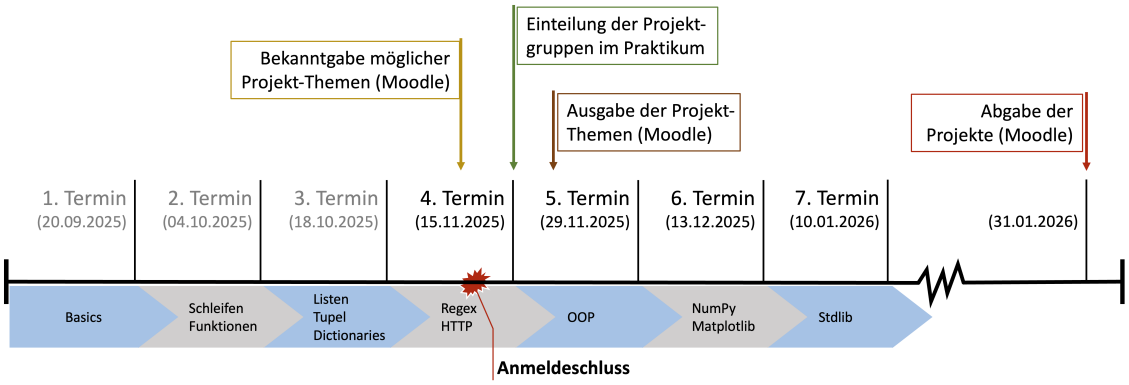
vl-04

November 15, 2025

1 Programmierung für KI

1.0.1 Wintersemester 2025/26

Prof. Dr. Heiner Giefers



1.1 Projektthemen

1. Interaktives Finanz-Dashboard
2. Analyse und Visualisierung von Wetterdaten
3. Rezept-Empfehlungssystem
4. Social-Media-Stimmungsanalyse
5. Automatisierter Nachrichten-Aggregator mit Zusammenfassung
6. Pokémon-Kampfsimulator
7. Analyse von Open-Data-Stadtinformationen
8. Musik-Genre-Klassifikator
9. Preis-Tracker für Online-Shops
10. Persönlicher Ausgaben-Manager mit intelligenter Kategorisierung

1.2 1. Interaktives Finanz-Dashboard

1. **Finanzdaten abrufen**
   - Aktien oder Kryptowährungen mit `yfinance` oder Alpha Vantage abrufen
   - Kursverläufe, Tages- und Jahresstatistiken analysieren
2. **Darstellung und Visualisierung**
   - Interaktive Diagramme mit **Plotly** oder **Matplotlib**
   - Vergleich mehrerer Aktien oder Coins
3. **Optionale Erweiterungen**
   - **Portfolio-Verwaltung:** Benutzer können mehrere Werte verfolgen
   - **Technische Analyse:** Moving Average, RSI, MACD
   - **KI-Prognose:** Zeitreihenanalyse mit Prophet oder Scikit-learn
4. **Ziel**
   - Eigenständige Analyseplattform mit Datenabruf, Visualisierung und optionaler Prognosefunktion

1.3 2. Analyse und Visualisierung von Wetterdaten

1. **Datenerfassung**
   - Nutzung von OpenWeatherMap oder Meteostat
   - Aktuelle und historische Wetterdaten abrufen
2. **Datenanalyse**
   - Temperatur- und Niederschlagsverlauf über Zeit mit **pandas** analysieren
   - Statistische Auswertungen (z. B. Mittelwerte, Maxima)
3. **Visualisierung**
   - Zeitverläufe mit **Matplotlib** oder **Seaborn**
   - Interaktive Karten mit **Folium**
4. **Erweiterungen**
   - Vergleich verschiedener Orte oder Zeiträume
   - Clustering typischer Wettermuster mit Scikit-learn

1.4 3. Rezept-Empfehlungssystem

1. **Datengrundlage**

- Nutzung der TheMealDB API
- Alternativ: Web Scraping mit `BeautifulSoup`

2. **Funktionalität**
   - Benutzer gibt Zutaten ein → passende Rezepte werden angezeigt
   - Anzeige von Name, Zutatenliste und Bild
3. **Erweiterungen**
   - Fehlertolerante Eingabe mit NLP (`spaCy`, `NLTK`)
   - Filter nach Kategorie, Aufwand oder Küche
   - Empfehlungen ähnlicher Rezepte (Content-Based Filtering)
4. **Ziel**
   - Smartes Rezepttool mit lernender Vorschlagslogik

1.5 4. Social-Media-Stimmungsanalyse

1. **Datenerfassung**
   - Beiträge über Reddit oder Twitter-API abrufen (`praw`, `tweepy`)
   - Alternativ: Kommentare von Nachrichtenseiten scrapen
2. **Analyse**
   - Stimmungsbewertung mit **VADER**, **TextBlob** oder **NLTK**
   - Aggregation positiver/neutraler/negativer Werte
3. **Visualisierung**
   - Diagramme oder Wortwolken mit **Matplotlib** und **wordcloud**
   - Zeitverlauf der Stimmung zum Thema
4. **Erweiterungen**
   - Themenmodellierung (Topic Modeling mit Gensim)
   - Echtzeit-Dashboard mit Flask oder Dash

1.6 5. Automatisierter Nachrichten-Aggregator

1. **Datensammlung**
   - RSS-Feeds oder APIs (z. B. News API) verwenden
   - Artikelüberschriften, Quellen und Veröffentlichungszeiten speichern

2. **Verarbeitung**
   - Doppelte Artikel erkennen und zusammenfassen
   - Thematische Gruppierung (Politik, Wirtschaft, Sport ...)
3. **Darstellung**
   - Anzeige in einer einfachen Weboberfläche (Flask oder Django)
   - Option: tägliche Zusammenfassung per E-Mail
4. **Erweiterungen**
   - Automatische Textzusammenfassung mit **Transformers (BART/T5)**
   - Benutzerdefinierte Themenfilter

1.7 6. Pokémon-Kampfsimulator

1. **Datenbasis**
   - Nutzung der PokéAPI zur Abfrage von Pokémon-Daten
2. **Spielmechanik**
   - Rundenbasiertes Kampfsystem (KP, Schaden, Typvorteile)
   - Textbasiert oder grafisch mit **Pygame**
3. **Erweiterungen**
   - Team-Auswahl und Wechsel im Kampf
   - Status-Effekte wie Paralyse oder Gift
   - KI-Gegner mit Entscheidungslogik
4. **Ziel**
   - Strategisches Spiel mit API-Integration und optionaler KI

1.8 7. Analyse von Open-Data-Stadtinformationen

1. **Datengrundlage**
   - Offene Datensätze von Städten (z. B. Open Data Berlin)
   - Themen: Verkehr, Energie, Bevölkerung, Umwelt
2. **Analyse**
   - Verarbeitung mit **pandas** und Visualisierung mit **Matplotlib**
   - Korrelationen, Ranglisten, Trends
3. **Erweiterungen**

- Karten mit **Folium** oder **Geopandas**
- Clustering ähnlicher Regionen mit **Scikit-learn**
- Web-Dashboard für interaktive Darstellung

4. **Ziel**
   - Datengetriebene Stadtanalyse mit Visualisierung

1.9 8. Musik-Genre-Klassifikator

1. **Datengrundlage**
   - Nutzung des GTZAN-Datensatzes
   - Audiomerkmale (Tempo, Spektrum, Mel-Frequenzen) extrahieren mit **librosa**
2. **Modelltraining**
   - ML-Modell mit **Scikit-learn** (z. B. SVM, Random Forest)
   - Option: Deep-Learning-Modell (CNN) mit **TensorFlow** oder **PyTorch**
3. **Evaluation**
   - Trainings-/Testdaten splitten, Accuracy messen
   - Visualisierung mit Confusion Matrix
4. **UI**
   - Upload eigener Audiodateien, Anzeige der Klassifikation

1.10 9. Preis-Tracker für Online-Shops

1. **Datenerhebung**
   - Preise über Web Scraping (`requests`, `BeautifulSoup`) auslesen
   - Speicherung in CSV oder SQLite
2. **Analyse**
   - Preisverlauf grafisch darstellen
   - Günstigster Zeitpunkt oder Preisentwicklung erkennen
3. **Benachrichtigung**
   - Preisalarme per E-Mail oder Telegram-Bot
   - Automatische Aktualisierung (z. B. mit `schedule`)
4. **Erweiterungen**
   - Robustheit gegenüber dynamischen Seiten (`Selenium`)

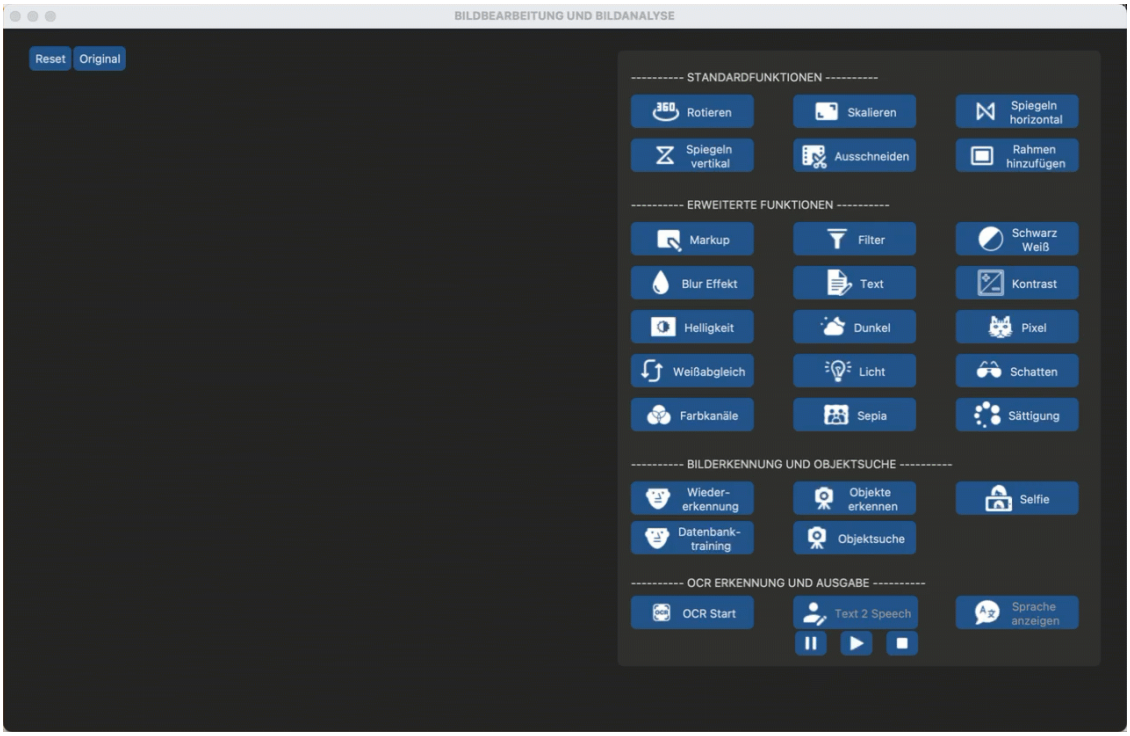
- Prognose kommender Preisänderungen (Regression)

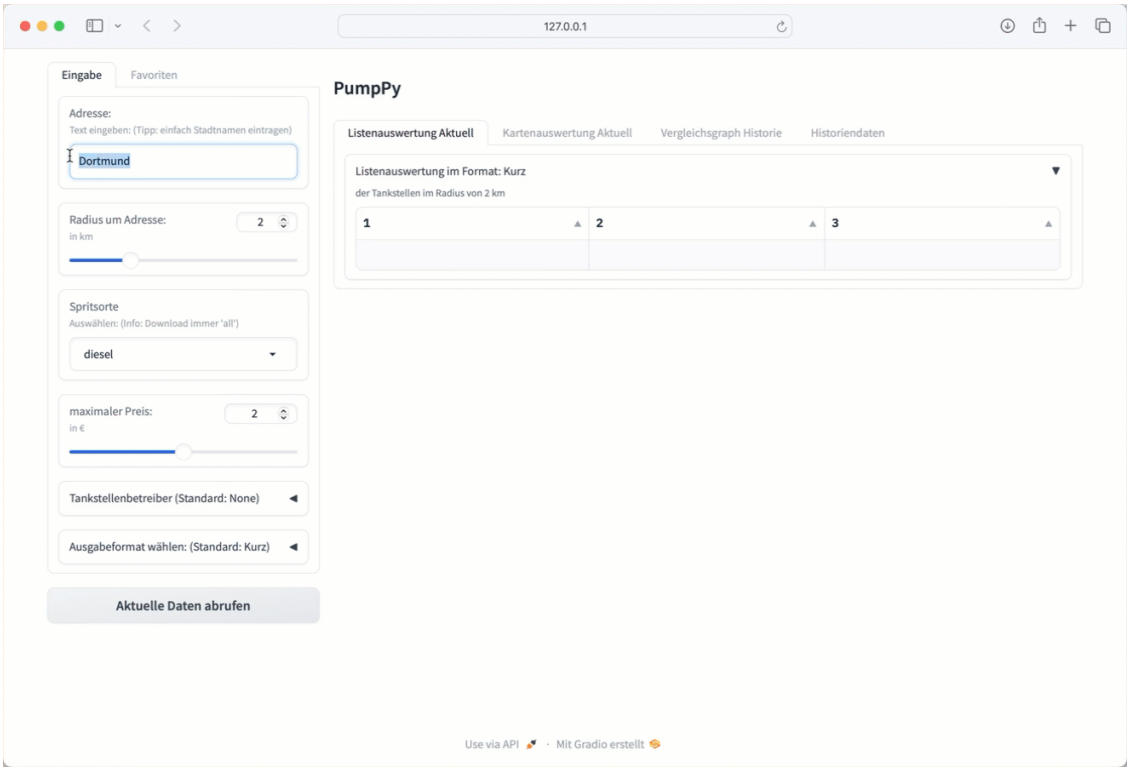
1.11 10. Persönlicher Ausgaben-Manager

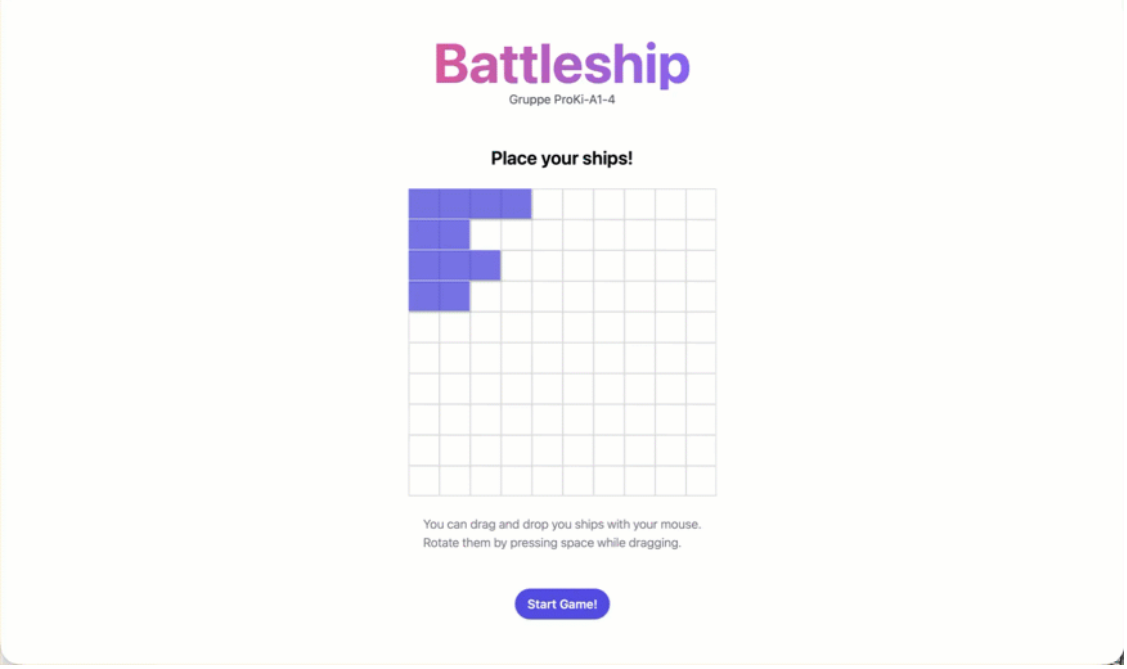
1. **Grundfunktionen**
   - Einnahmen und Ausgaben erfassen (CSV oder SQLite)
   - Monatliche Übersicht und Summenbildung
2. **Visualisierung**
   - Balken- oder Kreisdiagramme mit **Matplotlib** oder **Plotly**
3. **Erweiterungen**
   - Budgetlimits und Warnmeldungen
   - Automatische Kategorisierung mit **Scikit-learn**
   - CSV-Import von Kontoauszügen
4. **Ziel**
   - Einfaches Finanzmanagement-Tool mit Lernkomponente

1.12 Übersicht: APIs & Zugriffsmöglichkeiten

Projekt	Schnittstelle	Kosten / API-Zugriff
1. Finanzdashboard	yfinance	kostenlos, aktiv gepflegt
	Alpha Vantage	25 Req/Tag, E-Mail erforderlich
2. Wetterdaten	OpenWeatherMap	1000 Req/Tag, Account notwendig
	Meteostat	komplett kostenlos
3. Rezepte	TheMealDB	kostenlos für Edu, API-Key „1“
	Scraping	Terms vorher prüfen
4. Social Media	Reddit	Rate Limit (100/min), Account erforderlich
	Twitter/X	100 Reads/Monat, Account erforderlich
	Scraping	Terms vorher prüfen
5. News	RSS / offene APIs	z. B. Tagesschau API: 60 Req/Stunde
	NewsAPI	24h Delay, 100 Req/Tag, Account notwendig
6. Pokémon	PokeAPI	kostenlos, lokales Caching empfohlen
7. Städteinfo	Open Data Berlin	je nach Quelle unterschiedlich; z. B. VBB: Anmeldung
	GovData	je nach Quelle; z. B. Straßendaten RKN: kostenlos
8. Musik	GTZAN Dataset	kostenlos
9. Preise	Scraping	Terms vorher prüfen
10. Ausgaben	–	keine API


BILDBEARBEITUNG UND BILDANALYSE
Reset
Original
---------- STANDARDFUNKTIONEN ----------
Rotieren
Skalieren
Spiegeln horizontal
Spiegeln vertikal
Ausschneiden
Rahmen hinzufügen
---------- ERWEITERTE FUNKTIONEN ----------
Markup
Filter
Schwarz Weiß
Blur Effekt
Text
Kontrast
Helligkeit
Dunkel
Pixel
Weißabgleich
Licht
Schatten
Farbkanäle
Sepia
Sättigung
---------- BILDERKENNUNG UND OBJEKTSUCHE ----------
Wieder-erkennung
Objekte erkennen
Selfie
Datenbank-training
Objektsuche
---------- OCR ERKENNUNG UND AUSGABE ----------
OCR Start
Text 2 Speech
Sprache anzeigen


127.0.0.1
Eingabe
Favoriten
Adresse:
Text eingeben: (Tipp: einfach Stadtnamen eintragen)
Dortmund
Radius um Adresse:
2
in km
Spritsorte
Auswählen: (Info: Download immer 'all')
diesel
maximaler Preis:
2
in €
Tankstellenbetreiber (Standard: None)
Ausgabeformat wählen: (Standard: Kurz)
Aktuelle Daten abrufen
PumpPy
Listenauswertung Aktuell
Kartenauswertung Aktuell
Vergleichsgraph Historie
Historiendaten
Listenauswertung im Format: Kurz
der Tankstellen im Radius von 2 km
1
2
3
Use via API
Mit Gradio erstellt



1.13 Ablauf

1. Sie einigen sich in Ihrer Gruppe auf ein Thema
2. Tragen Sie das Thema in Moodle unter der Abgabe **Themenwahl** ein (1x pro Gruppe)
3. Betreuer der Praktikumsgruppe (Dorka, Kuzmic, Graef oder Eberts) begleitet auch das Projekt
4. Abgabe des Quellcodes bis zum **31.01.2026**

1.14 Umsetzung

- Ihr Team muss eines der Themen bearbeiten
- Wichtig: **Es gibt keine konkreten Zielvorgaben. Die Themen können (und sollen) nach Ihren Vorstellungen ausgestalltet werden**
- Bauen Sie das Thema nach Belieben aus
- Der Einsatz von LLMs/KI ist ausdrücklich zugelassen!

1.15 Verwendung von KI / Coding Assistenten

- Sie dürfern Sprachmodelle oder Coding Agents verwenden
- Geben Sie (im README) an, welche Tools Sie verwendet haben
- Über die FH haben Sie Zugriff auf den kimpuls Chatbot (https://openai.ki.fh-swf.de/), über das *GitHub Student Developer Pack* können Sie kostenfrei *Copilot* nutzen
- **Sie müssen den Code erklären können!**

1.16 Form der Abgabe

- Die Abgabe erfolgt über Moodle als Gruppenabgabe (Projektordner oder zip-Archiv)
- Verwenden Sie für das Projekt ein **Virtual Environment**
- Verwenden Sie ausschließlich Bibliotheken, die sich per `pip` installieren lassen

- Die Abgabe **muss** folgendes enthalten:
  - Eine **README.md** Datei mit einer (Kurz-) Anleitung und den wichtigsten Informationen zum Projekt
  - Einer `requirements.txt` Datei mit den benötigten Bibliotheken (mit `pip freeze` erzeugen)
  - Den kompletten Quellcode
  - Notwendige Daten (falls zu groß, bitte mit Betreuer absprechen)

1.17 UI-Programmierung

- Projekte können grundsätzlich auch als Konsolenprogramme umgesetzt werden

- Ein grafisches *User Interface (UI)* ist sinnvoll und oft benutzerfreundlicher als eine Konsolenschnittstelle

- Muss nicht *perfekt* sein
- UIs können als Jupyter-Notebook, Desktop- oder Web-Anwendungen realisiert werden

- Desktop-UIs: z. B. mit Tkinter oder DearPyGUI

- Web-UIs: z. B. mit Flask oder FastAPI, Gradio, Streamlit, ...

[ ]:
```
import sys
!{sys.executable} -m pip install dearpygui
```

[ ]:
```
%%writefile gui_app_dearpygui.py
import dearpygui.dearpygui as dpg

def save_callback():
    print("Save Clicked")

dpg.create_context()
dpg.create_viewport()
dpg.setup_dearpygui()

with dpg.window(label="DearPyGUI Beispiel"):
    dpg.add_text("Hello, World!")
    dpg.add_button(label="Save", callback=save_callback)
    dpg.add_input_text(label="string")
    dpg.add_slider_float(label="float")

dpg.show_viewport()
dpg.start_dearpygui()
dpg.destroy_context()
```

[ ]:
```
!python ./gui_app_dearpygui.py
```

[ ]:
```
%%writefile gui_app_tkinter.py
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.title("Tkinter Beispiel")
value = tk.DoubleVar()
def save_callback():
    print("Save clicked")
def slided(value):
    scale_label.config(text=f"Wert: {float(value):.2f}")

# Widgets
label = ttk.Label(root, text="Hello world")
label.pack(pady=5)
button = ttk.Button(root, text="Save", command=save_callback)
button.pack(pady=5)
entry = ttk.Entry(root)
entry.pack(pady=5)
scale_label = ttk.Label(root, text="Wert:")
scale_label.pack(pady=5)
scale = ttk.Scale(root, from_=0, to=100, orient="horizontal", variable=value,␣
 ↪command=slided)
scale.pack(pady=5)
root.mainloop()
```

[ ]:
```
!python ./gui_app_tkinter.py
```

1.18 Web-Anwendungen

```
import sys
!{sys.executable} -m pip install flask
```

[ ]:
```
%%writefile gui_app_flask.py
from flask import Flask, render_template, request
from geopy.geocoders import Nominatim

app = Flask(__name__)
geolocator = Nominatim(user_agent="python_flask_demo")


@app.route("/", methods=["GET", "POST"])
def index():
    location = None

    if request.method == "POST":
        address = request.form.get("address", "").strip()
```

```
        if address:
            location = geolocator.geocode(address)

    return render_template("index.html", location=location)


if __name__ == "__main__":
    app.run()
```

[ ]:
```
%%writefile ./templates/index.html
<!doctype html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Adress-Suche mit OpenStreetMap</title>
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"/
 ↪>
  <style>
    body { font-family: system-ui, sans-serif; margin: 2rem; }
    #map { height: 400px; margin-top: 1rem; border: 1px solid #ccc; }
  </style>
</head>
<body>
  <h1>Adress-Suche mit OpenStreetMap</h1>

  <form method="post">
    Adresse:
    <input type="text" name="address" placeholder="z.B. FH Südwestfalen␣
 ↪Iserlohn">
    <button type="submit">Suchen</button>
  </form>

  {% if location %}
    <p>
      <strong>{{ location.address }}</strong><br>
      Koordinaten: {{ location.latitude }}, {{ location.longitude }}
    </p>
    <div id="map"></div>
  {% else %}
    <p>Gib eine Adresse ein und klicke auf „Suchen“.</p>
  {% endif %}

  <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>

  {% if location %}
  <script>
    const lat = {{ location.latitude }};
```

```
    const lon = {{ location.longitude }};

    const map = L.map("map").setView([lat, lon], 15);

    L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
      maxZoom: 19
    }).addTo(map);

    L.marker([lat, lon]).addTo(map);
  </script>
  {% endif %}
</body>
</html>
```

[ ]:
```
!python ./gui_app_flask.py
```

1.19 Programmieren im Team

- Das Schreiben von Software im Team ist der *Normalfall*
- Das Gebiet der *Softwaretechnik* beschäftigt sich mit "*Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen*" (Balzert, 1996)
- Quellcode teilen, am besten per **Git** (Github Training, z.B. hier)
- Mein Tipp: Eine *agile Vorgehensweise* bringt Sie am schnellsten zum *Ziel*
- Erstellen Sie möglichst schnell einen **funktionsfähigen Prototypen**

1.20 Agile Softwareentwicklung: Iterativ & Inkrementell

1. Sie starten bei einem Prototyp mit minimalem Funktionsumfang (den haben Sie ja bereits haben) und legen einen Katalog von Zielen fest (*Product Backlog*)
2. Sie verabreden sich zu regelmäßigen Meetings (z.B. wöchentlich). Zwischen den Meetings wird entwickelt (*Sprints*)
3. In jedem Meeting wird der aktuelle Entwicklungsstand besprochen. Es können neue (Teil-) Ziele identifiziert werden, die in den Product Backlog aufgenommen werden.
4. Sie wählen in jedem Meeting diejenigen Ziele aus, die Sie realistischerweise bis zum nächsten Meeting umsetzen können (*Sprint Backlog*). Auf das erreichen dieser ausgewählten Ziele konzentrieren Sie sich (arbeitsteilig) bis zum nächsten Meeting.

1.21 Dokumentation und Präsentation

- Sie dürfen (und sollen) Beispiele und Referenzen aus dem Netz verwenden
- **Alle Quellen müssen angegeben werden**
- Wenn Sie Quellcode übernehemen, geben Sie an welche Teile (nicht) von Ihnen entwickelt wurden
- Ausführlich kommentieren!
- Keine Ausarbeitung, aber Präsentation bei der Projektvorstellung

1.22 Bewertung und Kriterien

- Wie komplex war das Projekt?
- Wie gut wurde es umgesetzt?
- Wie gut wurde es präsentiert?
- Es werden Individualnoten vergeben

1.23 Bewertungskriterien - Implementierung (50%)

- **Komplexität** Wie umfangreich ist der entwickelte Code? Wurden verschiedene Teilaspekte (GUI, KI, Ein-/Ausgabe, ...) behandelt? Wie schwierig/neu war die Umsetzung unter Berücksichtigung der Modulinhalte? (**40%**)
- **Resultat** Wie gut und wie gut verwendbar ist ist die entstandene Software? Wurden viele sinnvolle Funktionalitäten/Use-Cases implementiert? Wie Robust ist das Programm in der Benutzung? (**20%**)
- **Codequalität** Ist der Code sinnvoll kommentiert? Ist das Projekt sinnvoll strukturiert? Wurden sinvolle (Python-spezifische) Programmkostrukte verwendet? Gibt es Tests, Fehlerabfragen, Rückmeldungen, etc? Wurden geeignete Bibliotheken verwendet? (**30%**)
- **Dokumentation** Lässt sich das Programm einfach ausführen? Falls zusätzliche Schritte/Pakete notwendig sind, ist die Einrichtung dokumentiert? Sind Quellen korrekt angegeben? (**10%**)

1.24 Bewertungskriterien - Präsentation (50%)

- **Vortrag** Wird das Projekt verständlich vorgestellt? Werden interessante Teilaspekte detailiert vorgestellt? Wird der Redeanteil pro Person eingehalten? (**35%**)
- **Folien** Wie gut sind die Präsentationsmaterialien aufbereitet? Sind Code-Beispiele verständlich dargestellt? Ist der Foliensatz selbsterklärend? (**15%**)
- **Fragen** Wie klar und umfassend erden Fragen beantwortet? Wie spontan kann geantwortet werden. Gibt es eine erkennbare Aufteilung von Kometenzen und werden Fragen entsprechend abgefangen? (**50%**)

1.25 Heute im Praktikum: Regex

- Regex sind mächtige Werkzeuge zur Textsuche und -verarbeitung
- Viele klassische Regex-Aufgaben lassen sich heute direkt mit LLMs lösen
- LLMs erkennen Muster, extrahieren Informationen und transformieren Texte ohne komplexe Syntax
- Regex bleiben wichtig, wenn Präzision, Geschwindigkeit oder feste Regeln gefordert sind
- LLMs benötigen entweder Internetzugang zur API oder lokale Rechenressourcen

```
import sys
!{sys.executable} -m pip install openai
```

[ ]:
```
from openai import OpenAI
client = OpenAI()

code = "for i in range(0,10): print(i)"
```

```
resp = client.chat.completions.create(
    model="gpt-4.1-mini",
    messages=[{"role": "user", "content": f"Verbessere diesen Code:\n{code}"}]
)
print(resp.choices[0].message.content)
```

[ ]:
```
from openai import OpenAI
client = OpenAI()

text = """
In unserem neuen Projekt arbeiten Teams aus 10115 Berlin, 04109 Leipzig, 60311␣
 ↪Frankfurt, 20095 Hamburg und
70173 Stuttgart zusammen. Jede Stadt übernimmt dabei einen eigenen Schwerpunkt,␣
 ↪sodass wir regionale
Expertise optimal nutzen können.
"""

resp = client.chat.completions.create(
    model="gpt-4.1-mini",
    messages=[{"role": "user", "content": f"Gib mir eine Liste der PLZ in diesem␣
 ↪Text:\n{text}"}]
)
print(resp.choices[0].message.content)
```

1.26 Zum 5. Termin

- **Web-Services**: Funktionen und Datenquellen aus dem Internet verwenden
- **Objektorientierung:** Eigene Klassen und Methoden definieren