Codeklasse

Das Superhelden-Upgrade

Thema: Methoden überschreiben & super()

Name: _____

Klasse: _____ Datum: _____

Grundlagen & Merkkasten

WISSEN

Vererbung erlaubt es, Eigenschaften einer Basisklasse zu übernehmen. Häufig benötigt eine spezialisierte Unterklasse jedoch zusätzliche Attribute oder ein verändertes Verhalten.
Durch das Überschreiben von Methoden (Method Overriding) passen wir das Verhalten an.
Mit `super()` greifen wir dabei direkt auf die Methoden der Elternklasse zu, um bestehende Logik – wie die Initialisierung der Grundwerte – wiederzuverwenden, statt sie neu zu schreiben.

`class Held:`
`  def __init__(self, name, power):`
`  self.name = name`
`  self.power = power`
	Basisklasse (Elternklasse) definiert grundlegende Attribute.

`class Superheld(Held):`
`  def __init__(self, name, power, boost):`
`  super().__init__(name, power)`
`  self.boost = boost`
	`super().__init__()` ruft den Konstruktor der Elternklasse auf und ergänzt neue Attribute.

`  def angriff(self):`
`  super().angriff()`
`  print('Zusatz-Effekt!')`
	Erweitert eine vererbte Methode um zusätzliche Anweisungen.

1. Code-Puzzle (Reihenfolge & Einrückung)

AUFGABE 1

Die Basisklasse `Held` ist bereits fertig vordefiniert: Sie besitzt einen Konstruktor `__init__(self, name, power)` sowie eine Methode `angriff()`, die `"Standard-Schlag mit Kraft: <power>"` ausgibt.

Bringe die Code-Blöcke in die richtige Reihenfolge, um die Klasse `Superheld` von `Held` abzuleiten, den Basis-Konstruktor mit `super()` aufzurufen und die Methode `angriff` zu erweitern.

[ ]	`class Superheld(Held):`
[ ]	`def __init__(self, name, power, boost):`
[ ]	`super().__init__(name, power)`
[ ]	`self.boost = boost`
[ ]	`def angriff(self):`
[ ]	`super().angriff()`
[ ]	`print(f'Spezial-Boost: +{self.boost}')`
[ ]	`hero = Superheld('Volt-Runner', 40, 30)`
[ ]	`hero.angriff()`
[ ]	`super().__init__(self, name, power)`

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 Unterklasse `CyberHeld` fehlen zwei wichtige Stellen:
1. Der Aufruf des Eltern-Konstruktors mit `super()`.
2. In der Methode `status` soll zuerst die `status`-Methode der Elternklasse aufgerufen werden.

Ergänze die Lücken, sodass Name, Basis-Power und der Schildwert sauber ausgegeben werden.

Fehlerhafter Ausgangs-Code:

```
class BasisHeld:
    def __init__(self, name, power):
        self.name = name
        self.power = power

    def status(self):
        print(f"Held: {self.name} | Power: {self.power}")

class CyberHeld(BasisHeld):
    def __init__(self, name, power, shield):
        # LÜCKE 1: Rufe den Konstruktor der Elternklasse auf
        ___.__init__(name, power)
        self.shield = shield

    def status(self):
        # LÜCKE 2: Rufe die status-Methode der Elternklasse auf
        ___.___()
        print(f"Schild: {self.shield}")

hero = CyberHeld("Volt-Runner", 40, 100)
hero.status()
```

Deine Korrektur / Notizen:

3. Programmier-Challenge

AUFGABE 3

Erstelle ein modulares Klassensystem für Spezial-Helden:

1. Definiere die Basisklasse `Hero`:
- Konstruktor `__init__(self, name, base_dmg)` speichert `self.name` und `self.base_dmg`.
- Methode `total_dmg(self)` gibt den Basis-Schaden (`self.base_dmg`) zurück.

2. Definiere die abgeleitete Klasse `TunedHero(Hero)`:
- Konstruktor `__init__(self, name, base_dmg, boost)` ruft `super().__init__(name, base_dmg)` auf und speichert zusätzlich `self.boost = boost`.
- Überschreibe `total_dmg(self)`: Sie soll die Summe aus dem vererbten Basisschaden (mittels `super().total_dmg()`) und `self.boost` zurückgeben.

3. Erstelle ein Objekt `champion` der Klasse `TunedHero` mit:
- Name: `"Volt-Runner"`
- Basis-Schaden: `40`
- Boost: `30`

4. Speichere das Ergebnis von `champion.total_dmg()` in der Variablen `schaden` und gib aus:
`Gesamtschaden: <schaden>`

Dein Python-Code:

Transfer-Aufgabe: Warum `super()`?

AUFGABE 4

Ein Programmierer schreibt in einer abgeleiteten Klasse `self.name = name` und `self.power = power` erneut manuell in den `__init__`-Konstruktor, anstatt `super().__init__(name, power)` aufzurufen.

Erkläre, welches Software-Entwicklungsprinzip dadurch verletzt wird und welche Probleme entstehen können, wenn sich später die Basisklasse ändert.