Codeklasse

Was kann der Held?

Thema: Methoden & Objektzustand

Name: _____

Klasse: _____ Datum: _____

Grundlagen & Merkkasten

WISSEN

Objekte besitzen nicht nur Eigenschaften (Attribute), sondern auch Fähigkeiten.
In der objektorientierten Programmierung nennt man Funktionen, die direkt zu einer Klasse gehören, Methoden.
Methoden können den inneren Zustand eines Objekts abfragen, verändern oder Berechnungen mit den Attributen durchführen.
Über den Parameter `self` greift jede Methode auf die Attribute des konkreten Objekts zu.

`class Konto:`
`  def __init__(self, inhaber, guthaben):`
`  self.inhaber = inhaber`
`  self.guthaben = guthaben`
	Der Konstruktor legt die Startattribute des Objekts fest.

`def einzahlen(self, betrag):` `  self.guthaben += betrag`	Eine Methode verändert ein bestehendes Instanzattribut über `self`.
`mein_konto = Konto("Alex", 100)` `mein_konto.einzahlen(50)`	Methodenaufruf auf einem Objekt mit der Punkt-Notation.

1. Code-Puzzle (Reihenfolge & Einrückung)

AUFGABE 1

Bringe die Code-Blöcke in die richtige Reihenfolge und rücke sie korrekt ein, um die Klasse `Bankkonto` zu definieren, eine Einzahlung vorzunehmen und anschließend den Kontostand auszugeben.

[ ]	`class Bankkonto:`
[ ]	`def __init__(self, inhaber, guthaben):`
[ ]	`self.inhaber = inhaber`
[ ]	`self.guthaben = guthaben`
[ ]	`def einzahlen(self, betrag):`
[ ]	`self.guthaben += betrag`
[ ]	`konto = Bankkonto('Samira', 100)`
[ ]	`konto.einzahlen(50)`
[ ]	`print(f'Kontostand von {konto.inhaber}: {konto.guthaben} Euro')`
[ ]	`guthaben += betrag`
[ ]	`def einzahlen(betrag):`

Hinweis: Trage in die eckigen Klammern die korrekte Reihenfolge (1, 2, 3...) ein. Streiche überflüssige oder fehlerhafte Zeilen (Distraktoren) durch!

2. Fehlersuche & Code-Reparatur

AUFGABE 2

In der Methode `abheben` hat sich ein Fehler eingeschlichen: Der Parameter `self` fehlt im Methodenkopf und das Attribut wird nicht korrekt über die Instanz angesprochen.
Repariere die Methode `abheben`, sodass der Betrag ordnungsgemäß vom Guthaben abgezogen wird.

Fehlerhafter Ausgangs-Code:

```
class Bankkonto:
    def __init__(self, inhaber, guthaben):
        self.inhaber = inhaber
        self.guthaben = guthaben

    def abheben(betrag):
        guthaben = guthaben - betrag

    def anzeigen(self):
        print(f"Konto von {self.inhaber}: {self.guthaben} EUR")

konto = Bankkonto("Samira", 100)
konto.abheben(30)
konto.anzeigen()
```

Deine Korrektur / Notizen:

3. Programmier-Challenge

AUFGABE 3

Erstelle die Klasse `Bankkonto` mit folgenden Anforderungen:
1. Konstruktor `__init__(self, inhaber, guthaben)` speichert `self.inhaber` und `self.guthaben`.
2. Methode `einzahlen(self, betrag)` erhöht `self.guthaben` um `betrag`.
3. Methode `abheben(self, betrag)` verringert `self.guthaben` um `betrag`.
4. Methode `kontostand(self)` gibt den formatierten String `"Inhaber: <inhaber> | Guthaben: <guthaben>"` auf der Konsole aus (`print`).

Erstelle anschließend ein Objekt `mein_konto` für `'Samira'` mit `100` Euro Startguthaben.
Führe danach folgende Schritte aus:
- Zahle `50` Euro ein.
- Hebe `30` Euro ab.
- Rufe die Methode `kontostand()` auf.

Dein Python-Code:

Reflexion: Kapselung und Zustand

AUFGABE 4

Warum ist es in größeren Programmen vorteilhaft, den Kontostand eines Objekts über eine Methode wie `einzahlen(betrag)` oder `abheben(betrag)` zu verändern, anstatt direkt von außen `konto.guthaben = konto.guthaben + 50` zu schreiben? Welche Sicherheits- und Validierungsmechanismen werden dadurch möglich?