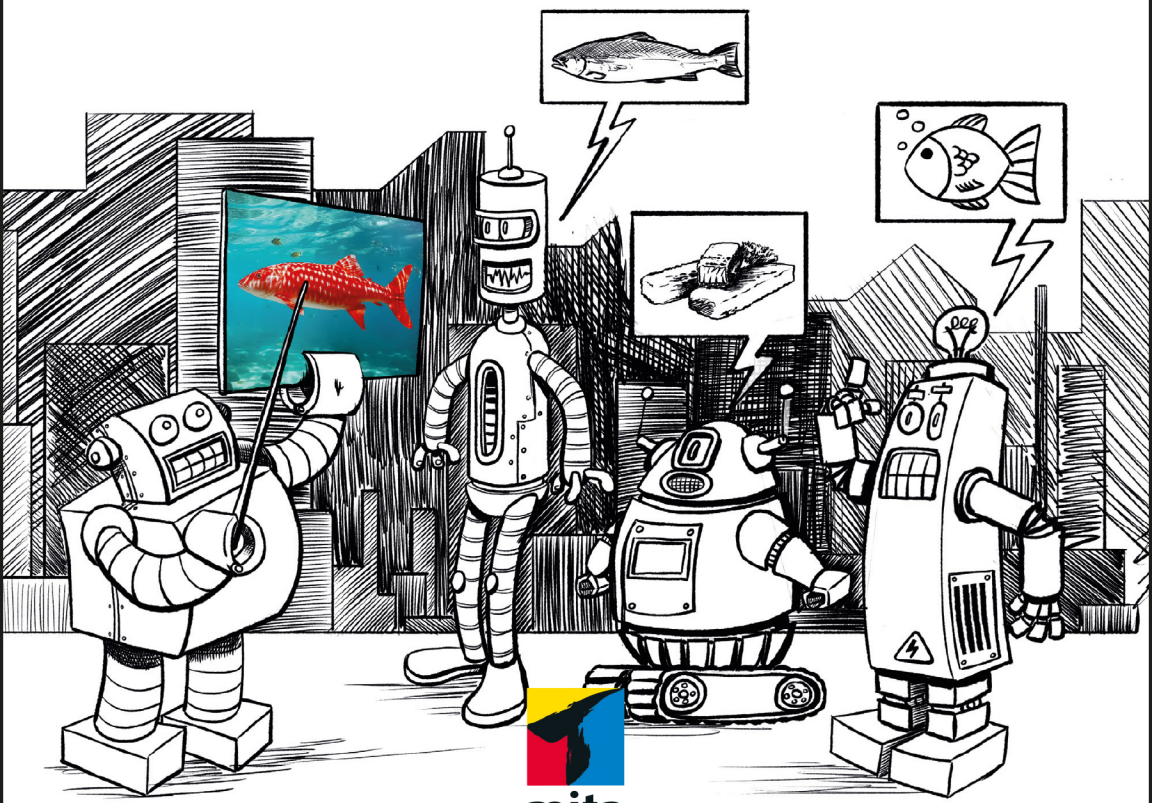
MICHAEL WEIGEND

KÜNSTLICHE INTELLIGENZ

KAPIEREN & PROGRAMMIEREN

VISUELL LERNEN & VERSTEHEN

mit Illustrationen und Projekten zum Experimentieren



mitp

Inhaltsverzeichnis

Einleitung

Künstliche Intelligenz (KI) und Machine Learning sind Themen, über die heute viel geredet wird. Die Grundlagen dieser neuen Technik verstehen nur wenige. Sie sind auch nicht ganz einfach. Um eine Vorstellung davon zu bekommen, wie KI funktioniert, musst du dich auf einige neue Denkweisen einlassen.

Dieses Buch folgt der pädagogischen Idee des Konstruktionismus: Baue etwas und lerne dabei! Von Anfang an wirst du ermuntert, selbst zu programmieren. Du kannst die Programmtexte aus dem Buch abschreiben. Das geht, denn sie sind nicht sehr lang. Du kannst die Programme aber auch von der Webpräsenz des mitp-Verlags herunterladen. Besuche die Adresse `https://www.mitp.de/0652`, wähle die Registerkarte DOWNLOADS und klicke auf den Link PROGRAMMBEISPIELE. Nach dem Entpacken hast du für jedes Kapitel ein Verzeichnis mit den Programmen, die im Buch erklärt werden. Im Downloadbereich findest du auch die Lösungen zu allen Aufgaben in diesem Buch.

Ein Programmbeispiel ist ein guter Ausgangspunkt für eine eigene Entwicklung. Es funktioniert, so wie es ist. Du kannst es genau kopieren, aber du kannst es auch abwandeln. Am besten änderst du so viel wie möglich und machst so das Beispiel zu deinem eigenen Projekt. Ändere die Variablennamen. Füge weitere Features in dein Programm ein. Schreibe neue Texte für die Ausgaben des Programms. Experimentiere! Mache Fehler! Am meisten lernt man aus den eigenen Fehlern. Wenn deine eigene Programmversion nicht laufen will, hast du immer noch das Beispiel. Achte auf die Unterschiede zu deinem Projekt. Dann wirst du sicher die Stelle finden, an der es hakt.

Dieses Buch ist gleichzeitig eine Einführung in Python. Es werden keine Programmiervorkenntnisse erwartet. Wenn du schon programmieren kannst, werden dir vielleicht einige Besonderheiten auffallen. Ich habe versucht, mit möglichst wenigen Programmierkonzepten auszukommen und viele Sachen weggelassen, die in anderen Programmierbüchern vorkommen. Es gibt z.B. keine objektorientierte Programmierung. Wir verwenden zwar Objekte, aber wir werden keine Klassen definieren. Andererseits lernst du einige spezielle Module kennen, die nicht unbedingt zum Standard gehören. Wir werden z.B. mit dem sehr schnellen Modul NumPy arbeiten, Diagramme von Funktionen mit Matplotlib erstellen, und mit PIL und OpenCV Bilder bearbeiten. Keine Angst: Das meiste ist einfache Python-Programmierung.

Alle Projekte drehen sich um künstliche Intelligenz, maschinelles Lernen und neuronale Netze. Du entwickelst einfache Chatbots, lernfähige Programme, die mit Hilfe symbolischer oder subsymbolischer KI Dinge erkennen können, und schließlich Programme, die Livebilder deiner Kamera auswerten und Gesten erkennen können. Zum Training deiner selbst programmierten neuronalen Netze verwendest du freie Datensätze aus dem Internet und Bildmaterial, das du eigenhändig erstellt hast.

Das Ziel ist es, durch aktives Programmieren ein Gefühl für die Grundprinzipien und Möglichkeiten der neuen Technik zu gewinnen. Die mathematischen Passagen in diesem Buch sind möglicherweise echte Herausforderungen. Aber oft ist es so, dass man die Formeln besser versteht, wenn man sieht, dass sie wirklich funktionieren.

Viel Erfolg bei deinem Ausflug in die Welt der künstlichen Intelligenz!

Michael Weigend

Kapitel 1

Denkende Maschinen

Das erste Kapitel gibt dir einen leicht verständlichen Einstieg in die Welt der KI. Du gewinnst einen Überblick über Formen des Machine Learnings und erfährst, welche Rolle künstliche neuronale Netze hierbei spielen. Ganz nebenbei lernst du die Grundlagen der Programmiersprache Python.

1.1 Was ist künstliche Intelligenz?

Der Begriff »Künstliche Intelligenz« (KI) ist gar nicht so neu, wie man vielleicht glauben mag. Tatsächlich hat man die ersten funktionierenden Computer in den 1960er-Jahren gerne als »denkende Maschinen« oder »Elektronengehirne« bezeichnet. Heute würde man diese Vorstellung belächeln. Die damaligen Computer konnten zwar schon große Datenmengen zur Buchhaltung und Verwaltung verarbeiten, aber sie konnten nicht im Entferntesten selbstständig denken wie ein Mensch. Inzwischen ist viel Zeit vergangen. Heute gibt es tatsächlich digitale Systeme, die Autos steuern, Musik komponieren und Aufsätze schreiben können. Und es gibt ernst gemeinte Warnungen, dass künstliche Intelligenzen den Menschen überflügeln und ihm gefährlich werden könnten.

»Künstliche Intelligenz« umfasst ein weites Gebiet der Digitaltechnik. Man unterscheidet grob zwischen symbolischer KI und subsymbolischer KI:

- Bei **symbolischer KI** wird das intelligente Verhalten durch klare Regeln bestimmt, die jeder Mensch nachvollziehen kann. Beispiele sind konventionelle Chatbots, mit denen man einfache Gespräche führen kann, und Programme, die auf der Grundlage von vorgegebenen Merkmalen Objekte erkennen können.
- Bei **subsymbolischer KI** erlernt das System durch viele Beobachtungen intelligente Verhaltensweisen, etwa die Fähigkeit, Buchstaben zu erkennen. Jedoch ist das Wissen, das das Verhalten bestimmt, für Menschen nicht nachvollziehbar. Das System ist wie eine »Black Box«. Das Innere bleibt verborgen und rätselhaft. Diese Art von Intelligenzerwerb nennt man »Machine Learning« oder auf Deutsch »maschinelles Lernen«.

In diesem Buch geht es vor allem um die Programmierung künstlicher neuronaler Netze. Sie gehören zum zweiten Typ, der subsymbolischen KI. Um besser zu verstehen, was den Unterschied ausmacht, werden wir uns zunächst mit zwei Bei-

spielen symbolischer KI beschäftigen. Du wirst in diesem Kapitel einen Chatbot und ein lernfähiges Programm, das Personen erkennen kann, entwickeln.

1.2 Chatbots

Im Jahr 1950 beschrieb der englische Wissenschaftler Alan Turing in seinem Artikel »Computing Machinery and Intelligence« eine Methode, mit der man – seiner Meinung nach – beweisen kann, ob eine Maschine intelligent ist und denken kann wie ein Mensch. Er nannte sie »imitation game«; inzwischen ist sie als »Turing-Test« in die Geschichte eingegangen. Die Grundidee des Turing-Tests ist, dass sich Intelligenz im Gespräch zeigt. Ein Mensch (der Befrager) unterhält sich über geschriebenen Text mit zwei Gesprächspartnern A und B. Der eine Partner ist ein Computer, der andere ein Mensch. Wenn der Befrager aufgrund der Antworten seiner Gesprächspartner Mensch und Computer nicht unterscheiden kann, zeigt der Computer Intelligenz.

1966 entwickelte der deutsch-amerikanische Informatiker Joseph Weizenbaum am MIT (Massachusetts Institute of Technology) in den USA das erste Computerprogramm, das in Ansätzen den Turing-Test bestehen konnte. Das Programm heißt Eliza und wurde rasch weltberühmt. Eliza verhält sich wie eine Gesprächstherapeutin. Du kannst dich mit ihr unterhalten und sie versucht, dich zum Reden zu bringen.

Im Internet findest du einige Online-Versionen von Eliza, zum Beispiel auf der Webpräsenz von Medical Artificial Intelligence, `http://www.med-ai.com/models/eliza.html.de`.

Ein Dialog mit Eliza kann z.B. so verlaufen:

```
Eliza: Hallo, ich bin Eliza. Was hast du auf dem Herzen?
Du: Ich hasse meinen Job.
Eliza: Hass kann Wertvolles zerstören.
Du: Außerdem schlafe ich schlecht.
Eliza: Kannst du mir das Problem näher erklären?
...
```

Eliza funktioniert so: Bei jeder Benutzereingabe prüft das Programm, ob bestimmte Wörter wie z.B. *hass* vorkommen, und gibt dann eine Antwort, die irgendwie dazu passt. So hat man das Gefühl, von Eliza verstanden zu werden. Im folgenden Projekt kannst du diese Idee selbst umsetzen (Abschnitt 1.4). Bevor es an das Programmieren geht, musst du allerdings noch einige Dinge vorbereiten.

1.3 Vorbereitung: Python installieren

Für dieses Projekt benötigst du keinerlei Vorkenntnisse, aber auf deinem Computer muss die Programmiersprache Python installiert sein. Der Kasten gibt dir einige Hinweise, was zu tun ist.

Python installieren

Die Programmiersprache Python ist leicht zu erlernen und dennoch sehr mächtig. Viele Apps, die du aus deinem Alltag kennst, sind in Python geschrieben, z.B. Routenplaner, Gesichtserkennung oder Wettervorhersagen. Python ist kostenlos und kann einfach von der Webpräsenz der Python Software Foundation heruntergeladen werden. Besuche die Webseite `https://www.python.org/`, klicke auf DOWNLOADS, und wähle die aktuelle Python-Version, die zu deinem Betriebssystem passt.

Unter Microsoft Windows lädst du eine ausführbare Datei (Name endet auf `.exe`) herunter, die du durch Doppelklick startest.

Auf einem Mac läuft die Installation genauso, mit dem kleinen Unterschied, dass der Name der heruntergeladenen Datei auf `.pkg` endet.

Auf Linux-Rechnern ist Python meist schon vorinstalliert. Die neuste Version kannst du auf den meisten Linux-Systemen mit folgendem Befehl installieren:

```
sudo apt-get install python3
```

Wenn du Python installiert hast, befindet sich auf deinem Computer neben der eigentlichen Programmiersprache auch eine Entwicklungsumgebung namens IDLE. Die Abkürzung steht für *Integrated Development and Learning Environment*. Mit einer Entwicklungsumgebung kann man Programmtexte erstellen und testen. Außer IDLE gibt es auch viele andere Entwicklungsumgebungen für Python. Dieses Buch bezieht sich aber ausschließlich auf IDLE.

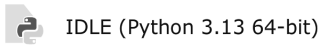
Projektordner erstellen

Richte mit dem Dateimanager (bei Windows also im Explorer) einen Projektordner ein, in dem du alle deine Programme zu diesem Buch speicherst. Dieser Ordner könnte z.B. `Python-Programme` heißen. Ich empfehle dir, in diesem übergeordneten Verzeichnis für jedes Kapitel des Buchs ein eigenes Verzeichnis anzulegen (`Kapitel_1`, `Kapitel_2`, ...). Verwende besser nicht die Ordner mit den fertigen Programmbeispielen, die du von der Webseite des mitp-Verlags heruntergeladen hast. Die Originaldateien bleiben dann unverändert. Das kann vor allem bei der Suche nach Fehlern in den eigenen Programmversionen helfen.

Wenn dein eigenes Programm nicht läuft, kannst du in der Downloadversion nachsehen, was da anders ist.

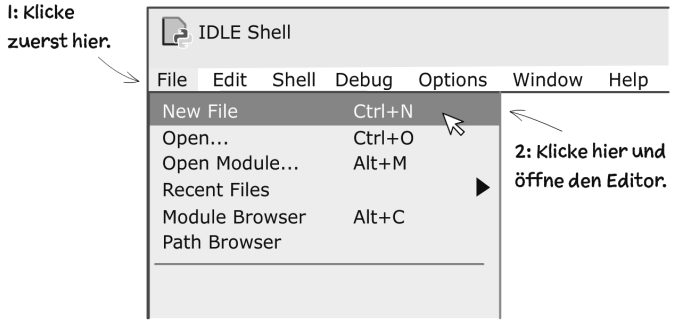
Den IDLE-Programmeditor starten

Starte die Entwicklungsumgebung IDLE. Unter Windows gibst du in das Suchfeld am unteren Bildschirmrand `idle` ein und doppelklickst dann das Programmicon.

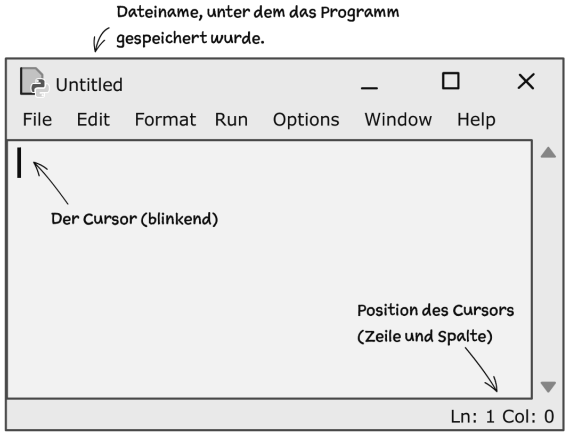


Auf dem Mac öffnest du ein Spotlight-Suchfenster, gibst `idle` ein und wählst dann die IDLE-App. Unter Linux gibst du auf der Kommandozeile den Befehl `idle3` ein.

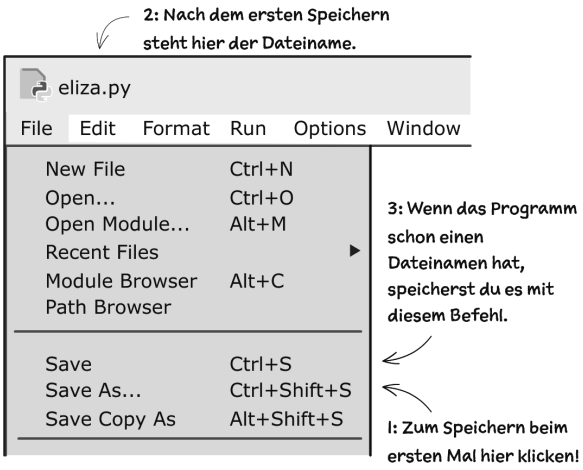
Es öffnet sich das Fenster der IDLE-Shell. Klicke in der Menüleiste oben links auf FILE. Klicke dann im Pulldown-Menü auf den Befehl NEW FILE.



Nun öffnet sich das Editor-Fenster. Hier schreibst du deinen Programmtext.



Speichere als Erstes den noch leeren Programmtext in deinem Projektordner ab. Wähle als Dateinamen `eliza.py`.



Du bist soweit. Beginnen wir mit dem Programmieren.

1.4 Projekt: Mini-Eliza

In diesem Projekt entwickeln wir in fünf Schritten eine Mini-Version eines Chatbots. Das Programm ist ganz einfach und wird bestimmt keinen Turing-Test bestehen. Dennoch zeigt es einige Features von Chatbots. Falls du noch nie mit Python programmiert hast, wirst du jetzt einige grundlegende Programmiertechniken kennenlernen.

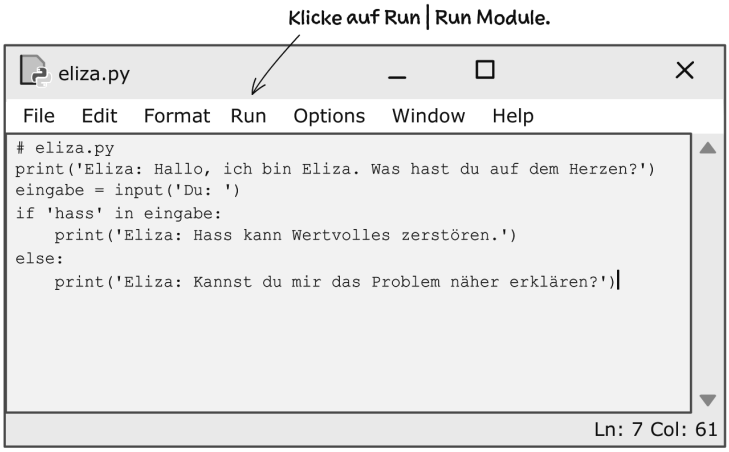
1.4.1 Schritt 1: Auf eine Eingabe reagieren

Ein Gespräch zwischen Mensch und Computer ist ein Wechselspiel von Texten. Das folgende Programm schreibt etwas auf einen Bildschirm, wartet auf eine Benutzereingabe und liefert dann eine passende Antwort. Das ist alles. Aber es ist ja auch nur der erste Schritt auf dem Weg zum Chatbot. Gib den Programmtext in das Editorfenster ein und speichere ihn, indem du im Menü FILE auf den Befehl SAVE klickst. Beachte, dass einige Zeilen um 4 Leerstellen eingerückt sind.

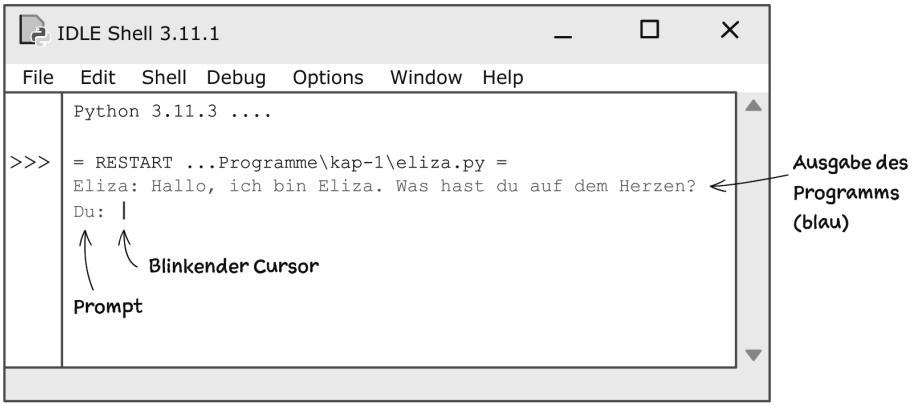
Programm:

```
# eliza.py
print('Eliza: Hallo, ich bin Eliza. Was hast du auf dem Herzen?')
eingabe = input('Du: ')
if 'hass' in eingabe:
    print('Eliza: Hass kann Wertvolles zerstören.')
else:
    print('Eliza: Kannst du mir das Problem näher erklären?')
```

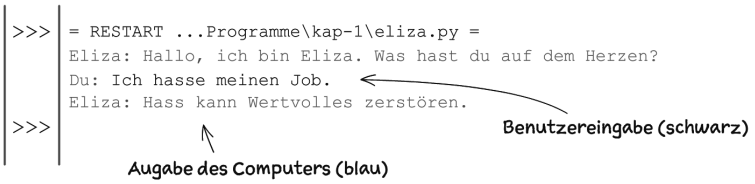
Starte das Programm vom Editor aus. Klicke zuerst in der Menüleiste auf RUN und dann auf den Befehl RUN MODULE.



Es öffnet sich ein Shell-Fenster. Oben stehen einige Informationen zu der Python-Version, die du verwendest. Hinter dem Prompt >>> erscheint eine Meldung, die mit = RESTART beginnt. Sie zeigt dir an, dass dein Programm ausgeführt wird. Darunter steht in blauer Schrift die Ausgabe des Programms.



Die zweite Zeile der Ausgabe besteht aus dem `Text Du:` . Hinter dem Doppelpunkt ist ein Leerzeichen und dahinter blinkt der Cursor. Der Computer wartet nun darauf, dass du etwas über die Tastatur eingibst. Den Text vor dem Cursor, also `Du:` (einschließlich Leerzeichen), nennt man *Prompt*. Ein Prompt signalisiert dem Benutzer: »Ich warte auf eine Eingabe.«. Sobald du die Taste Enter drückst, wird die Eingabe vom Computer übernommen und das Programm läuft weiter.



So funktioniert es:

Ein Python-Programm ist ein Text, der vom Python-Interpreter ausgeführt werden kann. Ein Programm besteht aus Anweisungen. Gehen wir nun den Programmtext Zeile für Zeile durch.

```
# eliza.py
```

Die erste Zeile ist keine Anweisung. Sie ist ein *Kommentar*, der für Menschen bestimmt ist und vom Python-Interpreter ignoriert wird. Ein Kommentar beginnt mit einem Hashtag # und kann so vom Python-Interpreter erkannt werden. Kommentare sollen die Lesbarkeit eines Programms verbessern. Oft wird eine Programmpassage stichwortartig erklärt. In diesem Fall wird der Name der Programmdatei angegeben. Alle Programmbeispiele in diesem Buch beginnen mit einem solchen Kommentar. So kannst du die Programme in den Ordnern des Downloadmaterials leicht wiederfinden.

```
print('Eliza: Hallo, ich bin Eliza. Was hast du auf dem Herzen?')
```

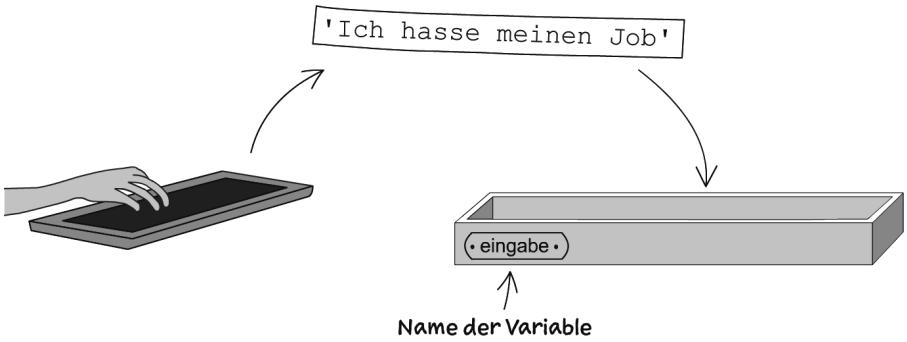
Diese Anweisung bewirkt, dass auf dem Bildschirm der Begrüßungstext ausgegeben wird. Die Anweisung ist ein Aufruf der Funktion `print()`. Dieser Aufruf ist so aufgebaut: Zuerst kommt der Name der Funktion, dahinter dann in Klammern der String `'Eliza: Hallo, ich bin Eliza.'`. Ein *String* ist eine Folge von beliebigen Zeichen, die durch Anführungszeichen eingerahmt sind.

```
eingabe = input('Du: ')
```

Hier wird die Funktion `input()` aufgerufen. Der Wert, der bei einem Funktionsaufruf in Klammern hinter dem Namen einer Funktion steht, heißt *Argument* der Funktion. Hier ist das Argument der String `'Du: '`.

Der Aufruf der Funktion bewirkt Folgendes: Auf dem Bildschirm wird der Prompt `Du:` ausgegeben – ohne die Anführungszeichen. Dann wartet der Computer. Der Benutzer des Programms kann nun über die Tastatur Zeichen eingeben. Sobald die Taste Enter gedrückt worden ist, wird ein String, der aus den eingegebenen Zeichen zusammengesetzt wird, der Variablen `eingabe` zugewiesen. Eine *Varia-*

ble kann man sich als Speicher für Daten vorstellen. Das Bild veranschaulicht an einem Beispiel, was bei der Zuweisung passiert.

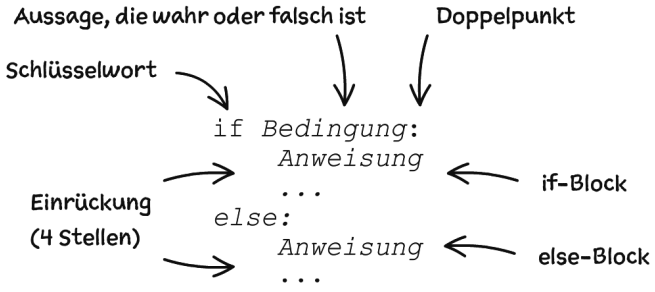


In dem Bild wird die Variable als Behälter dargestellt, der einen String als Inhalt aufnimmt. Der Clou ist: Über den Namen der Variablen kann man auf ihren Inhalt zugreifen. Das machen wir gleich in der nächsten Anweisung.

```
if 'hass' in eingabe:
    print('Eliza: Hass kann Wertvolles zerstören.')
else:
    print('Eliza: Kannst du mir das Problem näher erklären?')
```

Hier haben wir eine zusammengesetzte Anweisung, die über vier Zeilen geht. Es handelt sich um eine *if-else-Anweisung* (wenn-dann-sonst-Anweisung). Die Abbildung zeigt, wie eine if-else-Anweisung grundsätzlich aufgebaut ist. Die wichtigsten »Bauteile« sind

- eine *Bedingung*, d.h. eine Aussage, die wahr oder falsch sein kann,
- der *if-Block*, das sind Anweisungen, die ausgeführt werden, wenn die Bedingung erfüllt ist, also wahr ist,
- der *else-Block*, das sind Anweisungen, die ausgeführt werden, wenn die Bedingung nicht erfüllt ist, die Aussage also falsch ist.

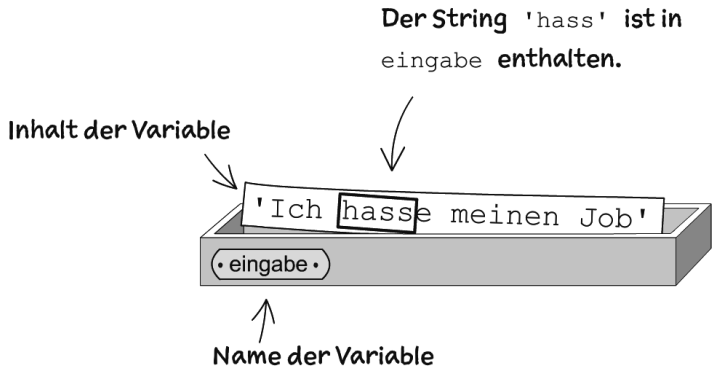


Beachte, dass if-Block und else-Block etwas eingerückt sind, typischerweise um 4 Stellen. Die Schlüsselwörter `if` und `else` müssen genau untereinanderstehen.

In diesem Fall lautet die Bedingung:

```
'hass' in eingabe
```

Das folgende Bild veranschaulicht an einem Beispiel, was damit gemeint ist.



Wenn die Bedingung erfüllt ist, wird die Anweisung

```
print('Eliza: Hass kann Wertvolles zerstören.')
```

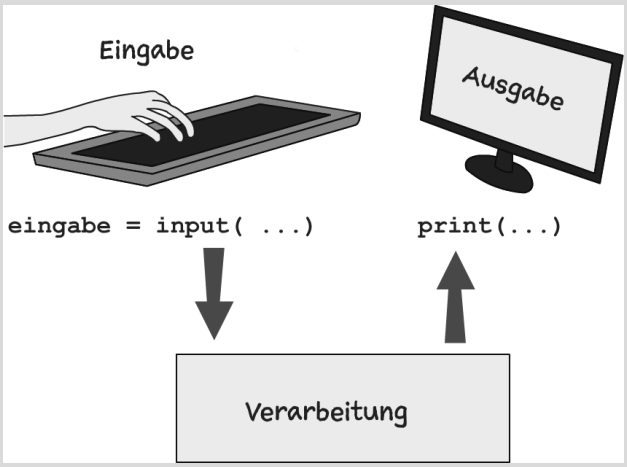
ausgeführt. Ansonsten wird

```
print('Eliza: Kannst du mir das Problem näher erklären?')
```

ausgeführt. Das Programm untersucht also die Eingabe und gibt einen passenden Text aus. Im Prinzip machen wir Menschen das auch, wenn wir uns mit jemandem unterhalten.

EVA-Prinzip

Computerprogramme sind oft nach dem EVA-Prinzip aufgebaut. Der Benutzer gibt z.B. über die Tastatur Daten ein (Zahlen, Texte). Der Computer verarbeitet die Daten und gibt ein Ergebnis in lesbarer Form auf dem Bildschirm aus.



1.4.2 Schritt 2: Mehr Intelligenz durch verschachtelte if-else-Anweisungen

Im zweiten Schritt erweitern wir die Möglichkeiten des Chatbots, den Text des Gesprächspartners oder der Gesprächspartnerin zu analysieren. Falls der String `'hass'` nicht vorkommt, wird geprüft, ob `'liebe'` enthalten ist, und so weiter. Es werden also mehrere Fälle unterschieden und für jeden Fall wird eine passende Antwort ausgegeben. Dieses Verhalten kann man mit verschachtelten if-else-Anweisungen programmieren. Ein else-Block enthält wieder eine if-else-Anweisung. In dein Programm fügst du einige neue Zeilen ein. Hier ist der geänderte Programmtext. Alle geänderten oder hinzugefügten Passagen sind fett gedruckt. Achte darauf, dass jedes `else` genau unter dem zugehörigen `if` steht.

Programm:

```
# eliza
print('Eliza: Hallo, ich bin Eliza. Was hast du auf dem Herzen?')
eingabe = input('Du: ')
if 'hass' in eingabe:
    print('Eliza: Hass kann Wertvolles zerstören.')
else:
    if 'liebe' in eingabe:
        print('Eliza: Liebe ist etwas Wunderbares.')
    else:
        if 'schlaf' in eingabe:
            print('Eliza: Schlaf ist wichtig.')
        else:
            print('Eliza: Kannst du mir das Problem näher erklären?')
```

Beispieldialog:

```
Eliza: Hallo, ich bin Eliza. Was hast du auf dem Herzen?
Du: Ich liebe meine Judolehrerin.
Eliza: Liebe ist etwas Wunderbares.
```

Man könnte in den Programmtext noch viele weitere if-else-Anweisungen einbauen und so den Chatbot verfeinern. Oder? Vielleicht ist dir schon ein Problem aufgefallen. Mit jeder neuen if-else-Anweisung entsteht ein Block, der weiter eingerückt ist. Die Programmzeilen werden immer länger und passen irgendwann nicht mehr auf den Bildschirm. Zum Glück gibt es Abhilfe.

1.4.3 Schritt 3: Mit elif die technische Qualität verbessern

Mit dem Schlüsselwort `elif` kannst du else und if zusammenfassen und sparst eine Einrückung. Genauer: Die if-elif-Anweisung

```
if Bedingung:
    Anweisung
elif Bedingung:
    Anweisung
```

hat genau die gleiche Wirkung wie die verschachtelte if-else-Anweisung

```
if Bedingung:
    Anweisung
else:
    if Bedingung:
        Anweisung
```

Mit diesem Trick wird das Programm kürzer und besser lesbar. Es leistet das Gleiche, aber seine technische Qualität hat sich verbessert. Im folgenden Listing sind die geänderten Passagen wieder fett gedruckt.

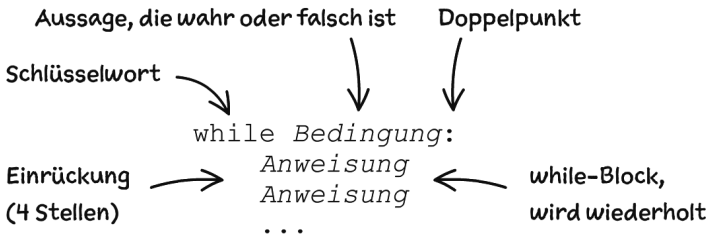
Programm:

```
# eliza
print('Eliza: Hallo, ich bin Eliza. Was hast du auf dem Herzen?')
eingabe = input('Du: ')
if 'hass' in eingabe:
    print('Eliza: Hass kann Wertvolles zerstören.')
elif 'liebe' in eingabe:
```

```
    print('Eliza: Liebe ist etwas Wunderbares.')
elif 'schlaf' in eingabe:
    print('Eliza: Schlaf ist wichtig.')
else:
    print('Eliza: Kannst du mir das Problem näher erklären?')
```

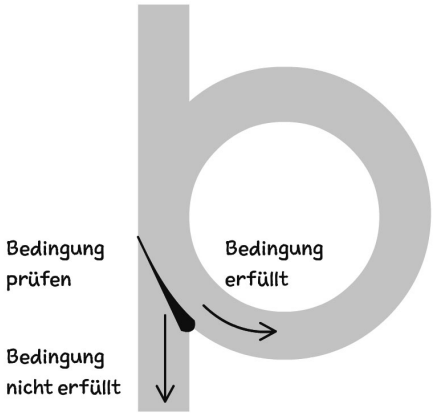
1.4.4 Schritt 4: Von der einfachen Reaktion zum Gespräch

Bisher hat der Chatbot nur ein einziges Mal auf einen eingegebenen Text reagiert. Dann endete der Programmlauf. Ein richtiges Gespräch dauert länger. Es ist ein Wechselspiel von Beiträgen der Gesprächspartner. Um das zu programmieren, muss die Folge Eingabe-Verarbeitung-Ausgabe wiederholt werden. Eine Wiederholung kannst du mit einer while-Anweisung programmieren. Das Bild zeigt den Aufbau.



Solange die Bedingung wahr ist, wird der eingerückte Anweisungsblock durchlaufen.

Wiederholungen werden im Jargon der Programmierung oft *Schleifen* genannt. Man denkt dann an einen Zug, der einen Schienenkreis mehrfach durchläuft. Die Weiche entspricht dem Prüfen der Bedingung. Wenn die Bedingung nicht erfüllt ist (also falsch ist), sorgt die Weiche dafür, dass der Zug die Schleife verlässt.



Wir programmieren den Chatbot so, dass die while-Schleife verlassen wird, wenn der menschliche Gesprächspartner keinen Text mehr eingibt, sondern direkt auf die Enter-Taste drückt. In diesem Fall ist der String `eingabe` leer. Er besteht nur aus zwei Anführungszeichen `''` oder `""`. Die Bedingung der while-Schleife lautet dann `eingabe != ''`. Das Operatorsymbol `!=` hat die Bedeutung *ungleich*.

Programmtext:

```
# eliza
print('Eliza: Hallo, ich bin Eliza. Was hast du auf dem Herzen?')
eingabe = 'x'                                                    #1
while eingabe != '':                                             #2
    eingabe = input('Du: ')                                      #3
    if 'hass' in eingabe:
        print('Eliza: Hass kann Wertvolles zerstören.')
    elif 'liebe' in eingabe:
        print('Eliza: Liebe ist etwas Wunderbares.')
    elif 'schlaf' in eingabe:
        print('Eliza: Schlaf ist wichtig.')
    elif eingabe != '':
        print('Eliza: Kannst du mir das Problem näher erklären?')
print('Es war wunderbar, mit dir zu reden. Bis bald!')      #4
```

So funktioniert es:

Beachte, dass die Kommentare `#1`, `#2`, ... vom Python-Interpreter ignoriert werden. Es sind keine Anweisungen. Sie haben keinen Einfluss auf den Programmlauf. Diese Kommentare mit Nummern sollen nur das Erklären des Programms erleichtern.

#1: Die Variable erhält als Inhalt den String `'x'`. Es hätte auch ein beliebiger anderer Wert außer dem leeren String `''` sein können. So wird sichergestellt, dass die while-Schleife wenigstens einmal durchlaufen wird.

#2: Solange der Inhalt der Variablen `eingabe` kein leerer String ist, wird der folgende eingerückte Block wiederholt. Vergiss nicht den Doppelpunkt am Ende der Zeile.

#3: Hier beginnt der Anweisungsblock, der wiederholt wird. Alle Anweisungen dieses Blocks sind gleich weit eingerückt.

#4: Diese Anweisung gehört nicht mehr zum while-Block. Sie wird erst dann ausgeführt, wenn die Schleife verlassen worden ist.

Suchwörter von Groß- und Kleinschreibung unabhängig machen

Der Ausdruck `'schlaf' in eingabe` ist dann wahr, wenn der String `eingabe` genau die Zeichenfolge `'schlaf'` enthält. Wenn du willst, dass der Test mit dem in-Operator auch für `'Schlaf'` oder `'SCHLAF'` funktioniert, musst du aus dem String `eingabe` einen String erzeugen, der nur kleine Buchstaben enthält. Das erledigt ein Aufruf der String-Methode `lower()`. Füge einfach die fett gedruckte Anweisung hinter die `input()`-Anweisung ein:

```
eingabe = input('Du: ')
eingabe = eingabe.lower()
```

Damit wird z.B. aus `'Mir fehlt Schlaf. '` der String `'mir fehlt schlaf.'`.

1.4.5 Schritt 5: Jetzt kommt der Zufall ins Spiel

Momentan kann man den Chatbot sehr leicht von einem Menschen unterscheiden, weil er häufig exakt das Gleiche sagt. Kein Mensch macht das. Im letzten Schritt der Chatbot-Entwicklung sorgen wir wenigstens an einer Stelle dafür, dass der Computer einen zufällig gewählten Text ausgibt. Auf diese Weise ist das Verhalten des Chatbots nicht mehr hundertprozentig vorhersehbar.

Programm:

```
# eliza
import random                                                    #1
PHRASEN = ['Kannst du mir das Problem näher erklären?',
           'Erzähle mir mehr darüber!',
           'Warum ist das wichtig für dich?']                    #2
print('Eliza: Hallo, ich bin Eliza. Was hast du auf dem Herzen?')
eingabe = 'x'
while eingabe != '':
    eingabe = input('Du: ')
    if 'hass' in eingabe:
        print('Eliza: Hass kann Wertvolles zerstören.')
    elif 'liebe' in eingabe:
        print('Eliza: Liebe ist etwas Wunderbares.')
    elif 'schlaf' in eingabe:
        print('Eliza: Schlaf ist wichtig.')
    elif eingabe != '':
        text = random.choice(PHRASEN)                            #3
        print('Eliza:', text)                                    #4
print('Eliza: Es war wunderbar, mit dir zu reden. Bis bald!')
```

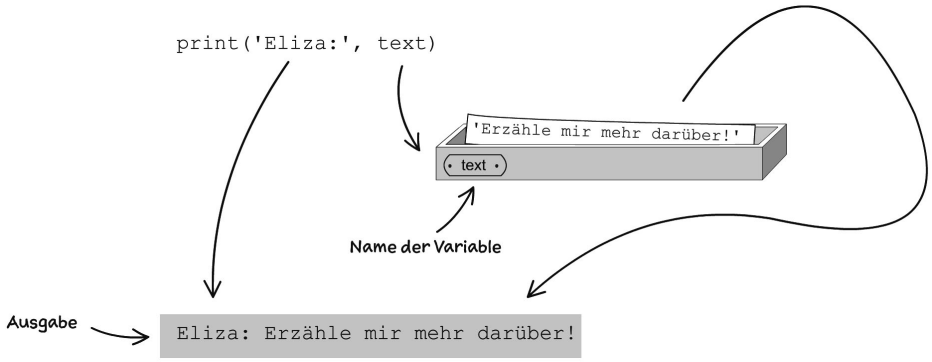
So funktioniert es:

#1: Zufallsfunktionen gehören nicht zum normalen Sprachumfang von Python. Deshalb musst du das Modul `random` importieren, um später eine spezielle Zufallsfunktion zu verwenden, die in dem Modul enthalten ist. Üblicherweise schreibt man Import-Anweisungen ganz an den Anfang eines Programms. Es gibt viele weitere Module für spezielle Anwendungszwecke, zum Beispiel das Modul `time` für Zeitfunktionen oder das Modul `math` für spezielle mathematische Berechnungen.

#2: Hier definieren wir eine *Liste* von Strings, aus der später ein String zufällig ausgewählt wird. Jede Liste beginnt und endet mit eckigen Klammern. Die enthaltenen Strings sind durch Kommas getrennt. Die Liste wird dem Namen `PHRASEN` zugewiesen. Diese Variable hat eine Besonderheit. Sie wird während des Programmlaufs nicht geändert, sondern sie bleibt, wie sie ist. Solche Variablen nennt man *Konstanten* und man schreibt ihre Namen üblicherweise mit Großbuchstaben.

#3: Hier wird die Funktion `choice()` aus dem Modul `random` aufgerufen. Weil das Modul `random` importiert worden ist, musst du beim Aufruf vor den Namen der Funktion den Namen des Moduls und dann einen Punkt schreiben. Das Argument ist die Liste `PHRASEN`. Die Funktion `choice()` wählt nach dem Zufallsprinzip einen String aus der Liste aus. Dieser Zufallsstring wird der Variablen `text` zugewiesen.

#4: Hier wird die `print()`-Funktion mit zwei Argumenten aufgerufen. Ausgegeben werden der String `'Eliza:'` und der Inhalt der Variablen `text`.



Bedingung

Bedingungen sind Aussagen, die wahr oder falsch sind. Eine einfache Bedingung kann durch Tests oder Vergleiche gebildet werden.

Operator	Bedeutung	Beispiel	Wahrheitswert
==	gleich	`1 == 2`	falsch
!=	ungleich	`1 != 2`	wahr
>	größer	`10 > 5`	wahr
<	kleiner	`5 < 5`	falsch
in	enthalten in	`'I' in 'team'`	falsch

Mit den logischen Operatoren `and` (und), `or` (oder) und `not` (nicht) kann man aus einfachen Bedingungen zusammengesetzte Bedingungen bilden.

`A and B` ist wahr, wenn sowohl A als auch B wahr sind, ansonsten ist es falsch.

`A or B` ist wahr, wenn wenigstens eine der beiden Aussagen A oder B wahr ist.

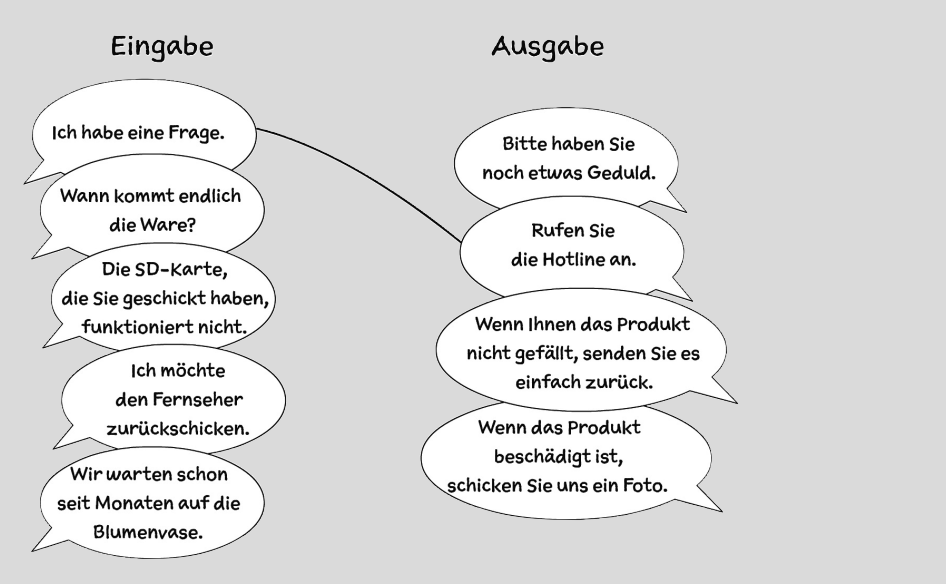
`not A` ist wahr, wenn A falsch ist, sonst ist es falsch.

Aufgabe 1: Digitaler Kummerkasten

Hier ist ein Chatbot eines Online-Shops, der Beschwerden entgegennimmt.

```
# kummerkasten.py
while True:
    eingabe = input(': ')
    if 'Wann' in eingabe or 'warte' in eingabe:
        print('Bitte haben Sie noch etwas Geduld.')
    elif 'beschädigt' in eingabe:
        print('Wenn das Produkt beschädigt ist,',
              'schicken Sie uns ein Foto.')
    elif 'zurück' in eingabe:
        print('Wenn Ihnen das Produkt nicht gefällt,',
              'senden Sie es einfach zurück.')
    else:
        print('Rufen Sie die Hotline an.')
```

Frage: Wie reagiert der Computer auf die verschiedenen Eingaben? Zeichne Verbindungslinien zwischen Eingabe und Ausgabe.



1.4.6 Fazit

Wir sind am Ende des ersten Python-Projekts. Du hast einen Chatbot entwickelt, der sich sehen lassen kann. Er wird vermutlich keinen Turing-Test bestehen, aber er hat Merkmale eines intelligenten Systems, denn er kann ein Gespräch führen und bei einer Problemlösung helfen. Moderne Chatbots wie ChatGPT sind viel komplexer und benutzen auch Methoden der subsymbolischen KI. Dennoch enthalten auch sie Techniken, mit denen schon vor mehr als einem halben Jahrhundert Eliza programmiert worden ist.

Übrigens wurde Joseph Weizenbaum, der Erfinder von Eliza, später zum Kritiker der Computertechnik und warnte vor ihrer Überschätzung. Er war damals entsetzt darüber, dass manche Leute sich mit Eliza wie mit einem Therapeuten unterhielten und ihr ihre intimsten Geheimnisse anvertrauten. Zum Glück hat Eliza sie nicht verstanden.

Stichwortverzeichnis

Z