1.9 Übung: Funktionen lesen, testen und erweitern

Leitidee

Im Abschnitt über Funktionen hast du gelernt, eigene Funktionen zu schreiben. Jetzt übst du, was Programmiererinnen und Programmierer jeden Tag tun: fremden Code **lesen** und verstehen, **Fehler finden**, Funktionen **testen** und aus kleinen Funktionen ein größeres Programm **zusammensetzen**. Dabei lernst du eine wichtige neue Idee kennen: Eine Funktion darf andere Funktionen aufrufen. So entstehen Programme aus Bausteinen, die man einzeln prüfen kann.

Lernziele

Nach diesem Abschnitt kannst du

- 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.

1.9.1 Das brauchst du

Wiederhole kurz das Wichtigste über Funktionen:

```
1 def quadrat(zahl):
2     return zahl * zahl
3
4 print(quadrat(3))
5 print(quadrat(10))
```

AUSGABE

```
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

Neu ist nur eine Kleinigkeit: Im Rumpf einer Funktion darf man auch **andere Funktionen** aufrufen. So kann eine größere Funktion kleinere **Hilfsfunktionen** benutzen:

```
1 def quadrat(zahl):
2     return zahl * zahl
3
4 def summe_der_quadrate(a, b):
5     return quadrat(a) + quadrat(b)
6
7 print(summe_der_quadrate(3, 4))
```

AUSGABE

```
25
```

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

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

Merke

- `def name(parameter):` definiert eine Funktion, `return wert` gibt einen Wert zurück.
- Eine Funktion kann **mehrfach** mit unterschiedlichen Argumenten aufgerufen werden.
- Funktionen können **andere Funktionen aufrufen** (z. B. eine Hauptfunktion nutzt Hilfsfunktionen).

1.9.2 Code lesen und erklären

Bevor du ein fremdes Programm ausführst, lohnt es sich, es zu **lesen** und vorherzusagen, was passiert. So trainierst du, Programme wirklich zu verstehen – und nicht nur auszuprobieren.

Tipp

Erinnere dich an die Rechenzeichen aus dem Abschnitt über Operatoren: `a % b` ist der **Rest** der Division (`7 % 3` ergibt `1`), `a // b` ist 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.

Arbeitsauftrag 105: Code lesen

Lies den folgenden Code, **ohne** ihn auszuführen.

```
1  def ist_teilbar(a, b):
2      return a % b == 0
3
4  def verarbeite(x, y):
5      if ist_teilbar(x, y):
6          return x // y
7      else:
8          return x * y
9
10 print(verarbeite(6, 3))
11 print(verarbeite(5, 3))
```

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

Führe den Code danach aus und überprüfe deine Vermutung.

Meine Erklärung und Vorhersage:

1.9.3 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. Für die nächste Aufgabe brauchst du die Kreiszahl π.

Werkzeug

Python bringt viele mathematische Werte und Funktionen in einem **Modul** namens `math` mit. Ein Modul ist eine Sammlung fertiger Bausteine. Damit du es benutzen kannst, schreibst du 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)
```

AUSGABE

```
3.141592653589793
```

Arbeitsauftrag 106: Kreisumfang

Die folgende Funktion soll den Umfang eines Kreises berechnen (Formel: $U = 2 \cdot \pi \cdot r$). Sie gibt aber ein falsches Ergebnis zurück. Finde den Fehler und behebe ihn. Teste die verbesserte Funktion.

```
1 import math
2
3 def kreisumfang(radius):
4     umfang = 2 * radius
5     return umfang
6
7 # Test: sollte bei radius = 5 etwa 31.4 ergeben
8 print(kreisumfang(5))
```

AUSGABE

```
10
```

Der Fehler:

Arbeitsauftrag 107: Grenzfälle testen

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

```
1 def rabatt_preis(preis):
2     if preis > 50:
3         return preis * 0.9
4     else:
5         return preis
```

1. Rechne von Hand aus, was für die Einkaufswerte 20, 50 und 100 Euro herauskommen **muss**.
2. Teste die Funktion mit diesen drei Werten. Welcher Test schlägt fehl?
3. Erkläre den Fehler und verbessere die Funktion.

Gymnasium Ottweiler

Erwartete Werte, Testergebnis und Verbesserung:

1.9.4 Ein Programm aus mehreren Funktionen

Tipp

Bei der nächsten Aufgabe schreibst du zuerst die beiden kleinen Funktionen und testest jede einzeln. Erst wenn sie stimmen, schreibst du `beschreibe_rechteck`. Diese Funktion ruft die beiden anderen auf, so wie `summe_der_quadrate` oben `quadrat` aufruft. Den Satz setzt du mit `+` zusammen. Denk daran: Zahlen müssen dafür mit `str(...)` in Text umgewandelt werden.

Arbeitsauftrag 108: Rechteck

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 zusammenfassenden 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.“

Arbeitsauftrag 109 (Zusatz): Maximum von drei Zahlen

Im Abschnitt über Funktionen hast du `maximum(a, b)` geschrieben:

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

Erweitere das Programm um eine Funktion `maximum3(a, b, c)`, die die größte von **drei** Zahlen zurückgibt. Benutze dafür die Funktion `maximum` – `maximum3` selbst soll ohne `if` auskommen. Teste mit mindestens drei Aufrufen, bei denen die größte Zahl jeweils an einer anderen Stelle steht.

1.9.5 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 110: Verschachtelte Aufrufe ●○○

Lies das Programm, **ohne** es auszuführen. Gib an, was jede `print`-Zeile ausgibt. Überprüfe danach am Computer.

```python
def plus_eins(x):
    return x + 1

def doppelt(x):
    return 2 * x

print(doppelt(plus_eins(3)))
print(plus_eins(doppelt(3)))
print(doppelt(doppelt(plus_eins(0))))
```

Meine Vorhersage:

Arbeitsauftrag 111: Kreisfläche ●○○

Die Fläche eines Kreises berechnet man mit der Formel $A = \pi \cdot r \cdot r$.

1. Implementiere eine Funktion `kreisflaeche(radius)`, die die Fläche zurückgibt. Verwende `math.pi`.
2. Teste die Funktion mit den Radien 1, 0 und 2. Zur Kontrolle: Für den Radius 1 muss genau π herauskommen, also etwa 3,14.

Arbeitsauftrag 112: Gute Testwerte wählen ●○○

Die Funktion `ist_teenager(alter)` soll `True` zurückgeben, wenn jemand 13 bis einschließlich 19 Jahre alt ist, sonst `False`.

```
def ist_teenager(alter):
    return alter >= 13 and alter <= 19
```

Nenne vier Testwerte, mit denen du die Funktion besonders gut prüfen kannst, und gib jeweils das erwartete Ergebnis an. Begründe deine Auswahl in einem Satz.

Testwerte und Begründung:

Arbeitsauftrag 113: Ungerade mit Hilfsfunktion ●○○

Im Abschnitt über Funktionen hast du `ist_gerade(zahl)` geschrieben:

```
def ist_gerade(zahl):
    return zahl % 2 == 0
```

Implementiere eine Funktion `ist_ungerade(zahl)`, die `True` zurückgibt, wenn die Zahl ungerade ist. Benutze dafür die Funktion `ist_gerade` und den Operator `not`. Teste mit 7 und mit 10.

Arbeitsauftrag 114: Wo steckt der Fehler? ●●○

Eine Fahrt dauert 2 Stunden und 15 Minuten. Das Programm soll die Fahrzeit in Minuten berechnen, also 135. Es gibt aber 215 aus.

```
1 def stunden_zu_minuten(stunden):
2     return stunden * 100
3
4 def fahrzeit_minuten(stunden, minuten):
5     return stunden_zu_minuten(stunden) + minuten
6
7 print(fahrzeit_minuten(2, 15))
```

1. Beschreibe, wie du durch einen **einzelnen Test der Hilfsfunktion** herausfindest, in welcher der beiden Funktionen der Fehler steckt.

2. Behebe den Fehler und teste erneut.

Mein Vorgehen und der Fehler:

Arbeitsauftrag 115: Versandkosten ●●○

Ein Online-Shop berechnet 5 Euro Versandkosten. Ab einem Warenwert von 30 Euro ist der Versand kostenlos.

1. Implementiere eine Funktion `versandkosten(warenwert)`, die die Versandkosten zurückgibt (0 oder 5).
2. Implementiere eine Funktion `endpreis(warenwert)`, die den Warenwert plus Versandkosten zurückgibt. Sie soll `versandkosten` aufrufen.
3. Teste `endpreis` mit den Warenwerten 20, 30 und 45. Überlege vorher, was herauskommen muss.

Arbeitsauftrag 116: Pizza-Vergleich ●●○

Was ist mehr Pizza: **eine** große Pizza mit 32 cm Durchmesser oder **zwei** kleine mit je 22 cm Durchmesser?

1. Implementiere eine Funktion `pizza_flaeche(durchmesser)`, die die Fläche einer runden Pizza zurückgibt. Sie soll die Funktion `kreisflaeche(radius)` aus dem Auftrag *Kreisfläche* aufrufen. Denk daran: Der Radius ist der halbe Durchmesser.
2. Berechne damit die Fläche der großen Pizza und die Fläche der beiden kleinen Pizzen zusammen. Gib beide Werte gerundet aus und beantworte die Frage.

Arbeitsauftrag 117: Primzahlen mit Hilfsfunktionen ●●●

Im Auftrag *Primzahl-Test* (Übung zu for- und while-Schleife) hast du geprüft, ob eine Zahl eine Primzahl ist. Jetzt zerlegst du die Aufgabe in Funktionen.

1. Implementiere eine Funktion `ist_teiler(t, zahl)`, die `True` zurückgibt, wenn `t` ein Teiler von `zahl` ist (die Division geht ohne Rest auf).

2. Implementiere eine Funktion `ist_primzahl(zahl)`, die `True` oder `False` zurückgibt. Sie soll `ist_teiler` benutzen.
3. Gib mit einer Schleife alle Primzahlen von 1 bis 30 aus.

Tipp

Eine Primzahl ist eine Zahl größer als 1, die nur durch 1 und sich selbst teilbar ist. Gehe in `ist_primzahl` wie im Auftrag *Primzahl-Test* vor: Starte mit `ergebnis = zahl > 1` und prüfe in einer Schleife alle möglichen Teiler von 2 bis `zahl - 1`. Findest du einen Teiler, setzt du `ergebnis = False`.

1.9.6 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.

Selbstcheck

- ☐ Ich kann ein Programm mit Funktionen lesen und seine Ausgabe vorhersagen.
- ☐ Ich kann erklären, was eine Funktion tut, die eine andere Funktion aufruft.
- ☐ Ich kann einen Fehler in einer Funktion durch gezieltes Testen finden und beheben.
- ☐ Ich kann Testwerte so wählen, dass auch Grenzfälle geprüft werden.
- ☐ Ich kann ein Programm aus mehreren Funktionen zusammensetzen.
- ☐ Ich kann `math.pi` in einer Rechnung verwenden.

Gymnasium Ottweiler