DS 12 – Algorithmen auf Listen: Summe, Minimum und Suche

Informatik 10 – MINT-Zweig | Datenstrukturen

Lernziele

Ich kann

- mit einer Schleife über die Elemente oder über die Indizes einer Liste laufen und die passende Art auswählen,
- die Summe und den Mittelwert aller Elemente einer Liste berechnen,
- das kleinste und das größte Element einer Liste bestimmen,
- mit der linearen Suche herausfinden, ob und an welchem Index ein Wert in einer Liste vorkommt,
- zählen, wie oft ein Wert in einer Liste vorkommt,
- meine Ergebnisse mit `sum()`, `min()`, `max()` und `sorted()` kontrollieren.

🗣️ Sprachhilfe

- *„Die Funktion findet Tom an Stelle 2."* → „Der Funktion `lineare_suche` werden die Liste `namen` und der Wert `"Tom"` übergeben. Sie durchläuft die Liste Element für Element und gibt den Index 2 zurück."
- *„Bei Lea kommt −1."* → „Ist der gesuchte Wert nicht in der Liste enthalten, gibt die Funktion −1 zurück."
- *„maximum merkt sich die größte Zahl."* → „Die Hilfsvariable `maximum` wird mit dem ersten Listenelement initialisiert und aktualisiert, sobald ein größeres Element gefunden wird."

Worum es geht

In einer Liste stehen die Noten einer Klassenarbeit. Wie gut war die Klasse im Durchschnitt? Was war die beste Note? Wie viele Einsen gab es? Ein Computer kann nicht „auf einen Blick" sehen. Er muss die Liste **Element für Element** durchgehen. Ein **Algorithmus** ist eine genaue Schritt-für-Schritt-Anleitung, so eindeutig, dass auch ein Computer sie ausführen kann.

1 Zwei Arten, eine Liste zu durchlaufen

Schleife	Schleifenvariable	geeignet, wenn …
`for element in liste:`	jedes **Element**	nur Werte nötig sind
`for i in range(len(liste)):`	jeder **Index**	die Position nötig ist

🔮 **Vorhersage:** Beide Zellen sollen alle Preise verdoppeln. Welche schafft es?

```
preise = [2.0, 3.5, 1.0]
for i in range(len(preise)):
    preise[i] = preise[i] * 2
print(preise)
```

```
[4.0, 7.0, 2.0]
```

```python
preise = [2.0, 3.5, 1.0]
for preis in preise:
    preis = preis * 2
print(preise)
```

```
[2.0, 3.5, 1.0]
```

In der ersten Zelle steht ein schreibender Zugriff `preise[i] = …`. In der zweiten bekommt `preis` in jedem Durchlauf nur den **Wert** eines Elements. `preis = preis * 2` ändert nur diese Variable, die Liste bekommt davon nichts mit. Es gibt keine Fehlermeldung, die Liste bleibt einfach gleich.

> 📌 **Merksatz:** Über eine Liste kann man **über die Elemente** (`for element in liste:`) oder **über die Indizes** (`for i in range(len(liste)):`) iterieren. Zum Verändern der Liste braucht man die Indizes.

2 Summenbildung

An der Supermarktkasse werden die Preise nacheinander zu einer **Zwischensumme** addiert. Am Anfang steht sie auf 0, bei jedem Artikel kommt sein Preis dazu. Genauso arbeitet der Algorithmus. Die Zwischensumme ist eine **Hilfsvariable**.

```python
zahlen = [7, 2, 9, 4, 1, 8]
summe = 0
for zahl in zahlen:
    summe = summe + zahl
print("Summe:", summe)
```

```
Summe: 31
```

- **Zeile 2:** `summe` wird **vor** der Schleife mit 0 **initialisiert**.
- **Zeilen 3–4:** In jedem Durchlauf wird das aktuelle Element addiert.
- **Zeile 5:** Erst **nach** der Schleife steht das Endergebnis in `summe`.

Ablaufprotokoll für die kürzere Liste `[7, 2, 9]`:

Schritt	Zeile	Anweisung	`zahl`	`summe`
1	1	`zahlen = [7, 2, 9]`	-	-
2	2	`summe = 0`	-	`0`
3	3	`for zahl in zahlen:`	`7`	`0`
4	4	`summe = summe + zahl`	`7`	`7`
5	3	`for zahl in zahlen:`	`2`	`7`
6	4	`summe = summe + zahl`	`2`	`9`
7	3	`for zahl in zahlen:`	`9`	`9`
8	4	`summe = summe + zahl`	`9`	`18`
9	3	`for zahl in zahlen:`	`9`	`18`
10	5	`print("Summe:", summe)`	`9`	`18`

> ⚠️ **Stolperstein:** Steht `summe = 0` **in** der Schleife, wird die Zwischensumme bei jedem Durchlauf wieder auf 0 gesetzt. Am Ende steht dann nur das letzte Element in `summe`.

Der Mittelwert

Den **Mittelwert** (Durchschnitt) erhältst du, wenn du die Summe durch die Anzahl `len(liste)` teilst. Das funktioniert für jede Liste, egal wie viele Noten darin stehen.

```
noten = [2, 3, 1, 4, 2]
summe = 0
for note in noten:
    summe = summe + note
mittelwert = summe / len(noten)
print("Durchschnitt:", mittelwert)
```

```
Durchschnitt: 2.4
```

> 📌 **Merksatz: Summenbildung:** Eine Hilfsvariable wird vor der Schleife auf `0` gesetzt und in jedem Durchlauf um das aktuelle Element erhöht. **Mittelwert** = Summe / `len(liste)`.

3 Maximum und Minimum

Stell dir vor, du deckst Spielkarten nacheinander auf. Die erste Karte behältst du in der Hand. Bei jeder weiteren vergleichst du: Ist die neue Zahl größer? Dann tauschst du. Am Ende hältst du die größte Zahl in der Hand. Die „Karte in der Hand“ ist wieder eine Hilfsvariable.

```
zahlen = [7, 2, 9, 4, 1, 8]
maximum = zahlen[0]
for zahl in zahlen:
    if zahl > maximum:
        maximum = zahl
print("Maximum:", maximum)
```

```
Maximum: 9
```

- **Zeile 2:** `maximum` wird mit dem **ersten Element** initialisiert, der ersten Karte.
- **Zeilen 4–5:** Nur wenn das aktuelle Element größer ist, wird `maximum` **aktualisiert**.

Für das **Minimum** dreht man den Vergleich um: `<` statt `>`.

⚠️ Stolperstein: Warum nicht mit 0 beginnen?

🔮 **Vorhersage:** Was gibt das Programm aus? Was wäre richtig?

```
temperaturen = [-5, -2, -8]
maximum = 0
for t in temperaturen:
    if t > maximum:
        maximum = t
print("Höchste Temperatur:", maximum)
```

```
Höchste Temperatur: 0
```

Keine Temperatur ist größer als 0, also wird `maximum` nie aktualisiert. Ausgegeben wird 0, ein Wert, der gar nicht in der Liste steht. Richtig wäre −2. Mit `maximum = temperaturen[0]` stimmt das Ergebnis.

> 📌 **Merksatz: Minimum/Maximum:** Eine Hilfsvariable wird mit dem **ersten Element** initialisiert und bei jedem kleineren bzw. größeren Fund aktualisiert.

4 Lineare Suche und Zählen

Du suchst in einem unsortierten Stapel Klassenarbeiten die von Tom. Du nimmst die oberste, schaust auf den Namen, dann die nächste, bis du Toms Arbeit gefunden hast oder der Stapel zu Ende ist. Das ist die **lineare Suche**. Die Funktion gibt den **Index** des Treffers zurück, sonst `-1`.

```
1 def lineare_suche(liste, gesuchter_wert):
2     for i in range(len(liste)):
3         if liste[i] == gesuchter_wert:
4             return i
5     return -1
6
7 namen = ["Ben", "Ada", "Tom", "Mia"]
8 print(lineare_suche(namen, "Tom"))
9 print(lineare_suche(namen, "Lea"))
```

```
2
-1
```

- **Zeile 2:** Weil der Index zurückgegeben werden soll, läuft die Schleife über die **Indizes**.
- **Zeile 4:** Bei einem Treffer beendet `return i` die Funktion **sofort**, auch mitten in der Schleife. Mia wird gar nicht mehr angeschaut.
- **Zeile 5:** wird nur erreicht, wenn die Schleife ohne Treffer zu Ende gelaufen ist.

Warum `-1`? Von vorne gezählt ist ein Index nie negativ. Aber Vorsicht: `namen[-1]` wäre `"Mia"`! Prüfe das Ergebnis also mit `if position == -1:`, bevor du es als Index benutzt.

⚠️ Stolperstein: `return -1` im `else`

🔮 **Vorhersage:** Tom steht in der Liste. Was gibt diese Fassung aus, und warum?

```
1 def lineare_suche(liste, gesuchter_wert):
2     for i in range(len(liste)):
3         if liste[i] == gesuchter_wert:
4             return i
5         else:
6             return -1
7
8 namen = ["Ben", "Ada", "Tom", "Mia"]
9 print(lineare_suche(namen, "Tom"))
```

```
-1
```

Beim ersten Element `"Ben"` ist der Vergleich falsch, der `else`-Teil wird ausgeführt, und `return` beendet die Funktion sofort. „Nicht gefunden“ steht erst fest, wenn **alle** Elemente verglichen wurden, also **nach** der Schleife.

> 📌 **Merksatz: Lineare Suche:** Man durchläuft die Liste Element für Element und vergleicht mit dem Suchwert, bis er gefunden wurde oder die Liste zu Ende ist.

Zählen

Beim Zählen führt man eine **Strichliste**: Die Hilfsvariable beginnt bei 0 und wird um **1** erhöht, wenn ein Element die Bedingung erfüllt.

```python
noten = [2, 3, 1, 4, 2, 5, 1]
anzahl = 0
for note in noten:
    if note <= 2:
        anzahl = anzahl + 1
print("Anzahl der Noten 1 oder 2:", anzahl)
```

```
Anzahl der Noten 1 oder 2: 4
```

> 📌 **Merksatz: Zählen:** Hilfsvariable mit 0 beginnen und um 1 erhöhen, wenn ein Element die Bedingung erfüllt.

5 Übungen

🧰 Werkzeug: fertige Funktionen zur Kontrolle

Python bringt für diese Aufgaben fertige Funktionen mit. Benutze sie, um deine eigenen Funktionen zu prüfen, zum Beispiel `print(minimum(zahlen) == min(zahlen))`.

Funktion	Ergebnis
`sum(liste)`	die Summe aller Elemente
`min(liste)`, `max(liste)`	das kleinste bzw. größte Element
`sorted(liste)`	eine **neue**, aufsteigend sortierte Liste
`wert in liste`	`True`, wenn `wert` vorkommt

```python
zahlen = [7, 2, 9, 4, 1, 8]
print(sum(zahlen))
print(min(zahlen), max(zahlen))
print(sorted(zahlen))
print(zahlen)
print(9 in zahlen)
```

```
31
1 9
[1, 2, 4, 7, 8, 9]
[7, 2, 9, 4, 1, 8]
True
```

Schritt für Schritt selbstständig

Vorgemacht sind die Muster **Summe** und **Maximum** oben. Bei Übung 1 füllst du Lücken `___` aus, bei Übung 2 steht ein Gerüst aus Kommentaren, bei Übung 3 nur die Kopfzeile und der Test. Übung 4 ist eine Leseaufgabe, die Zusatzaufgabe schreibst du allein.

📝 Übung 1 – Durchschnitt berechnen (Lücken füllen)

Schreibe eine Funktion `durchschnitt(liste)`, die den Durchschnitt aller Zahlen einer Liste zurückgibt. Kontrolle: `durchschnitt([7, 2, 9, 4, 1, 8])` ergibt etwa 5,17 (Summe 31, 6 Elemente).

Lückencode

```python
def durchschnitt(liste):
    summe = ___
    for wert in liste:
        summe = ___
    return ___ / len(liste)

print(durchschnitt([7, 2, 9, 4, 1, 8]), "erwartet: etwa 5.17")
```

📝 Übung 2 – Minimum finden (mit Gerüst)

Schreibe eine Funktion `minimum(liste)`, die das kleinste Element zurückgibt, ohne `min()` zu verwenden. Teste mit `[7, 2, 9, 4, 1, 8]` und mit einer Liste aus negativen Zahlen. Vergleiche mit `min()`.

```
def minimum(liste):
    # Hilfsvariable mit dem ersten Element initialisieren
    # für jeden Wert in der Liste:
    #     wenn der Wert kleiner ist als die Hilfsvariable:
    #         Hilfsvariable aktualisieren
    # nach der Schleife: Hilfsvariable zurückgeben
    pass

print(minimum([7, 2, 9, 4, 1, 8]), "erwartet: 1")
print(minimum([-5, -2, -8]), "erwartet: -8")
```

```
None erwartet: 1
None erwartet: -8
```

📝 Übung 3 – Häufigkeit zählen (Kopfzeile und Test vorgegeben)

Schreibe eine Funktion `zaehle_vorkommen(liste, wert)`, die zählt, wie oft `wert` in `liste` vorkommt. Anders als die lineare Suche soll sie **alle** Vorkommen zählen. Kontrolle: `zaehle_vorkommen([1, 2, 2, 3, 2, 4], 2)` ergibt 3.

```
def zaehle_vorkommen(liste, wert):
    pass   # hier deine Programmzeilen (Vorbild: Zählen mit Strichliste)

print(zaehle_vorkommen([1, 2, 2, 3, 2, 4], 2), "erwartet: 3")
```

```
None erwartet: 3
```

🔍 Übung 4 – Code lesen

Lies die Funktion, **ohne** sie auszuführen.

1. Gib an, was das Programm ausgibt.
2. Beschreibe in einem Satz, was `geheim` allgemein berechnet.
3. Erkläre, warum das Ergebnis nicht `3` ist, obwohl am Index 3 auch eine 8 steht.

```
def geheim(liste):
    position = 0
    for i in range(len(liste)):
        if liste[i] > liste[position]:
            position = i
    return position

print(geheim([3, 8, 5, 8, 1]))
```

```
1
```

Meine Antworten:

…

🚀 Zusatzaufgabe – Zwei größte Werte (selbstständig)

Finde mit einer Schleife die **zwei größten** Werte der Liste und gib sie aus, ohne `sorted()`. Kontrolle: 22 und 17.

> 💡 **Tipp:** Merke dir in **zwei** Hilfsvariablen den größten und den zweitgrößten Wert. Initialisiere sie mit den ersten beiden Elementen, in der richtigen Reihenfolge. Ist ein weiteres Element größer als der größte Wert,

> oder nur größer als der zweitgrößte?

```
zahlen = [4, 17, 9, 22, 3, 15]

# Zusatzaufgabe – zwei größte Werte finden
```

📌 Das Wichtigste in Kürze

- Über eine Liste iteriert man über die Elemente (`for element in liste:`) oder über die Indizes (`for i in range(len(liste)):`). Zum Verändern braucht man die Indizes.
- **Summe:** Hilfsvariable vor der Schleife auf 0 setzen, in jedem Durchlauf das Element addieren. **Mittelwert** = Summe / `len(liste)`.
- **Maximum/Minimum:** Hilfsvariable mit dem ersten Element initialisieren und bei jedem größeren bzw. kleineren Element aktualisieren.
- **Lineare Suche:** Element für Element vergleichen, bis der Wert gefunden oder die Liste zu Ende ist. Ergebnis: der Index (oder `-1`) bzw. `True`/`False`.
- **Zählen:** Hilfsvariable mit 0 beginnen und um 1 erhöhen, wenn ein Element die Bedingung erfüllt.
- `sum`, `min`, `max`, `sorted` und `in` erledigen das fertig, gut zur Kontrolle eigener Funktionen.

Zum Weiterüben: `DS12_Weitere_Uebungen.ipynb`.