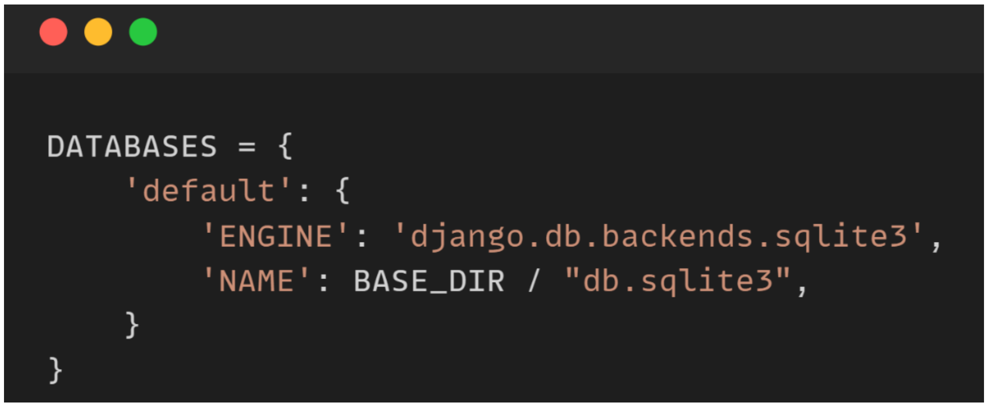
1. How do you set up a database connection in Django?

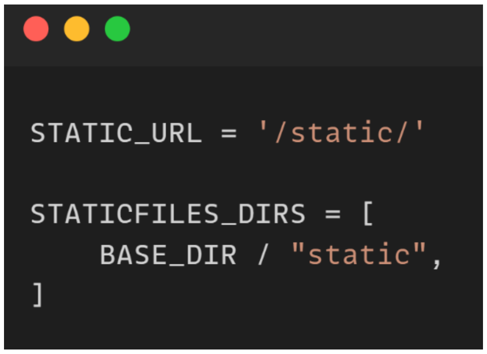
To set up a database connection in Django, we configure the DATABASES setting in settings.py. Specify the database engine, name, and other connection details. For example, to use SQLite:



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}
```

2. How can you serve static files during development in Django?

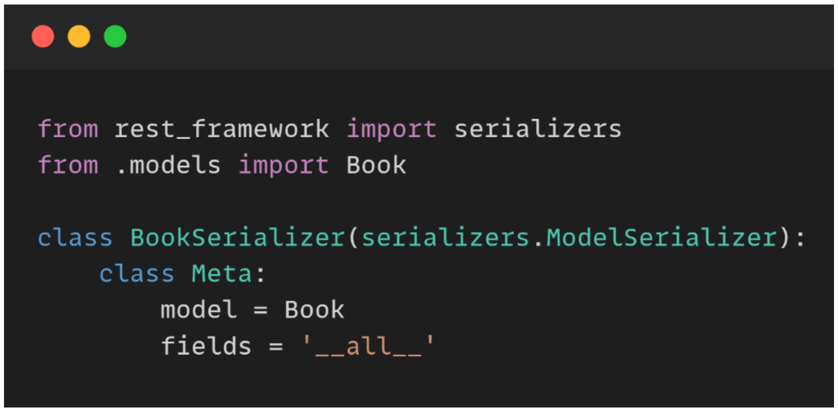
To serve static files during development in Django, we can configure the STATICFILES_DIRS setting in settings.py to specify additional directories where Django should look for static files. Here’s how to set it up:



```
STATIC_URL = '/static/'

STATICFILES_DIRS = [
    BASE_DIR / "static",
]
```

3. What is Django REST Framework (DRF)?

In Django, DRF stands for Django Rest Framework, a powerful toolkit for building Web APIs. It simplifies the process of creating RESTful APIs by providing features like serialization, authentication, and view sets, allowing developers to create robust APIs quickly and efficiently. For example:

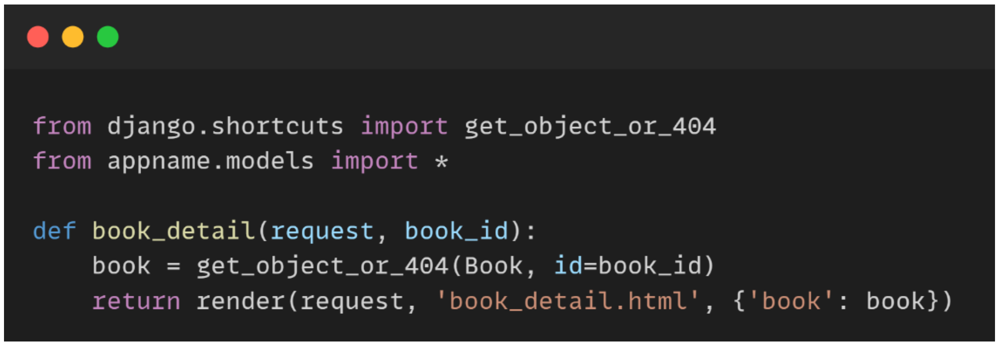


```
from rest_framework import serializers
from .models import Book

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = '__all__'
```

4. What is get_objects_or_404 in Django?

In Django, get_object_or_404 is a shortcut function that retrieves a single object from the database based on a given model and query parameters. If the object is found, it returns it; if not, it raises an Http404 exception, which renders a 404 error page.

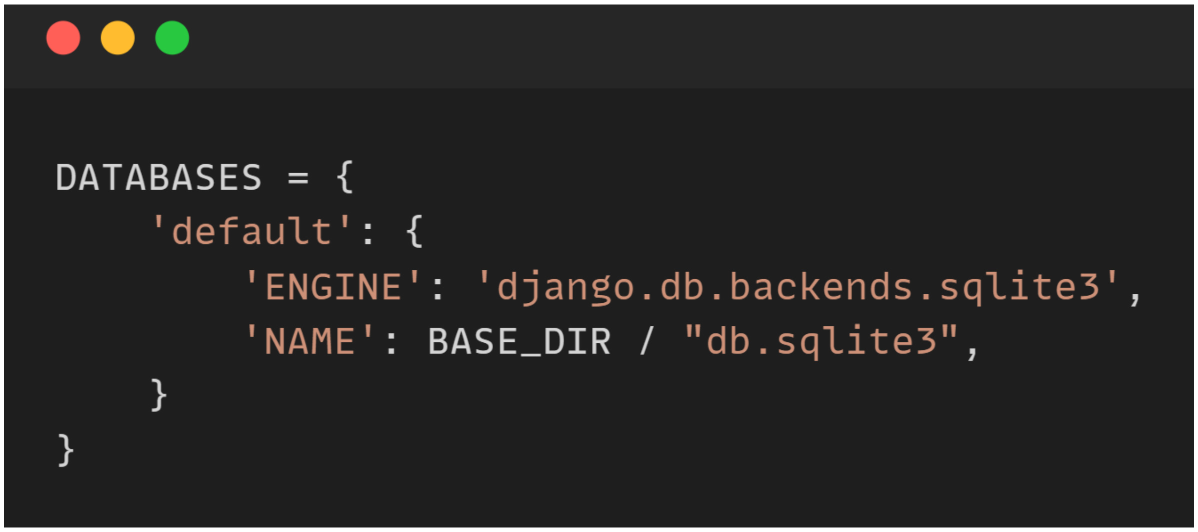
Here's a basic example:



```
from django.shortcuts import get_object_or_404
from appname.models import *

def book_detail(request, book_id):
    book = get_object_or_404(Book, id=book_id)
    return render(request, 'book_detail.html', {'book': book})
```

5. What is the purpose of settings.py in a Django project?

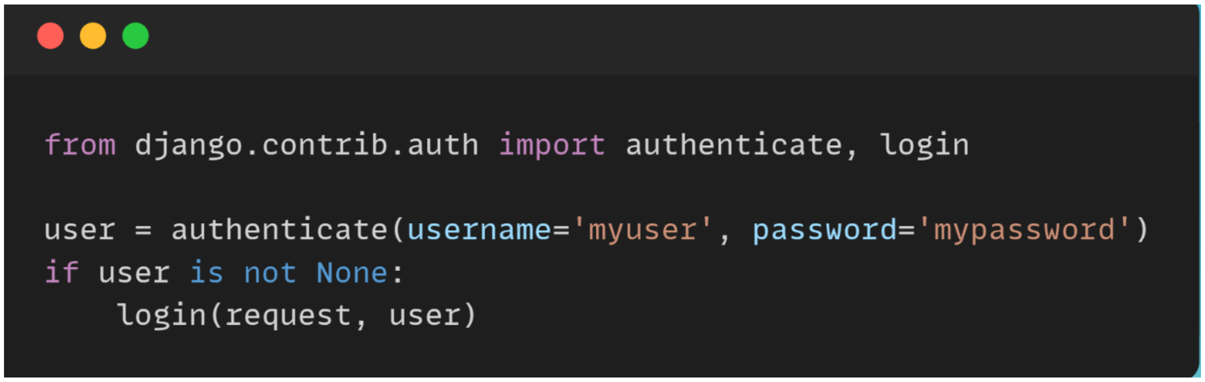
The settings.py file in a Django project configures key aspects of the application, including database connections, security settings, installed apps, and middleware. For example, it specifies the database engine and name:



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}
```

6. How does Django’s authentication system work?

Django’s authentication system provides a way to manage users, including login, logout, and permission checks. It uses a built-in User model to handle user accounts. For example, to log in a user, we can use the authenticate() and login() functions:

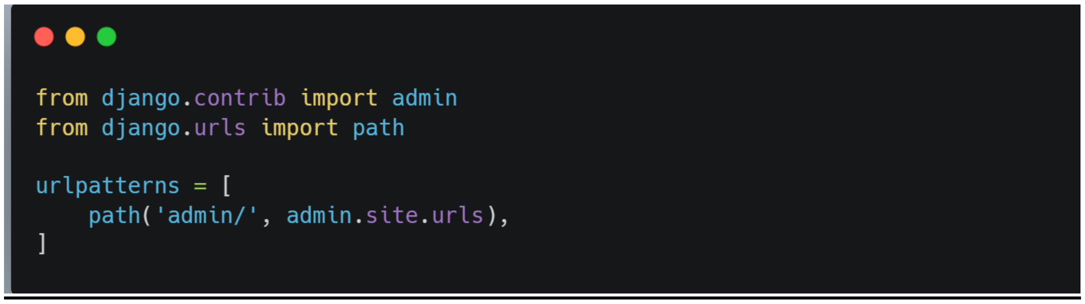


```python
from django.contrib.auth import authenticate, login

user = authenticate(username='myuser', password='mypassword')
if user is not None:
    login(request, user)
```

7. What is Django Admin, how is it used?

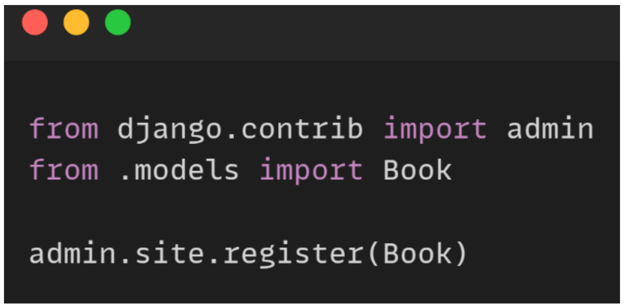
Django admin site is a built-in web-based interface that provides a powerful and customizable administrative interface for managing a Django web application. To use the Django admin site, add the URL path for the admin panel in your urls.py file, register your models in admin.py, and create a superuser to access it.

urls.py



```python
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

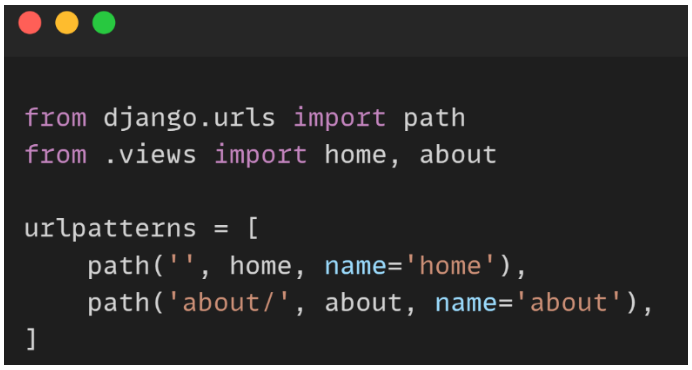
admin.py



```python
from django.contrib import admin
from .models import Book

admin.site.register(Book)
```

8. What is the use of the urls.py file in Django?

The urls.py file in Django is used to define URL patterns for routing incoming web requests to the appropriate views. It maps URL paths to their corresponding view functions or classes, enabling clean and organized URL handling. For example:

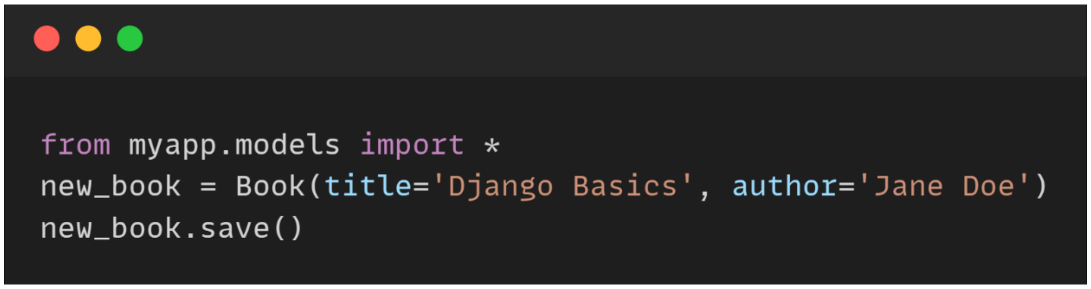


```python
from django.urls import path
from .views import home, about

urlpatterns = [
    path('', home, name='home'),
    path('about/', about, name='about'),
]
```

9. What is Django ORM? What database operations can be performed using Django's ORM, give example?

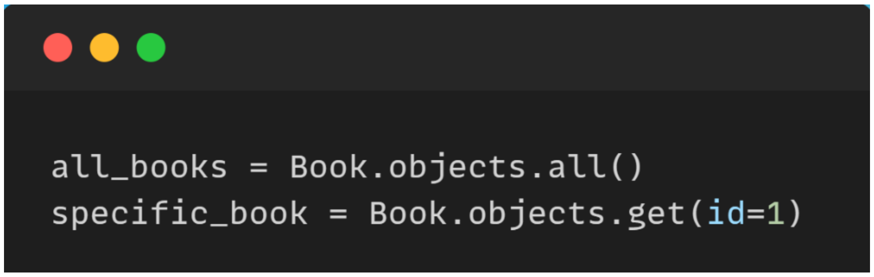
Django ORM (Object-Relational Mapping) is a feature that allows developers to interact with the database using Python code. It translates Python objects into database tables and allows for easy querying and manipulation of data.

Creating records:



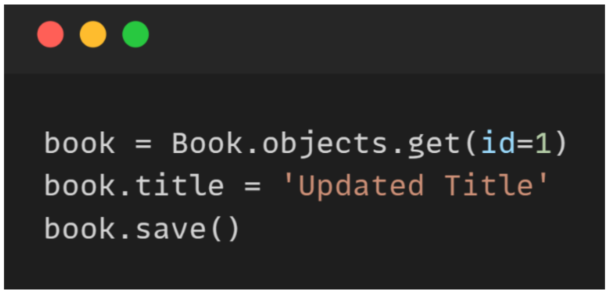
```python
from myapp.models import *
new_book = Book(title='Django Basics', author='Jane Doe')
new_book.save()
```

Reading records:



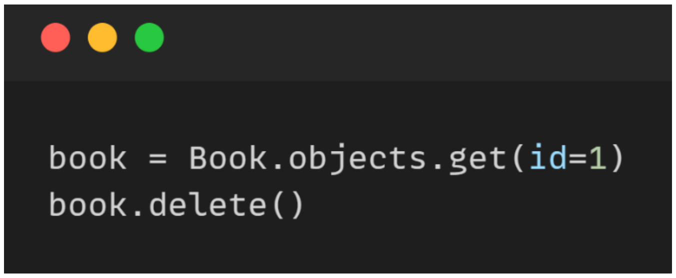
```python
all_books = Book.objects.all()
specific_book = Book.objects.get(id=1)
```

Updating records:



```
book = Book.objects.get(id=1)
book.title = 'Updated Title'
book.save()
```

Deleting records:

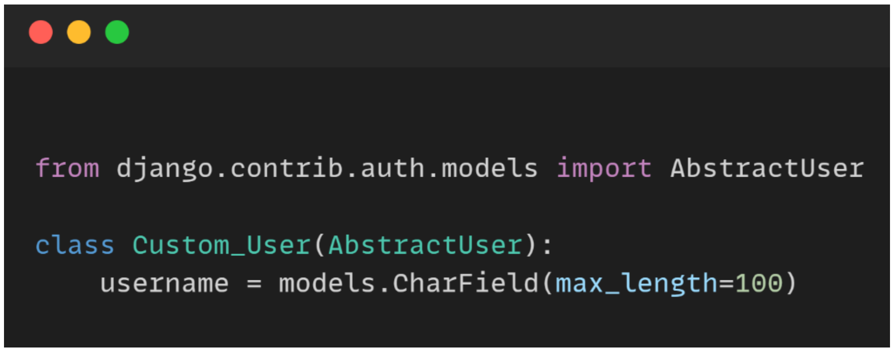


```
book = Book.objects.get(id=1)
book.delete()
```

10. How does Django's ORM help manage help manage relationships between models?

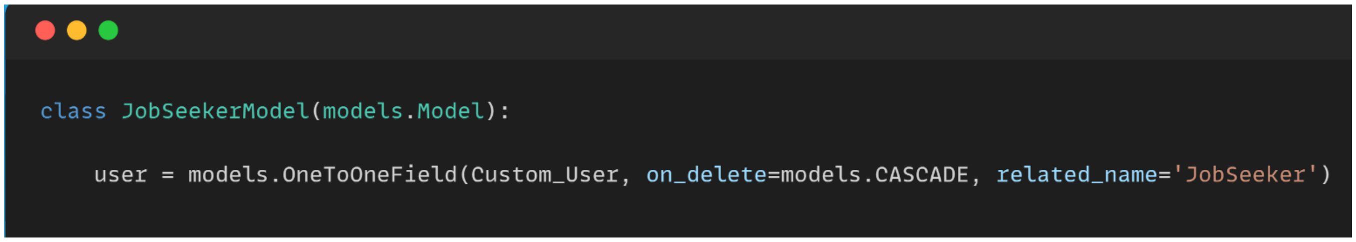
Django's ORM helps manage relationships between models through the use of fields that define how models are connected. The main types of relationships are:

models.py



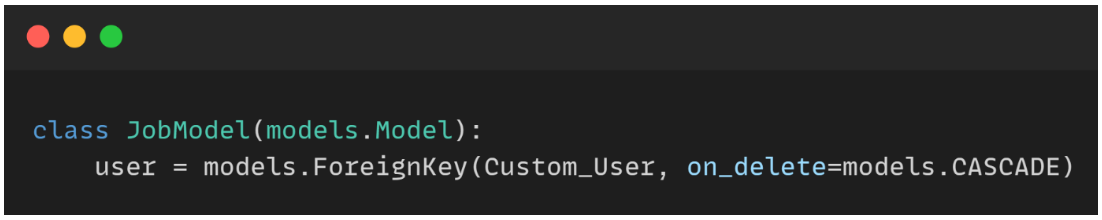
```
from django.contrib.auth.models import AbstractUser

class Custom_User(AbstractUser):
    username = models.CharField(max_length=100)
```

- **One to One:**



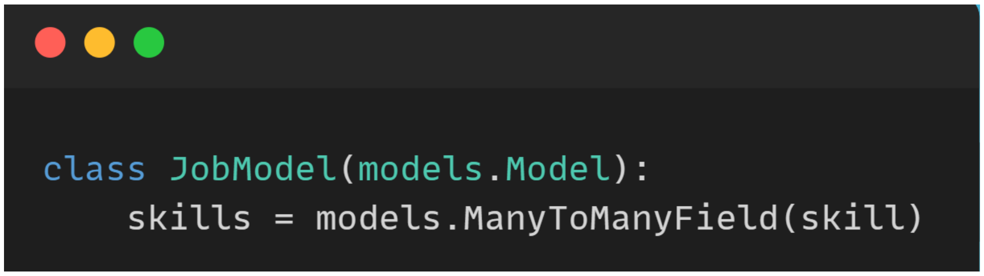
```
class JobSeekerModel(models.Model):

    user = models.OneToOneField(Custom_User, on_delete=models.CASCADE, related_name='JobSeeker')
```

- **One-to-Many:**



```
class JobModel(models.Model):
    user = models.ForeignKey(Custom_User, on_delete=models.CASCADE)
```

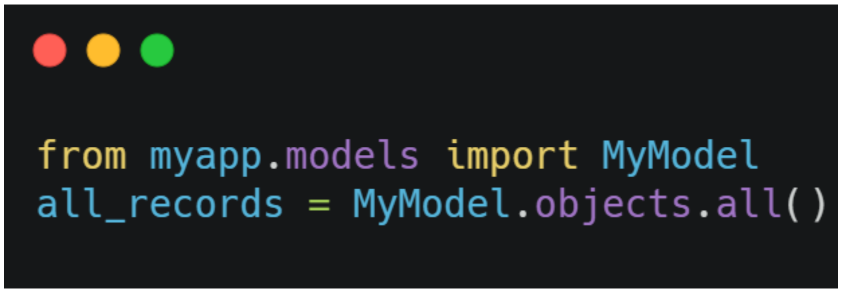
- **Many-to-Many:**



```
class JobModel(models.Model):
    skills = models.ManyToManyField(skill)
```

11. How do you retrieve all records of a model using Django's ORM?

To retrieve all records of a model using Django's ORM, use the **.all()** method on the model's objects:

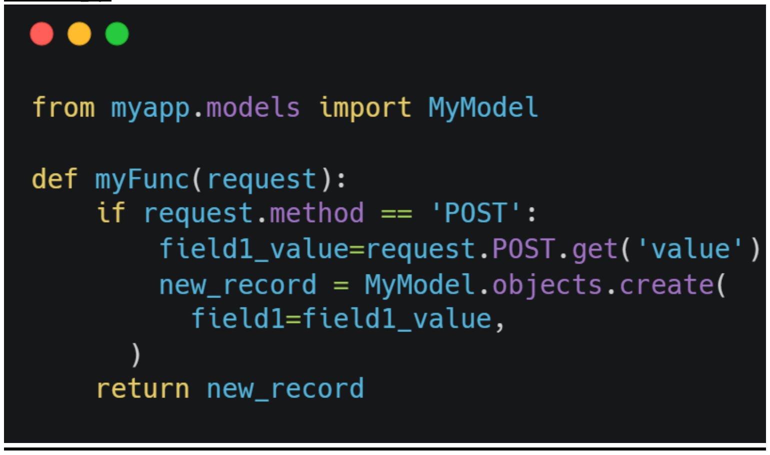


```
from myapp.models import MyModel
all_records = MyModel.objects.all()
```

12. How do you create a new record in the database using Django ORM?

To create a new record in the database using Django ORM, use the **.create()** method or save an instance of the model:

views.py



```
from myapp.models import MyModel

def myFunc(request):
    if request.method == 'POST':
        field1_value=request.POST.get('value')
        new_record = MyModel.objects.create(
          field1=field1_value,
      )
    return new_record
```

Or,

```
from myapp.models import MyModel

def myFunc(request):
    if request.method == 'POST':
        field1_value=request.POST.get('value')
        new_record = MyModel(
          field1=field1_value
      )
        new_record.save()
    return new_record
```

13. How can you delete multiple records in Django ORM? Explain the use of the delete() method with an example.

Here's an example of how to delete multiple records where age is less than the specified value in Django ORM:

```
from myapp.models import MyModel

def delete_records_less_than(age):
    MyModel.objects.filter(age__lt=age).delete()
```

14. What is a view in Django?

In Django, a view is a function or class that handles HTTP requests and returns an HTTP response. Views are the logic behind what gets displayed on a webpage.

Here's an example of a Django view that renders an HTML file:

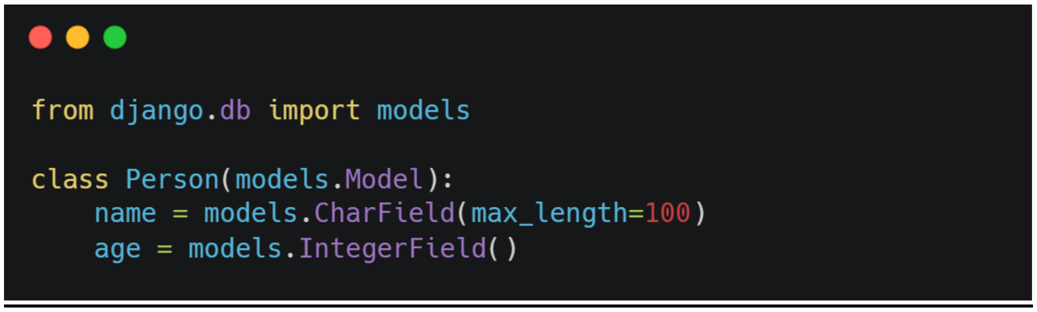
```
from django.shortcuts import render

def my_view(request):
    return render(request, 'my_template.html')
```

15. What is a model in Django?

In Django, a model is a Python class that defines the structure of database tables, including fields and behaviors.

Example:

models.py



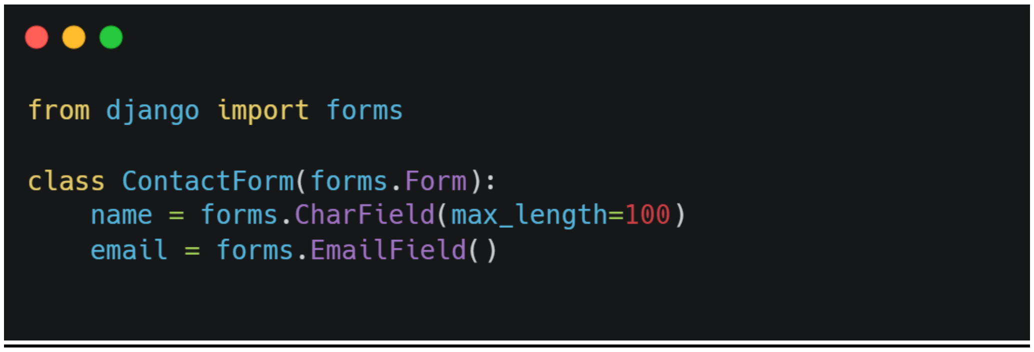
```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
```

16. How do you handle forms in Django?

In Django, forms are handled using the forms module, allowing us to create and validate user input easily. We define a form class by inheriting from **forms.Form** or **forms.ModelForm**, and then use it in views to process user input, and render the form in templates.

Example:

forms.py



```
from django import forms

class ContactForm(forms.Form):
    name = forms.CharField(max_length=100)
    email = forms.EmailField()
```

17. What is the use of manage.py in Django?

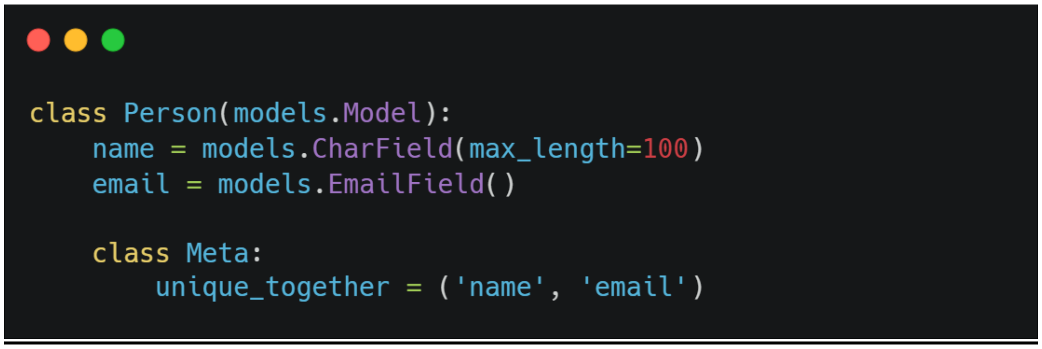
The **manage.py** file in Django is used to run administrative commands for our project. For example, we can start the development server with the command **python manage.py runserver** or **py manage.py runserver .**

18. What is the purpose of Meta class in a Django model?

The Meta class in a Django model is used to define metadata options, such as the database table name, ordering, and unique constraints.

Example:

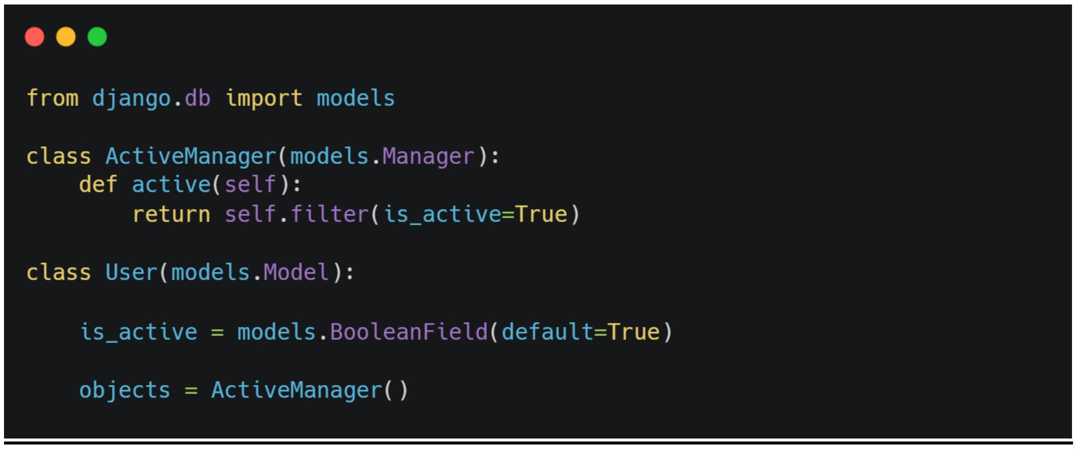
models.py



```
class Person(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()

    class Meta:
        unique_together = ('name', 'email')
```

19. How do you create a custom model manager in Django?

To create a custom model manager in Django, inherit from **models.Manager** and add custom methods. Then, set this manager as an attribute of our model. For example:
models.py

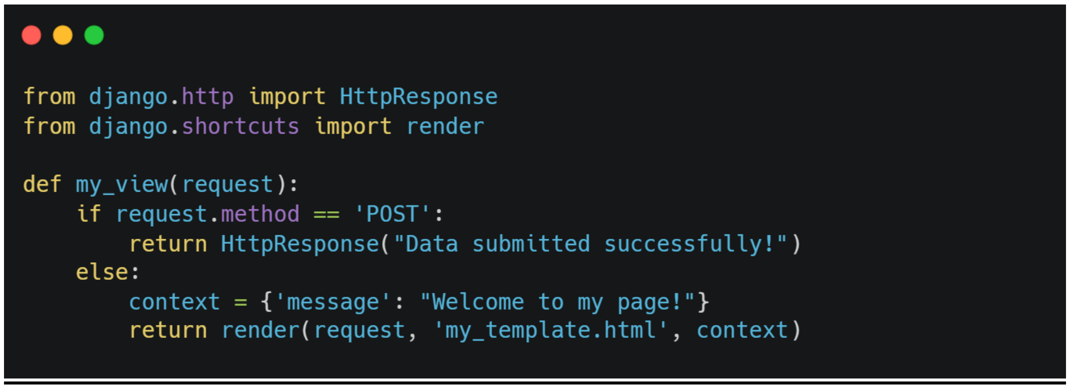


```
from django.db import models

class ActiveManager(models.Manager):
    def active(self):
        return self.filter(is_active=True)

class User(models.Model):

    is_active = models.BooleanField(default=True)

    objects = ActiveManager()
```

20. How do you create a function-based view in Django?

To create a function-based view in Django that can either render a template or return an HTTP response based on certain conditions, we can use both **render** and **HttpResponse**. Here's an example:

views.py

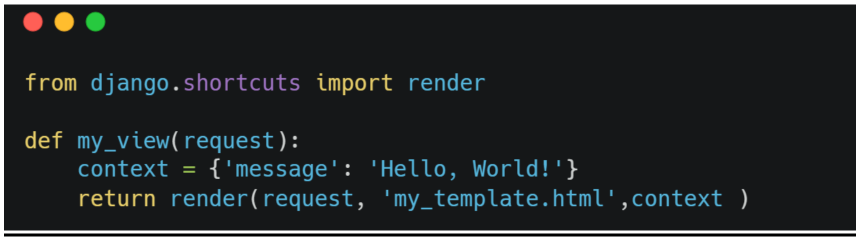


```python
from django.http import HttpResponse
from django.shortcuts import render

def my_view(request):
    if request.method == 'POST':
        return HttpResponse("Data submitted successfully!")
    else:
        context = {'message': "Welcome to my page!"}
        return render(request, 'my_template.html', context)
```

21. What is render() in Django views, and what does it do?

In Django, **render()** is a function that creates an HTML response by combining a template file with data from a context dictionary. It makes it easy to return HTML pages from views. For example:

views.py

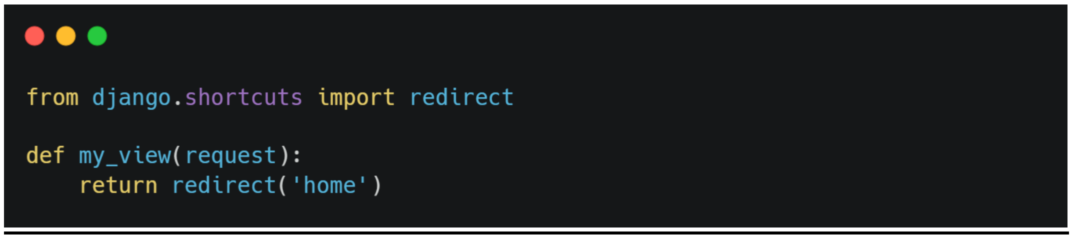


```python
from django.shortcuts import render

def my_view(request):
    context = {'message': 'Hello, World!'}
    return render(request, 'my_template.html',context )
```

22. What is the purpose of redirect() in Django views?

In Django views, the **redirect()** function is used to send the user to a different webpage. It helps change the URL the user sees after performing an action, like submitting a form. For example:

views.py

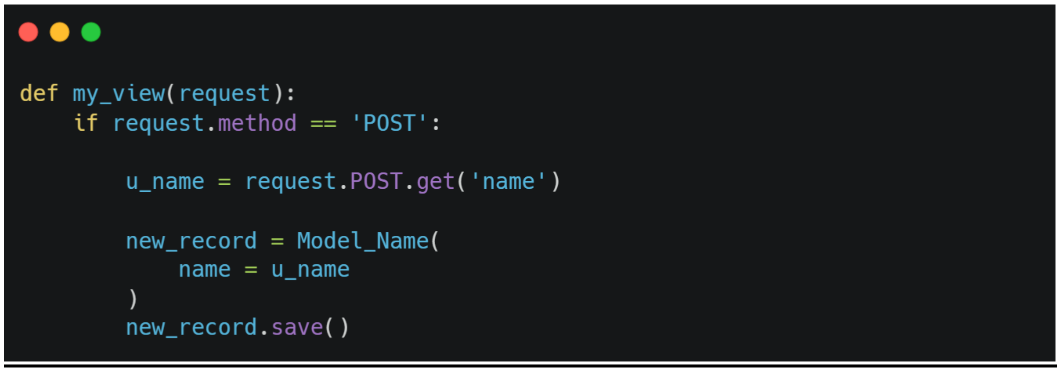


```python
from django.shortcuts import redirect

def my_view(request):
    return redirect('home')
```

23. How do you handle form submission in Django views?

To handle form submission in Django views, we check if the request method is **POST**, which indicates that the form has been submitted. Then, we can process the form data, validate it, and save it if it's valid. For example:

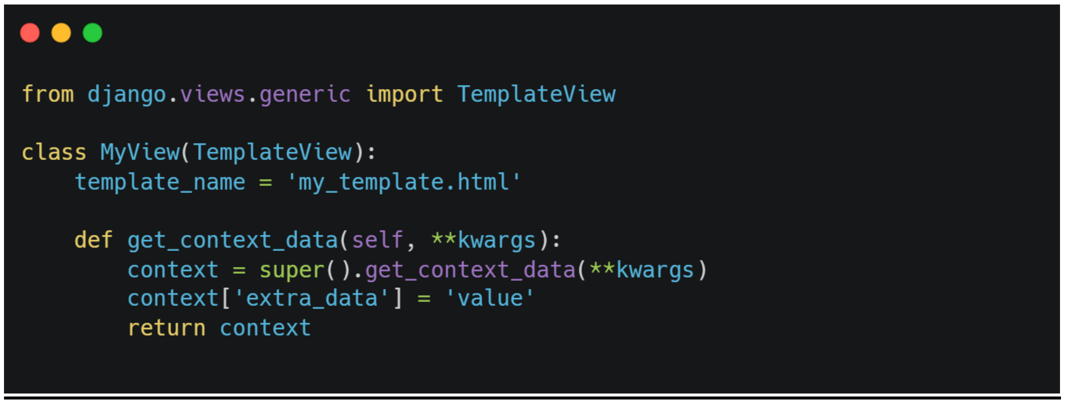
views.py



```python
def my_view(request):
    if request.method == 'POST':

        u_name = request.POST.get('name')

        new_record = Model_Name(
            name = u_name
        )
        new_record.save()
```

24. What is get_context_data() in a Django class-based view?

In a Django class-based view, **get_context_data()** is a method that provides data to be used in a template. For example:

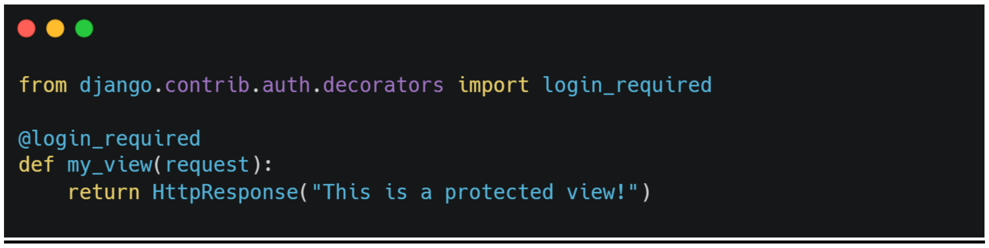


```python
from django.views.generic import TemplateView

class MyView(TemplateView):
    template_name = 'my_template.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['extra_data'] = 'value'
        return context
```

25. How do you create a view that only logged-in users can access?

To create a view that only logged-in users can access, use the **@login_required** decorator from **django.contrib.auth.decorators** above the view function.

views.py



```python
from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
    return HttpResponse("This is a protected view!")
```

26. How do you use a for loop in Django templates?

In Django templates, we can use a for loop with the **{% for %}** tag to iterate over a list or queryset and display each item. For example:

```
<ul>
    {% for item in item_list %}
        <li>{{ item }}</li>
    {% endfor %}
</ul>
```

27. What is a template inheritance in Django?

Template inheritance in Django allows us to create a base template that defines a common structure, which can be extended by child templates to include specific content. For example:

base.html

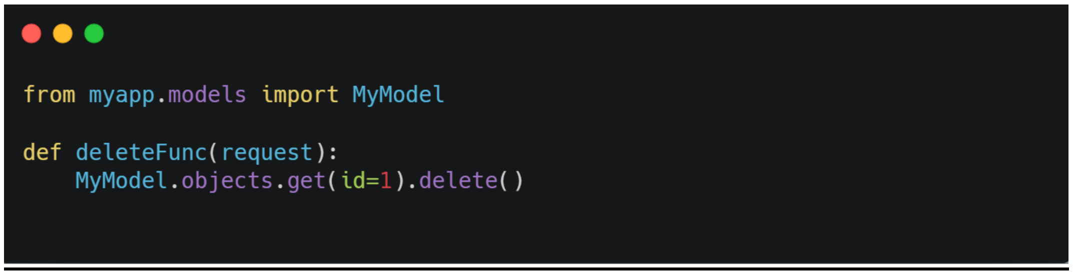
```
<!DOCTYPE html>
<html>
<head>
    <title>Simple Title</title>
</head>
<body>
      {% block body %}
      {% endblock %}

</body>
</html>
```

child.html

```
{% extends 'base.html' %}


{% block body %}
    <p>This is the content of the child page.</p>
{% endblock %}
```

28. How do you delete a record in Django ORM?

To delete a record in Django ORM, we can retrieve the object and call the **.delete()** method on it. For example:



```python
from myapp.models import MyModel

def deleteFunc(request):
    MyModel.objects.get(id=1).delete()
```