Michael Weigend

KI-Projekte programmieren mit Python

Dein einfacher Einstieg

Generative KI, Chatbots, Objekterkennung & Co.

mitp

Inhaltsverzeichnis

Kapitel 1

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.

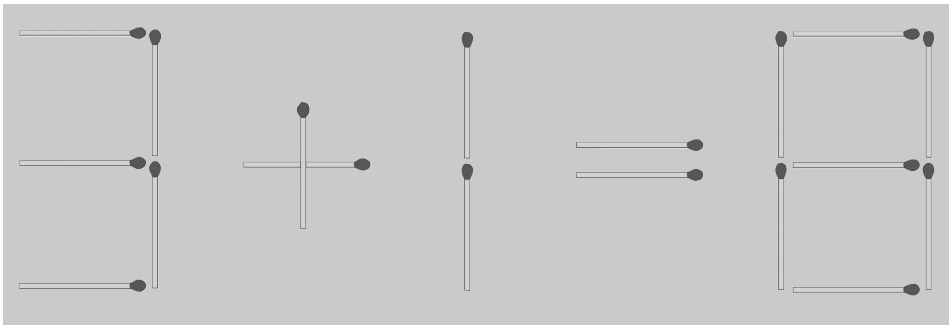
Im Rest des Kapitels werden einige Programme erläutert, die auf diesem KI-Verständnis (»KI als Ersatzmensch«) basieren: Chatbots, mit denen man sich unterhalten kann oder die von sich aus Menschen ansprechen, und ein Spiel mit einer KI als Spielgegner (»Nim«). Alle in diesem Kapitel beschriebenen Programme 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
```

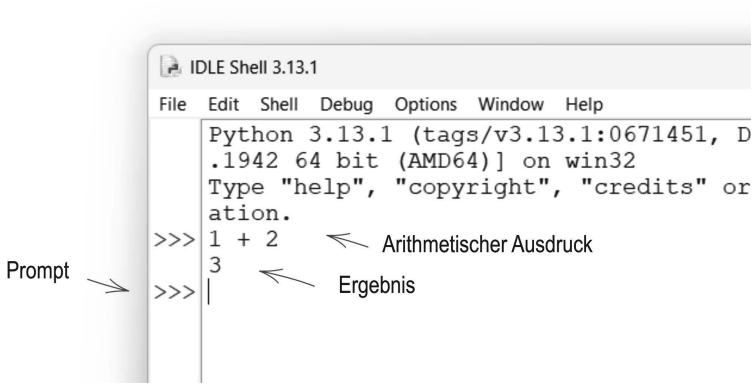


Abb. 1.2: Die IDLE-Shell als Taschenrechner

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 = 10³ = 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).

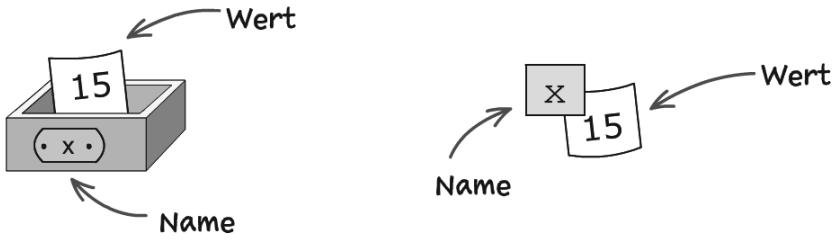


Abb. 1.3: Zwei Veranschaulichungen für Variablen

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.

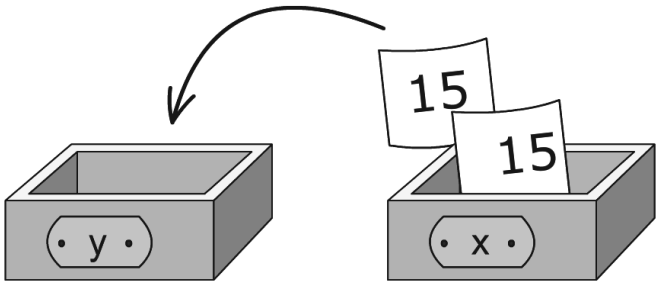


Abb. 1.4: Wertübertragung

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 Variablennamen 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 Variablennamen 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.

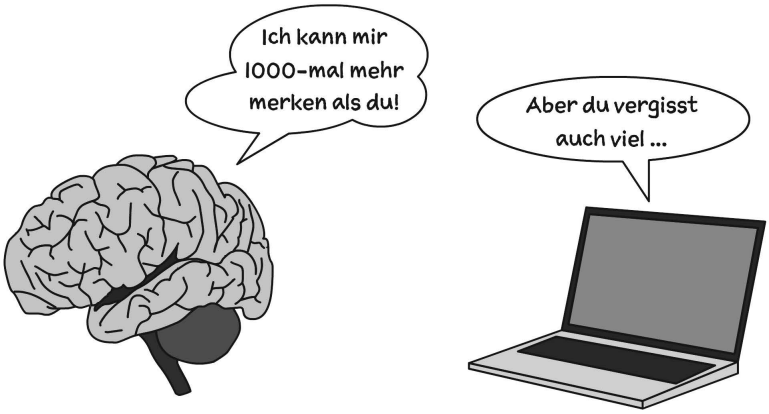
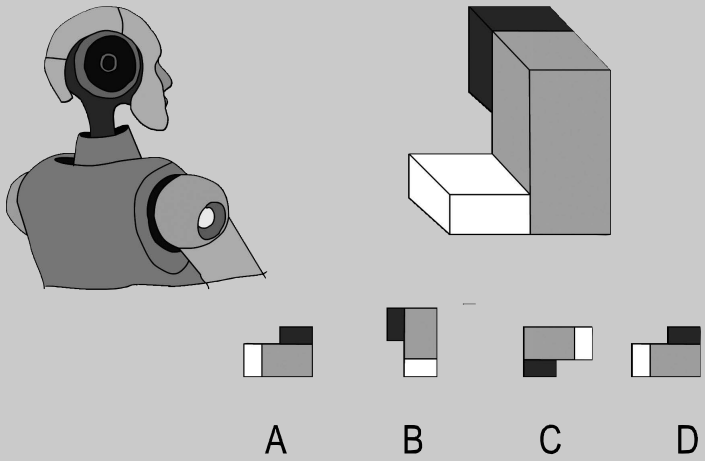


Abb. 1.5: Der Computer hat Speicher – das Gehirn hat Erinnerungen.

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?



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

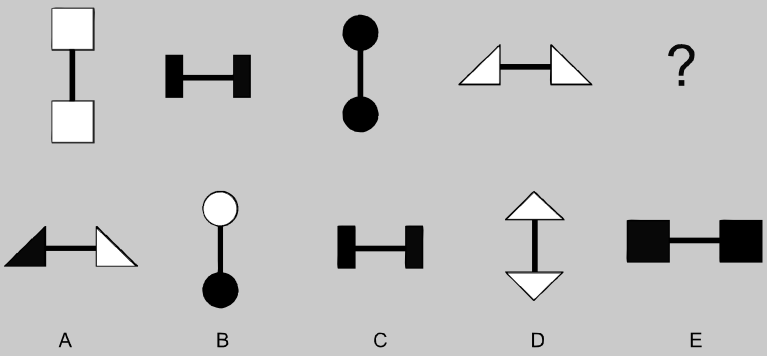
Schlussfolgerndes Denken *(reasoning)* ist die Fähigkeit, Regeln und Muster zu erkennen, logische Zusammenhänge herzustellen und aus gegebenen Informationen neue Erkenntnisse abzuleiten. Es gibt mehrere Formen des schlussfolgernden Denkens:

- 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)
```

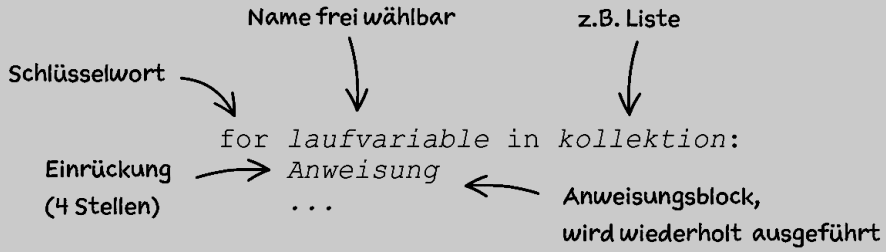
Die Ausgabe erscheint in den nächsten Zeilen:

```
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 ...

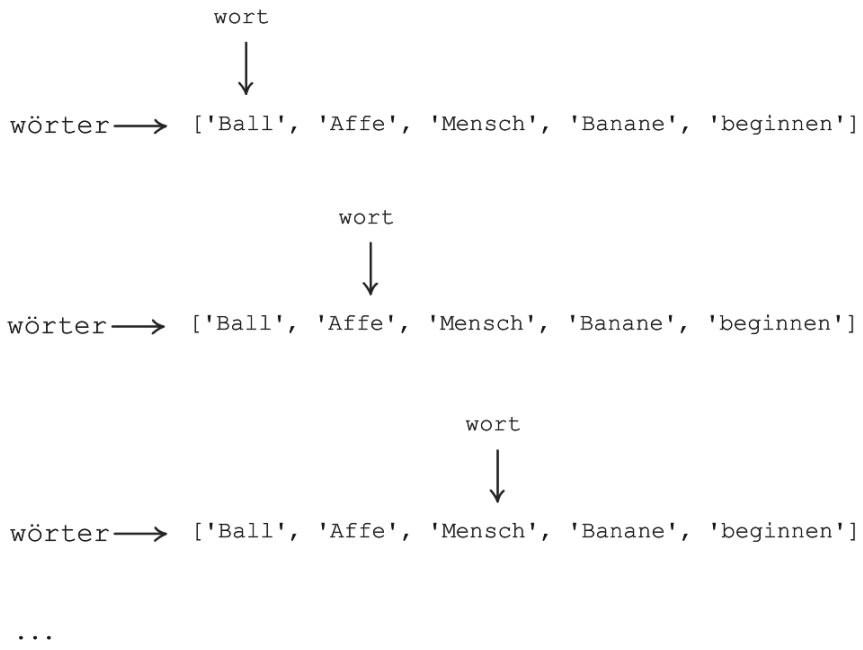


Abb. 1.6: Bei einer Iteration über eine Liste bezeichnet die Laufvariable nacheinander die Elemente der Liste.

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**?

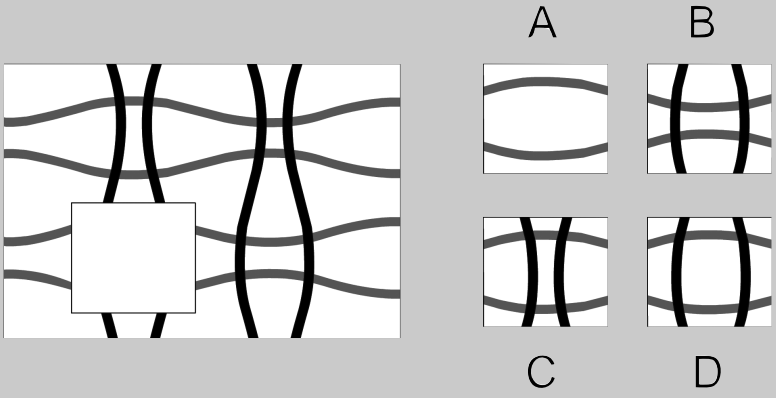
A: 266453557386623146

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
```

Bilderkennung 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.

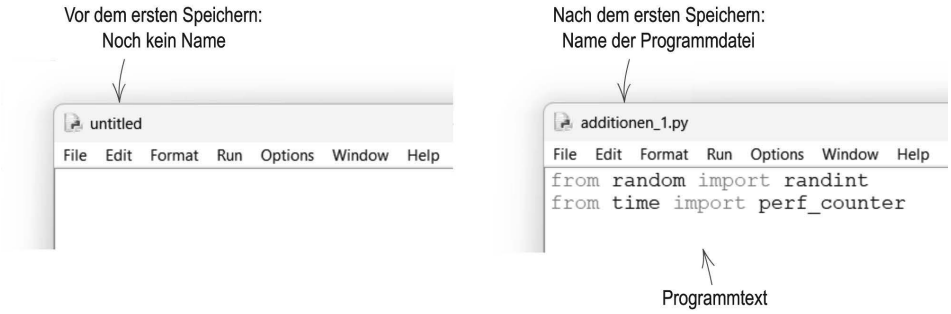


Abb. 1.7: Der Dateiname erscheint im Rahmen des Editorfensters.

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