1.2 Operatoren: rechnen, vergleichen, verknüpfen

Leitidee

Ein Film dauert 135 Minuten. Wie viele Stunden und Minuten sind das? Ist eine Zahl gerade oder ungerade? Darf eine 15-Jährige ohne Ausweis ins Kino? Für solche Fragen muss ein Programm **rechnen**, **vergleichen** und mehrere Aussagen **verknüpfen**. Die Zeichen, mit denen man das in Python aufschreibt, heißen **Operatoren**. Einige kennst du schon: `+`, `-`, `*` und `/`. In diesem Abschnitt kommen neue Rechenzeichen dazu und vor allem Operatoren, deren Ergebnis `True` oder `False` ist. Mit diesen Wahrheitswerten kann ein Programm im nächsten Abschnitt Entscheidungen treffen.

Lernziele

Nach diesem Abschnitt kannst du

- die Rechenoperatoren `//`, `%` und `**` benutzen und erklären, was sie berechnen,
- mit `//` und `%` Minuten in Stunden und Minuten umrechnen und prüfen, ob eine Zahl gerade ist,
- zwei Werte mit den Vergleichsoperatoren `==`, `!=`, `<`, `>`, `<=` und `>=` vergleichen,
- Wahrheitswerte mit `and`, `or` und `not` verknüpfen und das Ergebnis vorhersagen,
- die Zuweisung `=` und den Vergleich `==` sicher unterscheiden.

1.2.1 Was ist ein Operator?

In der Rechnung `7 + 3` steht das Pluszeichen zwischen zwei Zahlen. Es sagt Python, **was** mit den beiden Zahlen geschehen soll: Sie werden addiert. Ein solches Zeichen heißt **Operator**. Die Werte, mit denen gerechnet wird, heißen **Operanden**. Das Ergebnis ist wieder ein Wert, hier `10`. Eine solche Verbindung aus Werten, Variablen und Operatoren, die Python zu einem Wert ausrechnet, nennt man einen **Ausdruck**.

Definition: Operator

Ein **Operator** ist ein Zeichen oder ein Wort, das eine Operation mit einem oder mehreren Werten (den **Operanden**) ausführt und daraus einen neuen Wert berechnet. Beispiel: In `7 + 3` ist `+` der Operator, `7` und `3` sind die Operanden.

In diesem Abschnitt lernst du drei Gruppen von Operatoren kennen: **arithmetische Operatoren** zum Rechnen (Ergebnis ist eine Zahl), **Vergleichsoperatoren** und **logische Operatoren** (Ergebnis ist jeweils ein Wahrheitswert, also `True` oder `False`).

1.2.2 Arithmetische Operatoren: mehr als die Grundrechenarten

Die vier Grundrechenarten `+`, `-`, `*` und `/` kennst du schon aus dem Abschnitt über Variablen. Python hat drei weitere Rechenoperatoren. Hier sind alle sieben auf einen Blick, jeweils mit den Zahlen 7 und 3:

Operator	Name	Beispiel	Ergebnis
`+`	Addition	`7 + 3`	`10`
`-`	Subtraktion	`7 - 3`	`4`
`*`	Multiplikation	`7 * 3`	`21`
`/`	Division	`7 / 3`	`2.3333333333333335`
`//`	Ganzzahldivision	`7 // 3`	`2`
`%`	Modulo (Rest)	`7 % 3`	`1`
`**`	Potenz	`7 ** 3`	`343`

Die drei neuen Operatoren schauen wir uns jetzt genauer an.

Ganzzahldivision `//` und Rest `%`

Erinnere dich an die Grundschule: Dort hast du „mit Rest“ geteilt. 17 Kekse werden gerecht an 5 Kinder verteilt. Jedes Kind bekommt 3 Kekse, und 2 Kekse bleiben übrig. Man schreibt: 17 : 5 = 3 Rest 2.

Genau diese beiden Zahlen liefern dir die Operatoren `//` und `%`:

```
1 print(17 // 5)
2 print(17 % 5)
3 print(17 / 5)
```

AUSGABE

```
3
2
3.4
```

- **Zeile 1:** `//` ist die **Ganzzahldivision**. Sie gibt an, wie oft die 5 *ganz* in die 17 passt: 3-mal. Bei positiven Zahlen fallen die Nachkommastellen einfach weg. Wie viele Kekse bekommt jedes Kind? 3.
- **Zeile 2:** `%` heißt **Modulo** und liefert den **Rest** der Division: $17 - 3 \cdot 5 = 2$. Wie viele Kekse bleiben übrig? 2.
- **Zeile 3:** Zum Vergleich die normale Division mit `/`. Sie rechnet mit Nachkommastellen weiter und liefert einen `float`.

Du kannst das Ergebnis immer überprüfen: $3 \cdot 5 + 2 = 17$. Ganzzahl-Ergebnis mal Teiler plus Rest ergibt wieder die Ausgangszahl. Übrigens: Das Zeichen `%` hat in Python **nichts mit Prozent** zu tun. `17 % 5` bedeutet „Rest von 17 geteilt durch 5“.

Anwendung: Minuten in Stunden und Minuten umrechnen

Wozu braucht man die Division mit Rest? Ein typisches Beispiel ist das Umrechnen von Zeiten. Eine Stunde hat 60 Minuten. Wie viele *volle* Stunden in einer Anzahl von Minuten stecken, liefert `// 60`. Die Minuten, die danach übrig bleiben, liefert `% 60`:

```
1 minuten_gesamt = int(input("Wie viele Minuten dauert der Film? "))
2 stunden = minuten_gesamt // 60
3 minuten = minuten_gesamt % 60
4 print(minuten_gesamt, "Minuten sind", stunden, "Stunden und",
5       minuten, "Minuten.")
```

AUSGABE
```
Wie viele Minuten dauert der Film? 135
135 Minuten sind 2 Stunden und 15 Minuten.
```

- **Zeile 1:** Die Eingabe wird gleich in eine ganze Zahl umgewandelt und in `minuten_gesamt` gespeichert.
- **Zeile 2:** `135 // 60` ergibt `2`, denn die 60 passt zweimal ganz in die 135.
- **Zeile 3:** `135 % 60` ergibt `15`, denn $135 - 2 \cdot 60 = 15$ Minuten bleiben übrig.
- **Zeilen 4–5:** Die Ausgabe ist lang und deshalb auf zwei Zeilen verteilt. Innerhalb der Klammern von `print` darf man in der nächsten Zeile weiterschreiben.

Versuch: Probiere es aus

Starte das Programm mit den Eingaben `60`, `59` und `600`. Stimmen die Ergebnisse? Was passiert bei `0`? Ändere das Programm dann so, dass es eine Anzahl von **Sekunden** in volle Minuten und restliche Sekunden umrechnet. Zum Beispiel sind 200 Sekunden 3 Minuten und 20 Sekunden.

Anwendung: Ist eine Zahl gerade?

Eine Zahl ist **gerade**, wenn sie sich ohne Rest durch 2 teilen lässt. Der Rest beim Teilen durch 2 ist also `0`. Bei ungeraden Zahlen bleibt immer der Rest `1`:

```
print(14 % 2, 15 % 2, 2024 % 4, 2026 % 4)
```

AUSGABE
```
0 1 0 2
```

14 ist gerade (Rest `0`), 15 ist ungerade (Rest `1`). Dasselbe funktioniert mit jedem Teiler: 2024 ist durch 4 teilbar (Rest `0`), 2026 nicht (Rest `2`).

Merke

Eine Zahl `a` ist genau dann durch `b` teilbar, wenn `a % b` den Wert `0` ergibt. Insbesondere ist eine Zahl gerade, wenn `zahl % 2` gleich `0` ist.

Potenzieren mit **

Mit zwei Sternchen berechnest du eine **Potenz**. `2 ** 3` bedeutet $2^3 = 2 \cdot 2 \cdot 2 = 8$:

```
print(2 ** 3, 10 ** 6, 5 ** 2)
```

AUSGABE
```
8 1000000 25
```

Die erste Zahl ist die Basis, die zweite der Exponent. `5 ** 2` ist also „5 hoch 2“, das Quadrat von 5.

Stolperstein

In manchen Programmen (z. B. in Tabellenkalkulationen) schreibt man „hoch“ mit dem Zeichen `^`. In Python bedeutet `^` etwas ganz anderes. Es gibt **keine** Fehlermeldung, aber ein falsches Ergebnis:

```
print(2 ^ 3)
```

AUSGABE
```
1
```

Für Potenzen brauchst du in Python immer `**`.

Merke

Arithmetische Operatoren: `+`, `-`, `*`, `/` (Division, Ergebnis ist immer ein `float`), `//` (Ganzzahldivision), `%` (Modulo, also der Rest der Division) und `**` (Potenz).

Was wird zuerst gerechnet?

Wie in Mathe gilt eine feste Reihenfolge: zuerst Klammern, dann Potenzen `**`, dann Punktrechnung, zuletzt Strichrechnung `+` und `-`. `//` und `%` zählen zur Punktrechnung, genau wie `*` und `/`.

```
1 print(2 + 3 ** 2)
2 print((2 + 3) ** 2)
3 print(10 - 7 % 4)
```

AUSGABE

```
11
25
7
```

- **Zeile 1:** Erst die Potenz: $3^2 = 9$, dann $2 + 9 = 11$.
- **Zeile 2:** Erst die Klammer: $2 + 3 = 5$, dann $5^2 = 25$.
- **Zeile 3:** Erst `7 % 4`, das ergibt `3`. Dann `10 - 3 = 7`.

Wenn du unsicher bist, setze Klammern. Sie schaden nie und machen das Programm leichter lesbar.

Arbeitsauftrag 15: Rechenausdrücke

Bestimme das Ergebnis jedes Ausdrucks zuerst im Kopf und trage es in die Tabelle ein. Überprüfe dann mit `print(...)` am Computer.

Ausdruck	Ergebnis
`23 // 4`	
`23 % 4`	
`23 / 4`	
`2 ** 5`	
`10 % 2`	
`7 % 10`	
`3 + 2 ** 3`	
`(3 + 2) ** 2`	

1.2.3 Vergleichsoperatoren: wahr oder falsch?

Viele Fragen kann man nur mit Ja oder Nein beantworten: Ist 7 größer als 3? Ist die Eingabe gleich „ja“? Python beantwortet solche Fragen mit einem **Wahrheitswert**: `True` (wahr, ja) oder `False` (falsch, nein). Den passenden Datentyp `bool` kennst du schon aus dem Abschnitt über Datentypen.

Eine solche Frage stellst du mit einem **Vergleichsoperator**:

Operator	Bedeutung	Beispiel	Ergebnis
==	ist gleich	`7 == 3`	`False`
!=	ist ungleich	`7 != 3`	`True`
<	ist kleiner als	`7 < 3`	`False`
>	ist größer als	`7 > 3`	`True`
<=	ist kleiner oder gleich	`7 <= 7`	`True`
>=	ist größer oder gleich	`3 >= 7`	`False`

Das Zeichen `!=` liest man als „ungleich“. Das Ausrufezeichen steht für „nicht“. Bei `<=` und `>=` steht das Gleichheitszeichen immer **rechts**.

```
1 a = 7
2 b = 3
3 print(a > b)
4 print(a == b)
5 print(a != b)
6 print(a <= 7)
```

AUSGABE

```
True
False
True
True
```

- **Zeilen 1–2:** Die Variablen `a` und `b` bekommen die Werte `7` und `3`.
- **Zeilen 3–5:** Ist 7 größer als 3? Ja (`True`). Ist 7 gleich 3? Nein (`False`). Ist 7 ungleich 3? Ja (`True`).
- **Zeile 6:** Ist 7 kleiner oder gleich 7? Ja, denn „gleich“ reicht schon. Also `True`.

Ergebnisse von Vergleichen speichern

Ein Wahrheitswert ist ein ganz normaler Wert. Du kannst ihn deshalb in einer Variablen speichern:

```
1 alter = int(input("Wie alt bist du? "))
2 ist_volljaehrig = alter >= 18
3 print("Volljährig:", ist_volljaehrig)
4 print(type(ist_volljaehrig))
```

AUSGABE

```
Wie alt bist du? 15
Volljährig: False
<class 'bool'>
```

- **Zeile 2:** Wie bei jeder Zuweisung wird zuerst die rechte Seite ausgewertet: Ist 15 größer oder gleich 18? Nein, das Ergebnis ist `False`. Erst dann wird dieser Wert in `ist_volljaehrig` gespeichert.
- **Zeile 4:** Der Typ ist `bool`, also ein Wahrheitswert.

Für Variablen mit Wahrheitswerten wählt man gern Namen, die wie eine Ja-Nein-Frage klingen: `ist_volljaehrig`, `hat_ausweis`, `ist_gerade`.

Texte vergleichen

Auch Texte kann man mit `==` und `!=` vergleichen. Das brauchst du, wenn eine Nutzerin zum Beispiel „ja“ oder „nein“ eingibt. Dabei kommt es auf jedes Zeichen an, auch auf Groß- und Kleinschreibung:

```
print("ja" == "ja", "Ja" == "ja", "ja " == "ja")
```

AUSGABE

```
True False False
```

Im zweiten Vergleich unterscheiden sich die Texte im ersten Buchstaben, im dritten durch ein Leerzeichen am Ende. Für Python sind das verschiedene Texte.

Stolperstein

Vergleichst du eine Eingabe mit einer Zahl und hast die Typumwandlung vergessen, gibt es **keine** Fehlermeldung. Das Ergebnis ist aber immer `False`:

```
zahl = input("Gib eine Zahl ein: ")
print(zahl == 7)
```

AUSGABE

```
Gib eine Zahl ein: 7
False
```

Der Text `"7"` ist nicht dasselbe wie die Zahl `7`. Abhilfe: `zahl = int(input(...))`.

Zuweisung = oder Vergleich ==?

Diese beiden Zeichen sehen sich ähnlich, bedeuten aber etwas völlig anderes:

- `x = 5` ist eine **Zuweisung**: „`x` bekommt den Wert 5.“ Dabei wird nichts geprüft.
- `x == 5` ist ein **Vergleich**: „Ist `x` gleich 5?“ Das Ergebnis ist `True` oder `False`. Dabei wird nichts verändert.

Verwechselst du die beiden, meldet Python oft einen Fehler. Python ahnt sogar, was du gemeint hast:

```
print(7 = 7)
```

FEHLERMELDUNG

```
  File "programm.py", line 1
    print(7 = 7)
          ^^^
SyntaxError: expression cannot contain assignment, perhaps you meant "=="?
```

Expression cannot contain assignment heißt: „Ein Ausdruck darf keine Zuweisung enthalten.“ Und *perhaps you meant* `"=="?` bedeutet „Meintest du vielleicht ==?“

Merke

- **Vergleichsoperatoren** liefern immer einen Wahrheitswert (`bool`: `True` oder `False`): == (gleich), != (ungleich), <, >, <=, >=.
- Verwechsle nicht = (Zuweisung) und == (Vergleich)!

So kannst du es sagen

So beschreibst du einen Vergleich in Fachsprache – vom einfachen zum genauen Satz:

- *„Da kommt True raus.“ → „Für `zahl = 7` liefert der Vergleich `zahl > 4` den Wahrheitswert `True`, weil 7 größer als 4 ist.“*
- *„Ich bekomme raus, ob die Zahl größer als 4 ist.“ → „Die Anweisung prüft, ob eine Zahl größer als 4 ist, und teilt das Ergebnis mit.“*

1.2.4 Logische Operatoren: `and`, `or` und `not`

Oft reicht eine einzige Frage nicht aus. Ein Film ist ab 12 Jahren freigegeben, und an der Kasse wird ein Ausweis verlangt. Du darfst also nur hinein, wenn **beide** Bedingungen erfüllt sind: Du bist mindestens 12 **und** du hast deinen Ausweis dabei. Solche Verknüpfungen schreibt man mit **logischen Operatoren**. Sie verknüpfen Wahrheitswerte und liefern wieder einen Wahrheitswert.

Es gibt drei logische Operatoren:

- `and` heißt „und“. Das Ergebnis ist nur dann `True`, wenn **beide** Seiten `True` sind. Sobald eine Seite `False` ist, ist das Ganze `False`.
- `or` heißt „oder“. Das Ergebnis ist `True`, wenn **mindestens eine** Seite `True` ist – also **auch, wenn beide** wahr sind. Das ist anders als bei „entweder … oder“ im Alltag („Tee oder Kaffee?“). Nur wenn beide Seiten `False` sind, ist das Ergebnis `False`.
- `not` heißt „nicht“. Es steht **vor** einem einzelnen Wahrheitswert und kehrt ihn um: Aus `True` wird `False` und aus `False` wird `True`.

```
1 print(True and False)
2 print(True or False)
3 print(not True)
4 print(not 7 > 3)
```

AUSGABE

```
False
True
False
False
```

- **Zeile 1:** Die rechte Seite ist `False`. Für `and` müssten beide Seiten wahr sein, also `False`.
- **Zeile 2:** Für `or` genügt eine wahre Seite, also `True`.
- **Zeile 3:** `not` kehrt `True` um.
- **Zeile 4:** Zuerst wird der Vergleich ausgewertet: `7 > 3` ist `True`. Dann kehrt `not` das Ergebnis um.

Die drei Operatoren auf einen Blick. Dabei stehen `A` und `B` für zwei beliebige Wahrheitswerte:

A	B	A and B	A or B	not A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Merke

Logische Operatoren verknüpfen Wahrheitswerte:

- `and` – beide Bedingungen müssen wahr sein,
- `or` – mindestens eine Bedingung muss wahr sein,
- `not` – kehrt einen Wahrheitswert um.

Auch Wahrheitswerte kann man mit `==` und `!=` vergleichen. `A == B` ist `True`, wenn beide **denselben** Wahrheitswert haben. `A != B` ist `True`, wenn sie **verschieden** sind – also genau dann, wenn **genau einer** von beiden wahr ist:

A	B	A == B	A != B
False	False	True	False
False	True	False	True
True	False	False	True
True	True	True	False

Die Spalte `A != B` merkst du dir am besten als „entweder … oder“: wahr, wenn genau eine Seite wahr ist. Diese Verknüpfung begegnet dir in Kapitel 2 als XOR wieder.

Vergleiche verknüpfen

Meistens verknüpft man mit `and` und `or` keine festen Werte, sondern Vergleiche. Hier wird geprüft, ob eine Person jugendlich ist, also mindestens 13 und jünger als 18 Jahre alt:

```
1 alter = int(input("Wie alt bist du? "))
2 ist_jugendlich = alter >= 13 and alter < 18
3 print("Jugendlich:", ist_jugendlich)
```

AUSGABE

```
Wie alt bist du? 15
Jugendlich: True
```

Python wertet Zeile 2 in dieser Reihenfolge aus:

1. `alter >= 13`, also `15 >= 13`, ergibt `True`.
2. `alter < 18`, also `15 < 18`, ergibt `True`.
3. `True and True` ergibt `True`.

Rechnungen und Vergleiche werden also **vor** den logischen Operatoren ausgewertet. Unter den logischen Operatoren gilt: zuerst `not`, dann `and`, zuletzt `or`. Wenn du `and` und `or` in einem Ausdruck mischst, setze Klammern. Dann sieht jeder sofort, was zusammengehört.

Stolperstein

Im Deutschen sagt man „älter als 12 und jünger als 18“. Python braucht aber auf **beiden** Seiten von `and` einen vollständigen Vergleich:

```
alter = 15
print(alter > 12 and < 18)
```

FEHLERMELDUNG

```
SyntaxError: invalid syntax
```

Richtig ist: `alter > 12 and alter < 18`.

Klammern entscheiden: Darf ich zocken?

Bei drei Bedingungen kommt es darauf an, was zusammengehört. Ein Beispiel aus dem Alltag:

- `A` = Die Hausaufgaben sind fertig.
- `B` = Das Zimmer ist aufgeräumt.
- `C` = Es ist Wochenende.

Zwei Eltern formulieren ihre Regel fürs Zocken unterschiedlich:

- **Regel 1:** `(A and B) or C` – „Du darfst zocken, wenn die Hausaufgaben fertig **und** das Zimmer aufgeräumt ist – **oder** wenn Wochenende ist." Am Wochenende reicht also `C = True` allein.
- **Regel 2:** `A and (B or C)` – „Du darfst zocken, wenn die Hausaufgaben fertig sind **und** zusätzlich das Zimmer aufgeräumt **oder** Wochenende ist." Hier müssen die Hausaufgaben **immer** fertig sein.

Probiere es aus: Die Hausaufgaben sind nicht fertig, das Zimmer ist nicht aufgeräumt, aber es ist Wochenende.

```
1 A = False   # Hausaufgaben sind fertig
2 B = False   # Zimmer ist aufgeräumt
3 C = True    # Es ist Wochenende
4
5 ausdruck_1 = (A and B) or C
6 ausdruck_2 = A and (B or C)
7
8 print("(A and B) or C =", ausdruck_1)
9 print("A and (B or C) =", ausdruck_2)
```

AUSGABE

```
(A and B) or C = True
A and (B or C) = False
```

- **Zeilen 1–3:** Die drei Bedingungen werden als Wahrheitswerte gespeichert. Die Kommentare hinter `#` erklären, wofür jede Variable steht.
- **Zeile 5:** Zuerst wird die Klammer berechnet: `False and False` ergibt `False`. Dann folgt `False or True`, das ist `True`.
- **Zeile 6:** Zuerst die Klammer: `False or True` ergibt `True`. Dann `False and True`, das ist `False`.

Nach Regel 1 darfst du also zocken, nach Regel 2 nicht. Die **Klammerung** entscheidet darüber, welche Bedingung **zwingend** erfüllt sein muss. Die Tabelle zeigt alle acht Möglichkeiten:

A	B	C	A and B	B or C	(A and B) or C	A and (B or C)
False	False	False	False	False	False	False
False	False	True	False	True	True	False
False	True	False	False	True	False	False
False	True	True	False	True	True	False
True	False	False	False	False	False	False
True	False	True	False	True	True	True
True	True	False	True	True	True	True
True	True	True	True	True	True	True

In vier Zeilen unterscheiden sich die beiden letzten Spalten. Die Ausdrücke `(A and B) or C` und `A and (B or C)` bedeuten also **nicht** dasselbe – man sagt: Sie sind **nicht äquivalent**.

Merke

Klammern bestimmen, welcher Teilausdruck zuerst ausgewertet wird. `(A and B) or C` und `A and (B or C)` sind **nicht äquivalent**. Setze deshalb immer Klammern, wenn du `and` und `or` in einem Ausdruck mischst.

Bedingungen verneinen: die Regeln von De Morgan

Manchmal will man das **Gegenteil** einer verknüpften Bedingung prüfen, zum Beispiel: Ist eine Person **nicht** jugendlich? Am einfachsten setzt man `not` vor die ganze Bedingung in Klammern:

```
1 alter = 20
2 print(not (alter >= 13 and alter < 18))
3 print(alter < 13 or alter >= 18)
```

AUSGABE

```
True
True
```

- **Zeile 2:** Die Klammer wird zuerst ausgewertet: `20 >= 13` ist `True`, `20 < 18` ist `False`, zusammen mit `and` also `False`. `not` macht daraus `True`. Die Person ist nicht jugendlich.
- **Zeile 3:** Dieselbe Frage ohne `not`: Die Person ist nicht jugendlich, wenn sie **jünger als 13 oder mindestens 18** ist. Auch das ergibt `True`.

Beide Zeilen liefern für **jedes** Alter dasselbe Ergebnis. Probiere es mit anderen Werten aus, zum Beispiel `alter = 15` oder `alter = 9`. Beim Umformen ist zweierlei passiert: Jeder einzelne Vergleich wurde verneint (aus >= wurde <, aus < wurde >=), und aus `and` wurde `or`. Das ist kein Zufall, sondern eine allgemeine Regel. Sie heißt nach dem englischen Mathematiker **Augustus De Morgan**:

A	B	not (A and B)	not A or not B	not (A or B)	not A and not B
False	False	True	True	True	True
False	True	True	True	False	False
True	False	True	True	False	False
True	True	False	False	False	False

Die dritte und vierte Spalte stimmen in jeder Zeile überein, ebenso die fünfte und sechste. Im Alltag kennst du das auch: „Es ist **nicht** so, dass es regnet **oder** windet“ bedeutet „Es regnet **nicht und** es windet **nicht**“.

Merke

Regeln von De Morgan:

- `not (A and B)` ist dasselbe wie `not A or not B`,
- `not (A or B)` ist dasselbe wie `not A and not B`.

Verneint man eine verknüpfte Bedingung, wird **jede** Teilbedingung verneint, und aus `and` wird `or` (und umgekehrt).

In der Mathematik und in Kapitel 2 schreibt man $\wedge$ für `and`, $\vee$ für `or` und $\neg$ für `not`. Die beiden Regeln lauten dann $\neg(A \wedge B) \equiv \neg A \vee \neg B$ und $\neg(A \vee B) \equiv \neg A \wedge \neg B$. Das Zeichen $\equiv$ heißt: Beide Seiten haben für alle Werte von A und B dasselbe Ergebnis.

Um einen einzelnen Vergleich zu verneinen, brauchst du kein `not`. Es genügt, den Vergleichsoperator umzudrehen:

Vergleich	verneint
`a == b`	`a != b`
`a < b`	`a >= b`
`a > b`	`a <= b`

Stolperstein

Der häufigste Fehler: Man verneint die Teilbedingungen, vergisst aber, `and` in `or` umzuwandeln.

```
alter = 20
print(alter < 13 and alter >= 18)
```

AUSGABE

```
False
```

Kein Mensch ist gleichzeitig jünger als 13 **und** mindestens 18 Jahre alt. Diese Bedingung ist also **immer** `False`, egal welches Alter eingegeben wird. Richtig ist `alter < 13 or alter >= 18`.

Arbeitsauftrag 16: Bedingungen verneinen

Schreibe zu jeder Bedingung eine Bedingung **ohne** `not`, die genau das Gegenteil prüft. Teste deine Antworten am Computer mit ein paar Werten.

1. `zahl >= 1 and zahl <= 6` (eine Würfelzahl)
2. `note == 1 or note == 2`
3. `not (temperatur > 30 or regen)` *(Tipp: `regen` ist eine Variable mit einem Wahrheitswert; „nicht `regen`“ darfst du hier als `not regen` stehen lassen.)*

Meine Bedingungen:

Alles zusammen

Das folgende Programm zeigt noch einmal alle drei Gruppen von Operatoren:

```
1 a = 7
2 b = 3
3
4 print("Summe:", a + b)
5 print("Ganzzahldivision:", a // b)
6 print("Rest (Modulo):", a % b)
7 print("Potenz:", a ** 2)
8
9 print("a größer b?", a > b)
10 print("a gleich b?", a == b)
11
12 alter = 15
13 hat_ausweis = True
14 print("Darf ins Kino (ab 12) und hat Ausweis dabei:",
15       alter >= 12 and hat_ausweis)
```

AUSGABE

```
Summe: 10
Ganzzahldivision: 2
Rest (Modulo): 1
Potenz: 49
a größer b? True
a gleich b? False
Darf ins Kino (ab 12) und hat Ausweis dabei: True
```

- **Zeilen 4–7:** arithmetische Operatoren. Die 3 passt 2-mal ganz in die 7, Rest 1. `7 ** 2` ist $7^2 = 49$.
- **Zeilen 9–10:** Vergleiche. Ihr Ergebnis ist ein Wahrheitswert.
- **Zeilen 12–13:** Die Variable `hat_ausweis` speichert direkt den Wahrheitswert `True`.
- **Zeilen 14–15:** Zuerst wird `alter >= 12` ausgewertet, das ergibt `True`. Dann wird `True and True` berechnet. Das Ergebnis ist `True`. Probiere aus, was sich ändert, wenn du `hat_ausweis = False` oder `alter = 11` setzt.

Arbeitsauftrag 17: Wahr oder falsch?

Es gilt `x = 5` und `y = 10`. Bestimme für jeden Ausdruck, ob er `True` oder `False` ergibt. Überprüfe deine Antworten danach am Computer.

Ausdruck	Ergebnis
`x < y`	
`x == 5 and y == 5`	
`x == 5 or y == 5`	
`not x > y`	
`x % 2 == 0`	
`x + y >= 15 and not y == 10`	

1.2.5 Übungen

Arbeitsauftrag 18: Gerade oder ungerade?

Schreibe ein Programm, das eine Ganzzahl einliest und mit Hilfe des Modulo-Operators `%` einen Wahrheitswert ausgibt, der angibt, ob die Zahl gerade ist.

So könnte es aussehen:

AUSGABE
```
Ganzzahl eingeben: 14
Ist gerade: True
```

Arbeitsauftrag 19: Schaltjahr-Check

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

Schreibe ein Programm, das eine Jahreszahl einliest und einen Wahrheitswert ausgibt, der angibt, ob es ein Schaltjahr ist. Verwende dafür `and`, `or` und `not` bzw. Vergleichsoperatoren. Du brauchst dafür nur Operatoren, sonst nichts.

Tipp

Zerlege die Regel in Teile und übersetze jeden Teil einzeln:

- „durch 4 teilbar“ → `jahr % 4 == 0`
- „nicht durch 100 teilbar“ → `jahr % 100 != 0`
- „durch 400 teilbar“ → `jahr % 400 == 0`

Ein Jahr ist also Schaltjahr, wenn *die ersten beiden Teile zusammen* wahr sind **oder** wenn der dritte Teil wahr ist. Setze Klammern um das, was zusammengehört. Teste dein Programm mit 2024 (Schaltjahr), 2023 (kein Schaltjahr), 1900 (kein Schaltjahr) und 2000 (Schaltjahr).

1.2.6 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 20: Teilen mit Rest ●○○

Bestimme das Ergebnis jedes Ausdrucks zuerst im Kopf und trage es in die Tabelle ein. Überprüfe dann mit `print(...)` am Computer.

Ausdruck	Ergebnis	Ausdruck	Ergebnis
`29 // 5`		`2 ** 4`	
`29 % 5`		`17 % 2`	
`40 % 8`		`10 - 4 // 2`	
`3 % 8`		`(10 - 4) // 2`	

Arbeitsauftrag 21: True oder False? ●○○

Es gilt `a = 4` und `b = 9`. Bestimme für jeden Vergleich, ob er `True` oder `False` ergibt. Überprüfe deine Antworten danach am Computer.

Ausdruck	Ergebnis	Ausdruck	Ergebnis
`a < b`		`a * 2 > b`	
`b != 9`		`b % a == 1`	
`a >= 5`		`a + 5 == b`	

Arbeitsauftrag 22: Wahrheitstabelle ausfüllen ●○○

Fülle die Tabelle aus. Bestimme zuerst die Spalte `not B`. Mit ihrer Hilfe kannst du die beiden letzten Spalten ausfüllen.

A	B	not B	A and not B	A or not B
False	False			
False	True			
True	False			
True	True			

Arbeitsauftrag 23: Eier verpacken ●○○

Eier werden in Kartons zu je 6 Stück verpackt. Das Programm soll eine Anzahl von Eiern einlesen und ausgeben, wie viele Kartons voll werden und wie viele Eier übrig bleiben. Ergänze die beiden Lücken `___` mit `//` und `%`. Teste mit `20`, `18` und `5`.

```
1 eier = int(input("Wie viele Eier? "))
2 kartons = ___     # volle Kartons
3 rest = ___        # übrige Eier
4 print(eier, "Eier ergeben", kartons, "volle Kartons und",
5       rest, "übrige Eier.")
```

Arbeitsauftrag 24: Bedingungen aufschreiben ●●○

Schreibe zu jeder Aussage einen Python-Ausdruck, der `True` ergibt, wenn die Aussage stimmt, und sonst `False`. Teste jeden Ausdruck am Computer mit zwei passenden Werten.

1. Die Zahl `x` ist durch 3 teilbar.
2. Das Alter `alter` liegt zwischen 10 und 15 (beide Grenzen gehören dazu).
3. Die Punktzahl `punkte` ist kleiner als 0 oder größer als 100.
4. Die Zahl `zahl` ist ungerade.
5. Die Anzahl `stunden` ist nicht 0.

Meine Ausdrücke:

Arbeitsauftrag 25: Fehler finden: Teilbar durch 5? ●●○

Das Programm soll eine ganze Zahl einlesen und einen Wahrheitswert ausgeben, der angibt, ob die Zahl durch 5 teilbar ist. Es enthält **drei Fehler**.

```
1 zahl = input("Zahl: ")
2 durch_fuenf = zahl % 5 = 0
3 print("Durch 5 teilbar:" durch_fuenf)
```

1. Finde die drei Fehler. Notiere jeweils die Zeile und was falsch ist.
2. Schreibe das Programm richtig auf und teste es mit 35 und 42.

Die Fehler:

Gymnasium Ottweiler

Arbeitsauftrag 26: Ausleihe in der Bücherei ●●○

In der Stadtbücherei gilt: Man darf ein Buch ausleihen, wenn man einen Ausweis hat **und** entweder keine offenen Gebühren hat **oder** jünger als 18 ist. Im Programm sieht das so aus:

```python
hat_ausweis = True
gebuehren = 0
alter = 30
darf_ausleihen = hat_ausweis and (gebuehren == 0 or alter < 18)
print("Darf ausleihen:", darf_ausleihen)
```

1. Bestimme `darf_ausleihen` für die vier Fälle in der Tabelle. Prüfe am Computer, indem du die Werte in den Zeilen 1 bis 3 änderst.
2. Jemand lässt die Klammern weg: `hat_ausweis and gebuehren == 0 or alter < 18`. Bestimme das Ergebnis für den vierten Fall und erkläre den Unterschied.

hat_ausweis	gebuehren	alter	darf_ausleihen
True	0	30	
True	2	15	
True	2	40	
False	0	12	

Zu 2.:

Gymnasium Ottweiler

Arbeitsauftrag 27: Ziffern zerlegen ●●●

Schreibe ein Programm, das eine dreistellige Zahl einliest und ihre Ziffern einzeln ausgibt: die Hunderterziffer, die Zehnerziffer und die Einerziffer. Zum Schluss soll es die **Quersumme** ausgeben, also die Summe der drei Ziffern. Verwende dafür `//` und `%`.

So könnte es aussehen:

AUSGABE

```
Dreistellige Zahl: 472
Hunderter: 4
Zehner: 7
Einer: 2
Quersumme: 13
```

Tipp

Die Einerziffer ist der Rest beim Teilen durch 10. Die Hunderterziffer sagt, wie oft die 100 ganz in die Zahl passt. Für die Zehnerziffer brauchst du zwei Schritte: `472 // 10` ergibt `47`. Welche Rechnung liefert daraus die `7`?

1.2.7 Das Wichtigste in Kürze

- Ein **Operator** verarbeitet einen oder mehrere Werte (Operanden) zu einem neuen Wert.
- **Arithmetische Operatoren:** `+ - * /` sowie `//` (Ganzzahldivision), `%` (Rest) und `**` (Potenz). Potenz vor Punkt vor Strich, Klammern zuerst.
- `zahl // 60` und `zahl % 60` zerlegen z. B. Minuten in Stunden und restliche Minuten. `zahl % 2 == 0` prüft, ob eine Zahl gerade ist.
- **Vergleichsoperatoren** `== != < > <= >=` liefern immer `True` oder `False`.
- **Logische Operatoren:** `and` (beide wahr), `or` (mindestens eine wahr), `not` (umkehren).
- **Klammern** legen fest, was zuerst ausgewertet wird: `(A and B) or C` und `A and (B or C)` sind nicht äquivalent.
- **Regeln von De Morgan:** `not (A and B)` ist dasselbe wie `not A or not B`, `not (A or B)` dasselbe wie `not A and not B`. So verneint man verknüpfte Bedingungen.
- `=` weist einen Wert zu, `==` vergleicht zwei Werte.

Selbstcheck

- ☐ Ich kann erklären, was `//`, `%` und `**` berechnen, und Beispiele im Kopf ausrechnen.
- ☐ Ich kann eine Anzahl Minuten mit `//` und `%` in Stunden und Minuten umrechnen.
- ☐ Ich kann mit `%` prüfen, ob eine Zahl gerade oder durch eine andere Zahl teilbar ist.
- ☐ Ich kenne die sechs Vergleichsoperatoren und weiß, dass ihr Ergebnis ein Wahrheitswert ist.
- ☐ Ich kann vorhersagen, was `and`, `or` und `not` ergeben.
- ☐ Ich kann erklären, warum `(A and B) or C` etwas anderes bedeutet als `A and (B or C)`.
- ☐ Ich kann eine verknüpfte Bedingung mit den Regeln von De Morgan verneinen.
- ☐ Ich verwechsle `=` und `==` nicht mehr.

Gymnasium Ottweiler