# Sessions

## Sessions in Flask

HTTP ist zustandslos – jede Anfrage steht für sich allein

Eine Session speichert Daten zwischen verschiedenen Anfragen

In Flask verhält sich eine Session wie ein Dictionary

```python
from flask import Flask, session
from flask_session import Session

app = Flask(__name__)
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)

@app.route('/spiel')
def spielstand():
    if 'level' in session:
        session['level'] = session['level'] + 1
    else:
        session['level'] = 1
    return f"Level {session['level']}"
```

## Wie funktionieren Sessions?

Der Server generiert eine
zufällige, eindeutige Session-ID

Die ID wird per Set-Cookie-Header
an den Browser geschickt

Bei jeder Anfrage schickt der Browser
die Session-ID automatisch an den Server

```
# Server generiert Session-ID
session_id = "x7k2j9..." # Vereinfachtes Beispiel!
```

## Übertragung an Server in Cookie

Der Browser sendet die Session-ID bei jeder Anfrage im Cookie mit

```
GET /spiel HTTP/1.1
Cookie: session=x7k2j9...
```

## Speicherort am Server

Die Session-Daten bleiben auf dem Server, der Browser bekommt nur die ID

Der Speicherort ist konfigurierbar (in unseren Beispielen: Dateisystem)

```
from flask import Flask, session
from flask_session import Session

app = Flask(__name__)
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)
```

## Praxisbeispiel: Login

Bei erfolgreichem Login den Benutzernamen in der Session speichern

Bei jeder Anfrage prüfen, ob die Session einen Benutzer enthält

```python
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "GET":
        return render_template("login.html")
    if check_password(request.form.get("password")):
        session["user"] = request.form.get("username")
        return redirect("/dashboard")
    return "Login fehlgeschlagen"

@app.route("/dashboard")
def dashboard():
    if "user" not in session:
        return redirect("/login")
    return f"Willkommen, {session['user']}!"
```

## Beispiel: Warenkorb

Der Warenkorb existiert nur in der Session

```python
@app.route("/add_to_cart/<item_id>")
def add_to_cart(item_id):
    if "cart" not in session:
        session["cart"] = []
    session["cart"] = session["cart"] + [item_id]
    return redirect("/cart")

@app.route("/cart")
def show_cart():
    cart_items = session.get("cart", [])
    return render_template("cart.html", items=cart_items)
```

## Beispiel: Warenkorb

Sessions lassen sich auch
länger am Leben erhalten:

```
from datetime import timedelta

app.config["PERMANENT_SESSION_LIFETIME"] = timedelta(days=7)
app.config["SESSION_PERMANENT"] = True # vor Session(app)
```

## Sicherheitsgrundlagen

Sensible Daten wie Passwörter gehören NIE in die Session

Sessions haben ein Ablaufdatum – das ist gut so!

Bei einem Logout sollten Sie die Session löschen:

```
@app.route("/logout")
def logout():
    session.clear()
    return redirect("/")
```

## Weiteres Beispiel

Prüfen, ob jemand eingeloggt ist –
sonst zur Login-Seite weiterleiten

```
@app.route("/")
def index():
    if "user" not in session:
        return redirect("/login")

    return render_template("index.html",
                           name=session["user"],
                           message="Willkommen zurück!")
```

## In der Praxis

Login/Authentifizierung
nicht selbst implementieren

Bewährte Bibliotheken nutzen
(z.B. Flask-Login)

Sie kümmern sich um die
wichtigen Sicherheitsdetails

HTTP ist zustandslos – Sessions speichern Daten zwischen Anfragen

In Flask verhält sich eine Session wie ein Dictionary

Der Server vergibt eine Session-ID, die per Cookie übertragen wird

Die Session-Daten bleiben am Server; der Browser bekommt nur die ID

Typische Anwendungen sind der Login-Status und der Warenkorb

Sensible Daten gehören nie in die Session; für Login bewährte Bibliotheken nutzen