

# Einführung in das Programmieren

## Dateien und Dateisystem in Python

Christoph Draxler, Vanessa Reichel  
`draxler@phonetik.uni-muenchen.de`  
`vanessa.reichel@campus.lmu.de`  
`eva.thoma@campus.lmu.de`



22. November 2024

# Motivation

Wir kennen nun die wesentlichen Daten- und Kontrollstrukturen in Python:

- ▶ Variablen für den Zugriff auf Werte
- ▶ Funktionen als benannte Abläufe von Befehlen
- ▶ Fallunterscheidungen für verschiedene Abläufe in Programmen
- ▶ Listen (`list` und `dictionary`) zum Speichern vieler Werte unter einem Namen
- ▶ Schleifen zum wiederholten Ausführen von Befehlen

*Dateien* erlauben die dauerhafte Speicherung von Daten unabhängig von Programmen.

# Dateien: Einführung

Eine Datei (engl. *file*) ist ein benannter Speicherbereich im Dateisystem eines Rechners, d. h. auf einem Speichermedium wie Festplatte, Netzwerklaufwerk oder Speicherkarte. Moderne Dateisysteme sind hierarchisch – man sagt auch *baumartig* – strukturiert:

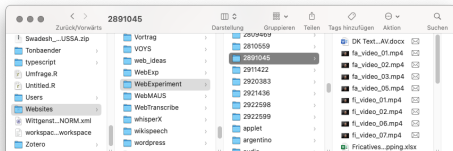
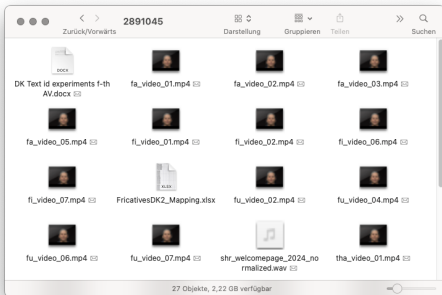
**Verzeichnis** oder *Ordner* enthält Dateien oder weitere Verzeichnisse

**Datei** enthält die eigentlichen Inhalte: Texte, Tabellenkalkulationsdaten, Grafiken, Fotos, Programmcode, oder Audio- und Videodaten

Das *Arbeitsverzeichnis* ist üblicherweise das Verzeichnis, aus dem heraus man ein -Programm startet. *Wurzel* bezeichnet das oberste Verzeichnis eines Teilbaums im Dateisystem.

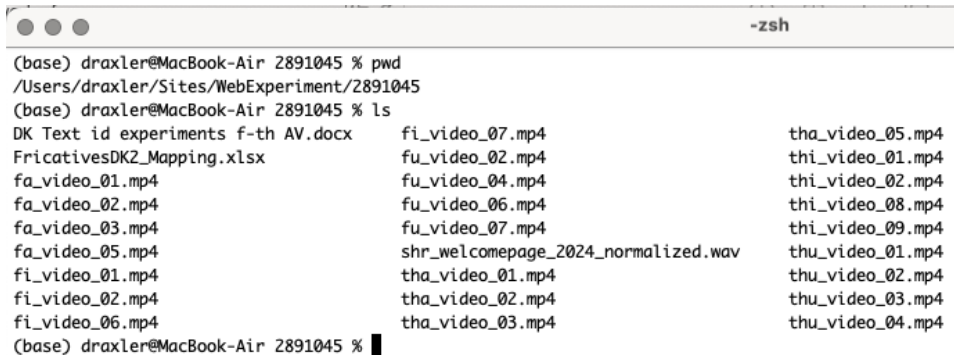
# Verzeichnisse und Dateien

Moderne Betriebssysteme bieten verschiedene Ansichten des Dateisystems (z. B. als Symbole, Liste oder Spalten).



# Verzeichnisse und Dateien im Terminal

Im Terminal befindet man sich immer in irgendeinem Verzeichnis, das man sich anzeigen lassen kann:

A screenshot of a macOS Terminal window. The title bar shows three window control buttons (red, yellow, green) on the left and the text "-zsh" on the right. The terminal text shows the user draxler@MacBook-Air in the directory 2891045. The 'pwd' command shows the full path: /Users/draxler/Sites/WebExperiment/2891045. The 'ls' command lists the contents of the directory in three columns. The files include text documents, Excel spreadsheets, and various video and audio files.

```
(base) draxler@MacBook-Air 2891045 % pwd
/Users/draxler/Sites/WebExperiment/2891045
(base) draxler@MacBook-Air 2891045 % ls
DK Text id experiments f-th AV.docx      fi_video_07.mp4      tha_video_05.mp4
FricativesDK2_Mapping.xlsx              fu_video_02.mp4      thi_video_01.mp4
fa_video_01.mp4                          fu_video_04.mp4      thi_video_02.mp4
fa_video_02.mp4                          fu_video_06.mp4      thi_video_08.mp4
fa_video_03.mp4                          fu_video_07.mp4      thi_video_09.mp4
fa_video_05.mp4                          shr_welcomepage_2024_normalized.wav  thu_video_01.mp4
fi_video_01.mp4                          tha_video_01.mp4      thu_video_02.mp4
fi_video_02.mp4                          tha_video_02.mp4      thu_video_03.mp4
fi_video_06.mp4                          tha_video_03.mp4      thu_video_04.mp4
(base) draxler@MacBook-Air 2891045 %
```

pwd gibt das aktuelle Verzeichnis an, ls zeigt die Dateien im Verzeichnis.

# Namenskonventionen

Je nach Betriebssystem gelten eigene Namenskonventionen für Verzeichnisse und Dateien.

- ▶ Prinzipiell können Datei- und Verzeichnisnamen alle Buchstaben und Ziffern sowie den Binde- und den Unterstrich enthalten, und ihre Länge ist beschränkt.
- ▶ Der *Pfad* ist die Beschreibung des Speicherorts einer Datei in einem Dateisystem. Er besteht aus den Verzeichnisnamen und je einem Trennzeichen dazwischen, z. B. `/usr/bin/sox` für die Datei `sox`<sup>1</sup> im Verzeichnis `bin` im Verzeichnis `usr`.

---

<sup>1</sup>sox ist der Befehl zum Ausführen der Audiosoftware sox

# Dateitypen

Generell unterscheidet man *Text-* und *Binärdateien*.

- ▶ Eine Textdatei enthält druckbare Zeichen, und sie ist in der Regel sowohl für Menschen als auch Rechner lesbar.
- ▶ Eine Binärdatei enthält Daten in maschinenlesbaren Kodierungen, z. B. Audiodaten als Folgen von 2 Byte langen ganzen Zahlen.

Python unterscheidet Text- und Binärdateien und stellt je nach Dateityp unterschiedliche Befehle zur Verfügung.

# Dateien öffnen

Mit dem Befehl `open()` öffnet man in Python eine Datei, entweder zum

**Lesen** ('r') Standardeinstellung, der Inhalt wird eingelesen

**Anfügen** ('a') neuer Inhalt wird an eine bestehende Datei angefügt

**Schreiben** ('w') Inhalt wird in eine Datei geschrieben, dabei wird vorhandener Inhalt überschrieben

**Anlegen** ('x') eine neue Datei wird erzeugt, wenn es sie nicht schon gibt

Mit 't' bzw. 'b' werden Text- und Binärdateien unterschieden.



## Textdatei öffnen und einlesen

Der folgende Code öffnet die Textdatei `zifferntabelle.txt` zum Lesen und weist den eingelesenen Inhalt der Variablen `text` zu.

```
with open('zifferntabelle.txt', 'tr') as datei:  
  
    text = datei.read()
```

Mit dem Ende des `with`-Blocks wird die Datei automatisch wieder geschlossen.

Zum Lesen muss die Datei bereits existieren, sonst bekommt man eine Fehlermeldung beim Ausführen des Befehls.

## Lesen aus Textdateien

Der Befehl `readline()` liest genau eine Zeile ein. Er ist nur für Textdateien definiert.

```
with open('zifferntabelle.txt', 'rt') as datei:
```

```
    zeile1 = datei.readline()
```

```
    zeile2 = datei.readline()
```

```
print(f'{zeile1} gefolgt von {zeile2}')
```

liest genau zwei Zeilen ein und gibt sie im Terminal aus.

## Kodierung explizit angeben

Damit der Dateiinhalt korrekt eingelesen wird, muss die Kodierung passen. Python erwartet eine Kodierung in UTF-8. Sieht die eingelesene Zeichenkette nicht so aus wie erwartet, muss die Kodierung evtl. angepasst werden.

```
with open('zifferntabelle.txt', 'rt', encoding='UTF-8')  
    as datei:
```

Hier bekommt die optionale Argumentvariable `encoding` den Wert `'UTF-8'` zugewiesen. Weitere übliche Kodierungen sind `'ascii'`, `'UTF16'`, `'latin-1'` uvam.

# Einlesen in einer Schleife

Eine praktische Notation ist das Einlesen von Zeilen in einer for-Schleife:

```
with open('zifferntabelle.txt', 'rt') as datei
```

```
    for zeile in datei:  
        print(f'{zeile}')
```

schreibt alle Zeilen der Datei `zifferntabelle.txt` der Reihe nach auf das Terminal<sup>2</sup>.

---

<sup>2</sup>Schauen Sie genau hin: Sind hier nicht ein paar Leerzeilen hinzugekommen?

## Einlesen in eine Liste

In einer for-Schleife kann der Inhalt einer Textdatei auch in eine Liste eingelesen werden:

```
with open('zifferntabelle.txt', 'rt') as datei
```

```
    zeilen = []  
    for zeile in datei:  
        zeilen.append(zeile)
```

```
print("Eingelesene Zeilen:", len(zeilen))
```

Die Liste `zeilen` enthält nun eine Zeile pro Eintrag.

## Und noch knapper: List Comprehension

Mit *list comprehension* kann man es noch knapper formulieren:

```
with open('zifferntabelle.txt', 'rt') as datei:  
    zeilen = [zeile.strip() for zeile in datei]
```

```
print(f'Eingelesene Zeilen: {len(zeilen)}')
```

Der Inhalt der Datei steht nach Ende der Ausführung in der Variablen `zeilen`, die Datei ist ordnungsgemäß geschlossen.

Was ist `.strip()` und was bewirkt es? Wie kriegt man das heraus?

# Dateipfade

Ein Pfad kann *absolut* oder *relativ* zum aktuellen Arbeitsverzeichnis sein.

Hier gibt es Unterschiede zwischen den Betriebssystemen:

- ▶ Unter Windows beginnt ein absoluter Pfad mit dem Laufwerksbuchstaben, z. B.  
`D:\\meineDateien\\willkommen.txt`.

Ein relativer Pfad fängt ohne den Laufwerksbuchstaben an.

- ▶ Unter Linux oder MacOS beginnt ein absoluter Pfad mit /, z. B.  
`/meineDateien/willkommen.txt`, ein relativer Pfad ohne.

- ▶ `./` bezeichnet das aktuelle Verzeichnis, `../` das übergeordnete.

`../Dateien/zifferntabelle.txt` ist ein relativer Pfad, der im übergeordneten Verzeichnis (`..`) das Verzeichnis `Dateien` und darin die Datei `zifferntabelle.txt` bezeichnet.

## Dateinamen eingeben und Datei einlesen

Mit der Funktion `input()` kann man eine Eingabe vom Terminal einlesen. Damit ist es z. B. möglich, während des Programmablaufs einen Dateinamen zu erfragen.

```
dateiname = input("Bitte Dateiname eingeben: ")  
with open(dateiname, 'rt') as datei:  
    inhalt = datei.read()  
  
print(inhalt)
```



# Code zur Eingabe eines Zielverzeichnis

```
import os.path

aktuell = os.getcwd();
ziel = input(f'Du bist aktuell im Verzeichnis {aktuell}. In welches Verzeichnis
möchtest Du gehen?') or aktuell

pfad = os.path.realpath(ziel)
if os.path.exists(pfad):
    if os.path.isdir(pfad):
        dateien = os.listdir(pfad)
        print(f'Dateien im Verzeichnis {pfad}: {dateien}')
    elif os.path.isfile(pfad):
        print(f'{pfad} ist eine Datei, kein Verzeichnis.')
    else:
        print(f'{pfad} ist weder Datei noch Pfad.')
else:
    print(f'Fehler: {pfad} ist kein gültiger Pfad.')
```

# Analyse des Codes

Der Code ist bereits recht komplex:

- ▶ Benötigt wird die Library `os.path`
- ▶ `aktuell = os.getcwd()` weist `aktuell` das aktuelle Arbeitsverzeichnis zu
- ▶ `ziel = input()` or `aktuell` weist `ziel` entweder die Eingabe zu, oder, wenn diese leer ist, den Wert aus `aktuell`

Er verwendet Funktionen zum Konvertieren und Prüfen von Pfaden:

`os.path.realpath()` konvertiert einen relativen in einen absoluten Pfad

`os.path.exists()` existiert dieser Pfad?

`os.path.isdir()` verweist der Pfad auf ein Verzeichnis?

`os.path.isfile()` verweist der Pfad auf eine Datei?

# Tabellen einlesen

Sehr häufig haben Textdateien eine innere Struktur – sie enthalten z. B. Tabellen, die pro Zeile mehrere durch Tabulator getrennte Felder enthalten.

```
with open('zifferntabelle.txt') as datei:

    for zeile in datei:
        ziffer, ortho, sampa, ipa = zeile.strip().split("\t")
        print(f'{ziffer}: {ipa}')
```

Hier wird in der ersten Zeile der Schleife die eingelesene Zeile am Tabulator getrennt und in einem Befehl jedes Feld einer eigenen Variablen zugewiesen.

# Zeilenende und Wagenrücklauf

In Textdateien sind Zeilenende und Wagenrücklauf eigene Zeichen, die den Text strukturieren. Die Funktion `strip()` löscht Leerzeichen, Zeilenende oder Wagenrücklauf am Beginn und am Ende einer Zeile, `strip('ab')` löscht alle Vorkommen der Zeichenkette 'ab'.

# Zugriff auf das Betriebssystem

Die Library `os` stellt Funktionen für den Zugriff auf das Betriebssystem zur Verfügung, z. B.

`os.getcwd()` gibt das aktuelle Arbeitsverzeichnis zurück

`os.listdir(Verzeichnis)` gibt die Dateien im Verzeichnis als Liste zurück

`os.remove(Datei)` löscht eine Datei

`os.mkdir(Verzeichnis)` legt ein neues Verzeichnis an

Falls das Verzeichnis bzw. die Datei nicht existiert, meldet Python einen Fehler.

## Achtung!

Einige dieser Befehle sind gefährlich: sie löschen, überschreiben oder verändern Daten  
unwiderruflich.

## Ein spezieller Befehl: `os.walk()`

`os.walk(Verzeichnis)` geht durch die komplette Dateihierarchie unterhalb des aktuellen Verzeichnisses.

```
for wurzel, verzeichnisse, dateien in os.walk('..'):  
    print(f'{wurzel} {verzeichnisse} {dateien}')
```

gibt zuerst die Verzeichnisse und Dateien im aktuellen Verzeichnis zurück, dann für jedes Verzeichnis darunter usw.

Dieser Befehl ist sehr praktisch, wenn man eine Auswahl hierarchisch organisierter Dateien bearbeiten möchte.

## Beispiel: Zifferntabelle einlesen

```
dateiname = input("Welche Datei?")

with open (dateiname, encoding="UTF-8") as datei:
    woerterbuch = {}

    # lies zunächst die Kopfzeile
    _ = datei.readline()

    # und nun lies den Rest der Datei ein
    for zeile in datei:
        ziffer, ortho, sampa, ipa = zeile.strip().split("\t")
        woerterbuch[ortho] = (sampa, ipa)
```

Die anonyme Variable `_` verwendet man in Python um anzuzeigen, dass man das Ergebnis einer aufgerufenen Funktion gar nicht weiter verwenden wird.

# Bibliotheken für spezielle Dateitypen

Verschiedene Python-Bibliotheken (engl. *library*) erlauben den einfachen Zugriff auf spezielle Datei-Typen:

`wave` Audiodateien inkl. Zugriff auf wichtige Signalparameter

`pandas` Excel, csv und andere Formate

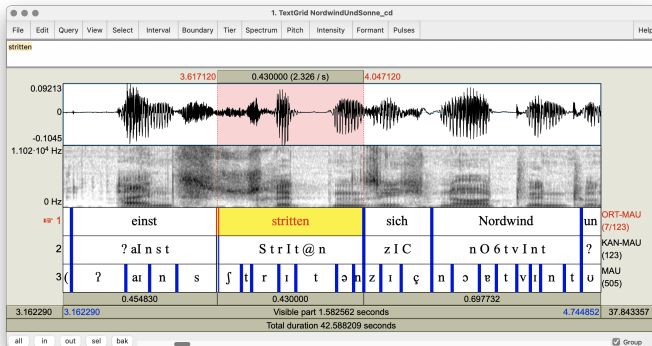
`praat-textgrids` Praat-TextGrid-Dateien

Meist muss eine Library erst installiert werden, bevor sie verwendet werden kann. Das erfolgt mit dem Befehl `pip install LIBRARY`.

In der Regel verwenden diese Bibliotheken die Nomenklatur der jeweiligen Datenformate, z.B. *Tier*, *Interval*, *Point* für TextGrid-Dateien.



# Praat TextGrid



Drei Annotationsebenen (engl. *tier*) mit *Segmenten*, z. B. dem gelb markierten Segment mit Start (3.617120) und Ende (4.047120) sowie dem Text "stritten" auf Ebene ORT-MAU.

# Beispiel Praat TextGrid: Tiers lesen

```
pip install praat-textgrids
```

Wenn diese Library installiert ist, können Sie TextGrid-Dateien einlesen:

```
import textgrids

def getTiersAsTexts (filename):
    grid = textgrids.TextGrid(filename)
    tiers = {}

    for tier in grid:
        tiertext = ""
        for segment in grid[tier]:
            tiertext += segment.text + " "

        tiers[tier] = tiertext.strip()

    return tiers
```

Die Funktion verbindet alle Segment-Texte jedes Tiers zu einem langen Text und gibt ein Dictionary dieser Texte zurück, mit dem Tier-Namen als Schlüssel.

# Aufgaben

## check Schröter seminar

1. Schreiben Sie eine Funktion `einlesen()`, die zur Eingabe eines Dateinamens auffordert, diese Datei dann zeilenweise in eine Liste einliest und dieses zurückgibt.
2. Schreiben Sie eine Funktion `woerter(zeile)`, die eine Textzeile in ihre Wörter zerlegt und diese als Liste zurückgibt.
3. Schreiben Sie eine Funktion `anzahlWoerter(zeilen)`, die die Anzahl aller Wörter aller Zeilen zählt.
4. Schreiben Sie eine Funktion `wortHaeufigkeiten(zeilen)`, die die Häufigkeit der einzelnen Wörter zählt.
5. Schreiben Sie eine Funktion, die die oben definierten Funktionen in sinnvoller Weise kombiniert.