Python-Konzepte: Zusätzliche Tipps

Dieses Dokument fasst einige Python-Konzepte und -Befehle zusammen, die für die Bearbeitung der Übungsaufgaben hilfreich sein können.

Variablen & Operatoren

In-Place-Operatoren (+=)

Ein Operator wie `+=` ist eine Kurzschreibweise, um eine Variable zu verändern. Er kombiniert eine Rechenoperation mit einer Zuweisung.

- `x += 1` ist die Kurzform für `x = x + 1` .
- `text += "abc"` ist die Kurzform für `text = text + "abc"` .

```
zaehler = 10
zaehler += 5  # zaehler ist jetzt 15

name = "Jane"
name += " Doe" # name ist jetzt "Jane Doe"
```

Typumwandlung (Type Cast)

Manchmal muss ein Datentyp in einen anderen umgewandelt werden, z.B. einen Text (String) in eine Zahl, um damit rechnen zu können.

- `int(wert)` : Wandelt `wert` in eine Ganzzahl um (z.B. `int("10")` wird zu `10`).
- `float(wert)` : Wandelt `wert` in eine Fließkommazahl um (z.B. `float("12.5")` wird zu `12.5`).
- `str(wert)` : Wandelt `wert` in einen String um (z.B. `str(10)` wird zu `"10"`).

```
text_zahl = "150"
echte_zahl = int(text_zahl)
ergebnis = echte_zahl + 50 # ergebnis ist 200
```

Vergleiche und logische Operatoren

Vergleiche werden hauptsächlich in `if` -Bedingungen verwendet und ergeben immer einen Wahrheitswert: `True` (wahr) oder `False` (falsch).

- `==` : Prüft, ob zwei Werte **gleich** sind.
- `!=` : Prüft, ob zwei Werte **ungleich** sind.
- `>` : Größer als.
- `<` : Kleiner als.
- `>=` : Größer als oder gleich.
- `<=` : Kleiner als oder gleich.
- `in` : Prüft, ob ein Element in einer Sequenz (z.B. einer Liste oder einem String) enthalten ist.
- `not` : Kehrt einen Wahrheitswert um (`not True` wird zu `False`).

```
alter = 30
if alter >= 18:
    print("Volljährig") # Wird ausgegeben

name = "Python"
if "th" in name:
    print("'th' ist im Namen enthalten") # Wird ausgegeben

if "Java" not in name:
    print("'Java' ist nicht im Namen enthalten") # Wird ausgegeben
```

Nützliche eingebaute Funktionen

Diese Funktionen sind in Python immer verfügbar.

- `len(objekt)` : Gibt die Länge (Anzahl der Elemente) eines Objekts zurück. Funktioniert bei Listen, Strings, Dictionaries etc.
- `sum(liste)` : Berechnet die Summe aller Zahlen in einer Liste.

```
meine_liste = [10, 20, 30]
print(len(meine_liste)) # Ausgabe: 3
print(sum(meine_liste))  # Ausgabe: 60
```

Methoden für Strings (Texte)

Methoden werden mit einem Punkt an eine Variable angehängt (z.B. `variable.methode()`).

- `.lower()` : Wandelt alle Buchstaben in Kleinbuchstaben um.
- `.upper()` : Wandelt alle Buchstaben in Großbuchstaben um.
- `.capitalize()` : Macht nur den ersten Buchstaben des Strings groß.
- `.replace(alt, neu)` : Ersetzt alle Vorkommen des `alt` -Strings durch den `neu` -String.
- `.split()` : Teilt einen String an den Leerzeichen auf und gibt eine Liste von Wörtern zurück.

```
text = "Hallo Welt, hallo Python!"
print(text.lower())    # "hallo welt, hallo python!"
print(text.upper())    # "HALLO WELT, HALLO PYTHON!"
print(text.replace("hallo", "servus")) # "Hallo Welt, servus Python!"
print(text.split())    # ['Hallo', 'Welt,', 'hallo', 'Python!']
```

Methoden für Listen

- `.append(element)` : Fügt ein Element am Ende einer Liste hinzu.

```
zahlen = [1, 2, 3]
zahlen.append(4)
print(zahlen) # Ausgabe: [1, 2, 3, 4]
```

Schleifen-Helfer

`enumerate()`

Wenn in einer Schleife sowohl der Index (die Position) als auch das Element selbst benötigt wird, kann `enumerate()` verwendet werden.

```
laender = ["Deutschland", "Österreich", "Schweiz"]
for index, land in enumerate(laender):
    print(f"Position {index}: {land}")
# Ausgabe:
# Position 0: Deutschland
# Position 1: Österreich
# Position 2: Schweiz
```

`.items()` und das Komma in `for` -Schleifen

Um durch ein Dictionary zu iterieren und dabei auf Schlüssel und Wert gleichzeitig zuzugreifen, verwendet man die `.items()` -Methode. Das Komma in der `for` -Schleife "entpackt" jedes Schlüssel-Wert-Paar in zwei separate Variablen.

```
hauptstaedte = {"Deutschland": "Berlin", "Frankreich": "Paris"}
for land, stadt in hauptstaedte.items():
    print(f"Die Hauptstadt von {land} ist {stadt}.")
```

Datei-Operationen

Allgemein

- `.read()` : Liest den **gesamten** Inhalt einer geöffneten Datei als einen einzigen String.
- `.write(text)` : Schreibt den übergebenen `text` in eine geöffnete Datei.

Speziell für CSV-Dateien

Diese Befehle gehören zum `csv` -Modul.

- `csv.DictWriter` : Ein Helfer, um eine Liste von Dictionaries zeilenweise in eine CSV-Datei zu schreiben.
- `.writeheader()` : Schreibt die erste Zeile (die Kopfzeile mit den Spaltennamen) in die CSV-Datei.
- `.writerows(liste_von_dicts)` : Schreibt alle Dictionaries aus der übergebenen Liste als Zeilen in die CSV-Datei.

```
import csv

daten = [
    {'Name': 'Alice', 'Alter': 30},
    {'Name': 'Bob', 'Alter': 25}
]

with open('output.csv', 'w', newline='', encoding='utf-8') as f:
    # Die Feldnamen müssen den Schlüsseln in den Dictionaries entsprechen
    feldnamen = ['Name', 'Alter']
    writer = csv.DictWriter(f, fieldnames=feldnamen)

    writer.writeheader() # Schreibt "Name,Alter"
    writer.writerows(daten) # Schreibt die zwei Zeilen für Alice und Bob
```

Erstellen von Dictionaries mit `{}`

Die geschweiften Klammern `{}` werden in Python hauptsächlich verwendet, um **Dictionaries** (Wörterbücher) zu erstellen. Ein Dictionary ist eine Sammlung von **Schlüssel-Wert-Paaren** (`key-value pairs`). Jeder Wert ist über einen eindeutigen Schlüssel erreichbar.

Ein Dictionary mit Inhalt erstellen

Ein Dictionary kann direkt mit seinen Schlüssel-Wert-Paaren initialisiert werden. Der Schlüssel und der Wert werden durch einen Doppelpunkt `:` getrennt.

```
# Syntax: {schlüssel1: wert1, schlüssel2: wert2}
person = {"name": "Max", "alter": 30, "stadt": "Berlin"}

# Zugriff auf einen Wert über seinen Schlüssel
print(person["name"]) # Ausgabe: Max
```

Ein leeres Dictionary erstellen

Sehr oft startet man mit einem leeren Dictionary, um es später – zum Beispiel in einer Schleife – schrittweise mit Daten zu füllen.

```
# Ein leeres Dictionary erstellen
wort_zaehler = {}

# Später Schlüssel und Werte hinzufügen
wort_zaehler["python"] = 3
wort_zaehler["ist"] = 2

print(wort_zaehler) # Ausgabe: {'python': 3, 'ist': 2}
```

Hinweis: `{}` kann auch **Sets** erstellen, eine Sammlung von einzigartigen Werten ohne Schlüssel (`meine_zahlen = {1, 2, 3}`). Für die Aufgaben in diesem Workshop ist die Verwendung als Dictionary jedoch relevanter. Ein leeres `{}` erzeugt **immer** ein leeres Dictionary, kein leeres Set.