**Q1:What are the "makemigrations" and "migrate" commands used for? Give code examples**

The makemigrations command is used to create new migration files based on the changes made to Django models. These changes can include creating new models, adding or removing fields, or modifying existing fields.

| python manage.py makemigrations |
| --- |

The migrate command is used to apply and unapply the changes made in migration files to the database schema.

| python manage.py migrate |
| --- |

**Q2: How do you rollback a migration?**

Migrations can be reversed with migrate by passing the number of the previous migration. For example, to reverse migration books.0003:

| python manage.py migrate books 0002 |
| --- |

If you want to reverse all migrations applied for an app, use the name zero:

| python manage.py migrate books zero |
| --- |

**Q3: Give code examples of retrieving every item in a model or just one item by its key.**

**Retrieve all items:**

| from myapp.models import MyModel  items = MyModel.objects.all() |
| --- |

**Retrieve an item:**

| from myapp.models import MyModel item = MyModel.objects.get(pk=1) |
| --- |

**Q4: How would you go about executing a specific method prior to a Model's save() method execution? Give a code example.**

You can use the signals to call the method before saving the model’s save():

| ​​from django.db.models.signals import pre\_save from django.dispatch import receiver from myapp.models import MyModel  @receiver(pre\_save, sender=MyModel) def my\_model\_pre\_save(sender, instance, \*\*kwargs):  # Any operation you can perform before save. |
| --- |

Another way, you can override the save() method and write own functionality:

| class MyModel(models.Model):  def save(self, \*args, \*\*kwargs):  # Call your pre-processing method here  self.my\_method()  # Call the original `save()` method to save the model  super().save(\*args, \*\*kwargs)   def my\_method(self):  # Define your pre-processing method here |
| --- |

**Q5:Give a code example of displaying multiple Model rows in a template.**

| # views.py  from django.shortcuts import render from .models import Person  def persons\_list(request):  persons = Person.objects.all()  context = {'persons': persons}  return render(request, 'persons\_list.html', context) |
| --- |

| <!-- persons\_list.html -->  {% **extends** 'base.html' %}  {% **block** content %}  <**h1**>Persons List</**h1**>  <**ul**>  {% **for** person in persons %}  <**li**>{{ person.name }} ({{ person.age }})</**li**>  {% **endfor** %}  </**ul**> {% **endblock** %} |
| --- |

**Q6: Give an example of referencing static resources in a template**

Add the static folder path in setting.py file.

| ***{%*** **load** static ***%}***  <**img** src="{% **static** 'logo.png' %}" alt="My Logo"> |
| --- |

**Q7: How do you ensure a View is only accessible to authenticated users? Give a code**

**Example.**

In Django, you can use the login\_required decorator to ensure that a view is only accessible to authenticated users. Here's an example:

| from django.contrib.auth.decorators import login\_required from django.shortcuts import render  @login\_required def my\_view(request):  # Your view logic here  return render(request, 'my\_template.html') |
| --- |

**Q8: How do you validate a Form field's input before saving? Give a code example.**

| **class MyForm(forms.Form):  my\_field = forms.CharField()   def clean\_my\_field(self):  data = self.cleaned\_data['my\_field']  # Perform validation on the data  if not data.isalpha():  raise ValidationError('Input must be alphabetic')  return data** |
| --- |

**Q9: Explain what middleware is used for.**

Middleware is a way to add extra functionality to the request/response processing in Django. It's a series of hooks or functions that are executed before or after a view is called, which can modify the request or response. Middleware can be used for various purposes such as authentication, caching, logging, compression, and more.

**Q10: Describe how would you would construct a Model where some of the columns are dynamically defined at runtime?**

There are a few approaches:

* key/value model (To implement a key/value model, we can create a model with two fields: a key field and a value field. The key field will be used to store the attribute name, and the value field will store the value associated with that attribute. We can also add a foreign key to another model to establish a relationship between the two.)
* JSON data in a TextField