GANZ EINFACH

Python

Programmieren lernen ohne Kopfschmerzen

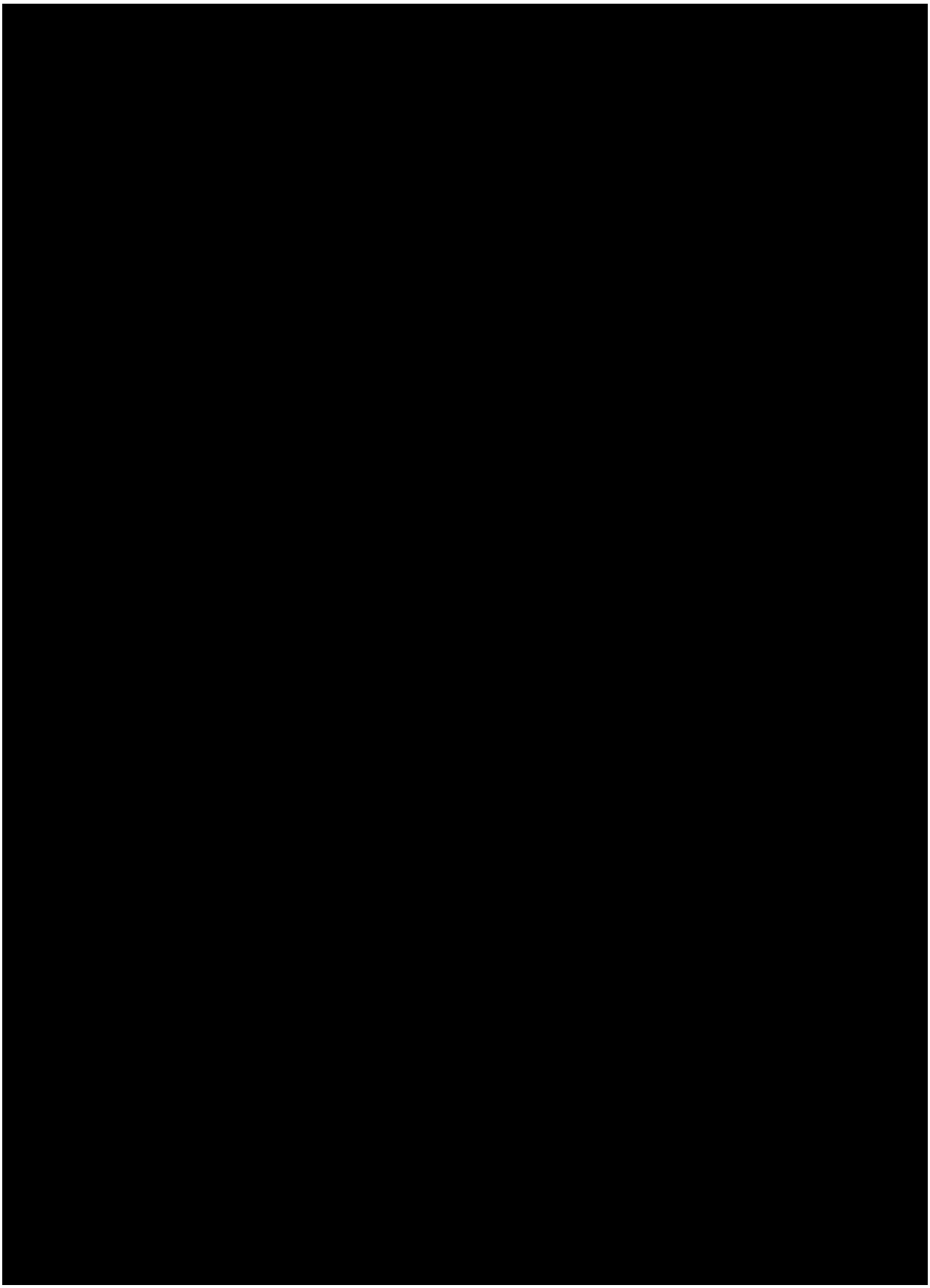
Danilo Sieren

GANZ EINFACH

Python

Programmieren lernen ohne Kopfschmerzen

Danilo Sieren



Dein Code, deine Freiheit
Vom ersten Skript bis zur intelligenten Automatisierung in deiner Cloud

Einfach erklärt

Python

Programmieren lernen ohne Kopfschmerzen

Danilo Sieren

Auflage 1

Inhaltsverzeichnis

Vorwort

Warum dieses Buch dein Leben verändern wird

Schön, dass du da bist.

Wenn du dieses Buch aufgeschlagen hast, dann wahrscheinlich deshalb, weil du die Schnauze voll davon hast, nur ein Passagier in der digitalen Welt zu sein. Du hast vielleicht schon unsere Reise durch die Welt der privaten Clouds, die Zähmung von Windows und die Freiheit von Linux mitgemacht. Du weißt jetzt, wie man Systeme absichert und konfiguriert. Aber heute zünden wir die nächste Stufe. Heute lernst du, wie man Systeme **erschafft**.

Unter uns: Programmieren lernen hat einen schlechten Ruf. Viele denken an blasse Gestalten in dunklen Kellern, die kryptische Zeichenketten in schwarze Bildschirme hämmern und Unmengen an Energy-Drinks konsumieren. Vergiss dieses Bild. Programmieren ist im 21. Jahrhundert das, was das Lesen und Schreiben im Mittelalter war: Es ist die ultimative Befreiung.

Python ist dein digitaler Dietrich

In den letzten Jahren haben wir erlebt, wie Software immer mehr Kontrolle über unser Leben übernimmt. Algorithmen entscheiden, was wir sehen, Abos entscheiden, was wir nutzen dürfen, und Cloud-Giganten entscheiden, wo unsere Daten liegen. Mit Python nimmst du das Heft des Handelns wieder selbst in die Hand.

Stell dir Python wie ein Schweizer Taschenmesser vor, das niemals stumpf wird. Du willst 5.000 PDF-Dateien in **deiner Cloud** nach einem bestimmten Wort durchsuchen? Ein

Zehnzeiler in Python erledigt das in Sekunden. Du willst dein Smart Home steuern, ohne dass ein Server in Übersee mitlauscht? Python ist die Lösung. Du willst eine KI nutzen, die nur auf deinem Rechner läuft? Python ist die Sprache, die sie versteht.

Ein Gespräch unter Freunden

Ich verspreche dir eines: Dieses Buch wird dich nicht mit akademischer Theorie langweilen. Wir schreiben hier kein Lehrbuch für eine Prüfung, sondern einen **Battle Plan** für echte Anwender. Wir werden Python so besprechen, wie wir es unter Freunden bei einem guten Kaffee tun würden – locker, ehrlich, mit vielen Beispielen aus der Praxis und ohne unnötiges Fachchinesisch.

Wir werden tief graben. Wir begnügen uns nicht damit, dass etwas funktioniert. Wir wollen wissen, **warum** es funktioniert. Jedes Kapitel ist ein Puzzleteil auf deinem Weg vom Anfänger zum souveränen Anwender. Wir haben uns ein ehrgeiziges Ziel gesetzt: 15 Kapitel, vollgepackt mit Wissen, Hintergrundinfos und echten Projekten, die dein digitales Leben in **deiner Cloud** bereichern werden.

Deine Reise beginnt jetzt

Du musst kein Mathe-Genie sein, um Python zu lernen. Du brauchst nur Neugier und die Bereitschaft, Dinge auszuprobieren. Programmieren ist ein Handwerk, genau wie Tischlern oder Kochen. Man lernt es, indem man die Späne fliegen lässt oder sich mal die Finger verbrennt – aber am Ende steht ein Werkstück, auf das man stolz sein kann.

In den kommenden Kapiteln werden wir deine Arbeitsweise revolutionieren. Wir bauen Brücken zwischen Windows, Linux und macOS. Wir lassen Skripte für uns arbeiten,

während wir den Feierabend genießen. Und wir sorgen dafür, dass dein Computer endlich genau das tut, was **du** willst – und nicht das, was ein Konzern in Redmond oder Cupertino für dich vorgesehen hat.

Bist du bereit, die Kontrolle über die Maschine zu übernehmen? Dann lass uns keine Zeit verlieren. Die erste Zeile Code wartet schon auf dich.

Willkommen in der Freiheit. Willkommen in der Welt von Python.

Lass uns loslegen!

Danilo Sieren

Kapitel 1

Die Grundsteinlegung – Dein Start in die Welt von Python

1.1 Warum ausgerechnet Python?

Bevor wir die erste Zeile Code in den Editor hämmern, müssen wir mal kurz darüber reden, was wir hier eigentlich tun. Wenn du dich heute entscheidest, Python zu lernen, dann ist das so, als würdest du entscheiden, eine Weltsprache zu lernen – aber eine, die nicht nur Menschen verstehen, sondern auch fast jede Maschine auf diesem Planeten.

Unter uns: Programmieren hat oft diesen Beigeschmack von "Mathe-Genie im dunklen Keller". Aber Python bricht mit diesem Klischee. Es wurde Ende der 80er Jahre von einem Niederländer namens Guido van Rossum erfunden. Und weißt du, was sein Ziel war? Er wollte eine Sprache schaffen, die so einfach zu lesen ist wie Englisch. Er war ein großer Fan der britischen Komikertruppe "Monty Python" (daher auch der Name!), und diesen Humor und die Leichtigkeit merkt man der Sprache bis heute an.

Python ist eine **High-Level-Sprache**. Das bedeutet im Grunde nur, dass sie sehr weit weg von den Nullen und Einsen der Hardware ist und sehr nah an unserer menschlichen Sprache. Wenn du in einer Sprache wie C++ sagen willst, dass der Computer "Hallo" sagen soll, musst du ihm erst erklären, wie er den Speicher verwaltet, welche Bibliotheken er laden soll und wo das Semikolon hinkommt. In Python sagst du einfach: `print("Hallo")`. Fertig. Das ist Freiheit.

Die Philosophie: "The Zen of Python"

Wusstest du, dass Python eine eigene Philosophie hat? Wenn du später in deinem Terminal mal `import this` eingibst, erscheint ein Gedicht namens "The Zen of Python". Darin stehen Sätze wie: *„Beautiful is better than ugly“* oder *„Simple is better than complex“*. Das ist genau unser Ansatz in diesem Buch. Wir wollen keine komplizierten Monster-Programme bauen, sondern elegante Lösungen, die wir auch in zwei Jahren noch verstehen, wenn wir sie in **deiner Cloud** wieder ausgraben.

1.2 Die Werkstatt einrichten: Installation und Umgebung

So, genug der Vorrede. Ein Handwerker braucht eine saubere Werkbank. Da wir in unseren vorherigen Projekten dein Windows-System schon gehärtet und optimiert haben, nutzen wir diese Stärke jetzt aus.

Der Python-Interpreter: Das Herzstück

Wie ich schon im Vorwort erwähnt habe, ist Python eine **interpretierte Sprache**. Stell dir das so vor: Du schreibst ein Rezept (deinen Code). Der Python-Interpreter ist der Koch. Er liest das Rezept Zeile für Zeile und setzt es sofort um. Er wartet nicht, bis das ganze Menü fertig geschrieben ist, sondern fängt direkt an zu arbeiten.

Die Installation unter Windows (Der souveräne Weg): Wir nutzen natürlich nicht den Microsoft Store (wir erinnern uns: wir wollen Unabhängigkeit!). Wir nutzen unsere PowerShell.

1. Öffne die PowerShell als Administrator.

2. Tippe ein: winget install Python.Python.3.12 (oder die aktuellste Version).
3. **Ganz wichtig:** Wenn du doch den grafischen Installer nutzt, aktiviere unbedingt das Kästchen **"Add Python to PATH"**.

Was bedeutet "PATH"? Das ist ein kleiner, aber feiner technischer Hintergrund. Wenn du in der Konsole python tippst, muss Windows wissen, in welchem dunklen Ordner auf der Festplatte das Programm eigentlich liegt. "PATH" ist wie ein Adressbuch für Windows. Steht Python darin, findet Windows es sofort. Wenn nicht, schreit dich die Konsole an: "Befehl nicht gefunden". Und das wollen wir am frühen Morgen nicht, oder?

Visual Studio Code: Dein Cockpit

Wir könnten Python-Code auch im Windows-Editor (Notepad) schreiben, aber das wäre, als würde man versuchen, mit einem Löffel einen Garten umzugraben. Es geht, macht aber keinen Spaß. Wir nutzen **Visual Studio Code (VS Code)**. Es ist schlank, gehört zwar zu Microsoft, ist aber in weiten Teilen Open Source und das absolute Standard-Werkzeug für Python-Entwickler weltweit.

- **Installation:** winget install Microsoft.VisualStudioCode
- **Extensions:** Sobald VS Code offen ist, drücke Strg + Umschalt + X. Suche nach "Python" (von Microsoft) und installiere es. Jetzt bekommt dein Editor "Augen" – er erkennt Fehler, während du tippst, und macht dir Vorschläge (IntelliSense). Das ist wie ein Co-Pilot, der neben dir sitzt.

1.3 Das erste Projekt in deiner Cloud

Hier schlagen wir die Brücke zu deinem ersten Buch. Wir speichern unseren Code niemals einfach nur auf C:\. Wir speichern ihn dort, wo er sicher ist: in **deiner Cloud**.

1. Erstelle in deinem synchronisierten Cloud-Ordner ein neues Verzeichnis: Programmierung/Python_Masterclass.
2. Öffne diesen Ordner in VS Code (*Datei > Ordner öffnen*).
3. Erstelle eine neue Datei namens start.py.

Warum die Endung .py? Das ist das Signal für das Betriebssystem: "Achtung, hier kommt Python-Code!".

1.4 Variablen – Die Umzugskartons deines Programms

Jetzt wird es ernst. Wir fangen an zu programmieren. Das wichtigste Konzept in jeder Sprache sind **Variablen**.

Unter Freunden erklärt: Eine Variable ist wie ein Umzugskarton. Du klebst ein Etikett drauf (den Namen) und legst etwas hinein (den Wert). Später kannst du den Karton öffnen, nachschauen was drin ist oder den Inhalt austauschen.

Schreib mal folgendes in deine start.py:

Python

```
benutzername = "Abenteurer"
alter = 25
```

```
status = "online"

print(benutzername)
```

Was passiert hier im Hintergrund? Python ist extrem schlau. In anderen Sprachen (wie Java oder C) müsstest du dem Computer erst sagen: "Achtung, jetzt kommt eine Zahl!" oder "Achtung, jetzt kommt Text!". Python schaut sich den Inhalt an und weiß es von selbst. Das nennt man **dynamische Typisierung**.

- benutzername ist ein **String** (Zeichenkette) – erkennbar an den Anführungszeichen.
- alter ist ein **Integer** (Ganzzahl).
- status ist wieder ein String.

Ein kleiner Tipp am Rande: Nenne deine Variablen niemals einfach `a`, `b` oder `x1`. Wenn du in drei Monaten dein Skript in **deiner Cloud** öffnest, hast du keine Ahnung mehr, was `x1` war. Nenne sie `rechnungs_betrag` oder `letzter_login`. Dein "Zukunfts-Ich" wird dir dankbar sein.

Dann lass uns die Ärmel noch ein Stück weiter hochkrempeln. Wir haben gerade die Umzugskartons (Variablen) kennengelernt, aber jetzt schauen wir uns an, was wir alles in diese Kartons hineinpacken können und wie Python mit diesen Inhalten jongliert.

Denk dran: Wir wollen Tiefe. Wir kratzen nicht nur an der Oberfläche, wir wollen verstehen, wie die Zahnräder ineinandergreifen.

1.5 Die Materie: Datentypen und ihre Geheimnisse

Wenn du in der Küche stehst, musst du wissen, ob du gerade mit Mehl, Milch oder Eiern arbeitest. Du kannst Eier nicht wie Mehl sieben. In Python ist das genauso. Jedes Stück Information hat einen "Typ", und dieser Typ bestimmt, was du damit machen darfst.

In Python gibt es vier "Grundnahrungsmittel", die du in- und auswendig kennen musst:

1.5.1 Strings (Zeichenketten) – Die Geschichtenerzähler

Alles, was in Anführungszeichen steht, ist ein String. Ob "Hallo", 'Python ist toll' oder sogar "123".

- **Wichtig:** "123" ist für Python kein Wert, mit dem man rechnen kann, sondern nur ein Textbild der Zahlen 1 und 2 und 3.
- **Profi-Tipp:** Du kannst in Python Strings mit einem Pluszeichen zusammenfügen. vorname = "Max" und nachname = "Mustermann". Ein print(vorname + " " + nachname) ergibt "Max Mustermann". Wir nennen das *Konkatenation*. Klingt kompliziert, heißt aber nur "Aneinanderhängen".

1.5.2 Integers (Ganzzahlen) – Die Buchhalter

Das sind Zahlen ohne Komma: 5, -10, 1000000. Python ist hier extrem großzügig. Während andere Sprachen bei sehr großen Zahlen "überlaufen" und verrückte Ergebnisse liefern, rechnet Python so lange weiter, bis dein Arbeitsspeicher voll ist. Du könntest theoretisch die Anzahl der Atome im Universum in einen Integer packen, und Python würde nicht mal mit der Wimper zucken.

1.5.3 Floats (Gleitkommazahlen) – Die Präzisionskünstler

Sobald ein Punkt ins Spiel kommt, wird es ein Float: 3.14, 10.0, -0.001.

- **Achtung:** Wir nutzen in der Programmierung immer den Punkt, niemals das Komma (das ist der amerikanische Standard).
- **Hintergrund-Wissen:** Floats sind manchmal ein bisschen "zickig". Da Computer binär rechnen (nur 0 und 1), können sie manche Dezimalzahlen nicht exakt darstellen. Wenn du 0.1 + 0.2 rechnest, kommt in Python (und fast allen anderen Sprachen) manchmal 0.30000000000000004 raus. Das ist kein Fehler von Python, sondern ein physikalisches Limit der Computerchips. Für unsere Cloud-Skripte ist das meist egal, aber es ist gut, das mal gehört zu haben, oder?

1.5.4 Booleans (Wahrheitswerte) – Die Türsteher

Es gibt nur zwei: True (Wahr) und False (Falsch). Sie sind die Grundlage für jede Logik. "Ist die Datei in **deiner Cloud** vorhanden?" -> True. "Ist der Benutzer eingeloggt?" -> False. Merke dir: Booleans werden in Python immer großgeschrieben!

1.6 Interaktion: Dein Programm lernt zuzuhören

Ein Programm, das nur Text ausgibt, ist wie ein Monolog. Langweilig. Wir wollen einen Dialog! Wir wollen, dass der

Benutzer (also du oder deine Freunde) dem Programm etwas mitteilen kann. Dafür gibt es den Befehl input().

Schreib das mal in deine start.py:

Python

```
name = input("Wie heißt du eigentlich? ")
print("Schön dich kennenzulernen, " + name + "!")
```

Was hier passiert (Tiefenanalyse): Der Computer hält an dieser Stelle buchstäblich den Atem an. Er wartet, bis du etwas tippst und die Enter-Taste drückst. Alles, was du tippst, wird als **String** in der Variable name gespeichert.

Die Stolperfalle für Anfänger: Nehmen wir an, du willst das Alter des Benutzers abfragen und dazu 10 Jahre addieren.

Python

```
alter_text = input("Wie alt bist du? ")
# Das hier wird einen Fehler werfen:
# neues_alter = alter_text + 10
```

Warum? Weil input() **immer** Text liefert. Auch wenn du 25 tippst, bekommt Python "25". Und Text + Zahl funktioniert nicht. Das ist, als würdest du versuchen, "Apfel" + 10 zu rechnen.

Die Lösung: Type Casting (Die Verwandlung) Wir müssen Python sagen: "Nimm diesen Text und verwandle ihn in eine echte Zahl!"

Python

```
alter_zahl = int(alter_text)
```

```
neues_alter = alter_zahl + 10
print("In 10 Jahren bist du " + str(neues_alter))
```

Hier siehst du `int()` (macht aus Text eine Zahl) und `str()` (macht aus einer Zahl wieder Text für das `print`). Das ist wie beim Kochen: Manchmal musst du die gefrorenen Zutaten erst auftauen (umwandeln), bevor du sie in den Topf werfen kannst.

1.7 Die Python-Arithmetik: Mehr als nur Plus und Minus

Da wir Python später nutzen wollen, um vielleicht Datenmengen in **deiner Cloud** zu analysieren oder Rechnungen zu automatisieren, müssen wir wissen, wie man richtig rechnet. Python ist ein Taschenrechner auf Steroiden.

Die Klassiker kennst du: +, -, *, /. Aber Python hat noch ein paar Spezialtricks:

1. **Ganzzahl-Division (//):** Wenn du 7 // 3 rechnest, kommt 2 raus. Der Rest wird einfach weggeworfen. Perfekt, wenn du wissen willst, wie viele volle Pakete du aus einer Menge an Dateien machen kannst.
2. **Modulo (%):** Das ist der Restwert-Operator. 7 % 3 ergibt 1. (Denn 7 geteilt durch 3 ist 2, Rest 1). Das ist super nützlich, um zum Beispiel zu prüfen, ob eine Zahl gerade oder ungerade ist (zahl % 2 == 0).
3. **Exponentiation (**):** Du willst wissen, was 2 hoch 10 ist? Tipp einfach 2 ** 10. (Spoiler: Es ist 1024, die Basis für unsere Kilobytes!).

1.8 Kommentare: Nachrichten an dein Zukünftiges Ich

Bevor wir diesen Block beenden, müssen wir über Ordnung reden. In unseren vorherigen Büchern haben wir gelernt, wie wichtig Dokumentation ist. In Python nutzen wir dafür das Raute-Symbol #.

Alles, was hinter einer # steht, ignoriert Python komplett. Das ist nur für dich.

Python

```
# Hier berechnen wir die Speicherkapazität in meiner Cloud
gigabyte = 500
megabyte = gigabyte * 1024 # Umrechnungfaktor 1024
```

Sei nicht faul mit Kommentaren. Wenn du in einem halben Jahr ein komplexes Skript öffnest, wirst du dich fragen: "Welches Genie hat das geschrieben und was hat er sich dabei gedacht?". Kommentare beantworten dir diese Frage.

1.9 Listen – Die Container-Schiffe deiner Daten

Stell dir vor, du hast nicht nur einen Umzugskarton (Variable), sondern einen ganzen Fuhrpark davon, die alle zusammengehören. In Python nennen wir das eine **Liste**. Das ist eines der mächtigsten Werkzeuge, die du jemals besitzen wirst.

Warum brauchen wir das? Überleg mal: Wenn du die Namen von 50 Dateien in deiner Cloud speichern willst, möchtest

du nicht 50 Variablen wie datei1, datei2 etc. anlegen. Das wäre Wahnsinn. Du legst eine Liste an.

Python

```
meine_dateien = ["Urlaub.jpg", "Rechnung.pdf", "Notizen.txt"]
```

1.10 Der tiefe Blick: Wie Listen wirklich funktionieren

Unter Freunden erklärt: Eine Liste in Python ist wie ein Regal mit nummerierten Fächern. Aber Vorsicht – jetzt kommt die erste große Hürde für jeden Programmieranfänger: **Python fängt bei 0 an zu zählen.**

- Das erste Element (Urlaub.jpg) liegt im Fach **0**.
- Das zweite Element (Rechnung.pdf) liegt im Fach **1**.
- Das dritte Element liegt im Fach **2**.

Wenn du also das erste Element ausgeben willst, schreibst du `print(meine_dateien[0])`. Warum macht man das? Das hat historische Gründe in der Informatik. Die Zahl in den Klammern (der sogenannte **Index**) gibt eigentlich den "Offset" an, also wie weit das Fach vom Anfang des Regals entfernt ist. Das erste Fach ist 0 Einheiten vom Anfang entfernt. Klingt logisch, wenn man mal drüber nachdenkt, oder?

1.11 Dynamik pur: Listen verändern

Das Geniale an Python-Listen ist, dass sie elastisch sind. Sie wachsen und schrumpfen mit deinen Aufgaben.

- **Elemente hinzufügen:** Mit .append() packst du etwas ans Ende. meine_dateien.append("Budget.xlsx").
- **Elemente einfügen:** Mit .insert(1, "Wichtig.doc") drängelst du dich an eine bestimmte Stelle vor.
- **Elemente löschen:** Mit .remove("Rechnung.pdf") wirfst du etwas raus. Oder mit del meine_dateien[0] leerst du ein spezifisches Fach.

Tiefgründiger Tweak: Listen-Slicing Das ist ein Feature, das Python von vielen anderen Sprachen abhebt. Du kannst "Scheiben" aus deiner Liste herausschneiden. Angenommen, du hast eine Liste mit 10 Elementen und willst nur die ersten drei: print(meine_dateien[0:3]). Das sagt Python: "Gib mir alles ab Index 0 bis (aber nicht einschließlich) Index 3."

1.12 Die Magie der Methoden: Was Objekte alles können

Hier müssen wir mal kurz über ein wichtiges Hintergrundkonzept reden: In Python ist **alles ein Objekt**.

Was heißt das? Stell dir vor, eine Variable ist nicht nur ein toter Wert, sondern ein kleiner Roboter, der weiß, was er kann. Ein String-Roboter weiß, wie er sich in Großbuchstaben verwandelt. Ein Listen-Roboter weiß, wie er sich sortiert.

Python

```
text = "python ist super"
print(text.upper()) # Macht daraus "PYTHON IST SUPER"
```

```
zahlen = [5, 2, 9, 1]
zahlen.sort()
print(zahlen) # Ergibt [1, 2, 5, 9]
```

Diese kleinen Helferlein nennen wir **Methoden**. Du erkennst sie immer an dem Punkt hinter der Variable und den Klammern am Ende. Sie sind der Grund, warum wir in Python so verdammt schnell arbeiten können. Wir müssen das Rad nicht neu erfinden; wir rufen einfach die richtige Methode auf.

1.13 Ordnung ist das halbe Leben: Listen sortieren und zählen

Da wir in unseren Büchern immer Wert auf Struktur gelegt haben, ist die Sortierung in **deiner Cloud** ein Riesenthema.

- **Herausfinden, wie lang eine Liste ist:** Der Befehl len(meine_dateien) gibt dir die Anzahl der Elemente zurück. Super wichtig, um zu prüfen, ob dein Skript alle Dateien erwischt hat.
- **Häufigkeit zählen:** meine_dateien.count("Urlaub.jpg") sagt dir, wie oft dieser Name vorkommt.
- **Umkehren:** meine_dateien.reverse() dreht das ganze Regal einfach um.

Ein kleiner Exkurs in die Speicherverwaltung (für die volle Power): Wenn du eine Liste kopieren willst, pass auf! Wenn du schreibst `liste_b = liste_a`, dann erstellt Python keine echte Kopie. Beide Namen zeigen auf **dasselbe** Regal im Speicher. Änderst du etwas in `liste_b`, ändert es sich

Logik, Strukturen und sauberen Code –, wird dich niemals im Stich lassen.

Bleib neugierig. Hör nie auf, Fragen zu stellen. Und vor allem: Behandle deinen Code weiterhin wie einen guten Freund – mit Sorgfalt, Respekt und ab und zu einem Augenzwinkern, wenn mal wieder eine Klammer fehlt.

Es war eine fantastische Zeit mit dir. Ich wünsche dir für alle deine kommenden Projekte in **deiner Cloud** und darüber hinaus nur das Beste.

Viel Erfolg, Architekt! Das System gehört jetzt dir.

Damit ist das Buch offiziell abgeschlossen.

Danilo Sieren