USING DATA STRUCTURES EFFECTIVELY

Chapter 3

Maren Beßler, Enya Fielitz

AGENDA

01 ***RELEVANZ DES THEMAS***

02 ***NATIVE PYTHON DATA STRUCTURES***
Lists, Tuples, Dictionaries, Sets

03 ***NUMPY ARRAYS***
Dask, Arrays in Machine Learning

04 ***PANDAS DATAFRAMES***
Functionality, Performance, Considerations

01 Relevanz des Themas

- ✅ **Performance:** optimieren von Berechnungen und Speicherverbrauch
- ✅ **Lesbarkeit:** leichtere Wartung und Verständlichkeit bei klar strukturiertem Code
- ✅ **Methoden:** leichterer Umgang mit Daten durch spezifische Methoden
- ✅ **Speicherverwaltung:** effiziente Nutzung von Ressourcen, besonders bei großen Datensätzen

→ Veränderbare, dynamische Arrays, die verschiedene Datentypen enthalten können

```
my_list = [1, 2, 3, "Hello"]
my_list.append(4)
print(my_list)  # [1, 2, 3, "Hello", 4]
```

Wann nutzen?

- geordnete Sammlung von Elementen benötigt
- Änderungen an Größe und Reihenfolge notwendig

→ unveränderlich (immutable), schneller als Listen

```
my_tuple = (1, 2, 3)
print(my_tuple[0])  # 1
```

Wann nutzen?

- Veränderung der Daten nicht erwünscht

02 DICTIONARIES

→ speichert Schlüssel-Wert-Paare für schnellen Zugriff

```
my_dict = {"name": "Alice", "age": 25}
print(my_dict["name"])  # Alice
```

Wann nutzen?

- schnelle Suche nach eindeutigen Schlüsseln erforderlich
- Speicherung in Hash-Tabellen

→ enthält nur eindeutige Elemente, ungeordnet

```
my_set = {1, 2, 3}
my_set.add(2)  # Keine Änderung, weil 2 schon existiert
print(my_set)  # {1, 2, 3}
```

Wann nutzen?

- Entfernung von Duplikaten
- Mengenoperationen wie Schnittmengen oder Vereinigungen
- Suche nach bestimmten Elementen schneller als in Listen

03 NUMPY ARRAYS

→ n-dimensionales Array

gespeichert als zusammenhängender Speicherblock

```
arr = np.array([1, 2, 3])
print(arr * 2)   # Elementweise Multiplikation: [2, 4, 6]
print(arr + 5)   # Addition zu jedem Element: [6, 7, 8]
```

Wann nutzen?

- **Effizienz**: Speicherplatz- und geschwindigkeitsoptimiert für numerische Berechnungen
- **Vektorisierung**: Mathematische Operationen ohne Schleifen
- **Multidimensionalität**: Unterstützung für Matrizen, Tensoren und komplexe numerische Berechnungen

04 PANDAS DATAFRAMES

→ 2 Hauptdatenstrukturen : DataFrames, Series

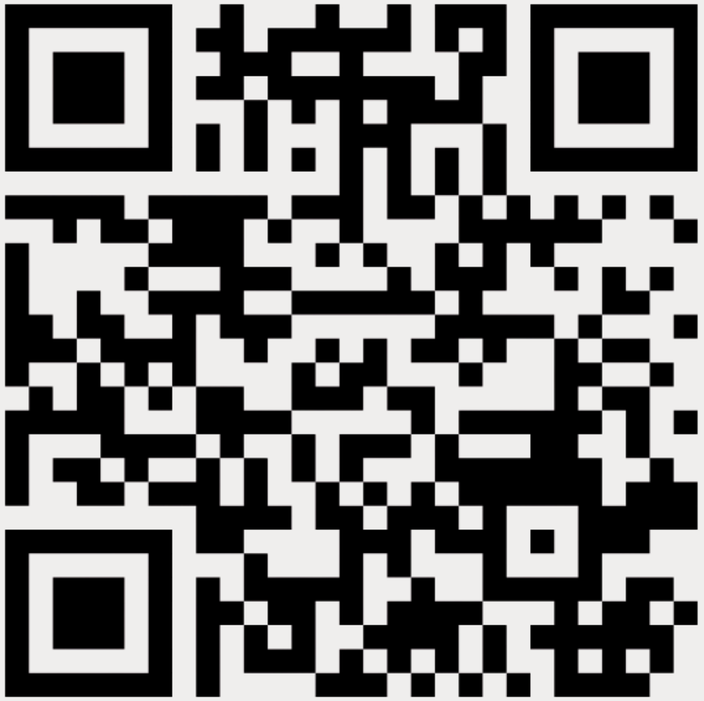
erlaubt einfache Manipulation und Analyse großer Datenmengen

```
import pandas as pd
data = {"Name": ["Alice", "Bob", "Charlie"], "Alter": [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
```

Wann nutzen?

- effiziente, leistungsstarke Filter-, Sortier- oder Gruppierungsfunktionen nötig
- für erste Schritte in der Datenanalyse & Data Science

MENTIMETER



Code: 5307 6382

MENTIMETER

https://github.com/Bessi958/Chapter-03

TAKEAWAYS

- Bewusstsein über Anwendungsfall und mögliche Datenverarbeitung
- nested lists vermeiden
- vectorized operations bevorzugen
- wenn vorhanden, built-in Funktionen benutzen