

Informatik 2 für Regenerative Energien

Klausur vom 11. Juli 2025: Lösungen

Jörn Loviscach

Versionsstand: 11. Juli 2025, 22:09



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Germany License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/de/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Anmerkungen stehen in Kursivschrift.

1. Die Fehler:

Zeile	korrekter Programmtext
11	pass
30	self.zustand = wert
35	wert = self.zustand + änderung
40	self.setze_zustand(wert)
52	class Raum:
53	def __init__(self, name):
55	self.größen = []
78	continue
80	regel = raum.modus[größe]
82	aktueller_wert = sensor.messen()
87	diff = -diff
95	aktor.ändere_zustand(inkr)
98	mein_hörsaal = Raum('D2')
113	automation = Automation()
114	automation.setze_modus(mein_hörsaal, klausurmodus)

Das Erben von ABC in class Raum(ABC) : in Zeile 52 führt nicht zu Fehlern, weil die Klasse keine abstrakten Methoden hat; aber es ist von der Bedeutung her falsch, weil die Klasse nicht abstrakt verwendet wird. Im selben Sinn habe ich den Vorschlag als korrekt gewertet, die Klasse Aktor abstrakt zu machen. – Und eine Denkaufgabe: Welche dieser Fehler wären unterkringelt gewesen, wenn der Code Typangaben gehabt hätte?

2. Die ersten drei Zeilen der Ausgabe sind:

```
LichtD2: 27.0  
JalousieD2: 24.0  
LichtD2: 31.0
```

Begründung: Beleuchtung und Jalousie starten beide bei 23. Die Helligkeit ist konstant 30 und liegt damit 40 unter dem Soll; in jedem Schritt wird die Beleuchtung um ein Zehntel davon hochgeregelt. Die Blendwirkung von 90 ist über dem Maximum 30; in jedem Schritt wird die Jalousie eine Stufe weiter geschlossen.

3. Dies am Anfang dem Methode `regle_raum()`:

```
if raum.modus is None:
    raise ValueError(f'Raum {raum.name} ohne Modus.')
```

Oder besser statt ValueError eine selbstdefinierte Unterklasse von Exception.

4. *Zum Beispiel so:* Die `__init__()`-Methode der Klasse `Sensor` wird um `self.werte = []` erweitert; alle Unterklassen von `Sensor` erhalten in ihrer Methode `messen()` noch ein `self.werte.append(wert)`, wozu dort noch die Variable `wert` anzulegen und mit dem Messwert zu füllen ist. *Programmiertechnisch unschön: Man könnte das in (neuen?) Unterklassen vergessen.*

5. *Zum Beispiel so:*

```
class Aktorgruppe(Aktor):
    def __init__(self, name, aktoren):
        super().__init__(name)
        self.aktoren = list(aktoren)

    def setze_zustand(self, wert):
        self.zustand = wert
        for aktor in self.aktoren:
            aktor.setze_zustand(wert)
```

Die Methode `ändere_zustand()` muss nicht überschrieben werden, weil ihre Implementierung in der Oberklasse `Aktor` schon (polymorph) die Methode `setze_zustand()` der Unterklasse `Aktorgruppe` aufruft. Das `super().__init__(name)` sorgt dafür, dass `self.zustand` auch für die gesamte `Aktorgruppe` angelegt wird und nicht nur für jeden darin enthaltenen `Aktor`. – Alternativ könnte man auf `super().__init__(name)` und `self.zustand = wert` verzichten, müsste dann aber auch `ändere_zustand()` überschreiben.

Das `... = list(aktoren)` statt `... = aktoren` ist eine in dieser Klausur nicht verlangte technische Feinheit. Sie ist hier notwendig, damit die Gruppe eine eigene Kopie der übergebenen Aktoren erhält. Andernfalls würde die Liste von außerhalb der Klasse verändert werden können (Referenz!), was zu unerwartetem Verhalten führen könnte.

6. *Zum Beispiel so:*

```
class Beleuchtung(Aktor):
    auf_hohe_werte_gestellt = 0
    def setze_zustand(self, wert):
        if wert > 90:
            Beleuchtung.auf_hohe_werte_gestellt += 1
            super().setze_zustand(wert)
```

7. `dict[str, dict[str, list[tuple[str, int]] | set[str]]]`

8. `x == 'Prag'`

Weil `plan_a['Route']` eine Referenz auf die dieselbe Liste ist, auf die auch `städte` verweist, ist `städte.append('Prag')` auch dort sichtbar. Deshalb ist der hinterste Eintrag eine Referenz auf 'Prag'.

`y==8`

Da sowohl `plan_a` als auch `plan_b` unter 'Dauer' auf dieselbe Liste verweisen, sind Änderungen dieser einen Liste über `plan_a` sowie `plan_b` möglich und sichtbar. Die Liste wird als `[3, 4]` erstellt, dann zu `[3, 6]` verändert und schließlich zu `[2, 6]`. Deren Summe ist 8.

`z==False`

Die Variable `z` enthält das Ergebnis der Überprüfung, ob 'Prag' in `plan_b['Route']` enthalten ist. Aber `plan_b['Route']` verweist von Anfang an auf eine eigene Liste `['Rom', 'Madrid']`, die unverändert bleibt. 'Prag' wird dort nie eingefügt.