Django Cheatsheet

What is Django?

Django is a **high-level, Python-based web framework** that encourages rapid development, clean design, and pragmatic code. It follows **MVT (Model-View-Template)** architecture and comes with batteries included (ORM, authentication, admin, etc.).

Installation & Project Setup

Install Django:

```
pip install django
```

Check version:

```
django-admin --version
```

Creating a Project

```
django-admin startproject projectName
cd projectName
```

Starting the Development Server

```
python manage.py runserver
```

Server runs by default on `http://127.0.0.1:8000/`

You can specify a custom port:

```
python manage.py runserver 8080
```

Django MVT (Model-View-Template)

Sample Model

Models represent database tables. Always remember to add `()` to fields!

```
from django.db import models

class Product(models.Model):
    product_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=100)
    price = models.FloatField()
    created_at = models.DateTimeField(auto_now_add=True)

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

Sample View (views.py)

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Django CodeWithHarry Cheatsheet")
```

For templates:

```python
from django.shortcuts import render

def index(request):
    return render(request, "index.html", {"title": "Welcome"})
```

Sample HTML Template

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ title }}</title>
</head>
<body>
    <h1>This is a sample template file.</h1>
</body>
</html>
```

Views in Django

Views can be **Function-Based** or **Class-Based**.

Function-Based View

```python
from django.http import HttpResponse

def index(request):
    return HttpResponse("This is a function-based view")
```

Class-Based View

```
from django.views import View
from django.http import HttpResponse

class SimpleClassBasedView(View):
    def get(self, request):
        return HttpResponse("Hello from a class-based view")
```

URLs in Django

`urls.py` maps paths to views.

Example urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name='index'),
    path('about/', views.about, name='about'),
]
```

For modular apps, use `include()` :

```
from django.urls import include, path

urlpatterns = [
    path('community/', include('aggregator.urls')),
    path('contact/', include('contact.urls')),
]
```

Forms in Django

Example Form

```
from django import forms


class SampleForm(forms.Form):
    name = forms.CharField(max_length=50)
    description = forms.CharField(widget=forms.Textarea)
```

Apps in Django

Create a new app:

```
python manage.py startapp AppName
```

Register it in `settings.py`:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    ...
    'AppName',
]
```

Templates in Django

Configure `TEMPLATES` in `settings.py`:

```
TEMPLATES = [
    {
```

```
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / "templates"],
        'APP_DIRS': True,
        'OPTIONS': {},
    },
]
```

Rendering template in view:

```
from django.shortcuts import render

def index(request):
    return render(request, 'index.html', {"name": "Harry"})
```

Access variable in template:

```
<h1>Hello, {{ name }}</h1>
```

Migrations

Create migration files

```
python manage.py makemigrations
```

Apply migrations

```
python manage.py migrate
```

View SQL queries

```
python manage.py sqlmigrate appName 0001
```

Django Admin

Create admin user:

```
python manage.py createsuperuser
```

Register model in `admin.py`:

```
from django.contrib import admin
from .models import Product

admin.site.register(Product)
```

Page Redirection

```
from django.shortcuts import redirect

def redirecting(request):
    return redirect("https://www.codewithharry.com")
```

Additional Useful Commands

- **Create Django Shell** (interact with models):

  ```
  python manage.py shell
  ```

- **Collect static files for production**

  ```
  python manage.py collectstatic
  ```

- **Check for issues**

```
python manage.py check
```

Best Practices

- Use **virtual environments** for each project.
- Always commit your `requirements.txt` file:

```
pip freeze > requirements.txt
```

- Separate `settings.py` for dev & production (e.g., use `django-environ` for secrets).
- Use `.env` files to store sensitive information.
- Prefer **class-based views** for reusable code and scalability.