Fehlerbehandlung & Debugging

Python 3 · kompakte Grundlagen mit direkt nutzbaren Beispielen

1 | Drei Fehlerarten

Art	Beispiel
Syntaxfehler	Doppelpunkt fehlt
Laufzeitfehler	Division durch 0
Logikfehler	falsche Formel
Datenfehler	ungültige Eingabe
Umgebungsfehler	Datei fehlt

2 | Traceback lesen

- Letzte Zeile: Fehlertyp und Nachricht.
- Darüber: Aufrufkette und Zeilennummern.
- Eigene Datei zuerst untersuchen.
- Fehlerstelle kann Folge eines früheren Problems sein.
- Meldung vollständig kopieren.

3 | try und except

```
try:
    wert = float(input("Wert: "))
except ValueError:
    print("Keine gültige Zahl.")
```

4 | Gezielte Exceptions

- Nur erwartete Fehler abfangen.
- Mehrere Typen getrennt behandeln.
- Kein nacktes except ohne guten Grund.

```
try:
    ergebnis = a / b
except ZeroDivisionError:
    print("b darf nicht 0 sein")
```

5 | else und finally

- else läuft ohne Fehler.
- finally läuft immer, z. B. zum Aufräumen.

```
try:
    datei = open(pfad)
except OSError:
    print("Öffnen fehlgeschlagen")
else:
    print(datei.read())
finally:
    print("fertig")
```

6 | raise

- Eigene ungültige Zustände melden.
- Passenden Exception-Typ und klare Nachricht nutzen.

```
def prozent(wert):
    if not 0 <= wert <= 100:
        raise ValueError("0 bis 100 erwartet")
```

Fehlerbehandlung & Debugging

Python 3 · kompakte Grundlagen mit direkt nutzbaren Beispielen

7 | Debug-Ausgaben

- Zwischenwerte mit Bezeichnung ausgeben.
- Später entfernen oder logging nutzen.

```
print(f"DEBUG masse={masse_g!r}")
print(f"DEBUG kosten={kosten:.4f}")
```

8 | Debugger nutzen

- Breakpoint vor verdächtiger Stelle setzen.
- Schrittweise ausführen: step over/into/out.
- Variablen und Call Stack beobachten.
- Bedingte Breakpoints bei großen Schleifen.
- Nach Fix Breakpoints entfernen.

9 | Minimalbeispiel

- Unbeteiligten Code entfernen.
- Eingabedaten stark verkleinern.
- Externe Systeme durch feste Testwerte ersetzen.
- Fehler reproduzierbar machen.
- Dann Ursache statt Symptom korrigieren.

10 | Assertions

- Prüfen interne Annahmen während Entwicklung.
- Nicht für normale Benutzereingaben verwenden.

```
def mittelwert(werte):
    assert len(werte) > 0, "Liste leer"
    return sum(werte) / len(werte)
```

11 | Fehlerprotokoll

- Schritte zur Reproduktion notieren.
- Eingaben, Version und Umgebung erfassen.
- Erwartetes und tatsächliches Ergebnis vergleichen.
- Traceback unverändert sichern.
- Fix mit Regressionstest absichern.

12 | Mini-Übung: sichere Division

Aufgabe

Fange ungültige Zahlen und Division durch null getrennt ab.

Musterlösung

```
try:
    a = float(input("a: "))
    b = float(input("b: "))
    print(a / b)
except ValueError:
    print("Zahl erwartet")
except ZeroDivisionError:
    print("b darf nicht 0 sein")
```