DS 8 – Weitere Übungen

Funktionen: Programme aus Bausteinen

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: Fachbegriffe am Beispiel ●○○

Am Kiosk kostet jede Pfandflasche 25 Cent Pfand zusätzlich zum Preis. Lies das Programm:

```
1 def preis_mit_pfand(preis, flaschen):
2     return preis + flaschen * 0.25
3
4 gesamt = preis_mit_pfand(3.5, 6)
5 print(gesamt)
```

```
5.0
```

Nenne zu diesem Programm:

1. den Namen der Funktion,
2. die Parameter,
3. die Argumente beim Aufruf,
4. die Zeile, in der die Funktion aufgerufen wird,
5. den Rückgabewert für diesen Aufruf.

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

1. Die Funktion heißt `preis_mit_pfand`.
2. Die Parameter sind `preis` und `flaschen` (sie stehen in der Definition in Zeile 1).
3. Die Argumente sind `3.5` und `6` (sie stehen beim Aufruf in Zeile 4).
4. Der Aufruf steht in Zeile 4. Zeile 1 ist die Definition, kein Aufruf.
5. Der Rückgabewert ist $3,5 + 6 \cdot 0,25 = 5,0$:

Lösung

```
1 def preis_mit_pfand(preis, flaschen):
2     return preis + flaschen * 0.25
3
4 gesamt = preis_mit_pfand(3.5, 6)
5 print(gesamt)
```

```
5.0
```

Lösung

Ausgabe:

```
5.0
```

Lösung

Typischer Fehler: Parameter und Argumente verwechseln. Merkhilfe: **P**arameter stehen bei der **P**lanung (Definition), Argumente beim Aufruf.

Aufgabe 2: Rückgabewerte vorhersagen ●○○

Gegeben ist die Funktion `rechne`:

```
def rechne(x):
    return 3 * x - 1
```

Bestimme **ohne Computer**, welchen Wert die Aufrufe zurückgeben. Prüfe danach am Computer, z. B. mit `print(rechne(2))`.

Aufruf	Rückgabewert
`rechne(2)`	
`rechne(0)`	
`rechne(-1)`	
`rechne(10)`	
`rechne(rechne(1))`	

Lösung

✅ **Lösung**

Lösung

Aufruf	Rückgabewert
`rechne(2)`	`5`
`rechne(0)`	`-1`
`rechne(-1)`	`-4`
`rechne(10)`	`29`
`rechne(rechne(1))`	`5`

Lösung

```
1 def rechne(x):
2     return 3 * x - 1
3
4 print(rechne(2), rechne(0), rechne(-1), rechne(10))
5 print(rechne(rechne(1)))
```

```
5 -1 -4 29
5
```

Lösung

Ausgabe:

```
5 -1 -4 29
5
```

Lösung

Punkt vor Strich: Zuerst wird `3 * x` gerechnet, dann `1` abgezogen. Den letzten Aufruf liest du von innen nach außen: `rechne(1)` liefert `2`, und `rechne(2)` liefert `5`.

Aufgabe 3: Eintritt im Freibad ●○○

Im Freibad kostet der Eintritt 4 Euro. Für jede Stunde, die man bleibt, kommen 2 Euro dazu.

1. Implementiere eine Funktion `eintritt(stunden)`, die den Preis in Euro **zurückgibt**.
2. Teste sie mit den Werten aus der Tabelle. Rechne die erwarteten Werte vorher von Hand aus.

Aufruf	erwartet	tatsächlich
`eintritt(1)`		
`eintritt(3)`		
`eintritt(0)`		

Lösung

✅ **Lösung**

Lösung

```python
def eintritt(stunden):
    return 4 + 2 * stunden

print(eintritt(1), "erwartet: 6")
print(eintritt(3), "erwartet: 10")
print(eintritt(0), "erwartet: 4")
```

```
6 erwartet: 6
10 erwartet: 10
4 erwartet: 4
```

Lösung

Ausgabe:

```
6 erwartet: 6
10 erwartet: 10
4 erwartet: 4
```

Lösung

Der Parameter `stunden` ist der einzige Wert, den die Funktion braucht. Der Test mit `0` ist ein Sonderfall: Dann bleibt nur der Grundpreis übrig. Häufiger Fehler: `print` statt `return` in der Funktion. Dann erscheint im Test zusätzlich `None`.

Aufgabe 4: Drei Fehler ●○○

Die Funktion soll die Fläche eines Dreiecks berechnen (Grundseite mal Höhe durch 2). Das Programm enthält **drei Fehler**. Finde sie und schreibe das Programm richtig auf. Teste es: Für die Grundseite 4 und die Höhe 3 muss `6.0` herauskommen.

```python
def flaeche_dreieck(g, h)
return g * h / 2

print(flaeche_dreieck(4))
```

```
  File "<Zelle 6>", line 1
    def flaeche_dreieck(g, h)
                            ^
SyntaxError: expected ':'
```

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

1. Nach `def flaeche_dreieck(g, h)` fehlt der Doppelpunkt (`SyntaxError: expected ':'`).
2. Die Zeile mit `return` ist nicht eingerückt. Sie gehört aber zum Funktionsrumpf (`IndentationError`).
3. Beim Aufruf fehlt das zweite Argument. Die Funktion hat zwei Parameter, also braucht der Aufruf zwei Argumente (`TypeError: ... missing 1 required positional argument: 'h'`).

Lösung

```
def flaeche_dreieck(g, h):
    return g * h / 2

print(flaeche_dreieck(4, 3))
```

```
6.0
```

Lösung

Ausgabe:

```
6.0
```

Lösung

Python meldet immer nur den **ersten** Fehler. Wenn du einen behoben hast, starte das Programm erneut und lies die nächste Meldung.

Aufgabe 5: Ein Test deckt den Fehler auf ●●○

Die Funktion soll den Mittelwert von zwei Zahlen zurückgeben. Sie läuft ohne Fehlermeldung.

```
def mittelwert(a, b):
    return a + b / 2
```

1. Rechne von Hand aus, was für `mittelwert(4, 6)` herauskommen **muss**.
2. Teste die Funktion mit `mittelwert(4, 6)` und mit `mittelwert(0, 0)`. Notiere die Ergebnisse.
3. Erkläre den Fehler und verbessere die Funktion.
4. Erkläre, warum der Test mit `0` und `0` allein den Fehler nicht gefunden hätte.

Deine Antwort:

...

Lösung

✅ **Lösung**

Lösung

1. Der Mittelwert von 4 und 6 ist (4+6) : 2 = 5.
2. Die fehlerhafte Funktion liefert:

Lösung

```
def mittelwert(a, b):
    return a + b / 2

print(mittelwert(4, 6), "erwartet: 5")
print(mittelwert(0, 0), "erwartet: 0")
```

```
7.0 erwartet: 5
0.0 erwartet: 0
```

Lösung

Ausgabe:

```
7.0 erwartet: 5
0.0 erwartet: 0
```

Lösung

3. Wegen „Punkt vor Strich“ rechnet Python zuerst `b / 2` und addiert dann `a`: 4 + 3 = 7. Die Summe muss in Klammern stehen:

Lösung

```python
def mittelwert(a, b):
    return (a + b) / 2

print(mittelwert(4, 6), "erwartet: 5")
print(mittelwert(0, 0), "erwartet: 0")
```

```
5.0 erwartet: 5
0.0 erwartet: 0
```

Lösung

Ausgabe:

```
5.0 erwartet: 5
0.0 erwartet: 0
```

Lösung

4. Bei `0` und `0` liefern beide Rechnungen zufällig dasselbe Ergebnis. Ein einzelner Test kann also „Glück haben“. Deshalb testet man immer mit mehreren, verschiedenen Werten.

Aufgabe 6: Kinokarten ●●○

Ein Kino verlangt je nach Alter einen anderen Preis:

- unter 6 Jahren: 0 Euro (frei),
- von 6 bis unter 18 Jahren: 6 Euro,
- ab 18 Jahren: 9 Euro.

1. Implementiere eine Funktion `kinopreis(alter)`, die den Preis **zurückgibt**. Verwende eine mehrseitige Verzweigung.
2. Teste die Funktion an den Grenzen: mit den Altern 5, 6, 17 und 18.

Lösung

✅ **Lösung**

Lösung

```python
def kinopreis(alter):
    if alter < 6:
        return 0
    elif alter < 18:
        return 6
    else:
        return 9

print(kinopreis(5), "erwartet: 0")
print(kinopreis(6), "erwartet: 6")
print(kinopreis(17), "erwartet: 6")
print(kinopreis(18), "erwartet: 9")
```

```
0 erwartet: 0
6 erwartet: 6
6 erwartet: 6
9 erwartet: 9
```

Lösung

Ausgabe:

```
0 erwartet: 0
6 erwartet: 6
6 erwartet: 6
9 erwartet: 9
```

Lösung

Die Funktion hat drei `return`-Anweisungen, aber bei jedem Aufruf wird nur **eine** davon ausgeführt. Die Bedingung `alter < 18` im `elif` wird nur geprüft, wenn `alter < 6` falsch war. Deshalb muss man dort nicht noch einmal `alter >= 6` schreiben. Die vier Testwerte liegen genau an den Grenzen, wo Fehler wie `<=` statt `<` auffallen würden.

Aufgabe 7: Funktion in der Schleife ●●○

Fülle das Ablaufprotokoll des Hauptprogramms aus. Trage nach jedem Schritt die Werte von `i` und `summe` ein (– bedeutet: noch kein Wert). Gib dann an, was das Programm ausgibt.

```python
def dreifach(zahl):
    return 3 * zahl

summe = 0
for i in range(1, 4):
    summe = summe + dreifach(i)
print("Summe:", summe)
```

```
Summe: 18
```

Schritt	Zeile	Anweisung	i	summe
1	1	`def dreifach(zahl):`	-	-
2	4	`summe = 0`	-	
3	5	`for i in range(1, 4):`		
4	6	`summe = summe + dreifach(i)`		
5	5	`for i in range(1, 4):`		
6	6	`summe = summe + dreifach(i)`		
7	5	`for i in range(1, 4):`		
8	6	`summe = summe + dreifach(i)`		
9	5	`for i in range(1, 4):`		
10	7	`print("Summe:", summe)`		

Deine Antwort:

...

Lösung

✅ **Lösung**

Lösung

Schritt	Zeile	Anweisung	i	summe
1	1	`def dreifach(zahl):`	-	-
2	4	`summe = 0`	-	`0`
3	5	`for i in range(1, 4):`	`1`	`0`
4	6	`summe = summe + dreifach(i)`	`1`	`3`
5	5	`for i in range(1, 4):`	`2`	`3`
6	6	`summe = summe + dreifach(i)`	`2`	`9`
7	5	`for i in range(1, 4):`	`3`	`9`
8	6	`summe = summe + dreifach(i)`	`3`	`18`
9	5	`for i in range(1, 4):`	`3`	`18`
10	7	`print("Summe:", summe)`	`3`	`18`

Lösung

```python
def dreifach(zahl):
    return 3 * zahl

summe = 0
for i in range(1, 4):
    summe = summe + dreifach(i)
print("Summe:", summe)
```

```
Summe: 18
```

Lösung

Ausgabe:

```
Summe: 18
```

Lösung

In jedem Durchlauf springt Python in die Funktion und kommt mit dem Dreifachen von `i` zurück: 3, 6 und 9. Zusammen ergibt das 3 + 6 + 9 = 18. Die Zeilen 1 und 2 werden beim ersten Lesen nicht ausgeführt, Python merkt sich dort nur die Funktion.

Aufgabe 8: Quersumme als Funktion ●●●

Im Auftrag *Quersumme* (Abschnitt über die while-Schleife) hast du die Quersumme einer Zahl berechnet (Quersumme von 384 ist 3 + 8 + 4 = 15). Zur Erinnerung: `zahl % 10` liefert die letzte Ziffer, `zahl // 10` entfernt sie.

1. Implementiere eine Funktion `quersumme(zahl)`, die die Quersumme **zurückgibt**. Teste sie mit 384 (erwartet: 15), 7 (erwartet: 7) und 1000 (erwartet: 1).
2. Benutze die Funktion in einer `for`-Schleife: Gib alle Zahlen von 1 bis 100 aus, deren Quersumme genau 10 ist.

> **Tipp:** Schreibe die Schleife aus dem Abschnitt über die while-Schleife in den Funktionsrumpf und ersetze das `print` am Ende durch `return`. Im zweiten Teil steht der Aufruf `quersumme(i)` in der Bedingung einer `if`-Anweisung, so wie `ist_gerade(i)` im Auftrag *Gerade Zahlen*.

Lösung

✅ **Lösung**

Lösung

```python
def quersumme(zahl):
    summe = 0
    while zahl > 0:
        summe = summe + zahl % 10
        zahl = zahl // 10
    return summe

print(quersumme(384), "erwartet: 15")
print(quersumme(7), "erwartet: 7")
print(quersumme(1000), "erwartet: 1")

for i in range(1, 101):
    if quersumme(i) == 10:
        print(i)
```

```
15 erwartet: 15
7 erwartet: 7
1 erwartet: 1
19
28
37
46
55
64
73
82
91
```

Lösung

Ausgabe:

```
15 erwartet: 15
7 erwartet: 7
1 erwartet: 1
19
```

```
28
37
46
55
64
73
82
91
```

Lösung

- **Zeilen 1–6:** Die Funktion arbeitet mit ihrem Parameter `zahl` wie das Programm aus dem Abschnitt über die while-Schleife. Statt die Quersumme auszugeben, gibt sie sie mit `return` zurück. Wichtig: `return` steht **nach** der Schleife, also nicht eingerückt unter `while`.
- **Zeilen 12–14:** Für jede Zahl von 1 bis 100 wird die Funktion aufgerufen. Nur wenn der Rückgabewert 10 ist, wird die Zahl ausgegeben.

Die Variable `zahl` wird in der Funktion bis auf 0 verkleinert. Das stört das Hauptprogramm nicht: Der Parameter ist eine lokale Variable, und `i` behält seinen Wert.