DS 9 – Übungsstunde: Funktionen lesen, testen und erweitern

Informatik 10 – MINT-Zweig | Textuelle Programmierung und Datenstrukturen

Lernziele

Ich kann

- gegebene Funktionen lesen und ihr Verhalten vorhersagen und erklären,
- einen Fehler in einer Funktion durch geschicktes Testen finden und beheben,
- Funktionen schreiben, die andere Funktionen aufrufen,
- eine vorhandene Funktion zu einer neuen Funktion erweitern,
- mit `math.pi` die Kreiszahl π verwenden.

🗣️ Sprachhilfe

- „Der Aufruf `verarbeite(6, 3)` ruft die Hilfsfunktion `ist_teilbar(6, 3)` auf. Sie liefert `True`, deshalb gibt `verarbeite` den Wert … zurück."
- „Ich teste mit dem Wert …, weil das ein Sonderfall / der Grenzwert der Bedingung ist."
- „Erwartet habe ich …, herausgekommen ist … . Also steckt der Fehler in Zeile …, weil …"

1 Das brauchst du: Funktionen wiederholen

Führe die Zelle aus. Erkläre dann mit den Fachbegriffen **Definition**, **Parameter**, **Argument**, **Aufruf** und **Rückgabewert**, was hier passiert.

```
def quadrat(zahl):
    return zahl * zahl

print(quadrat(3))
print(quadrat(10))
```

```
9
100
```

- `def quadrat(zahl):` **definiert** die Funktion `quadrat` mit dem **Parameter** `zahl`. Der eingerückte Rumpf wird erst beim Aufruf ausgeführt.
- `return zahl * zahl` gibt das Ergebnis als **Rückgabewert** zurück und beendet die Funktion.
- `quadrat(3)` und `quadrat(10)` sind zwei **Aufrufe** derselben Funktion mit verschiedenen **Argumenten**.

Funktionen rufen Funktionen auf

Im Rumpf einer Funktion darf man auch **andere Funktionen** aufrufen. So kann eine größere Funktion kleinere **Hilfsfunktionen** benutzen.

🔮 **Vorhersage:** Was gibt die letzte Zeile aus? Überlege zuerst, dann führe die Zelle aus.

```python
def quadrat(zahl):
    return zahl * zahl

def summe_der_quadrate(a, b):
    return quadrat(a) + quadrat(b)

print(summe_der_quadrate(3, 4))
```

```
25
```

- **Zeilen 1–2:** die bekannte Funktion `quadrat`.
- **Zeilen 4–5:** `summe_der_quadrate` rechnet selbst nichts mit `*`. Sie ruft zweimal `quadrat` auf und addiert die beiden Rückgabewerte.
- **Zeile 7:** Der Aufruf springt in `summe_der_quadrate` mit `a = 3` und `b = 4`. Dort springt `quadrat(a)` in die Funktion `quadrat` und kommt mit `9` zurück, `quadrat(b)` liefert `16`. Zum Schluss wird `9 + 16 = 25` zurückgegeben.

Der Vorteil: Jede Funktion erledigt eine kleine, klar umrissene Aufgabe. Ist `quadrat` einmal getestet, kann man sich darauf verlassen, wo immer man sie benutzt.

> 📌 **Merksatz:** Funktionen können **andere Funktionen aufrufen**. Eine größere Funktion nutzt dann kleinere **Hilfsfunktionen**.

2 Code lesen und erklären

Bevor du ein fremdes Programm ausführst, lohnt es sich, es zu **lesen** und vorherzusagen, was passiert.

> 💡 **Tipp:** `a % b` ist der **Rest** der Division (`7 % 3` ergibt `1`), `a // b` die **ganzzahlige Division** (`7 // 3` ergibt `2`). Ist der Rest `0`, geht die Division auf. Lies verschachtelte Aufrufe von innen nach außen und schreibe dir für jeden Aufruf die Werte der Parameter auf.

🔍 Übung 1 – Code lesen

Lies den Code in der nächsten Zelle, **ohne** ihn auszuführen.

1. Beschreibe in eigenen Worten, was die Funktion `verarbeite` macht.
2. Gib an, was `print(verarbeite(6, 3))` und `print(verarbeite(5, 3))` ausgeben.

Schreibe deine Antwort in die Textzelle darunter. Führe den Code erst danach aus und überprüfe deine Vorhersage.

```python
def ist_teilbar(a, b):
    return a % b == 0

def verarbeite(x, y):
    if ist_teilbar(x, y):
        return x // y
    else:
        return x * y

print(verarbeite(6, 3))
print(verarbeite(5, 3))
```

```
2
15
```

Meine Erklärung und Vorhersage:

...

3 Fehler finden durch Testen

Manche Fehler erzeugen keine Fehlermeldung: Das Programm läuft, aber das Ergebnis ist falsch. Solche Fehler findest du nur durch **Testen** mit Werten, deren richtiges Ergebnis du kennst.

🧰 Werkzeug: das Modul `math`

Python bringt viele mathematische Werte und Funktionen im **Modul** `math` mit. Du schreibst einmal am Programmanfang `import math`, so wie `import random` beim Würfeln. Danach steht die Kreiszahl als `math.pi` zur Verfügung.

```
import math
print(math.pi)
```

```
3.141592653589793
```

📝 Übung 2 – Kreisumfang

Die Funktion soll den Umfang eines Kreises berechnen (Formel: U = 2 · π · r). Sie gibt aber ein falsches Ergebnis zurück. Finde den Fehler und behebe ihn. Teste die verbesserte Funktion, auch mit dem Sonderfall `radius = 0`.

```
import math

def kreisumfang(radius):
    umfang = 2 * radius
    return umfang

# Test: sollte bei radius = 5 etwa 31.4 ergeben
print(kreisumfang(5))
```

```
10
```

Der Fehler:

...

📝 Übung 3 – Grenzfälle testen

Ein Online-Shop gibt **ab** einem Einkaufswert von 50 Euro 10 % Rabatt. Die Funktion `rabatt_preis` soll den Preis nach Abzug des Rabatts zurückgeben.

1. Rechne **zuerst von Hand** aus, was für 20, 50 und 100 Euro herauskommen muss. Trage die Werte hinter `erwartet:` ein.
2. Führe die Tests aus. Welcher Test schlägt fehl?
3. Erkläre den Fehler und verbessere die Funktion.

```python
def rabatt_preis(preis):
    if preis > 50:
        return preis * 0.9
    else:
        return preis

print(rabatt_preis(20), "erwartet: ")
print(rabatt_preis(50), "erwartet: ")
print(rabatt_preis(100), "erwartet: ")
```

```
20 erwartet:
50 erwartet:
90.0 erwartet:
```

Welcher Test schlägt fehl, und warum?

…

> 📌 **Merksatz:** Falsche Ergebnisse ohne Fehlermeldung findet man durch **Testen** mit Werten, deren Ergebnis man **vorher** kennt. Besonders wichtig sind **Sonder- und Grenzfälle** wie `0` oder der Grenzwert einer Bedingung.

4 Ein Programm aus mehreren Funktionen

Die Übungen dieser Stunde werden schrittweise selbstständiger: In Übung 1 liest du nur, in Übung 2 und 3 verbesserst du fertigen Code, in Übung 4 füllst du ein Gerüst aus, und die Zusatzaufgabe schreibst du allein. Vorbild für Übung 4 ist das vorgemachte Beispiel `summe_der_quadrate` aus Teil 1.

📝 Übung 4 – Rechteck (mit Gerüst)

Schreibe drei Funktionen:

- `flaeche_rechteck(a, b)` gibt die Fläche eines Rechtecks mit den Seitenlängen `a` und `b` zurück,
- `umfang_rechteck(a, b)` gibt den Umfang dieses Rechtecks zurück,
- `beschreibe_rechteck(a, b)` nutzt die beiden Funktionen oben und liefert einen Satz, den das Hauptprogramm ausgibt, z. B. für die Seiten 4 und 5:

  „Ein Rechteck mit den Seiten 4 und 5 hat die Fläche 20 und den Umfang 18.“

> 💡 **Tipp:** Schreibe zuerst die beiden kleinen Funktionen und teste jede einzeln (Kontrolle: Fläche 20, Umfang 18). Erst wenn sie stimmen, schreibst du `beschreibe_rechteck`. Sie ruft die beiden anderen auf, so wie `summe_der_quadrate` oben `quadrat` aufruft. Den Satz setzt du mit `+` zusammen. Zahlen musst du dafür mit `str(...)` in Text umwandeln.

Gerüst: Ersetze jedes `___`. Führe die Zelle zuerst nur mit Schritt 1 aus und prüfe die Kontrollwerte, bevor du Schritt 2 bearbeitest.

Lückencode

```python
# Schritt 1: die beiden Hilfsfunktionen
def flaeche_rechteck(a, b):
    return ___

def umfang_rechteck(a, b):
    return ___

print(flaeche_rechteck(4, 5), umfang_rechteck(4, 5), "erwartet: 20 18")

# Schritt 2: beschreibe_rechteck ruft die beiden Hilfsfunktionen auf
def beschreibe_rechteck(a, b):
    flaeche = ___
    umfang = ___
    return ("Ein Rechteck mit den Seiten " + str(a) + " und " + str(b)
            + " hat die Fläche " + ___ + " und den Umfang " + ___ + ".")

print(beschreibe_rechteck(4, 5))
```

🚀 Zusatzaufgabe – Maximum von drei Zahlen (selbstständig)

Jetzt ohne Gerüst. In DS 8 hast du `maximum(a, b)` geschrieben. Ergänze eine Funktion `maximum3(a, b, c)`, die die größte von **drei** Zahlen zurückgibt. Benutze dafür `maximum`: `maximum3` selbst soll ohne `if` auskommen. Teste mit mindestens drei Aufrufen, bei denen die größte Zahl jeweils an einer anderen Stelle steht.

```python
def maximum(a, b):
    if a > b:
        return a
    else:
        return b

# TODO: maximum3(a, b, c)
```

📌 Das Wichtigste in Kürze

- Programme **lesen**: von innen nach außen auswerten und für jeden Aufruf die Werte der Parameter notieren.
- Funktionen können **andere Funktionen aufrufen**. Große Aufgaben zerlegt man so in kleine **Hilfsfunktionen**, die man einzeln testet.
- Falsche Ergebnisse ohne Fehlermeldung findet man durch **Testen** mit Werten, deren Ergebnis man vorher kennt.
- Besonders wichtig sind Tests mit **Sonder- und Grenzfällen** wie `0` oder dem Grenzwert einer Bedingung.
- Mit `import math` steht die Kreiszahl als `math.pi` zur Verfügung.

Zum Weiterüben: `DS09_Weitere_Uebungen.ipynb`.