TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Alexander Jenke, Theodor Straube

Übung4: Data processing

Programmierkurs Python // Montag, 18. November 2019

ToC

- Files
- Data Structures
- List comprehentions
- Packets
  - os
  - sys
  - numpy
  - matplotlib.pyplot

Files

öffnen

```
def open(file, mode='r', encoding=None, ...):
```

- Gibt einen File-Descriptor zurück
- file: String mit Dateipfad (relativ oder absolut)
- mode: Modus wie die Datei geöffnet wird
  - r: lesen
  - w: schreiben
  - x: neue Datei erstellen
  - a: an bestehende Datei anfügen
  - b: binäre Datei
  - t: Textdatei
- encoding: verwendete Zeichencodierung
  - utf-8
  - ascii
- am Ende mit `.close()` schließen, um Datenverlust zu vermieden, alternativ I/O-Operationen mit `with open("file", 'rwa') as fd:` „wrappen“

lesen & schreiben

```
fd = open("file.txt", 'rwa')
```

File-Descriptor fd unterstützt:

- `fd.write(str)` schreiben in Datei
- `fd.read()` die gesamte Datei lesen
- `fd.readline()` die nächste Zeile lesen

Mit `print("Lorem ipsum", file=fd)` kann man direkt in eine Datei schreiben anstatt wie gewohnt auf die Konsole

Pakete bieten Funktionen zum speichern von Datenstrukturen:

```
pickle.dump(obj, fd)  pickle.load(fd)
numpy.save(fd, obj)   numpy.load(fd)
torch.save(obj, fd)   torch.load(fd)
```

Teilweise kann anstelle des File-Descriptors auch der Pfad als String angegeben werden

Data Structures

- **Dict**:
```
person = {'Name': "Alexander Jenke",
          'Alter': 23}
```
  - Speichern von Key-Value-Pairs
  - Zugriff auf Wert über eindeutigen Schlüssel
    `person['Name']`
  - .keys() gibt eine Liste aller Schlüssel
    `person.keys() -> ['Name', 'Alter']`
  - .values() gibt eine Liste aller Werte
    `person.values() -> ["Alexander Jenke", 23]`

- **List**: `teilnehmer = ["Max", "Fritz"]`
  - Speichern von Werten in fester Reihenfolge
  - Am Ende hinzufügen `teilnehmer.append("Felix")`
  - Zugriff per Index `teilnehmer[0] -> "Max"`

- **Tuple**: `tutoren = ("Alex", "Theo")`
  - Speichern von Werten in fester Reihenfolge
  - Schneller und speichereffizienter als List
  - Inhalte können nicht bearbeitet werden

- **Set**: `a = {"a", "2", 3}`
  - Liste von eindeutigen Werten
    (kein Wert kann doppelt vorkommen)
  - Keine feste Reihenfolge
  - Logische Operationen möglich:
    - a|b alle Elemente aus a und b
    - a&b Elemente die sowohl in a als auch in b sind
    - a^b Elemente die entweder in a oder b sind aber nicht in beiden
    - a-b Elemente aus a die nicht in b vorkommen
    - a<b Wahrheitswert ob a ein Subset von b ist

List comprehentions

- List comprehentions (dt.: Listen-Abstraktionen) bieten die Möglichkeiten Listen kurz & prägnant zu definieren
- Definiert werden können:
  - List [ ]
  - Dict & Set { }
  - Tuple ( )

- Syntax:
  - Grundbaustein: Tu etwas mit Variable *for* Variable *in* Listenähnliche Menge
  - if-else-bedingung vor dem Grundbaustein (Achtung: if-Bedingung nach auszuführender Expression wenn Wahr)
  - if ohne else typischerweise hinter dem Grundbaustein

```
elements = range(10)

res = []
for e in elements:
    if e > 2:
        if e < 5:
            res.append(e * 2)
        else:
            res.append(e)

res2 = [e * 2 if e < 5 else e for e in elements if e > 2]
```

- *res* und *res2* liefern beide das selbe Ergebnis
- *res2 ist* doppelt so schnell

Formale Syntax-Definition:
https://docs.python.org/3/reference/expressions.html#displays-for-lists-sets-and-dictionaries

Packets
build in interfaces

- Pakete erweitern die Basisfunktionalität von Python um jede erdenkliche Funktion

- Pakete werden mit *import* eingebunden
- Beim import kann ein alias Name für das Paket festgelegt werden, um Namens konflikte aufzulösen

```
import numpy as np
```

- Nach dem import können Funktionen des Pakets genutzt werden: `np.load("file")`

- Die Funktionen von den üblichen Paketen sind effizienter implementiert als selbstgeschriebene Funktionen mit dem selben Zweck

Die Build-in Interfaces zum interagieren mit dem Betriebssystem sind ebenfalls wie Pakete zugänglich:

- os - operating system interface
  - chmod, chown, mkdir: Funktion wie die gleichnamigen Shell-Befehle
  - listdir: Auflisten aller Dateien einer Directory
  - path.join: zusammensetzen von Strings zu einem gültigen Pfad
  - path.isfile, path.isdir: Prüft ob ein Pfad gültig ist
  - path.realpath: relativer Pfad in absoluten Pfad
- sys - System-specific parameters and functions
  - argv: Liste der dem Interpreter übergebenen Argumente
  - exit: beendet das Program mit dem übergebenen (Fehler-)Code
  - stdout, stderr, stdin: Die Standard-Datenströme

Packets

external extensions

- Externe Pakete können einfach über pip3 installiert werden.
- Häufig gibt es verschiedene Pakete für die selbe Aufgabe

```
import matplotlib.pyplot as plt
```

pyplot ermöglicht ein schnelles und einfaches plotten von Graphen:

- *plot*() erzeugt einen Graphen
- *hist*() erzeugt ein Histogramm
- *show*() zeigt die Grafik an

Docs: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html

```
import numpy as np
```

numpy ermöglicht ein effizientes verarbeiten von Matrizen und Arrays:

- *ndarray*() erzeugt eine Matrix
- *asarray*() wandelt eine Liste in eine Matrix um
- *random* ist ein Submodul zur Erzeugung von Zufallszahlen
- *size* Attribut mit Dimensionsinfos einer Matrix

```
a = np.asarray(range(10))
# Mit Slicing kann auf Teile der Daten zugegriffen werden
a[2:]  # Elemente von a ab Index 2 bis Ende (inklusive 2)
a[:3]  # Elemente von Anfang bis Index 3 (exklusive 3)
a[-1]  # Das erste Element von hinten
a[3:7:2]  # jedes zweite Element von Index 3 bis 7
          # (inklusive 3 exklusive 7)
```

Docs: https://numpy.org/doc/1.17/genindex.html

nächste Woche

- Mehr Zeit für Aufgaben
- Zeit für Fragen
- Lösungen