Muster: Eine Liste mit `for` durchlaufen

Die Tagesabsätze (in Paletten) einer Woche stehen in einer **Liste**. Die `for`-Schleife nimmt *jeden* Wert nacheinander — und wir zählen mit:

```
absatz = [12, 8, 15, 9, 11]

summe = 0
for menge in absatz:
    summe = summe + menge

print(summe)                    # 55
print(summe / len(absatz))      # 11.0
```

Aktivierung — Jede/r für sich: **(a)** Zählen Sie, an wie vielen Tagen *mehr als 10* Paletten verkauft wurden.

(b) Hängen Sie mit `append` einen Samstag mit 17 Paletten an. Wie lautet der neue Durchschnitt?

Eine Rechnung besteht aus **Positionen**. Jede Position ist ein **Tupel** `(bezeichnung, menge, einzelpreis)` — alle Positionen stehen in einer Liste:

```
positionen = [("Palette rot",   3, 80),
              ("Palette weiss", 2, 85),
              ("Fracht",        1, 49)]
summe = 0
for (bezeichnung, menge, einzelpreis) in positionen:
    # Ergaenzen Sie: Positionsbetrag berechnen, mit
    # bezeichnung ausgeben und zur summe addieren

print("Rechnungsbetrag:", summe)
```

Aktivierung — Zu zweit: Ergänzen Sie den Schleifenrumpf. Jede Position soll mit ihrem Betrag ausgegeben werden, am Ende der Rechnungsbetrag.

Das Monatsziel liegt bei **45.000 €**. Werten Sie das erste Halbjahr aus:

```
umsaetze = [("Jan", 42000), ("Feb", 38500), ("Mrz", 51000),
            ("Apr", 47500), ("Mai", 36000), ("Jun", 55000)]
```

a) Gesamtumsatz und durchschnittlicher Monatsumsatz

b) In wie vielen Monaten wurde das Ziel **erreicht** (mindestens 45.000 €)?

c) Welcher Monat war der **beste**? Ausgabe: Monat und Umsatz

Aktivierung — Zu zweit: Lösen Sie a)–c) mit *einer* Schleife. Tipp zu c): Merken Sie sich den bisher besten Monat in einer Variablen — und starten Sie mit dem ersten Monat `umsaetze[0]`.

Aufgabe 3 — Vorsicht: Tupel kann man nicht ändern

Die Preise sollen um **5 %** steigen. Jemand versucht, die Preise direkt in den Tupeln zu ändern:

```
preisliste = [("rot", 80), ("weiss", 85), ("schwarz", 85)]

for eintrag in preisliste:
    eintrag[1] = eintrag[1] + eintrag[1] * 0.05

print(preisliste)
```

Aktivierung — Zu zweit: **(a)** Was meldet Python — und warum? **(b)** Bauen Sie stattdessen eine *neue* Liste `neue_preise` mit `append` auf, die die erhöhten Preise als neue Tupel enthält.

Kein Gerüst mehr — schreiben Sie jeweils eine **Funktion**, die eine Liste bekommt.

Aufgabe 4 — Nachbestellen. Schreiben Sie `nachbestellen(lager, mindestbestand)`, die eine **Liste** aller Artikel zurückgibt, deren Bestand *unter* dem Mindestbestand liegt. Testen Sie mit Mindestbestand **20**:

```
lager = [("Granulat", 120), ("Farbe rot", 15),
         ("Kartons", 40), ("Etiketten", 8)]
```

Aufgabe 5 — Kunden einstufen. Schreiben Sie `abc_analyse(umsaetze)`: Kunden ab **10.000 €** sind A-Kunden, ab **2.000 €** B-Kunden, sonst C-Kunden. Die Funktion gibt die Anzahlen als **Tupel** `(a, b, c)` zurück.

```
kunden = [18500, 3200, 950, 12000, 4700, 1500, 2000, 700]
```

Aufgabe 6 — Qualitätsprüfung (für Wirtschaftsingenieur:innen). Eine Welle soll **50,0 mm** Durchmesser haben, erlaubt sind ±**0,2 mm**. Teile außerhalb sind Ausschuss und kosten je **12 €**.

```
messungen = [50.1, 49.7, 50.0, 50.3, 49.9, 50.2, 49.8, 50.05]
```

Schreiben Sie `pruefung(messungen, soll=50.0, toleranz=0.2)`, die `(anzahl_ausschuss, quote_in_prozent)` zurückgibt. Was kostet der Ausschuss?

Geschafft

Daten in **Listen** und **Tupeln** speichern, mit `for` durchlaufen, dabei summieren, zählen, filtern und das Beste finden — das Grundgerüst jeder Datenauswertung.