



Grundbegriffe des Programmierens

Vorlesung *Einführung in die Digital Humanities*
MSc Digital Humanities | Wintersemester 2020/21

Prof. Dr. Christof Schöch



Einstieg

Semesterüberblick

- 10.11.: Digital Humanities im Überblick
- 17.11.: **Grundbegriffe des Programmierens**
- 24.11.: Digitalisierung: Text und Bild
- 01.12.: Datenmodellierung 1: Modellierung
- 08.12.: Datenmodellierung 2: Datenbanken
- 15.12.: Textkodierung und Digitale Edition
- 22.12.: Informationsvisualisierung
- *Weihnachtspause*
- 05.01.: Geschichte der Digital Humanities
- 12.01.: Natural Language Processing
- 19.01.: Grundlagen der Statistik
- 26.01.: Quantitative Analyse: Maschinelles Lernen
- 02.02.: Open Access, Open Science
- letzte Woche: Klausurtermin

Sitzungsüberblick

1. Einstieg: Programmiersprachen
2. Grundbegriffe: Variablen, Datentypen, Operatoren
3. Grundbegriffe: Datenstrukturen, Funktionen und Methoden
4. Programmstrukturen: Schleifen, Bedingungen, Verzweigungen
5. Bausteine eines Programms
6. Beispiel: Anzahl der Worte in einem Satz
7. "Think like a programmer" -- aber wie?

1. Einstieg: Programmiersprachen

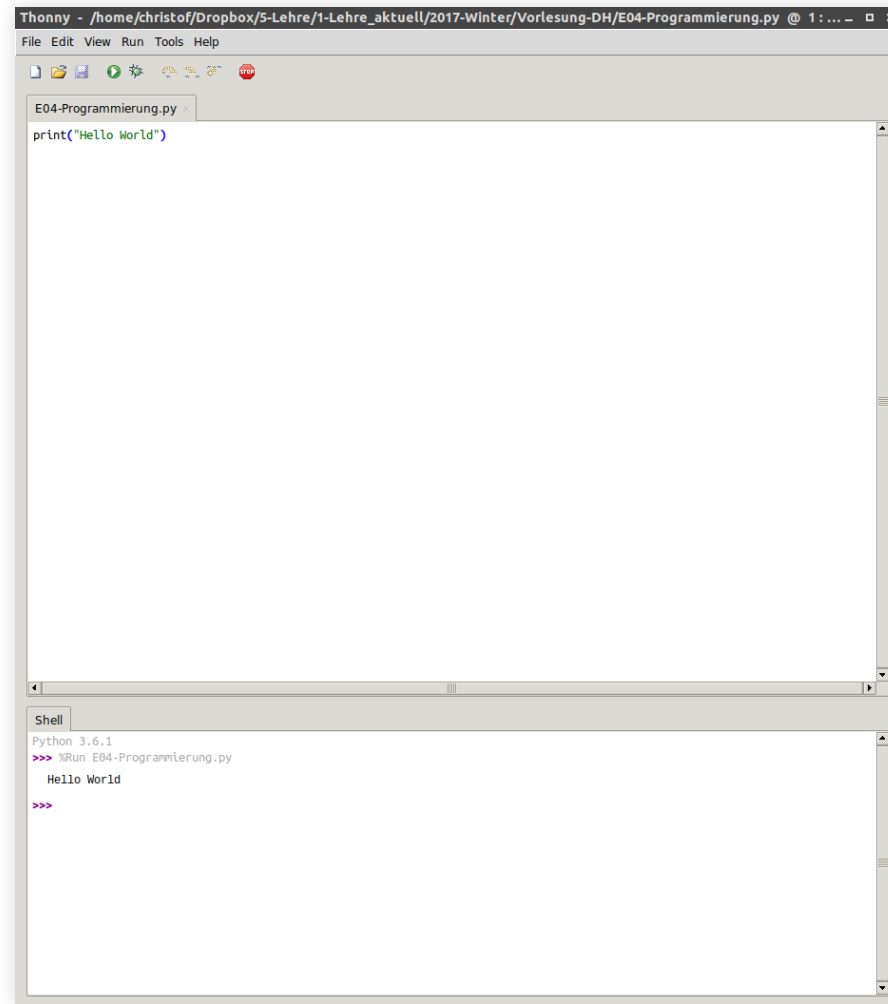
Typen von Programmiersprachen

- Ausführung immer in "Maschinensprache"
- Assembler-Programmierung: sehr nah an der Maschinensprache
- "kompilierte" Programmiersprachen
 - Pascal, Java, C / C++
 - Kompilieren zu Assembler, dann in Maschinensprache
 - mächtig und performant
- "interpretierte" Sprachen / Skript-Sprachen
 - PHP, Perl oder Python
 - der Interpreter stellt Routinen für die Ausführung bereit
 - einfache Fehlersuche, weniger performant

Beispiel-Sprache: Python

- Eine Skript-Sprache, die uns einige Details abnimmt
 - interpretierte Sprache (Prototyping)
 - implizite Datentypen in Variablen
 - automatische Speicherverwaltung
- für Menschen ziemlich gut lesbar
- performant v.a. durch darunterliegende Implementierungen in C
- Zahlreiche sog. IDEs
 - "Integrated Development Environments"
 - Beispiele: Thonny, Geany, Atom, Spyder, PyCharm, etc.

Thonny



<http://thonny.org/>

2. Grundbegriffe

Algorithmus

- Folge von Anweisungen, um ein bestimmtes Ziel zu erreichen
- (ähnlich: Kochrezept)

Variablen

```
busnummer = 13  
busnummer = 10 + 3
```

- "busnummer" ist der Variablenname
- "=" ist das Zuweisungssymbol
- "13" ist der Variablenwert
- "10 + 3" ist ein Ausdruck
- Struktur: "Variable = Ausdruck"
- alles zusammen ist eine "Anweisung"

Umgang mit Variablen

```
preis = 130  
print(preis)  
preis = preis + 10  
print(preis)  
preis += 10 # kürzer  
print(preis)
```

Datentypen

- "Zusammenfassung konkreter Wertebereiche und darauf definierten Operationen zu einer Einheit"
- Beispiele für "Wertebereiche":
 - int ("integer", also ganze Zahlen)
 - float (Gleitkommazahl)
 - bool (Wahrheitswert: True oder False)
 - string (Zeichenkette)
- Definierte Operationen:
 - dazugehörige Methoden
 - passende Operatoren
 - geeignete Funktionen

Datentypen

```
alter = 12  
name = "Maus"  
groesse = 12.26  
groesse2 = "12,6"  
istmaus = True  
  
print(type(alter))
```

Datentypen und Operatoren / Methoden (1)

```
mystring = "Eine Maus"  
print(mystring)  
print(type(mystring))  
length = len(mystring)  
print(length)  
print(mystring.upper())  
print(mystring.lower())
```

Datentypen und Operatoren (2)

```
wert1 = 3  
wert2 = 7  
ergebnis = wert1 + wert2  
print(ergebnis)
```


Datentypen und Operatoren (3)

```
flaschenWein = 23  
anzahlGaeste = 15  
ergebnis = flaschenWein / anzahlGaeste  
print(ergebnis)
```

Datentypen und Operatoren (4)

```
wert1 = "Heinrich"  
wert2 = "Faust"  
ergebnis = wert1 + wert2  
print(ergebnis)
```

Datentypen und Operatoren (5)

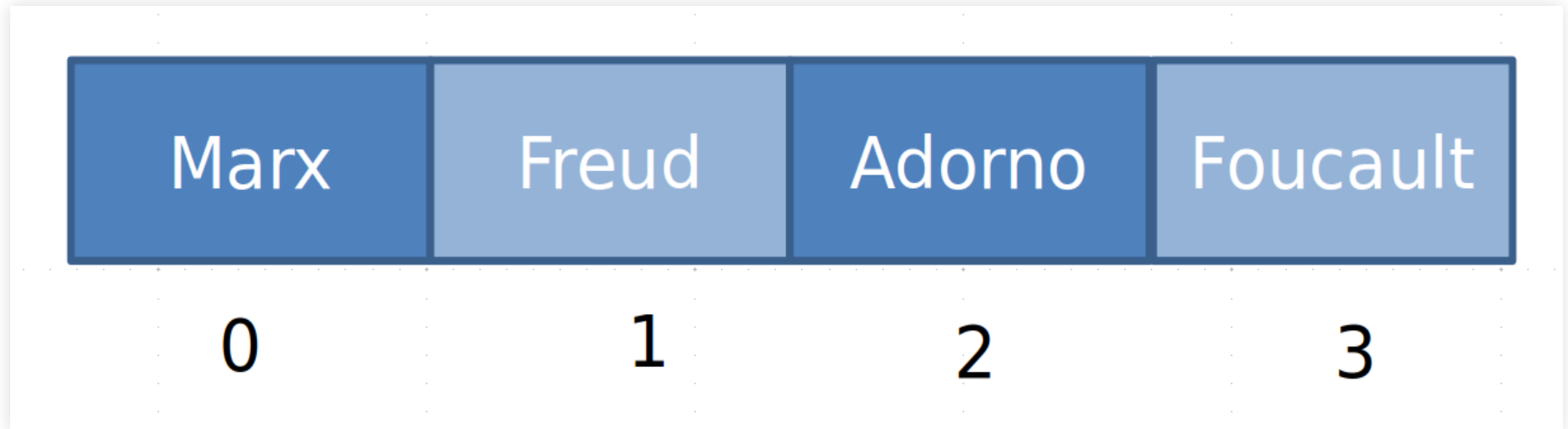
```
alter = 8  
alter = "acht"  
ergebnis = age + alter  
print(ergebnis)
```

3. Datenstrukturen und Operatoren

Datenstrukturen

- Bestimmtes Format, in dem die Daten vorliegen und vom Algorithmus angesprochen werden
- Beispiele:
 - Liste (list)
 - Wörterbuch (dict)
 - Matrix (dataframe)

Datenstruktur: Liste



- Sequenzielle Struktur, durchgezählt ("Hausnummern")
- Jede Position in der Struktur kann einen Wert haben ("Familiennamen")
- Python: Zählung startet by 0 (!)

Liste und Operatoren

```
meineListe = ['Marx', 'Freud', 'Adorno', 'Foucault']  
print(meineListe)  
print(meineListe[0])  
print(len(meineListe))  
print(meineListe[1:3])
```

Liste und Operatoren

```
meineListe = ['Marx', 'Freud', 'Adorno', 'Foucault']  
print(meineListe)  
meineListe[1] = 'Weber' # Zuweisung überschreibt  
print(meineListe)  
meineListe.append('Luhmann') # Anhängen  
print(meineListe)
```


Funktion

```
meinText = "Ein kurzer Text."  
len(meinText)  
len("Ein kurzer Text")  
textlaenge = len(meinText)  
print(textlaenge)
```

- Funktion (bspw.: len, print)
- Parameter (Input für die Funktion: Daten, Parameter)

Objekt und Methode

```
text = "Aber Hallo"  
neuerText = text.lower()  
print(neuerText)
```

- "text": Ein Objekt vom Typ string
- "neuerText": Rückgabewert der Methode (in Variable)
- "lower()": Methode aller Objekte vom Typ "string"
- "()": kann Parameter der Methode enthalten

Interaktion

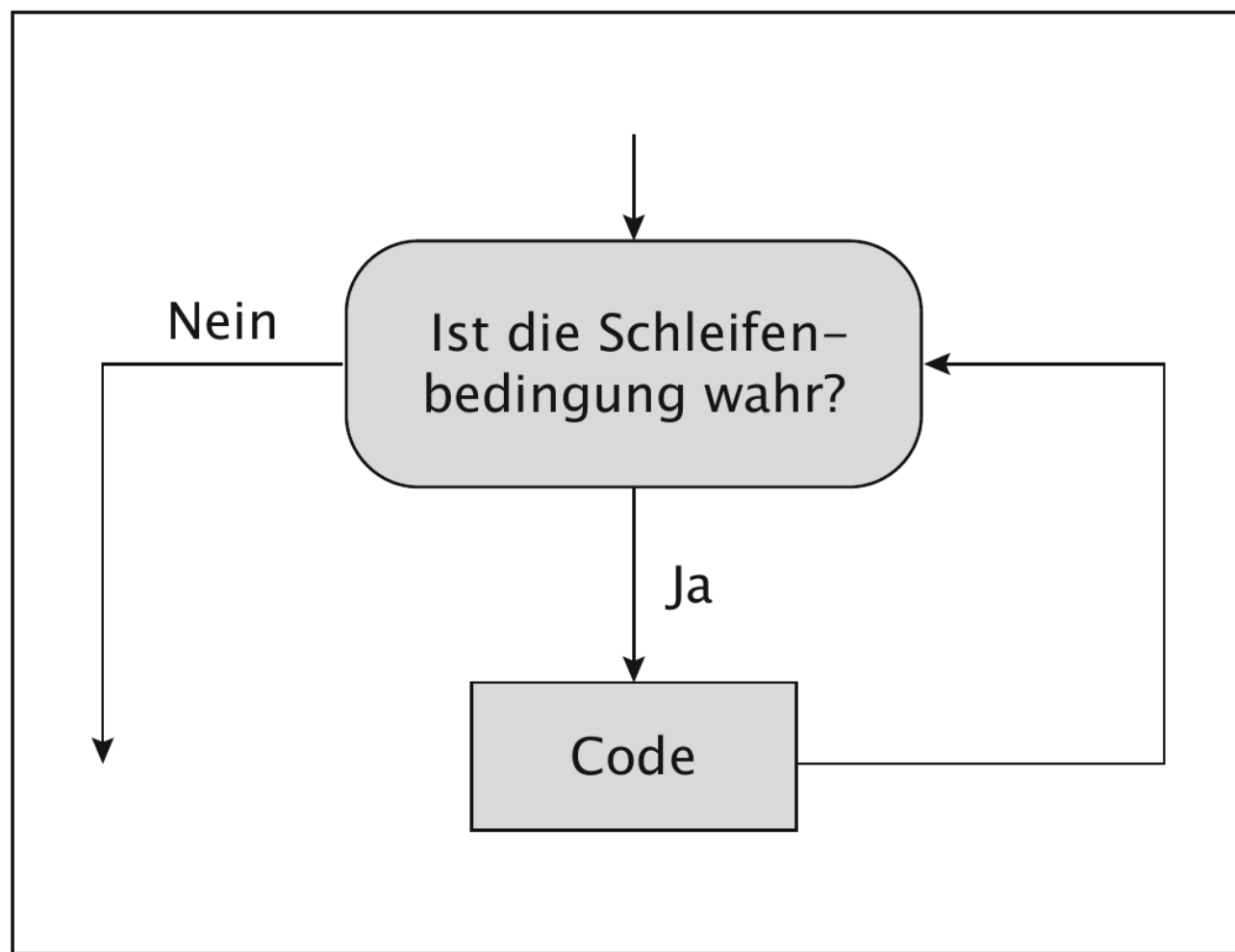
```
name = input("Bitte Namen eingeben: ")  
print(name, " - so ein schöner Name!")
```

Interaktion

```
zahl1 = int(input("Ich multipliziere gerne. Bitte erste Zahl eingeben: "))  
zahl2 = int(input("Danke. Bitte eine zweite Zahl eingeben: "))  
print("Danke! Das Ergebnis lautet", zahl1 * zahl2)
```

4. Programmstrukturen: Schleifen, Bedingungen, Verzweigungen

while-Schleife



while-Schleife

- So lange eine Bedingung wahr ist, führe einen bestimmten Code aus und prüfe wieder die Bedingung; wenn die Bedingung nicht wahr ist, führe den Code nicht aus.

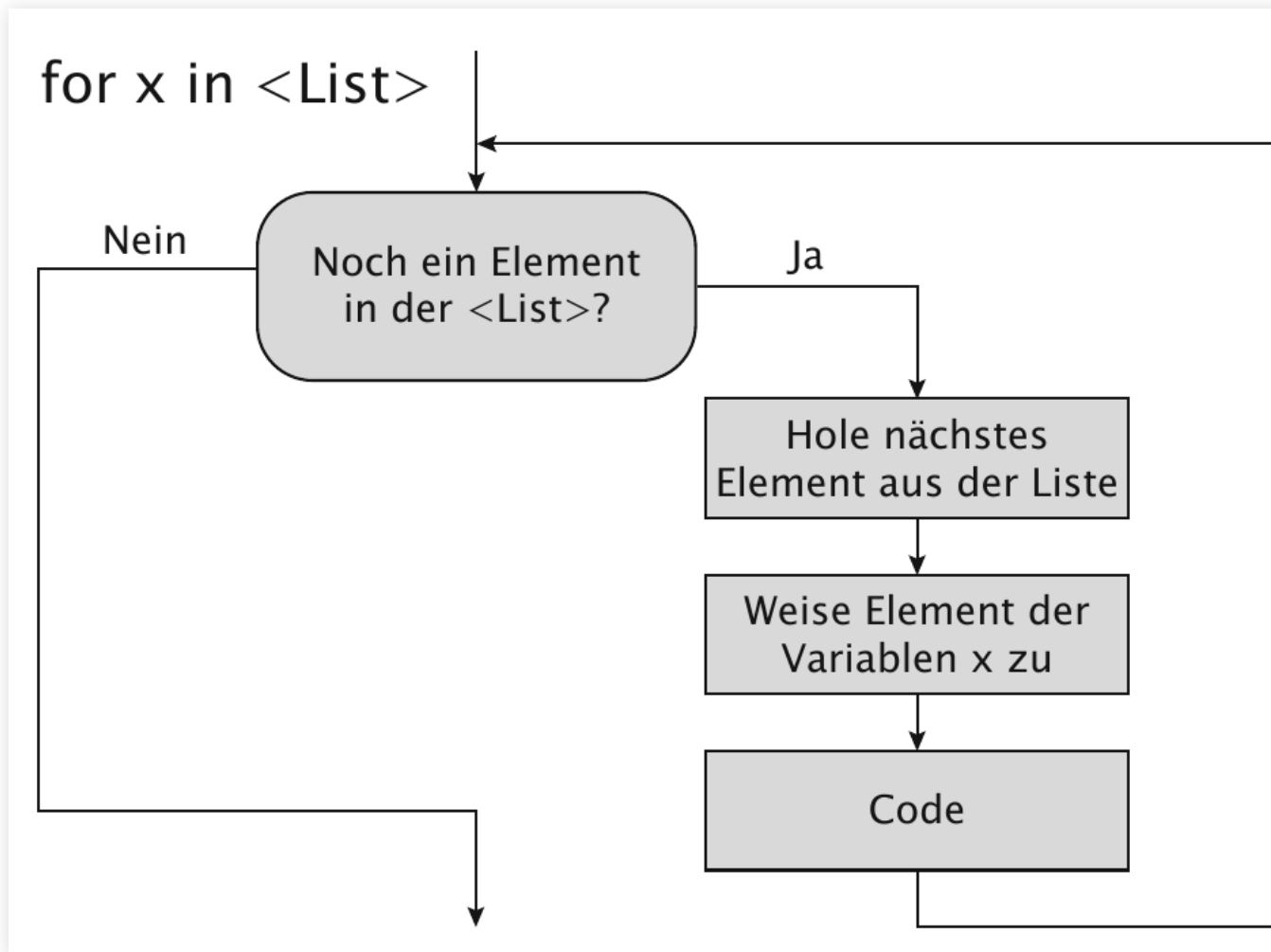
```
value = 12
while value > 5:
    print(value)
    value = value - 2
print("End of loop")
```

Bedingungen

- Bedingungen geben einen TRUE oder FALSE-Wert zurück

```
x = 1
y = 0
print(x < y)    # testet "kleiner als"
print(x == y)   # testet auf Gleichheit
print(x >= y)    # testet „größer/gleich“
print(x != y)   # testet auf Ungleichheit
print("AU" in "LAUFEN") # enthalten?
print("AU" not in "LAUFEN") # nicht enthalten?
```


for-Schleife



for-Schleife

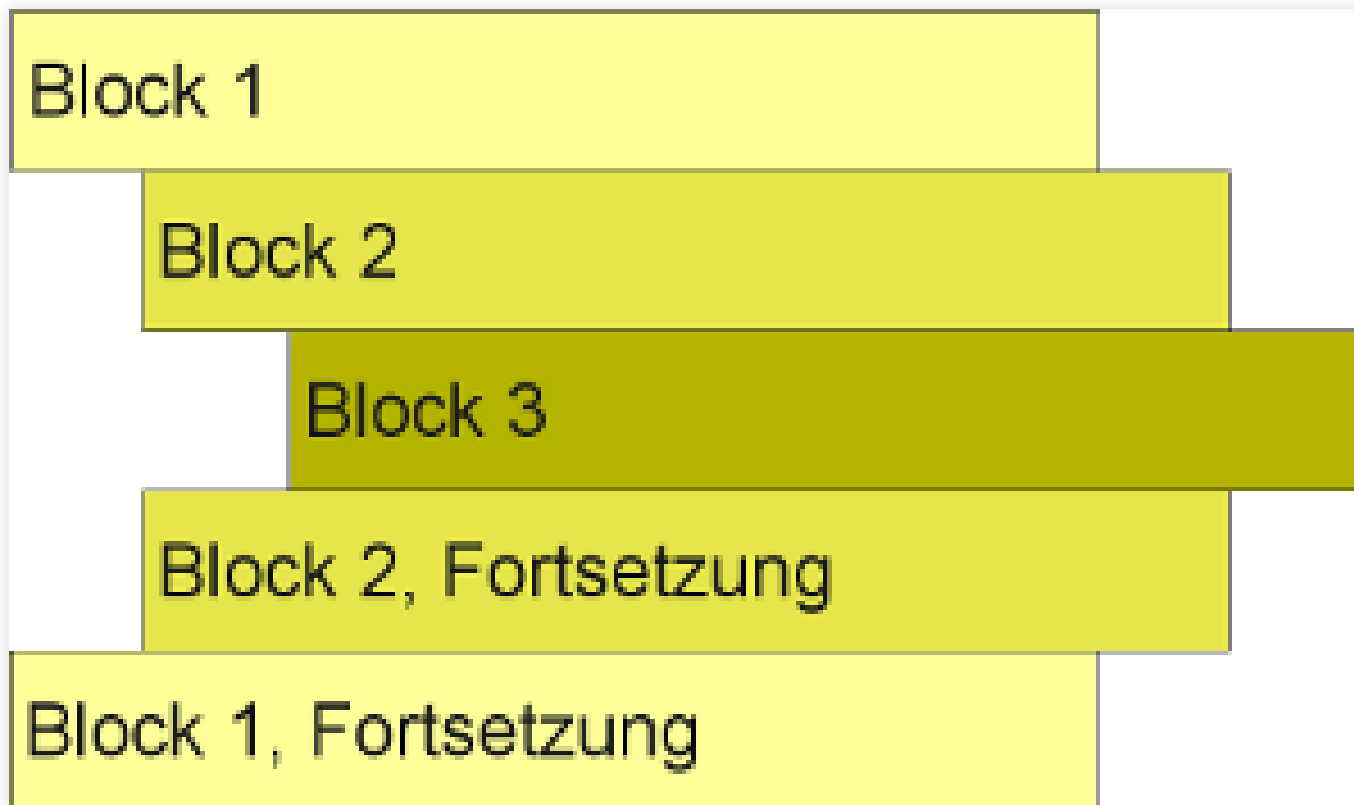
- Für jedes Element einer sequenziellen Datenstruktur, führe einen bestimmten Code aus, bis das Ende der Sequenz erreicht ist.

```
mysequence = [4,2,8,7]
for element in mysequence:
    print(element)
print("End of loop")
```

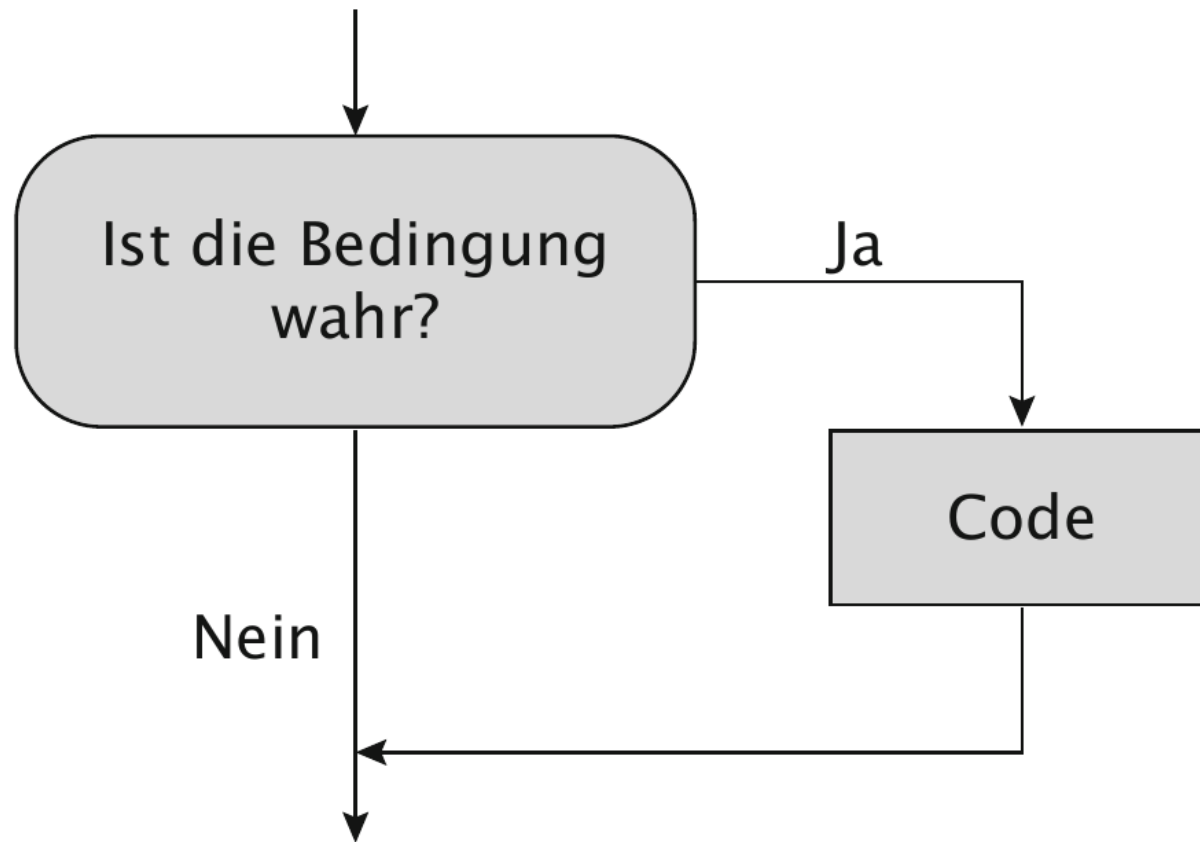
for-Schleife (Iteration)

```
mysequence = "Katze"  
for element in mysequence:  
    print(element)  
print("End of loop")
```

Blöcke in Python: Einrückung



Verzweigung



if/elif/else-Verzweigung

- Wenn die Bedingung 1 wahr ist, dann führe die Anweisungen 1 aus.
- (Ansonsten, wenn die Bedingung 2 wahr ist, dann führe die Anweisungen 2 aus.)
- Ansonsten, führe die Anweisungen 3 aus.

Beispiel if/elif/else

```
string = "Käse"  
print(string)  
if "ä" in string:  
    print("Ah! There is an ä.")  
elif "ö" in string:  
    print("Ah! There is an ö.")  
else:  
    print("Oh. No matches.")
```

Zusammengesetzte Bedingungen

```
string = "Käsekönig"  
print(string)  
if "ä" in string and "ö" in string:  
    print("Ah! There is an ä and an ö.")  
else:  
    print("Oh. No matches.")
```

- Bedingung and Bedingung
- Bedingung or Bedingung

5. Bausteine eines Skripts

Abstrakte Ebene: Funktionen

- Input (Daten und Parameter)
- Algorithmus
 - Operationen, Methoden
 - Kontrollstrukturen: Bedingungen, Schleifen
- Output (Ergebnis)

Beispielaufgabe

- Zerlegen Sie einen Satz in seine Wörter; zeigen Sie die Wörter an und geben Sie die Anzahl der Wörter aus.
- Beispielsatz: "Die Katze sucht den Käsekönig"

Pseudocode

- Input: Beispielsatz als string
- Lege fest, was ein Buchstabe ist / was ein Trennzeichen ist
- Datenstruktur list (für die Wörter)
- Datenstruktur string (für die Buchstaben eines Worts)
- Kontrollstruktur:
 - Gehe durch den String, sammle die Buchstaben in der Buchstabenliste, bis ein Leerzeichen kommt
 - Wenn ein Leerzeichen kommt, sind die gesammelten Buchstaben ein Wort; füge das Wort der Wörterliste hinzu; lösche den Inhalt des Wortes
 - Wiederhole dieses Verfahren, bis das Ende des strings erreicht ist
- Zeige die Wörter in der Wörterliste an
- Zeige die Länge der Wörterliste an

Potentielle Probleme

- Was ist ein Buchstabe, was ist ein Trennzeichen?
- Wie können wir Daten zwischenspeichern?

Beispiellösung

```
satz = "Die Katze sucht den Käsekönig"
trenner = " "
wörter = []
wort = ""
for element in satz:
    if element != trenner:
        wort = wort+element
    else:
        wörter.append(wort)
        wort = ""
print(wörter)
print(len(wörter))
```

Beispiellösung (Funktion)

```
satz = "Die Katze sucht den Käsekönig"
def count_words(satz):
    trenner = " "
    wörter = []
    wort = ""
    for element in satz:
        if element != trenner:
            wort = wort+element
        else:
            wörter.append(wort)
            wort = ""
    return wörter

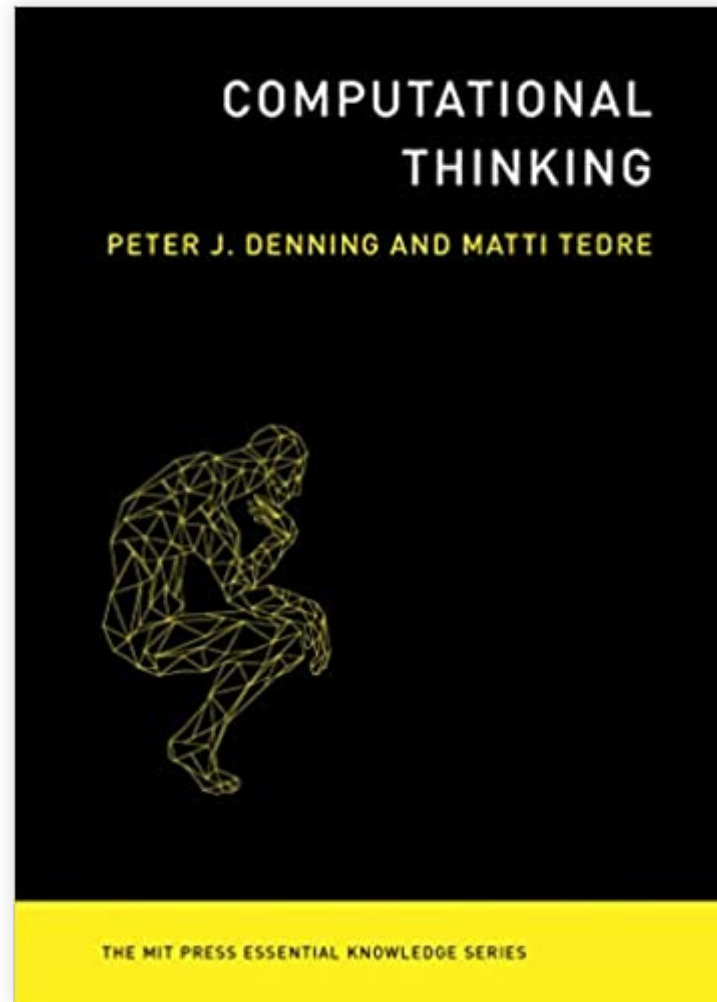
wörter = count_words(satz)
print(wörter)
print(len(wörter))
```

Etwas komplexeres Skript

- test.py (separat)
- Siehe: Funktionen, Funktionsaufrufe
- Input, Output
- Lesbarkeit

6. "Think like a programmer"
– aber wie?

Computational Thinking



Peter J. Denning & Matti Tedre, Computational Thinking, 2019.

Aspekte

- Fragestellungen operationalisieren
- Abläufe zerlegen
- Vorausschauend denken
- Überblick bewahren
- Effiziente Lösungen finden

Entwicklungsschritte

1. Problem analysieren
2. Spezifikation erstellen
3. Design erstellen
4. Design implementieren
5. Programm testen, Fehler suchen
6. Programm dokumentieren
7. Programm warten / weiterentwickeln

Abschluss

Lektürehinweise

- Fotis Jannidis, "Grundbegriffe des Programmierens", in: *Digital Humanities: Eine Einführung*, hg. von Fotis Jannidis, Hubertus Kohle und Malte Rehbein. Stuttgart: Metzler, 2017, S. 68-95.

Weitere Empfehlungen

- Bernd Klein, *Python 3-Tutorial*, 2014. https://www.python-kurs.eu/python3_kurs.php
- *Das Python-Tutorial 3.3*. Read the docs. <http://py-tutorial-de.readthedocs.io/de/python-3.3/>
- Peter J. Denning & Matti Tedre, *Computational Thinking*, MIT Press, 2019.

Darüber hinaus

- John Zelle. *Python Programming. An Introduction to Computer Science*. 2nd edition. Franklin, Beedle and Associates, 2009.
- Steven Bird, Ewan Klein, Edward Loper. *Natural Language Processing with Python*. O'Reilly, 2009.
- Kathrin Passig und Johannes Jander. *Weniger schlecht programmieren*. O'Reilly, 2013.



Christof Schöch, 2020
<http://www.christof-schoech.de>

Lizenz: Creative Commons Attribution 4.0