

Michael Weigend

KI-Projekte programmieren mit Python

Dein einfacher Einstieg

Generative KI, Chatbots, Objekterkennung & Co.



Inhaltsverzeichnis

	Einleitung	9
	Wie ist das Buch aufgebaut?	9
	Programmbeispiele zum Download und Colab Notebooks	10
1	Denkende Maschinen	11
1.1	Was ist Intelligenz?	11
1.1.1	Rechnen mit Zahlen.	12
1.1.2	Gedächtnis	15
1.1.3	Räumliches Vorstellungsvermögen	19
1.1.4	Sprachverständnis	20
1.1.5	Schlussfolgerndes Denken	20
1.1.6	Wortflüssigkeit	22
1.1.7	Wahrnehmungsgeschwindigkeit	25
1.1.8	Starke und schwache KI.	26
1.2	Projekt: Wie schnell kann ein Computer rechnen?	27
1.3	Maschinen als Ersatzmenschen – Der Turing-Test	31
1.4	Projekt: Eine freundliche Maschine.	33
1.4.1	Den Computer sprechen lassen	35
1.5	Projekt: Ein Stichwort-Chatbot	35
1.6	Projekt: Eine KI als Spielgegner	42
1.6.1	Das Nim-Spiel.	42
1.6.2	Die Programmierung der Basisversion	43
1.7	Rückblick	49
1.8	Lösungen	49
2	Sprache und Denken – Natural Language Processing (NLP)	53
2.1	Trainingscamp: Strings.	53
2.1.1	Station 1: Strings erstellen	53
2.1.2	Station 2: Unicode-Zeichen.	55
2.1.3	Station 3: Strings durchsuchen	56
2.1.4	Station 4: Neue Strings aus alten Strings	57
2.1.5	Station 5: Formatierung	58
2.2	Projekt: Storytelling	59
2.2.1	Mad Libs	59
2.2.2	Eine digitale Version	60
2.3	Sprachmodelle nutzen – Natural Language Processing (NLP)	61
2.3.1	spaCy in einer virtuellen Umgebung installieren	62
2.4	NLP mit spaCy ausprobieren	64
2.4.1	Tokenisierung.	64

2.4.2	Stoppwörter	68
2.4.3	Einen Text in Sätze zerlegen	69
2.5	Projekt: Mit NLP einen Text automatisch zusammenfassen	70
2.5.1	Vorüberlegung: Wie erkennt man, was wichtig ist?	70
2.5.2	Programmierung	72
2.6	Was ist ein Apfel? Die Bedeutung von Wörtern	76
2.6.1	Semantisches Netz	76
2.6.2	Embeddings und Ähnlichkeit – Experimente mit spaCy.	77
2.6.3	Ähnlichkeit	81
2.7	Projekt: Schlag nach bei Goethe! Ähnliche Wörter finden	82
2.8	Projekt: Fitness-Chatbot für Reisende	85
2.9	Rückblick	88
2.10	Lösungen	89
3	Lernende Programme	93
3.1	Wie Computer »intelligent« werden	93
3.1.1	Überwachtes Lernen: Regression	93
3.1.2	Überwachtes Lernen: Klassifizieren	102
3.1.3	Unüberwachtes Lernen	104
3.1.4	Verstärkungslernen	106
3.2	Projekt: Lernen durch Versuch und Irrtum – Wegesuche auf dem Mond	107
3.2.1	Turtle-Grafik	108
3.2.2	Das Simulationsprogramm	109
3.2.3	Fazit	120
3.3	Rückblick	121
3.4	Lösungen	121
4	Inspiziert durch das Gehirn – Künstliche neuronale Netze	125
4.1	Das Gehirn – Ein natürliches neuronales Netz	125
4.2	Künstliches neuronales Netz	127
4.2.1	Aktive Knoten	129
4.2.2	Aktivierungsfunktionen	131
4.3	Ein neuronales Netz entwerfen und trainieren	135
4.3.1	Uneinigkeit erkennen	136
4.3.2	Vorhersagen berechnen	137
4.3.3	Trainingsprozess und Batchgröße	138
4.3.4	Verlustfunktion (Loss)	139
4.3.5	Gradientenabstieg	141
4.3.6	Klassischer Gradientenabstieg und Stochastischer Gradientenabstieg	144
4.3.7	Adam-Optimierer	144
4.4	Trainingscamp: Arrays verarbeiten mit NumPy	146
4.4.1	Vorbereitung	147

4.4.2	Station 1: Arrays erzeugen	147
4.4.3	Station 2: Rechnen mit Arrays.	150
4.4.4	Station 3: Arrays in Form bringen	152
4.5	Projekt: Uneinigkeit erkennen	155
4.5.1	Vorbereitung	155
4.5.2	Google Colab verwenden	156
4.5.3	Programmierung	158
4.6	Projekt: Ziffern erkennen	163
4.6.1	Programmierung	164
4.7	Rückblick	171
4.8	Lösungen	171
5	Der Computer als Künstler – Bilder generieren mit einem GAN	175
5.1	Was ist generative KI?	175
5.2	Wie funktioniert ein GAN?	177
5.3	Trainingscamp: Tensoren	178
5.3.1	Station 1: Eigenschaften eines Tensors.	178
5.3.2	Station 2: Tensoren erzeugen	182
5.3.3	Station 3: Tensoren in Form bringen	184
5.3.4	Station 4: Zugriff auf Teile eines Tensors	186
5.3.5	Station 5: Tensoren als Bilder anzeigen	187
5.4	Projekt: Kreative Ziffern	191
5.4.1	Programmierung	192
5.4.2	Weiterentwicklung: Modelle speichern und trainierte Modelle nutzen.	204
5.5	Projekt: Künstliche Gesichter	206
5.5.1	Vorbereitung	207
5.5.2	Programmierung	208
5.6	Rückblick	210
5.7	Lösungen	211
6	Textproduktion mit rekurrenten neuronalen Netzen (RNN)	217
6.1	»Es war einmal« – Ein Sprachmodell mit Bleistift und Papier	217
6.2	Wie funktioniert ein RNN?	220
6.2.1	Hat ein künstliches neuronales Netz ein Gedächtnis?	220
6.2.2	Neuronen mit Gedächtnis – Rekurrente neuronale Netze (RNNs)	221
6.2.3	Gradientenverschwinden und Gradientenexplosionen.	223
6.2.4	Die Arbeitsweise von LSTM-Zellen.	225
6.3	Projekt: In sechs Schritten zum Textgenerator	229
6.3.1	Die Grundidee	230
6.3.2	Die Programmierung	232

6.4	Das Projekt weiterentwickeln	242
6.4.1	Mehr Zufall	243
6.4.2	Mehr Komplexität durch zusätzliche Schichten	244
6.4.3	Eine moderne Weiterentwicklung: Gated Recurrent Unit (GRU)	245
6.4.4	Preprocessing: Satzzeichen verarbeiten	246
6.5	Eine Alternative zu RNN: Transformer	248
6.5.1	Programmierung	249
6.6	Rückblick	251
6.7	Lösungen	252
7	Musik komponieren mit KI	255
7.1	Trainingscamp: Noten verarbeiten mit music21	255
7.1.1	Station 1: Vorbereitung	255
7.1.2	Station 2: Melodien mit Streams	256
7.1.3	Station 3: Eine Melodie definieren und abspielen	257
7.1.4	Station 4: Mehrstimmige Partituren mit Scores	260
7.1.5	Station 5: Aus einer MIDI-Datei die Melodie gewinnen	263
7.1.6	Station 6: Eine Melodie transponieren	266
7.2	Projekt: Melodien komponieren	267
7.2.1	Erweiterung: Mehr Trainingsdaten	275
7.2.2	Anregungen für Experimente und Entwicklungen	278
7.3	Rückblick	279
7.4	Lösungen	280
8	Diffusionsmodelle – Bilder aus Rauschen	283
8.1	Die Grundidee	283
8.2	Faltung (Konvolution) – Ein Bild durch Features beschreiben	285
8.2.1	Features im Alltag	285
8.2.2	Mit Filtern Features erkennen	286
8.2.3	Padding und Stride	288
8.2.4	Lernende Filter in einem Convolutional Neural Network (CNN)	289
8.2.5	Faltung als neuronales Netz	290
8.3	Wie entrauscht man ein verrauschtes Bild?	291
8.3.1	Das U-Net und Bildsegmentierung in der Medizin	291
8.3.2	Faltung, Pooling und Skip-Connections – Die Magie der U-Nets	293
8.3.3	Veranschaulichung: Ein beschädigtes Bild rekonstruieren	299
8.4	Projekt: Künstliche Ziffern	300
8.5	Rückblick	312
8.6	Lösungen	313
	Stichwortverzeichnis	315

Denkende Maschinen

Was ist eigentlich Intelligenz? In diesem Kapitel gehen wir dieser Frage auf den Grund und schaffen gleichzeitig die Grundlage für die im Buch folgenden KI-Projekte. Falls du mit der Programmiersprache Python noch nicht vertraut bist, erhältst du ganz nebenbei eine kurze Einführung.

Zunächst werfen wir einen Blick auf ein psychologisches Modell der menschlichen Intelligenz. Die »Primärfaktoren« dieses Modells sind dir wahrscheinlich von populärwissenschaftlichen Intelligenztests und Denksportaufgaben bekannt (Gegenstände in Gedanken rotieren, Bildfolgen ergänzen, memorieren, Synonyme finden etc.). Einige dieser Fähigkeiten beherrschen Computer ganz hervorragend. Mit 3D-Editoren kann man 3D-Objekte aus beliebigen Blickwinkeln betrachten, Computer besitzen ein hervorragendes Gedächtnis etc.

Im Anschluss wird der Turing-Test (»Imitation Game«) behandelt. Demzufolge ist ein System intelligent, wenn es in seinem Verhalten nicht von einem Menschen unterschieden werden kann.

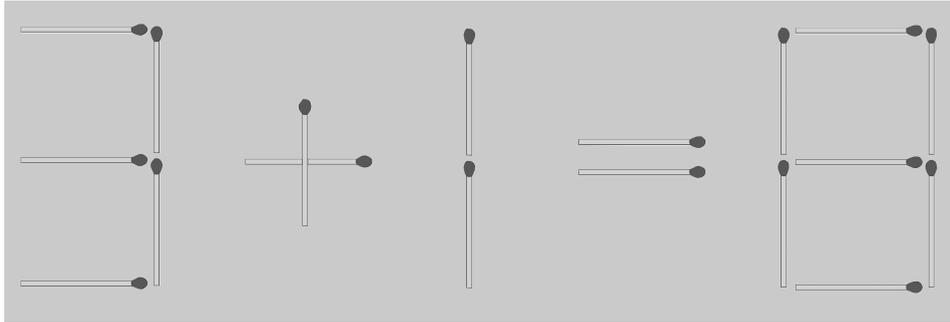
Im Rest des Kapitels werden einige Programme erläutert, die auf diesem KI-Verständnis (»KI als Ersatzmensch«) basieren: Chatbots, mit denen man sich unterhalten kann oder die von sich aus Menschen ansprechen, und ein Spiel mit einer KI als Spielgegner (»Nim«). Alle in diesem Kapitel beschriebenen Programme sind Beispiele für *symbolische KI*, deren Arbeitsweise durch nachvollziehbare Regeln bestimmt ist. Später schauen wir uns *subsymbolische KI* an. Dazu gehören *künstliche neuronale Netze*. Sie lernen aus Beispielen, aber wie sie Entscheidungen treffen, ist nicht zu durchschauen.

1.1 Was ist Intelligenz?

Bist du intelligent? Aber klar, alle Menschen sind intelligent. Intelligenz ist eine menschliche Eigenschaft. Auch wenn es in der Wissenschaft keine einheitliche Definition gibt, verstehen die meisten Psychologen darunter die Fähigkeit, Probleme zu lösen. Hier ist ein Problem. Kannst du es lösen?

Aufgabe 1: Ein Streichholzproblem

Verschiebe ein einziges Streichholz und Sorge dafür, dass die Gleichung stimmt.



Um eine Lösung zu finden, nutzt du mehrere Fähigkeiten deines Verstandes. Zunächst einmal bist du in der Lage, den Text der Aufgabenstellung zu lesen und zu verstehen. Du erkennst die Bedeutung der mathematischen Symbole und kannst durch Nachrechnen feststellen, dass die Gleichung $3 + 1 = 8$ nicht stimmt. Um das Problem zu lösen, brauchst du kein einziges Streichholz anzufassen, sondern kannst in Gedanken die Streichhölzer verschieben und dir das Ergebnis vorstellen.

Der US-amerikanische Psychologe Louis Leon Thurstone hat sieben voneinander unabhängige Grundfähigkeiten beschrieben, die unsere Intelligenz ausmachen. Er nannte sie *Primärfaktoren der Intelligenz*:

- Rechnen mit Zahlen
- Räumliches Vorstellungsvermögen
- Gedächtnis
- Sprachverständnis
- Schlussfolgerndes Denken
- Wortflüssigkeit (Fähigkeit, schnell und leicht Wörter zu produzieren)
- Wahrnehmungsgeschwindigkeit

Dieses Modell bildet die Grundlage vieler moderner Intelligenztests. Werfen wir einen genaueren Blick auf einige der Primärfaktoren und vergleichen die Leistungsfähigkeit von Menschen und Computern. Falls du mit der Programmiersprache Python nicht vertraut bist, erhältst du jetzt ganz nebenbei eine kleine Einführung in einige Grundtechniken.

1.1.1 Rechnen mit Zahlen

Das Rechnen mit Zahlen (*number facility*) haben wir mühsam in der Schule gelernt. Wir können kleinere Zahlen im Kopf addieren, subtrahieren und multiplizieren. Das klappt einigermaßen und hilft uns, durch den Alltag zu kommen. Bei spezielleren Operationen wie Wurzelziehen wird es schon schwieriger. Das können nur wenige Menschen im Kopf. Wie steht es um die Geschwindigkeit? Der

Weltmeister im Kopfrechnen ist der Inder Aaryan Shukla. Bei der Weltmeisterschaft in Paderborn 2024 brauchte er für die Addition von zehn zehnstelligen Zahlen 96,39 Sekunden. Dein Computer rechnet sehr viel schneller. Selbst, wenn du alle Zahlen eintippen musst, kann du mit einem Taschenrechner den Weltmeister im Kopfrechnen leicht schlagen.



Abb. 1.1: Der Weltmeister im Kopfrechnen: Aaryan Shukla (2024, 14 Jahre alt)

Rechnen mit Python

Sorge dafür, dass auf deinem Computer Python installiert ist. Python ist eine weitverbreitete Programmiersprache, die leicht zu lernen ist. Die meisten KI-Programme sind in Python geschrieben.

Hinweis: Python installieren

Python ist kostenlos und kann einfach von der Webpräsenz der Python Software Foundation heruntergeladen werden. Besuche die Webseite <https://www.python.org/> und klicke auf **DOWNLOAD**. Es kann sinnvoll sein, nicht die aktuellste Python-Version zu wählen, sondern z.B. Python 3.11, denn manche Module, die wir später installieren werden, funktionieren noch nicht mit der aktuellsten Version.

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 neueste 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.

Um ein Programm zu schreiben, brauchst du eine Entwicklungsumgebung (*Integrated Development Environment, IDE*). Wir verwenden in diesem Buch ausschließlich IDLE. Wenn du IDLE startest, öffnet sich die IDLE-Shell. Darin kannst du einzelne Python-Befehle ausprobieren. Insbesondere kannst du die IDLE-Shell zum Rechnen verwenden. Schreibe einfach hinter den Prompt `>>>` einen arithmetischen Ausdruck und drücke die Taste `Enter`. In der nächsten Zeile erscheint das Ergebnis.

```
>>> 1 + 2
3
```

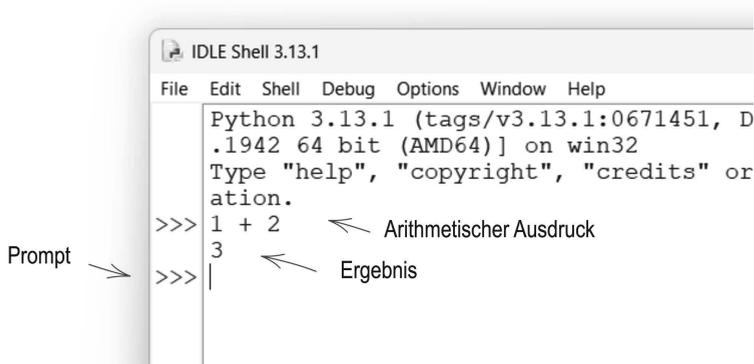


Abb. 1.2: Die IDLE-Shell als Taschenrechner

Für Multiplikationen verwendest du `*` und für Divisionen `/`:

```
>>> 2 * 3
6
>>> 3 / 2
1.5
```

Der Operator `//` wird für ganzzahlige Divisionen verwendet. Das Ergebnis einer ganzzahligen Division ist immer eine ganze Zahl, nämlich das nach unten gerundete Ergebnis einer exakten Division:

```
>>> 3 // 2
1
```

In einigen Projekten werden wir auch den Modulo-Operator `%` verwenden. Der Ausdruck `a % b` ist der Rest der Division `a//b`. Beispiele:

```
>>> 5 % 2
1
```

Wenn man 5 durch 2 teilt, ergibt das 2 und es bleibt der Rest 1.

```
>>> 4 % 2
0
```

Wenn man 4 durch 2 teilt, geht das glatt auf und es bleibt kein Rest.

Für das Potenzieren verwendet man den Operator `**`. Beispiel: $10^{**}3 = 10^3 = 1000$.

Python unterscheidet zwischen ganzen Zahlen (Typ `int`) und Gleitkommazahlen (Typ `float`). Eine Gleitkommazahl wird mit einem Punkt (und nicht mit einem Komma) geschrieben, z.B. `1.234`. Die Stellen nach dem Punkt sind die Nachkommastellen. Besonders große Zahlen oder Zahlen, die sehr nahe an 0 sind, werden in der Exponentenschreibweise dargestellt. Hinter dem Buchstaben `e` (oder `E`) steht der Exponent einer Zehnerpotenz. Der Ausdruck `1.3e6` ist $1,3 \cdot 10^6$, also 1.300.000. `2.9E-5` ist $2,9 \cdot 10^{-5}$, also 0,000029.

Du kannst auch komplizierte arithmetische Ausdrücke mit Klammern auswerten lassen:

```
>>> 3 * (2 + 5)
21
```

1.1.2 Gedächtnis

Thurstone definierte das assoziative Gedächtnis (*associative memory*) als die Fähigkeit, sich an Paare von Elementen zu erinnern und sie schnell abrufen zu können. Menschen mit gutem assoziativem Gedächtnis fällt es leicht, sich an zusammengehörige Informationen zu erinnern, z.B. Wortpaare oder Ursache-Wirkungs-Beziehungen. Dies hilft bei der Entwicklung von Problemlösungen.

In einem Python-Programm kannst du Daten durch eine Zuweisung speichern. Das ist eine Anweisung der Form

```
Name = Wert
```

In einer Zuweisung wird einer Variablen ein Wert zugewiesen. Das kannst du in der IDLE-Shell ausprobieren.

```
>>> x = 15
```

Hier wird der Variablen mit dem Namen `x` der Wert 15 zugewiesen. Man kann auch sagen, die Variable `x` enthält als Wert die Zahl 15. Du kannst dir die Variable als Behälter vorstellen, die einen Wert enthält (Abbildung 1.3 links).

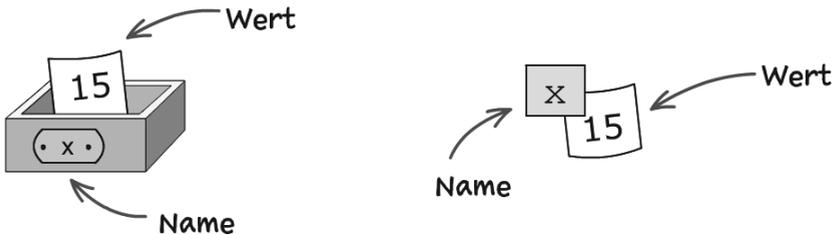


Abb. 1.3: Zwei Veranschaulichungen für Variablen

Eine andere Anschauung ist, dass man an die Zahl 15 ein Etikett mit dem Namen `x` geklebt hat (Abbildung 1.3 rechts). Über den Namen einer Variablen kommt man an ihren Inhalt.

Mit einer `print()`-Anweisung kannst du den Inhalt einer Variablen ausgeben.

```
>>> print(x)
15
```

In der IDLE-Shell reicht es auch, wenn du einfach nur den Namen einer Variablen eingibst. Ihr Wert erscheint dann in der nächsten Zeile.

```
>>> x
15
```

Du kannst den Inhalt einer Variablen einer anderen Variablen zuweisen:

```
>>> y = x
```

Jetzt hat y den gleichen Wert wie x. Prüfe es nach!

```
>>> y
15
```

Du kannst dir anschaulich vorstellen, dass eine Kopie des Inhalts von x in der Variablen y gespeichert wird.

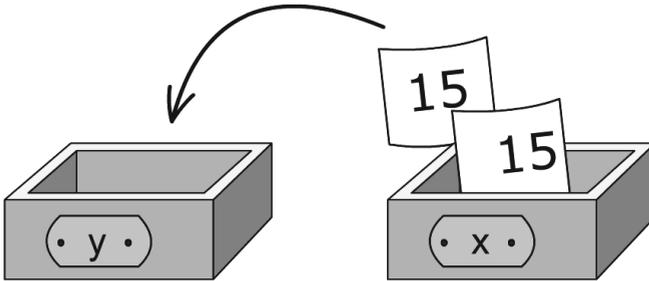


Abb. 1.4: Wertübertragung

Den Namen einer Variablen kannst du frei wählen, du musst nur einige wenige Regeln beachten:

- Der Name besteht aus beliebigen Buchstaben, Ziffern oder `_` (Unterstrich).
- Er muss mit einem Buchstaben oder einem Unterstrich beginnen.
- Er darf kein Schlüsselwort sein (z.B. `if`, `else`, `not`). Schlüsselwörter gehören zur Programmiersprache und sind schon mit bestimmten Bedeutungen belegt.

Erlaubte Variablenamen sind z.B. `zahl_1`, `_name_` oder `höhe`. Nicht erlaubt sind z.B. `1x` (beginnt mit einer Ziffer) und `zahl-1` (enthält nicht erlaubtes Zeichen `-`). Meistens schreibt man Variablenamen klein.

Variablen sind ein sehr wichtiges Programmierkonzept. Sie können nicht nur Zahlen, sondern auch andere Daten speichern, zum Beispiel:

- Zeichenketten, z.B. `'Guten Morgen'`
- Listen, z.B. `[0, 1, 1, 2, 3, 5, 8]`
- Tupel, z.B. `('Tina', 25)`
- Mengen, z.B. `{1, 2, 3}`
- Dictionaries, z.B. `{'Mond': 'moon', 'Sonne': 'sun'}`

Wir werden später in Programmierprojekten mit diesen Datentypen arbeiten.

Zum Abschluss dieses Abschnitts noch etwas zur Technik des Speicherns: Dein Computer hat unterschiedliche Speicher, einen Arbeitsspeicher und einen Peripheriespeicher. Der Arbeitsspeicher (RAM = Random Access Memory) ist der schnelle, kurzfristige Speicher. Er speichert Daten und Programme nur vorübergehend, solange der Computer eingeschaltet ist. Wenn der Strom ausfällt, ist alles weg. Der Peripheriespeicher umfasst alle Speichermedien, die Daten dauerhaft speichern:

- Festplatte (HDD, SSD)
- SD-Karte
- USB-Stick
- CD, DVD

Wenn du Daten auf der Festplatte gespeichert hast, sind sie sicher und gehen nicht verloren, wenn du den Computer ausschaltest.

Die Einheit für Speicher ist Byte. Mit einem Byte kann man eine Zahl zwischen 0 und 255 darstellen. Ein typischer Laptop hat einen Arbeitsspeicher mit 16 Gigabyte (16 Milliarden Bytes) und als Peripheriespeicher eine SSD mit ein Terabyte (eine Billion Bytes).

Die Variablen in deinem Python-Programm sind im Arbeitsspeicher gespeichert. Mit den folgenden Anweisungen kannst du in der IDLE-Shell prüfen, wie viel Arbeitsspeicher für Variablen zur Verfügung steht. Dazu musst du zunächst eine spezielle Funktion aus dem Modul `psutil` importieren.

```
>>> from psutil import virtual_memory
```

Diese Funktion kann nun aufgerufen werden. Sie liefert ein Objekt, das den augenblicklichen Zustand des Arbeitsspeichers beschreibt.

```
>>> virtual_memory()
svmem(total=16856489984, available=5359079424, percent=68.2, used=114974
10560, free=5359079424)
```

In diesem Fall sind noch mehr als fünf Gigabyte frei. Das ist eine Menge. Wir werden aber später neuronale Netze programmieren, die beim Training so riesige Datenmengen verarbeiten, dass nur ein Teil davon im Arbeitsspeicher gehalten werden kann.

Übrigens, die Speicherkapazität des menschlichen Gehirns wird auf etwa ein Petabyte geschätzt, was 1.000.000 Gigabyte (GB) oder 1.000 Terabyte (TB) entspricht. In puncto Speicherkapazität ist also dein Gehirn deinem Laptop haushoch überlegen.

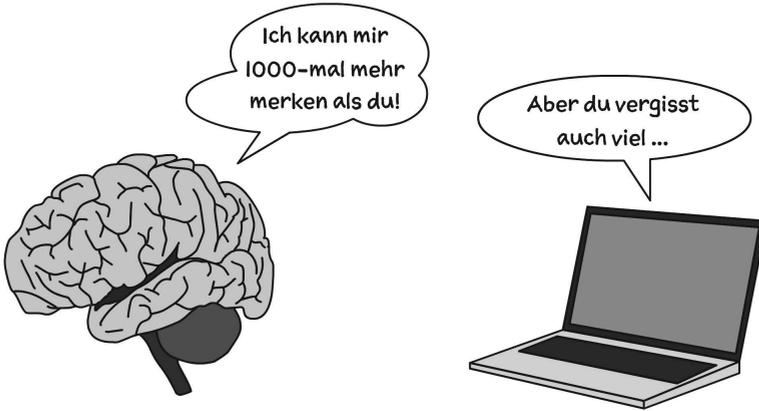


Abb. 1.5: Der Computer hat Speicher – das Gehirn hat Erinnerungen.

1.1.3 Räumliches Vorstellungsvermögen

Unter räumlichem Vorstellungsvermögen (*spatial visualization*) versteht man die Fähigkeit, sich Gegenstände und deren räumliche Beziehungen in Gedanken vorzustellen, sie zu bewegen und aus verschiedenen Perspektiven zu betrachten. Am Beispiel der Streichholzaufgabe hast du schon gesehen, wie wichtig räumliches Denken für Problemlösungen sein kann. Mit einem 3D-Grafikprogramm wie z.B. Blender kann du Gegenstände als dreidimensionale Modelle gestalten und dir auf dem Bildschirm von allen Seiten ansehen.

Aufgabe 2: Was sieht der Roboter?

A

B

C

D

1.1.4 Sprachverständnis

Sprachverständnis (*verbal comprehension*) ist die Fähigkeit, die Bedeutung von Wörtern und Sätzen zu erfassen und mit Sprache umzugehen. Das ist eine komplexe Kompetenz, die viele Facetten umfasst. In Intelligenztestaufgaben zum Sprachverständnis werden u.a. folgende Leistungen geprüft:

- Zu einem Begriff ein Synonym oder ein Antonym finden
- Einen Text lesen und Fragen zum Text beantworten
- In einem unvollständigen Satz ein passendes Wort ergänzen
- Grammatikfehler in Sätzen erkennen
- Die Bedeutung von Sprichwörtern erkennen
- Wörter in die richtige Reihenfolge bringen

Aufgabe 3: Wortbeziehungen erkennen

»Speichern« verhält sich zu »Festplatte« wie »Schreiben« zu ...

- a) Stift
- b) Papier
- c) Lesen
- d) Tastatur

Moderne Chatbots wie ChatGPT haben ein sehr gutes Sprachverständnis. Sie können grammatische Fehler finden, Texte stilistisch verbessern und sogar in andere Sprachen übersetzen.

In einem Python-Programm werden Texte als Zeichenketten (Strings) gespeichert. Ein Stringliteral beginnt und endet mit einfachen oder doppelten Anführungszeichen, z.B. 'Intelligenz' oder "Python".

Für die Verarbeitung natürlicher Sprache gibt es spezielle Python-Module, die KI verwenden, wie z.B. NLTK (Natural Language Tool Kit) oder SpaCy. In Kapitel 2 werden wir uns damit beschäftigen.

1.1.5 Schlussfolgerndes Denken

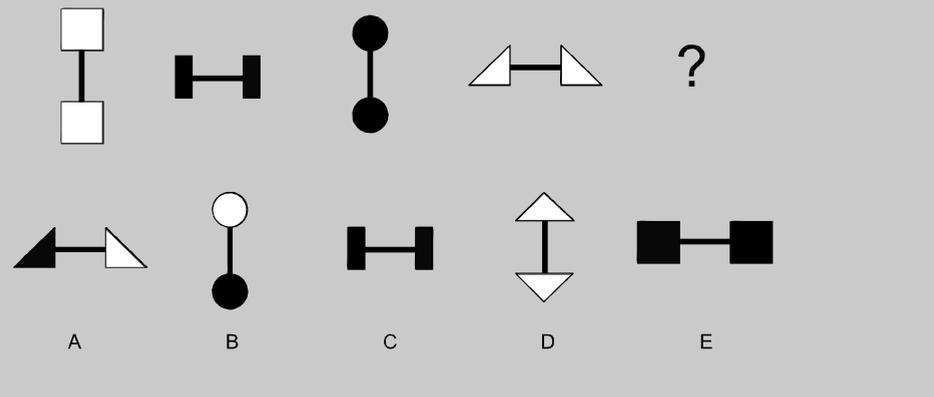
Schlussfolgerndes Denken (*reasoning*) ist die Fähigkeit, Regeln und Muster zu erkennen, logische Zusammenhänge herzustellen und aus gegebenen Informationen neue Erkenntnisse abzuleiten. Es gibt mehrere Formen des schlussfolgernden Denkens:

- Induktives Denken – Von einzelnen Beobachtungen auf allgemeine Regeln schließen.
- Deduktives Denken – Aus allgemeinen Regeln spezifische Schlüsse ziehen.
- Analogie-Denken – Ähnlichkeiten zwischen verschiedenen Sachverhalten erkennen.

In Aufgabe 4 werden sowohl induktives als auch deduktives Denken verlangt. Zuerst musst du durch Induktion in einer Folge von vier konkreten Figuren Regeln entdecken. Dann musst du Deduktion anwenden, um mithilfe dieser Regeln die nächste Figur zu finden.

Aufgabe 4: Fortsetzung einer Folge

Was kommt als Nächstes? Ersetze das Fragezeichen durch eine passende Figur (A bis E).



Heutige Chatbots sind noch nicht besonders gut im schlussfolgernden Denken. Probiere einmal folgende Frage mit ChatGPT oder einem anderen Chatbot aus:

»Tina hat zwei Brüder und eine Schwester. Wie viele Schwestern hat ihr Bruder?«

Die richtige Antwort ist zwei. Denn unter den Geschwistern sind Tina und noch ein weiteres Mädchen.

GPT-4o liefert aber die folgende Antwort (Februar 2025):

»Tina hat eine Schwester. Da ihr Bruder auch Tina als Schwester hat, hat er insgesamt eine Schwester.«

Sowohl die Antwort als auch die Begründung sind falsch. Allerdings werden Chatbots rasch besser und werden sicher bald auch solche Schlussfolgerungen beherrschen.

1.1.6 Wortflüssigkeit

Wortflüssigkeit (*Word Fluency*) ist die Fähigkeit, schnell und leicht Wörter zu produzieren. Hier sind einige typische Fragen, mit denen die Wortflüssigkeit einer Person gemessen wird:

- »Finde so viele Wörter wie möglich, die sich auf *Haus* reimen.«
- »Nenne in einer Minute so viele Wörter wie möglich, die mit B beginnen.« (Lexikalische Wortflüssigkeit)
- »Nenne in einer Minute so viele Tiere wie möglich.« (Semantische Wortflüssigkeit)

Lexikalische Wortflüssigkeit ist für einen Computer kein Problem, denn er kann über einen riesigen Wortschatz verfügen. Ein Mensch hat im Schnitt einen aktiven Wortschatz von 16.000 Wörtern. Das heißt, so viele Wörter kennt er oder kann sie aktiv verwenden. Der Duden enthält in seiner aktuellen 29. Auflage etwa 151.000 Wörter. Dafür werden nur wenige Megabyte Speicherplatz benötigt. Auf deinem Laptop (mit mehreren Gigabyte Arbeitsspeicher) kannst du also problemlos eine Liste mit sämtlichen Wörtern des Dudens erstellen und verarbeiten. Aus einer solchen Liste können alle Wörter ausgefiltert werden, die mit B beginnen.

Beispiel: Finde alle Wörter, die mit B beginnen

Zum Ausprobieren erstellst du in der IDLE-Shell eine Liste mit einigen Wörtern, z.B.:

```
>>> wörter = ['Ball', 'Affe', 'Mensch', 'Banane', 'beginnen']
```

Hier wird eine Liste mit dem Namen `wörter` erzeugt. Achte auf die eckigen Klammern. Eine solche Liste kann in einer Iteration (man sagt auch `for`-Anweisung oder `for`-Schleife) durchlaufen und verarbeitet werden. Die folgende Iteration sorgt dafür, dass alle Wörter der Liste, die mit einem B beginnen, ausgegeben werden:

```
>>> for wort in wörter:  
    if wort[0] == 'B':  
        print(wort)
```

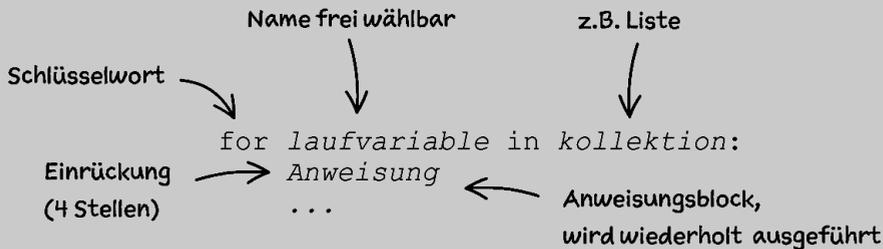
Die Ausgabe erscheint in den nächsten Zeilen:

```
Ball  
Banane
```

Die erste Zeile der for-Anweisung ist so aufgebaut: Zuerst kommt das Schlüsselwort `for`, dann der Name der Laufvariablen, dann das Schlüsselwort `in`, der Name einer Liste und schließlich ein Doppelpunkt. Der Doppelpunkt zeigt an, dass nun ein neuer Anweisungsblock beginnt. Dieser Anweisungsblock muss um einige Stellen eingerückt sein. Die IDLE-Shell macht die Einrückung automatisch. Bei der Ausführung der `for`-Anweisung wird dieser Anweisungsblock wiederholt. Für jedes Element der Liste wird er einmal ausgeführt. Die Laufvariable `wort` enthält immer das aktuelle Element der Liste. In diesem Fall besteht der Anweisungsblock aus einer einzigen `if`-Anweisung. Der Ausdruck `wort[0]` ist der erste Buchstabe des Strings `wort`. (In der Informatik beginnt eine Nummerierung immer mit 0.) Wenn der erste Buchstabe von `wort` das Zeichen `B` ist, dann wird die eingerückte Anweisung `print(wort)` ausgeführt. Der Inhalt von `wort` wird auf dem Bildschirm ausgegeben.

Iteration – for-Anweisung

In einer Iteration (`for`-Anweisung) werden alle Elemente (Items) einer Kollektion (z.B. Liste) durchlaufen und verarbeitet. Das aktuelle Element wird in der Laufvariablen gespeichert. Das Bild zeigt den allgemeinen Aufbau.



Führen wir die Iteration ein Stück weit aus und beobachten wir die Inhalte der Variablen:

- Beim ersten Durchlauf der Iteration bezeichnet die Laufvariable `wort` das erste Element der Liste, also `'Ball'`. Dann wird geprüft, ob der erste Buchstabe von `wort` ein großes `B` ist. Ja, das ist der Fall. Deshalb wird die `print()`-Anweisung ausgeführt und das Wort `Ball` ausgegeben.
- Beim zweiten Durchlauf der Iteration bezeichnet `wort` das zweite Element der Liste, also `'Affe'`. Dann wird geprüft, ob der erste Buchstabe von `wort` ein großes `B` ist. Nein, das ist nicht der Fall. Deshalb passiert nichts.
- Und so weiter ...

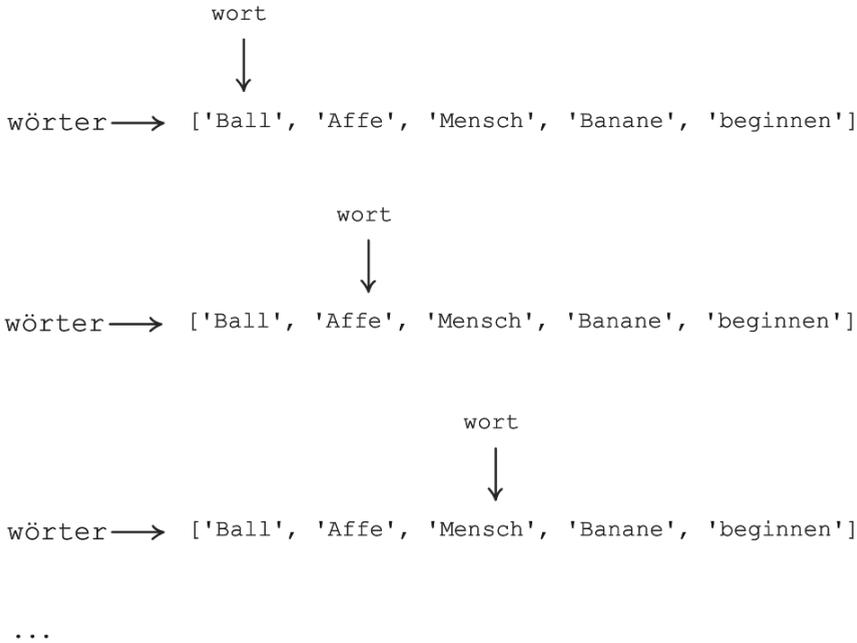


Abb. 1.6: Bei einer Iteration über eine Liste bezeichnet die Laufvariable nacheinander die Elemente der Liste.

Die Aufgabe »Nenne in einer Minute so viele Tiere wie möglich.« ist viel schwieriger zu programmieren, denn hier spielt die Bedeutung der Wörter (Semantik) eine Rolle. In Kapitel 2 wirst du KI-Techniken kennenlernen, wie man die Semantik von Texten erfassen kann.

Kollektion

Eine Kollektion ist eine Sammlung von Elementen (Items). Wichtige Typen von Kollektionen sind:

- Liste (Typ: `list`): eine änderbare Folge von beliebigen Elementen, die man in *eckige* Klammern schreibt, z.B. `[1, 17, 3, 1, 1]`
- Tupel (Typ: `tuple`): eine **nicht** änderbare Folge von beliebigen Elementen, die man in *runde* Klammern schreibt, z.B. `('Tom', 29)`
- String (Typ: `str`): eine Folge von beliebigen Zeichen in Anführungszeichen, z.B. `'Ball'` oder `"Ball"` oder `'01234'`
- Menge (Typ: `set`), eine Sammlung von beliebigen Elementen ohne bestimmte Reihenfolge, von denen jedes nur einmal vorkommt, in *geschweiften* Klammern, z.B. `{1, 17, 3}`

1.1.7 Wahrnehmungsgeschwindigkeit

Die Wahrnehmungsgeschwindigkeit (*perceptual speed*) ist die Fähigkeit, einfache visuelle oder symbolische Information schnell zu verarbeiten. Typische Aufgaben, die die Wahrnehmungsgeschwindigkeit testen, sind:

- Vergleichsaufgaben
- Zahlen-Suchaufgaben
- Aufgaben zur Bilderkennung (visuelle Muster finden)

Aufgabe 5: Zahlensuche

Welche Zahl enthält die Ziffernfolge 5738?

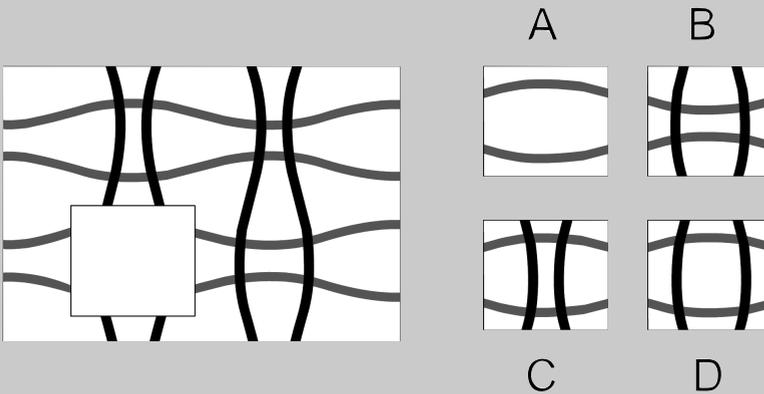
A: 266453557386623146

B: 551320034599583812

C: 443529384746594511

Aufgabe 6: Bilderkennung

Welches Bild passt in das weiße Rechteck?



Das Vergleichen und Finden von Symbolen ist leicht zu programmieren. Für Vergleiche verwendest du den Vergleichsoperator `==` (zwei Gleichheitszeichen). Damit kannst du z.B. Zahlen oder Texte auf Gleichheit testen. Das Ergebnis ist immer ein Wahrheitswert: `True` (wahr) oder `False` (falsch). Probiere in der IDLE-Shell einige Vergleiche aus:

```
>>> 129 == 129
True
>>> 1 == 0
False
>>> 'Baum' == 'Baum'
True
```

Mit dem Operator `in` kannst du prüfen, ob in Kollektionen bestimmte Elemente enthalten sind.

In den folgenden Anweisungen wird geprüft, ob in einem langen String ein kürzerer String enthalten ist:

```
>>> '5738' in '266453557386623146'
True
>>> '5738' in '551320034599583812'
False
```

Beachte, dass die Zahlen als Strings mit Anführungszeichen geschrieben sind. In diesen Python-Anweisungen werden also keine Zahlen, sondern Ziffernfolgen verarbeitet.

In Listen kann man leicht prüfen, ob ein bestimmtes Item enthalten ist:

```
>>> gruppe = ['Lucas', 'Maja', 'Johanna', 'Mike', 'Brick', 'Paul']
>>> 'Brick' in gruppe
True
```

Bildererkennung ist eine schwierigere Aufgabe für Computer. In Kapitel 4 entwickeln wir neuronale Netze, die Bilder von handgeschriebenen Ziffern erkennen können.

1.1.8 Starke und schwache KI

Wir haben einen Blick auf menschliche Intelligenz aus Sicht der Psychologie geworfen und festgestellt, dass moderne digitale Systeme tatsächlich einige Merkmale von Intelligenz besitzen.

Ein künstliches System, das menschliche Intelligenz komplett nachbildet und zu freiem Handeln fähig ist, nennt man *starke KI*. Von starker KI sind wir aber heute noch weit entfernt. Alle existierenden Techniken, von denen du einige in diesem Buch kennenlernen wirst, implementieren nur einige Facetten von intelligentem Handeln. Man nennt sie *schwache KI*. Sie sind für bestimmte Zwecke konstruiert und haben kein echtes Bewusstsein oder allgemeines Verständnis von der Welt.

1.2 Projekt: Wie schnell kann ein Computer rechnen?

Aaryan Shukla, der Weltmeister im Kopfrechnen, brauchte rund 96 Sekunden, um zehn zehnstelligen Zahlen zu addieren. Was schätzt du, wie viel Zeit dein Computer für diese Aufgabe benötigt? Wir entwickeln Schritt für Schritt ein Python-Programm, das zehn zehnstellige Zufallszahlen addiert und am Ende den Zeitbedarf ausgibt.

Vorbereitung

In der IDLE-Shell kannst du einzelne Python-Anweisungen ausführen. Wir wollen aber ein Programm schreiben, das aus mehreren Anweisungen besteht und als Programmdatei gespeichert werden kann. Dazu verwendest du den IDLE-Programmeditor. Öffne eine IDLE-Shell und klicke oben in der Menüleiste auf FILE|NEW FILE. Es öffnet sich ein neues Fenster, der IDLE-Programmeditor. Als Erstes speicherst du dein neues Programm in deinem Projektordner ab. Dazu klickst du auf FILE|SAVE AS. Gib deinem Programm einen sinnvollen Namen, z.B. `additionen_1.py`. Nach dem Speichern erscheint der Name oben im Rand des Editorfensters.

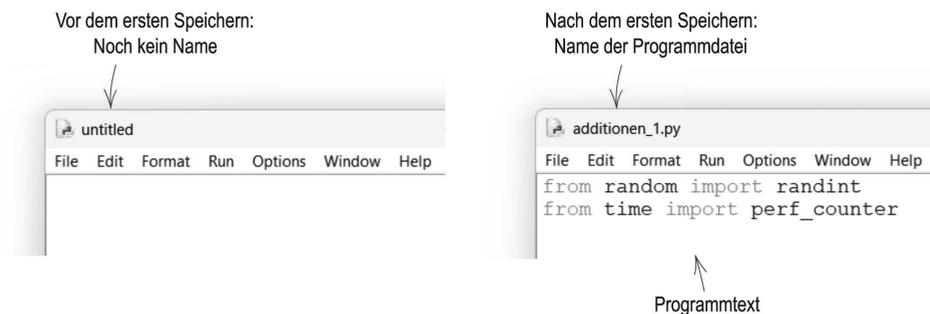


Abb. 1.7: Der Dateiname erscheint im Rahmen des Editorfensters.

Du schreibst ein Python-Programm, das Folgendes macht: Es erzeugt zuerst eine Liste von zehn zehnstelligen Zufallszahlen, merkt sich mit einer sehr genauen Uhr den Zeitpunkt, addiert alle zehn Zahlen und gibt am Ende den Zeitbedarf aus.

Schritt 1: Funktionen importieren

Dein Programm benötigt einige spezielle Funktionen, die nicht zum Python-Kern gehören und daher zuerst aus Modulen importiert werden müssen, bevor sie verwendet werden können.

Programm

```
# addition_1.py #1
from random import randint #2
from time import perf_counter #3
```

So funktioniert's

- #1: Die erste Zeile beginnt mit einem Hashtag #. Alles, was in einer Zeile hinter # steht, ist ein Kommentar und wird vom Python-Interpreter ignoriert. Kommentare sind für menschliche Leser bestimmt und sollen helfen, den Programmtext zu verstehen. In den Programmlistings in diesem Buch enthält der Kommentar in der ersten Zeile immer den Dateinamen des Programms. Das soll dir helfen, das Programm im Downloadmaterial zu finden. Auch die Zeichenfolgen #1, #2 und #3 sind Kommentare.
- #2: Aus dem Modul `random` wird die Funktion `randint()` importiert. Sie kann ganze Zufallszahlen erzeugen. Das Modul `random` enthält eine ganze Reihe von Zufallsfunktionen.
- #3: Aus dem Modul `time` wird die Funktion `perf_counter()` importiert. Sie ermöglicht sehr genaue Zeitmessungen. Das Modul `time` enthält Funktionen, die mit Datum und Uhrzeit zu tun haben.

Schritt 2: Eine Liste mit Zufallszahlen erzeugen

Wir erzeugen eine Liste mit zehn zehnstelligen Zufallszahlen. Eine Liste kann geändert werden. Man kann z.B. Elemente (*Items*) der Liste löschen oder ändern oder neue Elemente an die Liste anhängen. In unserem Programm bauen wir eine Liste auf, indem wir in einer Iteration an eine anfangs leere Liste nacheinander Zufallszahlen anhängen.

Programm

```
zahlen = [] #4
for i in range(10): #5
    zahl = randint(1000000000, 9999999999) #6
    zahlen.append(zahl) #7
```

So funktioniert's

- #4: In dieser Zuweisung wird eine leere Liste erzeugt und dem Namen `zahlen` zugewiesen. Wir haben also jetzt eine Liste, die den Namen `zahlen` trägt. Noch ist die Liste leer. Sie enthält kein einziges Element.
- #5: Bei der Iteration wird immer eine *Kollektion* von Elementen durchlaufen. In diesem Fall ist die Kollektion die Zahlenfolge 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Sie wird durch den Ausdruck `range(10)` erzeugt. Achte auf den Doppelpunkt : am Ende der Zeile. Er bedeutet, dass jetzt ein neuer Anweisungsblock folgt, der in der Iteration mehrmals ausgeführt wird. Der Anweisungsblock besteht aus den Zeilen #5 und #6 – sie müssen um die gleiche Stellenzahl eingerückt sein. So erkennt der Python-Interpreter, dass sie zusammengehören.
- #6: Durch den Aufruf `randint(1000000000, 9999999999)` wird eine ganze Zufallszahl zwischen 1.000.000.000 und 9.999.999.999 (jeweils einschließ-

lich) erzeugt und dem Namen `zahl` zugewiesen. Man sagt auch: Die Variable `zahl` enthält eine zehnstellige ganze Zufallszahl.

- #7: Listen sind änderbare Objekte und besitzen Methoden, die diese Änderungen durchführen können. In dieser Zeile findet ein Aufruf der Methode `append()` statt. Er bewirkt, dass an die Liste `zahlen` der Inhalt der Variablen `zahl` (also eine Zufallszahl) als neues Element angehängt wird. Methoden werden auf besondere Weise aufgerufen: Zuerst kommt der Name des Objekts (hier: `zahlen`), dann ein Punkt, dann der Name der Methode (hier: `append`) und dann in Klammern die Argumente der Methode. In diesem Fall gibt es nur ein Argument namens `zahl`.

Nach der `for`-Anweisung enthält die Liste `zahlen` zehn zehnstellige Zufallszahlen.

Schritt 3: Summe bilden und Zeitmessung

Programm

```

summe = 0                                #8
start = perf_counter()                   #9
for zahl in zahlen:                       #10
    summe += zahl                         #11
stop = perf_counter()                     #12
zeit = stop - start                       #13

```

So funktioniert's

- #8: Die Variable `summe` speichert das Ergebnis der Additionen. Die Variable erhält den Anfangswert 0. In einer Iteration werden dann nacheinander zehn Zufallszahlen addiert.
- #9: Der Aufruf `perf_counter()` liefert einen sogenannten *Zeitstempel*. Das ist eine Kommazahl wie z.B. 293463.942994, die eine gewisse Sekundenzahl angibt. Der Bezugspunkt – also der Zeitpunkt, von dem an diese Anzahl von Sekunden verstrichen ist – ist nicht definiert. Das macht aber nichts, weil wir nur an den Differenzen von Zeitstempeln interessiert sind. Das Programm speichert den Zeitstempel in der Variablen `start`.
- #10: Hier ist eine Iteration über die Liste `zahlen`. Die Laufvariable `zahl` nimmt nacheinander die Werte der Liste `zahlen` an.
- #11: Zum Inhalt der Variablen `summe` wird der Inhalt der Variablen `zahl` addiert. Die Anweisung hat die gleiche Wirkung wie `summe = summe + zahl`.
- #12: Der Aufruf `perf_counter()` liefert einen neuen Zeitstempel, eine Sekundenzahl. Diese Zahl wird in der Variablen `stop` gespeichert.
- #13: Der Zeitbedarf für die zehn Additionen wird berechnet.

Stichwortverzeichnis

A

Abhängigkeiten 62
Ableitung
 partielle 143
Accuracy 170
Adam-Optimierer 144, 167
Addition 298, 306
Ähnlichkeit 81
Aktivierungsfunktion 130
 auswählen 131
Arbeitsspeicher 18
 prüfen 18
Argument 29
Array
 Dimension 149
 Dimension ermitteln 149
 Dimension festlegen 154
 eindimensional 148
 erzeugen 147
 Form 152
 NumPy 147
 rechnen 150
 runden 151
 Spalten 153
 Transposition 154
 Typ 148
 umformen 153
 Zeilen 153
 zweidimensional 148
Attention 248
Attention-Matrix 248
Attribut 66
Aufmerksamkeit 248
Ausgabeschicht 128, 138, 167
Auskommentieren 309
Axon 125

B

Batch 138, 181
 Erstellung 304
 Größe festlegen 194
Batch-Gradientenabstieg 144
Baumann, Wilfried 108
BCE 140
Bedingung 36
 zusammengesetzte 39,
 40
BGD 144

Bias
 aktualisieren 141
Bibliothek 61
Bild
 entrauschen 291
Bilderkennung 163
Bildgenerierung 175, 191,
 206, 283
Bildsegmentierung 291
Bilineare Interpolation 298
Binärdatei 204
Binäre Kreuzentropie 140
Binary Crossentropy 140
Bottleneck 297
Byte 18

C

Cell State 225
Chatbot 35
 Projekt 85
ChatGPT 175, 217, 248
 Musik 260
Cho, Kyunghyun 245
Cleaning 234
Clustering 106
CNN 284, 289
 Vorteile 291
Code-Zelle 156
Colormap 189
Computer Generated Imaging 175
Convolutional Neural Network 284
Convolution Layer 285

D

DALL-E 283
Dataset-Objekt 301
Datei
 herunterladen 233
Dateiname 28
Daten
 speichern 17
Datenaugmentation 279
Datentyp 17
Datum und Uhrzeit 28
Decoder
 Faltung 298
Decoding 298, 300

Deduktion 21, 103
Deepfake 175
Deep-Learning-Architektur 248
Dendrit 125
Denken
 Analogie-Denken 21
 deduktives 21
 induktives 21
 schlussfolgerndes 20
Dense Layer 291
Dependencies 62
Deterministisch 215
Dictionary 71
 Splits 301
Diffusion 283
Diffusionsmodell 176, 283
 Grundidee 284
Diffusionsnetz
 Lernprozess 291
Diskriminator 33, 175
 programmieren 196
Doc-Objekt 66
Dropout 170, 244
Dropout-Schicht 170
Dur 267

E

Eigenschaft 66
Eingabeschicht 128, 137, 166
elif 39
Eliza 35
Embedding 79, 80, 81, 225,
 237, 238, 250, 273
Embedding-Schicht 237
Encoder 292
Encoding 299
Endlosschleife 34
Entropie 168
Entscheidungsbaum 102
Entwicklungsumgebung 14
Epoche 144
Erfahrung 103
Erkennungsmerkmal 285
Error-Backpropagation 293
Escape-Sequenz 56
Etikett 104

F

False 25
 Faltung 285, 286, 290, 294, 297, 300
 Bottleneck 297
 Training 295
 Faltungsnetz
 CNN 284
 Faltungsschicht 285, 295
 Knoten 296
 Farbtabelle 189
 Feature 285
 Feature-Extraktion 292
 Feature-Map 285, 288
 Berechnung 294
 mehrkanalige 295
 Fehler 95
 berechnen 97, 139
 Fehlerquadrat 99
 mittleres 100
 Figure 189
 Filter 286, 294
 Element 294
 trainieren 289
 for-Anweisung 22, 23
 Forget-Gate 226
 Formatstring 58
 Funktion 34
 definieren 45, 110
 Funktionskörper 46

G

GAN 33, 175
 Aktivierungsfunktion 134
 Arbeitsweise 177
 trainieren 198
 Verlust 199
 Gate 225, 245
 Gated Recurrent Unit 245
 Gatter 225
 Gauß, Carl Friedrich 183
 Gaußsche Glockenkurve 183
 Gedächtnis 15, 71, 176, 220, 222
 Gehirn 18, 125
 Genauigkeit 139
 Generalisierung 170
 Generative Adversarial Network 134, 175
 Generative KI 33, 175, 217
 Generator 175
 programmieren 195
 speichern 204
 Gerichteter Graph 127
 Geschwindigkeit 146
 Gewicht 72, 73, 74, 128, 129
 aktualisieren 141

Aufmerksamkeitsgewicht 248
 negativ 130
 Shared Weights 291

Gleichgewichtete Linearität 133
 Glockenkurve
 gaußsche 183
 Goethe 82
 Google 248
 Google Brain Team 155
 Google Colab 156
 Code-Zelle 156
 Gradient 142, 224
 Explosion 223
 Verschwinden 223, 224, 225
 Gradientenabstieg 144, 224
 stochastischer 144
 Gradientenverfahren 141
 Graph
 gerichteter 127
 Groß- und Kleinschreibung 246
 GRU 245
 TensorFlow 246
 Grundform 66
 Guard Clause 87

H

Head 250
 Helligkeitswert 190
 Hidden State 222, 225
 Hochreiter, Sepp 225
 Hopfield, John 221
 Hugging Face 207
 Hyperbolische Tangensfunktion 134

I

IDLE 14, 27
 if-Block 36
 if-else-Anweisung 36
 verschachtelte 38
 Imitation Game 32
 Import 147
 Index 56, 270
 Induktion 21
 Input-Gate 226
 Instrument 259
 Intelligenz 11
 Primärfaktoren 12
 Interpolation
 bilineare 298
 Intervall 266
 Item 28, 71
 Iteration 22, 23, 24, 99

J

Jupyter Notebook 156

K

Kaggle 207
 Kanal 295
 Kandidat 226
 Kanten erkennen 288, 289
 Kategoriale Kreuzentropie 141, 168
 Kernel 286
 Key-Objekt 267
 KI
 generative 33, 175, 217
 schwache 26
 starke 26
 subsymbolische 11, 289
 symbolische 11
 Klassifikation 77, 102, 140, 163
 KNN 127, 129
 Knoten 128, 129
 Zustand speichern 222
 Kollektion 24
 Kommentar 28
 Kommentarzeichen 194
 Kompilieren 206
 des Modells 160
 Komplexität 244
 Konditionieren
 operantes 106
 Konstante 34
 Konvolution 285
 Kreuzentropie
 binäre 140
 kategoriale 141, 168
 sparsame kategoriale 141
 Künstliches neuronales Netz 127
 Knoten 128
 Schichten 128

L

Label 104, 138
 Lambda 305
 Large Language Model 217
 Laufvariable 23, 24
 Lazy Evaluation 305
 Lemmatisierung 65, 66
 Lernarten 93
 Lernen 95
 überwachtes 93, 102
 unüberwachtes 93, 105, 232, 235
 Verstärkungslernen 93
 Lernerfolg dokumentieren 116
 Lernrate 96
 Lernschritt 95, 97

- Lineareinheit
 gleichgewichtete 133
 Lineare Regression 100
 Liste 24, 28
 LLM 217
 Logarithmus 244
 Logische Operation 135
 Logischer Operator 40
 Logisches Schließen 103
 Logits
 unnormalisierte 243
 Log-Wahrscheinlichkeit 243
 Long Short-Term Memory 220
 Loss 139
 LSTM 225
 Arbeitsweise 225
 Kandidat 226
 Schicht hinzufügen 245
 trainierbare Parameter 226, 227
 Training 227
 Zustand 225
- M**
- Mad Libs 59
 matplotlib.pyplot 188
 Matrix 146, 286
 zweidimensionale 149
 Max-Pooling 296, 297, 299
 Mean Squared Error 139
 Measure 262
 Melodie
 definieren 257
 speichern 258
 Menge 24
 Merkmal 102
 Methode
 aufrufen 29
 Metrik 160
 MIDI-Datei 255
 Download 276
 Mischpuffer 304
 MIT 35
 Mittelwert 183
 Mittlerer quadratischer Verlust 139
 MNIST-Datenbank 163, 192
 MNIST-Datensatz 300
 Modell 125, 129
 Definition anpassen 244
 generatives 237
 laden 204
 speichern 204
 trainieren 161, 239
 Training vorbereiten 160
 vortrainiertes 207
 Modul 27
 Moll 267
- MSE 139
 Multi-Head Attention 250
 MuseScore 256
 music21 255
 installieren 255
 Musik
 Elemente 259
 mehrstimming 260
 Modell definieren 272
 modellieren 255
 Training 263
- N**
- Nächster Nachbar 298, 306
 Named Entity Recognition 66
 Natural Language Processing 61
 Natural Language Toolkit 61
 Netz
 künstliches neuronales 127
 neuronales 125
 semantisches 76
 Neuron 125
 Neuronales Netz 125
 definieren 165
 entwerfen 135
 Training vorbereiten 160
 Neurotransmitter 126
 Nichtdeterminismus 120
 Nimrod 42
 Nim-Spiel 42
 NLP 61, 64
 Projekt 70
 NLP-Pipeline 65
 NLTK 61
 Normalisieren 234, 251, 304
 Note 256
 Länge 257
 Notebook 156
 Notensatzprogramm 256
 NumPy 125, 146
 Array 146
 NVIDIA 176
- O**
- Objekt 66
 Identität 268
 iterierbares 194
 sequential 197
 Objekterkennung 163
 One-Hot-Codierung 165
 Operantes Konditionieren 106
 Operator
 logischer 39, 135
 Optimierer 144, 160
 Adam 144
 Optimierungsproblem 141
- os 277
 Oszillation 201
 Output-Gate 227
 Overfitting 170
- P**
- Padding 235, 288
 Papert, Seymour 108
 Parameter 138, 141
 Parameterliste 45
 Part 261
 Partitur 261
 Part-of-Speech-Tagging 65
 Patch 286, 294
 Peripheriespeicher 18
 Pipeline 198
 Platzhalter 58
 Namen 59
 Pooling 296
 Prefetching 304
 Preprocessing 234, 246
 ProGAN 209
 Fehler 210
 Projekt
 Ähnliche Wörter finden 82
 Fitness-Chatbot 85
 Freundliche Maschine 33
 KI als Spielgegner 42
 Kreative Ziffern 191
 Künstliche Gesichter 206
 Künstliche Ziffern 300
 Melodien komponieren 267
 Stichwort-Chatbot 35
 Storytelling 59
 Textgenerator 229
 Text zusammenfassen 70
 Uneinigkeit erkennen 155
 Versuch und Irrtum 107
 Wie schnell kann ein Computer rechnen 27
 Ziffern erkennen 163
 Projekt Gutenberg 231
 Python
 Argument 29
 Bibliothek 61
 installieren 13
 Kommentar 28
 Liste 22, 26
 Methode 29
 Modul 27
 Programm starten 30
 Schlüsselwort 23
 Standardausgabe 30
 Zahlentypen 15
 Zuweisung 16

PyTorch 208

R

RAM 18

Räumliches Vorstellungsvermögen 19

Rauschen 176, 178, 283, 291

Rauschvektor 195

Reasoning 20

Rechnen 12

Recurrent Neural Network 176

Region 296

Regression 93, 140, 160
lineare 100

Regressionsgerade 94

Reinforcement Learning 106

Relevanz 74

ReLU 133, 196

Reset-Gate 245

RL 106

RNN 176, 221

Funktionsweise 220

Gewichte 223

Training 225

Rückkopplung 221, 310
direkte 221**S**

Satzzeichen 246

SCCE 167

Schicht 137

verborgene 167

vollständig verknüpfte 135

Schleife 40

Schließen

logisches 103

Schlüsselwort 72

Schlüsselwortargument 60

Schlussfolgerndes Denken 20

Schmidhuber, Jürgen 225

Schrittweite 289

Schwache KI 26

Schwellenwert 74

Schwellenwertfunktion 131

Score 255, 261

Self-Attention 248, 249

Semantik 77

Semantisches Netz 76

Sequential-Objekt 197

Sequenz 236, 271

SGD 144

Shukla

Aaryan 13

Sigmoid 167

Sigmoid-Funktion 132, 138,
226

Skalar 150

Skip-Connection 250, 292,
296, 298, 300

Slice 187, 236

Sliding Window 271

Softmax 134, 167, 238

Sonderzeichen 56

spaCy 53, 61, 77

installieren 62

Spam 106

Spam-Filter 129

Sparsame kategoriale Kreuz-
entropie 141Sparse Categorical Cross-
entropy 167, 239

Speicher 18

Einheiten 18

Spracherkennung 129

Sprachmodell 61, 62, 217, 229

Sprachverarbeitung 61, 64

Sprachverständnis 20

Standardabweichung 183

Starke KI 26

Statistik 101

Steigung 131, 142

Stochastischer Gradienten-
abstieg 144

Stoppwort 68

Stream 255

Stream-Objekt 256

Stride 289

String 20, 24, 53

durchsuchen 56

erstellen 53

Formatierung 58

konkateneren 54

Platzhalter 58

verändern 57

Stringformatierung 58

Struktur

komplexe 297

StyleGAN2 176

Subsymbolische KI 11, 289

Swish-Funktion 132, 294

Symbolische KI 11

Synapse 127

T

Takt 261

Taktart 257, 259

Tangensfunktion

hyperbolische 134

tanh 134, 193, 196

Target 284

Tempo 259

Tensor 146, 178

Batch 181

Bilder darstellen 187

Datentyp 179

Dimension 179

Dimension erweitern 185

erstellen 194

erzeugen 178, 182

Form 181

Form anpassen 184

in Array umwandeln 182

Index 186

Rang 179

Slice 187

Verarbeitung 178

Zufallszahlen 183

Zugriff auf 186

TensorFlow 146, 155, 220, 305

importieren 178

installieren 155

Modell speichern 204

Modell trainieren 160

Tensoren verarbeiten 178

TensorFlow Hub 206

Terminalfenster

schließt sich automatisch
31

Testdaten 104

Text

generieren 176, 229, 240

Produktion 221

zerlegen 69

zufällig generieren 243

zusammenfassen 70

Thurstone, Louis Leon 12

Token 64, 225, 231

Attribut 66

Liste erstellen 233

Relevanz 74

voraussagen 237

Tokenizer 246

Tokenizing 64, 234

Token-Liste 234

Tonart 259

ändern 266

bestimmen 266

Tonika 275

Training 95, 138, 169

Ablauf 138

Argumente 161

Fortschrittmeldung 169

Musik generieren 273

Trainingsdaten 138, 301

vorbereiten 301

Trainingsdatensatz 128

Trainingszeit 278

Transformer 176, 248

Modell trainieren 251

Transponieren 266

Transposition 153

True 25

Tupel 24

Turing, Alan 31

Turing-Test 31

Turtle-Grafik 108

U

Überwachtes Lernen 93, 102, 128
 Umgebung
 virtuelle 62
 U-Net 283, 291, 293, 299
 Aufbau 292
 Modell erstellen 305
 testen 307
 trainieren 292
 Training 307
 Unicode 55
 Universal Part-of-Speech
 Tagset 68
 Unnormalisierte Logits 243
 Unüberwachtes Lernen 93, 105, 232, 235
 Update-Gate 245
 Upsampling 298, 300

V

Vanishing Gradient 225
 Variable 16
 abhängige 95
 Name 17
 unabhängige 95
 Vektor 77

Verborgene Schicht 128, 137, 167
 Vergleiche 40
 Vergleichsoperator 25
 Verlustfunktion 139, 160
 Verstärker 120
 Verstärkungslernen 93, 106, 107
 Versuch und Irrtum 107, 117
 Verstärker 120
 Virtuelle Umgebung 62
 Vorhersage 95, 138
 berechnen 137
 Vorstellungsvermögen
 räumliches 19
 Vorverarbeitung 234

W

Wächter-Klausel 87
 Wahrheitswert 25, 135
 Wahrnehmungsgeschwindigkeit 25
 Wahrscheinlichkeit 138
 in Logarithmen umrechnen 243
 Wahrscheinlichkeitsverteilung 167, 230
 Weg
 kürzester 117

Wegesuche 117
 Weizenbaum, Joseph 35
 while-Anweisung 40
 Williams, Roland J. 221
 Wortart 67
 Wort-Embedding *siehe* Embedding
 Wortflüssigkeit 22

Z

Zeichenkette *siehe* String
 Zeichnen 109
 Zeilenumbruch 56
 Zeilenvektor 152
 Zeitmessung 28
 Zeitstempel 29
 Zelle 222
 Zustand 225
 Zustand aktualisieren 227
 Ziffern
 erkennen 129, 163
 generieren 300
 Zipser, David 221
 Zufälligkeit 120
 Zufallszahl 28
 erzeugen 28
 Zustand 222
 Zuweisen 29