PROFESSOR ERKLÄRBÄR

Serie Informatik · Folge 15 · Grundlagen

INFORMATIK

LEHRERINFO ZUM VIDEO

Wir programmieren einen Taschenrechner

FACH	STUFE	DAUER	VIDEO
Informatik	Klasse 7–9	13 min	youtube.com/@Erklaerbaer

01 Lernziele

Nach dem Video können die Schülerinnen und Schüler …

- ein Programm von den Anforderungen über den Ablaufplan bis zum fertigen Code entwickeln,
- Variablen, Eingaben, Bedingungen, Schleifen und Funktionen in einem Projekt verbinden,
- Eingaben prüfen und Fehler abfangen (Division durch null, keine Zahl, Komma statt Punkt),
- typische Fehler im Code finden und beheben,
- ein Programm mit einer Testtabelle systematisch testen,
- das Programm erweitern (Prozent, Potenz, Sparrechner).

02 Voraussetzungen

Ablaufplan (Folge 6), `input`, `float`, `print` (Folge 7), `if`/`elif`/`else` und `while` (Folge 8), Funktionen und Listen (Folge 9). Python 3 mit einer einfachen Umgebung, z. B. Thonny. Ganze Ausdrücke mit Punkt vor Strich folgen in Folge 15 · Advanced.

03 Ablauf des Videos

ZEIT	KAPITEL	INHALT UND PRAXISBEISPIEL
0:00	Einstieg	15 · 20 = 300: Taschengeld bis zum Sommer · Ident · Fahrplan
0:26	Anforderungen	Checkliste · Pizza 17,50 € zu zweit = 8,75 €
1:06	Der Plan	Ablaufplan · Liste der Funktionen · Versionen · Merksatz
1:56	Version 0.1	`input`, `float`, `print` · ValueError bei «3,5» · `replace`
2:39	Version 0.2	`if`/`elif`/`else` · Division durch null · `while`-Schleife
3:22	Version 0.3	vier Rechenfunktionen · Verlauf als Liste

4:03	Fehler abfangen	`try`/`except` · `read_number` · Merksatz
4:56	Fehlersuche	drei eingebaute Fehler: `=` statt `==`, fehlendes `float` («75»), Einrückung
5:59	Testen	Testtabelle · 0,1 + 0,2 und `round` PAUSE-FRAGE fehlender Testfall · Merksatz
7:28	Erweitern	19 % von 80 · 2 ** 10 · Sparrechner · Umrechner PAUSE-FRAGE Rest `%` · Ausblick Knöpfe
8:57	Typische Fehler	vier Irrtümer (siehe 04)
9:43	Rückblick	Bausteine aus 6, 7, 8, 9 und 14 · Taschenrechner komplett
10:15	Zusammenfassung	Kernaussagen · zurück zum Einstieg
10:58	Quiz	4 Fragen · Endkarte Folge 16

04 Typische Fehlvorstellungen

- „Einfach drauflosprogrammieren geht schneller“ – erst Anforderungen und Ablauf, dann Code in kleinen Versionen (Kapitel 2 und 10).
- „Mit 7 + 5 ist das Programm getestet“ – auch Randfälle testen: 0, negative Zahlen, Text, Komma, sehr grosse Zahlen (Kapitel 8 und 10).
- „Ein langer Block ist übersichtlicher“ – Funktionen trennen die Aufgaben und lassen sich einzeln testen (Kapitel 5 und 10).
- „`eval(input())` ist eine praktische Abkürzung“ – `eval` lässt beliebigen Code laufen, eine Sicherheitslücke (Kapitel 10, Folge 13 · Advanced).

05 Einsatz im Unterricht

Mitbauen: Das Video nach jeder Version anhalten; die Klasse tippt die Version in Thonny nach und testet sie.

Fehlerjagd: In Partnerarbeit baut eine Person drei Fehler in den Code ein, die andere sucht sie.

Testtabelle: Jede Gruppe erfindet fünf Testfälle; danach tauschen und gegenseitig testen.

Erweitern: Aufgaben 5 und 6 als Differenzierung; schnelle Lernende bauen den Umrechner.

Hinweis: Der Code im Anhang ist mit Python 3.12 getestet. `float("3,5")` löst einen ValueError aus, darum ersetzt `replace` zuerst das Komma. 0,1 + 0,2 ergibt 0.30000000000000004, weil 0,1 im Binärsystem nicht endlich darstellbar ist (Folge 2 · Advanced); `round(…, 10)` zeigt 0.3. Die Funktions- und Variablennamen sind englisch (add, history) wie in Folge 9; `nochmal` stammt aus Folge 8.

06 Quellen

- Python-Dokumentation: Tutorial «Errors and Exceptions» – docs.python.org/3/tutorial/errors.html
- Python-Dokumentation: «Floating-Point Arithmetic: Issues and Limitations» – docs.python.org/3/tutorial/floatingpoint.html

- Python-Dokumentation: Built-in Functions `input`, `float`, `round`, `eval` – docs.python.org/3/library/functions.html
- Thonny – Python-Umgebung für Einsteiger, thonny.org · tkinter – docs.python.org/3/library/tkinter.html
- Lehrplan 21, Modul „Medien und Informatik“, MI.2.2 Algorithmen und Programmieren – lehrplan21.ch

NAME _____ KLASSE _____ DATUM _____

Wir programmieren einen Taschenrechner

1 Merksätze ergänzen ●○○

Gute Programme entstehen in kleinen _____: planen, _____, testen, verbessern. · Benutzereingaben immer _____. · Ein Test vergleicht das _____ mit dem _____ Ergebnis.

2 Was kommt heraus? ●○○

a) `float("2,5")` b) `"2,5".replace(",", ".")` c) `"4" + "2"` d) `7 / 2` e) `10 / 0`

3 Testtabelle ●●○

Trage das erwartete Ergebnis ein und **ergänze** zwei eigene Testfälle.

EINGABE	ERWARTET	TATSÄCHLICH
9 − 12		
3,5 · 2		
1 : 4		
8 : 0		
«zwei» als Zahl 1		

4 Fehler finden ●●○

In diesem Code stecken drei Fehler. **Finde** und **verbessere** sie.

```
def multiply(a, b)
    return a * b
x = input("Zahl: ")
if x == 0:
print("null")
```

5 Rest-Rechnung einbauen ●●○

Schreibe die Funktion `remainder(a, b)` und den passenden `elif`-Zweig. Was ergibt `17 % 5`?

6 Sparrechner ●●●

a) **Schreibe** ein Programm, das nach dem Betrag pro Woche und der Anzahl Wochen fragt und die Summe ausgibt. b) **Erweitere** es: Du sparst 15 € pro Woche und willst 200 €. Nach wie vielen Wochen hast du das Ziel erreicht?

Lösungen

1 Merksätze

Schritten, bauen · prüfen · erwartete, tatsächlichen

2 Was kommt heraus?

a) ValueError (Komma statt Punkt) · b) "2.5" (ein Text) · c) "42" (Texte werden aneinandergehängt) · d) 3.5 · e) ZeroDivisionError

3 Testtabelle

−3 · 7 · 0,25 · Meldung «Division durch null!» · Meldung «Bitte eine Zahl eingeben.» und neue Eingabe · eigene Fälle z. B. eine unbekannte Rechenart (7 ? 5), eine sehr grosse Zahl, 0 + 0.

4 Fehler finden

1. Nach `def multiply(a, b)` fehlt der Doppelpunkt (SyntaxError). 2. `x` ist Text, darum ist `x == 0` nie wahr: `x = float(input("Zahl: "))`. 3. `print("null")` muss eingerückt sein, sonst IndentationError.

5 Rest-Rechnung

`def remainder(a, b): return a % b` · `elif op == "%": result = remainder(a, b)` · Menütext `"+ - * / % ? "` · 17 % 5 = 2, denn 17 = 3 · 5 + 2. Auch hier vorher prüfen, ob b = 0 ist.

6 Sparrechner

a) `betrag = read_number("Betrag pro Woche: "), wochen = read_number("Anzahl Wochen: "), print("Gespart:", betrag * wochen, "Euro")` · b) 14 Wochen: nach 13 Wochen sind es 195 €, nach 14 Wochen 210 €. Mit einer Schleife: `summe = 0, wochen = 0, while summe < 200: summe = summe + 15; wochen = wochen + 1.`

Q Quiz im Video

1 B (Fehler, während das Programm läuft) · 2 C (ValueError) · 3 B (Aufgaben trennen, wiederverwenden, testen) · 4 A (Eingabe mit erwartetem Ergebnis)

Anhang: Taschenrechner 1.0

Vollständiger Code aus dem Video (Python 3, getestet mit Python 3.12). Zum Abtippen in Thonny oder als Vorlage für die Erweiterungen.

```
# Taschenrechner 1.0 - Serie Informatik, Folge 15 (Grundlagen)
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    return a / b

def read_number(prompt):
    while True:
        text = input(prompt).replace(",", ".")
        try:
            return float(text)
        except ValueError:
            print("Bitte eine Zahl eingeben.")

history = []
nochmal = "j"
while nochmal == "j":
    a = read_number("Zahl 1: ")
    op = input("+ - * / ? ")
    b = read_number("Zahl 2: ")
    result = None
    if op == "+":
        result = add(a, b)
    elif op == "-":
        result = subtract(a, b)
    elif op == "*":
        result = multiply(a, b)
    elif op == "/" and b == 0:
        print("Division durch null!")
    elif op == "/":
        result = divide(a, b)
    else:
        print("Unbekannte Rechenart")
    if result is not None:
        print("=", round(result, 10))
        history.append([a, op, b, result])
    nochmal = input("Nochmal? (j/n) ")

for eintrag in history:
    print(eintrag)
```

`result = None` heisst «noch kein Ergebnis»; nur wenn gerechnet wurde, zeigt der Rechner das Ergebnis und schreibt es in den Verlauf.