# Django Templates: Step-by-Step Anleitung

## 1. Einführung: Was sind Templates?

Templates in Django sind textbasierte Dateien, die dynamische Inhalte mithilfe der **Django Template Language (DTL)** generieren. Sie kombinieren statisches HTML mit dynamischen Inhalten wie Variablen, Schleifen und Bedingungen.

---

## 2. Einrichten eines Django-Projekts

Bevor du mit Templates arbeiten kannst, stelle sicher, dass dein Django-Projekt eingerichtet ist.

```
django-admin startproject myproject
cd myproject
python manage.py startapp myapp
```

- **Registriere die App** in der Datei `settings.py`:

  ```
  INSTALLED_APPS = [
      ...
      'myapp',
  ]
  ```

- **Projektstruktur**:

  ```
  myproject/
      myproject/
          settings.py
          urls.py
      myapp/
          views.py
          templates/
              myapp/
                  hello.html
  ```

---

## 3. Erstellen eines Templates

Templates befinden sich im Ordner `templates` in deinem Projekt oder in der jeweiligen App. Standardmäßig durchsucht Django den Ordner `templates` nach HTML-Dateien.

- **Beispiel für ein Template**: Datei: `hello.html`

  ```
  <!DOCTYPE html>
  <html>
  <head>
      <title>Meine Django-Seite</title>
  </head>
  <body>
      <h1>Hello, {{ name }}</h1>
  </body>
  </html>
  ```

---

## 4. Views und Rendern von Templates

In den Views werden die Templates mit Daten befüllt und zurückgegeben.

- **Ein einfaches View**:

```
from django.shortcuts import render

def hello_view(request, name):
    context = {"name": name}
    return render(request, 'myapp/hello.html', context)
```

- **URL-Konfiguration**: Datei: `urls.py`

```
from django.urls import path
from myapp import views

urlpatterns = [
    path('hello/<str:name>/', views.hello_view, name='hello'),
]
```

Rufe im Browser auf: `http://127.0.0.1:8000/hello/John/`.

---

## 5. Dynamische Inhalte einfügen

Dynamische Inhalte werden mit Variablen eingefügt:

- **Beispiel:**

```
<h1>Hallo, {{ name }}</h1>
<p>Alter: {{ age }}</p>
```

- **Python-Daten als Kontext übergeben**:

```
def hello_view(request):
    context = {
        "name": "Maria",
        "age": 30
    }
    return render(request, 'myapp/hello.html', context)
```

---

## 6. Bedingungen in Templates

Django-Templates unterstützen Bedingungstags:

- **Syntax**:

```
{% if condition %}
    Inhalt, wenn wahr
{% else %}
    Inhalt, wenn falsch
{% endif %}
```

- **Beispiel:**

```
<h1>Hello, {{ name }}</h1>
{% if age >= 18 %}
    <p>Du bist erwachsen.</p>
{% else %}
    <p>Du bist minderjährig.</p>
{% endif %}
```

---

## 7. Schleifen in Templates

Datenlisten können mit `{% for %}`-Tags durchlaufen werden.

- **Syntax**:

```
{% for item in items %}
    <p>{{ item }}</p>
{% endfor %}
```

- **Beispiel**:

```
def item_view(request):
    items = ["Apfel", "Banane", "Kirsche"]
    return render(request, 'myapp/items.html', {"items": items})
```

Datei: `items.html`

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

---

## 8. Template-Inheritance (Vererbung)

Erstelle eine Basisvorlage, um Code-Wiederholungen zu vermeiden.

- **Basisvorlage (`base.html`)**:

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Meine Seite{% endblock %}</title>
</head>
<body>
    <header>
        <h1>Willkommen</h1>
    </header>
    <main>
        {% block content %}{% endblock %}
    </main>
    <footer>
        <p>© 2024 Meine Webseite</p>
    </footer>
</body>
</html>
```

- **Abgeleitete Vorlage (`about.html`)**:

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

{% block title %}Über Uns{% endblock %}

{% block content %}
    <h2>Über Uns</h2>
    <p>Wir sind ein modernes Unternehmen.</p>
{% endblock %}
```

---

## 9. Filter in Templates

Filter manipulieren Variablenwerte.

- **Syntax**:

```
{{ variable | filter_name }}
```

- **Beispiele**:
  - `{{ name | upper }}` → gibt den Namen in Großbuchstaben aus.
  - `{{ text | wordcount }}` → zählt die Wörter in einem Text.
  - `{{ list | length }}` → gibt die Anzahl der Elemente in einer Liste zurück.

---

## 10. Beispielprojekt: Restaurant-Menü

**View:**

```
def menu_view(request):
    dishes = [
        {"name": "Pizza", "price": 8.99},
        {"name": "Pasta", "price": 6.99},
        {"name": "Salat", "price": 4.99},
    ]
    return render(request, 'myapp/menu.html', {"dishes": dishes})
```

**Template (`menu.html`):**

```
<h1>Unser Menü</h1>
<ul>
  {% for dish in dishes %}
      <li>{{ dish.name }}: {{ dish.price }} €</li>
  {% endfor %}
</ul>
```

---

## 11. Abschluss: Vorteile von Templates

- Trennung von Design und Logik.
- Wiederverwendbarer Code durch Vererbung.
- Einfaches Einfügen dynamischer Inhalte.