DS 9 – Weitere Übungen

Übung: Funktionen lesen, testen und erweitern

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.

Die Aufgaben stehen auch im Skript im Abschnitt „Weitere Übungen“; die Lösungen im Lösungsheft.

Aufgabe 1: 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))))
```

```
8
7
4
```

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

```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))))
```

```
8
7
4
```

Lösung

Ausgabe:

```
8
7
4
```

Lösung

Verschachtelte Aufrufe liest du von innen nach außen:

- `plus_eins(3)` liefert `4`, und `doppelt(4)` liefert `8`.

- `doppelt(3)` liefert `6`, und `plus_eins(6)` liefert `7`. Die Reihenfolge der Funktionen macht also einen Unterschied.
- `plus_eins(0)` liefert `1`, `doppelt(1)` liefert `2`, und `doppelt(2)` liefert `4`.

Aufgabe 2: 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.

Lösung

✅ **Lösung**

Lösung

```python
import math

def kreisflaeche(radius):
    return math.pi * radius * radius

print(kreisflaeche(1))
print(kreisflaeche(0))
print(round(kreisflaeche(2), 2))
```

```
3.141592653589793
0.0
12.57
```

Lösung

Ausgabe:

```
3.141592653589793
0.0
12.57
```

Lösung

Wie beim Kreisumfang muss `import math` am Anfang stehen, sonst kennt Python `math.pi` nicht (`NameError: name 'math' is not defined`). Für den Radius 2 ist die Fläche $\pi \cdot 4 \approx 12,57$. Statt `radius * radius` darfst du auch `radius ** 2` schreiben.

Aufgabe 3: Gute Testwerte wählen ●○○

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

```python
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.

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

Gute Testwerte liegen direkt an den beiden Grenzen, jeweils einer innerhalb und einer außerhalb:

Lösung

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

print(ist_teenager(12), "erwartet: False")
print(ist_teenager(13), "erwartet: True")
print(ist_teenager(19), "erwartet: True")
print(ist_teenager(20), "erwartet: False")
```

```
False erwartet: False
True erwartet: True
True erwartet: True
False erwartet: False
```

Lösung

Ausgabe:

```
False erwartet: False
True erwartet: True
True erwartet: True
False erwartet: False
```

Lösung

Begründung: An den Grenzen passieren die häufigsten Fehler, zum Beispiel `>` statt `>=`. Ein Wert wie 15 oder 40 würde einen solchen Fehler nicht zeigen. Auch andere Werte sind als Ergänzung richtig, solange 13 und 19 dabei sind.

Aufgabe 4: 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.

Lösung

✅ **Lösung**

Lösung

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

def ist_ungerade(zahl):
    return not ist_gerade(zahl)

print(ist_ungerade(7), "erwartet: True")
print(ist_ungerade(10), "erwartet: False")
```

```
True erwartet: True
False erwartet: False
```

Lösung

Ausgabe:

```
True erwartet: True
False erwartet: False
```

Lösung

`ist_ungerade` ruft die Hilfsfunktion `ist_gerade` auf und kehrt ihr Ergebnis mit `not` um. Eine Zahl ist genau dann ungerade, wenn sie nicht gerade ist. Auch richtig, aber ohne Hilfsfunktion: `return zahl % 2 == 1`.

Aufgabe 5: 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.

```python
def stunden_zu_minuten(stunden):
    return stunden * 100

def fahrzeit_minuten(stunden, minuten):
    return stunden_zu_minuten(stunden) + minuten

print(fahrzeit_minuten(2, 15))
```

```
215
```

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.

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

1. Man testet `stunden_zu_minuten` allein, zum Beispiel mit `print(stunden_zu_minuten(1))`. Erwartet sind 60 Minuten, die Funktion liefert aber 100. Der Fehler steckt also in der Hilfsfunktion. `fahrzeit_minuten` selbst rechnet richtig: Sie addiert die Minuten zum Rückgabewert der Hilfsfunktion.
2. Eine Stunde hat 60 Minuten, nicht 100:

Lösung

```python
def stunden_zu_minuten(stunden):
    return stunden * 60

def fahrzeit_minuten(stunden, minuten):
    return stunden_zu_minuten(stunden) + minuten

print(stunden_zu_minuten(1), "erwartet: 60")
print(fahrzeit_minuten(2, 15), "erwartet: 135")
```

```
60 erwartet: 60
135 erwartet: 135
```

Lösung

Ausgabe:

```
60 erwartet: 60
135 erwartet: 135
```

Lösung

Das ist der Vorteil von Hilfsfunktionen: Man kann jede einzeln testen und so den Fehler eingrenzen.

Aufgabe 6: 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.

Lösung

✅ **Lösung**

Lösung

```python
def versandkosten(warenwert):
    if warenwert >= 30:
        return 0
    else:
        return 5

def endpreis(warenwert):
    return warenwert + versandkosten(warenwert)

print(endpreis(20), "erwartet: 25")
print(endpreis(30), "erwartet: 30")
print(endpreis(45), "erwartet: 45")
```

```
25 erwartet: 25
30 erwartet: 30
45 erwartet: 45
```

Lösung

Ausgabe:

```
25 erwartet: 25
30 erwartet: 30
45 erwartet: 45
```

Lösung

`endpreis` rechnet selbst nur eine Addition. Ob Versandkosten anfallen, entscheidet die Hilfsfunktion. „Ab 30 Euro“ schließt 30 mit ein, deshalb steht dort `>=`. Der Test mit genau 30 prüft diese Grenze.

Aufgabe 7: 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.

Lösung

✅ **Lösung**

Lösung

```python
import math

def kreisflaeche(radius):
    return math.pi * radius * radius

def pizza_flaeche(durchmesser):
    return kreisflaeche(durchmesser / 2)

gross = pizza_flaeche(32)
zwei_kleine = 2 * pizza_flaeche(22)
print("Eine große Pizza:", round(gross), "Quadratzentimeter")
print("Zwei kleine Pizzen:", round(zwei_kleine), "Quadratzentimeter")
```

```
Eine große Pizza: 804 Quadratzentimeter
Zwei kleine Pizzen: 760 Quadratzentimeter
```

Lösung

Ausgabe:

```
Eine große Pizza: 804 Quadratzentimeter
Zwei kleine Pizzen: 760 Quadratzentimeter
```

Lösung

Die eine große Pizza ist etwas mehr Pizza, obwohl $2 \cdot 22 = 44$ cm nach mehr aussieht. Die Fläche wächst nämlich mit dem **Quadrat** des Radius. `pizza_flaeche` rechnet nur den Radius aus und überlässt den Rest der Hilfsfunktion `kreisflaeche`. Typischer Fehler: `kreisflaeche(durchmesser)` ohne `/ 2`. Dann wird jede Fläche viermal zu groß.

Aufgabe 8: 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`.

Lösung

✅ Lösung

Lösung

```python
def ist_teiler(t, zahl):
    return zahl % t == 0

def ist_primzahl(zahl):
    ergebnis = zahl > 1
    for t in range(2, zahl):
        if ist_teiler(t, zahl):
            ergebnis = False
    return ergebnis

print(ist_teiler(3, 12), ist_teiler(5, 12))
print(ist_primzahl(7), ist_primzahl(9), ist_primzahl(1))

for zahl in range(1, 31):
    if ist_primzahl(zahl):
        print(zahl)
```

```
True False
True False False
2
3
5
7
11
13
17
19
23
29
```

Lösung

Ausgabe:

```
True False
True False False
2
3
5
7
11
13
17
19
23
29
```

Lösung

- **Zeilen 1–2:** `zahl % t == 0` ist `True`, wenn beim Teilen kein Rest bleibt.
- **Zeilen 4–9:** `ergebnis` ist zuerst `True` für alle Zahlen größer als 1 und `False` für 1 (und kleinere Zahlen). Wird ein Teiler zwischen 2 und `zahl - 1` gefunden, wird `ergebnis` auf `False` gesetzt. Das `return` steht nach der Schleife.
- **Zeilen 11–12:** Erst die Hilfsfunktionen einzeln testen, dann zusammensetzen.
- **Zeilen 14–16:** Das Hauptprogramm ruft `ist_primzahl` für jede Zahl von 1 bis 30 auf.

Ein häufiger Fehler ist `return True` im `else`-Teil innerhalb der Schleife. Dann endet die Funktion schon beim ersten Nicht-Teiler, und zum Beispiel 9 gilt als Primzahl (9 ist nicht durch 2 teilbar).