

Python verstehen

Dein Lernweg von Null auf Code

Das Playbook für Neugierige, die mit KI auf Augenhöhe sprechen wollen. Du kennst die Theorie. Jetzt lernst du die Sprache.

powered by AI: Recherche, Iteration, Kuration & Redaktion: Carmen Frohne | September 2026

www.carmen.coach

on(:);

n(:);

ge 10;

ge 10;

for i in range 10;

fnunctione(:);

for i in range 10)

nowarling;

#

Wie dieses Playbook aufgebaut ist

Dieses Playbook ist kein Lehrbuch. Es ist dein persönlicher Reiseführer durch Python – Schritt für Schritt, mit echten Beispielen aus dem Alltag. Jedes Kapitel folgt demselben Rhythmus: eine Alltagsszene öffnet die Tür, eine Erklärung führt dich hindurch, ein Workbook-Moment lädt dich ein, das Gelernte sofort anzuwenden.



Alltagsszene

Jedes Kapitel beginnt mit einer echten Situation. Du erkennst das Prinzip, bevor du den Code siehst.



Workbook-Moment

Kurze Aufgaben, die dich ins Tun bringen. Kein Warten, kein Aufschieben – gleich hier.



Erklärung

Klare Sprache, keine Fachbegriff-Lawinen. Du verstehst, was passiert – und warum.



Kapitel-Abschluss

Ein kurzes Fazit verankert das Wichtigste. Du nimmst immer etwas Konkretes mit.

Was ist Python? – Die Wurzeln

Python ist eine Programmiersprache. Sie wurde 1991 von Guido van Rossum veröffentlicht – inspiriert von der britischen Komikertruppe Monty Python, nicht von der Schlange. Van Rossum wollte eine Sprache, die sich liest wie ein Brief an einen guten Freund: klar, direkt, ohne unnötiges Rauschen.

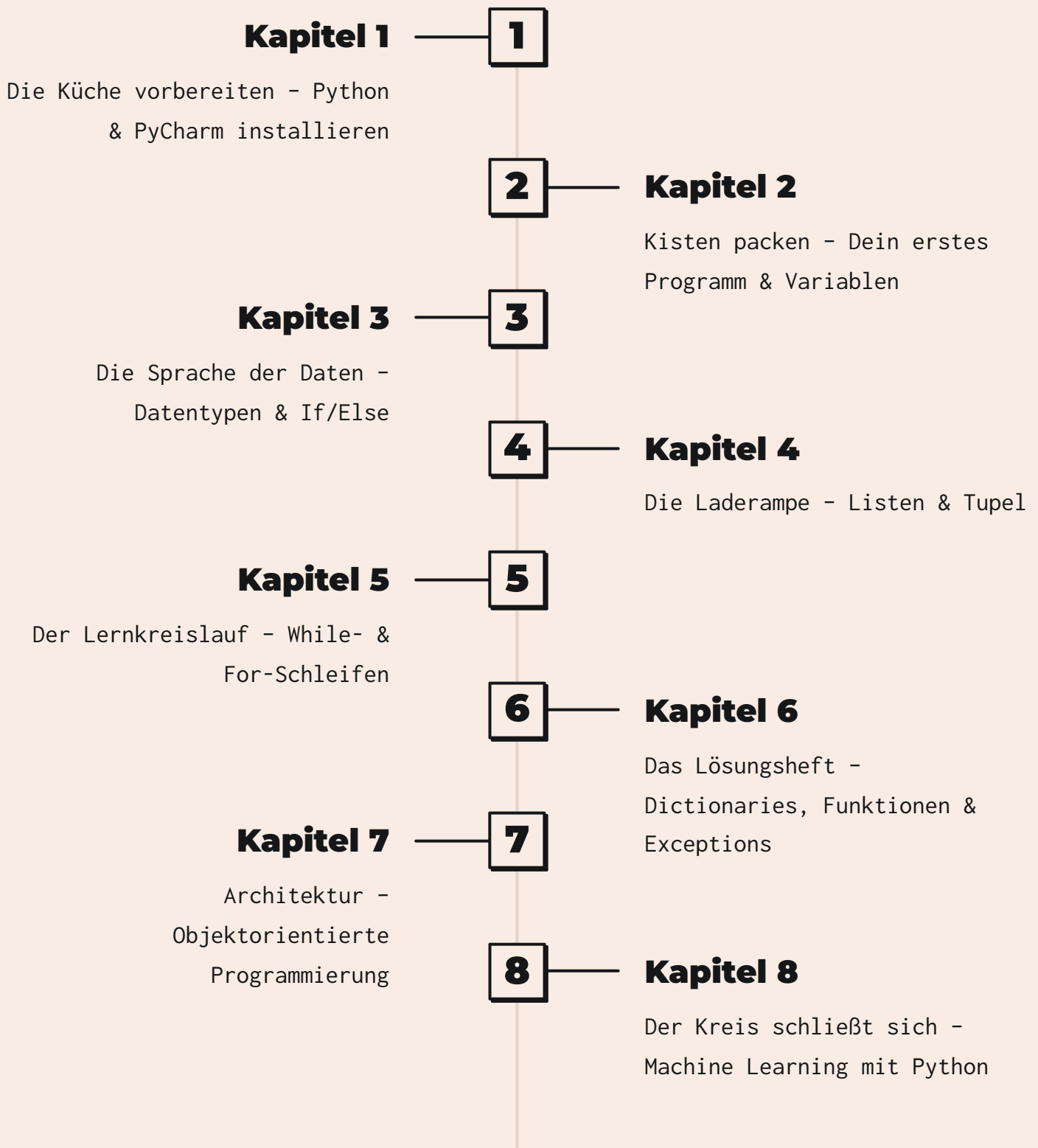
Heute ist Python die meistgenutzte Programmiersprache der Welt. Data Scientists in Tokio, Lehrerinnen in Lagos, Forscher in Berlin – alle schreiben Python. Nicht weil es hip ist. Sondern weil es funktioniert. Weil es verständlich ist. Weil es offen ist für alle.

Python ist besonders für KI und maschinelles Lernen der Standard. Fast alle großen KI-Bibliotheken – TensorFlow, PyTorch, scikit-learn – sind in Python geschrieben oder über Python bedienbar. Wer KI verstehen will, lernt Python.



Roadmap – Alle Kapitel auf einen Blick

Hier ist deine Reiseroute. Acht Kapitel, ein Ziel: Du verstehst Python so gut, dass du eigene KI-Projekte starten kannst.



Die Küche vorbereiten

Warum Python, Installation & PyCharm

Eine Szene aus Bogotá

Es ist früh morgens in Bogotá. Valentina Restrepo - 34 Jahre alt, Sozialarbeiterin - sitzt in ihrer kleinen Küche. Auf dem Tisch: ein Notizbuch, ein Laptop, eine Tasse Tinto. Heute möchte sie zum ersten Mal eine Zeile Python schreiben. Sie hat keine Ahnung, wo sie anfangen soll. Die Küche ist ihr sicherer Ort. Also fängt sie hier an.

Valentina macht das, was alle guten Köchinnen tun: Sie richtet ihre Küche ein, bevor sie loskocht. Sie legt die Messer bereit, schaltet den Herd ein, räumt die Arbeitsfläche frei. Für Python bedeutet das: zuerst die Werkzeuge installieren, dann losstarten.

Diese Geschichte ist deine Geschichte. Du brauchst keine Vorkenntnisse. Du brauchst nur einen Computer, eine Internetverbindung und die Bereitschaft, loszulegen.



Python installieren – Schritt für Schritt

Python herunterladen ist einfach. Geh auf **python.org**, klick auf „Download“ – der Knopf springt dir direkt entgegen. Wähle die neueste stabile Version. Beim Installieren auf Windows: Hake das Kästchen „Add Python to PATH“ an. Das ist wichtig. Auf Mac und Linux ist Python oft schon vorinstalliert.

1

python.org besuchen

Neueste stabile Version
herunterladen

2

Installieren

„Add to PATH“ ankreuzen – nicht
vergessen!

3

Terminal öffnen

`python --version` eintippen und
prüfen

4

Fertig!

Python ist bereit. Die Küche
steht.

PyCharm – Deine Küche

PyCharm ist eine Entwicklungsumgebung – auf Englisch: IDE (Integrated Development Environment). Das klingt groß. Gemeint ist: ein Programm, das dir beim Schreiben von Code hilft. Es zeigt dir Fehler sofort an, schlägt Ergänzungen vor und macht deinen Code farblich lesbar.

Du kannst dir PyCharm wie eine professionelle Küche vorstellen. Alles liegt griffbereit. Messer, Töpfe, Herd – alles hat seinen Platz. Wenn du einen Fehler machst, piepst die Küche. So lernst du schneller.

Geh auf jetbrains.com/pycharm und lade die kostenlose „Community Edition“ herunter. Die reicht vollkommen für alles, was wir in diesem Playbook tun.





Workbook-Moment: Dein Antrieb

Bevor wir den Herd einschalten: Warum bist du hier? Wähle, was am meisten auf dich zutrifft – oder schreibe deine eigene Antwort.

Grund A

Ich möchte KI-Modelle nicht nur verstehen, sondern selbst bauen.

Grund B

Ich will Routineaufgaben in meinem Alltag automatisieren.

Grund C

Ich lerne einfach gerne neue Sprachen – ob Spanisch, Python oder Japanisch.

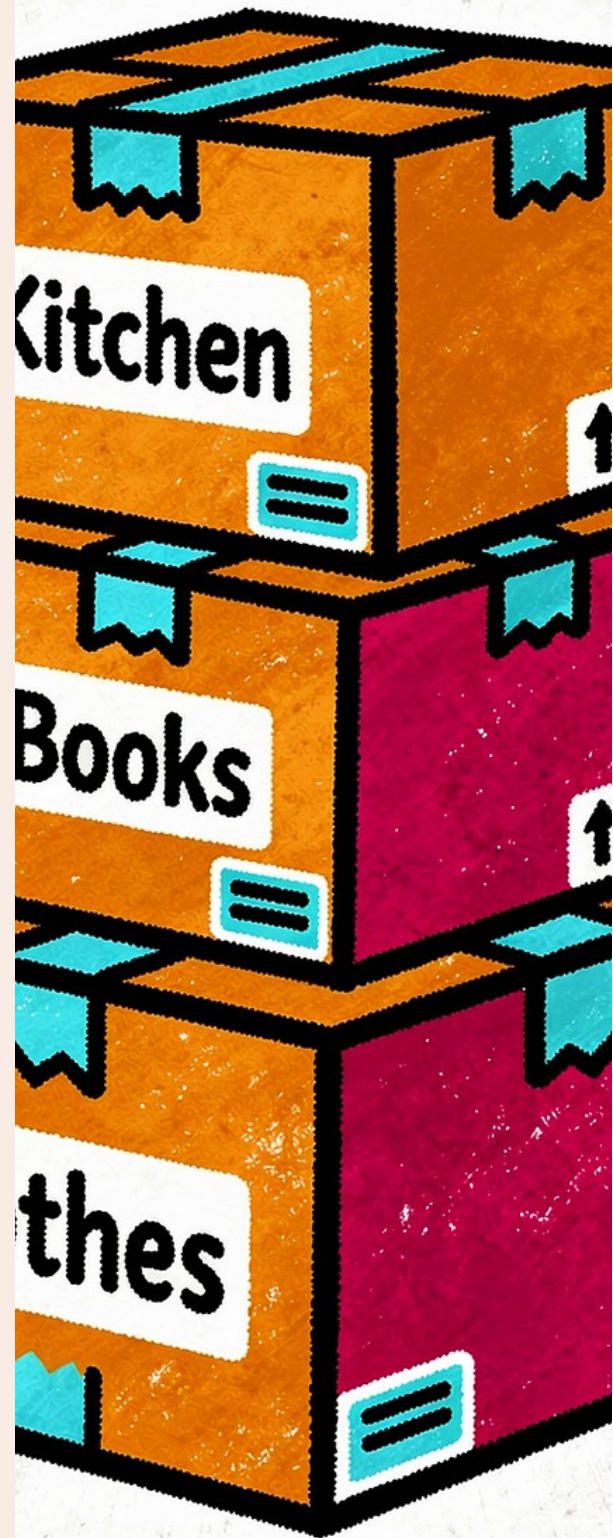
Meine eigene Antwort



Dein Antrieb ist dein Kompass. Schreib ihn auf. Lies ihn an schwierigen Tagen noch einmal durch.

Kisten packen und beschriften

Dein erstes Programm & Variablen



Eine Szene aus Nairobi

Kamau Mwangi - 28 Jahre alt, Grafikdesigner in Nairobi - zieht um. Sein Zimmer ist voller Kisten. Auf jeder Kiste steht ein Name: „Bücher“, „Küchenutensilien“, „Ladekabel“. Er weiß sofort, wo alles ist. Er muss nicht jede Kiste öffnen, um die Gitarre zu finden - er liest einfach das Etikett.

Genau das macht Python mit Variablen. Du gibst einer Information einen Namen. Python merkt sich den Inhalt. Wenn du den Namen später abrufst, bekommst du den Wert zurück. Kein Suchen, kein Raten.

Kamau schreibt auf die erste Kiste: `gitarre = "Yamaha"`. Auf die zweite: `gewicht_kg = 2.5`. Auf die dritte: `bereits_gepackt = True`. Das ist Python - und das ist sein erstes Programm.



Hello, World! – Dein erstes Programm

Das erste Programm in Python lautet in der ganzen Welt gleich. Von Seoul bis São Paulo, von Helsinki bis Harare. Es ist eine Tradition, ein Ritual, ein Willkommen:

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

Du schreibst diese eine Zeile. Du drückst Enter. Der Computer gibt aus: Hallo Welt!. Das klingt klein. Es ist riesig. Du hast gerade dem Computer befohlen, etwas zu tun – und er hat gehorcht. Das ist Programmieren.

print() ist eine Funktion. Sie sagt Python: „Zeig das hier auf dem Bildschirm.“ Was in den runden Klammern und Anführungszeichen steht, wird ausgegeben. Du kannst alles reinschreiben: `print("Ich lerne Python!")` oder `print("Kamau packt Kisten.")`

Variablen – Die beschrifteten Kisten

Eine Variable ist ein Name für einen Wert. Du wählst den Namen. Python speichert den Wert dahinter. Später rufst du den Namen ab – und bekommst den Wert zurück. Kein Suchen, kein Raten.

```
tageszeit = "Mittags"  
hunger_level = 8  
kaffee_bereit = True
```

Drei Variablen. Drei Kisten. Jede hat ein Etikett. Jede hat einen Inhalt. Das Gleichheitszeichen (=) ist der Edding-Strich: Es weist den Wert der rechten Seite dem Namen der linken Seite zu.



Workbook-Moment: Deine eigenen Kisten

Schau dich in deinem Zimmer um. Was siehst du? Erstelle drei Python-Variablen für drei Dinge, die direkt vor dir liegen. Das erste Beispiel gehört dir gratis.

Beispiel (schon ausgefüllt)

```
kaffee_temperatur = "heiß"
```

Deine Variable 1

```
_____ =  
_____
```

Deine Variable 2

```
_____ =  
_____
```

Deine Variable 3

```
_____ =  
_____
```

Die Sprache der Daten

Strings, Zahlen,

Booleans & If/Else

Eine Szene aus Istanbul

Defne Yıldız - 31 Jahre alt,
Wetterforscherin an der Universität
Istanbul - schaut jeden Morgen aus dem
Fenster. Sie stellt sich drei Fragen: Wie
warm ist es? Wie viele Grad? Regnet es - ja
oder nein? Drei Fragen. Drei verschiedene
Arten von Antworten.

Die erste Antwort ist ein Wort: „warm“. Die
zweite ist eine Zahl: 22.5. Die dritte ist
eine Entscheidung: Ja oder Nein. Defne weiß
intuitiv: Nicht alle Informationen sind
gleich. Ein Wort ist etwas anderes als eine
Zahl. Eine Ja/Nein-Antwort ist etwas
anderes als beides.

Python denkt genauso. Es gibt verschiedene
Arten von Daten - Datentypen. Jeder Typ hat
seine Stärken. Jeder passt zu bestimmten
Aufgaben. Defnes drei Fragen führen uns
direkt zu den vier wichtigsten Python-
Datentypen.



Die vier Grunddatentypen

String (str)

Text in Anführungszeichen.

"warm"

"Käsehäppchen"

"Hallo Defne!"

Integer (int)

Ganze Zahlen ohne Komma.

22

100

-5

Float (float)

Zahlen mit Komma (Punkt!).

22.5

3.14

0.001

Boolean (bool)

Nur zwei Werte: wahr oder falsch.

True

False

Booleans kennst du bereits aus dem letzten Playbook: One-Hot-Encoding hat „Warm“ in 1 (True) und 0 (False) übersetzt. Python macht dasselbe – nur mit Worten statt Zahlen.

If / Else – Die kleine Entscheidung

Ein einzelnes künstliches Neuron entscheidet: Feuert es oder nicht? Ja oder nein? Genau das macht **If/Else** in Python. Es ist die einfachste Form von Logik – und die mächtigste.

```
regenwahrscheinlichkeit = 75
```

```
if regenwahrscheinlichkeit > 60:  
    print("Schirm mitnehmen!")  
else:  
    print("Sonnenbrille reicht.")
```

Python liest das so: „Wenn die Regenwahrscheinlichkeit größer als 60 ist – zeige die Schirm-Nachricht. Sonst – zeige die Sonnenbrille-Nachricht.“ Nur einer der beiden Wege wird gegangen. Niemals beide.

Achte auf den Doppelpunkt nach `if` und `else`. Und achte auf die Einrückung (vier Leerzeichen). Python misst Struktur durch Abstände. Das ist seine Art, Ordnung zu halten.





Workbook-Moment: Dein Alltags-Algorithmus

Denk an deinen heutigen Morgen. Welche If/Else-Entscheidungen hast du getroffen? Fülle die Lücken mit deinen echten Morgen-Situationen.

Situation A – Wecker

Wenn der Wecker
klingelt und es
vor 7 Uhr ist,
dann:

Sonst:

Situation B – Kaffee

Wenn die
Kaffeemaschine
fertig ist und ich
Zeit habe,
dann:

Sonst:

Situation C – Deine eigene

Wenn

_____,

dann:

Sonst:

Die Laderampe und Pakete

Listen & Tupel

Eine Szene aus Hanoi

Linh Nguyễn - 26 Jahre alt,
Softwareentwicklerin in Hanoi - kauft jeden
Samstagmorgen auf dem Bến Thành-Markt ein.
Sie macht eine Liste: Rambutan,
Drachenfrucht, Kokosnuss, Mango. Eine
geordnete, veränderliche Sammlung. Wenn die
Kokosnüsse aus sind, streicht sie sie durch
und schreibt Papaya dazu.

Das ist eine Python-Liste. Eine geordnete
Sammlung von Werten. Du kannst etwas
hinzufügen, etwas entfernen, etwas
verändern. Die Liste passt sich an - so wie
Linh's Einkaufszettel.

Im letzten Playbook haben wir Vektoren als
das Datenpaket für einen Moment
kennengelernt - eine geordnete Liste von
Zahlen. In Python nennen wir genau das eine
Liste. Und Linh's Einkaufszettel ist ein
perfektes Beispiel dafür.



Listen – Geordnet und veränderlich

Listen stehen in eckigen Klammern. Jeder Wert ist durch ein Komma getrennt. Du kannst Text, Zahlen, Booleans – alles – in eine Liste packen.

```
einkaufsliste = ["Rambutan", "Drachenfrucht", "Mango"]
```

```
input_vektor = [0, 1, 1, 0, 1, 0]
```

```
gemischt = ["warm", 22.5, True]
```



Hinzufügen

```
einkaufsliste.append("Papaya")
```

Fügt am Ende einen neuen Wert hinzu.



Entfernen

```
einkaufsliste.remove("Mango")
```

Löscht einen Wert aus der Liste.



Abrufen

```
einkaufsliste[0]
```

Gibt den ersten Wert zurück. Python zählt ab 0.

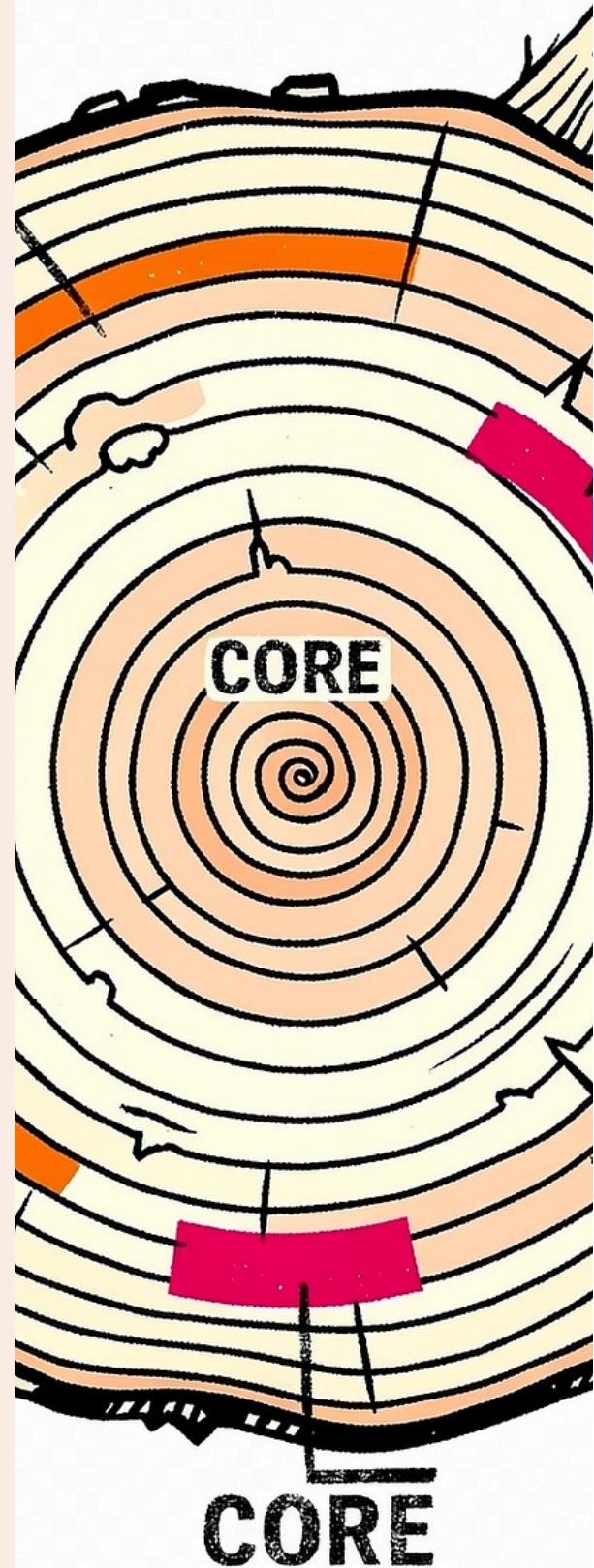
Tupel – Unveränderlich wie Jahresringe

Ein Tupel ist wie eine Liste – aber eingefroren. Was einmal drin ist, bleibt drin. Du kannst nichts hinzufügen, nichts löschen, nichts verändern. Tupel stehen in runden Klammern.

```
geburtsdatum = (15, 8, 1995)  
koordinaten = (10.8231, 106.6297)
```

Wann nimmst du ein Tupel statt einer Liste? Wenn die Daten sich niemals verändern dürfen. Historische Messungen. GPS-Koordinaten. Geburtsdaten. Dinge, die in Stein gemeißelt sind. Tupel schützen deine Daten vor versehentlichen Änderungen.

Im Kontext von KI: Wenn du die Architektur deines neuronalen Netzes als Tupel speicherst – `architektur = (6, 4, 2)` – dann kann niemand aus Versehen eine Schicht löschen. Das Netz bleibt, wie du es gebaut hast.





Workbook-Moment: Dein Input-Vektor

Im letzten Playbook galt: *Garbage In, Garbage Out*. Fehlerhafte Eingabedaten erzeugen unsinnige Vorhersagen. Saubere Daten sind alles. Deine Aufgabe: Erstelle eine Python-Liste mit drei Zutaten, die jetzt gerade in deinem Kühlschrank liegen.

Deine Zutatenliste

```
meine_zutaten  
= [_____,  
_____,  
_____]
```

Bonus: Dein persönlicher Input-Vektor

Welche sechs
Merkmale
beschreiben dein
heutiges
Frühstück?
Schreibe 1 für
„vorhanden“, 0 für
„nicht vorhanden“:

```
fruehstueck_ve  
ktor = [__, __,  
____, __, __, __]
```

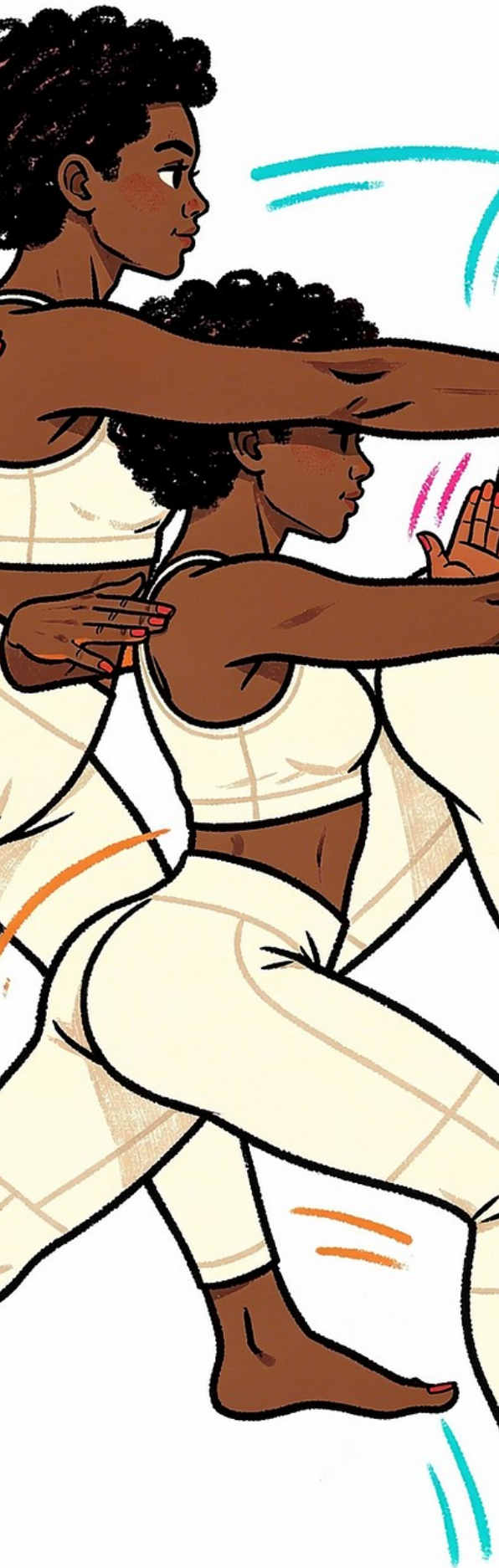
Reflexion

Welches deiner
Lebensmittel würde
ich niemals in ein
Tupel packen (weil
es sich ständig
verändert)?

Der Lernkreislauf in Code

While- & For-

Schleifen



Eine Szene aus Lagos

Chioma Eze - 29 Jahre alt,
Physiotherapeutin in Lagos - übt jeden
Morgen zwölf Yogaübungen. Zwölf Mal
hintereinander, immer in derselben
Reihenfolge. Sie zählt nicht laut. Sie
fragt sich nur: „Habe ich alle zwölf
gemacht?“ Wenn nein - nächste Übung. Wenn
ja - fertig.

Das ist eine Schleife. Chioma führt
denselben Ablauf wieder und wieder aus, bis
eine Bedingung erfüllt ist. Kein händisches
Wiederholen. Kein Copy-Paste. Ein einziger
Ablauf, der sich selbst wiederholt.

Schleifen sind der Motor der
Automatisierung. Und das Herzstück jedes
Machine-Learning-Trainings. Eine KI lernt
nicht in einem Schritt. Sie lernt durch
geduldiges Wiederholen - tausende Runden,
immer ein kleines bisschen besser.

Die For-Schleife – Zähle durch

Die For-Schleife nutzt du, wenn du weißt, wie oft etwas passieren soll. „Gehe durch alle 12 Übungen.“ „Schau dir 100 Bilder an.“ „Führe 1000 Trainingsrunden durch.“

```
yogauebungen = ["Baum", "Krieger", "Hund", "Katze"]  
  
for uebung in yogauebungen:  
    print("Chioma übt:", uebung)
```

Python geht die Liste von Anfang bis Ende durch. Für jeden Wert führt es den eingerückten Code einmal aus. Dann geht es zum nächsten. Automatisch. Ohne Unterbrechung.

```
# 1000 Trainingsrunden  
for runde in range(1000):  
    print("Runde", runde, "– Netz lernt...")
```

`range(1000)` erzeugt die Zahlen 0 bis 999. Tausend Runden. Tausend Mal lernen. Genau das, was eine KI braucht.

Die While-Schleife – Bleib dran

Die While-Schleife nutzt du, wenn du nicht weißt, wie oft etwas passieren soll – sondern nur, bis wann. „Trainiere das Netz, SOLANGE der Fehler über 5 liegt.“ „Koche die Suppe, SOLANGE sie noch nicht kocht.“

```
fehler = 100

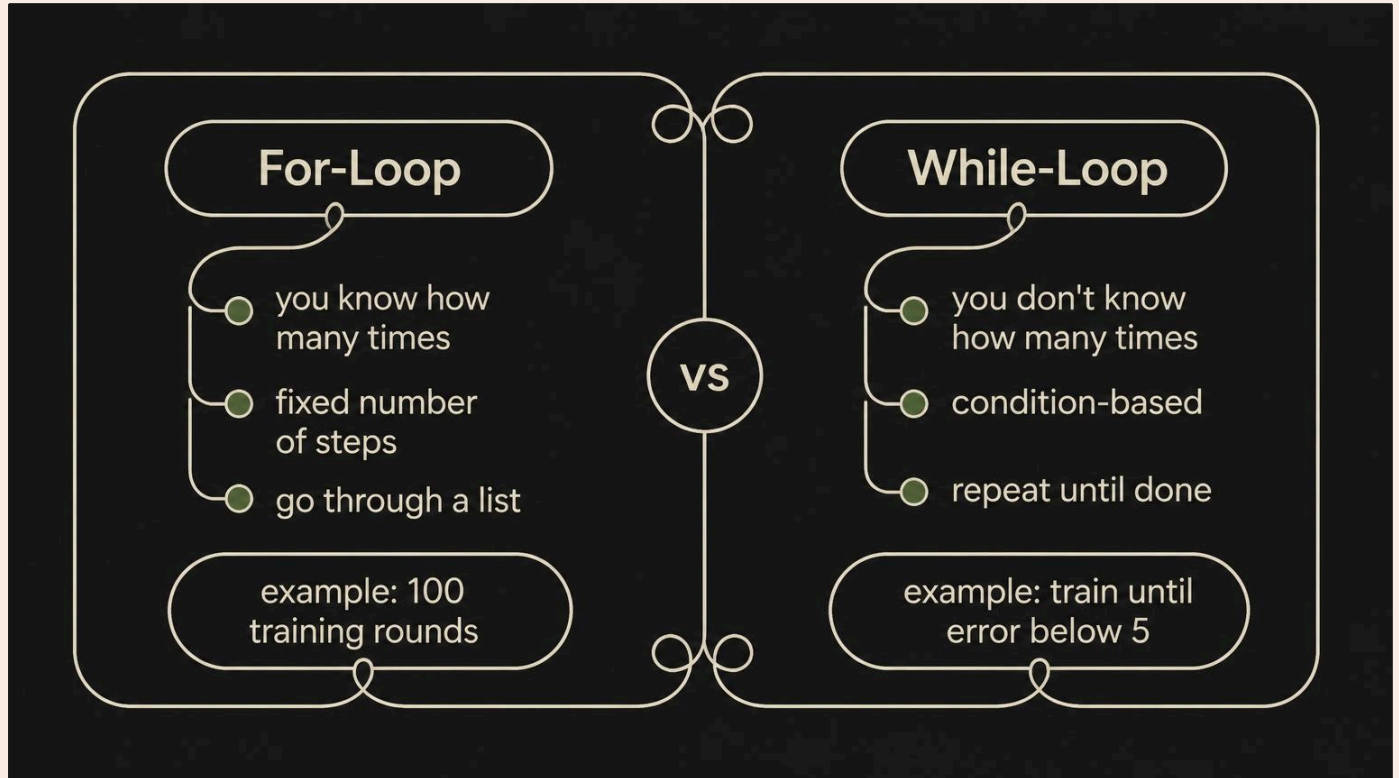
while fehler > 5:
    fehler = fehler * 0.9
    print("Fehler jetzt:", round(fehler, 2))
```

Python prüft die Bedingung vor jedem Durchlauf. Ist sie wahr – führt es den Code aus. Ist sie falsch – hört es auf. Das Netz trainiert, bis es gut genug ist. Nicht mehr, nicht weniger.

- ❏ Achtung: Eine While-Schleife, deren Bedingung nie falsch wird, läuft ewig. Stelle sicher, dass sich die Bedingung mit jeder Runde dem Ende nähert.



For vs. While – Wann nimme ich was?





Workbook-Moment: Die Lernkurve zeichnen

Stell dir vor, du trainierst ein KI-Modell mit einer For-Schleife.
Formuliere in eigenen Worten, was in jeder Runde passieren soll – ganz ohne Code.

Der Auftrag

```
for runde in range(100):  
    # Was passiert hier?
```

In jeder dieser 100 Runden
möchte ich, dass das
Programm...

Deine While-Bedingung

Welche Bedingung würde dich
persönlich motivieren,
weiterzumachen?
„Ich lerne Python, SOLANGE...”

Das Lösungsheft und kleine Helfer

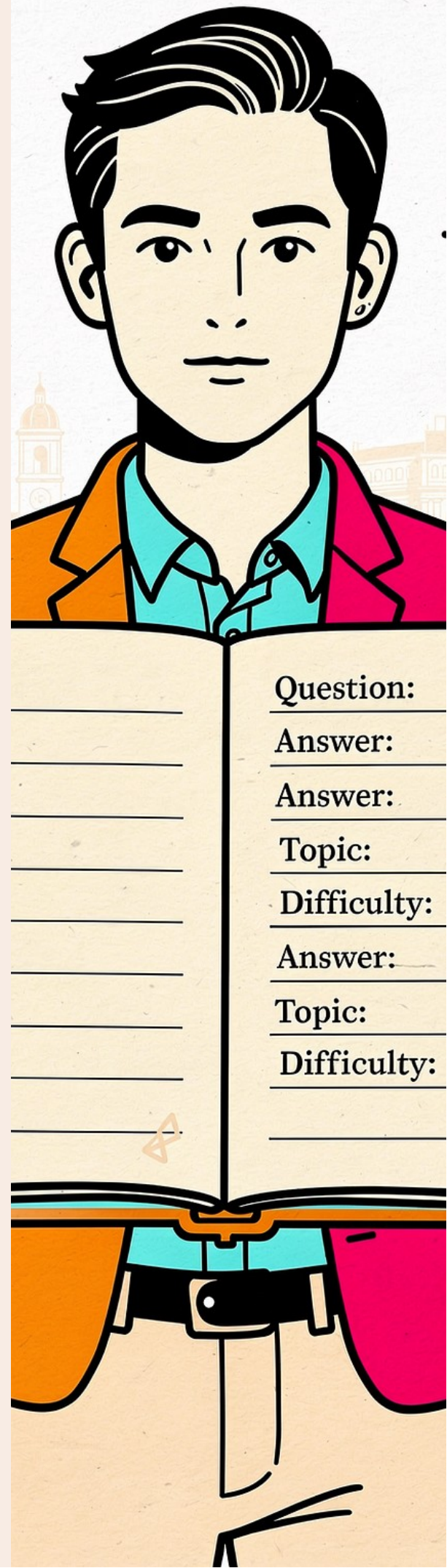
Dictionaries, Funktionen & Exceptions

Eine Szene aus Wien

Parisa Sadeghi - 38 Jahre alt, Lehrerin in Wien - hat für ihre Schüler*innen ein Lösungsheft. Auf der linken Seite steht die Frage. Auf der rechten Seite die Antwort. „Was ist die Hauptstadt von Österreich?" - „Wien." „Was bedeutet Photosynthese?" - „Umwandlung von Licht in Energie."

Ein Dictionary in Python funktioniert genauso. Auf der linken Seite steht der Schlüssel (Key). Auf der rechten Seite steht der Wert (Value). Du gibst Python den Schlüssel - Python gibt dir den Wert zurück.

Parisa nutzt ihr Lösungsheft täglich. Sie schlägt auf, findet die Antwort, schlägt zu. Kein Suchen, kein Blättern durch endlose Seiten. Direkt. Schnell. Präzise. Genau das ist ein Dictionary.



Dictionaries – Das Lösungsheft

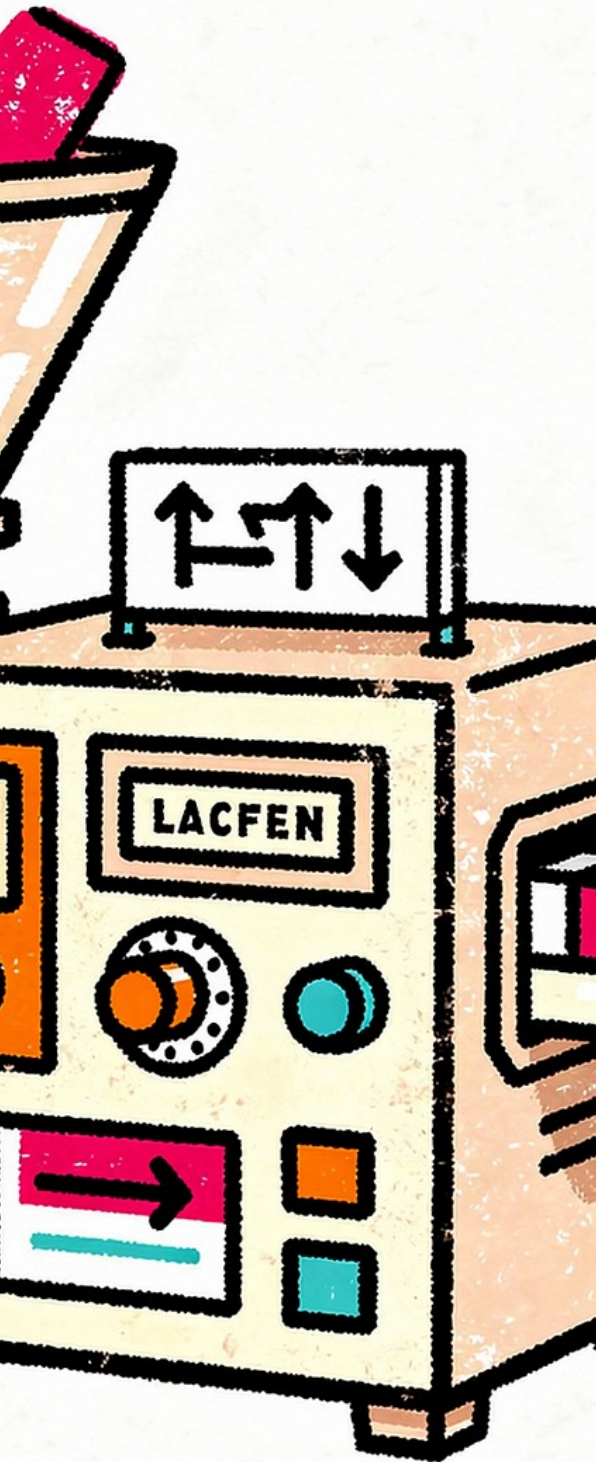
Dictionaries stehen in geschweiften Klammern. Schlüssel und Wert werden durch einen Doppelpunkt getrennt. Jedes Paar wird durch ein Komma abgeschlossen.

```
loesungsheft = {  
    "Käsehäppchen": "Lecker",  
    "Regen": "Schirm mitnehmen",  
    "Python": "Programmiersprache"  
}
```

```
print(loesungsheft["Regen"])  
# Ausgabe: Schirm mitnehmen
```

Du kannst Einträge hinzufügen, verändern, löschen. Dictionaries wachsen mit deinen Daten. In der KI-Welt sind sie unverzichtbar: Sie speichern Hyperparameter, Konfigurationen, Trainingsergebnisse.

```
hyperparameter = {  
    "lernrate": 0.01,  
    "epochen": 1000,  
    "hidden_neuronen": 4  
}
```



Funktionen – Die kleine Maschine

Eine Funktion ist eine kleine Maschine. Du steckst oben etwas rein – den Input. Die Maschine rechnet unsichtbar im Bauch – wie ein Hidden Layer. Unten kommt ein Ergebnis raus – der Output. Du schreibst die Maschine einmal. Dann nutzt du sie so oft du willst.

```
def begruesse(name):  
    nachricht = "Hallo, " + name + "!"  
    return nachricht  
  
print(begruesse("Parisa"))  
# Ausgabe: Hallo, Parisa!
```

`def` bedeutet: „Ich definiere jetzt eine Funktion.“ Der Name dahinter ist ihr Name. Was in den Klammern steht, ist der Input (Argument). `return` gibt das Ergebnis zurück. Fertig. Die Maschine läuft.

Exceptions – Kein Grund zur Panik

Wenn etwas schief läuft, stürzt ein Programm ab. Das nennt man einen Fehler – oder Exception. Mit **Try/Except** sagst du Python: „Versuch es. Wenn ein Fehler passiert – sag mir einfach Bescheid, aber bitte nicht abstürzen.“

```
try:
    ergebnis = 10 / 0
except ZeroDivisionError:
    print("Durch null teilen geht leider nicht.")
finally:
    print("Das Programm läuft weiter.")
```

Try/Except ist wie ein Sicherheitsnetz beim Jonglieren. Du jonglierst. Du weißt, dass du vielleicht eine Kugel fallen lässt. Das Netz fängt sie auf. Das Publikum merkt kaum etwas. Das Spiel geht weiter.

- ❏ Exceptions machen deinen Code robuster. Besonders wenn du mit echten Daten arbeitest, die manchmal fehlerhaft oder unvollständig sind.



Workbook-Moment: Deine Funktion

Im letzten Playbook haben wir die Funktion **Softmax** kennengelernt – sie wandelt Zahlen in Prozentwerte um. Funktionen haben Namen, die beschreiben, was sie tun. Deine Aufgabe: Erfinde Funktionsnamen für alltägliche Aufgaben.

Kaffeekoche n

Welchen Namen
gibst du einer
Funktion, die
deinen Kaffee
kocht?

def _____(
_____):

Entscheidun g treffen

Eine Funktion, die
entscheidet:
Schirm oder
Sonnenbrille?

def _____(
_____r
egenwahrscheinlich
keit):

Dein eigenes Beispiel

Welche Aufgabe in
deinem Alltag
würde gut in eine
Funktion passen?

def _____(
_____):

Was soll sie
zurückgeben?

-

Architektur wie Jahresringe

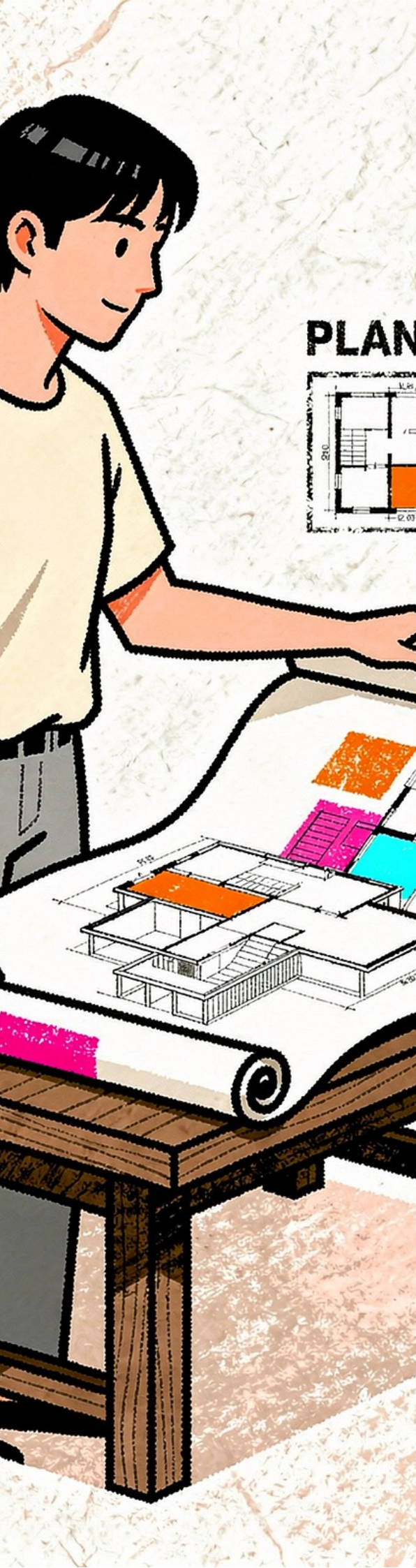
Objektorientierte Programmierung

Eine Szene aus Kyoto

Hiroshi Tanaka - 45 Jahre alt, Architekt in Kyoto - entwirft traditionelle Teehäuser. Er zeichnet einen Bauplan: Grundriss, Maße, Materialien. Aus diesem einen Bauplan können zehn verschiedene Teehäuser gebaut werden - jedes einzigartig, aber alle nach demselben Grundprinzip.

In Python nennen wir diesen Bauplan eine **Klasse**. Aus einer Klasse können wir beliebig viele Objekte erschaffen - jedes mit eigenen Eigenschaften, aber alle mit demselben Grundaufbau. Das ist Objektorientierte Programmierung, kurz OOP.

OOP klingt abstrakt. Aber du kennst das Prinzip bereits. Im letzten Playbook hatten wir den Bauplan unserer Koch-KI: 6 Input-Neuronen, 4 Hidden-Neuronen, 2 Output-Neuronen. In Python schreiben wir das als Klasse.



Klassen – Der Bauplan

Eine Klasse hat zwei Zutaten: **Eigenschaften** (was ein Objekt hat) und **Methoden** (was ein Objekt kann). Hiroshis Teehaus hat Eigenschaften: Größe, Holzsorte, Ausrichtung. Und Methoden: öffnen, schließen, reinigen.

```
class Teehaus:
    def __init__(self, groesse, holzsorte):
        self.groesse = groesse
        self.holzsorte = holzsorte

    def beschreibe(self):
        print("Teehaus aus", self.holzsorte)
        print("Größe:", self.groesse, "qm")

haus1 = Teehaus(20, "Zeder")
haus2 = Teehaus(35, "Kiefer")

haus1.beschreibe()
haus2.beschreibe()
```

Zwei Teehäuser. Ein Bauplan. `__init__` ist der Konstruktor – er läuft automatisch, wenn ein neues Objekt entsteht. `self` bezieht sich auf das aktuelle Objekt.

OOP und das neuronale Netz

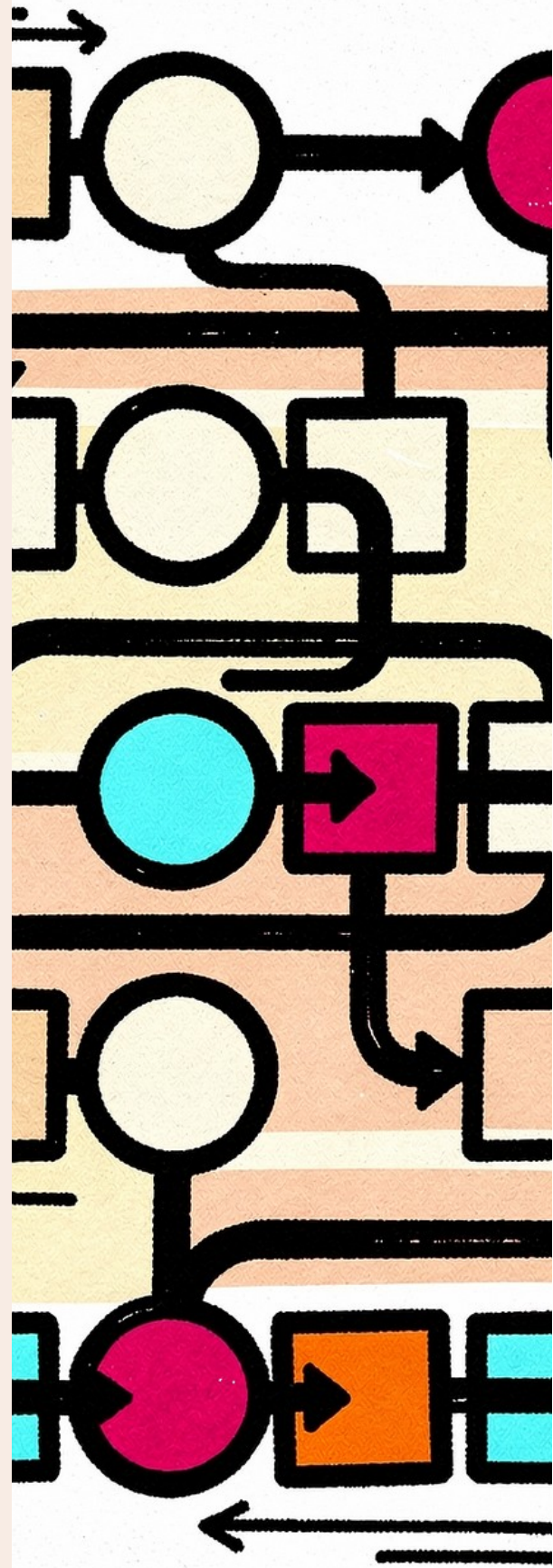
Jetzt wird es konkret. Das neuronale Netz aus dem letzten Playbook lässt sich wunderbar als Klasse schreiben. Der Bauplan enthält alles: Anzahl der Neuronen, Gewichte, die Vorwärtsrechnung.

```
class NeuronalesNetz:
    def __init__(self, input_n, hidden_n,
output_n):
        self.input_n = input_n
        self.hidden_n = hidden_n
        self.output_n = output_n

    def beschreibe(self):
        print("Netz:", self.input_n,
            "→", self.hidden_n,
            "→", self.output_n)

kochki = NeuronalesNetz(6, 4, 2)
kochki.beschreibe()
# Ausgabe: Netz: 6 → 4 → 2
```

Aus diesem Bauplan kannst du hundert verschiedene Netze erschaffen. Jedes mit eigenen Hyperparametern. Jedes für eine eigene Aufgabe. Alle nach demselben Grundprinzip.





Workbook-Moment: Dein eigener Bauplan

Wenn du eine Klasse für dich selbst schreiben müsstest – was würde darin stehen? Eigenschaften beschreiben, was du hast. Methoden beschreiben, was du kannst.

Eigenschaft 1

Was ist eine
messbare
Eigenschaft von
dir?

self. _____

__ =

Eigenschaft 2

Noch eine
Eigenschaft –
diesmal eine, die
sich verändern
kann:

self. _____

__ =

Methode – Deine Stärke

Was kannst du
besonders gut? Wie
würde eine
Funktion dafür
heißen?

def

(self):

Der Kreis schließt sich

Machine Learning

mit Python

Eine Szene aus Johannesburg

Amara Dlamini - 41 Jahre alt,
Datenwissenschaftlerin in Johannesburg -
sitzt an ihrem Schreibtisch. Vor ihr: ein
Datensatz mit tausend Rezepten. Ihr Ziel:
ein Modell trainieren, das vorhersagt, ob
ein Rezept vegan ist oder nicht. Sie öffnet
Python. Sie importiert scikit-learn. Drei
Zeilen Code. Das Modell lernt. Amara
lächelt.

Das ist der Moment, auf den wir
hingearbeitet haben. Du hast Variablen,
Listen, Schleifen, Funktionen, Klassen
gelernt. Jetzt bringst du alles zusammen.
Mit Python-Bibliotheken wie **scikit-learn**,
TensorFlow oder **PyTorch** musst du die
komplexe Mathematik der Backpropagation
nicht mehr selbst programmieren. Die
Bibliothek übernimmt das. Du steuerst.



Die großen Bibliotheken – Dein Werkzeugkasten



scikit-learn

Der Einstieg. Klassische ML-Algorithmen. Ideal für Klassifizierung, Regression, Clustering. Perfekt, um loszulegen.



TensorFlow

Googles Bibliothek für tiefe neuronale Netze. Industriestandard. Skalierbar von Laptop bis Rechenzentrum.



PyTorch

Metas Bibliothek. Besonders beliebt in der Forschung. Flexibel, intuitiv, Python-nah. Viele Tutorials verfügbar.

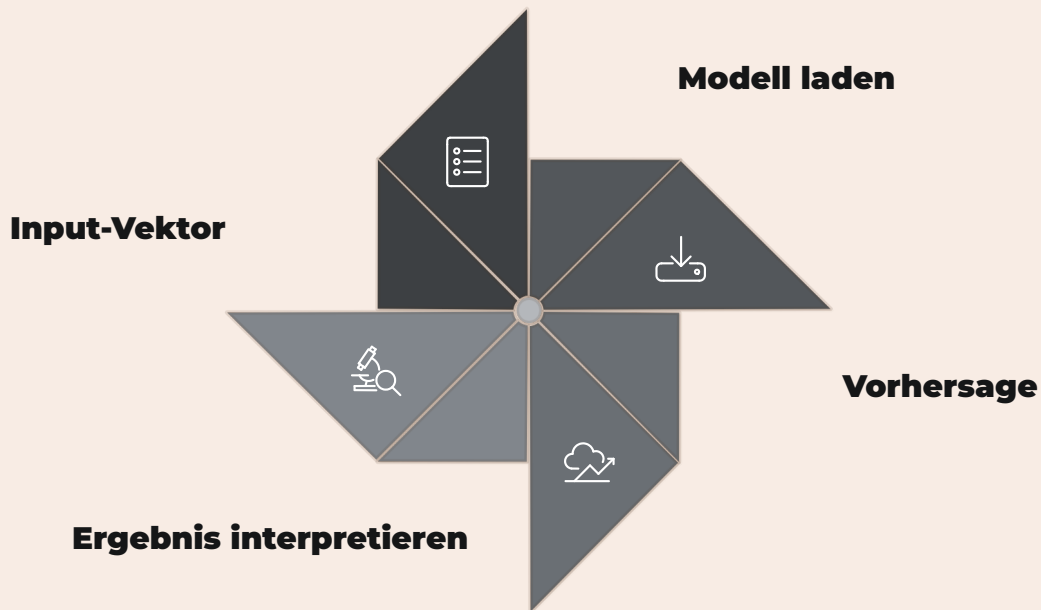


pandas & numpy

Datenvorbereitung und Mathematik. Bevor ein Modell lernt, müssen Daten sauber sein. Diese zwei helfen dabei.

Vom Input-Vektor zur Vorhersage

Alles, was du gelernt hast, fügt sich jetzt zusammen. Dein neuronales Netz aus dem letzten Playbook wird in wenigen Zeilen Python real.



```
from sklearn.neural_network import MLPClassifier

# Trainingsdaten
X = [[0,1,1,0,1,0], [1,0,0,1,0,1]]
y = ["Käsehäppchen", "Salat"]

# Modell erstellen und trainieren
modell = MLPClassifier()
modell.fit(X, y)

# Vorhersage
neuer_input = [[0,1,1,0,1,0]]
vorhersage = modell.predict(neuer_input)
print("Vorschlag:", vorhersage[0])
```

Was du jetzt kannst

Daten strukturieren

Variablen, Listen, Tupel, Dictionaries – du speicherst Informationen sauber und abrufbar.

Logik schreiben

If/Else, Schleifen – du steuerst, was wann passiert. Du automatisierst Entscheidungen.

Code strukturieren

Funktionen, Klassen – du baust wiederverwendbare Bausteine. Dein Code wächst geordnet.

KI bedienen

Mit Bibliotheken trainierst du echte Modelle. Du gibst Input rein, du liest Output aus.



Finaler Workbook- Moment: Vom Aha zum Code

Im letzten Playbook haben wir festgestellt: *Eine KI erfindet nichts. Sie entfaltet, was in den Daten steckt.* Du hast jetzt die Sprache, um Daten zum Sprechen zu bringen. Die Frage ist: Welche Geschichte willst du erzählen?

Dein erstes Projekt

Welches kleine
Projekt möchtest
du als erstes mit
Python angehen?

Dein Datensatz

Welche Daten aus
deinem Leben
möchtest du
analysieren oder
automatisieren?

Dein erster Schritt heute

Was genau machst
du in den nächsten
30 Minuten?

Glossar – Die Sprache der Sprache

Python-Grundbegriffe von A bis L

Glossar A-L

Begriff	Bedeutung	Beispiel
Boolean	Datentyp mit nur zwei Werten: wahr oder falsch	True / False
Datentyp	Art einer Information - Text, Zahl, Wahrheitswert	str, int, float, bool
Dictionary	Sammlung von Schlüssel-Wert-Paaren	{"Regen": "Schirm"}
Exception	Ein Fehler, der zur Laufzeit auftritt	ZeroDivisionError
Float	Kommazahl	3.14
For-Schleife	Wiederholt Code für jedes Element einer Liste	for x in liste:
Funktion	Wiederverwendbarer Codeblock mit Input und Output	def name():
IDE	Entwicklungsumgebung - Werkzeug zum Schreiben von Code	PyCharm, VS Code
Integer	Ganze Zahl ohne Komma	42
Klasse	Bauplan für Objekte in der OOP	class Netz:
Liste	Geordnete, veränderliche Sammlung von Werten	[1, 2, 3]

Glossar M–Z

Begriff	Bedeutung	Beispiel
Methode	Funktion, die zu einer Klasse gehört	<code>objekt.beschreibe()</code>
Modul	Datei mit wiederverwendbarem Python-Code	<code>import math</code>
Objekt	Konkretes Exemplar einer Klasse	<code>mein_netz = Netz(6,4,2)</code>
OOP	Objektorientierte Programmierung – Struktur durch Baupläne	Klassen & Objekte
<code>print()</code>	Gibt Text auf dem Bildschirm aus	<code>print("Hallo!")</code>
String	Text in Anführungszeichen	<code>"Hallo Welt"</code>
Tupel	Geordnete, unveränderliche Sammlung von Werten	<code>(1, 2, 3)</code>
Variable	Name für einen gespeicherten Wert	<code>hunger = 8</code>
While-Schleife	Wiederholt Code, solange eine Bedingung wahr ist	<code>while fehler > 5:</code>

Glossar – KI-Bibliotheken

scikit-learn

Python-Bibliothek für klassisches Machine Learning. Enthält vorgefertigte Algorithmen für Klassifizierung, Regression und Clustering. Perfekter Einstieg.

TensorFlow

Googles Open-Source-Framework für tiefe neuronale Netze. Skaliert von kleinen Experimenten bis zu industriellen Produktivsystemen.

PyTorch

Metas Framework. Dynamisch, flexibel, Python-nah. In der Forschungsgemeinschaft sehr beliebt. Viele Tutorials und eine aktive Community.

pandas

Datenverarbeitung und -analyse. Macht es einfach, Tabellen zu laden, zu bereinigen und zu transformieren. Unverzichtbar vor dem Training.

numpy

Numerisches Rechnen in Python. Vektoren, Matrizen, mathematische Operationen – schnell und präzise. Die Grundlage für fast alles in der KI.

Keras

Benutzerfreundliche API für TensorFlow. Weniger Code, schneller zu einem laufenden Modell. Ideal für Einsteigende in tiefe neuronale Netze.

Ausblick – Was kommt als Nächstes?

Du hast die Grundlagen von Python gelernt. Du kannst Daten speichern, Entscheidungen treffen, Schleifen schreiben, Funktionen bauen und Klassen entwerfen. Du hast eine KI mit echtem Python-Code angesteuert. Das ist keine Kleinigkeit.

Was öffnet sich jetzt? Drei Türen stehen vor dir. Du kannst durch alle drei – nacheinander oder nach Interesse.



Drei Wege nach vorne

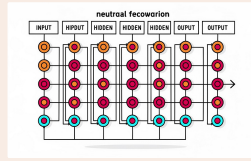


Weg 1: Daten erkunden

Lerne pandas und matplotlib. Lade echte Datensätze.

Visualisiere Muster.

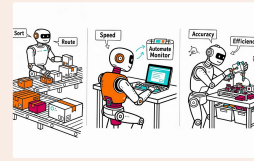
Finde Antworten in
Zahlen, die du vorher
nicht gesehen hast.



Weg 2: Tiefer in KI

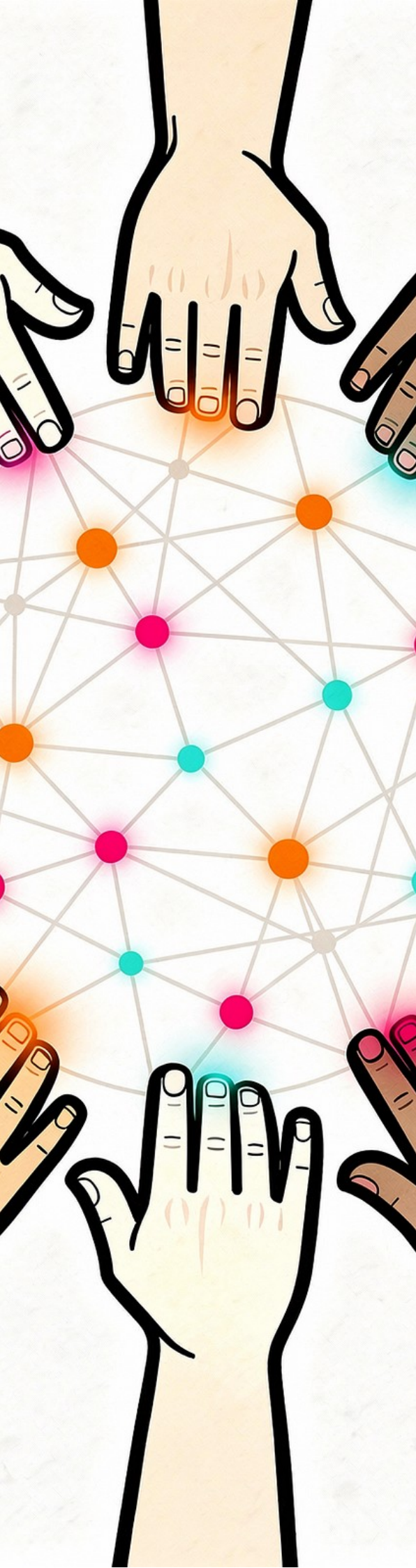
Lerne TensorFlow oder
PyTorch. Baue tiefe
neuronale Netze.

Trainiere
Bilderkennungsmodelle.
Nutze vortrainierte
Modelle und passe sie
an.



Weg 3: Automatisieren

Automatisiere
Alltagsaufgaben mit
Python. Lese E-Mails
aus, verarbeite
Tabellen, erstelle
Berichte. Python als
persönlicher Assistent.

An illustration on the left side of the page shows several hands of different skin tones reaching up and down, holding a network of interconnected nodes. The nodes are represented by small circles in orange, pink, and teal, connected by thin grey lines. The hands are drawn with thick black outlines and are positioned around the network, suggesting a collective effort or support.

KI für das Gute – AI for Good

Python und KI sind Werkzeuge. Wie jedes Werkzeug hängt ihr Wert davon ab, wofür wir sie einsetzen. Amara in Johannesburg analysiert Ernährungsdaten, um Mangelernährung in Gemeinden früher zu erkennen. Linh in Hanoi automatisiert Übersetzungen für Gesundheitsinformationen in ländlichen Regionen. Parisa in Wien erstellt personalisierte Lernpläne für Schüler*innen mit besonderen Bedürfnissen.

Du hast jetzt dieselben Werkzeuge in der Hand. Was du damit tust, liegt bei dir. Die Frage „Was soll mein Code bewirken?“ ist mindestens so wichtig wie die Frage „Wie schreibe ich ihn?“

Technik folgt Werten. Deine Werte sind dein Kompass. Starte immer mit der Frage: Wem nützt das? Wer wird davon berührt? Was verändert sich dadurch?

Lernressourcen – Dein weiterer Weg



python.org/doc

Die offizielle Python-Dokumentation. Klar, vollständig, kostenlos. Immer die verlässlichste Quelle.



kaggle.com

Echte Datensätze, Lernkurse und Wettbewerbe. Praktisches Lernen mit echter Community-Unterstützung.



fast.ai

Praxisorientierter Kurs für Deep Learning. Top-down-Ansatz: erst ausprobieren, dann verstehen.



cs50.harvard.edu

Harvards kostenloser Einführungskurs in Informatik. Rigoros, tiefgehend, weltweit beliebt.

Abschluss & Danke

„Eine KI erfindet nichts. Sie entfaltet, was in den Daten steckt.“

Du hast jetzt die Sprache. Du hast die Werkzeuge. Du hast die Konzepte. Was bleibt, ist das Tun.

Öffne PyCharm. Schreibe die erste Zeile. Mach den ersten Fehler. Lerne aus ihm. Geh zur nächsten Zeile. Das ist Programmieren. Das ist Lernen. Das ist Wachsen.

Dieses Playbook ist das zweite in einer Reihe. Es baut auf „*Neuronale Netze verstehen*“ auf und bereitet den Weg für das nächste: **„Daten sprechen lassen – Analyse, Visualisierung und erste KI-Projekte.“**

*powered by AI: Recherche, Iteration, Kuration & Redaktion: Carmen Frohne |
September 2026*

Bonus: Häufige Fehler und was sie bedeuten

Fehler sind keine Zeichen des Scheiterns. Sie sind Hinweise. Python erklärt dir im Klartext, was schiefgelaufen ist. Hier sind die häufigsten Fehler für Einsteigende - und was sie dir sagen wollen.

Fehlermeldung	Ursache	Lösung
SyntaxError	Python versteht den Code nicht - Schreibfehler, fehlender Doppelpunkt	Zeile genau ansehen. Doppelpunkte, Klammern, Anführungszeichen prüfen.
NameError	Variable wurde verwendet, bevor sie definiert wurde	Variable zuerst definieren: <code>name = "Wert"</code>
IndentationError	Einrückung fehlt oder ist falsch	Nach if, for, def immer 4 Leerzeichen einrücken.
TypeError	Falsche Datentypen werden kombiniert	<code>str(zahl)</code> oder <code>int(text)</code> zur Umwandlung nutzen.
IndexError	Zugriff auf einen Index, der nicht existiert	Python zählt ab 0. Liste mit 3 Elementen: Index 0, 1, 2 - nicht 3.

Bonus: Dein 30-Tage-Lernplan

Dreißig Tage. Je 20-30 Minuten. Das reicht, um Python wirklich zu verinnerlichen. Kein Sprint - ein ruhiger, gleichmäßiger Rhythmus. Wie Chiomas Yogaübungen.

Woche 1

Installation, Hello World, Variablen, Datentypen. Jeden Tag eine Übung aus Kapitel 1-3.

Woche 3

Funktionen, Klassen, Exceptions. Baue die Koch-KI als Python-Klasse nach. Verändere Hyperparameter.

Woche 2

Listen, Tupel, Dictionaries, Schleifen. Schreibe täglich ein kleines Programm mit echten Daten aus deinem Leben.

Woche 4

Dein erstes eigenes Projekt. Wähle einen Datensatz aus deinem Alltag. Analysiere ihn. Automatisiere etwas.

📌 Fortschritt entsteht nicht durch intensive Sprint-Sessions - sondern durch tägliche kurze Kontakte mit dem Stoff. 20 Minuten täglich schlagen 3 Stunden einmal pro Woche.



Bonus: Die Python-Philosophie

Python hat eine eigene Philosophie. Du kannst sie jederzeit in Python selbst lesen: Tippe `import this` in dein Terminal. Es erscheint das „Zen of Python“ – 19 Grundsätze von Tim Peters, 1999.

„Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Readability counts.“

Diese Philosophie ist kein Regelwerk. Sie ist eine Einladung. Python wurde für Menschen gebaut – nicht für Maschinen. Code, den ein Mensch lesen kann, ist guter Code. Komplexität ist kein Zeichen von Können. Klarheit ist es.

Eine letzte Geschichte – Aus Oslo

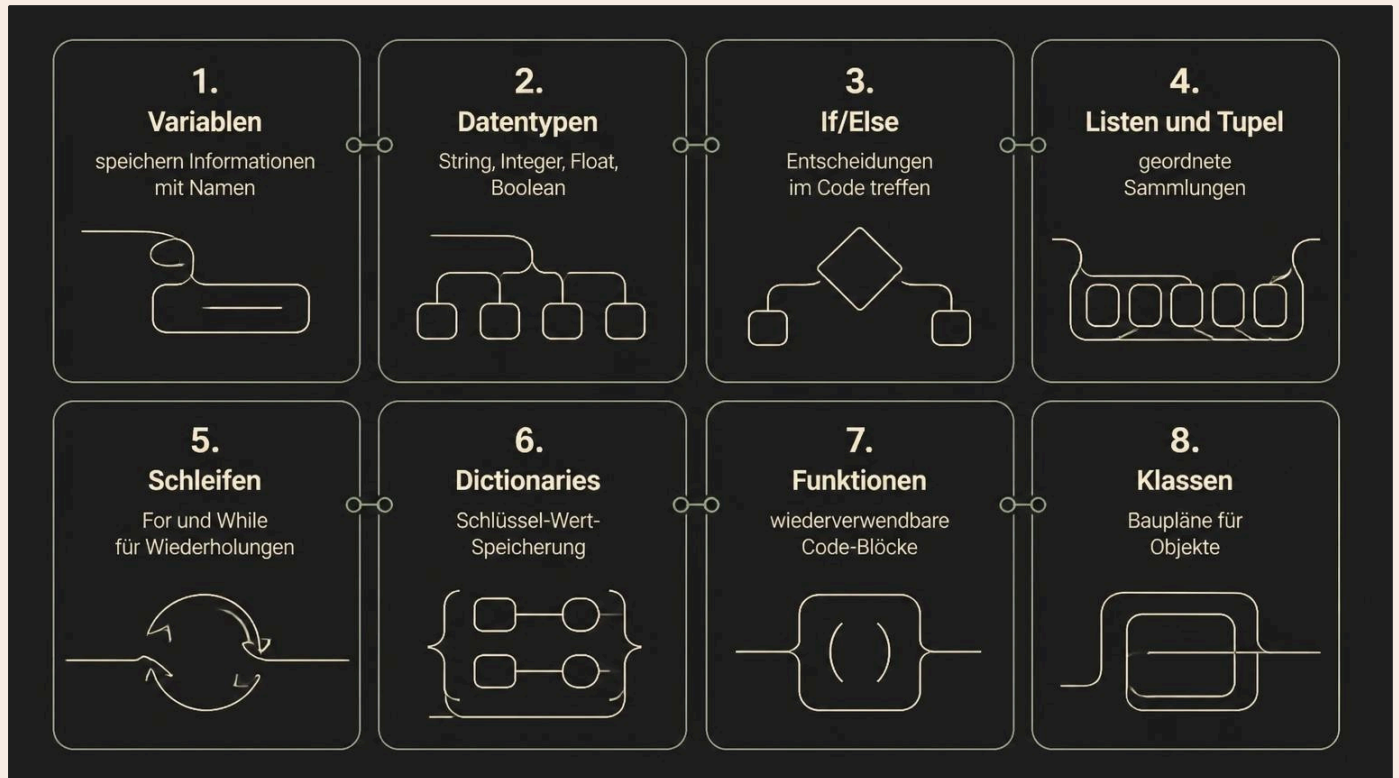
Erik Andersen – 47 Jahre alt, Lehrer für Naturwissenschaften in Oslo – hat dieses Playbook an einem Winterwochenende durchgearbeitet. Schnee draußen. Kaffee drinnen. Am Sonntagabend hat er sein erstes Python-Programm geschrieben: Es liest die Temperaturdaten seiner Wetterstation aus und berechnet den Wochendurchschnitt.

Achtzehn Zeilen Code. Drei Stunden Arbeit. Ein echter Nutzen in seinem Alltag.

Er schreibt in sein Notizbuch: *„Ich dachte, ich bin zu alt dafür. Ich war falsch.“*

Du bist nie zu alt, zu beschäftigt oder zu weit weg vom Thema, um anzufangen. Der einzige Moment, der zählt, ist dieser hier.

Zusammenfassung – Was du gelernt hast



Du bist bereit.

Öffne PyCharm.

Lass uns den ersten Forward Pass in Code gießen. Die Küche ist eingerichtet.
Die Zutaten liegen bereit. Du weißt, wie man kocht.

Schritt 1

PyCharm öffnen.
Neues Projekt
anlegen. Erste .py-
Datei erstellen.

Schritt 2

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

schreiben. Enter
drücken. Lächeln.

Schritt 3

Deine erste
Variable. Dann eine
Liste. Dann eine
Schleife. Einen Tag
nach dem anderen.

*Python verstehen - Dein Lernweg von Null auf Code | powered by AI | Carmen
Frohne | September 2026*