1.12 Algorithmen auf Listen: Summe, Minimum und Suche

Leitidee

In einer Liste stehen die Noten einer Klassenarbeit. Wie gut war die Klasse im Durchschnitt? Was war die beste Note? Hat jemand eine 6 geschrieben, und wenn ja, wie viele? Bei einer kurzen Liste siehst du das auf einen Blick. Ein Computer kann aber nicht „auf einen Blick“ sehen. Er muss die Liste **Element für Element** durchgehen. Ein **Algorithmus** ist eine genaue Schritt-für-Schritt-Anleitung, mit der man eine Aufgabe löst – so eindeutig, dass auch ein Computer sie ausführen kann. In diesem Abschnitt programmierst du einige grundlegende Algorithmen für Listen selbst: Summe und Mittelwert, Minimum und Maximum, die lineare Suche und das Zählen. Erst danach lernst du die fertigen Funktionen kennen, die Python dafür schon mitbringt.

Lernziele

Nach diesem Abschnitt kannst du

- 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,
- deine Ergebnisse mit `sum()`, `min()`, `max()` und `sorted()` kontrollieren.

1.12.1 Zwei Arten, eine Liste zu durchlaufen

Im Abschnitt über Listen hast du zwei Arten von `for`-Schleifen kennengelernt. Hier noch einmal beide nebeneinander:

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

Willst du Elemente einer Liste **verändern**, brauchst du die Schleife über die Indizes. Hier werden alle Preise verdoppelt:

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

AUSGABE

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

In Zeile 3 steht ein schreibender Zugriff: Das Element am Index `i` wird durch sein Doppeltes ersetzt.

Merke

Über eine Liste kann man auf zwei Arten iterieren (sie durchlaufen):

- **über die Elemente direkt:** `for element in liste:`
- **über die Indizes:** `for i in range(len(liste)):` – nützlich, wenn man den Index selbst braucht (z. B. für schreibenden Zugriff).

Stolperstein

Mit der Schleife über die Elemente kann man die Liste **nicht** verändern:

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

AUSGABE

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

Es gibt keine Fehlermeldung, aber die Liste ist unverändert. Die Variable `preis` bekommt in jedem Durchlauf den **Wert** eines Elements. Zeile 3 gibt nur der Variablen `preis` einen neuen Wert. Die Liste selbst bekommt davon nichts mit.

1.12.2 Summenbildung

An der Supermarktkasse werden die Preise nacheinander zu einer **Zwischensumme** addiert. Am Anfang steht die Zwischensumme auf 0. Bei jedem Artikel kommt sein Preis dazu. Nach dem letzten Artikel ist die Zwischensumme die Gesamtsumme.

Genauso arbeitet der Algorithmus zur **Summenbildung**. Die Zwischensumme ist eine **Hilfsvariable**, hier `summe`:

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

AUSGABE

```
Summe: 31
```

- **Zeile 3:** Die Hilfsvariable `summe` wird **vor** der Schleife auf 0 gesetzt. Man sagt: Sie wird mit 0 **initialisiert**.
- **Zeilen 4–5:** In jedem Durchlauf wird das aktuelle Element zu `summe` addiert.
- **Zeile 6:** Erst **nach** der Schleife steht in `summe` das Endergebnis. Deshalb ist `print` nicht eingerückt.

Wie sich `summe` Schritt für Schritt ändert, zeigt das Ablaufprotokoll. Damit es übersichtlich bleibt, steht hier eine kürzere Liste mit nur drei Zahlen:

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

Die `for`-Zeile wird viermal ausgeführt: dreimal mit einem neuen Element und ein viertes Mal, um festzustellen, dass die Liste zu Ende ist (Schritt 9).

Merke

Summenbildung: Eine Hilfsvariable wird vor der Schleife auf `0` gesetzt und in jedem Durchlauf um das aktuelle Element erhöht.

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`. Die Hilfsvariable muss deshalb immer **vor** der Schleife initialisiert werden.

Der Mittelwert

Den **Mittelwert** (Durchschnitt) erhältst du, wenn du die Summe durch die Anzahl der Elemente teilst. Die Anzahl liefert `len`:

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

AUSGABE

```
Durchschnitt: 2.4
```

Das Programm funktioniert für jede Liste, egal wie viele Noten darin stehen. Kommt eine Note dazu, muss man nichts am Programm ändern.

Arbeitsauftrag 144: Durchschnitt berechnen

Schreibe eine Funktion `durchschnitt(liste)`, die den Durchschnitt aller Zahlen in einer Liste berechnet und zurückgibt. Teste sie mit der Liste `[7, 2, 9, 4, 1, 8]`.

1.12.3 Maximum und Minimum

Stell dir vor, du deckst Spielkarten mit Zahlen nacheinander auf. Die erste Karte behältst du in der Hand. Bei jeder weiteren Karte vergleichst du: Ist die neue Zahl größer als die in deiner Hand? Dann tauschst du. Sonst legst du die neue Karte weg. Wenn alle Karten aufgedeckt sind, hältst du die größte Zahl in der Hand.

Im Programm ist die „Karte in der Hand“ wieder eine Hilfsvariable. Das folgende Programm bestimmt das **Maximum** derselben Liste wie oben:

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

AUSGABE

```
Maximum: 9
```

- **Zeile 2:** Die Hilfsvariable `maximum` wird mit dem **ersten Element** der Liste initialisiert. Das ist die erste Karte in der Hand.
- **Zeile 4:** In jedem Durchlauf wird geprüft, ob das aktuelle Element größer ist als das bisherige Maximum.
- **Zeile 5:** Nur wenn das der Fall ist, wird `maximum` **aktualisiert**: Es bekommt den neuen, größeren Wert.

Für das **Minimum** geht man genauso vor. Man dreht nur den Vergleich um: < statt >.

Merke

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

Das Ablaufprotokoll für die Liste `[7, 2, 9, 4]` zeigt, dass `maximum` sich nur einmal ändert – beim Element 9:

Schritt	Zeile	Anweisung	zahl	maximum
1	1	`zahlen = [7, 2, 9, 4]`	–	–
2	2	`maximum = zahlen[0]`	–	7
3	3	`for zahl in zahlen:`	7	7
4	4	`if zahl > maximum:`	7	7
5	3	`for zahl in zahlen:`	2	7
6	4	`if zahl > maximum:`	2	7
7	3	`for zahl in zahlen:`	9	7
8	4	`if zahl > maximum:`	9	7
9	5	`maximum = zahl`	9	9
10	3	`for zahl in zahlen:`	4	9
11	4	`if zahl > maximum:`	4	9
12	3	`for zahl in zahlen:`	4	9
13	6	`print("Maximum:", maximum)`	4	9

Stolperstein

Warum beginnt man mit dem ersten Element und nicht einfach mit 0? Weil 0 kein Element der Liste ist. Bei einer Liste aus lauter negativen Zahlen geht das schief:

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

AUSGABE

```
Höchste Temperatur: 0
```

Keine der Temperaturen ist größer als 0. Deshalb wird `maximum` nie aktualisiert, und das Programm gibt 0 aus – einen Wert, der gar nicht in der Liste steht. Richtig wäre −2. Mit `maximum = temperaturen[0]` stimmt das Ergebnis.

Arbeitsauftrag 145: Minimum finden

Schreibe eine Funktion `minimum(liste)`, die das kleinste Element einer Liste zurückgibt, ohne die eingebaute Funktion `min()` zu verwenden. Teste sie mit der Liste `[7, 2, 9, 4, 1, 8]`.

1.12.4 Lineare Suche

Du suchst in einem Stapel Klassenarbeiten die von Tom. Die Arbeiten sind nicht sortiert. Also nimmst du die oberste, schaust auf den Namen, dann die nächste – so lange, bis du Toms Arbeit gefunden hast oder der Stapel zu Ende ist. Dieses Vorgehen heißt **lineare Suche**: Man geht die Liste der Reihe nach durch, wie an einer Linie entlang.

Die folgende Funktion sucht einen Wert in einer Liste. Sie gibt den **Index** zurück, an dem der Wert steht. Kommt er nicht vor, gibt sie `-1` zurück:

```
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"))
```

AUSGABE

```
2
-1
```

- **Zeile 2:** Weil die Funktion den Index zurückgeben soll, läuft die Schleife über die **Indizes**.
- **Zeile 3:** Das Element am Index `i` wird mit dem gesuchten Wert verglichen.
- **Zeile 4:** Bei einem Treffer gibt `return i` den Index zurück. `return` beendet die Funktion **sofort** – auch mitten in der Schleife. Die restlichen Elemente werden nicht mehr angesehen.
- **Zeile 5:** Diese Zeile wird nur erreicht, wenn die Schleife bis zum Ende gelaufen ist, ohne etwas zu finden. Dann gibt die Funktion `-1` zurück.
- **Zeilen 8–9:** `"Tom"` steht am Index 2. `"Lea"` kommt nicht vor, deshalb lautet das Ergebnis `-1`.

Das Ablaufprotokoll zeigt die Zeilen **innerhalb** der Funktion beim Aufruf `lineare_suche(namen, "Tom")`:

Schritt	Zeile	Anweisung	i
1	2	`for i in range(len(liste)):`	0
2	3	`if liste[i] == gesuchter_wert:`	0
3	2	`for i in range(len(liste)):`	1
4	3	`if liste[i] == gesuchter_wert:`	1
5	2	`for i in range(len(liste)):`	2
6	3	`if liste[i] == gesuchter_wert:`	2
7	4	`return i`	2

Nach drei Vergleichen ist Tom gefunden (Schritt 6). In Schritt 7 wird `return i` ausgeführt, und die Funktion ist beendet. Mia wird gar nicht mehr angeschaut.

Warum gerade `-1`? Ein Index ist nie negativ, wenn man von vorne zählt. Deshalb ist `-1` ein eindeutiges Zeichen für „nicht gefunden“. Aber Vorsicht: Als **negativer Index** ist `-1` in Python gültig. `namen[-1]` wäre `"Mia"`! Prüfe das Ergebnis der Suche also immer, bevor du es als Index benutzt:

```
position = lineare_suche(namen, "Lea")
if position == -1:
    print("Lea ist nicht in der Liste.")
else:
    print("Lea steht am Index", position)
```

AUSGABE

```
Lea ist nicht in der Liste.
```

Manchmal will man nur wissen, **ob** ein Wert vorkommt. Dann kann die Suche einen Wahrheitswert zurückgeben. Den Index braucht man dafür nicht, also genügt die Schleife über die Elemente:

```
def enthaelt(liste, gesuchter_wert):
    for element in liste:
        if element == gesuchter_wert:
            return True
    return False

namen = ["Ben", "Ada", "Tom", "Mia"]
print(enthaelt(namen, "Mia"))
print(enthaelt(namen, "Lea"))
```

AUSGABE

```
True
False
```

Merke

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

Stolperstein

Ein sehr häufiger Fehler: `return -1` steht in einem `else` **innerhalb** der Schleife.

```
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"))
```

AUSGABE

```
-1
```

Tom steht in der Liste, trotzdem kommt `-1` heraus. Beim ersten Element (`"Ben"`) ist der Vergleich falsch, also wird der `else`-Teil ausgeführt – und `return` beendet die Funktion sofort. Die anderen Namen werden nie verglichen. „Nicht gefunden“ steht erst fest, wenn **alle** Elemente verglichen wurden, also **nach** der Schleife.

So kannst du es sagen

So beschreibst du einen Listen-Algorithmus in Fachsprache:

- *„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.“*

1.12.5 Zählen

Beim Zählen führt man eine **Strichliste**: Jedes Mal, wenn ein Element eine Bedingung erfüllt, kommt ein Strich dazu. Im Programm ist die Strichliste wieder eine Hilfsvariable, die mit 0 beginnt. Sie wird aber – anders als bei der Summe – nur um **1** erhöht, und nur dann, wenn die Bedingung erfüllt ist:

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

AUSGABE

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

Die Noten 2, 1, 2 und 1 erfüllen die Bedingung `note <= 2`. Bei jeder von ihnen wird `anzahl` um 1 erhöht.

Arbeitsauftrag 146: Häufigkeit zählen

Schreibe eine Funktion `zaehle_vorkommen(liste, wert)`, die zählt, wie oft `wert` in `liste` vorkommt. Anders als die lineare Suche, die beim ersten Treffer aufhört, soll die Funktion **alle** Vorkommen zählen. Teste sie mit der Liste `[1, 2, 2, 3, 2, 4]` und dem Wert `2`.

1.12.6 Fertige Funktionen zur Kontrolle

Python bringt für viele dieser Aufgaben fertige Funktionen mit. Du kannst sie ab jetzt benutzen – zum Beispiel, um deine eigenen Funktionen zu überprüfen:

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

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

AUSGABE

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

Beachte Zeile 5: `sorted` verändert die ursprüngliche Liste nicht. Es erzeugt eine neue, sortierte Liste. Den Operator `in` kennst du schon von den Zeichenketten. Bei Listen fragt er, ob ein Element mit diesem Wert vorkommt – im Hintergrund ist das eine lineare Suche.

Warum hast du die Algorithmen dann überhaupt selbst programmiert? Erstens verstehst du nun, was in den fertigen Funktionen passiert. Zweitens gibt es viele ähnliche Aufgaben, für die es keine fertige Funktion gibt: zum Beispiel den **Index** des größten Elements oder die zwei größten Werte. Die kannst du jetzt lösen, indem du die Algorithmen abwandelst.

Tipp

So testest du eine eigene Funktion mit einer fertigen: `print(minimum(zahlen) == min(zahlen))` muss `True` ausgeben. Probiere dabei verschiedene Listen aus, auch eine mit negativen Zahlen und eine mit nur einem Element.

1.12.7 Übungen

Arbeitsauftrag 147: Code lesen

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

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

Deine Antworten:

Arbeitsauftrag 148 (Zusatz): Zwei größte Werte

Finde mit einer Schleife die **zwei größten** Werte der Liste `zahlen = [4, 17, 9, 22, 3, 15]` und gib sie aus. Verwende dabei nicht `sorted()`.

Tipp

Merke dir in **zwei** Hilfsvariablen den größten und den zweitgrößten Wert. Initialisiere sie mit den ersten beiden Elementen (in der richtigen Reihenfolge!). Überlege dann für jedes weitere Element: Ist es größer als der bisher größte Wert? Oder nur größer als der zweitgrößte?

1.12.8 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 149: Maximum von Hand ●○○

Das Programm sucht das Maximum der Liste `[4, 9, 2, 9, 6]`:

```
zahlen = [4, 9, 2, 9, 6]
maximum = zahlen[0]
for zahl in zahlen:
    if zahl > maximum:
        maximum = zahl
print("Maximum:", maximum)
```

Fülle die Tabelle aus: Ist das Element größer als das bisherige Maximum? Welchen Wert hat `maximum` nach diesem Durchlauf? Gib dann an, wie oft `maximum` insgesamt geändert wird.

Durchlauf	`zahl`	`zahl > maximum`?	`maximum` danach
vor der Schleife	–	–	4
1	4		
2	9		
3	2		
4	9		
5	6		

Anzahl der Änderungen:

Arbeitsauftrag 150: Summe und Zählen ●○○

In der Liste stehen die Punkte, die ein Team in sechs Spielen geholt hat. Gib **ohne Computer** an, was das Programm ausgibt, und beschreibe in einem Satz, was `anzahl` zählt.

```
punkte = [3, 0, 1, 3, 0, 3]
summe = 0
anzahl = 0
for p in punkte:
    summe = summe + p
    if p == 0:
        anzahl = anzahl + 1
print(summe, anzahl)
```

Ausgabe und Erklärung:

Arbeitsauftrag 151: Playlist ●○○

In einer Liste steht die Länge jedes Songs einer Playlist in Minuten: `songs = [3, 4, 2, 5, 3]`.

1. Implementiere eine Funktion `gesamtdauer(liste)`, die die Summe aller Elemente zurückgibt.
2. Implementiere eine Funktion `mittlere_dauer(liste)`, die die durchschnittliche Länge zurückgibt. Sie soll `gesamtdauer` aufrufen.
3. Teste beide Funktionen mit der Playlist. Erwartet sind 17 Minuten und 3,4 Minuten.

Arbeitsauftrag 152: Kälteste Nacht ●○○

In einer Woche wurden nachts diese Temperaturen gemessen (in °C): `nachts = [-3, 2, -7, 0, 4, -1, 1]`.

Implementiere eine Funktion `kaelteste(temperaturen)`, die die niedrigste Temperatur zurückgibt. Verwende nicht `min()`. Teste mit der Liste. Erwartet ist −7.

Arbeitsauftrag 153: Freies Schließfach ●●○

In der Liste `faecher` steht, wem welches Schließfach gehört. Ein leeres Fach ist mit der leeren Zeichenkette `""` gekennzeichnet:

```
faecher = ["Ada", "Ben", "", "Mia", ""]
```

1. Implementiere eine Funktion `erstes_freies(faecher)`, die mit einer linearen Suche den **Index** des ersten freien Fachs zurückgibt. Gibt es kein freies Fach, soll sie `-1` zurückgeben.
2. Gib im Hauptprogramm aus, welches Fach frei ist. Die Fächer sind für Menschen ab 1 nummeriert, also ist der Index 2 das Fach Nummer 3. Ist kein Fach frei, soll „Alle Fächer sind belegt.“ erscheinen.
3. Teste auch mit einer Liste ganz ohne freies Fach.

Arbeitsauftrag 154: Zwei Fehler im Maximum ●●○

Die Funktion soll die beste Tordifferenz einer Saison zurückgeben. Sie liefert falsche Ergebnisse.

```
1 def beste_differenz(differenzen):
2     beste = 0
3     for d in differenzen:
4         if d > beste:
5             beste = d
6         return beste
```

1. Teste die Funktion mit `[3, -1, 5]` (erwartet: 5) und mit `[-2, -1, -5]` (erwartet: −1). Notiere die Ergebnisse.
2. Erkläre die beiden Fehler und verbessere die Funktion.

Testergebnisse und Fehler:

Arbeitsauftrag 155: Frosttage ●●○

Ein Tag mit einer Temperatur **unter** 0 °C heißt Frosttag.

1. Implementiere eine Funktion `frosttage(temperaturen)`, die zählt, wie viele Werte der Liste unter 0 liegen.
2. Teste mit der Liste aus dem Auftrag *Kälteste Nacht* (erwartet: 3) und mit `[0, 1, 2]` (erwartet: 0).

Arbeitsauftrag 156: Besser als der Durchschnitt ●●●

Implementiere eine Funktion `ueber_durchschnitt(liste)`, die zählt, wie viele Elemente **größer** als der Durchschnitt der Liste sind. Teste mit:

Aufruf	erwartet	tatsächlich
`ueber_durchschnitt([2, 4, 6, 8])`	2	
`ueber_durchschnitt([10, 10, 10])`	0	
`ueber_durchschnitt([1, 1, 1, 9])`	1	

Tipp

Du brauchst **zwei** Schritte nacheinander. Erst berechnest du mit einer Schleife den Durchschnitt. Erst danach kannst du mit einer zweiten Schleife zählen, denn vorher kennst du den Vergleichswert noch nicht.

1.12.9 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.

Selbstcheck

- ☐ Ich kann entscheiden, ob ich über die Elemente oder über die Indizes einer Liste iterieren muss.
- ☐ Ich kann die Summe und den Mittelwert der Elemente einer Liste berechnen.
- ☐ Ich kann das Minimum und das Maximum einer Liste ohne `min` und `max` bestimmen und erklären, warum ich mit dem ersten Element beginne.
- ☐ Ich kann eine lineare Suche programmieren, die den Index oder `-1` zurückgibt.
- ☐ Ich kann erklären, warum `return -1` erst nach der Schleife stehen darf.
- ☐ Ich kann zählen, wie oft ein Wert in einer Liste vorkommt.
- ☐ Ich kann meine Ergebnisse mit `sum`, `min`, `max` und `sorted` überprüfen.

Gymnasium Ottweiler