Codeklasse

Der Avatar-Konfigurator

Thema: Objektorientierte Programmierung: Konstruktor (__init__) & self

Name: _____

Klasse: _____ Datum: _____

Grundlagen & Merkkasten

WISSEN

Beim Erzeugen eines neuen Objekts aus einem Klassenbauplan müssen individuelle Startwerte wie Name, Rolle oder Ausdauer festgelegt werden.
In der objektorientierten Programmierung geschieht dies über eine spezielle Initialisierungsmethode: den Konstruktor `__init__`.
Der Parameter `self` fungiert dabei als Referenz auf die aktuell entstehende Instanz und bindet übergebene Daten als Attribute direkt an das jeweilige Objekt.

`def __init__(self, name, level):`	Konstruktormethode: Wird beim Erzeugen eines neuen Objekts automatisch ausgeführt.
`self.name = name`	Speichert den übergebenen Parameter als Attribut 'name' im konkreten Objekt.
`spieler = Charakter('Aria', 1)`	Instanziierung: Übergibt Startargumente direkt an den Konstruktor (self wird automatisch übergeben).
`print(spieler.name)`	Punktnotation: Greift auf das Attribut 'name' des Objekts 'spieler' zu.

1. Code-Puzzle (Reihenfolge & Einrückung)

AUFGABE 1

Bringe die Code-Blöcke in die richtige logische Reihenfolge und rücke sie passend ein, um die Klasse `Held` mit Konstruktor zu definieren, eine Instanz zu erzeugen und die Attribute auszugeben.

[ ]	`class Held:`
[ ]	`def __init__(self, name, rolle, energie):`
[ ]	`self.name = name`
[ ]	`self.rolle = rolle`
[ ]	`self.energie = energie`
[ ]	`spieler = Held('Aria', 'Waldlaeufer', 100)`
[ ]	`print(f'Held: {spieler.name}')`
[ ]	`print(f'Rolle: {spieler.rolle}')`
[ ]	`print(f'Energie: {spieler.energie}')`
[ ]	`name = self.name`
[ ]	`def init(self, name, rolle, energie):`

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

Im folgenden Code fehlen wesentliche Bestandteile der Konstruktormethode und Attributbindung:
1. Ergänze die Unterstriche der speziellen Konstruktormethode `__init__`.
2. Binde den Parameter `level` mit `self` dauerhaft an das Objekt.
3. Korrigiere die Erzeugung des Objekts `held`, sodass `"Aria"` und Level `1` übergeben werden.

Fehlerhafter Ausgangs-Code:

```
1  class Charakter:
2      def init(self, name, level):
3          self.name = name
4          level = level
5
6  held = Charakter()
7
8  print("Name:", held.name)
9  print("Level:", held.level)
```

Deine Korrektur / Notizen:

3. Programmier-Challenge

AUFGABE 3

Erstelle eine Klasse namens `Held` für einen Charakter-Generator:
1. Definiere die Methode `__init__(self, name, rolle, energie)`.
2. Speichere alle drei Parameter als Instanzattribute (`self.name`, `self.rolle`, `self.energie`).
3. Erstelle eine Instanz `spieler` mit den Werten `"Aria"`, `"Waldlaeufer"` und `100`.
4. Gib die Eigenschaften des Objekts in genau folgendem Format aus:

```
  Charakter: Aria
 Klasse: Waldlaeufer
 Energie: 100
```

Dein Python-Code:

Transfer-Aufgabe: Die Bedeutung von self

AUFGABE 4

Betrachte folgende Zeile innerhalb der Konstruktormethode:

```
self.energie = start_energie
```

Erkläre präzise den Unterschied zwischen `self.energie` und `start_energie`. Was passiert, wenn die Zuweisung stattdessen nur `energie = start_energie` lautet?