1.15 Kapitel 1 kompakt: der Lernzettel

Leitidee

Hier steht das Wichtigste aus Kapitel 1 auf wenigen Seiten, zum Wiederholen vor einem GLN. Wenn dir etwas unklar ist, lies im passenden Abschnitt nach: Dort ist alles ausführlich erklärt, mit Beispielen und Aufgaben.

1.15.1 Variablen, Ein- und Ausgabe (Abschnitt 1.1)

- Eine **Variable** ist ein benannter Speicherplatz. Mit `=` bekommt sie einen Wert: `alter = 15` (Zuweisung, gelesen „alter wird 15“).
- **Datentypen:** `int` (ganze Zahl, `15`), `float` (Kommazahl mit Punkt, `1.68`), `str` (Text in Anführungszeichen, `"Hallo"`), `bool` (`True`/`False`).
- `print(...)` gibt aus, mehrere Werte mit Komma getrennt: `print("Alter:", alter)`.
- `input(...)` liest ein. **Das Ergebnis ist immer ein Text (`str`)**, auch wenn eine Zahl eingegeben wird.
- **Typumwandlung:** Zum Rechnen umwandeln.

```
alter = int(input("Alter? "))        # ganze Zahl
groesse = float(input("Größe? "))    # Kommazahl
```

Typische Fehler: `int(...)` vergessen, dann gibt es beim Rechnen einen `TypeError`. Kommazahl mit Komma statt Punkt (`1,68`). Ein Variablenname mit anderer Groß-/Kleinschreibung (`Alter` statt `alter`) führt zum `NameError`.

1.15.2 Operatoren und Bedingungen (Abschnitt 1.2)

Art	Operatoren	Beispiel
Rechnen	`+ - * /`	`7 / 2` ergibt `3.5` (immer `float`)
ganzzahlig teilen, Rest	`//`, `%`	`7 // 2` ergibt `3`, `7 % 2` ergibt `1`
Potenz	`**`	`2 ** 3` ergibt `8`
Vergleich	`== != < > <= >=`	`3 != 4` ergibt `True`
logisch	`and`, `or`, `not`	`True and False` ergibt `False`

- `and`: **beide** Bedingungen müssen wahr sein. `or`: **mindestens eine**. `not`: kehrt um.
- **`=` weist zu, `==` vergleicht.**
- Umrechnen mit `//` und `%`: `200 // 60` ergibt 3 (volle Minuten), `200 % 60` ergibt 20 (Rest-Sekunden). Ob eine Zahl gerade ist: `zahl % 2 == 0`.
- **De Morgan** (eine verknüpfte Bedingung verneinen): Aus `and` wird `or` und umgekehrt, jeder Teil wird verneint.

statt ...	schreibt man ...
`not (A and B)`	`not A or not B`
`not (A or B)`	`not A and not B`
`not (alter >= 12 and groesse >= 140)`	`alter < 12 or groesse < 140`

Das Gegenteil von `>=` ist `<`, das Gegenteil von `<=` ist `>`, das von `==` ist `!=`.

1.15.3 Verzweigungen (Abschnitte 1.3 und 1.4)

```
if bedingung:            # Doppelpunkt!
    anweisungen          # eingerückt: 4 Leerzeichen
elif andere_bedingung:   # beliebig viele, freiwillig
    anweisungen
else:                    # freiwillig, ohne Bedingung
    anweisungen
```

- Es wird **nur der Block der ersten wahren Bedingung** ausgeführt, der Rest wird übersprungen.
- **Einseitig:** nur `if`. **Zweiseitig:** `if … else`. **Mehrseitig:** `if … elif … else`.
- Was eingerückt ist, gehört zum Block. Verzweigungen kann man **verschachteln**.
- Bei Bereichen (unter 1000, 1000 bis unter 2000, ab 2000) von unten nach oben prüfen: `if x < 1000:`, dann `elif x < 2000:`, dann `else:`.
- **Mehrfachauswahl (`CASE OF`)** vergleicht **einen** Wert mit **festen** Werten:

```
match wahl:
    case 1:
        print("Kurzstrecke")
    case 2:
        print("Einzelfahrt")
    case _:              # alle anderen Werte, steht immer zuletzt
        print("Gibt es nicht")
```

Typische Fehler: Doppelpunkt vergessen (`SyntaxError`). = statt == in der Bedingung. Mehrere `if` statt `elif`, dann können zwei Ausgaben kommen. Grenzwert falsch (<= statt <). Bei `match` ohne `int(...)` passt `case 1:` nie, weil die Eingabe ein Text ist.

1.15.4 Schleifen (Abschnitte 1.5 bis 1.7)

Schleife	wann?	Beispiel
`for`	Anzahl der Durchläufe steht **vorher fest**	`for i in range(5):`
`while`	Ende hängt von einer **Bedingung** ab, Anzahl vorher unbekannt	`while zahl != 0:`

- `range(5)` ergibt 0, 1, 2, 3, 4. `range(1, 5)` ergibt 1, 2, 3, 4. `range(0, 10, 3)` ergibt 0, 3, 6, 9. **Der Stoppwert gehört nie dazu.**
- Die **Zählvariable** (`i`) ändert sich bei jedem Durchlauf von selbst.
- In einer `while`-Schleife muss sich die Bedingung **im Rumpf ändern** können, sonst entsteht eine **Endlosschleife**.

Zwei Muster, die du sicher können solltest:

```
summe = 0                        # VOR der Schleife: einmal auf 0
for zahl in range(1, 11):
    summe = summe + zahl         # IN der Schleife: aufaddieren
print("Summe:", summe)           # NACH der Schleife: ausgeben
```

```python
summe = 0
anzahl = 0
wert = float(input("Wert (0 = Ende): "))      # erste Eingabe vor der Schleife
while wert != 0:
    summe = summe + wert
    anzahl = anzahl + 1
    wert = float(input("Wert (0 = Ende): "))  # neue Eingabe am Ende des Rumpfs
print(anzahl, "Werte, Summe:", summe)
```

Typische Fehler: `summe = 0` in die Schleife geschrieben, dann wird die Summe in jedem Durchlauf gelöscht. `range(1, 10)` statt `range(1, 11)`. In der `while`-Schleife die neue Eingabe vergessen (Endlosschleife). `summe + zahl` ohne `summe =` davor ändert nichts.

1.15.5 Funktionen (Abschnitte 1.8 und 1.9)

```python
def flaeche(laenge, breite):     # Definition mit zwei Parametern
    return laenge * breite       # Rückgabewert

ergebnis = flaeche(3, 4)         # Aufruf mit den Argumenten 3 und 4
print(ergebnis)                  # 12
```

- Eine **Funktion** fasst Anweisungen unter einem Namen zusammen und ist wiederverwendbar.
- **Parameter** sind die Platzhalter in der Definition, **Argumente** die Werte beim Aufruf.
- `return` gibt einen **Rückgabewert** zurück und **beendet** die Funktion. Ohne `return` liefert sie `None`.
- Die Definition allein tut nichts. Erst der **Aufruf** führt die Funktion aus.
- Funktionen dürfen andere Funktionen aufrufen.

Typische Fehler: `print` statt `return` in der Funktion, dann kann man mit dem Ergebnis nicht weiterrechnen. Funktion aufgerufen, bevor sie definiert ist. Falsche Anzahl von Argumenten.

1.15.6 Zeichenketten (Abschnitt 1.10)

Was	Beispiel mit `text = "Python"`	Ergebnis
Zeichen am Index	`text[0]`, `text[5]`	`"P"`, `"n"`
Länge	`len(text)`	6
Ausschnitt (Slice)	`text[0:3]`	`"Pyt"` (Index 3 nicht mehr dabei)
Verkettung	`text + "!"`	`"Python!"`
Kleinbuchstaben	`text.lower()`	`"python"`

- Eine Zeichenkette ist eine **Folge von Zeichen**. Der Index beginnt bei **0**, das letzte Zeichen hat den Index `len(text) - 1`.
- Durchlaufen: `for zeichen in text:` geht jedes Zeichen der Reihe nach durch.
- Ein Index, den es nicht gibt (`text[6]`), führt zum `IndexError`.

1.15.7 Listen und Listenalgorithmen (Abschnitte 1.11 und 1.12)

- Eine **Liste** fasst mehrere Elemente zusammen: `noten = [2, 3, 1, 4]`. Der **Index** ist die Position (ab 0), das **Element** der gespeicherte Wert.
- Lesen: `noten[0]` ergibt `2`. Ändern: `noten[1] = 2`. Anhängen: `noten.append(5)`. Länge: `len(noten)`.

- Durchlaufen **über die Elemente** (`for note in noten:`) oder **über die Indizes** (`for i in range(len(noten)):`, wenn man den Index braucht, zum Beispiel zum Ändern).

Die Standardalgorithmen:

```
groesste = zahlen[0]              # Maximum: mit dem ersten Element beginnen
for zahl in zahlen:
    if zahl > groesste:
        groesste = zahl

gefunden = False                  # lineare Suche
for zahl in zahlen:
    if zahl == gesucht:
        gefunden = True
```

Für die **Summe** gilt dasselbe Muster wie bei den Schleifen (vor der Schleife 0, in der Schleife aufaddieren). Für das **Minimum** dreht man den Vergleich um (<).

Typische Fehler: Index und Element verwechseln. Der letzte Index ist `len(liste) - 1`, nicht `len(liste)`. Das Maximum mit 0 statt mit dem ersten Element beginnen (falsch, wenn alle Zahlen negativ sind).

1.15.8 Projekt: Caesar-Verschlüsselung (Abschnitte 1.13 und 1.14)

- Jeder Buchstabe wird um den **Schlüssel** im Alphabet verschoben. Mit Schlüssel 3: A → D, X → A.
- Die Position eines Buchstabens liefert `alphabet.index(buchstabe)`, den Buchstaben an einer Position `alphabet[position]`:

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

- Neue Position: `(position + schluessel) % 26`. Das `% 26` sorgt dafür, dass es nach Z wieder bei A weitergeht.
- **Entschlüsseln** heißt um den Schlüssel zurückschieben: `(position - schluessel) % 26`.
- Ein Text wird Zeichen für Zeichen verschlüsselt, vorher mit `text.upper()` in Großbuchstaben umgewandelt. Eine Hilfsfunktion für einen Buchstaben, eine zweite für den ganzen Text, die die erste in einer Schleife aufruft.

Gymnasium Ottweiler