1.3 Verzweigungen: Programme treffen Entscheidungen

Leitidee

Jeden Tag triffst du Entscheidungen nach einfachen Regeln: *Wenn* es regnet, *dann* nehme ich einen Schirm mit. *Wenn* der Bus kommt, steige ich ein, *sonst* warte ich. Auch Programme müssen entscheiden: Ist das Passwort richtig? Hat die Spielerin genug Punkte? Ist die Zahl positiv? Bisher hat Python jede Zeile deines Programms ausgeführt, immer von oben nach unten. Jetzt lernst du, wie ein Programm abhängig von einer **Bedingung** nur bestimmte Anweisungen ausführt. Aus der Blockprogrammierung kennst du dafür die Bausteine „falls … dann“ und „falls … dann … sonst“.

Lernziele

Nach diesem Abschnitt kannst du

- erklären, woraus eine Verzweigung besteht, und die Fachbegriffe *Bedingung*, *dann-Teil* und *sonst-Teil* verwenden,
- ein-, zwei- und mehrseitige Verzweigungen mit `if`, `else` und `elif` schreiben,
- mit der Einrückung festlegen, welche Anweisungen zu einem Block gehören, und Einrückungsfehler beheben,
- den Ablauf eines Programms mit Verzweigung in einem Programmablaufplan und einem Ablaufprotokoll darstellen,
- Bedingungen mit `and` und `or` formulieren und Verzweigungen verschachteln.

1.3.1 Die einseitige Verzweigung: `if`

Stell dir eine Weiche bei der Eisenbahn vor. Je nachdem, wie sie gestellt ist, fährt der Zug geradeaus oder biegt ab. In einem Programm übernimmt eine **Bedingung** die Rolle der Weiche. Eine Bedingung ist eine Frage, deren Antwort `True` oder `False` ist – genau wie die Vergleiche aus dem Abschnitt über Operatoren.

```
1 temperatur = 30
2 if temperatur > 25:
3     print("Es ist heiß. Trink genug Wasser!")
4 print("Schönen Tag!")
```

AUSGABE

```
Es ist heiß. Trink genug Wasser!
Schönen Tag!
```

- **Zeile 1:** Die Variable `temperatur` bekommt den Wert `30`.
- **Zeile 2:** `if` ist englisch und heißt „wenn“ oder „falls“. Dahinter steht die Bedingung `temperatur > 25`. Python prüft sie: 30 ist größer als 25, die Bedingung ist also `True`. Am Ende der Zeile steht ein **Doppelpunkt**.
- **Zeile 3:** Diese Zeile ist **eingerückt**, sie beginnt mit vier Leerzeichen. Dadurch gehört sie zur `if`-Anweisung. Sie wird nur ausgeführt, weil die Bedingung wahr ist.
- **Zeile 4:** Diese Zeile ist nicht eingerückt. Sie gehört nicht mehr zur `if`-Anweisung und wird in jedem Fall ausgeführt.

Was passiert, wenn die Bedingung falsch ist? Überlege zuerst, dann lies weiter.

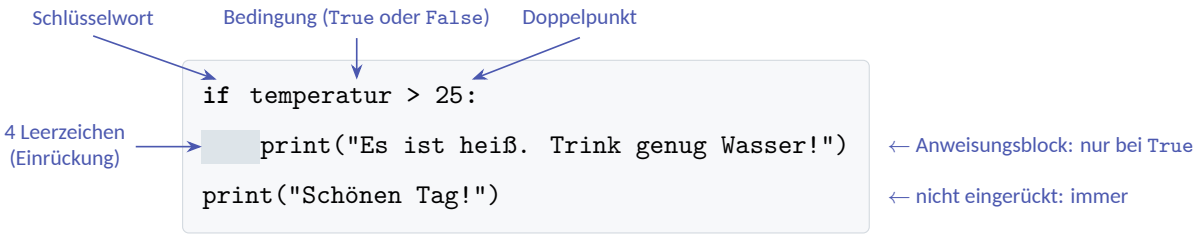
```
1 temperatur = 18
2 if temperatur > 25:
3     print("Es ist heiß. Trink genug Wasser!")
4 print("Schönen Tag!")
```

AUSGABE

```
Schönen Tag!
```

Jetzt ist `temperatur > 25` falsch (`False`). Python **überspringt** die eingerückte Zeile 3 und macht direkt mit Zeile 4 weiter.

Die `if`-Anweisung steuert also, welche Zeilen ausgeführt werden. Anweisungen, die so den Ablauf eines Programms steuern, nennt man **Kontrollstrukturen**.



Aufbau einer einseitigen Verzweigung in Python. Nur der eingerückte Anweisungsblock hängt von der Bedingung ab

Definition: Bedingung

Eine **Bedingung** ist ein Ausdruck, dessen Wert entweder wahr (`True`) oder falsch (`False`) ist, zum Beispiel `temperatur > 25`.

Definition: Verzweigung

Eine **Verzweigung** ist eine Kontrollstruktur, die abhängig von einer Bedingung festlegt, welche Anweisungen ausgeführt werden. Man nennt sie auch *bedingte Anweisung*.

Gymnasium Ottweiler

Merke

- Eine **Verzweigung** (bedingte Anweisung) besteht aus einer Bedingung und einem oder mehreren Anweisungsblöcken.
- **Einseitige Verzweigung:** `if Bedingung:` – der eingerückte Block wird nur ausgeführt, wenn die Bedingung `True` ist.

1.3.2 Einrückung: Was gehört zum Block?

Ein Block kann aus mehreren Zeilen bestehen. Woher weiß Python, wo der Block aufhört? An der **Einrückung**. Alle Zeilen direkt unter dem `if`, die gleich weit eingerückt sind, gehören zum Block. Die erste Zeile, die wieder ganz links beginnt, gehört nicht mehr dazu.

Vergleiche die beiden Programme. Sie unterscheiden sich nur in der Einrückung von Zeile 4:

```
1 punkte = 3
2 if punkte >= 10:
3     print("Gewonnen!")
4     print("Du bekommst einen Preis.")
5 print("Spiel beendet.")
```

AUSGABE

```
Spiel beendet.
```

```
1 punkte = 3
2 if punkte >= 10:
3     print("Gewonnen!")
4 print("Du bekommst einen Preis.")
5 print("Spiel beendet.")
```

AUSGABE

```
Du bekommst einen Preis.
Spiel beendet.
```

In beiden Programmen ist die Bedingung `3 >= 10` falsch. Im ersten Programm gehören die Zeilen 3 und 4 zum Block und werden beide übersprungen. Im zweiten Programm gehört nur Zeile 3 zum Block. Zeile 4 wird immer ausgeführt – und das Programm verteilt einen Preis, obwohl niemand gewonnen hat.

Merke

In Python bestimmt die **Einrückung** (4 Leerzeichen), welche Anweisungen zu einem Block gehören. Die Zeile mit der Bedingung endet mit einem **Doppelpunkt** `:`.

Stolperstein

Vergisst du den Doppelpunkt, meldet Python `expected ':'` („Doppelpunkt erwartet“). Das Zeichen `^` zeigt genau auf die Stelle, an der er fehlt:

```
1 zahl = 5
2 if zahl > 0
3     print("positiv")
```

FEHLERMELDUNG

```
  File "programm.py", line 2
    if zahl > 0
               ^
SyntaxError: expected ':'
```

Tipp

Die meisten Programmierumgebungen rücken nach einem Doppelpunkt automatisch ein, wenn du die Eingabetaste drückst. Mit der Tabulatortaste (Tab) rückst du selbst ein. Achte darauf, dass alle Zeilen eines Blocks **genau gleich weit** eingerückt sind.

Stolperstein

Fehlt die Einrückung nach dem Doppelpunkt ganz, bricht Python mit einem **IndentationError** ab (*indentation* heißt „Einrückung“):

```
1 zahl = 5
2 if zahl > 0:
3 print("Die Zahl ist positiv.")
```

FEHLERMELDUNG

```
  File "programm.py", line 3
    print("Die Zahl ist positiv.")
    ^^^^^
IndentationError: expected an indented block after 'if' statement on line 2
```

Übersetzt: „Nach der `if`-Anweisung in Zeile 2 wurde ein eingerückter Block erwartet.“ Python zeigt auf Zeile 3, denn dort hätte der Block beginnen müssen. Abhilfe: Zeile 3 um vier Leerzeichen einrücken.

Auch zu viel Einrückung ist ein Fehler. Eine Zeile darf nur eingerückt sein, wenn sie zu einem Block gehört, der mit einer Zeile mit Doppelpunkt beginnt:

```
zahl = 5
    print(zahl)
```

FEHLERMELDUNG

```
IndentationError: unexpected indent
```

Unexpected indent heißt „unerwartete Einrückung“. Prüfe die genannte Zeile und rücke sie richtig ein.

1.3.3 Die zweiseitige Verzweigung: `if` … `else`

Oft soll ein Programm in **beiden** Fällen etwas tun: wenn die Bedingung wahr ist, das eine, und sonst das andere. Dafür gibt es `else` (englisch „sonst“):

```
1 zahl = int(input("Gib eine Zahl ein: "))
2 if zahl > 4:
3     print("Die Zahl ist größer als 4.")
4 else:
5     print("Die Zahl ist nicht größer als 4.")
6 print("Fertig.")
```

AUSGABE

```
Gib eine Zahl ein: 7
Die Zahl ist größer als 4.
Fertig.
```

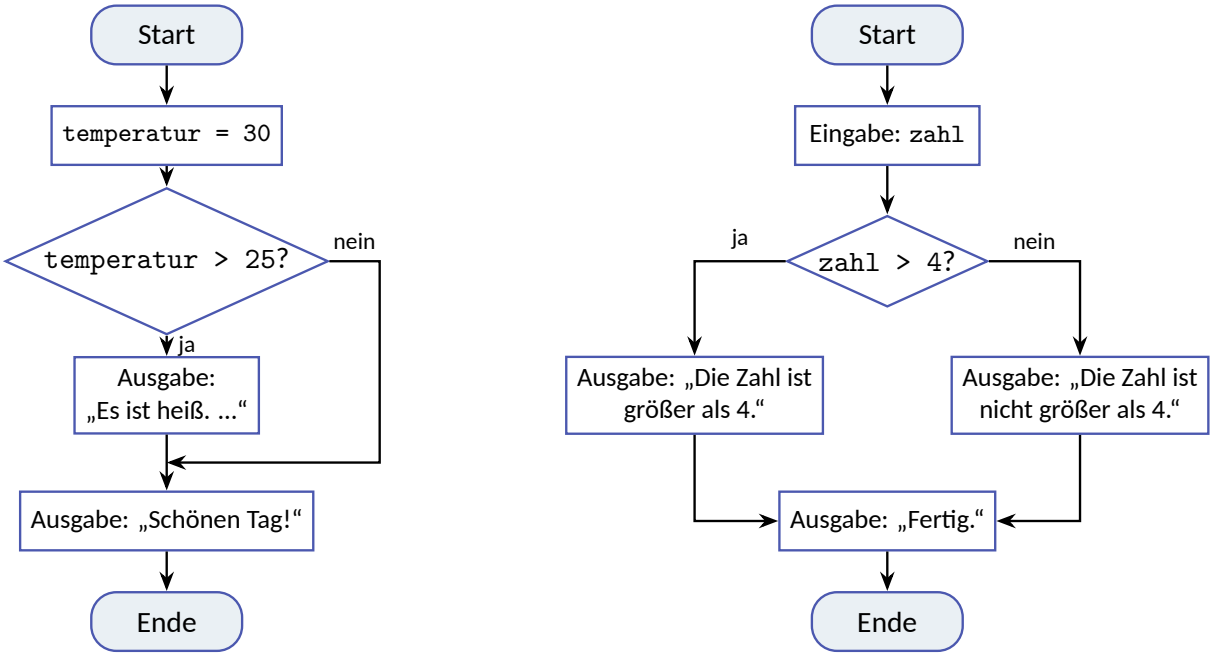
- **Zeile 1:** Die Eingabe wird in eine ganze Zahl umgewandelt und in `zahl` gespeichert.
- **Zeile 2:** Die Bedingung `zahl > 4` wird geprüft. Für 7 ist sie wahr.
- **Zeile 3:** Dieser eingerückte Block ist der **dann-Teil**. Er wird ausgeführt, wenn die Bedingung wahr ist.
- **Zeile 4:** `else:` steht genau unter dem `if` (gleiche Einrückung) und hat **keine** eigene Bedingung, aber einen Doppelpunkt. Schreibst du trotzdem eine Bedingung dahinter, etwa `else zahl < 4:`, meldet Python `SyntaxError: expected ':'`.
- **Zeile 5:** Der **sonst-Teil**. Er wird nur ausgeführt, wenn die Bedingung falsch ist. Bei der Eingabe 7 wird er übersprungen.
- **Zeile 6:** Nicht eingerückt, wird also in jedem Fall ausgeführt.

Es wird also **immer genau einer** der beiden Teile ausgeführt, nie beide und nie keiner. Weil das Programm zwischen zwei Möglichkeiten (Alternativen) wählt, nennt man die zweiseitige Verzweigung auch **Alternativanweisung**.

Den Ablauf darstellen

Wie ein Programm mit Verzweigung abläuft, kann man in einem **Programmablaufplan** zeichnen. Rauten stehen für Bedingungen, Rechtecke für Anweisungen. Die Pfeile zeigen, in welcher Reihenfolge Python die Anweisungen ausführt. An einer Raute teilt sich der Weg in „ja“ (Bedingung wahr) und „nein“ (Bedingung falsch).

Gymnasium Ottweiler



Programmablaufpläne: links die einseitige Verzweigung aus dem Temperatur-Beispiel, rechts die zweiseitige Verzweigung aus dem Programm oben

Ein **Ablaufprotokoll** zeigt, welche Zeilen bei einer bestimmten Eingabe wirklich ausgeführt werden. Hier für das Programm oben, links mit der Eingabe 7, rechts mit der Eingabe 2:

Schritt	Zeile	`zahl`
1	1	7
2	2	7
3	3	7
4	6	7

Schritt	Zeile	`zahl`
1	1	2
2	2	2
3	5	2
4	6	2

Bei der Eingabe 7 ist die Bedingung in Zeile 2 wahr, also geht es mit Zeile 3 weiter. Bei der Eingabe 2 ist sie falsch, also springt Python zu Zeile 5. Die Zeile `else:` selbst tut nichts. Sie zeigt Python nur, wo der sonst-Teil beginnt. Deshalb taucht Zeile 4 im Protokoll nicht auf.

Merke

Zweiseitige Verzweigung: `if … else:` – entweder der **dann-Teil** oder der **sonst-Teil** wird ausgeführt.

So kannst du es sagen

So beschreibst du eine Verzweigung in Fachsprache – vom einfachen zum genauen Satz:

- *„Ich bekomme raus, ob die Zahl größer als 4 ist.“*
- *„Die Anweisung gibt ‚Die Zahl ist größer als 4.‘ aus, wenn eine Zahl größer als 4 ist, und ‚Die Zahl ist nicht größer als 4.‘, wenn sie es nicht ist.“*
- *„Die Alternativanweisung prüft, ob eine Zahl größer als 4 ist. Ist dies der Fall, wird ‚Die Zahl ist größer als 4.‘ ausgegeben und sonst ‚Die Zahl ist nicht größer als 4.‘ “*

Nützliche Wendungen: *Die Verzweigung besteht aus … · Es wird geprüft, ob … · Ist die Bedingung erfüllt, wird der dann-Teil ausgeführt, sonst der sonst-Teil.*

Arbeitsauftrag 28: Glatteis-Warnung

Lies das Programm. Beschreibe in zwei bis drei Sätzen mit Fachbegriffen, was es tut. Gib dann an, was es bei den Eingaben `-3`, `0` und `5` ausgibt.

```
1 temperatur = int(input("Temperatur in Grad Celsius: "))
2 if temperatur <= 0:
3     print("Achtung, Glatteisgefahr!")
4 else:
5     print("Keine Glatteisgefahr.")
```

Gymnasium Ottweiler

Beschreibung und Ausgaben:

1.3.4 Die mehrseitige Verzweigung: `if` … `elif` … `else`

Manchmal gibt es mehr als zwei Möglichkeiten. Ein Programm soll aus einer Punktzahl von 0 bis 100 eine Note machen. Dafür gibt es `elif`. Das ist eine Abkürzung für *else if*, also „sonst, wenn“:

```
1  note = int(input("Punktzahl (0-100) eingeben: "))
2
3  if note >= 90:
4      print("Note: sehr gut")
5  elif note >= 75:
6      print("Note: gut")
7  elif note >= 50:
8      print("Note: befriedigend")
9  else:
10     print("Note: nicht bestanden")
```

AUSGABE

```
Punktzahl (0-100) eingeben: 80
Note: gut
```

- **Zeile 1:** Die Punktzahl wird eingelesen und in `note` gespeichert.
- **Zeile 3:** Die erste Bedingung wird geprüft: Ist 80 größer oder gleich 90? Nein.
- **Zeile 5:** Weil die erste Bedingung falsch war, wird die nächste geprüft: Ist 80 größer oder gleich 75? Ja.
- **Zeile 6:** Der zugehörige Block wird ausgeführt: „Note: gut“.
- **Zeilen 7–10:** Werden übersprungen. Sobald eine Bedingung zutrifft, prüft Python die weiteren **gar nicht mehr**, obwohl auch `80 >= 50` wahr wäre.

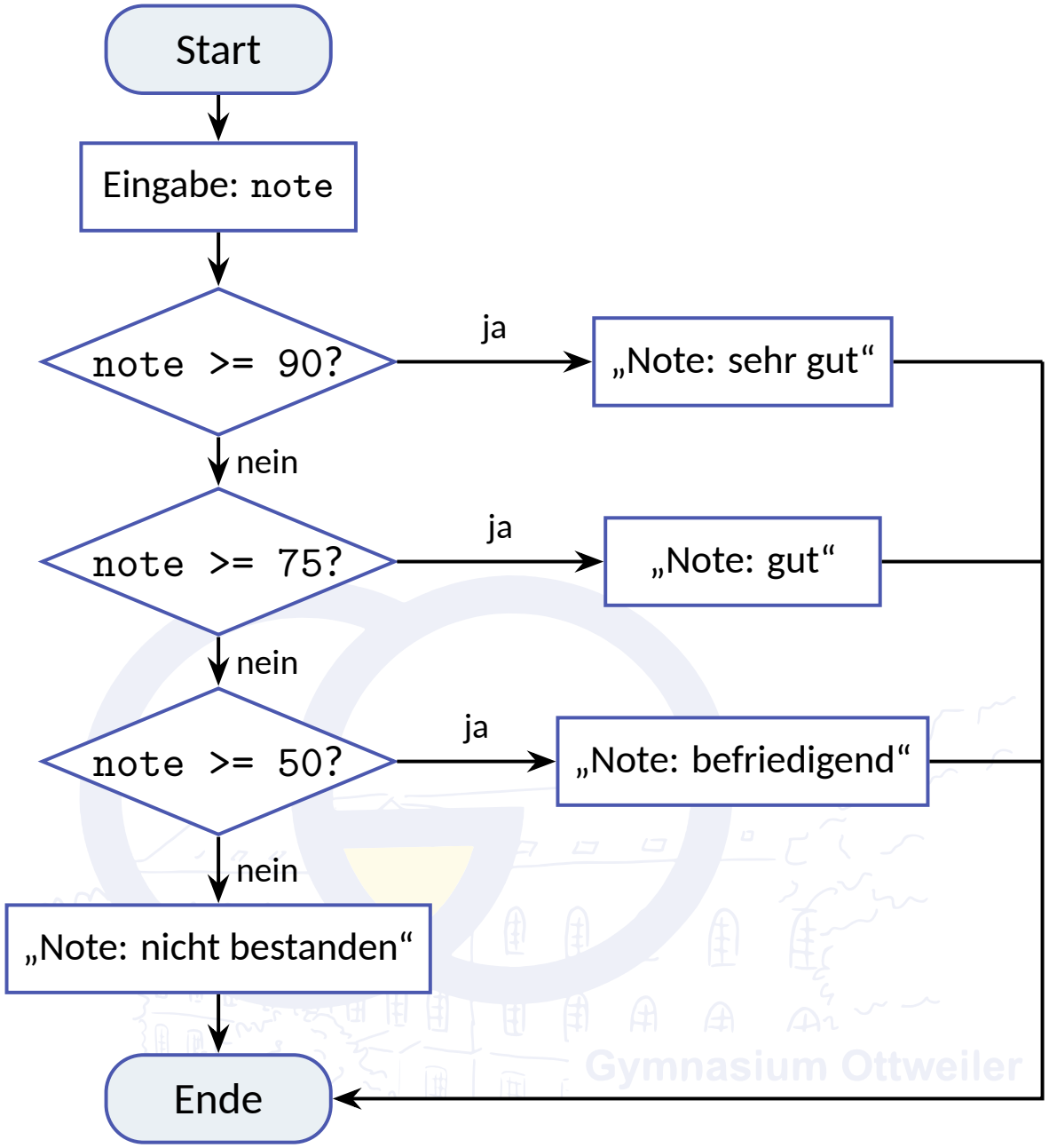
Der `else`-Teil am Ende fängt alle Fälle auf, in denen **keine** der Bedingungen wahr war. Er darf auch fehlen. Dann passiert in diesem Fall einfach nichts.

Das Ablaufprotokoll zeigt, welche Zeilen ausgeführt werden. Links für die Eingabe 80, rechts für die Eingabe 30:

Schritt	Zeile	note
1	1	80
2	3	80
3	5	80
4	6	80

Schritt	Zeile	note
1	1	30
2	3	30
3	5	30
4	7	30
5	10	30

Bei der Eingabe 30 sind alle drei Bedingungen falsch (Zeilen 3, 5 und 7). Deshalb wird der `else`-Teil in Zeile 10 ausgeführt.



Programmablaufplan des Notenrechners: Die Bedingungen werden von oben nach unten geprüft. Die erste, die zutrifft, entscheidet

Merke

Mehrseitige Verzweigung: `if … elif … else:` – es wird nur der Block der **ersten** zutreffenden Bedingung ausgeführt. Trifft keine zu, wird der `else`-Block ausgeführt.

Stolperstein

Bei `elif` kommt es auf die **Reihenfolge** an. Hier stehen die Bedingungen in der falschen Reihenfolge:

```
1 note = int(input("Punktzahl (0-100) eingeben: "))
2 if note >= 50:
3     print("Note: befriedigend")
4 elif note >= 75:
5     print("Note: gut")
6 elif note >= 90:
7     print("Note: sehr gut")
```

AUSGABE

```
Punktzahl (0-100) eingeben: 95
Note: befriedigend
```

95 Punkte, aber nur „befriedigend“! Die erste Bedingung `95 >= 50` ist wahr, also wird ihr Block ausgeführt und der Rest übersprungen. Die Fälle „sehr gut“ und „gut“ werden nie erreicht. Regel: Bei `>=` die **größte** Grenze zuerst prüfen, bei `<` die **kleinste**.

Arbeitsauftrag 29: Größte von zwei Zahlen

Schreibe ein Programm, das zwei Zahlen einliest und ausgibt, welche die größere ist – oder ob beide gleich sind. Teste dein Programm mit mindestens drei Eingabepaaren: einmal ist die erste Zahl größer, einmal die zweite, einmal sind beide gleich.

Arbeitsauftrag 30: Altersklassifikation

Schreibe ein Programm, das ein Alter einliest und es so einordnet:

- unter 6: „Kleinkind“
- 6 bis unter 13: „Kind“
- 13 bis unter 18: „Jugendlicher“
- ab 18: „Erwachsener“

Teste dein Programm besonders mit den **Grenzwerten** 5, 6, 12, 13, 17 und 18.

Gymnasium Ottweiler

1.3.5 Bedingungen mit `and` und `or`

Eine Bedingung darf alles sein, was `True` oder `False` ergibt – also auch Verknüpfungen mit `and`, `or` und `not` aus dem Abschnitt über Operatoren:

```
1 tag = input("Welcher Tag ist heute? ")
2 if tag == "Samstag" or tag == "Sonntag":
3     print("Wochenende! Ausschlafen.")
4 else:
5     print("Wecker stellen.")
```

AUSGABE

```
Welcher Tag ist heute? Sonntag
Wochenende! Ausschlafen.
```

- **Zeile 1:** Der Tag wird als Text eingelesen. Hier wird nicht umgewandelt, denn es soll ein Text verglichen werden.

- **Zeile 2:** Die Bedingung ist wahr, wenn **mindestens einer** der beiden Vergleiche wahr ist. `tag == "Samstag"` ist falsch, `tag == "Sonntag"` ist wahr. Also ist die ganze Bedingung wahr.

Stolperstein

Ein häufiger Fehler ist, den zweiten Vergleich abzukürzen: `if tag == "Samstag" or "Sonntag":`. Das sieht aus wie Deutsch, bedeutet in Python aber etwas anderes. Python liest `"Sonntag"` als eigene Bedingung, und ein Text, der nicht leer ist, zählt als wahr. Die Bedingung ist deshalb **immer** erfüllt:

```
1 tag = input("Welcher Tag ist heute? ")
2 if tag == "Samstag" or "Sonntag":
3     print("Wochenende! Ausschlafen.")
```

AUSGABE

```
Welcher Tag ist heute? Montag
Wochenende! Ausschlafen.
```

Schreibe auf **beiden** Seiten von `or` und `and` einen vollständigen Vergleich.

Stolperstein

Schreibst du in einer Bedingung `=` statt `==`, meldet Python einen Syntaxfehler und macht gleich einen Vorschlag:

```
1 zahl = 0
2 if zahl = 0:
3     print("Die Zahl ist null.")
```

FEHLERMELDUNG

```
   File "programm.py", line 2
     if zahl = 0:
        ^^^^^^^^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

„Meintest du vielleicht `==` …?“ – genau. Den zweiten Vorschlag `:=` brauchst du nicht.

1.3.6 Verschachtelte Verzweigungen

Im Block einer Verzweigung dürfen beliebige Anweisungen stehen – auch eine weitere Verzweigung. Man spricht dann von einer **verschachtelten** Verzweigung. Ein Beispiel ist eine Anmeldung: Nur wenn der Benutzername bekannt ist, wird überhaupt nach dem Passwort gefragt.

```
1 benutzer = input("Benutzername: ")
2 if benutzer == "alex":
3     passwort = input("Passwort: ")
4     if passwort == "geheim123":
5         print("Willkommen, Alex!")
6     else:
7         print("Falsches Passwort.")
8 else:
9     print("Unbekannter Benutzer.")
```

AUSGABE

```
Benutzername: alex
Passwort: geheim123
Willkommen, Alex!
```

- **Zeile 2:** Die äußere Verzweigung prüft den Benutzernamen.
- **Zeilen 3–7:** Das ist der dann-Teil der äußeren Verzweigung. Alle diese Zeilen sind mindestens vier Leerzeichen eingerückt.
- **Zeilen 4–7:** Darin steht eine zweite, **innere** Verzweigung. Ihre Blöcke (Zeilen 5 und 7) sind noch einmal vier Leerzeichen weiter eingerückt, also acht Leerzeichen.
- **Zeilen 8–9:** Das `else` steht genau unter dem äußeren `if`. Es gehört deshalb zur äußeren Verzweigung.

An der Einrückung erkennst du, welches `else` zu welchem `if` gehört: Es steht immer genau darunter. Das Ablaufprotokoll für die Eingaben `alex` und `1234` zeigt, dass Python die innere Verzweigung erreicht, dort aber den sonst-Teil ausführt:

Schritt	Zeile	`benutzer`	`passwort`
1	1	`"alex"`	–
2	2	`"alex"`	–
3	3	`"alex"`	`"1234"`
4	4	`"alex"`	`"1234"`
5	7	`"alex"`	`"1234"`

Der Strich – bedeutet: Die Variable `passwort` gibt es zu diesem Zeitpunkt noch nicht.

Merke

Verzweigungen können **verschachtelt** werden: Eine `if`-Anweisung steht im Block einer anderen. Die Einrückung zeigt, welcher Block zu welcher Bedingung gehört.

Arbeitsauftrag 31 (Zusatz): Achterbahn-Check

Ein Freizeitpark hat folgende Regeln für eine Achterbahn:

- Mindestgröße: 140 cm.
- Ist man kleiner als 140 cm, aber mindestens 120 cm, darf man nur in Begleitung eines Erwachsenen mitfahren.
- Ist man kleiner als 120 cm, darf man nicht mitfahren.

Schreibe ein Programm, das die Körpergröße (in cm) abfragt und ob eine erwachsene Begleitung dabei ist (`ja`/`nein`). Es soll die passende Meldung ausgeben. Nutze eine **verschachtelte** Verzweigung.

1.3.7 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 32: Bestanden? ●○○

Lies das Programm, **ohne** es auszuführen. Gib an, was es bei den Eingaben `70`, `50` und `49` ausgibt. Überprüfe danach am Computer.

```python
punkte = int(input("Punkte: "))
if punkte >= 50:
    print("bestanden")
else:
    print("nicht bestanden")
print("Ende")
```

Meine Vorhersagen:

Arbeitsauftrag 33: Einrückung entscheidet ●○○

Die beiden Programme unterscheiden sich nur in der Einrückung der letzten Zeile. Gib für beide an, was sie ausgeben. Erkläre den Unterschied in einem Satz.

Programm A:

```python
x = 3
if x > 5:
    print("groß")
    print("fertig")
```

Programm B:

```python
x = 3
if x > 5:
    print("groß")
print("fertig")
```

Ausgaben und Erklärung:

Arbeitsauftrag 34: Gerade oder ungerade? ●○○

Das Programm soll ausgeben, ob eine eingegebene ganze Zahl gerade oder ungerade ist. Ergänze die beiden Lücken ___ und teste dein Programm mit `8`, `7` und `0`.

```
zahl = int(input("Zahl: "))
if ___:
    print(zahl, "ist gerade.")
___
    print(zahl, "ist ungerade.")
```

Arbeitsauftrag 35: Fieber messen ●○○

Das Programm ordnet eine gemessene Körpertemperatur ein. Teste es mit den Eingaben aus der Tabelle. Überlege die Ausgabe zuerst und trage sie ein. Prüfe danach am Computer.

```
temp = float(input("Temperatur in Grad Celsius: "))
if temp < 37.5:
    print("normal")
elif temp < 38.5:
    print("erhöht")
else:
    print("Fieber")
```

Eingabe	`36.8`	`37.4`	`37.5`	`38.4`	`38.5`	`39.2`
Ausgabe						

Arbeitsauftrag 36: Fehler finden: Medaillen beim Sportfest ●●○

Beim Sportfest gibt es ab 60 Punkten Gold, ab 45 Punkten Silber und ab 30 Punkten Bronze. Wer weniger hat, bekommt keine Medaille. Das Programm soll die Medaille ausgeben. Es enthält **drei Fehler**. Einer davon erzeugt keine Fehlermeldung, sondern eine falsche Ausgabe.

```python
punkte = int(input("Punkte: "))
if punkte >= 30:
    print("Bronze")
elif punkte >= 45:
    print("Silber")
elif punkte >= 60
    print("Gold")
else:
print("keine Medaille")
```

1. Finde die drei Fehler. Notiere jeweils die Zeile und was falsch ist.
2. Schreibe das Programm richtig auf. Teste es mit den Grenzwerten 29, 30, 44, 45, 59 und 60.

Die Fehler:

Arbeitsauftrag 37: Was ziehe ich an? ●●○

Schreibe ein Programm, das die Temperatur (ganze Zahl in Grad Celsius) einliest und eine Kleidungsempfehlung ausgibt:

- unter 10 Grad: „Zieh eine Jacke an.“
- 10 bis unter 20 Grad: „Ein Pulli reicht.“
- ab 20 Grad: „T-Shirt-Wetter!“

Danach soll das Programm fragen, ob es regnet (`ja`/`nein`). Nur wenn die Antwort `ja` ist, gibt es zusätzlich „Nimm einen Regenschirm mit.“ aus. Teste mit allen Zweigen und den Grenzwerten.

Arbeitsauftrag 38: Schaltjahr mit Verzweigung ●●○

Im Abschnitt über Operatoren hast du im Auftrag *Schaltjahr-Check* einen Wahrheitswert berechnet. Schreibe das Programm jetzt so, dass es einen vollständigen Satz ausgibt, zum Beispiel `2024 ist ein Schaltjahr.` oder `2023 ist kein Schaltjahr.` Teste mit 2024, 2023, 1900 und 2000.

Zur Erinnerung: Ein Jahr ist ein Schaltjahr, wenn es durch 4 teilbar ist, aber nicht durch 100 – **außer** es ist auch durch 400 teilbar.

Arbeitsauftrag 39: Welches Dreieck? ●●●

Aus drei Seitenlängen `a`, `b` und `c` kann man nur dann ein Dreieck bauen, wenn je zwei Seiten zusammen länger sind als die dritte: `a + b > c`, `a + c > b` und `b + c > a`.

Schreibe ein Programm, das drei Seitenlängen einliest. Wenn sich kein Dreieck bauen lässt, soll es das ausgeben. Sonst soll es mit einer **verschachtelten** Verzweigung ausgeben, ob das Dreieck

- **gleichseitig** ist (alle drei Seiten gleich lang),
- **gleichschenklig** ist (genau zwei Seiten gleich lang) oder
- **allgemein** ist (alle Seiten verschieden).

Teste mit 3, 4, 5 und mit 5, 5, 8 und mit 5, 5, 5 und mit 1, 2, 5.

1.3.8 Das Wichtigste in Kürze

- Eine **Verzweigung** besteht aus einer **Bedingung** (Wert `True` oder `False`) und einem oder mehreren Anweisungsblöcken.
- `if Bedingung:` – der Block wird nur ausgeführt, wenn die Bedingung wahr ist (**einseitig**).
- `if … else:` – entweder der dann-Teil oder der sonst-Teil wird ausgeführt (**zweiseitig**).
- `if … elif … else:` – nur der Block der ersten zutreffenden Bedingung wird ausgeführt (**mehrseitig**). Die Reihenfolge ist wichtig.
- Hinter jeder Bedingung und hinter `else` steht ein **Doppelpunkt**. Die **Einrückung** (4 Leerzeichen) legt fest, was zum Block gehört.
- Bedingungen können mit `and`, `or` und `not` verknüpft werden – auf beiden Seiten mit vollständigem Vergleich.
- Verzweigungen können **verschachtelt** werden.

Selbstcheck

- ☐ Ich kann erklären, was eine Bedingung und was eine Verzweigung ist.
- ☐ Ich kann ein-, zwei- und mehrseitige Verzweigungen mit `if`, `else` und `elif` schreiben.
- ☐ Ich weiß, warum bei `elif` die Reihenfolge der Bedingungen wichtig ist.
- ☐ Ich kann an der Einrückung erkennen, welche Zeilen zu einem Block gehören, und kann einen `IndentationError` beheben.
- ☐ Ich kann zu einem Programm mit Verzweigung einen Programmablaufplan zeichnen und ein Ablaufprotokoll erstellen.
- ☐ Ich kann eine Verzweigung mit den Fachbegriffen dann-Teil und sonst-Teil beschreiben.