Flask

Nicht nur Kommandozeile

Python wird nicht nur für die Programmierung auf der Kommandozeile verwendet, obwohl das ein Haupt-anwendungsfall ist.

Python enthält native Funktionalität zur Unterstützung von Netzen und mehr, sodass damit auch Backend-Systeme für Websites geschrieben werden können.

Was ist Flask?

Web-Frameworks erleichtern das Entwickeln von Webanwendungen, indem sie die Details der Kommunikation abstrahieren und Hilfsfunktionen bereitstellen.

Einige der beliebtesten Frameworks für Python sind: Django, Pyramid und Flask.

Wir verwenden Flask, weil es leichtgewichtig und einfach zu handhaben ist, aber dennoch viele Funktionen bietet.

Beispiel: Aktuelle Uhrzeit

Websites, die nur reines
HTML verwenden, leiden
unter einer Einschränkung.

Beispiel: eine Website,
die die aktuelle Uhrzeit in
Bamberg minutengenau anzeigt.

```
<html>
<head>
    <title>Current Time in Bamberg</title>
</head>
<body>
    The current time in Bamberg is 14:08
</body>
</html>
```

Beispiel: Aktuelle Uhrzeit

```python
from flask import Flask
from datetime import datetime
from zoneinfo import ZoneInfo

app = Flask(__name__)

@app.route("/")
def time():
    now = datetime.now(ZoneInfo('Europe/Berlin'))
    now_formatted = now.strftime("%Y-%m-%d %H:%M:%S")
    return f"Date and time: {now_formatted}"
```

Beispiel: Aktuelle Uhrzeit

```
@app.route('/')
def index():
    return "Hello, World!"
```

Ausführung in VS Code

Eine Flask-Anwendung lässt sich einfach
in VS Code ausführen.

```
export FLASK_APP=app.py # nur nötig, wenn die Datei nicht app.py heißt
export FLASK_DEBUG=1
flask run
```

Routen mit Parametern

Daten können als Teil von URLs übergeben werden, auch das ist eine HTTP-GET-Anfrage.

Die Daten werden dabei direkt im Pfad-Teil der URL übermittelt.

```
@app.route("/show/<number>")
def show(number):
    return "You passed in {}".format(number)
```

POST-Daten empfangen

Daten können auch über HTML-Formulare übergeben werden (HTTP POST) – Flask muss explizit auf POST reagieren.

```
from helpers import apology # helpers.py from finance task

@app.route("/login", methods=['GET', 'POST'])
def login():
    if not request.form.get("username"):
        return apology("must provide username")
```

Fallunterscheidung: GET oder POST

Das Verhalten der Funktion je nach
Typ der HTTP-Anfrage variieren:

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == "POST":
        # mache etwas
        pass

    else:
        # mache etwas anderes
        pass
```

Nützliche Funktionen

`url_for()`

Generiert URLs zu benannten Routen

`redirect()`

Leitet den Benutzer zu einer anderen URL weiter

`session`

Speichert Daten, die sich über mehrere Anfragen eines Benutzers hinweg „merken“

`render_template()`

Rendert HTML-Templates mit dynamischen Daten

Jinja-Templates

Jinja ist eine Template-Engine für Python

Ermöglicht die Erstellung dynamischer HTML-Seiten

Unterstützt Variablen, Kontrollstrukturen und Funktionen

Templates werden in separaten Dateien gespeichert und von Flask gerendert

```
from flask import render_template

@app.route('/')
def index():
    name = "Alice"
    return render_template('index.html', name=name)
```

Jinja-Templates

```
<h1>Hallo, {{ name }}!</h1>

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

Jinja-Templates

```
{% if user.logged_in %}
    <a href="/logout">Logout</a>
{% else %}
    <a href="/login">Login</a>
{% endif %}

{{ variable | upper }}
```

Jinja-Templates

```
<html>
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <h1>Meine Anwendung</h1>
    {% block content %}{% endblock %}
</body>
</html>
```

Jinja-Templates

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

{% block title %}Startseite{% endblock %}

{% block content %}
    <h2>Willkommen auf der Startseite!</h2>
    <p>Hier steht der individuelle Inhalt der Seite.</p>
{% endblock %}
```

Beispiel-Anwendung

Eine einfache Aufgabenverwaltung:
Aufgaben hinzufügen, anzeigen, löschen

```
from flask import Flask, render_template, request, redirect, url_for

app = Flask(__name__)

tasks = []

@app.route('/')
def index():
    return render_template('index.html', tasks=tasks)

@app.route('/add', methods=['POST'])
def add():
    task = request.form['task']
    tasks.append(task)
    return redirect(url_for('index'))

@app.route('/remove/<int:index>')
def remove(index):
    tasks.pop(index)
    return redirect(url_for('index'))
```

Beispiel-Anwendung

```
<!DOCTYPE html>
<html>
<head>
    <title>Aufgabenliste</title>
</head>
<body>
    <h1>Aufgabenliste</h1>

    <form action="{{ url_for('add') }}" method="post">
        <input type="text" name="task" placeholder="Neue Aufgabe">
        <input type="submit" value="Hinzufügen">
    </form>

    <ul>
        {% for task in tasks %}
            <li>
                {{ task }}
                <a href="{{ url_for('remove', index=loop.index0) }}">Löschen</a>
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

Löschen mit POST

```
@app.route('/remove/<int:index>', methods=['POST'])
def remove(index):
    tasks.pop(index)
    return redirect(url_for('index'))
```

Löschen mit POST

```
<form action="{{ url_for('remove', index=loop.index0) }}" method="post">
    <input type="submit" value="Löschen">
</form>
```

EXTRAS IN 3 MINUTEN

FRAGEN – ANTWORTEN – RÄTSEL
UND KURZE ZUSAMMENFASSUNG

Die Drei Extra-Minuten

```
<li>
    {{ task }}
    <a href="#" onclick="deleteTask({{ loop.index0 }})">Löschen</a>
</li>

<script>
function deleteTask(index) {
    var form = document.createElement('form');
    form.method = 'POST';
    // Generate a URL with a dummy index (0)
    let url = "{{ url_for('remove', index=0) }}";
    // Replace the dummy index with the actual index
    url = url.replace('0', index);
    form.action = url;
    document.body.appendChild(form);
    form.submit();
}
</script>
```

Die Drei Extra-Minuten

```python
from flask import Flask, render_template, request, redirect, url_for

app = Flask(__name__)

shopping_list = []

@app.route('/')
def index():
    return render_template('index.html', shopping_list=shopping_list)

@app.route('/add', methods=['POST'])
def add():
    item = request.form['item']
    shopping_list.append({'name': item, 'bought': False})
    return redirect(url_for('index'))

@app.route('/remove/<int:index>', methods=['POST'])
def remove(index):
    shopping_list.pop(index)
    return redirect(url_for('index'))

@app.route('/buy/<int:index>', methods=['POST'])
def buy(index):
    shopping_list[index]['bought'] = True
    return redirect(url_for('index'))
```

Die Drei Extra-Minuten

```
<!DOCTYPE html>
<html>
<head>
    <title>Einkaufsliste</title>
</head>
<body>
    <h1>Einkaufsliste</h1>

    <form action="{{ url_for('add') }}" method="post">
        <input type="text" name="item" placeholder="Neuer Eintrag">
        <input type="submit" value="Hinzufügen">
    </form>
    <ul>
        {% for item in shopping_list %}
            <li>
                {{ item.name }}
                {% if not item.bought %}
                    <a href="#" onclick="buyItem({{ loop.index0 }})">Kaufen</a>
                {% else %}
                    (gekauft)
                {% endif %}
                <a href="#" onclick="deleteItem({{ loop.index0 }})">Löschen</a>
            </li>
        {% endfor %}
    </ul>
```

Die Drei Extra-Minuten

```
    <script>
    function deleteItem(index) {
        var form = document.createElement('form');
        form.method = 'POST';
        // Generate a URL with a dummy index (e.g., 0)
        let url = "{{ url_for('remove', index=0) }}";
        // Replace the dummy index with the actual index
        url = url.replace('0', index);
        form.action = url;
        document.body.appendChild(form);
        form.submit();
    }
    function buyItem(index) {
        var form = document.createElement('form');
        form.method = 'POST';
        // Same as above
        let url = "{{ url_for('buy', index=0) }}";
        url = url.replace('0', index);
        form.action = url;
        document.body.appendChild(form);
        form.submit();
    }
    </script>
</body>
</html>
```

Die Drei Extra-Minuten

Trennung von Logik (Python) und Präsentation (HTML/Jinja) – modularer und wartbarer Code.

Verwendung von semantischem HTML wo möglich (z.B. `<ul>` für die Liste).

Vermeidung von Codeduplikation durch Jinja-Schleifen und -Bedingungen.

Flask ist ein leichtgewichtiges Python-Web-Framework für das Backend

Routen verbinden URLs mit Python-Funktionen (@app.route)

Daten kommen über URL-Parameter (GET) oder Formulare (POST) zum Server

Jinja-Templates erzeugen dynamisches HTML mit {{ Variablen }} und {% Logik %}

Nützliche Funktionen: url_for(), redirect(), render_template(), session

Trennung von Logik (Python) und Präsentation (HTML/Jinja) hält Code wartbar