Unterlagen für den Informatikunterricht in der Klasse 1fP

Jacques Mock Schindler

2026-01-11

Table of contents

1 Obligatorisches Fach Informatik (2fP)

In diesem Repository sollen die Unterlagen für das obligatorische Fach Informatik zusammengetragen werden.

1.1 Programm

Das Programm entspricht dem aktuellen Stand der Planung. Es kann zu Änderungen kommen. Das bisherige Skript findet sich hier.

Datum	Thema
22.08.2025	Netzwerke: TCP/IP
29.08.2025	Netzwerke: Beobachtung von Netzwerkverbindungen
05.09.2025	Netzwerke: Anwendungsübung
12.09.2025	Kryptologie: Caesar
19.09.2025	Kryptologie: Vigenère Chiffre
26.09.2025	Kryptologie: Vigenère Brechen
24.10.2025	Test
31.10.2025	Kryptologie: Public Key Kryptographie (Graphengestützt)
07.11.2025	KI: The Social Dilemma
14.11.2025	Kryptologie: Dominante Graphen
21.11.2025	Kryptologie: RSA und Signaturen
28.11.2025	KI: Definition
05.12.2025	Test
12.12.2025	KI
19.12.2025	KI
09.01.2026	KI

1.2 Hilfsmittel für den Unterricht

Der Unterricht findet über weite Strecken am Computer statt. Die Anleitungen für die Installation von Python bzw. der Jupyter Notebooks sind in der Seitenliste aufrufbar.

1.3 Beurteilung

Die Note wird aus dem Durchschnitt der beiden schriftlichen Prüfungen sowie der Benotung der mündlichen Beteiligung berechnet. Der Durchschnitt der Prüfungen zählt zu 90%, die mündliche Beteiligung zu 10%.
Ab dem zweiten Semester wird konstruktive Kritik an den Unterlagen, welche durch ein mir zugewiesenes Issue (der Zugang findet sich unter dem GitHub-Logo  in der Kopfzeile) eingebracht wird, mit Maximal einem Notenpunkt auf die jeweils thematisch passende Prüfung angerechnet.

Falls jemand eine persönliche Besprechung wünscht, kann hier einen Termin für eine Sprechstunde reservieren (Rent-A-Mock).

Part I

Start

Part II

Kryptologie

2 File Handling in Python

Um Textdateien in Python zu bearbeiten, schreiben wir eine Funktion, mit welcher der Inhalt einer Datei einer Variabel als String zugewiesen wird.

```
def file_reader(path : str) -> str:

    with open(path, mode='r', encoding='utf-8') as f:
        text = f.read()

    return text
```

Um ver- oder entschlüsselte Texte in eine Datei zu schreiben, schreiben wir eine Funktion, die einen String in eine Datei schreibt.

```
def file_writer(path : str, text : str) -> None:
    i = 0
    grouped_text = ""
    for c in text:
        i += 1
        if i % 50 == 0:
            grouped_text += c + "\n"
        elif i % 5 == 0:
            grouped_text += c + " "
        else:
            grouped_text += c

    with open(path, mode='w', encoding='utf-8') as f:
        f.write(grouped_text)
```

Damit Texte ausschliesslich aus ASCII Grossbuchstaben bestehen, schreiben wir eine Funktion, die alle Kleinbuchstaben in Grossbuchstaben umwandelt und alle Umlaute in ihre aequivalenten Buchstaben umwandelt. Alle anderen Zeichen werden entfernt.

Damit alle Methoden für die Bearbeitung von Strings zur Verfügung stehen, muss das String-Modul importiert werden.

```
import string
```

```python
def text_cleaning(text : str) -> str:
    clean = text.upper() \
                .replace('Ä', 'AE') \
                .replace('Ö', 'OE') \
                .replace('Ü', 'UE') \
                .replace('ß', 'SS') \
                .replace(' ', '') \

    cleaned_text = ''

    for c in clean:
        if c.isalpha():
            cleaned_text += c

    return cleaned_text
```

3 Caesar Chiffre

In der Kryptologie wird mit Verschlüsselung jener Prozess beschrieben, der eine Nachricht so transformiert, dass sie idealerweise nur von autorisierten Parteien gelesen werden kann. Dieser Prozess wandelt die ursprüngliche Darstellung der Informationen, bekannt als Klartext, in eine alternative Form, bekannt als Chiffretext, um. Trotz seines Ziels verhindert die Verschlüsselung nicht selbst die Interferenz, sondern verweigert dem potenziellen Abhörer den verständlichen Inhalt.

Aus Wikipedia

Das erste Beispiel für eine Chiffre, das wir uns ansehen werden, ist die Caesar-Chiffre.

Die Caesar-Chiffre ist eine einfache Substitutionsverschlüsselungstechnik, bei der jeder Buchstabe des zu verschlüsselnden Textes durch einen Buchstaben ersetzt wird, der sich um eine feste Anzahl von Positionen im Alphabet verschiebt. Zum Beispiel würde bei einer Verschiebung um vier Positionen nach rechts `A` durch `E` ersetzt werden, und das Wort `CIPHER` würde zu `GMTLIV`. Die Technik ist nach Julius Caesar benannt, der sie in seinen Briefen verwendete. Die Einfachheit der Caesar-Chiffre macht sie zu einer beliebten Quelle für Freizeit-Kryptogramme.

Aus Encyclopedia Britannica

3.1 Python Implementation

Um die Caesar-Chiffre in Python zu implementieren, bauen wir auf den von Python angebotenen String-Methoden auf. Die Funktion `ord` gibt den Unicode-Codepunkt für ein gegebenes Zeichen zurück, und die Funktion `chr` gibt das Zeichen zurück, das dem gegebenen Unicode-Codepunkt entspricht.

```
# Was ist der Unicode Wert des Buchstabens 'A'?
```

3.1.1 Simple (naive) Implementation

Als erstes definieren wir eine Funktion, die einen gegebenen Klartext verschlüsselt, indem sie jeden Buchstaben um eine bestimmte Anzahl von Positionen im Alphabet verschiebt.

```
plain = "CIPHER"
shift = 4
```

```
def caesar_encode(plain, shift):
    cipher = ""

    for char in plain:
        shifted = ord(char) + shift
        cipher += chr(shifted)

    return cipher
```

3.1.2 Verbesserte Implementation

Was geschieht aber, wenn wir das Ende des Alphabets erreichen? Zum Beispiel, wenn wir `Z` um 4 Positionen verschieben, würden wir über `Z` hinausgehen. Um dies zu handhaben, können wir den Modulo-Operator `%` verwenden, um im Alphabet von vorne zu beginnen. Der Modulo-Operator gibt den Rest einer Division zurück, was es uns ermöglicht, wieder bei Null zu beginnen, wenn wir die Länge des Alphabets überschreiten. Im Fall der Caesar-Chiffre können wir ihn verwenden, um sicherzustellen, dass unsere verschobenen Positionen innerhalb der Grenzen des Alphabets bleiben.

Daher verwenden wir modulo 26 (die Anzahl der Buchstaben im lateinischen Alphabet), um sicherzustellen, dass unsere verschobenen Positionen korrekt "herumwickeln". Wenn wir `Z` um 4 Positionen verschieben, landen wir bei `D`. Dieses Wickelverhalten ist für die Funktionsweise der Caesar-Chiffre von entscheidender Bedeutung. Die Berechnung der neuen Position eines Buchstabens kann wie folgt ausgedrückt werden:

$$x' = (x + n) \mod 26$$

oder in einer kompakteren Form mit dem Modulo-Additionsoperator:

$$x' = x \oplus_{26} n$$

In Python können wir dies wie folgt implementieren:

```
def caesar_encrypt_mod(plain, shift):
    cipher = ""

    for char in plain:
        shifted = (ord(char) - ord('A') + shift) % 26 + ord('A')
        cipher += chr(shifted)

    return cipher
```

Weil

```
ord('A')
```

65 zurückgibt, müssen wir 65 von dem Ergebnis von `ord('A')` subtrahieren, um 0 zu erhalten. Dies gibt uns den 0-basierten Index des Buchstabens im Alphabet, der für unsere Berechnungen nützlich ist.

Daher die oben gezeigte Berechnung.

Für die Entschlüsselung können wir einfach den Verschiebungswert subtrahieren anstatt ihn zu addieren:

```
def caesar_decrypt_mod(plain, shift):
    cipher = ""

    for char in plain:
        shifted = (ord(char) - ord('A') - shift) % 26 + ord('A')
        cipher += chr(shifted)

    return cipher
```

Für die Bequemlichkeit implementieren wir eine Funktion die sowohl ver- als auch entschlüsselt.

```
def caesar(text : str, shift : int, encrypt=True) -> str:
    text = text.upper()
    result = ""

    if encrypt:
        for char in text:
            shifted = (ord(char) - ord('A') + shift) % 26 + ord('A')
            result += chr(shifted)
    else:
        for char in text:
            shifted = (ord(char) - ord('A') - shift) % 26 + ord('A')
            result += chr(shifted)

    return result
```

3.2 Caesar Chiffre brechen

Es gibt zwei Hauptmethoden, um eine Caesar-Chiffre zu brechen:

1. Brute Force Attack: Versuchen Sie alle möglichen Verschiebungen (1-25) und sehen Sie, welche eine sinnvolle Ausgabe erzeugt.

2. Frequency Analysis: Analysieren Sie die Häufigkeit von Buchstaben im Chiffretext und vergleichen Sie sie mit der erwarteten Häufigkeit von Buchstaben in der Sprache.

3.2.1 Brute Force Attack

```python
def caesar_bruteforce(ciphertext: str) -> None:
    for shift in range(1, 26):
        decrypted_text = caesar(ciphertext, shift, encrypt=False)
        print(f"Shift {shift}: {decrypted_text}")
```

3.2.2 Häufigkeitsanalyse

In der deutschen Sprache erscheinen bestimmte Buchstaben häufiger als andere. Zum Beispiel ist der Buchstabe 'E' der häufigste Buchstabe in deutschen Texten, gefolgt von 'N', 'I', 'S', 'R' und 'A'. Durch die Analyse der Häufigkeit von Buchstaben im Chiffretext können wir fundierte Vermutungen darüber anstellen, welche Buchstaben im Klartext welchen im Chiffretext entsprechen.

Für eine genauere Analyse siehe die folgende Häufigkeitstabelle:

Platz	Buchstabe	Relative Häufigkeit
1.	E	17,40 %
2.	N	9,78 %
3.	I	7,55 %
4.	S	7,27 %
5.	R	7,00 %
6.	A	6,51 %
7.	T	6,15 %
8.	D	5,08 %
9.	H	4,76 %
10.	U	4,35 %
11.	L	3,44 %
12.	C	3,06 %
13.	G	3,01 %
14.	M	2,53 %
15.	O	2,51 %
16.	B	1,89 %
17.	W	1,89 %
18.	F	1,66 %
19.	K	1,21 %
20.	Z	1,13 %
21.	P	0,79 %
22.	V	0,67 %

Platz	Buchstabe	Relative Häufigkeit
23.	J	0,27 %
24.	Y	0,04 %
25.	X	0,03 %
26.	Q	0,02 %

Quelle: Wikipedia

Wir brauchen also eine Funktion, die die Häufigkeit jedes Buchstabens im Chiffretext zählt. Das Ergebnis sollte ein Dictionary sein, bei dem die Schlüssel die Buchstaben und die Werte die Zählungen jedes Buchstabens sind. Oder noch besser, die Häufigkeiten in Prozent.

Um die Sache einfach zu halten, nehmen wir an, dass der Eingabetext nur Großbuchstaben von `A` bis `Z` und keine Leerzeichen oder Satzzeichen enthält.

```
def letter_frequency(text: str) -> dict:
    frequency = {}
    total_letters = 0

    for char in text:
        if char not in frequency:
            frequency[char] = 1
        else:
            frequency[char] += 1
        total_letters += 1

    for key, value in frequency.items():
        frequency[key] = (value / total_letters) * 100


    return frequency
```

Nachdem wir den häufigsten Buchstaben im Chiffretext gefunden haben, können wir annehmen, dass er dem Buchstaben ‘E’ im Klartext entspricht. Indem wir die Verschiebung berechnen, die erforderlich ist, um den häufigsten Buchstaben in ‘E’ zu verwandeln, können wir dann den gesamten Chiffretext mit diesem Verschiebungswert entschlüsseln.

```
def find_shift(char: str, e = 'E') -> int:
    return (ord(char) - ord(e)) % 26
```

4 Polyalphabetische Chiffren

(Vigenère Chiffre)

Bereits im 9. Jahrhundert wurde im islamischen Raum die grosse Schwachstelle monoalphabetischer Chiffren (Caesar-Chiffre) erkannt. Die Verteilung der Buchstaben folgt in jeder Sprache einem spezifischen aber konstanten Muster. Für die deutsche Sprache ist die Verteilung der folgenden Tabelle zu entnehmen.

Buchstabe	relative Häufigkeit	Buchstabe	relative Häufigkeit	Buchstabe	relative Häufigkeit
a	0.0651	l	0.0344	w	0.0189
b	0.0189	m	0.0253	x	0.0003
c	0.0306	n	0.0978	y	0.0004
d	0.0508	o	0.0251	z	0.0113
e	0.1740	p	0.0079		
f	0.0166	q	0.0002		
g	0.0301	r	0.0700		
h	0.0476	s	0.0727		
i	0.0755	t	0.0615		
j	0.0027	u	0.0435		
k	0.0121	v	0.0067		

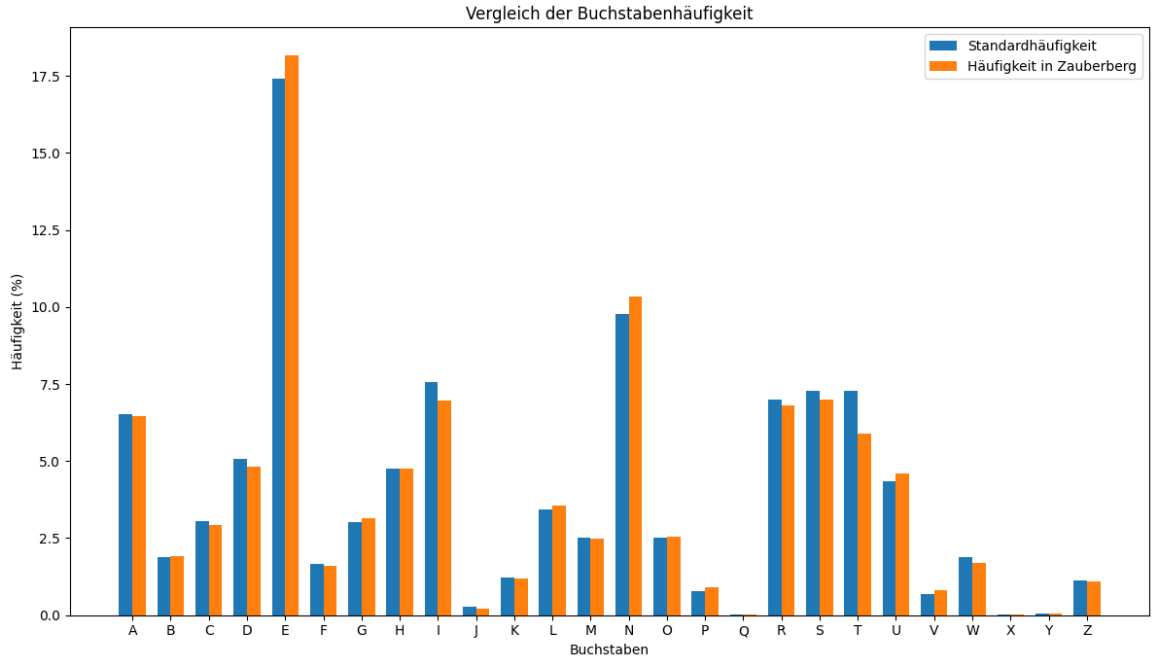
Um zu zeigen, dass dies sich mit (längeren) Texten deckt, wurde ein Kapitel aus dem Roman 'Der Zauberberg' von Thomas Mann ausgewertet. Die sich aus dieser Auswertung ergebende Verteilung wird in der folgenden Grafik der Verteilung aus der Tabelle gegenübergestellt.

```
import pdfplumber
import string
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
with pdfplumber.open("Mann_1989_Der Zauberberg - Roman.pdf") as pdf:
    pages = pdf.pages
    text = ""
    for i, pg in enumerate(pages):
        if i >= 23 and i < 42:
            text += pages[i].extract_text() + "\n"
def text_cleaning(text : str) -> str:
```

```
    clean = text.upper() \
                .replace('Ä', 'AE') \
                .replace('Ö', 'OE') \
                .replace('Ü', 'UE') \
                .replace('ß', 'SS') \
                .replace(' ', '') \

    cleaned_text = ''

    for c in clean:
        if c.isalpha():
            cleaned_text += c

    return cleaned_text

def file_writer(path : str, text : str) -> None:
    i = 0
    grouped_text = ""
    for c in text:
        i += 1
        if i % 50 == 0:
            grouped_text += c + "\n"
        elif i % 5 == 0:
            grouped_text += c + " "
        else:
            grouped_text += c

    with open(path, mode='w', encoding='utf-8') as f:
        f.write(grouped_text)
zauberberg = text_cleaning(text)
file_writer('zauberberg_bereinigt.txt', zauberberg)
def letter_frequency(text: str) -> dict:
    frequency = {}
    total_letters = 0

    for char in text:
        if char not in frequency:
            frequency[char] = 1
        else:
            frequency[char] += 1
        total_letters += 1

    for key, value in frequency.items():
        frequency[key] = (value / total_letters) * 100
```

```
    return frequency

frequency_zauberberg = letter_frequency(zauberberg)


standard_frequency = {
    'E': 17.4,
    'T': 7.27,
    'A': 6.51,
    'O': 2.51,
    'I': 7.55,
    'N': 9.78,
    'S': 7.27,
    'R': 7.0,
    'H': 4.76,
    'D': 5.08,
    'L': 3.44,
    'C': 3.06,
    'U': 4.35,
    'M': 2.53,
    'W': 1.89,
    'F': 1.66,
    'G': 3.01,
    'Y': 0.04,
    'P': 0.79,
    'B': 1.89,
    'V': 0.67,
    'K': 1.21,
    'J': 0.27,
    'X': 0.03,
    'Q': 0.02,
    'Z': 1.13
}


df = pd.DataFrame.from_dict([standard_frequency, frequency_zauberberg])
df.index = ['Standard Frequency', 'Zauberberg Frequency']
dft = df.T
dft = dft.sort_index()
dft['Standard Frequency'] = dft['Standard Frequency'].astype(float)
dft['Zauberberg Frequency'] = dft['Zauberberg Frequency'].astype(float)
# --- Erstellen Sie das Side-by-Side-Balkendiagramm ---
# Legen Sie die Breite der Balken fest
bar_width = 0.35
```

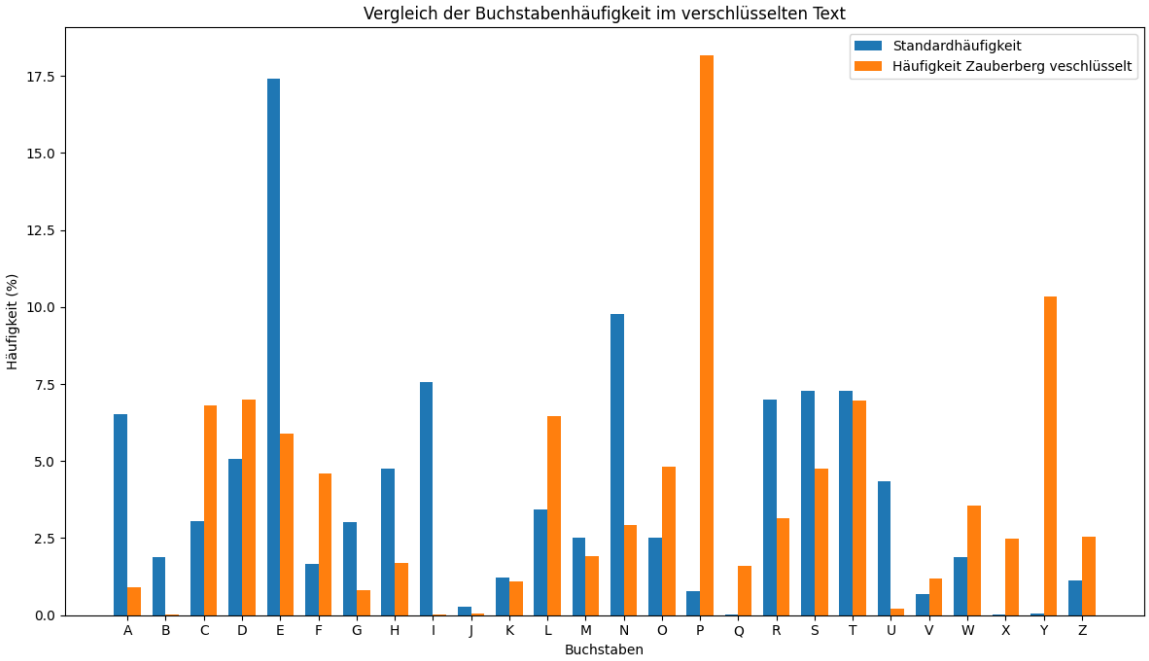
```
# Verwenden Sie den Index des transponierten DataFrames (dft)
x = np.arange(len(dft.index))

fig, ax = plt.subplots(figsize=(12, 7))

# Zeichnen Sie die Balken für die beiden Spalten aus dft
ax.bar(x - bar_width/2, dft['Standard Frequency'], bar_width, label='Standardhäufigkeit')
ax.bar(x + bar_width/2, dft['Zauberberg Frequency'], bar_width, label='Häufigkeit in Zaube

ax.set_xticks(x)
ax.set_xticklabels(dft.index)

ax.set_xlabel('Buchstaben')
ax.set_ylabel('Häufigkeit (%)')
ax.set_title('Vergleich der Buchstabenhäufigkeit')
ax.legend()
plt.tight_layout()

plt.show()
```



Die Grafik zeigt, dass bei einer Textlänge von 51'396 Buchstaben die Verteilung in einem literarischen Text nahezu identisch ist mit der allgemeinen Häufigkeitsverteilung in der deutschen Sprache.

Die nächste Grafik zeigt, was mit der Verteilung der Buchstaben geschieht, wenn der gleiche Text mit einer Caesar-Chiffre verschlüsselt worden ist.

```
def caesar(text : str, shift : int, encrypt=True) -> str:
    text = text.upper()
    result = ""

    if encrypt:
        for char in text:
            shifted = (ord(char) - ord('A') + shift) % 26 + ord('A')
            result += chr(shifted)
    else:
        for char in text:
            shifted = (ord(char) - ord('A') - shift) % 26 + ord('A')
            result += chr(shifted)

    return result


zauberberg_verschluesselt = caesar(zauberberg, 11, encrypt=True)
frequency_zauberberg_verschluesselt = letter_frequency(zauberberg_verschluesselt)
dft['Zauberberg Frequency Encrypted'] = pd.Series(frequency_zauberberg_verschluesselt)

# --- Erstellen Sie das Side-by-Side-Balkendiagramm ---
# Legen Sie die Breite der Balken fest
bar_width = 0.35

# Verwenden Sie den Index des transponierten DataFrames (dft)
x = np.arange(len(dft.index))

fig, ax = plt.subplots(figsize=(12, 7))

# Zeichnen Sie die Balken für die beiden Spalten aus dft
ax.bar(x - bar_width/2, dft['Standard Frequency'], bar_width, label='Standardhäufigkeit')
ax.bar(x + bar_width/2, dft['Zauberberg Frequency Encrypted'], bar_width, label='Häufigkei

ax.set_xticks(x)
ax.set_xticklabels(dft.index)

ax.set_xlabel('Buchstaben')
ax.set_ylabel('Häufigkeit (%)')
ax.set_title('Vergleich der Buchstabenhäufigkeit im verschlüsselten Text')
ax.legend()
plt.tight_layout()

plt.show()
```



Es ist deutlich zu erkennen, dass die Verteilung dem gleichen Muster folgt - verschoben um elf Positionen. Diese Auswertung ermöglicht die Entschlüsselung des Textes, ohne alle möglichen Schlüsselalphabete durchzuprobieren.

4.1 Vigenère Chiffre

Bei der Vigenère Chiffre handelt es sich um eine polyalphabetische Chiffre. Das Verfahren ist nach Blaise de Vigenère (1523 - 1596) benannt. Polyalphabetisch heisst, dass zur Verschlüsselung nicht eine Verschiebung vorgenommen wird, sondern - nach jedem Buchstaben wechselnd - mehrere Verschiebungen vorgenommen werden.

Um das zu erreichen, verwendet man ein sogenanntes Vigenère-Quadrat wie unten abgebildet.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Für die Verschlüsselung eines Klartextes braucht das Vigenère Verfahren ein Schlüsselwort. Das Schlüsselwort sollte möglichst lang sein. Das folgende Beispiel soll zeigen, wie das Vigenère Verfahren funktioniert. Der zu verschlüsselnde Klartext lautet ‘Kryptologie ist spannend’ und der Schlüssel ‘Buelrain’. Als Hilfestellung werden Text und Schlüssel in einer Tabelle dargestellt.

```
kryptologieistspannend
buelrainbuelrainbuelra
```

Der Schlüssel wird dabei ohne Wortabstand so oft wiederholt, bis die Buchstabenfolge des Schlüssels gleich lang ist, wie die Buchstabenfolge, welche zu verschlüsseln ist.
Als nächstes wird der zu verschlüsselnde Buchstabe in der Kopfzeile des Vigenère Quadrates

gesucht. Damit wird die Spalte mit dem verschobenen Alphabet identifiziert. Der chiffrierte Buchstabe ergibt sich, indem in der Spalte mit den Zeilenköpfen der unter dem zu chiffrierenden Buchstaben befindliche Buchstabe des Schlüssels gesucht wird. Der Schnittpunkt der Zeile mit der vorher gefundenen Spalte entspricht dem chiffrierten Buchstaben.

```
kryptologieistspannend
buelrainbuelrainbuelra

LLCAKOTBHCITJTACBHRPED
```

Alternativ kann eine Verschlüsselung mit der Vigenère Chiffre auch mit modularer Arithmetik umgesetzt werden. Dazu wird jedem Buchstaben ein Zahlenwert nach dem Muster $a = 0, b = 1, ..., z = 25$ zugewiesen. Die Verschlüsselung erfolgt anschliessend nach der 'Formel' $C_i = (P_i + K_i)\ mod\ 26$ Wobei die Buchstaben C für den chiffrierten Text, P für den Klartext (Englisch *plain text*) und K für den Schlüssel (Englisch *key*) stehen. Der Index $_i$ steht für den i-ten Buchstaben in der Textfolge.

Das obige Beispiel stellt sich dann folgendermassen dar:

```
 k  r  y  p  t  o  l  o  g  i  e  i  s  t  s  p  a  n  n  e  n  d
10 17 24 15 19 14 11 14 06 08 04 08 18 19 18 15 00 13 13 04 13 03
 b  u  e  l  r  a  i  n  b  u  e  l  r  a  i  n  b  u  e  l  r  a
01 20 04 11 17 00 08 13 01 20 04 11 17 00 08 13 01 20 04 11 17 00

11 37 28 26 36 14 19 27 07 28 08 19 35 19 26 28 01 33 17 15 30 03

11 11 02 00 10 14 19 01 07 02 08 19 09 19 00 02 01 07 17 15 04 03
 L  L  C  A  K  O  T  B  H  C  I  T  J  T  A  C  B  H  R  P  E  D
```

Für die Entschlüsselung wird die 'Formel' folgendermassen umgekehrt: $P_i = (C_i - K_i + 26)\ mod\ 26$. Die Addition von 26 in der Klammer erfolgt, um negative Zahlen zu vermeiden.

Wie sich die Vigenère Verschlüsselung auf die Verteilung der Buchstaben auswirkt, kann untenstehender Grafik entnommen werden.

```python
def vigenere_chiffre(text: str, key: str, encrypt=True) -> str:
    """
    Implementiert die Vigenère-Verschlüsselung für einen gegebenen Klartext und
    Schlüssel.

    Args:
        klartext (str): Der zu verschlüsselnde Text schluessel (str): Das
        Schlüsselwort für die Verschlüsselung
```

```
    Returns:
        str: Der verschlüsselte Text
    """

    # initialisiere den resultierenden Text
    resulting_text = ''

    # bestimme die Schlüssellänge für die anschliessende Modulo-Operation
    key_length = len(key)

    # itererie über den Eingabetext unter gleichzeitiger Erfassung des Index
    for i, char in enumerate(text):
        # berechne den Zahlwert des Buchstabens aus der ascii Tabelle
        char_no = ord(char) - 97
        key_no = ord(key[i % key_length]) - 97

        if encrypt == True:
            # berechne den Zahlwert des verschlüsselten Buchstabens
            ciph_no = (char_no + key_no) % 26
        else:
            # berechne den Zahlwert des entschlüsselten Buchstabens
            ciph_no = (char_no + (26 - key_no)) % 26

         # übernehme das Zeichen aufgrund seines Zahlwertes aus der ascii Tabelle
        ciph = chr(ciph_no + 97)

        # füge den Buchstaben am resultierenden Text an
        resulting_text += ciph
    return resulting_text

zauberberg_vigenere = vigenere_chiffre(zauberberg.lower(), 'buelrain', encrypt=True)

zauberberg_vigenere_frequency = letter_frequency(zauberberg_vigenere.upper())
dft['Zauberberg Frequency Vigenere'] = pd.Series(zauberberg_vigenere_frequency)

# --- Erstellen Sie das Side-by-Side-Balkendiagramm ---
# Legen Sie die Breite der Balken fest
bar_width = 0.35

# Verwenden Sie den Index des transponierten DataFrames (dft)
x = np.arange(len(dft.index))

fig, ax = plt.subplots(figsize=(12, 7))

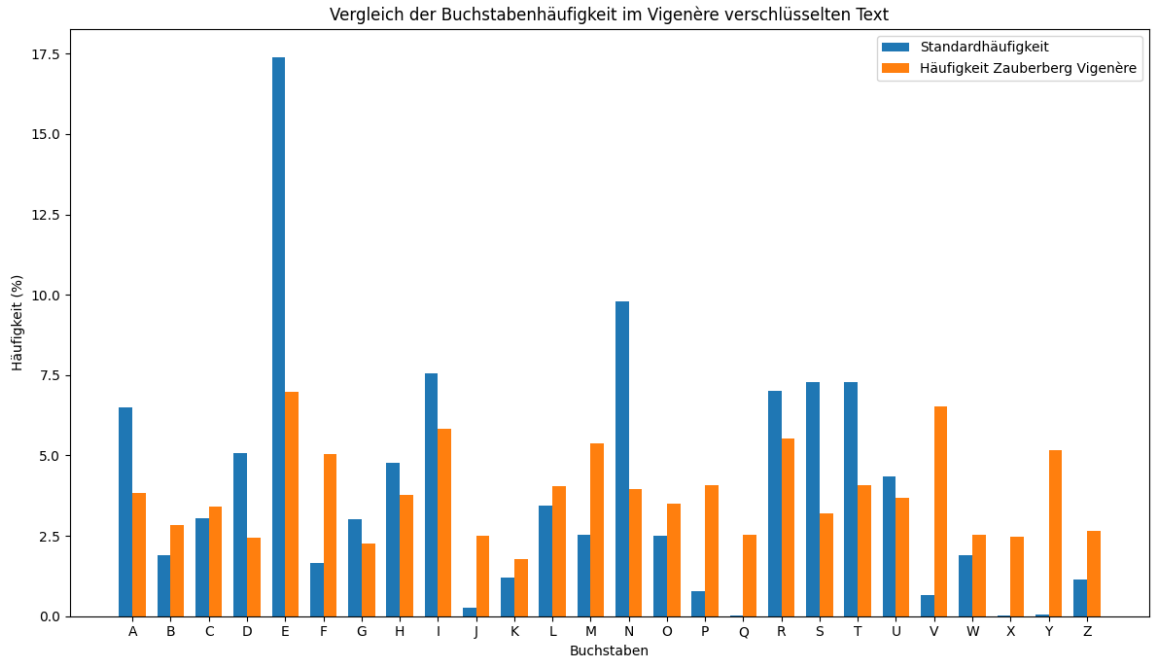
# Zeichnen Sie die Balken für die beiden Spalten aus dft
```

```
ax.bar(x - bar_width/2, dft['Standard Frequency'], bar_width, label='Standardhäufigkeit')
ax.bar(x + bar_width/2, dft['Zauberberg Frequency Vigenere'], bar_width, label='Häufigkeit

ax.set_xticks(x)
ax.set_xticklabels(dft.index)

ax.set_xlabel('Buchstaben')
ax.set_ylabel('Häufigkeit (%)')
ax.set_title('Vergleich der Buchstabenhäufigkeit im Vigenère verschlüsselten Text')
ax.legend()
plt.tight_layout()

plt.show()
```



Wie unschwer zu erkennen ist, stellt sich die Verteilung der Buchstaben in einem polyalphabetisch verschlüsselten Text deutlich anders dar, als dies in normalen Text der Fall ist. Die Vigenère Chiffre galt daher während ungefähr 300 Jahren als 'la chiffre indéchiffrable'.

Ein Spezialfall der Vigenère-Chiffre lässt sich tatsächlich nicht entschlüsseln. Das ist dann der Fall, wenn der Schlüssel länger ist als der Klartext. Man spricht in diesem Fall vom One-Time Pad.

5 Vigenère Chiffre Implementierung in Python

In diesem Notebook wird eine mögliche Implementierung der Vigenère Chiffre in Python vorgestellt.

```
def vigenere(text: str, key: str, mode: str) -> str:
    """Encrypts or decrypts a given text using the Vigenère cipher.

    The function processes a string of uppercase alphabetic characters
    using a provided key for encryption or decryption.

    Args:
        text: The string to be encrypted or decrypted. It should only contain
              uppercase alphabetic characters (A-Z).
        key: The key for the cipher. It should also only contain
             uppercase alphabetic characters (A-Z).
        mode: The operation to perform, must be 'encrypt' or 'decrypt'.

    Returns:
        The resulting ciphertext or plaintext.

    Raises:
        ValueError: If the mode is not 'encrypt' or 'decrypt'.
    """

    # Ensure the mode is valid before proceeding.
    if mode not in ['encrypt', 'decrypt']:
        raise ValueError("Mode must be 'encrypt' or 'decrypt'")

    key_length = len(key)

    if mode == 'encrypt':
       cipher = ''
       # Iterate through each character of the input text.
       for i, char in enumerate(text):
           # Convert the current text character and the corresponding key
           # character to a number (0-25).
           # The key character is determined using modulo to cycle through
```

```
            # the key.
            char_num = ord(char) - ord('A')
            key_num = ord(key[i % key_length]) - ord('A')

            # Calculate the new character's number using the Vigenère encryption formula.
            # The modulo operator ensures the result stays within the range 0-25.
            cipher_num = (char_num + key_num) % 26

            # Convert the resulting number back to an uppercase character and append it.
            cipher += chr(cipher_num + ord('A'))
        return cipher
    else:  # mode == 'decrypt'
        plain = ''
        # Iterate through each character of the input text.
        for i, char in enumerate(text):
            # Convert the current text character and the corresponding key
            # character to a number (0-25).
            char_num = ord(char) - ord('A')
            key_num = ord(key[i % key_length]) - ord('A')

            # Calculate the new character's number using the Vigenère
            # decryption formula.
            # The modulo operator handles negative results, ensuring the
            # result is correct.
            plain_num = (char_num - key_num) % 26

            # Convert the resulting number back to an uppercase
            # character and append it.
            plain += chr(plain_num + ord('A'))
        return plain
```

Die Funktion `vigenere()` wird im Folgenden detailliert erklärt.

Die Signatur der Funktion

```
def vigenere(text: str, key: str, mode: str) -> str:
```

zeigt, dass die Funktion drei Parameter erwartet: * `text`: Der zu verschlüsselnde oder zu entschlüsselnde Text als String. * `key`: Der Schlüssel zur Verschlüsselung oder Entschlüsselung als String. * `mode`: Der Modus, der angibt, ob der Text verschlüsselt oder entschlüsselt werden soll. Mögliche Werte sind `'encrypt'` für Verschlüsselung und `'decrypt'` für Entschlüsselung (dies ist allerdings nicht direkt aus der Signatur ersichtlich).

Anschliessend an die Signatur folgt ein ausführlicher Docstring, der die Funktion und ihre Parameter beschreibt. Dieser ist - obwohl in Englisch abgefasst - aus sich selber heraus verständlich.

Der Docstring wird gefolgt durch die Prüfung, ob zulässige Werte für den `mode`-Parameter übergeben wurden. Ist dies nicht der Fall, wird eine `ValueError`-Exception ausgelöst.

```
if mode not in ['encrypt', 'decrypt']:
    raise ValueError("Mode must be 'encrypt' or 'decrypt'")
```

Um diese Prüfung durchzuführen, werden die zulässigen Werte in einer Liste zur Verfügung gestellt. Geprüft wird dann, ob der `mode` zugewiesene Wert in der Liste enthalten ist. Falls dies nicht der Fall ist, wird eine Fehlermeldung ausgegeben und die Ausführung der Funktion abgebrochen.

Sofern der `mode`-Parameter einen zulässigen Wert enthält, beginnt die eigentliche Funktionslogik zu spielen.

Als erstes wird die Länge des Schlüssels in

```
key_length = len(key)
```

der Variablen `key_length` zugewiesen. Diese Information wird weiter unten benötigt, um in einer besonderen Art von Schlaufe über den Text des Schlüssels zu iterieren.

Nach dieser Zuweisung teilt sich die Funktion in die Zweige Ver- bzw. Entschlüsselung.

Zuerst wird die Verschlüsselung - eingeleitet mit `if mode == 'encrypt':` - betrachtet.

Innerhalb dieses Blocks wird als erstes ein leerer String `cipher` initialisiert, der später die verschlüsselten Zeichen aufnehmen wird.

Anschliessend wird mit einer `for`-Schlaufe über den zu verschlüsselnden Text iteriert.

```
for i, char in enumerate(text):
```

In dieser Schlaufe wird die Funktion `enumerate()` verwendet. Diese Funktion liefert ein Tupel aus Index und dem jeweiligen Element der Struktur, über die iteriert wird. Die Werte des Tupels werden den Variablen `i` und `char` zugewiesen.

Die Variabeln `i` und `char` werden innerhalb der Schlaufe verwendet, um die einzelnen Zeichen des Textes in eine Zahl umzuwandeln.

```
    char_num = ord(char) - ord('A')
    key_num = ord(key[i % key_length]) - ord('A')
```

Die Funktion `ord()` wandelt einen Buchstaben in die entsprechende Zahl aus der ASCII-Tabelle um. Damit die Zahlen im Bereich von 0 bis 25 liegen, wird der ASCII-Wert des Buchstabens ‘A’ abgezogen. Der Buchstabe ‘A’ wird verwendet, da die zu verschlüsselnden Zeichen in Grossbuchstaben vorgegeben sind.
Da der Schlüssel kürzer als der zu verschlüsselnde Text sein kann, wird mit `i % key_length`

über den Schlüssel iteriert. Der Modulo-Operator `%` sorgt dafür, dass der Index Wert des verwendeten Index immer zwischen 0 und der Länge des Schlüssels `key_length` bleibt. Dadurch wird sichergestellt, dass der Schlüssel wieder von vorne begonnen wird, sobald das Ende des Schlüssels erreicht ist. Anschliessend wird mit den einzelnen Buchstaben des Schlüssels genauso verfahren wie mit den Buchstaben des Textes.

Nachdem die Buchstaben des Textes sowie des Schlüssels in Zahlen umgewandelt wurden, wird die eigentliche Verschlüsselung gemäss der Formel $C_i = (P_i + K_i)\ mod\ 26$ durchgeführt.

```
    cipher_num = (char_num + key_num) % 26
```

Die Zahlenwerte des verschlüsselten Textes werden anschliessend wieder in Buchstaben umgewandelt und an den String `cipher` angehängt. Dazu wird die Funktion `chr()` verwendet, die eine Zahl in den entsprechenden Buchstaben der ASCII-Tabelle umwandelt. Da die Zahlenwerte im Bereich von 0 bis 25 liegen, wird der ASCII-Wert des Buchstabens 'A' addiert.

```
    cipher += chr(cipher_num + ord('A'))
```

Der unter `cipher` abgelegte String wird am Ende des Blocks zurückgegeben.

Im zweiten Block wird die Entschlüsselung - eingeleitet mit `else:` - durchgeführt. Der Prozess ist dem der Verschlüsselung sehr ähnlich, jedoch wird hier die umgekehrte Formel $(P_i = (C_i - K_i + 26)\ mod\ 26)$ verwendet.

```
    plain_num = (char_num - key_num + 26) % 26
    plain += chr(plain_num + ord('A'))
```

Alles andere entspricht dem Vorgehen bei der Verschlüsselung.

6 Die Vigenère-Verschlüsselung brechen

Die entscheidende Schwäche der Vigenère-Verschlüsselung wurde Mitte des 19. Jahrhunderts von zwei Männern unabhängig voneinander entdeckt. Einer von ihnen war Charles Babbage, ein britischer Universalgelehrter. Er entdeckte die Schwachstelle, um eine Wette zu gewinnen. Der andere war Friedrich Wilhelm Kasiski, ein pensionierter Major der preußischen Armee, der nach seiner Pensionierung seine Zeit mit Schreiben verbrachte. Die Methode zum Brechen der Vigenère-Verschlüsselung ist nach dem preußischen Major Kasiski benannt.

6.1 Die Kasiski-Methode – Zusammenfassung

Die **Kasiski-Methode** ist eine Technik zum Brechen der Vigenère-Verschlüsselung, bei der wiederholte Muster im Geheimtext ausgenutzt werden. So funktioniert sie:

6.1.1 Grundprinzip

Wenn dieselbe Klartextsequenz mit demselben Teil des Schlüssels verschlüsselt wird, entstehen identische Geheimtextsequenzen. Indem wir diese Wiederholungen finden, können wir die Schlüssellänge bestimmen.

6.1.2 Schritt-für-Schritt-Anleitung

1. **Wiederholte Sequenzen finden**
    - Den Geheimtext nach identischen Sequenzen von 3+ Zeichen durchsuchen.
    - Die Positionen aufzeichnen, an denen diese Sequenzen auftreten.
2. **Abstände berechnen**
    - Den Abstand zwischen wiederholten Sequenzen messen.
    - Die Schlüssellänge muss ein Teiler dieser Abstände sein.
3. **Schlüssellänge bestimmen**
    - Den größten gemeinsamen Teiler (ggT) aller Abstände finden.
    - Oder nach dem häufigsten gemeinsamen Teiler suchen.
    - Dies ergibt die wahrscheinliche Schlüssellänge.
4. **Häufigkeitsanalyse**

- Den Geheimtext basierend auf der Schlüssellänge in Gruppen aufteilen.
- Jede Gruppe wurde mit demselben Schlüsselzeichen verschlüsselt.
- Eine Häufigkeitsanalyse auf jede Gruppe separat anwenden.
- Die Buchstabenhäufigkeiten mit den erwarteten Mustern der Sprache abgleichen.

6.1.3 Beispiel

Wenn „THE“ mehrmals im Klartext an Positionen erscheint, an denen sich der Schlüssel wiederholt, sind die entsprechenden Geheimtextsequenzen identisch. Wenn diese Sequenzen 10 Positionen voneinander entfernt sind, ist die Schlüssellänge wahrscheinlich ein Teiler von 10 (1, 2, 5 oder 10).

Für ein konstruiertes Beispiel, wenn der Klartext „the codes in the word and the message“ lautet und der Schlüssel „crypt“ ist, erhalten wir den folgenden Geheimtext.

```
p: t h e c o d e s i n t h e w o r d a n d t h e m
k: c r y p t c r y p t c r y p t c r y p t c r y p
c: V Y C r h f v q x g V Y C l h t u y c w V Y C b
```

Beachten Sie, dass das Geheimtextmuster VYC dreimal im Geheimtext vorkommt und jedes Mal ein Abstand von 10 Buchstaben zwischen dem Anfang eines VYC und dem nächsten liegt. Dies geschieht, weil dasselbe Klartextmuster, „the“, jedes Mal auf denselben Teil des Schlüssels, „cry“, trifft, was zum selben Geheimtext führt. Die Wiederholung des Schlüsselworts ist hier die Schwachstelle. Die Duplikate im Geheimtext sind alle 10 Buchstaben voneinander entfernt. Babbage und Kasiski schlussfolgerten, dass dies bedeutet, dass die Länge des Schlüssels ein Teiler von 10 ist. Die Teiler von 10 sind 10, 5 und 2. Man könnte argumentieren, dass ein Schlüssel der Länge 2 zu kurz ist, um viel Sicherheit zu bieten, sodass ein Schlüssel der Länge 5 oder 10 wahrscheinlicher ist. Ein Schlüssel der Länge 10 hätte in einem so kurzen Geheimtext wahrscheinlich nicht so viele Wiederholungen, daher wird der Kryptoanalytiker seine Arbeit mit einer angenommenen Schlüssellänge von 5 beginnen.

Eine Schlüssellänge von 5 bedeutet, dass der 1., 6., 11., 16. usw. Buchstabe alle mit demselben Schlüsselbuchstaben und somit mit demselben Alphabet aus der Vigenère-Tabelle verschlüsselt werden. In ähnlicher Weise werden der 2., 7., 12., 17. usw. Buchstabe alle mit dem nächsten Schlüsselalphabet verschlüsselt. Wenn wir also das Kryptogramm in 5 Buchstabengruppen aufteilen, haben wir 5 monoalphabetische Substitutions-Geheimtexte. Wir können dann eine Häufigkeitsanalyse jeder Gruppe durchführen und jede Gruppe separat lösen. Und in einem standardmäßig verschobenen Alphabet wie in der normalen Vigenère-Tabelle haben wir, wenn wir einen einzigen Geheimtext-Alphabetbuchstaben finden können, das gesamte Alphabet.[1]

[1] Dooley, John F. History of Cryptography and Cryptanalysis: Codes, Ciphers, and Their Algorithms. History of Computing. Cham: Springer International Publishing, 2024. https://doi.org/10.1007/978-3-031-67485-3. S. 76.

6.1.4 Warum es funktioniert

Die Schwäche der Vigenère-Verschlüsselung liegt in der Wiederholung des Schlüssels. Sobald die Schlüssellänge bekannt ist, wird die Verschlüsselung zu mehreren einfachen Caesar-Verschlüsselungen, die mittels Häufigkeitsanalyse gebrochen werden können.

7 Public-Key-Kryptographie

Obwohl das One-Time-Pad theoretisch eine absolut sichere Verschlüsselung ermöglicht, ist es in der Praxis kaum praktikabel. Neben der Tatsache, dass der Schlüssel mindestens so lang sein muss, wie die Nachricht selbst, braucht es ein Verfahren den bzw. die Schlüssel sicher zwischen Sender und Empfänger zu teilen.

Um das Problem des Schlüsselaustausches zu lösen, verwendet man spezielle mathematische Funktionen. Diese Funktionen nennt man "Einwegfunktionen mit Hintertüren". Eine Einwegfunktion $f(x) = x'$ lässt sich leicht berechnen, aber ihre Umkehrung $f^{-1}(x') = x$ ist sehr schwierig zu berechnen. Die "Hintertür" ist ein Geheimwissen, mit dem die Umkehrfunktion dann doch einfach berechnet werden kann.

Die Einwegfunktion kann veröffentlicht werden, damit ein Absender mit deren Hilfe eine Botschaft verschlüsseln kann. Nur der Empfänger ist dann noch in der Lage, die Botschaft mit wenig Aufwand zu entschlüsseln. Allfällige 'Lauscher' können die Umkehrfunktion nicht innert nützlicher Frist berechnen.

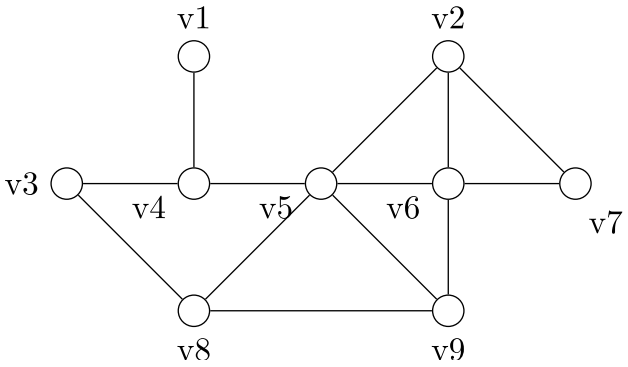
Um ein Beispiel einer solchen Einwegfunktion zu zeigen, wird der Umstand genutzt, dass Text als Zahlenfolge dargestellt werden kann. Ein als Zahlenfolge dargestellter Text kann dann mit Hilfe einer mathematischen Funktion verschlüsselt werden. Im folgenden soll ein Modell für ein solches Verschlüsselungsverfahren vorgestellt werden. In diesem Modell werden Graphen für die Modellierung einer Einwegfunktion verwendet.

7.1 Verschlüsselung mit Hilfe eines Graphen

Ein Graph besteht aus Knoten und Kanten. Die Knoten sind durch Kanten miteinander verbunden. Damit die Verschlüsselung mit Hilfe eines Graphen erfolgen kann, muss der Graph öffentlich bekannt und die Knoten nummeriert sein. Die Verschlüsselung erfolgt in den unten dargestellten Schritten.

1. Der Klartext wird als Folge von Zahlen dargestellt, welche folgendermassen verschlüsselt werden:
2. Die einzelnen Zahlen der Zahlenfolge werden in Summanden zerlegt. Die Zahl der Summanden entspricht der Anzahl der Knoten im Graphen.
3. Jeder Summand wird einem Knoten zugeordnet.
4. Der 'Wert' eines Knotens berechnet sich als Summe des dem Knoten zugeordneten Summanden und allen Summanden der Nachbarknoten.
5. Der verschlüsselte Wert der Zahl ist die Folge der Werte der Knoten.

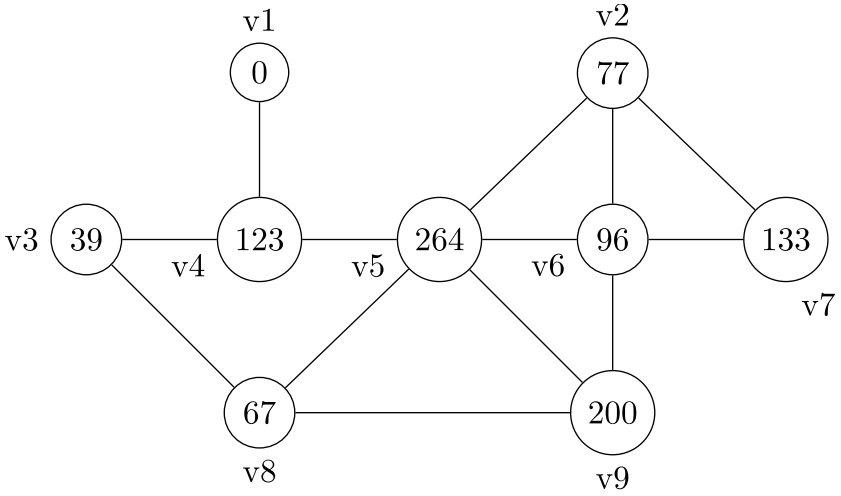
Das Vorgehen soll an einem Beispiel verdeutlicht werden. Dem Beispiel liegt der folgende Graph zu Grunde.



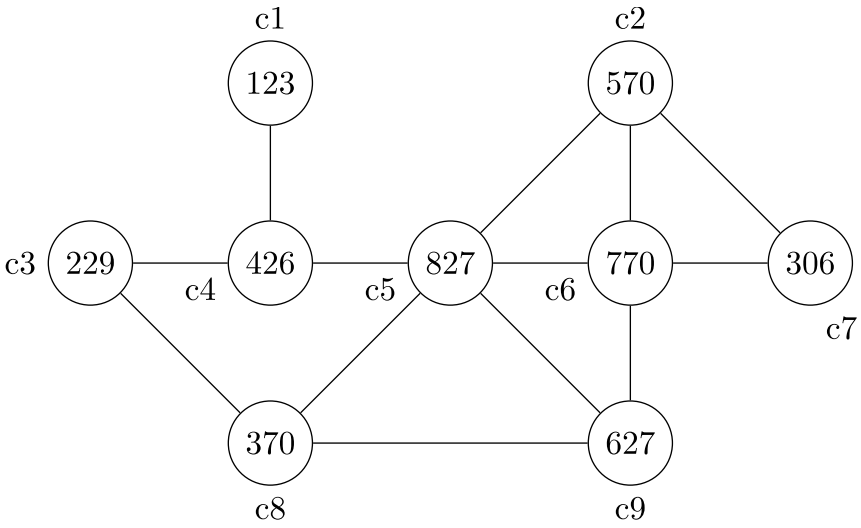
In diesem Graphen wird die Zahl 999 verschlüsselt. Die Aufteilung in Summanden sieht folgendermassen aus:

$$999 = 0 + 77 + 39 + 123 + 264 + 96 + 133 + 67 + 200$$

Die Summanden werden folgendermassen in den Graphen eingetragen:



Nach der Addition der Nachbarn stellt sich der Graph folgendermassen dar:



Der verschlüsselte Wert kann jetzt als Zahlenfolge 123/570/229/426/827/770/306/370/627 übermittelt werden.

Um die ursprüngliche Zahl zu rekonstruieren, muss ein unberechtigter Lauscher das folgende Gleichungssystem lösen:

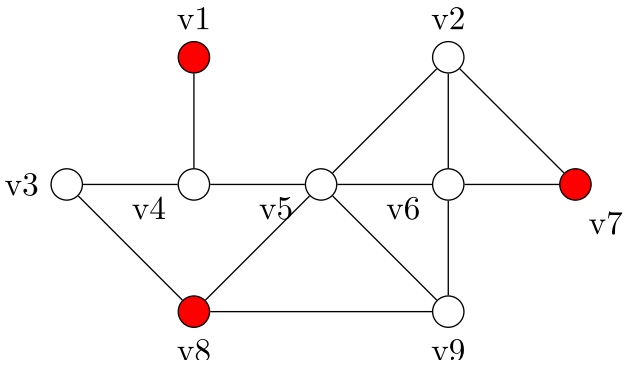
$$
\begin{aligned}
c_1 &= v_1 + v_4 \\
c_2 &= v_2 + v_5 + v_6 + v_7 \\
c_3 &= v_3 + v_4 + v_8 \\
c_4 &= v_1 + v_3 + v_4 + v_5 \\
c_5 &= v_2 + v_4 + v_5 + v_6 + v_8 + v_9 \\
c_6 &= v_2 + v_5 + v_6 + v_7 + v_9 \\
c_7 &= v_2 + v_6 + v_7 \\
c_8 &= v_3 + v_5 + v_8 \\
c_9 &= v_5 + v_6 + v_8 + v_9
\end{aligned}
$$

Wobei c_n die jeweilige übermittelte Zahl und v_n der Summand im jeweiligen Knoten darstellt. Dieses Gleichungssystem ist noch innerhalb nützlicher Frist lösbar. Wenn der Graph aber grösser wird, stösst man bald an zeitliche Grenzen.

Viel einfacher ist die Lösung, wenn man den Graphen in seine dominierende Menge zerlegt. Eine dominierende Menge ist eine Auswahl von Knoten, sodass jeder andere Knoten entweder in dieser Menge liegt oder einen Nachbarn darin hat. Dies so, dass innerhalb des Subgraphen die untereinander verbundenen Knoten mit einem einzigen Knoten verbunden sind.

7.2 Definition Dominierende Menge

“‘ovemvbmplz Definition Dominierende Menge Eine dominierende Menge in einem Graphen ist eine Teilmenge von Knoten mit einer besonderen Eigenschaft: Jeder Knoten des Graphen, der nicht zu dieser Teilmenge gehört, ist mit mindestens einem Knoten aus dieser Teilmenge verbunden. Anders ausgedrückt: Von den Knoten der dominierenden Menge aus kann man jeden anderen Knoten des Graphen mit genau einem Schritt erreichen.

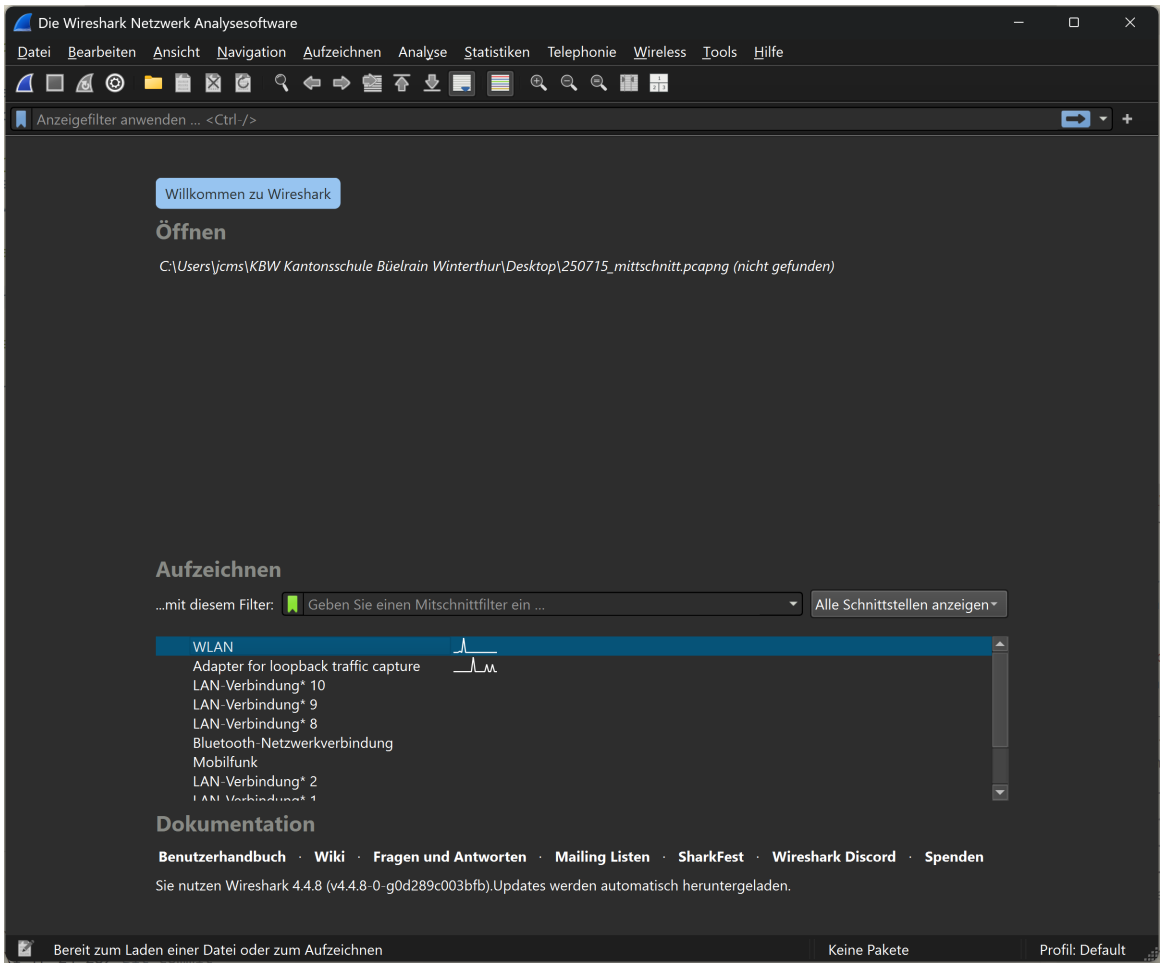
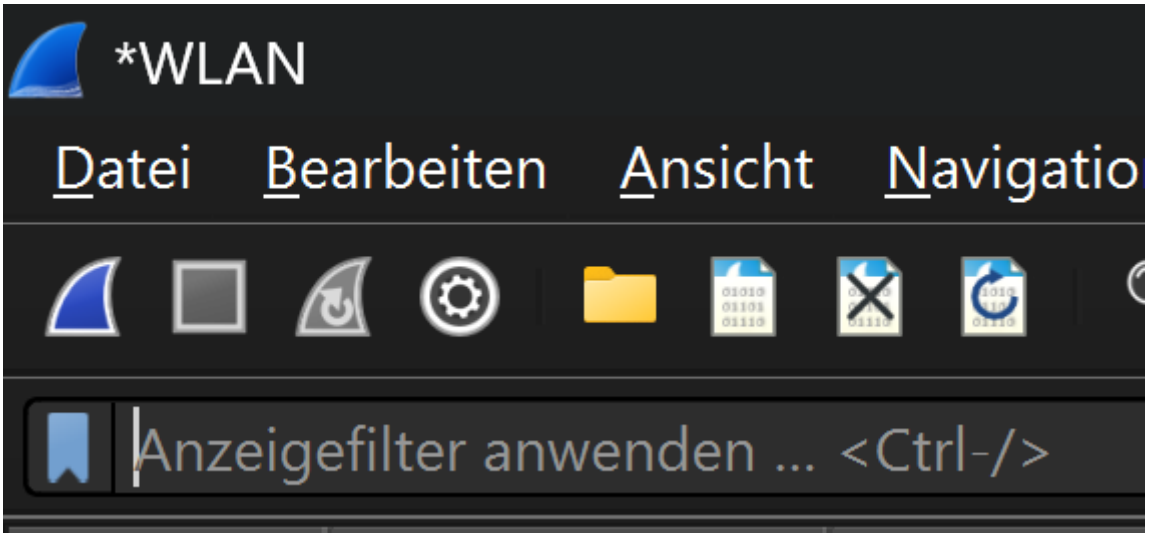
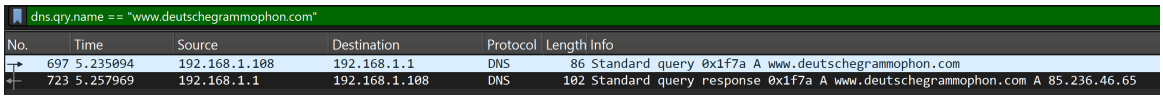
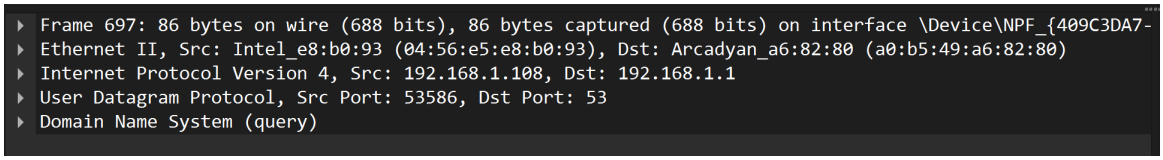
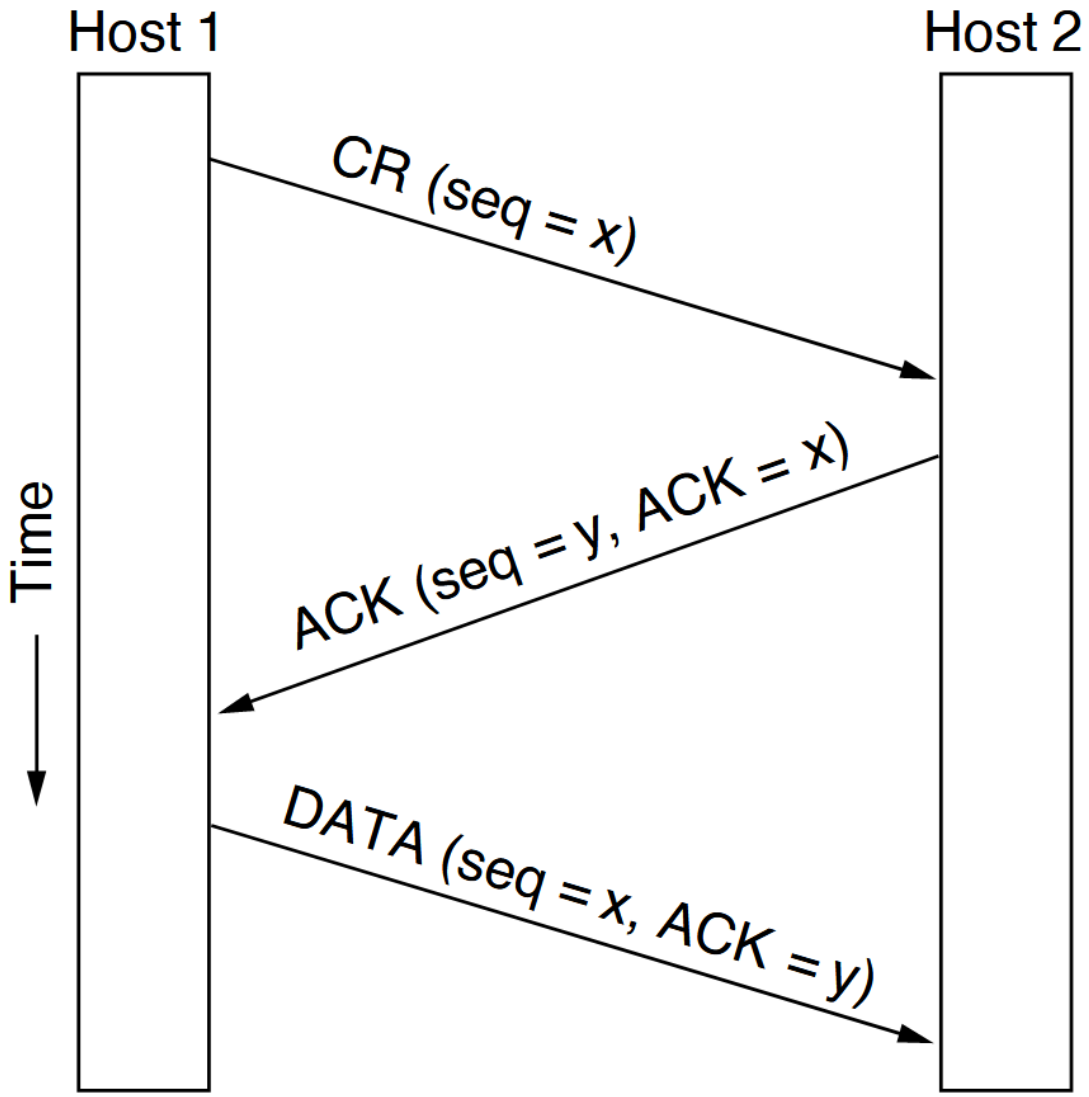
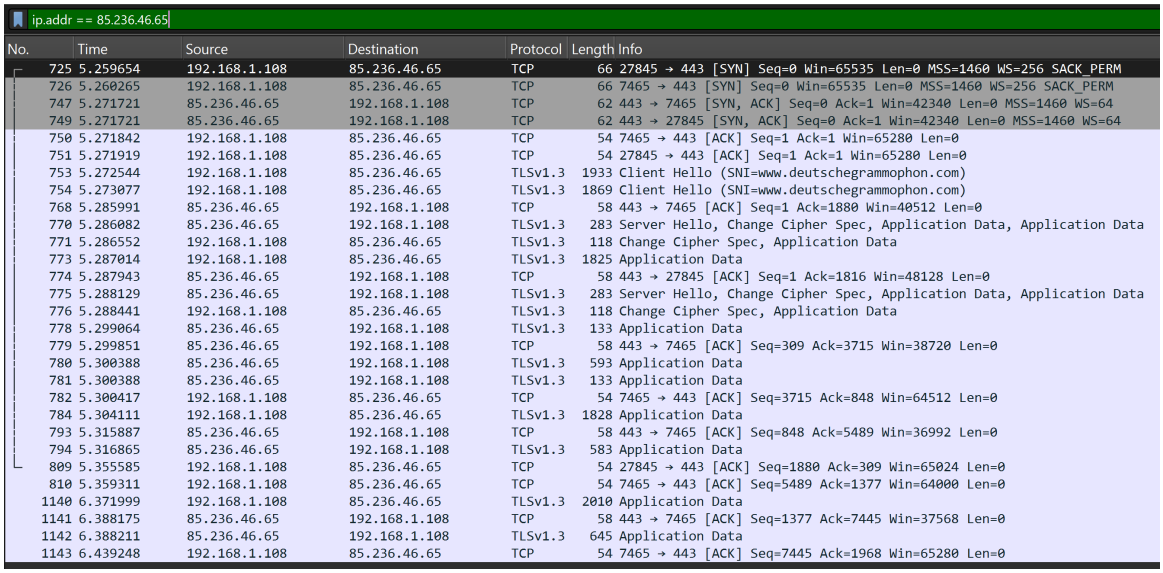
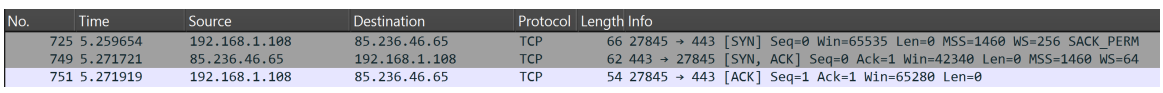



```
Die Kenntnis der dominierenden Menge stellt in unserem Verschlüsselungsverfahren
die "Hintertür" dar. Wer diese Menge kennt, kann das komplizierte
Gleichungssystem umgehen und die ursprünglichen Werte deutlich einfacher
rekonstruieren.

In den Knoten $v_1$, $v_7$ und $v_8$ sind alle Summanden enthalten. Um die
ursprüngliche Zahl zu rekonstruieren, reicht es also, die Werte dieser drei
Knoten zu summieren. Das finden der dominierenden Menge ist aber ein schwierig
zu lösendes Problem und stellt daher für den Lauscher eine unüberwindbare
Schranke dar[^1].

[^1]: Das Beispiel stammt aus dem Buch Freiermuth, Karin. Einführung in die
    Kryptologie: Lehrbuch für Unterricht und Selbststudium. 2., Überarb. Aufl.
    Einführung in die Kryptologie. Wiesbaden: Vieweg, 2014, Kapitel 9.


`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6IjI1MTExNF9yc2EifQ== -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6IjI1MTExNF9yc2EiLCJib29rSXRlbVR5cGUiOiJjaGF
```
```

8 RSA

RSA ist ein Kryptografie-System für asymmetrische Verschlüsselung und Signatur. Es wurde Jahrzehnte lang verwendet und wird nun stetig von neueren Systemen abgelöst.

8.1 Lernziele

1. Sie entschlüsseln und verschlüsseln eine Zahl mit dem gegebenen öffentlichen, beziehungsweise mit dem privaten Schlüssel.
2. Sie wissen, auf welcher mathematischen Schwierigkeit RSA beruht. Anders ausgedrückt: Was muss man machen, um vom öffentlichen auf den privaten Schlüssel zu kommen.
3. Sie erklären das Prinzip der Signatur. Sie berechnen eine RSA Signatur. Sie überprüfen, ob eine Nachricht zur RSA Signatur passt.

8.2 Die Mathematik hinter RSA

Wie RSA funktioniert, betrachten wir mit dieser interaktiven Webseite www.cryptool.org/de/cto/rsa-step-by-step/.

8.2.1 Tipps

Der Taschenrechner kann nicht gut modulo mit grossen Zahlen rechnen.

Python ist da besser geeignet. Wenn Sie $c = 88^{17} \ mod \ 143$ rechnen möchten, können Sie es in Python wie folgt berechnen:

```
c = 88**17 % 143
print(c)
# 121
```

:::{dropdown} Zusätzlicher Hintergrund zum Rechnen

Python kann `88000**80021` nicht ausrechnen, aber `88000**80021 % 89911` schon.

$88000 \cdot 80021$ ist zu gross für den Speicher. Python berechnet vor zu den Rest und braucht damit nicht nur weniger Speicher, sondern rechnet auch viel schneller.

:::

8.2.2 Signieren

Betrachten wir die Situation, dass Alice mitteilen möchte, wann Sie am Bahnhof ankommt. Sie kann es nicht direkt an Bob schreiben, sondern Mallory. Sie möchte nicht, dass Bob zu einer falschen Uhrzeit am Bahnhof wartet. Alice hat Bob deshalb im Vorfeld ihren öffentlichen Schlüssel geteilt.

- Privater Schlüssel: $(n = 143, d = 113)$
- Öffentlicher Schlüssel: $(n = 143, e = 17)$

Mallory darf den öffentlichen Schlüssel auch kennen. Der private Schlüssel kennt nur Alice.

Was Alice macht, ist sie Gibt die Stunde bekannt: Sie kommt um 15 Uhr an. Nun berechnet Alice zusätzlich

$$15^d \ mod \ n = 15^{113} \ mod \ 143 = 97$$

. Sie legt diese Zahl 97 als Signatur bei.

Wenn Mallory nichts ändert, bekommt Bob "Uhrzeit 15, Signatur 97". Bob kann nun den Inhalt überprüfen, indem er

$$signatur^e \ mod \ n = 97^{17} \ mod \ 143 = 15$$

berechnet. Er merkt, die Zahl stimmt überein.

Wenn Mallory eine falsche Zeit an Bob geben möchte und die Zeit zu 14 anpasst, merkt Alice, dass die Zeit nicht mit der Signatur über einpasst. Wenn Mallory die Signatur auch anpassen möchte, müsste sie die Signatur finden, sodass

$$x = 14^e \ mod \ n.$$

Dies ist genau so schwierig, wie das Entschlüsseln der Nachricht.

:::{tip} Aufgabe

1. Erstellen Sie auf der bekannten cryptool Seite ein Schlüsselpaar.
2. Wählen Sie (zufällig) 3 Zahlen.
3. Erstellen Sie zu jeder Zahl die Signatur.
4. Verändern Sie bei einem Tupel die Zahl, bei einem anderen die Signatur.
5. Tauschen Sie den öffentlichen Schlüssel mit den 3 Tupeln mit Ihrem Nachbar aus.
6. Testen Sie die Tupel, welches stimmt.

:::

Part III

Computernetzwerke

9 Computernetzwerke

Computernetzwerke sind Systeme, die es Computern und anderen Geräten ermöglichen, miteinander zu kommunizieren. Ursprünglich ging es vor allem um den Austausch von Nachrichten. Heute ermöglichen Netzwerke zusätzlich die gemeinsame Nutzung von Ressourcen wie Druckern, Dateien oder Internetzugängen. Sie bilden die Grundlage für viele alltägliche Anwendungen – von E-Mails über Videokonferenzen bis hin zu Cloud-Diensten.

Ein Computernetzwerk besteht aus mehreren miteinander verbundenen Geräten (z. B. Computer, Smartphones, Server), die über verschiedene Übertragungsmedien wie Kabel oder Funk kommunizieren. Wichtige Ziele eines Netzwerks sind: - effiziente Kommunikation - zuverlässige Verfügbarkeit von Informationen - Sicherheit der Datenübertragung (in der Praxis meist durch Verfahren auf höheren Schichten, z. B. Verschlüsselung über TLS)

Um die Kommunikation zu ermöglichen, werden in Netzwerken Regeln und Protokolle verwendet. Ein bekanntes Referenzmodell zur Beschreibung dieser Abläufe ist das OSI-Schichtenmodell. Es unterteilt die Netzwerkkommunikation in sieben aufeinander aufbauende Schichten – von der physischen Übertragung der Daten bis hin zur Anwendungsebene, auf der Programme wie Webbrowser oder E-Mail-Clients arbeiten.

9.1 Das OSI- bzw. TCP/IP-Modell

Das OSI-Modell (Open Systems Interconnection Model) ist ein Referenzmodell. Es hilft, die komplexen Abläufe der Datenübertragung zu strukturieren und zu verstehen. Die sieben Schichten (mit typischen Dateneinheiten, sogenannten PDUs) im Überblick:

1) Physical Layer

- Überträgt einzelne Bits als elektrische, optische oder Funksignale über das Übertragungsmedium (z. B. Kabel, Funk).
- PDU: Bits

2) Data Link Layer

- Sorgt für lokale, rahmenbasierte Übertragung zwischen direkt verbundenen Geräten; regelt den Zugriff auf das Medium (MAC).
- Bietet Fehlererkennung (z. B. CRC) und – je nach Technik – ggf. einfache Fehlerbehandlung/Wiederholungen (z. B. bei WLAN).
- PDU: Frames/Rahmen

3) Network Layer

- Leitet Pakete über mehrere Netzwerke hinweg weiter (Routing); Adressierung auf Netzwerkebene (z. B. IP-Adressen).
- Beispielprotokoll: Internet Protocol (IP).
- PDU: Packets/Pakete

4) Transport Layer

- Stellt Ende-zu-Ende-Transport zwischen Anwendungen bereit. Unterschiedliche Protokolle bieten unterschiedliche Dienste:
  - TCP: zuverlässig, geordnet, verbindungsorientiert; inkl. Fluss- und Staukontrolle.
  - UDP: verbindungslos, ohne Garantie für Zuverlässigkeit oder Reihenfolge.
- PDU: Segmente (TCP) bzw. Datagramme (UDP)

5) Session Layer

- Baut Sitzungen zwischen Anwendungen auf, verwaltet und beendet sie (z. B. Sitzungsverwaltung, Dialogsteuerung).

6) Presentation Layer

- Übersetzt Daten zwischen Darstellungsformaten (z. B. Zeichencodierung, Serialisierung) und kann Daten komprimieren/verschlüsseln.

7) Application Layer

- Stellt die Schnittstelle zu Anwendungsprogrammen und Anwendungsprotokollen bereit (z. B. HTTP, SMTP, DNS).

Das OSI-Schichtenmodell wurde in seiner reinen Form nicht als Protokollstapel umgesetzt. In der Praxis hat sich das TCP/IP-Modell durchgesetzt (TCP = Transmission Control Protocol, IP = Internet Protocol).

Das TCP/IP-Modell besteht aus vier Schichten, die Funktionen des OSI-Modells zusammenfassen:

1) Netzwerkzugangsschicht (Link/Network Access)

- Entspricht OSI Physical + Data Link (Schichten 1 und 2).
- Beispiele: Ethernet, WLAN; MAC-Adressen; Switches arbeiten typischerweise hier.
- PDU: Frames/Rahmen

2) Internetschicht

- Entspricht OSI Network Layer (Schicht 3).
- Beispiele: IP, Routing; Router arbeiten hier; IP-Adressen.
- PDU: Packets/Pakete

3) Transportschicht

- Entspricht OSI Transport Layer (Schicht 4).
- Beispiele: TCP (zuverlässig, geordnet), UDP (verbindungslos); Ports identifizieren Dienste.
- PDU: Segmente (TCP) / Datagramme (UDP)

4) Anwendungsschicht

- Fasst OSI Session, Presentation und Application (Schichten 5–7) zusammen.
- Beispiele: HTTP/HTTPS, SMTP, DNS. Verschlüsselung wie TLS liegt typischerweise oberhalb des Transports und wird Anwendungen zugeordnet.

Das TCP/IP-Modell ist heute die Grundlage für die Kommunikation im Internet und in den meisten Computernetzwerken.

9.2 Analogie zur Kapselung in der objektorientierten Programmierung (OOP)

Das Schichtenmodell in der Netzwerktechnik lässt sich gut mit dem Prinzip der Kapselung in der objektorientierten Programmierung vergleichen. In beiden Fällen werden komplexe Aufgaben in klar abgegrenzte, eigenständige Einheiten (Schichten bzw. Klassen) unterteilt. Jede Schicht im Netzwerkmodell hat eine genau definierte Aufgabe und kommuniziert nur mit der direkt darüber- oder darunterliegenden Schicht – ähnlich wie eine Klasse in der OOP ihre inneren Details verbirgt und nur über definierte Schnittstellen mit anderen Klassen interagiert.

Durch diese Kapselung wird die Komplexität reduziert, Änderungen können leichter vorgenommen werden und die einzelnen Komponenten bleiben unabhängig voneinander wartbar. So wie in der OOP eine Klasse ihre Daten und Methoden kapselt, kapselt im Schichtenmodell jede Ebene die Details ihrer Funktion und stellt nur die notwendigen Dienste für die nächste Schicht bereit.

10 IP Adressen und DNS

10.1 IP Adressen

Damit Computer in einem Netzwerk erreichbar sind, benötigen sie eine eindeutige Adresse. Diese Adresse wird als IP-Adresse (Internet Protocol Address) bezeichnet. Aktuell - das heisst seit ungefähr zwei Jahrzehnten - wird nach und nach vom IPv4 System auf IPv6 umgestellt.
IPv4 sind 32-Bit-Adressen, die in vier Oktetten dargestellt werden (z.B. 192.0.2.1). Eine Länge von 32 Bit ergibt 2^{32}, das heisst etwas mehr als 4.29 Milliarden mögliche Adressen. Das zeigt, dass IPv4-Adressen begrenzt sind. Diese Begrenztheit hat zur Entwicklung von IPv6 geführt.
IPv6 Adressen sind 128 Bit lang. Sie werden in Hexadezimaldarstellung angezeigt, wobei acht Gruppen von vier hexadezimalen Ziffern (z.B. 2001:0db8:85a3:0000:0000:8a2e:0370:7334) verwendet werden. Die Länge von 128 Bit ermöglicht 2^{128} Adressen, was in dezimaler Schreibweise $3.4 \cdot 10^{38}$, einer Zahl mit 39 Stellen, entspricht. Auf jeden Quadratmeter der Erdoberfläche kämen etwa $6.7 \cdot 10^{23}$ IPv6-Adressen – das ist ungefähr eine Avogadro-Zahl pro Quadratmeter.

10.2 DNS Lookup

Menschen können sich IP-Adressen nur schwer merken. Daher wurden Domainnamen eingeführt, die leichter zu erinnern sind. Ein Beispiel für einen Domainnamen ist www.google.com. Eine mögliche IP-Adresse für den Domainname von Google ist 172.217.168.68; oft bestehen mehrere IP-Adressen für einen Domainnamen. Für die "Übersetzung" eines Domainname in eine IP-Adresse ist das Domain Name System (DNS) verantwortlich.

10.2.1 Funktionsweise

1. **Anfrage:** Wenn ein Benutzer eine Website besucht, sendet der Browser eine DNS-Anfrage an einen DNS-Server, um die IP-Adresse der gewünschten Domain zu ermitteln.
2. **Auflösung:** Der DNS-Server überprüft seinen Cache auf eine vorhandene Zuordnung. Wenn keine Zuordnung gefunden wird, leitet der Server die Anfrage an andere DNS-Server weiter, bis die IP-Adresse ermittelt wird.
3. **Antwort:** Der DNS-Server sendet die gefundene IP-Adresse zurück an den Browser, der dann eine Verbindung zur Website herstellen kann.

10.2.2 Typen von DNS-Einträgen

- **A-Record:** Verknüpft einen Domainnamen mit einer IPv4-Adresse.
- **AAAA-Record:** Verknüpft einen Domainnamen mit einer IPv6-Adresse.
- **CNAME-Record:** Alias für einen anderen Domainnamen.
- **MX-Record:** Gibt den Mailserver für eine Domain an.

10.2.3 Bedeutung

Der Domainname ist entscheidend für die Benutzerfreundlichkeit des Internets, da er es ermöglicht, Websites über leicht merkbare Namen anstelle von numerischen IP-Adressen zu erreichen. Ohne DNS müssten Benutzer die IP-Adressen aller Websites kennen, die sie besuchen möchten.

11 Network Address Translation (NAT)

11.1 Was ist NAT?

Network Address Translation (NAT) ist ein Verfahren zur automatischen und transparenten Übersetzung von IP-Adressen in Datenpaketen. NAT ermöglicht es, dass mehrere Geräte in einem privaten Netzwerk über eine einzige öffentliche IP-Adresse mit dem Internet kommunizieren können. Diese Technik wurde entwickelt, um die Knappheit von IPv4-Adressen zu bewältigen und gleichzeitig die Sicherheit privater Netzwerke zu erhöhen.

In der Praxis bedeutet dies: Wenn Sie zu Hause mehrere Geräte (Laptop, Smartphone, Smart-TV) mit dem Internet verbinden, teilen sich alle diese Geräte eine einzige öffentliche IP-Adresse, die Ihnen von Ihrem Internetanbieter zugewiesen wurde.

11.2 Funktionsweise von NAT

11.2.1 Grundprinzip

NAT arbeitet auf der Schicht 3 des OSI-Modells und ändert die Adressen im IP-Header von Datenpaketen. Der Prozess läuft folgendermassen ab:

1. **Ausgehende Verbindung:** Ein Gerät im privaten Netzwerk (z.B. 192.168.1.10) sendet eine Anfrage an einen Server im Internet.
2. **Adressübersetzung:** Der NAT-Router ersetzt die private Quell-IP-Adresse durch seine öffentliche IP-Adresse (z.B. 203.0.113.5).
3. **Tabelleneintrag:** Der Router speichert die Zuordnung in einer NAT-Tabelle.
4. **Antwort:** Wenn der Server antwortet, verwendet der Router die NAT-Tabelle, um die Antwort an das richtige Gerät im privaten Netzwerk weiterzuleiten.

11.2.2 NAT-Tabelle

Die NAT-Tabelle ist das Herzstück des Verfahrens. Sie enthält folgende Informationen:

- **Private IP-Adresse** (z.B. 192.168.1.10)
- **Privater Port** (z.B. 54321)
- **Öffentliche IP-Adresse** (z.B. 203.0.113.5)
- **Öffentlicher Port** (z.B. 12345)
- **Ziel-IP-Adresse** (z.B. 93.184.216.34)

- **Ziel-Port** (z.B. 80)

11.3 Private IP-Adressbereiche

Für private Netzwerke sind spezielle IP-Adressbereiche reserviert, die nicht im öffentlichen Internet geroutet werden:

- **10.0.0.0 bis 10.255.255.255** (10.0.0.0/8) – 16'777'216 Adressen
- **172.16.0.0 bis 172.31.255.255** (172.16.0.0/12) – 1'048'576 Adressen

- **192.168.0.0 bis 192.168.255.255** (192.168.0.0/16) – 65'536 Adressen

Diese Bereiche können in privaten Netzwerken beliebig oft wiederverwendet werden, da sie nur lokal gültig sind.

11.4 Arten von NAT

11.4.1 1. Static NAT (One-to-One NAT)

Bei Static NAT wird eine private IP-Adresse fest einer öffentlichen IP-Adresse zugeordnet. Dies wird verwendet, wenn ein Server im privaten Netzwerk dauerhaft aus dem Internet erreichbar sein soll.

Beispiel: - Privat: 192.168.1.100 Öffentlich: 203.0.113.10

11.4.2 2. Dynamic NAT

Dynamic NAT ordnet private IP-Adressen dynamisch aus einem Pool öffentlicher IP-Adressen zu. Die Zuordnung erfolgt bei Bedarf und wird nach einer gewissen Zeit wieder freigegeben.

11.4.3 3. Port Address Translation (PAT)

11.4.3.1 Was ist ein Port?

Um zu verstehen, wie PAT funktioniert, müssen wir zuerst das Konzept der Ports verstehen. Eine IP-Adresse identifiziert einen Computer im Netzwerk – aber auf einem Computer laufen gleichzeitig viele Programme, die alle Netzwerkverbindungen nutzen möchten.

Analogie: Stellen Sie sich die IP-Adresse als Postadresse eines Hochhauses vor. Die Portnummer entspricht dann der Wohnungsnummer. So wie die Post weiss, in welche Wohnung ein Brief gehört, weiss der Computer durch die Portnummer, an welches Programm die Daten geliefert werden sollen.

Ports sind 16-Bit-Zahlen (0 bis 65'535) und werden in drei Kategorien unterteilt: - **Well-known Ports (0-1023):** Reserviert für Standarddienste - Port 80: HTTP (Webseiten) - Port 443: HTTPS (verschlüsselte Webseiten) - Port 22: SSH (sichere Verbindung) - Port 25: SMTP (E-Mail-Versand) - **Registered Ports (1024-49'151):** Für registrierte Dienste - **Dynamic/Private Ports (49'152-65'535):** Frei verwendbar

11.4.3.2 Funktionsweise von PAT

PAT, auch NAT-Overload genannt, ist die häufigste Form von NAT. Hier teilen sich viele private IP-Adressen eine einzige öffentliche IP-Adresse, wobei die Unterscheidung über Portnummern erfolgt.

Der Router merkt sich nicht nur, welche private IP-Adresse eine Verbindung aufgebaut hat, sondern auch über welchen Port. Dadurch können mehrere Geräte gleichzeitig über dieselbe öffentliche IP-Adresse kommunizieren.

Rechenbeispiel: Mit 16-Bit-Portnummern stehen theoretisch $2^{16} = 65'536$ Ports zur Verfügung. Abzüglich der reservierten Ports (0-1023) bleiben etwa 64'512 nutzbare Ports für gleichzeitige Verbindungen. In der Praxis bedeutet dies, dass ein Heimrouter theoretisch über 64'000 gleichzeitige Verbindungen verwalten könnte.

11.5 Vorteile und Nachteile

11.5.1 Vorteile

- **IP-Adressen-Einsparung:** Ein Haushalt mit 20 Geräten benötigt nur eine öffentliche IP-Adresse
- **Sicherheit:** Private IP-Adressen sind von aussen nicht direkt erreichbar
- **Flexibilität:** Interne Netzwerkstruktur kann geändert werden, ohne die öffentliche Adresse zu beeinflussen

11.5.2 Nachteile

- **Ende-zu-Ende-Konnektivität:** Direktverbindungen zwischen Geräten werden erschwert
- **Komplexität:** Bestimmte Anwendungen (z.B. VoIP, Online-Gaming) benötigen spezielle Konfigurationen
- **Performance:** Die Adressübersetzung benötigt Rechenleistung und kann zu Verzögerungen führen

11.6 Praktisches Beispiel

Betrachten wir einen typischen Heimrouter:

1. **Ihr Laptop** (192.168.1.15) öffnet die Webseite www.example.com
2. **HTTP-Anfrage:**
   - Quelle: 192.168.1.15:45678
   - Ziel: 93.184.216.34:80
3. **NAT-Router übersetzt:**
   - Quelle: 85.5.123.45:23456 (öffentliche IP)
   - Ziel: 93.184.216.34:80
4. **Server antwortet:**
   - Quelle: 93.184.216.34:80
   - Ziel: 85.5.123.45:23456
5. **Router übersetzt zurück:**
   - Quelle: 93.184.216.34:80
   - Ziel: 192.168.1.15:45678

11.7 Bedeutung für die Zukunft

Mit der Einführung von IPv6 und seinen 2^{128} möglichen Adressen wird NAT theoretisch überflüssig. In der Praxis wird NAT jedoch weiterhin verwendet werden, da:

- Viele Netzwerke noch auf IPv4 basieren
- NAT zusätzliche Sicherheit bietet
- Die Umstellung auf IPv6 schrittweise erfolgt

NAT bleibt somit eine wichtige Technologie für das moderne Internet, die sowohl die Adressknappheit löst als auch zur Netzwerksicherheit beiträgt.

12 Top Level Domains und ihre Vergabe

12.1 Was sind Top Level Domains?

Top Level Domains (TLDs) sind der letzte Teil eines Domainnamens – der Teil nach dem letzten Punkt. Bei der Adresse www.example.com ist ".com" die Top Level Domain. TLDs bilden die oberste Hierarchieebene im Domain Name System (DNS) und sind essentiell für die Organisation des Internets.

Jeder Domainname folgt einer hierarchischen Struktur, die von rechts nach links gelesen wird: - **www.kbw.ch** - TLD: .ch (Schweiz) - Second Level Domain: kbw - Subdomain: www

12.2 Arten von Top Level Domains

12.2.1 1. Generic Top Level Domains (gTLDs)

Die generischen TLDs sind nicht länderspezifisch und können weltweit registriert werden:

- **.com** – Commercial (ursprünglich für Unternehmen)
- **.org** – Organization (für Organisationen)
- **.net** – Network (ursprünglich für Netzwerkanbieter)
- **.edu** – Education (nur für akkreditierte Bildungseinrichtungen)
- **.gov** – Government (nur für US-Regierungsstellen)
- **.mil** – Military (nur für US-Militär)

12.2.2 2. Country Code Top Level Domains (ccTLDs)

Jedes Land erhält einen zweistelligen Code nach ISO 3166-1:

- **.ch** – Schweiz (Confoederatio Helvetica)
- **.de** – Deutschland
- **.fr** – Frankreich
- **.uk** – Vereinigtes Königreich
- **.us** – Vereinigte Staaten

Insgesamt existieren 249 ccTLDs für Länder und Territorien.

12.2.3 3. New Generic Top Level Domains (ngTLDs)

Seit 2013 wurden über 1'200 neue TLDs eingeführt:

- **.shop**, **.blog**, **.app**
- **.berlin**, **.paris**, **.swiss**
- **.google**, **.apple** (Marken-TLDs)

12.3 Die Vergabestelle: ICANN und IANA

Die **Internet Corporation for Assigned Names and Numbers (ICANN)** ist die zentrale Organisation für die Verwaltung des Domain Name Systems. Sie wurde 1998 gegründet und hat ihren Hauptsitz in Los Angeles, USA.

Die **Internet Assigned Numbers Authority (IANA)**, eine Abteilung von ICANN, verwaltet die Root Zone des DNS. Diese enthält alle TLDs und ihre zugehörigen Nameserver.

12.3.1 Hierarchie der Vergabe

1. **ICANN/IANA** → Verwaltet alle TLDs
2. **Registry** → Betreibt eine spezifische TLD (z.B. SWITCH für .ch)
3. **Registrar** → Verkauft Domains an Endkunden
4. **Registrant** → Der Domaininhaber

12.4 Der Vergabeprozess für neue gTLDs

12.4.1 Phase 1: Bewerbung

Organisationen können sich bei ICANN um neue TLDs bewerben. Die letzte grosse Bewerbungsrunde war 2012:

- **Bewerbungsgebühr:** 185'000 US-Dollar
- **Bewerbungszeitraum:** 4 Monate
- **Eingegangene Bewerbungen:** 1'930

12.4.2 Phase 2: Evaluation

ICANN prüft jede Bewerbung auf: - Technische Kompetenz - Finanzielle Stabilität - Keine Markenrechtsverletzungen - Öffentliches Interesse

12.4.3 Phase 3: Delegation

Nach erfolgreicher Prüfung wird die TLD in die Root Zone eingetragen. Der gesamte Prozess dauert typischerweise 18-24 Monate.

12.5 Kosten und Gebühren

12.5.1 Für Registry-Betreiber

- **Jährliche ICANN-Gebühr:** 25'000 US-Dollar pro TLD
- **Transaktionsgebühr:** 0.25 US-Dollar pro Domainregistrierung
- **Betriebskosten:** Technische Infrastruktur, Personal, Marketing

12.5.2 Für Endkunden (Beispiele)

- **.ch Domain:** CHF 10-20 pro Jahr
- **.com Domain:** CHF 15-25 pro Jahr
- **Premium ngTLDs:** CHF 50-500+ pro Jahr

12.6 Besondere Regelungen

12.6.1 Restricted TLDs

Einige TLDs haben spezielle Registrierungsbedingungen:

- **.edu** – Nur für akkreditierte US-Hochschulen
- **.museum** – Nur für Museen
- **.aero** – Nur für die Luftfahrtindustrie
- **.swiss** – Verbindung zur Schweiz erforderlich

12.6.2 Internationalized Domain Names (IDN)

Seit 2010 sind TLDs in nicht-lateinischen Schriften möglich:

- **.** (China)
- **.** (Russland)
- **.** (Griechenland)

12.7 Statistische Betrachtung

Stand 2025 gibt es: - **Aktive TLDs:** ~1'500 - **Registrierte Domains weltweit:** ~370 Millionen - **Grösste TLD:** .com mit ~160 Millionen Domains - **Schweizer .ch Domains:** ~2.5 Millionen

Die Verteilung folgt einem Potenzgesetz: Die 10 grössten TLDs beherbergen etwa 75% aller Domains.

12.8 Bedeutung und Ausblick

Top Level Domains sind mehr als technische Bezeichnungen – sie sind digitale Identitäten. Eine .ch-Domain signalisiert Schweizer Herkunft, eine .edu-Domain steht für Bildung. Mit der kontinuierlichen Expansion des Internets wächst auch die Bedeutung einer durchdachten TLD-Struktur.

Die nächste ICANN-Bewerbungsrunde für neue gTLDs ist für 2026 geplant. Experten erwarten weitere 500-1'000 neue TLDs, darunter vermehrt Städte- und Markennamen. Die Herausforderung wird sein, die Balance zwischen Innovation und Übersichtlichkeit zu wahren.

13 Beobachten von Internetverbindungen

Die folgenden Ausführungen basieren auf der Analyse von Netzwerkpaketen, welche mit Wireshark aufgezeichnet wurden. Dafür ist die Installation von Wireshark erforderlich. Die Website von Wireshark stellt dazu den entsprechenden Download zur Verfügung.

13.1 Aufzeichnen von Netzwerkpaketen

Um Netzwerkpakete aufzuzeichnen, wird Wireshark gestartet. Das Startfenster von Wireshark stellt sich folgendermassen dar:



Figure 13.1: Wireshark Startfenster

Unter dem Titel “Aufzeichnen” kann der gewünschte Netzwerkadapter ausgewählt und die Aufzeichnung gestartet werden. Die erfassten Pakete werden in Echtzeit angezeigt und können später analysiert werden. In der Schule wird die Verbindung zum Internet per WLAN hergestellt. Entsprechend ist der WLAN-Adapter auszuwählen. Sobald der entsprechende Adapter ausgewählt ist, kann die Aufzeichnung gestartet werden. Gestartet wird die Aufzeichnung durch einen Klick auf das blaue Haifischflossen-Symbol in der Symbolleiste. Die Aufzeichnung startet umgehend. Angehalten wird die Aufzeichnung mit einem Klick auf das rote Quadrat-Symbol in der Symbolleiste. Die Aufzeichnung kann entweder über das Menü Datei > Speichern, durch einen Klick auf das Dateisymbol oder durch die Tastenkombination `Strg + S` gespeichert werden.

13.2 Beobachten der DNS-Anfragen

13.2.1 Filtern der Aufzeichnung

Um die DNS-Anfragen zu beobachten, wird bei laufender Wireshark Aufzeichnung eine beliebige Website aufgerufen. Dadurch werden die entsprechenden DNS-Anfragen erfasst und können in Wireshark analysiert werden. Nach dem Aufruf der Website kann die Aufzeichnung angehalten und die erfassten Pakete analysiert werden (wenn die Aufzeichnung weiterläuft, bewegen sich die Pakete im Anzeigefenster ständig weiter).

Damit die relevanten Datenpakete angezeigt werden, kann der aufgezeichnete Datenverkehr gefiltert werden. Der Filter wird in der Eingabefeld für "Anzeigefilter" eingegeben.



Figure 13.2: Wireshark Anzeigefilter

Der entsprechende Filter für DNS-Anfragen zu einer gegebenen Website lautet

```
dns.qry.name == "www.example.com"
```

Der angewendete Filterbefehl ist relativ einfach nachzuvollziehen. An erster Stelle steht hier das Protokoll, nach dem gefiltert wird. Weil nach den DNS-Anfragen gefiltert wird, ist das hier `dns`. `dns` alleine wäre bereits ein gültiger Filter. Allerdings werden dann alle DNS-Pakete angezeigt. Der Filter wird daher zu `dns.qry.name` ergänzt Dabei steht `qry` als Abkürzung für query - Anfrage. Die Ergänzung `name` steht für den Domain Name, der Abgefragt wird. `==` ist die logische Verknüpfung, nach der gefiltert wird und bedeutet hier "ist gleich". Zwischen den Anführungszeichen steht der String, nach dem gesucht wird.
Sofern die Seite während der Aufzeichnung genau einmal aufgerufen wurde, wird der Filter zwei Pakete anzeigen: eine DNS-Anfrage und eine DNS-Antwort.



Figure 13.3: Gefilterte Wireshark-Pakete

Das Bild zeigt als erstes Paket die DNS-Anfrage für www.deutschegrammaphon.com und als zweites Paket die entsprechende Antwort.

13.2.2 Analyse der gefilterten Pakete

Für eine genaue Analyse der Kommunikation kann ein einzelnes Paket durch anklicken ausgewählt werden. Dadurch wird das Paket im unteren Bereich von Wireshark detailliert angezeigt und kann genauer untersucht werden.



Figure 13.4: Inhalt des Ausgewählten Pakets

Dass es sich hier im Bild um die Details des ausgewählten Paketes handelt, ist an der übereinstimmenden Paketnummer zu erkennen. Die Zeilen in der Detailansicht entsprechen den einzelnen Protokoll-Header-Feldern des ausgewählten Paketes. Das bildet auch das TCP/IP Schichtenmodell ab.

Die Detailansicht kann durch einen Klick auf die Dreiecke am Anfang der einzelnen Protokoll-Header-Felder erweitert werden. Dadurch werden weitere Informationen zu den jeweiligen Feldern angezeigt. Hier werden jedoch nur die Zusammenfassungen der Header-Felder erläutert.

Im Beispiel wird als erstes der Inhalt des Headers des Internetlayers erläutert.

```
Internet Protocol Version 4, Src: 192.168.1.108, Dst: 192.168.1.1
```

In der Zusammenfassung werden die Quell- und Ziel-Adressen des IP-Pakets angezeigt. Im vorliegenden Fall sind das jeweils die Privaten IP-Adressen 192.168.1.108 und 192.168.1.1. 192.168.1.108 ist die Quell-Adresse, erkennbar an der Abkürzung “Src” und 192.168.1.1 die Ziel-Adresse, erkennbar an der Abkürzung “Dst”. Beide Geräte befinden sich damit im gleichen LAN. Der Rechner mit der IP-Adresse 192.168.1.1 ist der Router. Dieses Gerät stellt die Internetverbindung her und kann DNS-Anfragen aus seinem Cache beantworten.

Im Header für das User Datagram Protocol (UDP) werden die Quell- und Ziel-Ports angezeigt.

```
User Datagram Protocol, Src Port: 53586, Dst Port: 53
```

Der Quellport wurde mit 53586 automatisch und weit oberhalb der sog. “Well-Known Ports” (0-1023) gewählt. Die “Well-Known Ports” sind Ports, die von bestimmten Anwendungen oder Diensten standardmässig verwendet werden. Entsprechend wurde der Zielport auf 53 gewählt, da dies der standardmässige Port für DNS-Anfragen ist. Eine Liste der “Well-Known Ports” findet sich in der offiziellen IANA-Portdatenbank. Der Quellport ermöglicht es dem Zielsystem, die Antwort an den korrekten Absender zurückzusenden. NAT-Geräte (vgl. Abschnitt Network Address Translation (NAT)) nutzen diese Port-Informationen zusätzlich für die Zuordnung zwischen privaten und öffentlichen Adressen.

Der zuunterst dargestellte Layer in der Detailansicht, beinhaltet die eigentliche Anfrage für die Übersetzung des Domainnamens in eine IP-Adresse.

```
Domain Name System (query)
    Transaction ID: 0x1f7a
    Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
    Queries
        www.deutschegrammophon.com: type A, class IN
    [Response In: 723]
```

Aus diesem Grund wird dieser Teil der Analyse hier auch aufgefaltet dargestellt.
Unter dem Stichwort `Queries` wird die gesuchte Adresse `www.deutschegrammophon.com` angezeigt. Das Stichwort `type A` zeigt an, dass es sich hier um eine Anfrage nach einer IPv4-Adresse handelt. IPv4-Adressen werden mit `A` bezeichnet, IPv6 mit `AAAA`. Das letzte Element in dieser Zeile ist die Klasse der Anfrage, in diesem Fall `IN` für das Internet. Obwohl heute fast ausschliesslich das Internet als Netzwerktyp verwendet wird, ist das Feld für die Klasse (IN) aus historischen Gründen weiterhin Teil jeder DNS-Anfrage.

Der entsprechende Inhalt der Antwort sieht folgendermassen aus:

```
Domain Name System (response)
    Transaction ID: 0x1f7a
    Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
    Queries
        www.deutschegrammophon.com: type A, class IN
    Answers
        www.deutschegrammophon.com: type A, class IN, addr 85.236.46.65
```

Das Paket wiederholt die Frage und liefert die Antwort des DNS-Servers. Der Domainname `www.deutschegrammophon.com` ist mit der IPv4-Adresse `85.236.46.65` verknüpft.

Damit kann die Verbindung zur Website `www.deutschegrammophon.com` hergestellt werden.

13.3 Beobachtung des Verbindungsaufbaus

Der Verbindungsaufbau zwischen Client (lokaler Rechner) und Server (Rechner im Internet) erfolgt in mehreren Schritten, die im sogenannten "Three-Way Handshake" zusammengefasst werden. Dieser Prozess stellt sicher, dass beide Seiten bereit sind, Daten zu senden und zu empfangen. Die folgende Abbildung zeigt eine schematische Darstellung des "Three-Way Handshake".



Figure 13.5: Schema Three-Way Handshake

Der Client sendet ein SYN-Paket an den Server, um eine Verbindung anzufordern. Dieser atwortet mit einem SYN-ACK-Paket. Das heisst, er bestätigt die Anfrage mit einem ACK und fragt seinerseits mit einem SYN nach, ob der Client (immer noch) bereit ist, die Verbindung aufzubauen. Damit klar ist, dass sich das ACK im SYN-ACK-Paket auf das ursprüngliche SYN-Paket bezieht, werden die einzelnen Pakete mit einer Sequenznummer (Sequence Number) versehen. Das ACK gibt die Sequenznummer des SYN-Paketes plus eins zurück.

Dieser Vorgang kann mit Wireshark beobachtet werden. Dafür braucht es einen kombinierten Wireshark Anzeigefilter. Als Beispiel wird der Verbindungsaufbau zwischen dem lokalen Rechner und der Website von www.deutschegrammophon.com betrachtet. Der erste Teil des Filters soll nur jene Pakete anzeigen, welche mit der IP-Adresse des Servers von www.deutschegrammophon.com (85.236.46.65) kommunizieren. Dieser Filter lautet

```
ip.addr == 85.236.46.65
```

Dieser Filter alleine zeigt jedoch noch zu viele Pakete an.



Figure 13.6: ip.addr Filter

Um die Resultate weiter einzuschränken sollen nur jene Pakete angezeigt werden, welche entweder das SYN-Flag oder das ACK-Flag (oder beides) gesetzt haben. Dies kann mit folgendem Filter erreicht werden:

```
ip.addr == 85.236.46.65 and (tcp.flags.syn == 1 or tcp.flags.ack == 1 )
```

Das sind allerdings immer noch zu viele Pakete. Daher sollen nur jene Pakete angezeigt werden, welche die Antworten auf eine SYN-Anfrage sind. Das kann erreicht werden, in dem ein Paket mit dem SYN-Flag mit der rechten Maustaste angeklickt wird und die Option “Follow” > “TCP Stream” ausgewählt wird. Dadurch wird der gesamte TCP-Stream, in dem dieses Paket steht, (die aufeinanderfolgenden Pakete) angezeigt. Der Filter wird automatisch angepasst.

Das folgende Listing zeigt den ganzen Filterbefehl für die Anzeige der Pakete, die zu diesem TCP-Stream gehören:

```
ip.addr == 85.236.46.65 and (tcp.flags.syn == 1 or tcp.flags.ack == 1) and !(tcp.stream eq
```



Figure 13.7: Three-Way Handshake Pakete

Dass die Pakte die Kommunikationsfolge des Three-Way Handshakes zeigen, ist an den gesetzten Flags zu erkennen. Im ersten Schritt sendet der Client ein SYN-Paket, worauf der Server mit einem SYN-ACK-Paket antwortet. Der Client bestätigt dies mit einem ACK-Paket. Diese drei Schritte sind im Wireshark-Filter sichtbar. Gut zu erkennen sind die jeweiligen Portnummern, die in den TCP-Paketen verwendet werden.

Anschliessend an diesen “Three-Way Handshake” kann der Client mit dem Server kommunizieren.

14 Python Sockets: Ein Einfaches Client-Server Beispiel

In diesem Abschnitt untersuchen wir, wie man eine einfache Client-Server-Anwendung mit Python-Sockets erstellt. Dieses Beispiel wird die grundlegenden Konzepte der Socket-Programmierung demonstrieren, einschliesslich der Herstellung einer Verbindung sowie des Sendens und Empfangens von Daten.

:::{note} Network Socket Ein Netzwerk-Socket ist ein Endpunkt innerhalb eines Hosts, der zum Senden und Empfangen von Daten über ein Netzwerk verwendet wird. :::

14.1 Struktur der Verbindung

Das Beispiel besteht aus einem Server, der auf eingehende Verbindungen wartet, und einem Client, der sich mit dem Server verbindet. Wir werden die Kommunikation als einfachen Chatdienst modellieren.

14.2 Server Code

Sie benötigen keine virtuelle Umgebung. Das Beispiel verwendet nur die Python-Standardbibliothek.
Unten ist das funktionierende Codebeispiel für den Socket-Server:

“‘lmuroetxqbgq python :linenos: # socket_server.py

import socket

def server_program(): # define host name and port host = ‘localhost’ # for communication on the local machine port = 5000 # use a port number above 1024

```
# create socket
server_socket = socket.socket()

# bind the socket to the host and port
server_socket.bind((host, port))

# set the server to listen for connections
server_socket.listen(2) # start listening
```

```
                        # with a backlog of 2
                        # (max queued connections)

# accept new connection from client
conn, address = server_socket.accept()
print("Connection from: " + str(address))

while True:
    # receive up to 1024 bytes per call
    data = conn.recv(1024).decode()
    if not data:
        # if no data is received, break
        break
    print("Received from connected client: " + str(data))
    # prompt the user to enter a message
    data = input(' -> ')
    # send data to the client as bytes
    conn.send(data.encode())

# close the connection
conn.close()
```

if **name** == '**main**': server_program()

```
Um das Skript direkt auszuführen, öffnen Sie ein Terminal im
Verzeichnis, in dem sich das Skript befindet, und führen Sie den
folgenden Befehl aus:

```bash
python socket_server.py
```

Was geschieht, wenn das Skript ausgeführt wird, wird in den folgenden Abschnitten erläutert.

Der Code besteht aus zwei Hauptteilen: der Funktion `server_program` und dem Block `if __name__ == '__main__'`. Die Funktion `server_program` enthält die Hauptlogik für den Server, während der `if`-Block verwendet wird, um den Servercode auszuführen, wenn das Skript direkt ausgeführt wird.

Wenn Sie das Skript ausführen, importiert Python zuerst das `socket`-Modul.

```
import socket
```

Als nächstes wird die Funktion `server_program` aufgerufen. Innerhalb dieser Funktion wird als erstes die Hostadresse und der Portnummer definiert. `localhost` ist eine spezielle
```

Adresse, die auf die lokale Maschine verweist. `localhost` entspricht der IPv4-Adresse `127.0.0.1`. Das bedeutet, dass der Server nur Verbindungen von Clients akzeptiert, die auf derselben Maschine ausgeführt werden. Dies ist eine Simulation eines Netzwerks.

```
def server_program():
    # define host name and port
    host = 'localhost'    # for communication on the local machine
    port = 5000           # use a port number above 1024
```

Im nächsten Code-Snippet wird der `server_socket` mit der Funktion `socket.socket()` erstellt, die ein neues Socket-Objekt erzeugt.

```
server_socket = socket.socket()
```

Dieses Socket wird verwendet, um auf eingehende Verbindungen von Clients zu hören. Damit die Verbindung funktioniert, muss der `server_socket` mit der Host- und Portnummer über die `bind()`-Methode verbunden werden.

```
server_socket.bind((host, port))
```

Als nächstes wird der Server so eingestellt, dass er mit der `listen()`-Methode auf eingehende Verbindungen wartet.

```
server_socket.listen(2) # start listening
                        # with a backlog of 2
                        # (max queued connections)
```

Der Server erstellt ein backlog von 2 ausstehenden Verbindungen. Der Code akzeptiert jedoch genau eine Client-Verbindung (accept() wird einmal aufgerufen). Um mehrere Clients zu verarbeiten, rufen Sie accept() in einer Schleife auf und verwenden Sie Threads oder asyncio. Dieses Limit wird festgelegt, um zu verhindern, dass der Server von zu vielen Verbindungen auf einmal überwältigt wird. Die `accept()`-Methode wird verwendet, um eine neue Verbindung von einem Client zu akzeptieren. Diese Methode gibt ein Tupel mit zwei Elementen zurück: `conn`, ein neues Socket-Objekt für die Kommunikation mit dem Client, und `address`, die Adresse des Clients. Der nächste Abschnitt zeigt, wie eine Nachricht ausgegeben wird, um die Adresse des verbundenen Clients anzuzeigen.

```
conn, address = server_socket.accept()
print("Connection from: " + str(address))
```

Die while-Schleife ermöglicht es dem Server, auf Daten vom Client zu warten. Die `recv()`-Methode wird verwendet, um Daten vom Client zu empfangen. Jeder `recv()`-Aufruf liest bis zu 1024 Bytes; zusätzliche Daten bleiben im Puffer und können durch nachfolgende Aufrufe gelesen werden. Wenn keine Daten empfangen werden, bricht der Server die Schleife ab und schliesst die Verbindung.

```
while True:
        # receive up to 1024 bytes per call
        data = conn.recv(1024).decode()
        if not data:
            # if no data is received, break
            break
```

Wenn der Server Daten empfängt, gibt er die Daten in der Konsole aus und fordert den Benutzer auf, eine Antwort einzugeben. Die Antwort wird mit der `send()`-Methode an den Client zurückgesendet.

```
print("Received from connected client: " + str(data))
        # prompt the user to enter a message
        data = input(' -> ')
        # send data to the client as bytes
        conn.send(data.encode())
```

14.3 Client Code

Unten ist der funktionierende Code des Client-Beispiels:

“‘lmuroetxqbgq python :linenos: # socket_client.py

import socket

def client_program(): # define host name and port (of the server to connect to) host = ‘localhost’ # as both pieces of code are # running on the same machine port = 5000 # socket server port number

```
# create socket
client_socket = socket.socket()

# connect to the server
client_socket.connect((host, port))

# prompt the user to enter a message
message = input(" -> ")  # take input

# message loop
while message.lower().strip() != 'bye':
    # send message to the server as bytes
    client_socket.send(message.encode())

    # receive response from the server
```

```
    data = client_socket.recv(1024).decode()

    print('Received from server: ' + data)

    # prompt the user to enter a new message
    message = input(" -> ")  # take new input

# close the connection
client_socket.close()
```

if **name** == '**main**': client_program()

```
Um dieses Skript auszuführen, befolgen Sie die gleichen Schritte wie für
das Server-Skript. Öffnen Sie ein Terminal im Verzeichnis, in dem sich
das Skript befindet, und führen Sie den folgenden Befehl aus:

```bash
python socket_client.py
```

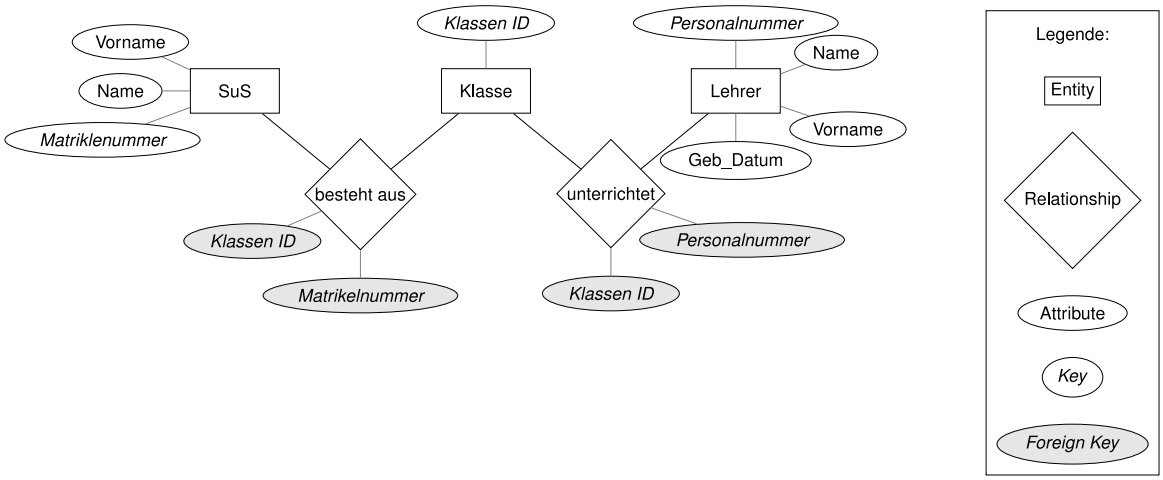
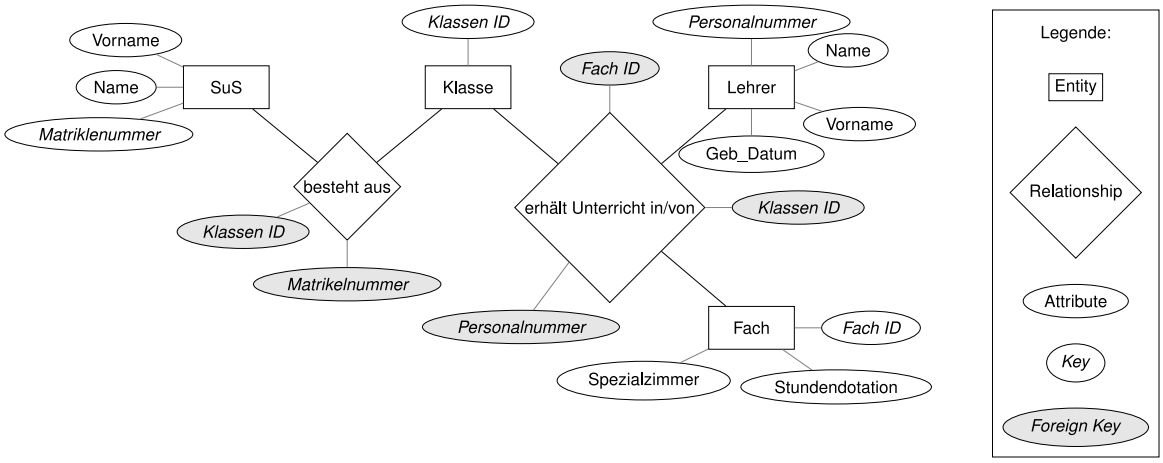
Starten Sie den Server, bevor Sie den Client starten. Wenn der Server nicht läuft, wird client_socket.connect(…) einen ConnectionRefusedError auslösen.

Wenn Sie das Skript ausführen, importiert Python zuerst das Modul `socket`.

Als nächstes wird die Funktion `client_program` aufgerufen. Innerhalb dieser Funktion wird die erste Aktion die Definition der Hostadresse und der Portnummer. Hier müssen die Hostadresse und die Portnummer mit denen im Server-Skript übereinstimmen.

Nachdem die Host- und Portinformationen definiert sind, erstellt der Client ein Socket-Objekt mit der Funktion `socket.socket()`. Dieses Socket wird verwendet, um eine Verbindung zum Server herzustellen. Die Methode `connect()` wird auf dem Socket-Objekt aufgerufen, um eine Verbindung zum Server herzustellen.

Sobald die Verbindung hergestellt ist, betritt der Client eine Schleife, in der er Nachrichten an den Server senden und Antworten empfangen kann. Der Client fordert den Benutzer auf, eine Nachricht einzugeben, die dann mit der Methode `send()` an den Server gesendet wird. Der Client wartet auch auf eine Antwort vom Server, indem er die Methode `recv()` verwendet.

Wenn der Benutzer `bye` eingibt, verlässt der Client die Schleife und schliesst die Verbindung zum Server.
```

14.4 Verbindung im LAN

Um die beiden Skripte im lokalen Netzwerk (LAN) zu verwenden, muss in beiden Skripten die IPv4-Adresse des Servers anstelle von `localhost` verwendet werden.
Die IPv4-Adresse des Servers kann durch Ausführen des folgenden Befehls im Terminal des Servers ermittelt werden:

```
C:\Users\username>ipconfig
Windows IP Configuration
...
Wireless LAN adapter Wi-Fi:
   Connection-specific DNS Suffix  . . . :
   IPv4 Address . . . . . . . . . . . .: 192.168.1.2
   Subnet Mask . . . . . . . . . . . . : 255.255.255.255
   Default Gateway . . . . . . . . . . : 192.168.1.1
...
```

Die Ausgabe des Befehls `ipconfig` zeigt alle Netzwerkadapter des entsprechenden Computers an. Die dargestellte Ausgabe hier im Beispiel wurde gekürzt.
Die IPv4-Adresse des Servers ist in diesem Fall `192.168.1.2`.

Diese ist die Adresse, welche in beiden Skripten anstelle von `localhost` verwendet werden muss. Dann kann innerhalb des lokalen Netzwerks eine Verbindung hergestellt werden.

14.5 Fazit

Die vorgestellten Codebeispiele veranschaulichen, wie Pythons integriertes Socket-Modul (Standardbibliothek) eine unkomplizierte Implementierung von netzwerkbasierten Anwendungen ermöglicht, ohne externe Abhängigkeiten zu erfordern. Der Fokus des Tutorials auf die Kommunikation über localhost bietet eine sichere, kontrollierte Umgebung für Experimente und Lernen, während der bidirektionale Nachrichtenaustausch reale Kommunikationsmuster demonstriert, die in Produktionssystemen zu finden sind.

Zu den wichtigsten Erfolgen dieser Implementierung gehören:

- Erfolgreiche Herstellung von TCP-Socket-Verbindungen zwischen separaten Prozessen
- Implementierung einer nachrichtenbasierten Kommunikation mit ordnungsgemässer Kodierung/Dekodierung
- Demonstration des Verbindungslebenszyklusmanagements von der Herstellung bis zur Beendigung
- Erstellung einer einfachen Befehlszeilenschnittstelle für sowohl Server- als auch Clientanwendungen

Das Wissen, das aus diesem Tutorial gewonnen wurde, dient als solide Grundlage für die Entwicklung anspruchsvollerer netzwerkbasierter Anwendungen. Zukünftige Verbesserungen könnten die Implementierung von Multithreading für die gleichzeitige Verarbeitung mehrerer Clients, die Hinzufügung von Fehlerbehandlungs- und Wiederverbindungslogik oder die Erweiterung des Kommunikationsprotokolls zur Unterstützung strukturierter Datenformate umfassen.

Part IV

Datenbanken

15 Einführung in Datenbanken

Die Einführung in Datenbanken basiert auf dem Einführungskapitel aus dem Buch Abraham Silberschatz, Henry F. Korth und S. Sudarshan; Database system concepts; Seventh edition; New York 2020.

15.1 Einführung

Die bisher besprochenen Datenstrukturen Dictionary, Stack, Queue und Binary Search Tree dienen der Bearbeitung von Daten im Arbeitsspeicher. Sie sind daher auf einen beschränkten Umfang von Datensätzen ausgelegt. Ausserdem dienen sie nicht der permanenten Ablage von Daten.

Im Gegensatz dazu dienen Datenbanken der dauerhaften Ablage grosser Datensätze. Darüber hinaus sollen sie die effiziente Verfügbarkeit und die Integrität der Daten sicherstellen.

15.2 Charakteristika von Datenbanken

Ein wichtiges Merkmal von Datenbanken ist es, dass die gespeicherten Daten nur einmal abgelegt werden. Damit kann verhindert werden, dass mehrfach abgespeicherte Daten (redundante Daten) lediglich an einer Stelle modifiziert werden und damit Widersprüche entstehen. Der Entwurf von Datenbanken muss dem Rechnung tragen. Ein Hilfsmittel für den Entwurf von Datenbanken ist das ER-Diagramm (Entity-Relationship-Diagramm). Das ER-Diagramm ist eine grafische Darstellung der Datenbankstruktur. Es zeigt die Entitäten (durch die Datenbank modellierte Dinge der realen Welt), die in der Datenbank gespeichert werden, mit ihren Attributen (Eigenschaften der modellierten Dinge) sowie die Beziehungen (Relationship) zwischen den Entitäten.

Um einen Eintrag in der Datenbank eindeutig identifizieren zu können, wird jedem Eintrag ein Primärschlüssel zugeordnet. Der Primärschlüssel ist ein Attribut oder eine Kombination von Attributen, die den Eintrag eindeutig identifiziert.
In den Beziehungen werden die Primärschlüssel der Entitäten als Fremdschlüssel verwendet. Ein Fremdschlüssel ist ein Attribut oder eine Kombination von Attributen, die auf den Primärschlüssel einer anderen Entität verweisen. So kann eine Beziehung zwischen zwei Entitäten hergestellt werden.

Die untenstehende Graphik zeigt eine Skizze eines ER-Diagramms, in welchem die Beziehungen zwischen Schülern, Klassen und Lehrern dargestellt wird. Ausserdem wird gezeigt, welche Attribute als Primär- bzw. Fremdschlüssel verwendet werden.



Figure 15.1: ER-Diagramm

Jede Datenbank ist immer eine Vereinfachung der Wirklichkeit und daher immer unvollständig bzw. erweiterbar. Das obige ER-Diagramm kann daher um eine zusätzliche Entität `Fach` erweitert werden und sieht dann folgendermassen aus:



Figure 15.2: ER-Diagramm

Diese Grafiken können in Datenbanktabellen übersetzt werden. In den Tabellen werden die Primärschlüssel unterstrichen. Die Primärschlüssel sind in der Regel die ersten Spalten der Tabellen. Für das Beispiel werden Tabellen für die Entitäten `Klasse`, `Fach` und `Lehrer` erstellt.

Die einfachste Tabelle ist die Tabelle `Klasse`. Sie enthält lediglich ein Attribut (den Primärschlüssel).

Klasse
Klassen ID
aW_24-28
bW_24-28
cW_24-28
dP_24-28
eW_24-28
fP_24-28

Figure 15.3: Klasse

Etwas umfangreicher sind die Tabellen `Fach` und `Lehrer`. Sie enthalten jeweils drei bzw. vier Attribute.

Fach		
Fach ID	*Stundendotation*	*Spezialzimmer*
Deutsch	4	—
Französisch	4	—
Englisch	4	—
Mathe	4	—
Biologie	2	Praktikumszimmer B
Physik	2	Praktikumszimmer P
Chemie	2	Chemielabor
Geographie	2	—
Geschichte	2	—
Informatik	2	—
WR	4	—
PPP	4	—
BG	2	Zeichensaal
Musik	2	—
Sport	3	Turnhalle

Le		
Personalnummer	*Name*	*Vo*
0001	Schiller	Fr
0002	Balzac	Ho
0003	Sand	Ge
0004	Gauss	Jo
0005	Darvin	Ch
0006	Curie	Ma
0007	Friedman	Mi
0008	Redlich	Os
0009	Dufour	Gu
0010	Kollwitz	Kä
0011	Callas	Ma
0012	Lovlace	Ad
0013	Piaget	Je
0014	Montessori	Ma

Im Folgenden Abschnitt sollen die Daten aus den Tabellen mit Hilfe von SQL Statements abgefragt werden.

16 Abfrage von Daten

Datenbanken werden mit einer spezifischen Datenbanksprache angesprochen. Im Gegensatz zur bisher im Unterricht verwendeten Programmiersprache Python ist die Datenbanksprache SQL (Structured Query Language) eine deklarative Sprache. In Python werden die Befehle grundsätzlich der Reihe nach abgearbeitet. In SQL wird das gewünschte Resultat beschrieben. Wie diese Beschreibung abgearbeitet wird, ist in den Grundlagen der Datenbank programmiert.

16.1 Grundstruktur einer SQL Abfrage

Die Grundstruktur einer SQL Abfrage ist im untenstehenden Code Snippet dargestellt.

```
SELECT <Spalten>
FROM <Tabelle>
WHERE <Bedingung>;
```

Das Schlüsselwort `SELECT` gibt an, welche Spalte(n) aus der Tabelle ausgegeben werden soll(en). Das Schlüsselwort `FROM` gibt an, aus welcher Tabelle die Daten ausgelesen werden. Das Schlüsselwort `WHERE` gibt die Bedingung an, die erfüllt sein muss, damit die Daten angezeigt werden. Dass die Schlüsselwörter in Grossbuchstaben geschrieben werden, ist technisch nicht nötig, entspricht aber der Konvention. Die Abfrage wird mit einem Semikolon abgeschlossen.

16.1.1 Einfache Abfrage

In einem ersten Beispiel sollen alle Vornamen aller Lehrer aus der Tabelle Lehrer aus dem vergangenen Abschnitt angezeigt werden:

```
SELECT Vorname
FROM Lehrer;
```

In diesem Beispiel wurde auf die Formulierung einer Bedingung verzichtet. Wenn die Ausgabe zusätzlich eine Bedingung erfüllen soll, wird diese mit dem Schlüsselwort `WHERE` angegeben. Im folgenden Beispiel sollen nur die Vornamen der Lehrer angezeigt werden, die vor dem Jahr 1800 geboren sind.

16.1.2 Abfrage mit Bedingung

```
SELECT Vorname
FROM Lehrer
WHERE Geburtsdatum < '1800-01-01';
```

Diese Abfrage führt zu folgendem Ergebnis:

Vorname
Friedrich
Honore de
Johann Carl Friedrich
Guillaume-Henri

16.1.3 Sortierung der Ausgabe

Falls die Ausgabe nicht nur die Vornamen, sondern auch die Nachnamen und das Geburtsdatum enthalten soll und die Ausgabe nach dem Geburtsdatum aufsteigend sortiert werden soll, wird die Abfrage entsprechend angepasst:

```
SELECT Name, Vorname, Geburtsdatum
FROM Lehrer
WHERE Geburtsdatum < '1800-01-01'
ORDER BY Geburtsdatum;
```

Diese Abfrage führt zu folgendem Ergebnis:

Name	Vorname	Geburtsdatum
Schiller	Friedrich	10.11.1759
Gauss	Johann Carl Friedrich	30.04.1777
Dufour	Guillaume-Henri	15.09.1787
Balzac	Honoré de	20.05.1799

Es können dem Schlüsselwort `SELECT` mehrere Spalten übergeben werden. Zusätzlich wurde in der Anfrage das Schlüsselwort `ORDER BY` verwendet. Mit diesem kann angegeben werden, nach welchem Kriterium die Ausgabe sortiert werden soll. Standardmässig wird aufsteigend sortiert. Mit dem Schlüsselwort `DESC` kann die Sortierung absteigend erfolgen. Die Abfrage sieht dann folgendermassen aus:

```
SELECT Name, Vorname, Geburtsdatum
FROM Lehrer
WHERE Geburtsdatum < '1800-01-01'
ORDER BY Geburtsdatum DESC;
```

Die Sortierreihenfolge wird hinter das Kriterium geschrieben. Wenn nach mehreren Kriterien sortiert werden soll, werden die zusätzlichen Kriterien mit einem Komma an das erste Kriterium angehängt.

16.2 Abfrage aus mehreren Tabellen

Interessanter, als die Abfrage von Daten aus einer einzigen Tabelle, ist die Abfrage aus mehreren Tabellen. So ist es im Beispiel möglich, Abzufragen, wer Deutsch unterrichtet. Aus diesem Grund wurde die Tabelle `erhält Unterricht in/von` angelegt.

erhält Unterricht in/von		
Fach ID	*Klassen ID*	*Personalnummer*
Deutsch	aW_24-28	0001
Deutsch	bW_24-28	0001
Deutsch	cW_24-28	0001
Deutsch	dP_24-28	0001
Deutsch	eW_24-28	0001
Deutsch	fP_24-28	0001
Französisch	aW_24-28	0002
...	...	...

Figure 16.1: erhält Unterricht in/von

Um abzufragen, wer Deutsch unterrichtet, müssen die Daten aus den Tabellen `Lehrer`, `Fach` und `erhält Unterricht in/von` zusammengeführt werden. Dies geschieht mit dem Schlüsselwort `JOIN`. Das Schlüsselwort `JOIN` kann unterschiedlich verwendet werden. Im vorliegenden Beispiel wird die Variante `INNER JOIN` verwendet.

```
SELECT DISTINCT l.Name, l.Vorname
FROM Lehrer AS l
INNER JOIN erhält_Unterricht_in AS u ON l.Personalnummer = u.Personalnummer
WHERE u.Fach_ID = 'Deutsch';
```

Das Resultat dieser Abfrage sieht wie folgt aus:

Name	Vorname
Schiller	Friedrich

In Ergänzung zu den bisherigen Abfragen, kommt neu das Schlüsselwort `DISTINCT` zum Einsatz. Dieses bewirkt, dass Daten, die mehrfach vorkommen, nur einmal ausgegeben werden. In diesem Beispiel wäre dies nicht nötig, da es nur einen Lehrer gibt, der Deutsch unterrichtet.
Unter dem Schlüsselwort `FROM` wird die Tabelle `Lehrer` mit dem Alias `l` angegeben. Der Alias wird verwendet, um die Abfrage leserlicher zu machen. Wenn mehrere Tabellen abgefragt werden, muss jede Spalte, die ausgeben werden soll, mit der Tabelle, aus der sie stammt, angegeben werden. Mit dem Alias kann dies abgekürzt werden. Das Schlüsselwort `AS` für den Alias ist nicht nötig, dient aber der besseren Lesbarkeit.
Mit dem Schlüsselwort `INNER JOIN` werden die Datensätze aus den beiden Tabellen `Lehrer` und `erhält_Unterricht_in` basierend auf übereinstimmenden Werten in der Spalte `Personalnummer` miteinander verbunden. Dabei entsteht eine neue Ergebnismenge, die alle Spalten beider Tabellen enthält, jedoch nur für diejenigen Zeilen, bei denen die `Personalnummer` in beiden Tabellen übereinstimmt.
Aus dieser Schnittmenge werden aus der Tabelle `erhält Unterricht in/von` die Lehrer ausgewählt, die Deutsch unterrichten. Dies geschieht mit dem Schlüsselwort `WHERE` und dem Kriterium `u.Fach_ID = 'Deutsch'`.

Die Abfrage, wer die Klasse fP_24-28 in PPP unterrichtet, sieht wie folgt aus:

```
SELECT l.Name, l.Vorname
FROM Lehrer AS l
INNER JOIN erhält_Unterricht_in AS u ON l.Personalnummer = u.Personalnummer
WHERE u.Fach_ID = 'PPP'
AND u.Klassen_ID = 'fP_24-28';
```

Die Abfrage gibt folgendes Resultat zurück:

Name	Vorname
Piaget	Jean

Gegenüber der Abfrage, wer Deutsch unterrichtet, wurde mit dem Schlüsselwort `AND` die zusätzliche Bedingung `u.Klassen_ID = 'fP_24-28'` hinzugefügt.

16.3 Ausblick

Der nächste Abschnitt dient dazu, SQL zu üben. Als Übungsplattform wird SQL Island genutzt. Diese Plattform ist unter sql-island.informatik.uni-kl.de zu finden.

17 Lernziele für die Prüfung vom 21. Mai 2025

Ich erwarte, dass Sie in der Lage sind

- die grundsätzlichen Unterschiede zwischen Python und SQL zu erklären;
- die Aufgaben einer Datenbank aufzuzählen;
- die Elemente einer grundlegenden SQL Abfrage aufzuzählen sowie
- eine SQL Abfrage in umgangssprachliches Deutsch zu übersetzen.

Für Fragen stehe ich Ihnen im Kanal *Allgemein* zur Verfügung.

Part V

Datenstrukturen und Codierung

18 Datenstrukturen Stack und Queue

Zwei wichtige Datenstrukturen zur Lösung alltäglicher Probleme sind der Stack (Stapelspeicher) und Queue (Warteschlange). In beiden Datenstrukturen können mehrere Werte zwischengespeichert werden. Die beiden Speicherformen sollen hier kurz erläutert werden.

In einem Stack werden die gespeicherten Werte *gestapelt*. Wie in einem Stapel in der realen Welt, wird der letzte gespeicherte Wert zuoberst auf den Stapel gelegt. Ebenso wie in einem realen Stapel, können die gespeicherten Werte nur in umgekehrter Reihenfolge zu ihrer Speicherung wieder abgerufen werden (last in - first out; LIFO). Mit Bezug auf das untenstehende Bild bedeutet dies, dass der letzte Brotlaib, der auf den Stapel gelegt wurde, auch der erste ist, der wieder vom Stapel weggenommen werden kann.
Ein Stack kann dazu verwendet werden, um die Verarbeitungsreihenfolge von Rechenoperationen in einem Programm abzubilden.

“‘cshetzsr brotstapel.png :alt: Brotstapel :width: 300px :name: brotstapel

Martin Mayer mit einem Stapel Brot (Quelle: Sasa Noël und Heike Grein, Brothandwerk, Aarau und München, 2021, Seite 50.)

```
In einer Queue werden die gespeicherten Werte in einer Warteschlange
*eingereiht*. Wie in einer Warteschlange in der realen Welt, wird jeder neu
gespeicherte Wert hinten eingereiht. Beim Abrufen der gespeicherten Werte wird
der zuerst gespeicherte Wert zuerst verarbeitet (first in - first out; FIFO).
Eine Queue kann dazu verwendet werden, eine Warteschlange abzubilden wie sie in
Netzwerken für die Übermittlung von Datenpaketen gebraucht wird.

```{figure} unemployment_line.jpg
---
height: 300px
name: unemployment-line
---
Dole Queue Great Depression (Quelle: https://view.genially.com/609aac10d34c960d5992809a/in
```

Für die Übungen finden Sie hier eine Klasse Node und eine Klasse Linked List zum Download.
```

19 Queues in Python

Queues sind Datenstrukturen, welche Daten speichern und grundsätzlich in der Reihenfolge, in der sie abgespeichert worden sind, wieder zurückgeben (First In - First Out, FIFO). Eine Queue funktioniert damit wie eine Warteschlange an einer Kasse: Wenn eine neue Person ankommt, stellt sie sich hinten an. Wenn sie die Kasse erreicht (nachdem alle, die vorher angestanden sind, bedient worden sind), verlässt sie die Schlange wieder.

Eine Queue ist eine derart fundamentale Datenstruktur, dass Python sie als Library zur Verfügung stellt.

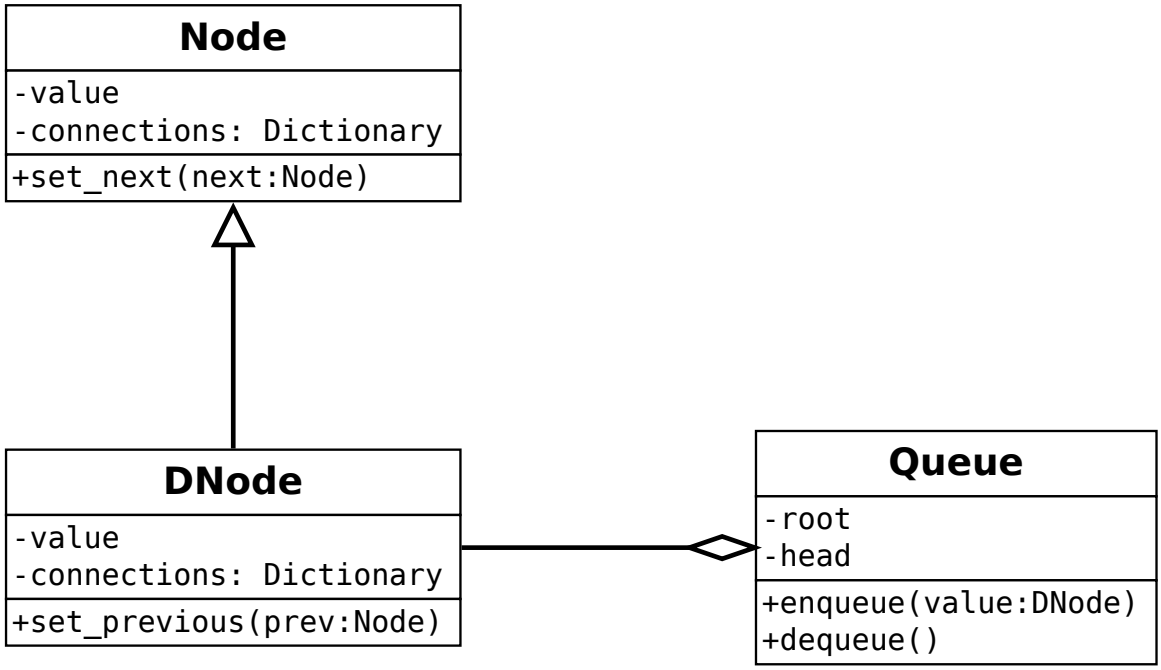
Um zu verstehen, wie eine Queue funktioniert, geht es im folgenden darum, eine eigene Klasse Queue in Python zu implementieren.

Zu Beginn ist zu überlegen, welche Eigenschaften, die Queue aufweisen muss. Sie muss Daten abspeichern und diese in der gleichen Reihenfolge wieder ausgeben können. Wir brauchen also eine Struktur für die Daten und eine Struktur, welche die Reihenfolge der Speicherung festhält. Die Struktur, welche die Reihenfolge festhält, muss ausserdem in der Lage sein, neue Daten abzuspeichern und bereits abgespeicherte Daten wieder zurückzugeben. Diese Anforderungen können mit Hilfe bereits programmierter Klassen umgesetzt werden. Um die Daten abzuspeichern, können wir Nodes verwenden und für die Struktur zum Erhalt der Reihenfolge die Linked List.

```
ovemvbmplz Linked List Die Linked List ist eine Datenstruktur, die aus
einer Anzahl von Nodes besteht. Jeder Node hat einen Bezug auf den nächsten
Node in der Liste. Damit ist es möglich, die Nodes in einer bestimmten
Reihenfolge abzuspeichern.  Im Node selber gibt es ein Datenfeld `key`
für die Bezeichnung des Nodes, eines `value` für die Daten, die gespeichert
werden sollen sowie ein solches für den Verweis auf den nächsten Node.
Die Datenstruktur  Linked List stellt in erster Linie den Verweis auf den
zuletzt eingefügten Node zur Verfügung.
```

In der aktuellen Implementation der Linked List gibt es nur einen Positionsbezug auf das letzte eingefügte Element (`self.root`). Damit die Linked List als Queue verwendet werden kann, muss auch ein Bezug auf das erste eingefügte Element angelegt werden (`self.head`). Für das Erste überhaupt in die Datenstruktur eingefügte Element ist dies kein Problem. Wenn aber weitere Elemente eingefügt oder entfernt werden, dann muss in den einzelnen Nodes nicht nur ein Bezug auf das folgende Element (`self.connections['next']`) sondern auch einer auf das Vorangehende Element (`self.connections['previous']`). Entsprechend müssen die beiden Klassen Node und Linked List angepasst werden.

Das kann umgesetzt werden, in dem basierend auf den bereits existierenden Klassen abgeleitete Klassen implementiert werden. Als UML-Klassendiagramm sieht das folgendermassen aus:

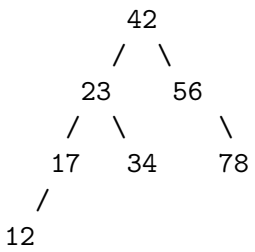


Für die Umsetzung der obigen Ausführungen stehen hier zwei Module (linked_list.py und nodes.py) zur Verfügung.

20 Binary Search Tree

Ein binärer Suchbaum (Binary Search Tree, BST) ist eine Datenstruktur, die es ermöglicht, Daten in einer hierarchischen Struktur zu speichern. Jeder Knoten im Baum hat maximal zwei Kinder, wobei das linke Kind kleiner und das rechte Kind grösser als der Knoten selbst ist. Dies ermöglicht das effiziente Suchen, Einfügen und Löschen von Elementen.

Sollen beispielsweise die Zahlen 42, 23, 17, 34, 56, 78 und 12 der Reihe nach in einen binären Suchbaum eingefügt werden, geschieht dies wie folgt:



Um einen Knoten zu löschen, gibt es drei Fälle zu beachten: 1. Der Knoten ist ein Blatt (keine Kinder): Der Knoten kann einfach entfernt werden. 2. Der Knoten hat ein Kind: Das Kind ersetzt den Knoten. 3. Der Knoten hat zwei Kinder: Der Knoten wird durch den kleinsten Knoten im rechten Teilbaum ersetzt.

In den folgenden Abschnitten findet sich eine Mögliche Implementierung eines binären Suchbaums in Python.

20.1 Klasse BSTNode

```python
class BSTNode:
    def __init__(self, key, value=None):
        self.key = key
        self.value = value
        self.parent = None
        self.left = None
        self.right = None

    def __str__(self):
        key = str(self.key)
```

```
        parent = 'None' if self.parent is None else str(self.parent.key)
        left = 'None' if self.left is None else str(self.left.key)
        right = 'None' if self.right is None else str(self.right.key)
        s = (
            f'\tParent = {parent}\n'
            f'\tKey = {key}\n'
            f'Left = {left}\tRight = {right}'
        )
        return s
```

20.2 Klasse BST

```
class BST:
    def __init__(self, key=None, value=None):
        if key is None:
            self.root = None
        else:
            node = BSTNode(key, value)
            self.root = node

    def insert(self, key, value=None, root=None):
        node = BSTNode(key, value)
        if self.root is None:
            self.root = node
            return

        if root is None:
            root = self.root

        if key < root.key and root.left is None:
            root.left = node
            node.parent = root
            return

        if key < root.key:
            root = root.left
            self.insert(key, value, root)


        if key > root.key and root.right is None:
            root.right = node
            node.parent = root
            return
```

```
        if key > root.key:
            root = root.right
            self.insert(key, value, root)

    def min(self, bst=None):
        if bst is None:
            minimum = self.root
        else:
            minimum =bst.root

        while minimum.left is not None:
            minimum = minimum.left

        return minimum

    def max(self, bst=None):
        if bst is None:
            maximum = self.root
        else:
            maximum = bst.root

        while maximum.right is not None:
            maximum = maximum.right

        return maximum

    def search(self, key, node=None):
        # If initial call or we've hit None in recursion
        if node is None:
            if self.root is None:  # Empty tree
                return -1
            node = self.root

        # Found the key
        if key == node.key:
            return node

        # Key doesn't exist in this path
        if key < node.key:
            if node.left is None:
                return -1
            return self.search(key, node.left)
        else:  # key > node.key
            if node.right is None:
                return -1
```

```
            return self.search(key, node.right)

    def delete(self, key):
        # Find the node to delete
        node = self.search(key)

        # If node not found, return
        if node == -1:
            return

        self._delete_node(node)

    def _delete_node(self, node):
        # Case 1: Node has no children (leaf node)
        if node.left is None and node.right is None:
            if node == self.root:
                self.root = None
            else:
                if node.parent.left == node:
                    node.parent.left = None
                else:
                    node.parent.right = None

        # Case 2: Node has only one child
        elif node.left is None:  # Has only right child
            if node == self.root:
                self.root = node.right
                node.right.parent = None
            else:
                if node.parent.left == node:
                    node.parent.left = node.right
                else:
                    node.parent.right = node.right
                node.right.parent = node.parent

        elif node.right is None:  # Has only left child
            if node == self.root:
                self.root = node.left
                node.left.parent = None
            else:
                if node.parent.left == node:
                    node.parent.left = node.left
                else:
                    node.parent.right = node.left
                node.left.parent = node.parent
```

```
        # Case 3: Node has two children
        else:
            # Find successor (smallest node in right subtree)
            successor = None
            current = node.right

            while current.left is not None:
                current = current.left

            successor = current

            # Copy successor's key and value to the node
            node.key = successor.key
            node.value = successor.value

            # Delete the successor (which has at most one right child)
            self._delete_node(successor)

    def iterate(self, node=None, result=None):
        # Initialize result list on first call
        if result is None:
            result = []

        # Use root if no starting node provided
        if node is None:
            if self.root is None:  # Empty tree
                return result
            node = self.root

        # In-order traversal: left -> current -> right
        if node.left is not None:
            self.iterate(node.left, result)

        result.append(node)

        if node.right is not None:
            self.iterate(node.right, result)

        return result
```

21 Das Binärsystem

Wir sind es gewohnt uns im Dezimalsystem zu bewegen. Das bedeutet, dass wir Zahlen in der Basis 10 darstellen. Wir verwenden dazu 10 verschiedene Ziffern, nämlich die Ziffern 0 bis 9.
Um Zahlen darzustellen, welche grösser sind als 9 darzustellen, verwenden wir die Zehnerpotenzen. Das heisst, dass die Ziffern von rechts nach links gelesen, die Zehnerpotenzen von 0 an aufsteigend sind. Die Zahl 123 entspricht also der Rechnung $1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$.

Das Binärsystem funktioniert nach dem gleichen Prinzip, nur dass wir nur die Ziffern 0 und 1 verwenden. Das bedeutet, dass wir Zahlen in der Basis 2 darstellen. Sobald eine Zahl grösser als 1 dargestellt werden soll, schreiben wir die Zahl als Summe von Zweierpotenzen. Die Zahl 101 entspricht also der Rechnung $1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$.

21.1 Umrechnung von Dezimalzahlen in Binärzahlen

Um eine Dezimalzahl in eine Binärzahl umzurechnen, gibt es eine systematische Methode, die auf wiederholter Division durch 2 basiert:

1. Teile die Dezimalzahl durch 2 und notiere den Rest (0 oder 1).
2. Teile den Quotienten (Ergebnis der Division) erneut durch 2 und notiere wieder den Rest.
3. Fahre damit fort, bis der Quotient 0 ist.
4. Die Binärzahl wird nun gebildet, indem man die Reste von unten nach oben (vom letzten zum ersten) liest.

21.1.1 Beispiel: Umrechnung von 42 ins Binärsystem

Lassen Sie uns die Dezimalzahl 42 in eine Binärzahl umrechnen:

Division	Rechnung	Quotient	Rest
$42 \div 2$	$42 = 21 \cdot 2 + 0$	21	0
$21 \div 2$	$21 = 10 \cdot 2 + 1$	10	1
$10 \div 2$	$10 = 5 \cdot 2 + 0$	5	0
$5 \div 2$	$5 = 2 \cdot 2 + 1$	2	1
$2 \div 2$	$2 = 1 \cdot 2 + 0$	1	0
$1 \div 2$	$1 = 0 \cdot 2 + 1$	0	1

Nun lesen wir die Reste von unten nach oben: 101010

Also ist 42 im Dezimalsystem gleich 101010 im Binärsystem.

21.1.2 Überprüfung:

$1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 32 + 0 + 8 + 0 + 2 + 0 = 42$

Eine alternative Methode zur Umrechnung ist die Verwendung der Zweierpotenzen. Man zerlegt die Dezimalzahl in eine Summe von Zweierpotenzen und schreibt dann eine 1 an den Stellen der verwendeten Potenzen und eine 0 an allen anderen Stellen.

21.2 Umrechnung von Binärzahlen in Dezimalzahlen

Die Umrechnung von Binärzahlen in Dezimalzahlen ist vergleichsweise einfach und basiert auf der Berechnung der Stellenwerte im Binärsystem.

21.2.1 Methode:

1. Identifiziere jede Stelle in der Binärzahl und ihre Position (von rechts nach links beginnend bei 0).
2. Multipliziere jede Binärziffer (0 oder 1) mit dem Wert von 2 hoch der Stellenposition.
3. Addiere alle resultierenden Werte zusammen.

21.2.2 Beispiel: Umrechnung von 10110 ins Dezimalsystem

Lassen Sie uns die Binärzahl 10110 in eine Dezimalzahl umrechnen:

Position (von rechts)	4	3	2	1	0
Binäre Ziffer	1	0	1	1	0
Berechnung	$1 \cdot 2$	$0 \cdot 2^3$	$1 \cdot 2^2$	$1 \cdot 2^1$	$0 \cdot 2$
Dezimalwert	16	0	4	2	0

Nun addieren wir alle Dezimalwerte: 16 + 0 + 4 + 2 + 0 = 22

Also ist 10110 im Binärsystem gleich 22 im Dezimalsystem.

21.2.3 Formel:

Für eine Binärzahl mit n Stellen (b , b , ..., b) gilt:

Dezimalzahl = $b \cdot 2 + b \cdot 2^1 + b \cdot 2^2 + \ldots + b \cdot 2^{1}$

wobei b die erste Stelle von rechts ist.

21.2.4 Praktischer Trick:

Bei der Umrechnung kann man auch eine Tabelle mit den Stellenwerten erstellen:

2	2	2	2	2^3	2^2	2^1	2
128	64	32	16	8	4	2	1

Man schreibt die Binärzahl unter die Tabelle und addiert nur die Werte, an deren Position eine 1 steht.

21.3 Binärcodierung von Strings

In der Informatik werden Textzeichen (Strings) durch binäre Codes repräsentiert. Bei Unicode-Zeichen wird eine fortschrittliche Codierung verwendet, um sowohl einfache als auch komplexe Schriftzeichen aus allen Sprachen der Welt darstellen zu können.

21.3.1 Unicode: Ein universeller Zeichencode

Unicode ist ein internationaler Standard zur Darstellung von Textzeichen aller Schriftsysteme. Jedes Zeichen erhält eine eindeutige Nummer (Codepoint), z.B. hat der Buchstabe "A" den Codepoint U+0041 (dezimal: 65).

21.3.2 UTF-8: Die häufigste Unicode-Codierung

UTF-8 ist die am weitesten verbreitete Codierungsform für Unicode. Sie hat folgende Eigenschaften:

1. **Variable Länge**: Ein Zeichen wird mit 1 bis 4 Bytes codiert
2. **Abwärtskompatibilität**: ASCII-Zeichen (0-127) werden mit nur 1 Byte dargestellt
3. **Selbstsynchronisierend**: Der Anfang eines Zeichens ist eindeutig erkennbar

21.3.3 UTF-8 Codierungsschema:

Anzahl Bytes	Binärmuster	Codepoint-Bereich
1	`0xxxxxxx`	U+0000 bis U+007F
2	`110xxxxx 10xxxxxx`	U+0080 bis U+07FF
3	`1110xxxx 10xxxxxx 10xxxxxx`	U+0800 bis U+FFFF
4	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`	U+10000 bis U+10FFFF

Die mit `x` markierten Stellen werden mit den Bits des Unicode-Codepoints gefüllt.

21.3.4 Beispiel: Codierung deutscher Umlaute

Nehmen wir als Beispiel den Buchstaben "ü" (U+00FC):

1. Der Codepoint von "ü" ist U+00FC (dezimal: 252)
2. Als Binärzahl: 11111100
3. Da der Wert > 127 ist, benötigen wir 2 Bytes
4. Nach dem 2-Byte-Schema: `110xxxxx 10xxxxxx`
5. Wir füllen den Unicode-Wert ein: `110(00000) 10(111100)` → `11000011 10111100`
6. Somit wird "ü" als die zwei Bytes `11000011 10111100` codiert

21.3.5 Weiteres Beispiel: Das Euro-Zeichen €

1. Der Codepoint von "€" ist U+20AC (dezimal: 8364)
2. Als Binärzahl: 10000010101100
3. Da der Wert > 2047 ist, benötigen wir 3 Bytes
4. Nach dem 3-Byte-Schema: `1110xxxx 10xxxxxx 10xxxxxx`
5. Wir füllen den Unicode-Wert ein: `1110(0010) 10(000010) 10(101100)` → `11100010 10000010 10101100`
6. Somit wird "€" als die drei Bytes `11100010 10000010 10101100` codiert

21.3.6 Beispiel: 'Informatik ist interessant.'

Lassen Sie uns den Satz "Informatik ist interessant." in die binäre UTF-8-Kodierung umwandeln:

Zeichen	Unicode-Codepoint	Dezimal	Binär	UTF-8 Kodierung
I	U+0049	73	1001001	01001001
n	U+006E	110	1101110	01101110
f	U+0066	102	1100110	01100110
o	U+006F	111	1101111	01101111
r	U+0072	114	1110010	01110010
m	U+006D	109	1101101	01101101
a	U+0061	97	1100001	01100001
t	U+0074	116	1110100	01110100
i	U+0069	105	1101001	01101001
k	U+006B	107	1101011	01101011
␣	U+0020	32	100000	00100000
i	U+0069	105	1101001	01101001
s	U+0073	115	1110011	01110011
t	U+0074	116	1110100	01110100
␣	U+0020	32	100000	00100000
i	U+0069	105	1101001	01101001
n	U+006E	110	1101110	01101110

Zeichen	Unicode-Codepoint	Dezimal	Binär	UTF-8 Kodierung
t	U+0074	116	1110100	01110100
e	U+0065	101	1100101	01100101
r	U+0072	114	1110010	01110010
e	U+0065	101	1100101	01100101
s	U+0073	115	1110011	01110011
s	U+0073	115	1110011	01110011
a	U+0061	97	1100001	01100001
n	U+006E	110	1101110	01101110
t	U+0074	116	1110100	01110100
.	U+002E	46	101110	00101110

Da alle Zeichen in diesem Beispiel im ASCII-Bereich liegen (0-127), wird jedes Zeichen mit genau einem Byte codiert. Der vollständige Text benötigt also 27 Bytes im UTF-8 Format.

Die komplette binäre Darstellung des Textes ist:

```
01001001 01101110 01100110 01101111 01110010 01101101 01100001 01110100
01101001 01101011 00100000 01101001 01110011 01110100 00100000 01101001
01101110 01110100 01100101 01110010 01100101 01110011 01110011 01100001
01101110 01110100 00101110
```

In hexadezimaler Schreibweise:

```
49 6E 66 6F 72 6D 61 74 69 6B 20 69 73 74 20 69 6E 74 65 72 65 73 73 61 6E 74 2E
```

Dieser String kann direkt in einem Computer gespeichert und verarbeitet werden. In Textdateien, bei der Übertragung im Internet oder in Datenbanken sind Strings immer in solchen binären Formaten gespeichert.

21.3.7 Vorteile der Unicode-Codierung:

1. **Universalität**: Alle Schriftsysteme und Sonderzeichen können dargestellt werden
2. **Effizienz**: Häufig verwendete Zeichen benötigen weniger Speicherplatz
3. **Kompatibilität**: Rückwärtskompatibel mit ASCII, was die Integration erleichtert

Die Umwandlung zwischen Textzeichen und ihrer binären Repräsentation wird in Computersystemen automatisch durch Codierungs- und Decodierungsprozesse abgewickelt, sodass Benutzer sich in der Regel nicht mit den Details befassen müssen.

22 Python Funktionen zum Umrechnen zwichen den Zahlensystemen

Python stellt Funktionen zur Verfügung, mit deren Hilfe Zahlen zwischen den verschiedenen Zahlensystemen umgerechnet werden können. Trotzdem sollen in diesem Notebook eigene Funktionen implementiert werden, mit deren Hilfe Zahlen zwischen dem Dezimal- und Binärsystem umgerechnet werden können.

Ausserdem sollen Funktionen implementiert werden, mit welchen Strings in ihre binäre Repräsentation umgewandelt werden können und umgekehrt.

22.1 Umrechnung von Binär- in Dezimalzahlen

```
def bin2dec(binary: str) -> int:
    # TODO: implementieren Sie hier die Funktion
    pass
```

22.2 Umrechnung von Dezimal- in Binärzahlen

```
def dec2bin(decimal: int) -> str:
    # TODO: implementieren Sie hier die Funktion
    pass
```

22.3 Binäre Repräsentation von Strings

Für die Darstellung eines Strings in binärer Form wird auf die dezimale Repräsentation des Unicode-Zeichens zurückgegriffen. Dazu stellt Python die Funktion `ord()` zur Verfügung. Diese Funktion gibt die dezimale Repräsentation eines Unicode-Zeichens zurück.

```
def str2bin(s: str) -> str:
    # TODO: implementieren Sie hier die Funktion
    pass
```

22.4 Umwandlung der binären Repräsentation in einen String

Um aus der dezimalen Repräsentation eines Unicode-Zeichens wieder den entsprechenden String zu erhalten, stellt Python die Funktion `chr()` zur Verfügung.

```
def bin2str(binary: str) -> str:
    # TODO: implementieren Sie hier die Funktion
    pass
```

23 Python Funktionen zum Umrechnen zwichen den Zahlensystemen

Python stellt Funktionen zur Verfügung, mit deren Hilfe Zahlen zwischen den verschiedenen Zahlensystemen umgerechnet werden können. Trotzdem sollen in diesem Notebook eigene Funktionen implementiert werden, mit deren Hilfe Zahlen zwischen dem Dezimal- und Binärsystem umgerechnet werden können.

Ausserdem sollen Funktionen implementiert werden, mit welchen Strings in ihre binäre Repräsentation umgewandelt werden können und umgekehrt.

23.1 Umrechnung von Binär- in Dezimalzahlen

```
def bin2dec(binary: str) -> int:
    decimal = 0
    length = len(binary)

    for i in range(1, length + 1):
        decimal += int(binary[-i]) * 2 ** (i-1)

    return decimal
```

23.2 Umrechnung von Dezimal- in Binärzahlen

```
def dec2bin(decimal: int) -> str:
    binary = ''
    while decimal > 0:
        binary = str(decimal % 2) + binary
        decimal = decimal // 2

    return binary
```

23.3 Binäre Repräsentation von Strings

Für die Darstellung eines Strings in binärer Form wird auf die dezimale Repräsentation des Unicode-Zeichens zurückgegriffen. Dazu stellt Python die Funktion `ord()` zur Verfügung. Diese Funktion gibt die dezimale Repräsentation eines Unicode-Zeichens zurück.

```
def str2bin(s: str) -> str:
    length_string = len(s)
    binary = ''
    for i in range(length_string):
        c = s[i]
        u10 = ord(c)
        b = dec2bin(u10)
        length_binary = len(b)
        if i % 5 == 0 and i != 0:
            binary += '\n'
        b = '0' * (8 - length_binary) + b
        binary += b + ' '
    return binary.strip()
```

23.4 Umwandlung der binären Repräsentation in einen String

Um aus der dezimalen Repräsentation eines Unicode-Zeichens wieder den entsprechenden String zu erhalten, stellt Python die Funktion `chr()` zur Verfügung.

```
def bin2str(binary: str) -> str:
    binary = binary.replace(' ', '')
    binary = binary.replace('\n', '')
    length = len(binary)
    s = ''
    for i in range(0, length, 8):
        b = binary[i:i+8]
        u10 = bin2dec(b)
        c = chr(u10)
        s += c
    return s
```

24 Base64-Codierung

Die Base64-Codierung ist ein Verfahren zur Umwandlung von Binärdaten in eine Zeichenkette, die nur aus lesbaren ASCII-Zeichen besteht. Diese Codierung wird häufig verwendet, um binäre Daten über textbasierte Systeme zu übertragen, die möglicherweise nicht mit Binärdaten umgehen können (z.B. E-Mails). Eine weitere Anwendung ist die Einbettung von Bildern in HTML- oder Markdown-Dokumente.
Untenstehende Octocat-Grafik ist ein Beispiel für eine Base64-codierte Grafik.

Figure 24.1: Base64 codierte Octocat

Sofern die Website in Chrome geöffnet ist, kann die Grafik mit einem Rechtsklick "untersucht" werden. Die "Untersuchung" zeigt den Quellcode der Seite an. Dort kann die Base64-codierte Grafik gefunden werden. Diese beginnt mit `` und setzt sich mit einer langen Zeichenkette fort. Diese Zeichenkette ist die Base64-Codierung der Grafik.

24.1 Prinzip der Base64-Codierung

Base64 wandelt 3 Bytes (24 Bits) Binärdaten in 4 druckbare ASCII-Zeichen um:

1. Die Binärdaten werden in 6-Bit-Blöcke aufgeteilt (2^6 = 64 mögliche Werte)
2. Jeder 6-Bit-Block wird in ein druckbares ASCII-Zeichen umgewandelt
3. Bei unvollständigen Blocks am Ende werden Füllzeichen (`=`) hinzugefügt

24.2 Base64-Zeichensatz

Der Base64-Zeichensatz besteht aus 64 Zeichen: - Die Großbuchstaben A-Z (26 Zeichen) - Die Kleinbuchstaben a-z (26 Zeichen) - Die Ziffern 0-9 (10 Zeichen) - Zwei zusätzliche Zeichen, meist '+' und '/' (2 Zeichen)

Diese 64 Zeichen repräsentieren die Werte 0-63 und können mit 6 Bits dargestellt werden.

24.3 Codierungsprozess

1. Die Binärdaten werden in Gruppen von 3 Bytes (24 Bits) aufgeteilt
2. Diese 24 Bits werden in vier 6-Bit-Blöcke umgewandelt
3. Jeder 6-Bit-Wert wird als Index für den Base64-Zeichensatz verwendet

Wenn die Anzahl der zu codierenden Bytes nicht durch 3 teilbar ist: - Bei einem übrig bleibenden Byte: Auffüllen mit vier Nullbits, Codierung ergibt zwei Zeichen und zwei '='-Zeichen - Bei zwei übrig bleibenden Bytes: Auffüllen mit zwei Nullbits, Codierung ergibt drei Zeichen und ein '='-Zeichen

24.4 Beispiele:

24.4.1 Beispiel 1: Codierung von "KBW"

Buchstabe	K								B								W							
Dezimal	75								66								87							
Binär	0	1	0	0	1	0	1	1	0	1	0	0	0	0	1	0	0	1	0	1	0	1	1	1
Dezimal	18								52								9							
Base64	S								0								J							

Figure 24.2: Base64 Beispiel

24.4.2 Beispiel 2: Codierung von "Hallo"

Betrachten wir das Wort "Hallo":

1. ASCII-Werte: H=72, a=97, l=108, l=108, o=111
2. Binär: 01001000 01100001 01101100 01101100 01101111
3. In 6-Bit-Gruppen aufteilen: 010010 000110 000101 101100 011011 000110 1111
4. Da die letzte Gruppe nur 4 Bits hat, wird sie mit zwei Nullen aufgefüllt: 010010 000110 000101 101100 011011 000110 111100
5. Die 6-Bit-Werte sind: 18, 6, 5, 44, 27, 6, 60
6. Diese werden in Base64-Zeichen umgewandelt: S G F s b G 8 =

Das Ergebnis "SGFsbG8=" ist die Base64-Codierung von "Hallo".

24.5 Anwendungen der Base64-Codierung

- E-Mail-Anhänge (MIME)
- Datenkodierung in URLs
- Einbettung von Bildern in HTML/CSS (Data-URLs)
- Übertragung binärer Daten in JSON
- Speicherung von Binärdaten in XML

Base64 vergrößert die Datenmenge um etwa 33% (da 3 Bytes in 4 Zeichen umgewandelt werden), bietet aber den Vorteil, dass die codierten Daten in jedem Textformat sicher übertragen werden können.

25 Lernziele für die Prüfung vom 26. März 2025

Ich erwarte, dass Sie in der Lage sind,

- die folgenden Datenstrukturen zu beschreiben und ihre Implementation in Python grundsätzlich zu erklären:
  - Linked List
  - Stack
  - Queue sowie
  - Binary Search Tree;
- Zahlen vom Dezimalsystem ins Binärsystem umzurechnen und umgekehrt sowie
- zu erklären, wozu die Base64-Codierung verwendet wird und wie sie funktioniert.

And der Prüfung dürfen Sie eine handschriftliche Zusammenfassung im Umfang von einer Seite A4 verwenden. Das einzige, was nicht auf der Zusammenfassung stehen darf, sind die für das Arbeiten mit Jupyter Notebooks notwendigen Einrichtungsschritte.

Part VI

Künstliche Intelligenz

26 The Social Dilemma

The Social Dilemma ist ein Film von Nextflix.

https://www.netflix.com/title/81254224

27 Was ist eine Künstliche Intelligenz?

Sie alle kennen heute **generative K.I.**-Tools. Wir werden zuerst K.I. anschauen, die keinen Inhalt generieren, sondern Entscheidungen fällen. Dies ist zu Beginn einfacher nachzuvollziehen, was ein Modell ist.

Nehmen Sie Kopfhörer nach vorne und absolvieren Sie selbstständig diesen Kurs: https://studio.code.org/courses/oceans/units/1/lessons/1/levels/1

28 Maschinelles Lernen: Punkt- und Linienklassifikatoren

Besuchen Sie die Seite https://ricardoheinzmann.com/ml-class/level-selection.

Spielen Sie in folgender Reihenfolge:

1. Linien-Klassifizieren mit 2D-Daten
2. Punkt-Klassifizieren mit 2D-Daten
3. Punkt-Klassifizieren mit 3D-Daten

29 Maschinelles Lernen: Gewinnstrategie

Spielen Sie das Spiel und entdecken Sie den Lernalgorithmus.

Bedienung:

- Klicken Sie auf einen blauen Spielstein, nachher auf das gefärbte Feld eins weiter unten.

https://aieducation.computatrum.ch/

Nach der “Übung 1: Lernalgorithmus in Aktion - Einfach” dürfen Sie selber entscheiden, ob Sie die selbe Übung mit einem *Mittel* schweren Spiel machen möchten, oder zur Übung 2 gehen.

Part VII

Anleitungen

30 Installation von Python

Hier ist eine einfache Schritt-für-Schritt-Anleitung zur Installation von Python:

30.1 Python herunterladen und installieren

1. Öffnen Sie Ihren Webbrowser und gehen Sie zur offiziellen Python-Website: www.python.org
2. Klicken Sie auf der Startseite auf den Button "Downloads"
3. Suchen Sie die neueste Python-Version für Windows (Juli 24 Python 3.12.x)
4. Klicken Sie auf den Download-Button für die 64-bit-Version (falls nicht automatisch ausgewählt)
5. Nachdem der Download abgeschlossen ist, öffnen Sie die heruntergeladene Datei (sie endet auf .exe)
6. Im Installationsfenster setzen Sie ein Häkchen bei "Add Python x.x to PATH" (**sehr wichtig!**)
7. Klicken Sie auf "Install Now" und warten Sie, bis die Installation abgeschlossen ist
8. Klicken Sie am Ende auf "Close"

30.2 Installation überprüfen

1. Öffnen Sie die Windows-Eingabeaufforderung (`cmd`/Terminal):
   - Drücken Sie die Windows-Taste + R
   - Geben Sie "`cmd`" ein und drücken Sie Enter
2. In der Eingabeaufforderung/Terminal (dem neu aufgegangen Fenster) tippen Sie:

   ```
   python --version
   ```

   und drücken Sie Enter
3. Wenn die Installation erfolgreich war, sehen Sie die installierte Python-Version (stimmt mit dem Überein, was auf dem Download-Button gestanden hat)

30.3 Erste Schritte mit Python

1. In der Eingabeaufforderung tippen Sie `python` und drücken Enter
2. Sie sehen nun den Python-Prompt (»>). Der Prompt ist eine Eingabeaufforderung. Hier können Sie direkt Python-Code eingeben, z.B.:

   ```
   print("Hallo, Welt!")
   ```

3. Drücken Sie Enter, um den Code auszuführen
4. Um die Python-Umgebung zu verlassen, geben Sie `exit()` ein und drücken Enter

Glückwunsch! Sie haben Python erfolgreich installiert und Ihr erstes Programm ausgeführt.

Mit dieser Anleitung sollten Sie Python problemlos installieren und erste Schritte in der Programmierung machen können. Viel Erfolg beim Lernen!

Dieser Text wurde mit Hilfe von Perplexity.ai erstellt.

31 Anleitung zum Arbeiten mit Python Virtual Environments

Eine Python Virtual Environment (venv) ist eine geschützte Umgebung, in der die Installation zusätzlicher Module zu keinen Konflikten mit dem Gesamtsystem führen. Aus diesem Grund ist es sinnvoll, für jedes neue Python Projekt eine solche venv anzulegen.

Hier eine Checkliste für das Arbeiten in einer venv:

Als Vorfrage ist zu prüfen, ob bereits eine venv angelegt ist.

31.1 Arbeiten mit einer bereits bestehenden venv

1. Aktivieren der venv:

   ```
   C:\Pfad\zum\aktuellen\Ordner>venv\Scripts\activate
   ```

   Dass die venv aktiviert worden ist, erkennt man daran, dass das Terminal folgenden Prompt aufweist:

   ```
   (venv) C:\Pfad\zum\aktuellen\Ordner>
   ```

2. Jupyter Server starten:

   ```
   (venv) C:\Pfad\zum\aktuellen\Ordner>jupyter notebook
   ```

   Das Terminal, in welchem der Jupyter Server gestartet wurde, darf **nicht** geschlossen werden.

3. Arbeiten am Projekt

4. Jupyter Server deaktivieren.

   Der Jupyter Server kann im Terminal, in welchem er gestartet wurde, mit der Tastenkombination [Ctrl] + [C] angehalten werden.

5. venv deaktivieren

   ```
   (venv) C:\Pfad\zum\aktuellen\Ordner>deactivate
   ```

31.2 Einrichten einer neuen venv

1. Projektordner erstellen
2. venv anlegen:

   ```
   C:\Pfad\zum\aktuellen\Ordner>python -m venv venv
   ```

3. Aktivieren der venv:

   ```
   C:\Pfad\zum\aktuellen\Ordner>venv\Scripts\activate
   ```

   Dass die venv aktiviert worden ist, erkennt man daran, dass das Terminal folgenden Prompt aufweist:

   ```
   (venv) C:\Pfad\zum\aktuellen\Ordner>
   ```

4. Installieren der erforderlichen Pakete:

   ```
   (venv) C:\Pfad\zum\aktuellen\Ordner>python -m pip install jupyter (und allfällige and
   ```

5. Jupyter Server starten:

   ```
   (venv) C:\Pfad\zum\aktuellen\Ordner>jupyter notebook
   ```

   Das Terminal, in welchem der Jupyter Server gestartet wurde, darf **nicht** geschlossen werden.

6. Arbeiten am Projekt
7. Jupyter Server deaktivieren.

   Der Jupyter Server kann im Terminal, in welchem er gestartet wurde, mit der Tastenkombination [Ctrl] + [C] angehalten werden.

8. venv deaktivieren

   ```
   (venv) C:\Pfad\zum\aktuellen\Ordner>deactivate
   ```

32 Jupyter Notebook

In einem Jupyter Notebook kann Text mit ausführbarem Programmcode kombiniert werden. Wir werden Jupyter Notebooks für die Anwendungsübungen im Unterricht verwenden.

32.1 Installationsanleitung

Hier ist eine Schritt-für-Schritt-Anleitung zur Installation von Jupyter Notebooks:

1. Öffnen Sie die Windows-Eingabeaufforderung (cmd).
2. Wechseln Sie mit

   ```
   cd \Pfad\zum\Arbeitsverzeichnis
   ```

   in Ihr Arbeitsverzeichnis.
3. Starten Sie eine Python Virtual Environment

   ```
   python -m venv venv

   venv\Scripts\activate
   ```

4. Geben Sie folgenden Befehl ein und drücken Sie Enter:

   ```
   python -m pip install jupyter
   ```

   Die Installation braucht etwas Zeit. Haben Sie Geduld mit Ihrem Computer.

   Falls pip aus irgendeinem Grund nicht installiert wurde oder nicht funktioniert, können Sie es manuell installieren, indem Sie den folgenden Befehl in der Eingabeaufforderung ausführen:

   ```
   python -m ensurepip --upgrade
   ```

   Dieser Befehl stellt sicher, dass pip installiert und auf dem neuesten Stand ist.

32.2 Jupyter Notebook starten

1. Öffnen Sie erneut die Eingabeaufforderung/das Terminal.
2. Geben Sie ein:

   ```
   jupyter notebook
   ```

3. Drücken Sie Enter. Ihr Standardbrowser öffnet sich automatisch mit der Jupyter Notebook-Oberfläche.

32.3 Ihr erstes Notebook erstellen

1. Klicken Sie in der Jupyter-Oberfläche auf "New" und wählen Sie "Notebook".
2. Wählen Sie im Pop-up Fenster "Python 3 (ipykernel)" und bestätigen Sie mit select.
3. Ein neues Notebook öffnet sich in einem neuen Tab. Der Tab trägt den Namen "Untitled".
4. Geben Sie in die erste Zelle etwas Python-Code ein, z.B.:

   ```
   print("Hallo Jupyter!")
   ```

5. Drücken Sie Shift+Enter, um die Zelle auszuführen.

32.4 Tipps zur Verwendung

- Verwenden Sie "Code"-Zellen für Python-Code und "Markdown"-Zellen für Text.
- Speichern Sie Ihr Notebook regelmäßig mit dem Disketten-Symbol oder Strg+S.
- Schließen Sie das Browserfenster und geben Sie in der Eingabeaufforderung/im Terminal Strg+C ein, um Jupyter zu beenden.

Glückwunsch! Sie haben nun Jupyter Notebooks installiert und können damit arbeiten. Erkunden Sie weitere Funktionen und Möglichkeiten, während Sie sich mit dem Tool vertraut machen.

32.5 Bearbeiten von Jupyter Notebooks in VS Code

1. Jupyter-Erweiterung installieren:
   - Suchen Sie im Erweiterungen-Bereich nach "Jupyter"
   - Installieren Sie die Erweiterung "Jupyter" von Microsoft
2. Neues Jupyter Notebook erstellen:

- Drücken Sie Strg+Shift+P, um die Befehlspalette zu öffnen
- Geben Sie "Jupyter: Create New Jupyter Notebook" ein und wählen Sie diesen Befehl aus
- Ein neues Notebook wird geöffnet

3. Mit dem Notebook arbeiten:

   - Um eine neue Zelle hinzuzufügen, klicken Sie auf den "+ Code" Button oben im Notebook
   - Geben Sie Ihren Python-Code in die Zelle ein
   - Um den Code auszuführen, klicken Sie auf den Play-Button links neben der Zelle oder drücken Sie Shift+Enter
   - Die Ausgabe erscheint direkt unter der Zelle

4. Notebook speichern:

   - Gehen Sie auf Datei > Speichern unter
   - Wählen Sie einen Speicherort und geben Sie einen Namen mit der Endung .ipynb ein

5. Bestehendes Notebook öffnen:

   - Gehen Sie auf Datei > Öffnen
   - Navigieren Sie zu Ihrem .ipynb-File und öffnen Sie es

Tipps: - Sie können zwischen Code- und Markdown-Zellen wechseln, indem Sie auf den Zellentyp oben im Notebook klicken - Nutzen Sie die Befehlspalette (Strg+Shift+P), um weitere Jupyter-bezogene Befehle zu finden

Dieser Text wurde mit Hilfe von Perplexity.ai erstellt.

33 Plain Text Writing

In diesem Abschnitt geht es um einen Aspekt der digitalen Nachhaltigkeit. Digitale Nachhaltigkeit wird hier so verstanden, dass digitale Produkte unabhängig von konkreten Herstellern dargestellt werden können.
Um zu erklären, wo das Problem liegt, muss zuerst zwischen den zwei Dateitypen Textdateien und Binärdateien unterscheiden werden.

33.1 Text- und Binärdateien

Textdateien speichern ihren Inhalt als reinen Text. Formatierung in solchen Dateien erfolgen durch Steuerzeichen (vgl. z.B. ASCII-Tabelle, dort insbesondere die Beispiele der Zeichen 9 für einen horizontalen Tabulatoren oder Zeichen 10 für den Zeilenumbruch) und durch Konventionen (vgl. z.B. die Formatierungen in Markdown oder LaTeX). Entsprechend können sie in beliebigen Texteditoren gelesen werden.

Im Gegensatz dazu sind Binärdateien für ihre Erstellung, Bearbeitung und Darstellung grundsätzlich an Programme gebunden, welche ihren Standard lesen können. Ihre Inhalte werden als eine Folge von Bytes gespeichert. Ein Beispiel für eine Binärdatei ist ein in MS Word erstelltes Dokument.

Diese Unterscheidung zeigt, dass Textdateien grundsätzlich auf jedem funktionierenden Computer mit einem Texteditor angezeigt werden können. Im Gegensatz dazu erfordern Binärdateien in der Regel spezielle Programme, die das jeweilige Dateiformat interpretieren können, was zu einer gewissen Abhängigkeit von entsprechender Software führen kann.

Weiterführende Überlegungen zum Verfassen von Texten in Plain Text können den hier angeführten Texten entnommen werden.

33.2 Plain Text Formate für die Textredaktion

Die einfachste Form eines Textfiles ist eine `.txt` Datei. Dabei handelt es sich in der Regel um reinen Text ohne Formatierungen. Das Bedürfnis nach formatierten Textdarstellungen hat zu verschiedenen Dateiformaten geführt. Dazu gehören zum Beispiel HTML-Dateien (`.html` oder `.htm`: **H**yper **T**ext **M**arkup **L**anguage). Dieses Format wurde ursprünglich unter anderem auch dazu geschaffen, Text formatiert am Bildschirm darzustellen. Ein anderes Beispiel ist LaTeX. Auch LaTeX ist eine sogenannte Auszeichnungssprache (Markup Language). Allerdings ist das Zielmedium von LaTeX weniger der Bildschirm sondern eher

der Druck, heute vor allem das Portable Document Format (PDF), welches sich zum Quasi-Standard entwickelt hat.

Allerdings ist der Umfang der Formatierungsmöglichkeiten sowohl in HTML wie auch in LaTeX so umfangreich, dass die Hürde für deren Verwendung relativ hoch ist.

Vor diesem Hintergrund ist eine eher minimalistische Auszeichnungssprache, Markdown (`.md`) geschaffen worden. Diese stellt alle wesentlichen Formatierungsmöglichkeiten, wie zum Beispiel hierarchische Überschriften oder das setzen von kursivem Text, direkt zur Verfügung. Dort wo die Formatierungsmöglichkeiten den Bedürfnissen nicht genügen, kann in vielen Fällen auf HTML oder LaTeX Auszeichnungen zurückgegriffen werden.

Eine Übersicht über die Formatierungsbefehle in Markdown findet sich auf der Website Markdown Cheat Sheet. Wichtige Formatierungen in LaTeX (mathematische Formeln) werden auf der Website Wikibooks LaTeX/Mathematics erklärt.

34 Konvertierung von Markdown in PDF

Markdown hat den klaren Vorzug, in jedem Texteditor gelesen werden zu können. Leider ist reines Markdown nicht sehr ästhetisch. Zudem müssen die meisten Texte zu Handen des Empfängers in ein PDF konvertiert werden.

34.1 Vorbereitung der Konvertierung

Für diese Konvertierung kann das Programm Pandoc verwendet werden. Pandoc kann Markdown in unzählige Formate Konvertieren. Die genaue Vorgehensweise wird im Folgenden Schritt für Schritt erläutert.

1. Pandoc installieren

   Das Installationsprogramm für Pandoc kann auf der Website von Pandoc heruntergeladen werden. Dazu ist dem Link "Download the latest Installer" zu folgen. Für Windows ist die Datei mit der Endung `.msi` herunterzuladen.

   Das heruntergeladene Installationsprogramm kann anschliessend mit einem Doppelklick für die Installation ausgeführt werden.

2. LaTeX installieren

   Damit Pandoc ein Markdown Dokument in ein PDF Konvertieren kann, braucht es auf dem Rechner eine Installation von LaTeX. LaTeX gibt es für Windows in verschiedenen Varianten. Die einfachste Installation erfolgt in der Variante von MiKTeX. Das entsprechende Installationspaket kann von der Website von MiKTeX heruntergeladen werden.

   Das heruntergeladene Installationsprogramm kann anschliessend mit einem Doppelklick für die Installation ausgeführt werden.

3. Pandoc verwenden

   Pandoc ist ein Programm für die Kommandozeile. Das heisst, es gibt keine grafische Oberfläche für die Verwendung von Pandoc. Am einfachsten kann Pandoc verwendet werden, wenn im Ordner, in dem sich das zu konvertierende File befindet, ein Terminal geöffnet wird.

   Die Konvertierung erfolgt mit folgendem Befehl:

   ```
   pandoc -o output.pdf input.md
   ```

Output.pdf und input.md muss an die eigenen Dateinamen angepasst werden. Alternativ können auch andere Dateiformate, wie `.docx` oder `.html` als Zielformate gewählt werden. Für eine abschliessende Liste aller möglichen Zielformate sei auf die Website von Pandoc verwiesen.

34.2 Spezifische Formatierung der Ausgabe

Die oben dargestellte Version führt zu einem sauber strukturierten PDF. Allerdings fehlen wesentliche Dokumentteile wie Deckblatt oder Inhaltsverzeichnis.

Für diese weiterführenden Formatierungen werden dem Dokument die erforderlichen Informationen entweder in einem speziellen Header vorangestellt oder in einem separaten Dokument dem Befehl für die Konvertierung übergeben. Hier soll die zweite Variante dargestellt werden.

Das Dokument mit den Informationen für die Formatierung ist ein sogenanntes YAML-Dokument `.yaml`. YAML steht dabei für **Y**et **A**nother **M**arkup **L**language. Das Dokument kann einen beliebigen Namen haben. Hier im Beispiel wird es `format.yaml` genannt. Das folgende Listing zeigt eine Formatierung mit dem Namen des Authors, dem Titel und weiteren Angaben. Was die einzelnen Einträge bedeuten, ist weitestgehend selbsterklärend.

```yaml
from: markdown                      # Ausgangsformat
to: pdf                             # Zielformat
standalone: true                    # Erzwingt eigenständiges Dokument
pdf-engine: xelatex                 # Wählt eine spezifische pdf-engine
output-file: output.pdf             # Name des Ausgabedokuments
metadata:                           # Dokumentspezifische Angaben
  title: "Anleitung"                # Titel
  author: "Jacques Mock Schindler"  # Verfasser
  lang: de-CH                       # Sprache des Dokuments
  fontsize: 11pt                    # Schriftgrösse
  geometry: left=2.5cm, right=3cm   # Seitenränder
  date: "9. Januar 2026"            # Datum
  toc: true                         # Erstellt ein Inhaltsverzeichnis
  number-sections: true             # Nummeriert die Titel
```

Die Einrückung ist für die Syntax im YAML Dokument wichtig. Sie beträgt zwei Leerzeichen.

Wenn die Datei `format.yaml` das Format des PDF steuern soll, lautet der Befehl für die Konvertierung des Markdown Dokuments in ein PDF

```
pandoc --defaults=format.yaml input.md
```

Falls der Name des neu erstellten PDF Dokuments bei jedem Befehlsaufruf individuell festgelegt werden soll, kann die Zeile `output-file: output.pdf` in der YAML Datei auch weggelassen werden. Dann lautet der Befehl für die Konvertierung

```
pandoc --defaults=format.yaml -o output.pdf input.md
```

Falls für komplexe Formatierungen auf externe LaTeX Pakete abgestellt werden muss, werden diese über ein separates `.tex` Dokument geladen. Auch dieses Dokument kann grundsätzlich beliebig benannt werden. Für das Beispiel hier wird die Datei als `header.tex` bezeichnet. Der Inhalt dieser Datei ist folgendermassen zu formatieren:

```
\usepackage{caption}
```

Der Befehl für die Konvertierung lautet jetzt

```
pandox --defaults=format.yaml -H header.tex input.md
```

Die Website CTAN Comprehensive TeX Archive Network gibt Auskunft darüber, was für LaTeX Pakete zur Verfügung stehen.

Part VIII

Alte Programme

35 Programm im Frühlingssemester 2025

Datum	Thema
19.02.2025	Queues Implementieren in Python
26.02.2025	BST Implementieren in Python
05.03.2025	BST Implementieren in Python
12.03.2025	Binärsystem
19.03.2025	Base64 Codierung
26.03.2025	Test (Datenstrukturen & Datencodierung)
02.04.2025	Datenbanken
09.04.2025	Datenbanken
16.04.2025	Datenbanken
07.05.2025	Datenbanken
14.05.2025	Datenbanken: Übungen
21.05.2025	Test (Datenbanken)
28.05.2025	Datenvisualisierung mit Python
04.06.2025	Datenvisualisierung mit Python
11.06.2025	Datenvisualisierung mit Python
18.06.2025	Datenvisualisierung mit Python
25.06.2025	Datenvisualisierung mit Python
02.07.2025	Datenvisualisierung mit Python
09.07.2025	Datenvisualisierung mit Python