Django Cheat Sheet

A quick reference guide for Django, covering models, views, templates, forms, and common commands.

Models

Model Definition

Defining a model involves creating a Python class that inherits from `django.db.models.Model`.

Example:

```
from django.db import models

class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

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

Field Types

`CharField(max_length=n)`	For strings of limited length.
`TextField()`	For long text entries.
`IntegerField()`	For integer values.
`FloatField()`	For floating-point numbers.
`BooleanField()`	For boolean (True/False) values.
`DateField()`	For dates (year, month, and day).
`DateTimeField()`	For dates and times.
`ForeignKey(OtherModel)`	For many-to-one relationships.
`ManyToManyField(OtherModel)`	For many-to-many relationships.

Model Methods

`save()`	Saves the current instance.
`delete()`	Deletes the current instance.
`__str__()`	Returns a human-readable string representation of the object.

Views

Function-Based Views

Function-based views are Python functions that take a request object as input and return a response.

Example:

```
from django.shortcuts import render

def my_view(request):
    # Your view logic here
    return render(request, 'my_template.html', {'my_variable': 'Hello'})
```

Class-Based Views

Class-based views are Python classes that inherit from Django's view classes (e.g., `View`, `TemplateView`, `ListView`, `DetailView`).

Example:

```
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['my_variable'] = 'Hello'
        return context
```

Common Decorators

`@login_required`	Requires the user to be logged in.
`@permission_required('app.permission')`	Requires the user to have a specific permission.
`@transaction.atomic`	Ensures that the view is executed within a transaction.

Templates

Template Syntax

`{{ variable }}`	Outputs a variable.
`{% tag %}`	Executes a template tag.
`{# comment #}`	Template comment.

Common Tags

`{% for item in list %}` `{% endfor %}`	Loops through a list.
`{% if condition %}` `{% endif %}`	Conditional statement.
`{% extends 'base.html' %}`	Extends a parent template.
`{% include 'template.html' %}`	Includes another template.
`{% url 'view_name' %}`	Reverses a URL pattern.
`{% csrf_token %}`	CSRF protection token.

Filters

`{{ value|lower }}`	Converts a value to lowercase.
`{{ value|upper }}`	Converts a value to uppercase.
`{{ value|date:'F j, Y' }}`	Formats a date.
`{{ value|default:'Default Value' }}`	Provides a default value.
`{{ value|length }}`	Returns the length of a value.

Forms

Form Definition

Defining a form involves creating a Python class that inherits from `django.forms.Form` or `django.forms.ModelForm`.

Example:

```
from django import forms

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

Form Rendering

Forms can be rendered in templates using `{{ form.as_p }}`, `{{ form.as_ul }}`, or manually rendering each field.

Example:

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
</form>
```

Field Types

`CharField(max_length=n)`	For strings of limited length.
`EmailField()`	For email addresses.
`IntegerField()`	For integer values.
`BooleanField()`	For boolean (True/False) values.
`DateField()`	For dates (year, month, and day).
`ChoiceField(choices=CHOICES)`	For select inputs.