1.4 Übung: Verzweigungen lesen, testen und erweitern

Leitidee

Programmiererinnen und Programmierer schreiben nicht nur neue Programme. Mindestens genauso oft **lesen** sie fremden Code, **suchen Fehler** darin und **erweitern** ein Programm, das schon da ist. Die Fehlersuche heißt auf Englisch *Debugging* – wörtlich „Entwanzen“, denn einen Programmfehler nennt man einen *Bug* („Käfer“). In diesem Abschnitt übst du genau diese drei Tätigkeiten mit Verzweigungen. Dazu lernst du, wie man ein Programm mit gut ausgewählten Eingaben **testet**, und du lernst eine neue Form der Verzweigung kennen: die **Mehrfachauswahl**.

Lernziele

Nach diesem Abschnitt kannst du

- Fehler in Programmen mit Verzweigungen finden, mit Hilfe der Fehlermeldung erklären und beheben,
- ein Programm mit Verzweigungen lesen und seine Ausgabe vorhersagen, ohne es auszuführen,
- zu einem Programm sinnvolle Testfälle auswählen, auch für die Grenzfälle,
- eine Mehrfachauswahl als `CASE OF` beschreiben und in Python mit `match/case` umsetzen,
- ein Programm mit einer Mehrfachauswahl und einer verschachtelten Verzweigung schreiben.

1.4.1 Das brauchst du

Alles, was du für diesen Abschnitt brauchst, kennst du aus den Abschnitten über Operatoren und Verzweigungen. Hier ist es noch einmal kurz zusammengefasst. Der Aufbau einer mehrseitigen Verzweigung sieht so aus (`elif` und `else` sind dabei freiwillig):

```
if Bedingung_1:
    Anweisungen, wenn Bedingung_1 wahr ist
elif Bedingung_2:
    Anweisungen, wenn Bedingung_1 falsch und Bedingung_2 wahr ist
else:
    Anweisungen, wenn keine Bedingung wahr ist
```

Merke

- `if` / `elif` / `else` – nur der Block der **ersten zutreffenden** Bedingung wird ausgeführt.
- Achte auf **Doppelpunkt** `:` und korrekte **Einrückung**.
- `==` vergleicht, `=` weist zu.

Die häufigsten Fehler bei Verzweigungen und was Python dazu meldet:

Fehler im Code	Meldung von Python	Abhilfe
Doppelpunkt vergessen: `if zahl > 0`	`SyntaxError: expected ':'`	`:` ans Zeilenende
Block nicht eingerückt	`IndentationError: expected an indented block …`	Block 4 Leerzeichen einrücken
Zeile ohne Grund eingerückt	`IndentationError: unexpected indent`	Einrückung entfernen
`=` statt `==` in der Bedingung	`SyntaxError: invalid syntax. Maybe you meant '==' …`	`==` schreiben
Bedingung hinter `else`: `else zahl < 0:`	`SyntaxError: expected ':'`	`elif` statt `else`
`elif`-Bedingungen in falscher Reihenfolge	keine – aber falsches Ergebnis	Reihenfolge prüfen

1.4.2 Programme testen

Ein Programm, das ohne Fehlermeldung läuft, ist noch lange nicht richtig. Es kann trotzdem eine falsche Ausgabe liefern. Deshalb **testet** man es: Man führt es mit verschiedenen Eingaben aus und vergleicht die Ausgabe mit dem, was herauskommen **soll**.

Eine Eingabe zusammen mit der erwarteten Ausgabe nennt man einen **Testfall**. Wichtig ist, dass du die erwartete Ausgabe **vorher** überlegst – nicht erst, wenn du das Ergebnis siehst. Bei Programmen mit Verzweigungen wählst du die Testfälle so:

1. **Jeder Zweig mindestens einmal.** Für jeden Block der Verzweigung brauchst du eine Eingabe, bei der genau dieser Block ausgeführt wird.
2. **Die Grenzfälle.** Das sind die Werte direkt an einer Grenze. Bei der Bedingung `alter < 18` sind das 17 und 18. Hier passieren die meisten Fehler, zum Beispiel wenn `<` statt `<=` geschrieben wurde.

Ein Beispiel: Ein Programm soll ausgeben, ob eine Zahl positiv, negativ oder null ist. Sinnvolle Testfälle sind:

Eingabe	erwartete Ausgabe	Warum dieser Test?
`5`	`Die Zahl ist positiv`	Zweig „positiv“
`-3`	`Die Zahl ist negativ`	Zweig „negativ“
`0`	`Die Zahl ist null`	Zweig „null“, zugleich die Grenze zwischen positiv und negativ
`1` und `-1`	positiv bzw. negativ	direkt neben der Grenze

Genau dieses Programm reparierst du in der nächsten Aufgabe.

1.4.3 Fehler finden

Arbeitsauftrag 40: Fehler finden (Debugging)

Das folgende Programm soll ausgeben, ob eine eingegebene Zahl positiv, negativ oder null ist. Es enthält **drei Fehler**.

```
1 zahl = int(input("Zahl eingeben: "))
2 if zahl = 0
3     print("Die Zahl ist null")
4 elif zahl > 0:
5 print("Die Zahl ist positiv")
6 else:
7     print("Die Zahl ist negativ")
```

1. Finde die drei Fehler, **bevor** du das Programm ausführst. Notiere jeweils die Zeile und was falsch ist.
2. Schreibe die korrigierte Version auf.
3. Teste dein korrigiertes Programm mit den Testfällen aus dem Abschnitt „Programme testen“.

Tipp

Wenn du das fehlerhafte Programm ausführst, meldet Python immer nur den **ersten** Fehler, den es findet. Behebe ihn und starte das Programm erneut. Dann erscheint die Meldung zum nächsten Fehler. Die Tabelle im Abschnitt „Das brauchst du“ hilft dir, die Meldungen zu verstehen.

Die Fehler:

Gymnasium Ottweiler

1.4.4 Code lesen

Arbeitsauftrag 41: Code lesen und erklären

Lies das folgende Programm, **ohne** es auszuführen.

```
1 zahl = 17
2 if zahl % 15 == 0:
3     print("FizzBuzz")
4 elif zahl % 3 == 0:
5     print("Fizz")
6 elif zahl % 5 == 0:
7     print("Buzz")
8 else:
9     print(zahl)
```

1. Notiere, was das Programm für `zahl = 17` ausgibt. Begründe mit den Bedingungen.
2. Sage ebenso die Ausgabe für `zahl = 9`, `zahl = 10` und `zahl = 30` voraus.
3. Erkläre, warum die Bedingung `zahl % 15 == 0` **zuerst** geprüft wird.

Führe das Programm danach aus (ändere dazu den Wert in Zeile 1) und prüfe deine Vorhersagen.

Meine Vorhersagen und Begründungen:

Gymnasium Ottweiler

Arbeitsauftrag 42: Programmablaufplan zeichnen

Zeichne einen Programmablaufplan zum Programm aus dem Auftrag *Code lesen und erklären*. Verwende Rauten für die Bedingungen und Rechtecke für die Anweisungen. Beschrifte die Pfeile an den Rauten mit „ja“ und „nein“.

Tipp

Orientiere dich am Programmablaufplan des Notenrechners im Abschnitt über Verzweigungen. Jedes `if` und jedes `elif` wird zu einer Raute. Der „nein“-Pfeil führt jeweils zur nächsten Raute.

Mein Programmablaufplan:

1.4.5 Mehrfachauswahl: `CASE OF`

Oft wird **ein und derselbe Wert** mit mehreren festen Möglichkeiten verglichen. Ein Programm soll zum Beispiel eine Schulnote in Worten ausgeben. Mit `if` und `elif` sieht das so aus:

```
note = int(input("Note (1-6): "))
if note == 1:
    print("sehr gut")
elif note == 2:
    print("gut")
elif note == 3:
    print("befriedigend")
elif note == 4:
    print("ausreichend")
elif note == 5:
    print("mangelhaft")
elif note == 6:
    print("ungenügend")
else:
    print("Das ist keine gültige Note.")
```

AUSGABE
```
Note (1-6): 2
gut
```

Das funktioniert, ist aber umständlich: In jeder Zeile steht wieder `note == ...`. Für solche Fälle gibt es eine eigene Form der Verzweigung, die **Mehrfachauswahl**. Man beschreibt sie unabhängig von einer Programmiersprache oft so:

```
CASE note OF
    1: Ausgabe "sehr gut"
    2: Ausgabe "gut"
    ...
    6: Ausgabe "ungenügend"
    OTHERWISE: Ausgabe "Das ist keine gültige Note."
END CASE
```

Lies das so: „Im Fall, dass `note` den Wert 1 hat, gib ‚sehr gut' aus. Im Fall 2 …" `OTHERWISE` („andernfalls") ist der **Auffangfall**. Er greift, wenn keiner der genannten Werte passt.

Definition: Mehrfachauswahl

Bei einer **Mehrfachauswahl** (`CASE OF`) wird **ein Ausdruck** mit mehreren **festen Werten** verglichen. Ausgeführt wird der Fall, dessen Wert passt. Passt keiner, wird der Auffangfall ausgeführt.

Die Mehrfachauswahl in Python: `match` und `case`

In Python schreibt man die Mehrfachauswahl mit den Schlüsselwörtern `match` („vergleiche") und `case` („Fall"):

```
1  note = int(input("Note (1-6): "))
2  match note:
3      case 1:
4          print("sehr gut")
5      case 2:
6          print("gut")
7      case 3:
8          print("befriedigend")
9      case 4:
10         print("ausreichend")
11     case 5:
12         print("mangelhaft")
13     case 6:
14         print("ungenügend")
15     case _:
16         print("Das ist keine gültige Note.")
```

AUSGABE
```
Note (1-6): 2
gut
```

- **Zeile 2:** `match note:` legt fest, **welcher Wert** verglichen wird. Wie bei `if` steht am Ende ein Doppelpunkt, und alles Weitere ist eingerückt.
- **Zeilen 3–14:** Jedes `case` nennt einen festen Wert, dahinter einen Doppelpunkt. Python geht die Fälle von oben nach unten durch. Beim ersten passenden Fall wird der eingerückte Block darunter ausgeführt, die übrigen Fälle werden übersprungen. Hier passt `case 2`.
- **Zeilen 15–16:** `case _` ist der **Auffangfall**. Der Unterstrich `_` passt zu jedem Wert. Dieser Fall entspricht dem `else` bzw. dem `OTHERWISE`. Er muss als **letzter** Fall stehen.

Die Mehrfachauswahl funktioniert genauso mit Texten. Das nutzt du gleich für einen Taschenrechner:

```
operator = "-"
match operator:
    case "+":
        print("Addition")
    case "-":
        print("Subtraktion")
    case _:
        print("Unbekannter Operator")
```

AUSGABE

```
Subtraktion
```

Merke

- `CASE OF` eignet sich, wenn **ein Ausdruck mit mehreren festen Werten** verglichen wird. In Python heißt diese Verzweigung `match`/`case`.
- `case _` ist der Auffangfall und entspricht hier `else`. Für Bereiche und frei formulierte Bedingungen ist `if` meist geeigneter.

Wann nimmst du was? Die Mehrfachauswahl passt nur, wenn ein Wert mit **festen** Werten verglichen wird: Noten 1 bis 6, Operatoren `+ - * /`, Menüpunkte `"a"`, `"b"`, `"c"`. Für **Bereiche** wie „unter 6 Jahren“ oder „zwischen 6 und 17“ und für Bedingungen mit `and` und `or` bleibt `if`/`elif`/`else` die richtige Wahl.

Stolperstein

`match` und `case` gibt es erst seit **Python 3.10** (erschienen 2021). Mit einer älteren Version meldet Python bei `match note:` einen `SyntaxError`. Thonny und die meisten aktuellen Umgebungen bringen eine neuere Version mit. Welche Version du hast, zeigt dir das Programm `import sys` und `print(sys.version)` – oder du startest einfach das Beispiel oben.

Ein zweiter häufiger Fehler: `case _` steht nicht am Ende. Dann passt er zu allem, und die Fälle darunter werden nie erreicht. Python meldet in diesem Fall einen `SyntaxError`.

1.4.6 Ein Programm erweitern

Im Abschnitt über Variablen hast du im Auftrag *Einfacher Rechner* ein Programm geschrieben, das zwei Zahlen einliest und mit ihnen rechnet. Eine kurze Version davon sieht so aus:

```
a = float(input("Erste Zahl: "))
b = float(input("Zweite Zahl: "))
print("Summe:", a + b)
```

AUSGABE

```
Erste Zahl: 7
Zweite Zahl: 2
Summe: 9.0
```

Dieses Programm rechnet immer plus. Mit einer Mehrfachauswahl kann die Nutzerin selbst wählen, welche Rechnung ausgeführt wird.

Arbeitsauftrag 43: Taschenrechner mit CASE OF

Schreibe einen Taschenrechner, der zwei Zahlen sowie einen Operator (`+`, `-`, `*`, `/`) abfragt. Setze die Mehrfachauswahl in Python mit `match`/`case` um. Fange den Fall ab, dass durch 0 geteilt werden soll, und gib bei einem unbekannten Operator eine verständliche Meldung aus. Teste dein Programm mit allen vier Operatoren, mit einer Division durch 0 und mit einem unbekannten Operator.

So könnte es aussehen:

AUSGABE

```
Erste Zahl: 7
Operator (+ - * /): /
Zweite Zahl: 2
Ergebnis: 3.5
```

Tipp

Gehe in drei Schritten vor:

1. Lies die erste Zahl, den Operator und die zweite Zahl ein. Der Operator ist ein Text und wird **nicht** umgewandelt.
2. Unterscheide die Operatoren mit `match op:` und je einem `case "+":`, `case "-":` und so weiter. `case _` übernimmt die Rolle des Auffangfalls für alle unbekannten Operatoren.
3. Im Fall `"/"` brauchst du zusätzlich eine **verschachtelte** Verzweigung mit `if`: Nur wenn die zweite Zahl ungleich 0 ist, darf geteilt werden.

Ohne diese Prüfung bricht das Programm bei einer Division durch 0 mit einem `ZeroDivisionError` (Division durch null) ab.

Arbeitsauftrag 44 (Zusatz): Eintrittspreis

Im Freibad gelten diese Preise:

- Kinder unter 6 Jahren: Eintritt frei
- 6 bis 17 Jahre: 2,50 Euro
- ab 18 Jahren: 4,00 Euro, mit Schüler- oder Studierendenausweis 3,00 Euro

Schreibe ein Programm, das zuerst das Alter abfragt. **Nur** bei Personen ab 18 Jahren soll es zusätzlich fragen, ob ein Ausweis vorliegt (`ja`/`nein`). Dann gibt es den Eintrittspreis aus. Lege vorher eine Tabelle mit Testfällen an, die alle Zweige und alle Grenzfälle abdeckt, und teste dein Programm damit.

Meine Testfälle (Eingabe, erwartete Ausgabe):

1.4.7 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 45: Welcher Tag? ●○○

Lies das Programm, **ohne** es auszuführen. Gib an, was es für `tag = 3` ausgibt. Gib ebenso die Ausgabe für `tag = 7` und für `tag = 1` an. Prüfe danach am Computer, indem du den Wert in Zeile 1 änderst.

```
1  tag = 3
2  match tag:
3      case 1:
4          print("Montag")
5      case 2:
6          print("Dienstag")
7      case 3:
8          print("Mittwoch")
9      case _:
10         print("anderer Tag")
```

Meine Vorhersagen:

Arbeitsauftrag 46: Ampel mit match ●○○

Schreibe das folgende Programm so um, dass es statt `if/elif/else` eine Mehrfachauswahl mit `match` und `case` verwendet. Das Programm soll danach genau dasselbe ausgeben. Teste mit `rot`, `grün` und `blau`.

```
1  farbe = input("Ampelfarbe: ")
2  if farbe == "rot":
3      print("Stehen bleiben!")
4  elif farbe == "gelb":
5      print("Warten!")
6  elif farbe == "grün":
7      print("Gehen!")
8  else:
9      print("Diese Farbe kennt die Ampel nicht.")
```

Arbeitsauftrag 47: Wochentage ●○○

Das Programm soll zu einem Kürzel von `Mo` bis `Fr` den ganzen Wochentag ausgeben. Bei allen anderen Eingaben soll es „Das ist kein Schultag.“ ausgeben. Ergänze die Lücken `___` und die fehlenden Fälle für `Mi`, `Do` und `Fr`.

```
kuerzel = input("Kürzel (Mo bis Fr): ")
match ___:
    case "Mo":
        print("Montag")
    case ___:
        print("Dienstag")
    # Hier fehlen die Fälle für Mi, Do und Fr.
    ___:
        print("Das ist kein Schultag.")
```

Arbeitsauftrag 48: Testfälle für ein Spiel ●○○

Ein Spiel teilt Spielerinnen und Spieler nach ihrem Level ein:

```
level = int(input("Level: "))
if level < 1:
    print("ungültiges Level")
elif level <= 10:
    print("Anfänger")
elif level <= 30:
    print("Fortgeschritten")
else:
    print("Profi")
```

Lege eine Tabelle mit Testfällen an: Jeder Zweig soll mindestens einmal vorkommen, außerdem alle Grenzfälle. Trage zu jedem Testfall die **erwartete** Ausgabe ein, **bevor** du das Programm ausführst. Prüfe dann am Computer.

Eingabe	erwartete Ausgabe	Warum dieser Test?

Arbeitsauftrag 49: Fehler finden: Pizzagröße ●●○

Das Programm soll zu einer Pizzagröße (`S`, `M` oder `L`) den Preis ausgeben. Es enthält **drei Fehler**.

```
groesse = input("Größe (S, M, L): ")
match groesse
    case "S":
        print("Preis: 6,50 Euro")
    case "M":
    print("Preis: 8,00 Euro")
    case "L":
        print("Preis: 9,50 Euro")
    else:
        print("Diese Größe gibt es nicht.")
```

1. Finde die drei Fehler. Notiere jeweils die Zeile und was falsch ist.
2. Schreibe das Programm richtig auf und teste es mit `M` und mit `XL`.

Die Fehler:

Arbeitsauftrag 50: match oder if? ●●○

Entscheide für jede Situation, ob eine Mehrfachauswahl mit `match`/`case` oder eine Verzweigung mit `if`/`elif`/`else` besser passt. Begründe kurz.

Situation	`match` oder `if`?
a) Eine Schulnote von 1 bis 6 in Worten ausgeben	
b) Aus einer Temperatur eine Warnstufe bestimmen (unter 0, 0 bis 30, über 30)	
c) Ein Menü mit den Punkten `a`, `b` und `c`	
d) Prüfen, ob jemand mindestens 16 ist **und** eine Einverständniserklärung hat	
e) Zu einer Monatsnummer von 1 bis 12 den Namen des Monats ausgeben	

Begründungen zu a) bis e):

Arbeitsauftrag 51: Längen umrechnen ●●○

Schreibe ein Programm, das eine Länge (Kommazahl) und eine Einheit (`km`, `m` oder `cm`) einliest und die Länge in Metern ausgibt. Verwende eine Mehrfachauswahl. Bei einer unbekannten Einheit soll eine verständliche Meldung erscheinen. Teste mit `2.5` und `km`, mit `150` und `cm` und mit `3` und `mm`.

So könnte es aussehen:

AUSGABE

```
Länge: 2.5
Einheit (km, m, cm): km
Das sind 2500.0 Meter.
```

Arbeitsauftrag 52: Taschenrechner erweitern ●●●

Erweitere deinen Taschenrechner aus dem Auftrag *Taschenrechner mit CASE OF* um drei weitere Operatoren: `//` (Ganzzahldivision), `%` (Rest) und `**` (Potenz). Auch bei `//` und `%` darf die zweite Zahl nicht 0

sein. Lege eine Tabelle mit Testfällen an, in der jeder neue Operator und jeder Sonderfall mindestens einmal vorkommt, und teste dein Programm damit.

Meine Testfälle (Eingaben, erwartete Ausgabe):

1.4.8 Das Wichtigste in Kürze

- Bei `if` / `elif` / `else` wird nur der Block der ersten zutreffenden Bedingung ausgeführt. Deshalb kommt es auf die Reihenfolge an: der speziellste Fall zuerst.
- Python meldet immer nur den **ersten** Fehler. Behebe Fehler einzeln, von oben nach unten, und starte das Programm danach jeweils neu.
- Ein **Testfall** besteht aus einer Eingabe und der erwarteten Ausgabe. Teste jeden Zweig mindestens einmal und alle **Grenzfälle**.
- Beim Code-Lesen hilft ein Ablaufprotokoll: Welche Bedingung ist wahr, welche Zeile wird als Nächstes ausgeführt?
- Wird **ein Wert** mit mehreren **festen** Werten verglichen, eignet sich die **Mehrfachauswahl** (`CASE OF`). In Python: `match wert:` mit `case …:` für jeden Fall und `case _:` als Auffangfall.
- Mit einer verschachtelten Verzweigung fängt man Sonderfälle ab, zum Beispiel die Division durch 0.

Selbstcheck

- ☐ Ich kann die Fehlermeldungen `expected ':'`, `IndentationError` und `invalid syntax … '=='` erklären und den Fehler beheben.
- ☐ Ich kann die Ausgabe eines Programms mit `if` / `elif` / `else` vorhersagen, ohne es auszuführen.
- ☐ Ich kann erklären, warum bei `elif` die Reihenfolge der Bedingungen wichtig ist.
- ☐ Ich kann zu einem Programm Testfälle für alle Zweige und alle Grenzfälle angeben.
- ☐ Ich kann eine Mehrfachauswahl als `CASE OF` beschreiben und in Python mit `match`/`case` umsetzen.
- ☐ Ich kann einen Taschenrechner mit `match`/`case` schreiben, der eine Division durch 0 abfängt.