anywhere, and Firebird using third-party packages. Note: Officially Django doesn't support any no-SQL databases.

20. What's the use of a session framework?

Using the session framework, you can easily store and retrieve arbitrary data based on the pre-site-visitors. It stores data on the server-side and takes care of the process of sending and receiving cookies. These cookies just consist of a session ID, not the actual data itself unless you explicitly use a cookie-based backend.

21. What's the use of Middleware in Django?

Middleware is something that executes between the request and response. In simple words, you can say it acts as a bridge between the request and response. Similarly In Django when a request is made it moves through middlewares to views and data is passed through middleware as a response.

22. What is context in the Django?

Context is a dictionary mapping template variable name given to Python objects in Django. This is the general name, but you can give any other name of your choice if you want.

23. What is django.shortcuts.render function?

When a view function returns a webpage as HttpResponse instead of a simple string, we use render(). Render function is a shortcut function that lets the developer easily pass the data dictionary with the template. This function then combines the template with a data dictionary via templating engine. Finally, this render() returns as HttpResponse with the rendered text, which is the data returned by models. Thus, Django render() bypasses most of the developer's work and lets him use different template engines.
The basic syntax:

```
render(request, template_name, context=None, content_type=None, status=None, using=None)
```

The request is the parameter that generates the response. The template name is the HTML template used, whereas the context is a dict of the data passed on the page from the python. You can also specify the content type, the status of the data you passed, and the render you are returning.

24. What's the significance of the settings.py file?

As the name suggests this file stores the configurations or settings of our Django project, like database configuration, backend engines, middlewares, installed applications, main URL configurations, static file addresses, templating engines, main URL configurations, security keys, allowed hosts, and much more.

25. How to view all items in the Model?

```
ModelName.objects.all()
```

26. How to filter items in the Model?

```
ModelName.objects.filter(field_name="term")
```

Advanced Django Interview Questions

27. How to use file-based sessions?

To use the same, you need to set the `SESSION_ENGINE` settings to

```
"django.contrib.sessions.backends.file"
```

Mixin is a type of multiple inheritances wherein you can combine behaviors and attributes of more than one parent class. It provides us with an excellent way to reuse code from multiple classes. One drawback of using these mixins is that it becomes difficult to analyze what a class is doing and which methods to override in case of its code being too scattered between multiple classes.

29. What is Django Field Class?

'Field' refers to an abstract class that represents a column in the database table. The Field class is just a subclass of RegisterLookupMixin. In Django, these fields are used to create database tables (db_types()) which are used to map Python types to the database using get_prep_value() and the other way round using from_db_value() method. Therefore, fields are fundamental pieces in different Django APIs such as models and querysets.

30. Why is permanent redirection not a good option?

Permanent redirection is used only when you don't want to lead visitors to the old URLs. The response of the permanent redirections is cached by the browser so when you try to redirect to something else it will cause issues. Since this is a browser-side operation if your user wants to move to a new page it will load the same page.

31. Difference between Django OneToOneField and ForeignKey Field?

Both of them are of the most common types of fields used in Django. The only difference between these two is that ForeignKey field consists of on_delete option along with a model's class because it's used for many-to-one relationships while on the other hand, the OneToOneField, only carries out a one-to-one relationship and requires only the model's class.

32. How can you combine multiple QuerySets in a View?

Initially, Concatenating QuerySets into lists is believed to be the easiest approach. Here's an example of how to do that:

```
from itertools import chain
  result_list = list(chain(model1_list, model2_list, model3_list))
```

33. How to get a particular item in the Model?

```
ModelName.objects.get(id="term")
```

Note: If there are no results that match the query, get() will raise a **DoesNotExist** exception. If more than one item matches the given get() query. In this case, it'll raise **MultipleObjectsReturned**, which is also an attribute of the model class itself.

34. How to obtain the SQL query from the queryset?

```
print(queryset.query)
```

35. What are the ways to customize the functionality of the Django admin interface?

There are multiple ways to customize the functionality of the Django admin interface. You can piggyback on top of an add/change form that's automatically generated by Django, you can add JavaScript modules using the js parameter. This parameter is basically a list of URLs that point to the JavaScript modules that are to be included in your project within a <script> tag. You can also write views for the admin if you want.

36. Difference between select_related and prefetch_related?

Though both the functions are used to fetch the related fields on a model but their functioning is bit different from each other. In simple words, select_related uses a foreign key relationship, i.e. using join on the query itself while on the prefetch_related there is a separate lookup and the joining on the python side. Let's try to illustrate this via an example:

```
from django.db import models
class Country(models.Model):
    country_name = models.CharField(max_length=5)
class State(models.Model):
    state_name = models.CharField(max_length=5)
    country = model.ForeignKey(Country)
>> states = State.objects.select_related('country').all()
>> for state in states:
…   print(state.state_name)
```Query Executed
SELECT state_id, state_name, country_name FROM State INNER JOIN Country ON (State.count
```
>> country = Country.objects.prefetch_related('state').get(id=1)
>> for state in country.state.all():
…   print(state.state_name)
```Query Executed
SELECT id, country_name FROM country WHERE id=1;
SELECT state_id, state_name WHERE State WHERE country_id IN (1);
```
```

37. Explain Q objects in Django ORM?

Q objects are used to write complex queries, as in filter() functions just `AND` the conditions while if you want to `OR` the conditions you can use Q objects. Let's see an example:

```
from django.db import models
from django.db.models import Q
>> objects = Models.objects.get(
   Q(tag__startswith='Human'),
   Q(category='Eyes') | Q(category='Nose')
)
```Query Executed
SELECT * FROM Model WHERE tag LIKE 'Human%' AND (category='Eyes' OR category='Nose')
```
```

38. What are Django exceptions?

In addition to the standard Python exceptions, Django raises of its own exceptions.List of the exceptions by Django (https://docs.djangoproject.com/en/3.1/ref/exceptions/)

Important Resources:

Django Projects

Node.js vs Django

Flask Vs Django

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions