0 Hello world

0.0.1. Traditionell ist das erste Programm, das man in einer neuen Programmiersprache lernt, ein Programm, das «Hello world!» ausgibt.

In Python geht dies so:

```
print("Hello world!")
```

oder gleichbedeutend

```
print('Hello world!')
```

Dies illustriert auch, dass es in Python meistens egal ist, ob man einen String (= eine Zeichenkette) mit doppelten Anführungszeichen (also `"` und `"`) oder einfachen Anführungszeichen (also `'` und `'`) umschliesst.

1 Variablen, Datentypen und Datenstrukturen

1.1 Allgemeines

1.1.1. Informatik ist die Wissenschaft von der Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen. Statt von Informationen spricht man oft von Daten und meint damit vor allem die maschinenlesbare Darstellung von Informationen.

Wir erklären deswegen zunächst, wie man Daten mit Hilfe von Variablen speichert.

Definition 1.1.2 Variable (beim Programmieren), (elementare) Datentypen: Integer, Float, String, Boolean

Eine **Variable** ist ein «mit einem Namen versehener Speicherplatz für Daten».
Konkret hat eine Variable einen Namen und einen Wert von einem gewissen **Typ** (= **Datentyp**).
Der Typ gibt an, welche Art Information die Variable speichert.
Die wichtigsten nachträglich ergänzt: elementaren Datentypen (= Mengen erlaubter Werte) sind:

- numerische Datentypen (Zahlen):
  - **Integers** = ganze Zahlen — von einer sehr kleinen, negativen bis zu einer sehr grossen, positiven ganzen Zahl
  - **Floats** = Fliesskommazahlen = Kommazahlen — bis zu einer gewissen Genauigkeit
- **Strings** = Zeichenketten
- **Booleans** = Boolescher Datentyp = Wahrheitswerte, also `False`, `True` (also wahr, falsch) (George Boole)

1.1.3. Namen von Variablen beginnen in der Regel mit einem Kleinbuchstaben, danach können weitere Kleinbuchstaben folgen, auch Grossbuchstaben, Ziffern und Unterstriche (= Bodenstriche) sind erlaubt – von der Verwendung von Umlauten und Sonderzeichen wird dringend abgeraten, Leerzeichen sind nicht erlaubt.

Variablennamen sollten kurz, aber aussagekräftig sein. Je aussagekräftiger der Variablenname ist, desto leichter tut man sich beim Programmieren und desto verständlicher sind die Programme für andere.

Anfangs ist es sinnvoll, deutsche Begriffe als Variablennamen zu verwenden: Da die Sprache Python gewisse englische Begriffe verwendet (etwa print, if, else, for, while, True, False, ...), ist dann stets klar, was man selbst definiert hat.

Beispiel 1.1.4. Wir möchten den Namen einer Person, ihre Körpergrösse, ihre letzte Mathenote und die Information, ob sie erwachsen ist, speichern.

In Python gibt es einige «built-in functions». Im obigen Programm tauchen zwei davon auf:

- `print(<Argument>)`: Gibt den Wert des Arguments im Terminal in einer Zeile aus. Es dürfen auch mehrere, durch Kommata getrennte Argumente angegeben werden; diese werden dann durch Leerzeichen getrennt ausgegeben.
- `type(<Argument>)`: Liefert den Datentyp des Arguments. Diese Funktion wird beim Programmieren sehr selten benötigt.

Merke 1.1.5 Definition von Variablen in Python, Zuweisungszeichen

Das Zeichen `=` wird in Python als **Zuweisungszeichen** verwendet.
Variablen werden in Python dadurch definiert, dass man ihren Namen links des Zuweisungszeichens aufschreibt und rechts davon den gewünschten Wert angibt.

```
<Variablenname> = <Wert>
```

Bereits definierten Variablen kann auf dieselbe Weise ein neuer Wert zugewiesen werden.

1.1.6. Variablen (in der Informatik) haben während des Programmablaufs in der Regel viele verschiedene Werte.

Dies ist ein wichtiger Unterschied zu Variablen in der Mathematik, die oft für allgemeine mathematische Objekte stehen (etwa «Sei P ein beliebiger Punkt in der Ebene.»). (Ein anderer Unterschied ist, dass Variablennamen in der Mathematik meist nur aus einem einzigen Buchstaben bestehen.)
Beispiel 1.1.7 (Forsetzung von Beispiel 1.1.4)**.** Wir nehmen an, dass unsere Person um 10 Zentimeter wächst, eine 5.5 in Mathe schreibt und erwachsen wird. Diese Datenveränderung können wir wie folgt in den Variablen speichern.

Die erste Zeile weist der bereits definierten Variablen `groesse` als neuen Wert den «alten Wert von `groesse` plus 10» zu. Beachte, dass `=` beim Programmieren ein Zuweisungszeichen ist. Die mathematische Gleichung $x = x + 10$ hingegen hat eine vollkommen andere Bedeutung (und keine Lösung).

1.2 Zahlen (integers, floats)

1.2.1 (Standardrechenzeichen)**.** Die üblichen Rechenoperationen werden in Python wie folgt geschrieben.

Operation	Symbol in Python	Beispiel in Python	mathematische Notation
Addition	`+`	`a+10`	$a + 10$
Subtraktion	`-`	`10-x`	$10 - x$
Multiplikation	`*`	`x*y`	xy oder $x \cdot y$
Division	`/`	`4/y`	$\frac{4}{y}$
Potenzieren	`**`	`a**b`	a^b

1.2.2 (Wurzeln in Python berechnen)**.** In der Mathematik lernt bzw. definiert man

$$x^{\frac{1}{2}} = \sqrt{x} \qquad x^{\frac{1}{3}} = \sqrt[3]{x} \qquad \dots \qquad x^{\frac{1}{n}} = \sqrt[n]{x}$$

Es gilt also (in einem Mischmasch aus mathematischer und Python-Notation)

$$\sqrt{x} = \text{x**0.5} = \text{x**(1/3)} \qquad \sqrt[3]{x} = \text{x**(1/2)} \qquad \sqrt[n]{x} = \text{x**(1/n)}$$

1.2.3. In Python gibt es zwei weitere Zeichen für Divisionsoperationen: Statt des oben erklärten Divisionszeichens `/` verwendet man ein «doppeltes» Divisionszeichen `//` bzw. ein Prozentzeichen = «Divisionszeichen mit zwei Kreislein» `%`.

Operation	Symbol in Python	Beispiel in Python	mathematische Notation
Rest einer Division	`%`	`63 % 15` (Ergebnis 3)	keine Standardnotation
Ganzzahlquotient	`//`	`63 // 15` (Ergebnis 4)	keine Standardnotation

Zu den Begriffen: Bei einer Division mit Rest erhält man zwei Ergebnisse: Den sogenannten **Ganzzahlquotienten** und den **Rest**. Hier ein Beispiel:

$$63 : 15 = \underbrace{4}_{\text{Ganzzahlquotient}} \text{ Rest } \underbrace{3}_{\text{Rest}} \qquad \text{gleichbedeutend} \qquad 63 = 15 \cdot 4 + 3$$

$$\text{gleichbedeutend} \qquad \frac{63}{15} = \frac{60}{15} + \frac{3}{15} = 4 + \frac{3}{15}$$

Vermeide den «beliebten Fehler»: Es gilt $\frac{63}{15} = 4.2$, aber 2 ist nicht der Rest der Division. Der Rest ist $0.2 \cdot 15 = 3$ wie soeben erklärt.

1.2.4 (Allgemeines zum Lösen von Programmieraufgaben)**.**

- Lege für jede neue Aufgabe ein neues Programm in deinem Ordner «`python`» an. Das Programm für die folgende Aufgabe A1 könntest du zum Beispiel «`rechnen-mit-variablen.py`» oder «`A1-rechnen-mit-variablen.py`» nennen.
- Das Zeichen `#` kennzeichnet in Python den Beginn eines Kommentars. Alles, was in derselben Zeile folgt, wird beim Ausführen ignoriert.

⚒ Aufgabe A1 Ergänze das folgende Programm so, dass sich der Wert der Variablen `a` so verändert, wie das in den Kommentaren in der jeweiligen Zeile angegeben ist.

Wenn du das Programm abtippst, musst du die Kommentare natürlich nicht abtippen.

```
a = 7
a = 2*a       # verdopple a
a =           # erhöhe a um 10
a =           # erhebe a in seine 3. Potenz
              # ersetze a durch seinen Ganzzahlquotient bei Division durch 7
              # erhebe a in seine 10. Potenz
              # ersetze a durch seinen Rest bei Division durch 100
              # erhöhe a um 5
              # ersetze a durch seine Quadratwurzel
print(a)
```

Zur Kontrolle deiner Lösung:

- Für `a=7` in der ersten Zeile sollte am Ende `9.0` herauskommen.
- Ändere die erste Zeile zu a=12345. Dann sollte `7.3484692283495345` ausgegeben werden.

1.2.5. Runde Klammern (also `(` und `)`) werden wie in mathematischen Termen verwendet. Auch in Python gilt die Regel «Potenzen-vor-Punkt-vor-Strich». Multiplikationspunkte (= Malpunkte) dürfen aber nicht weggelassen werden. Zum Beispiel wird $4 - 10\frac{(3^5-\sqrt{2})a}{7b}$ in Python zu `4-10*((3**5-2**(1/2))*a)/(7*b)`.

⚒ Aufgabe A2 Schreibe die folgende Formel (die von einer Variablen n abhängt) in Python-Notation in die zweite Zeile des folgenden Programms.

$$\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Bemerkung: Diese Formel ist die sogenannte Moivre-Binet-Formel zur Berechnung der n-ten Fibonacci-Zahl.

```
n = 10
f =          # hier ist die Formel in Python-Notation einzutragen
print(f)
```

Zur Kontrolle deiner Lösung:

- Für $n = 10$ liefert die obige Formel 55. Python liefert fast diese Zahl. Warum nur fast?
- Ändere die erste Zeile zu n=20. Dann ist das korrekte Ergebnis 6765.
- Die ersten n-Fibonacci-Zahlen sind $0, 1, 1, 2, 3, 5, 8, 13, 21, 35, 55, \ldots$

Aufgabe A3 In der Mathematik lernt man: Die quadratische Gleichung (in Standardform)

$$ax^2 + bx + c = 0$$

hat die beiden Lösungen (Mitternachtsformel)

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad \text{und} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

(a) Du willst die Gleichung $2x^2 + 12x - 182 = 0$ mit der Mitternachtsformel in Python lösen. Tippe dafür das folgende Programm ab und korrigiere die drei falschen Zeilen.

```
a = 2
b = 12
c = -182

x1 = 1000       # zu korrigieren
x2 = 1000       # zu korrigieren

print('Die Gleichung', a, 'x^2 +', b, 'x +', c, '=0 hat die Lösungen')
print('x1=', x1)
print('x2=', x2)

print('Probe (es sollte beide Male Null herauskommen):')
# Probe für x1
print(a*x1**2+b*x1+c)
# Probe für x2
print(1000)     # zu korrigieren
```

(b) Beantworte die folgenden Fragen, indem du die ersten drei Zeilen deines Programms anpasst:

(i) Welche Lösungen hat die Gleichung $4x^2 - 28x + 49 = 0$? Was fällt dir auf?

(ii) Welche Lösungen hat die Gleichung $x^2 - 6x + 10 = 0$? Was fällt dir auf?

Aufgabe A4 Betrachte das folgende Programm

```
a = 5
b = 7
print("a =", a, ", b =", b)
#
# Hier fehlt Code
#
print("a =", a, ", b =", b)
```

Ergänze die fehlenden Code-Zeilen so, dass die Werte von `a` und `b` miteinander vertauscht werden. Die Zeilen mit den `print`-Befehlen dürfen nicht verändert werden. Das Programm soll auch dann funktionieren, wenn die Anfangswerte von `a` und `b` verändert werden.

Hinweis: Verwende eine zusätzliche Hilfsvariable `h`.

Bonusfrage: Kann man die Vertauschung auch ohne Hilfsvariable durchführen?

Aufgabe A5 Was macht der folgende Ausschnitt aus einem Programm mit den Werten der Variablen `a` und `b`? Bitte durch Nachdenken lösen, ohne Computer.

```
a = a-b
b = a+b
a = b-a
```

1.3 Strings

1.3.1. Ich erinnere daran, dass mit einem String eine Zeichenkette gemeint ist und dass Strings in Python durch Anführungszeichen (einfache oder doppelte) gekennzeichnet werden.
Beispiel 1.3.2.

```
s = 'Hello 007!'              # String
t = '12345'                   # String, der nur aus Ziffern besteht
x = 12345                     # Integer
```

1.3.3. Die Länge eines Strings kann man mit der vordefinierten Funktion `len` (für «length») wie folgt erhalten.

```
s = 'Hello!'
laenge = len(s)
print(laenge)            # Ausgabe
```

«Rechnen» mit Strings

1.3.4. In Python ist es erlaubt,

(a) zwei Strings zu «addieren»;
(b) einen Integer mit einem String zu «multiplizieren» (und auch andersherum: «String mal Integer»). ✎

```
s = 'taschen'
t = 'tuch'
print(s+t)           # Ausgabe:
print(t+s)           # Ausgabe:
print(2*s)           # Ausgabe:
print(t*3)           # Ausgabe:
print(s+t*2)         # Ausgabe:
print(len(s+t))      # Ausgabe:
print(len(s)*'-')    # Ausgabe:
```

⚒ **Aufgabe A6** Überlege dir zunächst ohne Computer, welche Ausgabe das folgende Programm liefert. Teste dann deine Vermutung mit dem Computer.

```
print("Drei" + "Sieben")
print("3 + 7")
print(3 + 7)
print(3 * "Sieben")
print(3 * 7)
print("3 * 7")
print("3" * 7)
print(3 * '7')
# und etwas schwieriger:
print(2 * "Drei-" + "Sieben!")
print(2 * ("Drei-" + "Sieben! "))
print(3 * (2 * "A" + 3 * "X" + "! "))
print("(2+" + 2 * "2*" + "2)+2")
```

1.3.5 (Zeilenumbruch)**.** Die Zeichenkombination `\n` alias «new line» in einem String sorgt bei der Ausgabe für einen Zeilenumbruch (`\n` ist ein sogenannter «escape characters»). Erkläre, wie Backslash auf Tastatur ✎

```
print('Leitbild der Kanti:\nBegegnen\n Lernen\n  Wachsen')
# Ausgabe:
```

Aufgabe A7 Rechne mit Strings und verwende den «escape character» `\n`.

(a) Schreibe eine einzelne Code-Zeile in Python, die die folgende Ausgabe erzeugt.
Der String `"Ich liebe Informatik!"` darf darin nur einmal vorkommen.

```
Ich liebe Informatik!
Ich liebe Informatik!
Ich liebe Informatik!
```

(b) Schreibe eine weitere Code-Zeile, die die folgende Ausgabe erzeugt.
Innerhalb aller verwendeten Strings darf nur ein einziges Pluszeichen `+` vorkommen.

```
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
```

(c) Schreibe eine weitere Code-Zeile, die die folgende Ausgabe erzeugt.
Innerhalb aller verwendeten Strings dürfen nur zwei Pluszeichen `+` und zwei Minuszeichen `-` vorkommen.

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Einlesen von Tastatureingaben erkläre, wie das in VS Code geht! Terminal fenster anklicken, Eingabe, Enter=Return

1.3.6. Mit der Funktion `input` können Tastatureingaben vom Benutzer entgegengenommen werden.

```
s = input('Bitte gib etwas ein! ')
print(s)
```

Der `input`-Befehl zeigt im Terminal das Argument an (hier `'Bitte gib etwas ein!'`) und wartet danach darauf, dass der Benutzer etwas eingibt und die Eingabetaste (= Return, Enter) drückt.

Die `input`-Funktion liefert als **Rückgabewert** (return value) die Eingabe des Benutzers als String. Dieser Rückgabewert wird im obigen Beispielprogramm in der Variablen `s` gespeichert und dann ausgegeben.

1.3.7. Hier ist ein fehlerhaftes Programm.

```
x = input('Gib eine ganze Zahl ein: ')
print('Das Quadrat deiner Eingabe ist')
print(x*x)                                  # Problem:
```

Lösung des Problems: Ersetze die erste Zeile des obigen Programms durch

```
x = int(input('Gib eine ganze Zahl ein: '))
```

oder durch

```
s = input('Gib eine ganze Zahl ein: ')
x = int(s)
```

1.3.8. Erklärung: Die Funktion `int` wandelt einen String in eine ganze Zahl um (sofern sinnvoll möglich).

Andere nützliche Funktionen zur Umwandlung des Datentyps sind:

- `float` von String zu Float;
- `str` von Integer oder Float zu String.

Beispiel 1.3.9.

```
s = '4.2'
print(2*s)            # Ausgabe:
s = float(s)
print(2*s)            # Ausgabe:
```

Beispiel 1.3.10.

```
x = 4.2
print(2*x)            # Ausgabe:
x = str(x)
print(2*s)            # Ausgabe:
```

Ausgabe von Variablen als Teil von Strings

1.3.11. Nicht besonders elegant:

```
x = 7
print('Die Wurzel von', x, 'ist', x**0.5, '.')
```

Variante mit Typumwandlung mit Hilfe von `str` (vermeidet z. B. das Leerzeichen vor dem Punkt am Satzende):

```
print('Die Wurzel von ' + str(x) + ' ist ' + str(x**0.5) + '.')
```

Deutlich eleganter geht es mit **f-strings** (= «formatted strings») wie folgt. Schreibe erst die gewünschte Ausgabe «naiv» wie folgt:

```
print('Die Wurzel von x ist x**0.5.')         # noch nicht korrekt
```

Dann:

(a) Schreibe den Buchstaben `f` vor das erste Anführungszeichen (f wie f-string).
(b) Klammere alle Werte, die auszugeben sind, mit geschweiften Klammern ein. Erkläre, wie geschweifte Klammern auf Tastatur

Das Ergebnis ist die folgende elegante und lesbare Variante.

```
print(f'Die Wurzel von {x} ist {x**0.5}.')
```

Beispiel 1.3.12. Hier ist ein Beispielprogramm, dass diverse oben erklärte Sachverhalte illustriert (Eingabe, Typumwandlung, Ausgabe per f-string).

Im Wesentlichen handelt es sich um eine Variante von Aufgabe A3 mit Eingabe der Parameter.

```
print("Ich kann quadratische Gleichungen der Form ax^2+bx+c=0 lösen.")
print("Gib die Werte der Parameter nacheinander ein.")

a = float(input("a="))
b = float(input("b="))
c = float(input("c="))

diskriminante = b**2-4*a*c
x1 = (-b+diskriminante**0.5)/(2*a)
x2 = (-b-diskriminante**0.5)/(2*a)

print(f'Die Gleichung {a}x^2+{b}x+{c}=0 hat die Lösungen x1={x1} und x2={x2}.')
```

(Wer sich zurecht daran stört, dass bei negativem `b` oder `c` ein Pluszeichen direkt auf ein Vorzeichen strifft, sollte `{b}` durch `({b})` ersetzen und analog für `c`. Man könnte diese Fälle auch per `if`-Bedingung abfangen.)

Dieses Programm hat noch einen Nachteil: Gibt man für den Parameter `a` Null ein, so stürzt es ab («division by zero»). Sobald wir `if`-Bedingungen kennen, ist dieser Fall einfach zu behandeln.

Aufgabe A8 Schreibe ein Programm, das vom Benutzer die Längen a und b der beiden Katheten eines rechtwinkligen Dreiecks einliest und die Länge c der Hypotenuse ausgibt. Der Dialog soll bei Eingabe von `8` und `15` wie folgt aussehen.

```
Länge der Kathete a: 8
Länge der Kathete b: 15
Die Hypotenuse hat die Länge 17.0.
```

Aufgabe A9 Vorbemerkung: Du kannst die Variable `pi` wie folgt aus der Bibliothek `math` importieren und sie dann als (Näherungswert für) π nutzen.

```
from math import pi
print(pi)              # Ausgabe: 3.141592653589793
```

Schreibe ein Programm, das vom Benutzer den Radius r eines Kreises einliest und mit Hilfe der Variablen `pi` den Umfang $U = 2\pi r$ und die Fläche $A = \pi r^2$ berechnet. Der Dialog soll bei Eingabe von `2.5` wie folgt aussehen.

```
3.141592653589793
Welchen Radius hat der Kreis? 2.5
Der Kreis mit Radius 2.5 hat den Umfang 15.707963267948966
und die Fläche 19.634954084936208.
```

Aufgabe A10 Schreibe ein Programm, das vom Benutzer einen String einliest und diesen dann in einem Rahmen aus Sternen ausgibt.

Der Dialog soll bei Eingabe von `Hello World!` wie folgt aussehen. Der Rahmen aus Sternen soll sich der Länge des eingegebenen Strings jeweils genau anpassen.

```
Gib einen String ein! Hello world!
****************
* Hello world! *
****************
```

Etwas Bonusmaterial zu f-Strings

1.3.13. Wenn man in einem f-String hinter der Variable einen Doppelpunkt schreibt, kann man danach diverse Formatierungen festlegen.Experimentiere mit dem folgenden Code (etwa Zahlen verändern) und versuche herauszufinden, welche Bedeutung die Zeichen nach dem Doppelpunkt haben. Zum Beispiel in Zeile 5: Welche Formatierung bewirkt die 20, welche die 4?

```
# Floats
x = 123**0.5
print(x)
print(f'{x:.4f}')     # das erste f steht für f-string, das zweite für Float
print(f'{x:20.4f}')

# Integers
n = 123
print(n)
print(f'{n:20d}')     # das d steht für dezimal/decimal, also Zehnersystem
print(f'{n:b}')       # das b steht für binär/binary, also Binärsystem

# Strings
s = 'Python'
print(s)
print(f'X{s:<20}X')
print(f'X{s:^20}X')
print(f'X{s:>20}X')
```

1.4 Booleans

1.4.1. Erinnerung: Ein Boolean ist ein Datentyp, der nur die beiden Wahrheitswerte `True` und `False` annehmen kann (also *wahr* oder *falsch*).
Beispiel 1.4.2.

```
erwachsen = True
senior = False
```

In diesem Beispiel sind `erwachsen` und `senior` **Boolesche Variablen** oder kurz **Booleans** (abkürzende Sprechweisen für «Variablen vom Datentyp Boolean»).

Wahrheitswerte als Ergebnisse von Vergleichen

Beispiel 1.4.3. Beim Programmieren tauchen Wahrheitswerte meistens als Ergebnisse von Vergleichen auf.

```
# Vergleiche von Zahlen:
alter = 23
print(alter == 20)                          # Ausgaben:
# Falsch: print(alter = 20)
print(alter != 20)
print(alter <= 30)
print(alter % 2 == 0)
print(alter % 2 != 0)

erwachsen = alter >= 18
print(erwachsen)

print(-10**2 > (-10)**2)

# Vergleiche von Strings
print('Graphik' == 'Grafik')
print('Graphik' == "Graphik")
print('Ananas' < 'Banane')
print('ananas' < 'Banane')
print(len('ananas') < len('Banane'))

# Vergleiche von Booleans
print(True == False)

# Bonuswissen:
print('ananas'.upper())
print('ananas'.upper() < 'Banane'.upper())
```

1.4.4. In Python gibt es die folgenden Zeichen zum Vergleich zweier Werte:

Relation	Symbol in Python	mathematische Notation
Gleichheit	`==` Achtung: Zwei «Gleichheitszeichen»! (Das einfache Gleichheitszeichen `=` wird als Zuweisungszeichen verwendet.)	$=$
Ungleichheit	`!=`	$\neq$
Kleiner	`<`	$<$
Kleiner-gleich	`<=`	$\leq$
Grösser	`>`	$>$
Grösser-gleich	`>=`	$\geq$

«Rechnen» mit Wahrheitswerten: Logische Verknüpfungen

1.4.5. In der Logik rechnet man mit Wahrheitswerten.

- Verneinung (Negation): Man kann einen Wahrheitswert verneinen.
- logisches Und (Konjunktion): Man kann zwei Wahrheitswerte mit «und» verknüpfen.
  Das Ergebnis ist genau dann wahr, wenn ✎

- logisches Oder (Disjunktion): Man kann zwei Wahrheitswerte mit «oder» verknüpfen.
  Das Ergebnis ist genau dann wahr, wenn ✎

Beispiel 1.4.6. Wie diese drei **logischen Verknüpfungen** in Python notiert werden, geht aus dem folgenden Code hervor.

```
# Verneinung:
print(not True)                          # Ausgabe
print(not False)

# logisches Und:
print(False and False)
print(False and True)
print(True and False)
print(True and True)

# logisches Oder:
print(False or False)
print(False or True)
print(True or False)
print(True or True)

# kombinert:
print(False and not (True or not True))

# In Kombination mit Vergleichszeichen. # Ausgabe bei alter = 12,  15,  20
alter = 19
teenager = (alter >= 13) and (alter < 20)
print(teenager)

teenager = (alter >= 13) and not (alter >= 20)
print(teenager)

teenager = not ((alter < 13) or (alter > 19))
print(teenager)
```

Man hätte hier auch einige Klammern weglassen können, denn analog zur «Potenzen-vor-Punkt-vor-Strich-Regel» in der Mathematik gibt es in der Logik die Regel «Nicht-vor-Und-vor-Oder-Regel». Ich empfehle jedoch, lieber einige Klammern zu viel als zu wenig zu setzen.

Aufgabe A11 Die Regel, ob ein Jahr ein Schaltjahr ist, besteht aus den drei folgenden Regeln:

(a) Die durch 4 teilbaren Jahre sind, abgesehen von den folgenden Ausnahmen, Schaltjahre.
(b) Säkularjahre (= Jahre, die ein Jahrhundert abschliessen, z. B. 900, 1000, 1100) sind, abgesehen von der folgenden Ausnahme, keine Schaltjahre.
(c) Die durch 400 teilbaren Jahre sind Schaltjahre.

Der folgende Pytho-Code enthält vier Versuche, um festzustellen, ob die in der Variablen `jahr` gespeicherte (Jahres-)Zahl zu einem Schaltjahr gehört. Welche Versuche liefern stets das korrekte Ergebnis?

```
ist_schaltjahr_A = (jahr % 4 == 0) and (jahr % 100 != 0) and (jahr % 400 == 0)
ist_schaltjahr_B = (jahr % 4 == 0) and ((jahr % 100 != 0) or (jahr % 400 == 0))
ist_schaltjahr_C = (jahr % 4 == 0) or (jahr % 100 != 0) or (jahr % 400 == 0)
ist_schaltjahr_D = (jahr % 400 == 0) or ((jahr % 4 == 0) and (jahr % 100 != 0))
```

Hinweis: Fülle die folgende Tabelle aus (jeder Eintrag ist `True` oder `False`). Zum Beispiel ist in der ersten Spalte der Wert der vier angegebenen booleschen Variablen einzutragen, wenn die Variable `jahr` den Wert 2000 hat. In der letzten Zeile ist anzugeben, ob 2000 ein Schaltjahr ist.

Boolesche Variable	Wert für jahr = 2000	Wert für jahr = 2024	Wert für jahr = 2025	Wert für jahr = 2100
`ist_schaltjahr_A`				
`ist_schaltjahr_B`				
`ist_schaltjahr_C`				
`ist_schaltjahr_D`				
korrektes Ergebnis				

1.5 Datenstrukturen nachträglich ergänzt: = abstrakte Datentypen

1.5.1. Bisher haben wir die wichtigsten nachträglich ergänzt: elementaren Datentypen kennengelernt (Integer, Float, String, Boolean).

Oft möchte man Informationen, die man speichern möchte, strukturiert zusammenfassen. Dazu verwendet man sogenannte **Datenstrukturen** nachträglich ergänzt: = abstrakte Datentypen. Solche Datenstrukturen dienen der effizienten Verwaltung von Daten. Hier sind einige Beispiele.

- Bei einer Schulklasse möchte man die Namen aller Schüler speichern. Hier wirkt es naheliegend, eine Klassenliste anzulegen (die zum Beispiel alphabetisch geordnet sein mag). Die gesamte Liste möchte man dann in einer Variablen speichern und dann zum Beispiel auf den Schüler mit der Nummer 7 zugreifen.
  In Python gibt es hierfür eine Datenstruktur namens **Liste** (= **list**), die genau diesem Zweck dient.
  In unserem Beispiel könnte man etwa eine Variable `klassenliste` anlegen und dann per `klassenliste[7]` den Namen des Schülers mit der Nummer 7 erhalten.
- Den Datentyp String, den wir bereits kennen, kann man auch als Liste von einzelnen Zeichen auffassen. In diesem Sinne ist er auch eine Datenstruktur.
- In unserem ersten Beispiel haben wir von einer Person den Namen, die Grösse, die Mathenote und die Volljährigkeitsinformation gespeichert. Es wäre nett, all diese Personendaten zusammenzufassen und in einer Variablen speichern zu können.
  Dazu dienen in Python sogenannte **dictionaries**. Die Idee ist, dass man in diesen «Wörterbüchern» unter gewissen Begriffen gewisse Daten abfragen kann.
  In unserem Beispiel könnte man eine Variable `person` anlegen und dann durch die Abfrage `person["groesse"]` die Grösse erhalten.
- Wenn man den Stammbaum einer Familie abspeichern möchte, dann sind Beziehungen wie «ist Kind von», «ist Elternteil von» wichtig. Datenstrukturen, die solche Beziehungen abspeichern, werden **Bäume** genannt.
- Eine andere wichtige Datenstruktur sind Mengen, zum Beispiel die Menge der natürlichen Zahlen.

1.6 Listen

Nächstes Mal: Beispiele, wie man eine Spielstellung oder einen RGB-Wert oder ein Bild mit Listen speichert, oder einen Punkt in der Ebene. (oder andere Daten mit anderen Datentypen)

Definition 1.6.1 Liste

Eine **Liste** ist eine Datenstruktur, die eine geordnete endliche Folge von Elementen enthält. (Elemente dürfen mehrfach vorkommen.)
In Python werden Listen durch eckige Klammern definiert (siehe folgendes Beispiel).

Beispiel 1.6.2. Der folgende Code erklärt: Definition einer Liste, Länge der Liste, Zugriff auf Elemente, Ändern von Elementen, Anhängen von Elementen am Ende.

```
# Definition einer Liste (aus Strings)
a = ['St. Gallen', 'Appenzell', 'Herisau', 'Chur']
print(len(a))                # Ausgabe
print(a[2])
print(a[0])
print(a[4])
print('Sangalle' in a)
print('Appenzell' in a)
a[2] = 'Heiden'
print(a)
a.append('Glarus')
print(a)

# leere Liste
b = []
print(b)
b.append(5)
b.append('x')
b.append(b[1] == 'y')
b.append([5.3, 2, True])
print(b)
print(b[3][0])
```

Merke 1.6.3

Das vorderste Element einer Liste in Python hat den Index 0.

Aufgabe A12 Spiele Computer und ermittle jeweils, welche Ausgabe das Programm produziert. (Ohne Computer zu lösen; höchstens danach zur Probe.)

(a)

```
a = [1, 2, 3, 4]
a[1] = a[a[2]] * a[1]
a[0] = a[2] + 10*a[0]
a[2] = a[2] - 1
print(a)
```

(b)

```
x = [0, 1]
x.append(x[0]+x[1])
x.append(x[1]+x[2])
x.append(x[2]+x[3])
x.append(x[3]+x[4])
print(x)
```

1.7 Listen von ganzen Zahlen definieren

1.7.1. Wenn man zum Beispiel die Liste aller ganzen Zahlen von 0 bis 10 definieren möchte, so geht dies mit dem Befehl

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Wenn man die Liste aller ganzen Zahlen von 0 bis 1000000 definieren möchte, geht das mit sehr viel Schreibarbeit genauso, jedoch gibt es eine deutlich elegantere Methode, die wir im nächsten Beispiel erklären.
Beispiel 1.7.2. Mit Hilfe einer Kombination der beiden Befehle `list` und `range` kann man Listen von ganzen Zahlen kreiern.

```
a = list(range(10))            # Ausgabe; Achtung: Die Liste endet bei
print(a)
b = list(range(3, 11))
print(b)
c = list(range(4, 34, 5))
print(c)
# Dabei dürfen natürlich auch Variablen verwendet werden:
start = -100
stop = 1000
step = 3
print(list(range(start, stop, step))
# Bonus-Wissen:
print(list(range(34, 4, -5)))
```

Merke 1.7.3

Eine Liste aller ganzen Zahlen von 0 bis $\boxed{\text{stop} - 1}$ kann man mit dem folgenden Befehl erzeugen.

```
list(range(stop))
```

Wenn die Liste nicht bei 0, sondern bei einer beliebigen ganzen Zahl start starten soll, geht das mit dem folgenden Befehl.

```
list(range(start, stop))
```

Ausserdem kann man noch eine Schrittweite `step` angeben und erhält dann eine Liste, bei der jede Zahl um diese Schrittweite grösser ist als die vorherige Zahl.

```
list(range(start, stop, step))
```

Bei all diesen Befehlen müssen (leider) alle Parameter `start`, `stop`, `step` Integers sein.

1.8 Listen durch «list comprehension» erzeugen

1.8.1. Aus der Mathematik kennen Sie vermutlich die folgende Möglichkeit, eine Menge zu definieren.

$$\{x^2 \mid x \in \mathbb{N} \text{ und } 10 \leq x \leq 20\}$$

Sprechweise: «Die Menge aller x^2, wobei x eine natürliche Zahl zwischen 10 und 20 ist.»

Auf ganz ähnliche Art und Weise kann man in Python Listen definieren (und auch Mengen und Tupel).
Beispiel 1.8.2.

```
a = [x**2 for x in range(10, 21)]
# Man könnte auch statt list(range(10, 21)) schreiben, aber so ist es kürzer.
print(a)
b = [x+0.5 for x in range(-10, 10)]
print(b)
c = [i/2 for i in range(-19, 19)]
print(c)
d = [7 for t in range(6)]
print(d)
e = [10+0.1*t for t in range(6)]
print(e)
```

Merke 1.8.3 list comprehension

Unter **list comprehension** versteht man die Definition einer Liste mit Hilfe eines Ausdrucks der Form
[AUSDRUCK for LAUFVARIABLE in LISTE/BEREICH]
Die beiden Klammern und die Schlüsselworte `for` und `in` müssen genau so angegeben werden.
Die Liste wird dann wie folgt gebildet: Die Laufvariable nimmt der Reihe nach alle Werte der Liste (oder des Bereichs) an; für jeden dieser Werte wird der Ausdruck berechnet; die so der Reihe nach erhaltenen Werte bilden die Elemente der Liste.

lstlisting auskommentiert wegen zoom

Aufgabe A13 In einem Python-Programm:

(a) Definiere mit Hilfe der Befehle `list` und `range` die folgenden Listen und gib sie zur Kontrolle per `print` aus:
- eine Liste `a`, die aus allen natürlichen Zahlen von 0 bis 30 besteht.
- eine Liste `b`, die aus allen ganzen Zahlen von -10 bis 20 besteht.
- eine Liste `c`, die aus den natürlichen Zahlen 10, 13, 16, 19, 22, 25 besteht.

(b) Definiere nun dieselben drei Listen durch list comprehension. Ersetze dazu alle Fragezeichen im folgenden Code-Fragment durch geeignete Ausdrücke und gib die drei Listen per `print` aus.
Damit es nicht zu leicht wird, darf das Fragezeichen im `range`-Befehl durch nur eine Zahl (= den Stoppwert) ersetzt werden (Startwert und Schrittweite dürfen also nicht angegeben werden).

```
aa = [? for ? in range(?)]
bb = [? for ? in range(?)]
cc = [? for ? in range(?)]
```

(c) Definiere mit list comprehension die Liste $\sqrt[1]{1}$, $\sqrt[2]{2}$, …, $\sqrt[9]{9}$, $\sqrt[10]{10}$ aller Wurzeln $\sqrt[n]{n}$.

(d) Definiere mit list comprehension die Liste aller Kubikzahlen (= dritter Potenzen) von $1 = 1^3$ bis 20^3 und gib sie aus.

(e) Schreibe ein Programm, das vom Benutzer zwei natürliche Zahlen n und $k > 0$ einliest. Das Programm soll dann die Liste aller k-ten Potenzen von 1^k bis n^k ausgeben.

1.9 Graphen zeichnen mit `matplotlib` (= mathematical plotting library)

Aufgabe A14

(a) Tippe das folgende Programm ab (ohne Kommentare), führe es aus und versuche es zu verstehen.
(Vermutlich musst du die «mathematical plotting library» `matplotlib` installieren. Dass sollte im Terminal von Visual Studio Code mit dem Befehl `pip install matplotlib` (gefolgt von Enter/Return) funktionieren.)

```
import matplotlib.pyplot as plt      # Import der Bibliothek matplotlib
plt.gca().set_aspect('equal')        # Selbe Einheitslänge in x- und y-Richtung.
xwerte = list(range(-3, 4))
ywerte = [x**2 for x in xwerte]
print(xwerte)
print(ywerte)
plt.plot(xwerte, ywerte)     # Was ist der Unterschied zwischen plot und scatter?
plt.scatter(xwerte, ywerte)  # Zeile auskommentieren zum Herausfinden!
plt.show()
```

(b) Um die Knicke im Graphen zu glätten: Ändere das Programm so, dass statt der x-Werte von -3 bis 3 in Einserschritten alle x-Werte von -3 bis 3 in Schritten von 0.1 verwendet werden.

(c) Ergänze das Programm so, dass zusätzlich der Graph der Funktion $y = f(x) = \frac{1}{2}x + 2$ gezeichnet wird.

(d) Ergänze ganz am Anfang des Programms die Zeile

```
from math import sin, cos
```

Dann stehen die Kosinus- und die Sinusfunktion zur Verfügung. Zeichne die Graphen beider Funktionen über dem Intervall $[-7, 7]$ zusätzlich ein.

(e) Bonus: Finde heraus, wie man Exponentialfunktion $f(x) = e^x$ und Logarithmusfunktion $g(x) = \ln(x)$ in Python schreibt, und zeichne sie auf geeigneten Intervallen.

1.10 Slicing («Scheiben herausschneiden») – Teillisten von Listen extrahieren

1.10.1. Man kann aus einer Liste gewisse Teillisten extrahieren. Wir geben zuerst Codebeispiele und erklären die Syntex danach abstrakt.
Beispiel 1.10.2. In diesem Beispiel definieren wir gewisse Teillisten. Hinter jeder Zuweisung an eine Variable ist der neue Wert aufzuschreiben (entweder gemeinsam in der Lektion oder durch Ausprobieren am Computer).

```
a = list(range(7, 18))  # Wert von a:
b = a[4:9]
c = a[1:7]
d = a[1:7:2]
# Kurzformen: Statt
e = a[0:6]
# darf schreiben (Teilliste startet automatisch beim Index 0)
e = a[:6]
# Statt
f = a[5:len(a)]
# darf schreiben (Teilliste endet automatisch am Listenende)
f = a[5:]
# Wenn man die Schrittweite angibt,
# darf man Startindex oder Stoppindex weglassen.
g = a[:6:2]
h = a[5::2]
i = a[::2]
```

Merke 1.10.3

Möchte man aus einer Liste `liste` eine Teilliste extrahieren, die beim Index `start` beginnt und beim Index `stop`-1 endet, so geht dies mit dem Befehl

```
liste[start:stop]
```

Wenn man hierbei `start` bzw. `stop` nicht angibt (aber den Doppelpunkt schreibt), startet die Teilliste am Listenanfang bzw. endet am Listenende (d. h. für `start` wird der Standardwert («default value») `0` verwendet, für `stop` der Standardwert `len(liste)`).
Zusätzlich kann man als drittes Argument nach einem weiteren Doppelpunkt die «Schrittweite» `step` der gewünschten Teilliste angeben.

```
liste[start:stop:step]
```

Auch hier darf man die Argumente `start` und `stop` weglassen.

1.10.4. Man beachte die Ähnlichkeit der folgenden Python-Ausdrücke – jeweils werden Startwert/Startindex, Stoppwert/Stoppindex und Schrittweite angegeben.

- `list(range(10, 100, 5))` erzeugt eine Liste ganzer Zahlen von 10 bis 95 mit Schrittweite 5.
- `a[10:100:5]` liefert die Teilliste einer Liste `a`, die beim Index 10 startet und mit Schrittweite 5 bis zum Index 95 geht.
- (Erklärung im folgenden Abschnitt 1.11)
  `s[10:100:5]` liefert den Teilstring eines Strings `s`, der beim Index 10 startet und mit Schrittweite 5 bis zum Index 95 geht.

1.11 Nachtrag zu Strings: Zugriff auf einzelne Zeichen und Slicing

1.11.1. Der Zugriff auf einzelne Zeichen eines Strings erfolgt «genauso» wie der Zugriff auf die Elemente einer Liste. Wir erklären dies im folgenden Beispiel.
Beispiel 1.11.2. Schreibe jeweils die Werte der Variablen hinter die Zuweisungen.

```
s = 'Hello World!'
c = s[7]          # Wert von c:
d = s[len(s)]     # Wert von d:
e = s[len(c)]     # Wert von e:
```

1.11.3. Slicing bei Strings funktioniert genau wie bei Listen, wie das folgende Beispiel zeigt.

Beispiel 1.11.4. Schreibe jeweils die Werte der Variablen hinter die Zuweisungen. gut wäre: `x = s[2:3] # Wert von x: y = s[2] # Wert von y:`

```
s = 'Hello World!'
a = s[2:7]   # Wert von a:
b = s[:7]    # Wert von b:
c = s[6:]    # Wert von c:
d = s[3:11:2]  # Wert von a:
```

Aufgabe A15 Ermittle die Ausgabe des folgenden Programms (zuerst ohne Rechner; dann mit dem Rechner kontrollieren).

```
s = 'Kulturbanause'
print(s[0])
print(s[len(s)-1])
print(s[4:9])
print(s[:4])
print(s[len(s)-3:])
t = 'einparteisystem'
print(t[3::2])
t = 'frauenredakteurin'
print(t[4::2])
t = 'milliardenunternehmen'
print(t[::4])
t = 'jugendsexualität'
print(t[::3])
t = 'innenstadtviertel'
print(t[5::2])
# Bonuswissen: Mit negativen Indizes kann man
# in Strings (und Listen) Elemente adressieren.
x = 'Kulturbanause'
print(x[-1])
print(x[-3:])
```

Aufgabe A16 Im folgenden Programmcode sind die Zeilen mit den `print`-Befehlen so zu ergänzen, dass das darüber angegebene Wort ausgegeben wird. Dabei ist Slicing zu verwenden. Löse die Aufgabe zuerst von Hand und teste deine Befehle dann am Computer.

```
s = 'Donaudampfschifffahrt'
# dampf
print(s[

s = 'Versuchung'
# Versuch
print(s[

s = 'kantonsschule'
# antonsschule
print(s[

s = 'erbverzichtsvertrag'
# reiter
print(s[

s = 'kirchenstaatsvertrag'
# retter
print(s[

s = 'rohrfederzeichnung'
# hering
print(s[

s = 'kapitalanlagefirma'
# killer
print(s[
```

1.12 Lernkontrolle (und Vorbereitung auf die Prüfung)

Aufgabe A17 Datentypen (elementare Datentype und abstrakte Datentypen = Datenstrukturen):

(a) Nenne vier elementare Datentypen und gib jeweils ein Beispiel, wie man in Python eine Variable des entsprechenden Datentyps definiert (zum Beispiel gibt es zwei numerische Datentypen zum Speichern von Zahlen).

(b) Welcher Python-Befehl gibt den Typ (= Datentyp) einer Variablen `x` aus?

(c) Nenne eine Datenstruktur (= einen abstrakten Datentyp) und gib ein Beispiel, wie man in Python eine Variable dieses Typs definiert.

Aufgabe A18 Rechnen mit numerischen Datentypen (integer, float):

(a) Wie berechnet man in Python:
- 5^{1234}
- $\sqrt{2357}$
- den Rest der Division von 1234567890 durch 235?
- den Ganzzahlquotient der Division von 1234567890 durch 235?
- das Ergebnis der Division von 1234567890 durch 235 (als Kommazahl)?

(b) Wenn `n` eine Zahl-Variable in Python ist. Definiere eine Variable `gauss` mit Wert

$$\frac{n(n+1)}{2}$$

(Gaußsche Summenformel: $1+2+3+\cdots+n=\frac{n(n+1)}{2}$.)

(c) Wie schreibt man die Formel

$$V=\frac{1}{3}h\left(G+\sqrt{GD}+D\right)$$

in Python, wenn `h`, `G` und `D` Zahlvariablen sind?
(Diese Formel berechnet das Volumen eines Pyramidenstumpfs der Höhe h mit unterer Grundfläche G und oberer Grundfläche D.)

(d) Welche Ausgabe liefert das folgende Programm?

```
a = 5
b = 7
print(a, b)
a = (a**2)*b
b = a/b
a = a/b
print(a, b)
```

Aufgabe A19 Rechnen mit Strings, Ein- und Ausgabe:

(a) Welche Ausgabe liefert das folgende Programm?

```
s = "A"
t = "-"
s = 3*s
print(s)
s = t+s+t
print(s)
print(len(s)*s)
s = len(s)*'X' + '\n' + s
print(s)
```

(b) Schreibe ein Programm, das vom Benutzer zwei natürliche Zahlen x und y einliest und den Rest der Division von x durch y ausgibt. Der Dialog mit dem Benutzer soll bei Eingabe von `43` und `4` wie folgt aussehen.

```
Wert von x: 43
Wert von y: 4
Der Rest der Division von 43 durch 4 ist 3.
```

Aufgabe A20 Rechnen mit Wahrheitswerten (logische Verknüpfungen, Booleans):

(a) Welche Ausgabe liefert das folgende Programm?

```
print(True or False)
print(True and False)
x = 208
print(3 < x and x < 208)
print(x % 2 == 0)
```

(b) Schreibe ein Programm, das vom Benutzer eine natürliche Zahl x einliest. Das Programm soll `True` ausgeben, wenn x durch 2 teilbar ist, aber nicht durch 3; sonst soll `False` ausgegeben werden. Der Dialog mit dem Benutzer soll bei Eingabe von `10` (und jeder anderen, durch 2, aber nicht durch 3 teilbaren Zahl) wie folgt aussehen.

```
Wert von x: 10
True
```

Bei Eingabe von `18` (und jeder durch 2 und durch 3 teilbaren Zahl und jeder nicht durch 2 teilbaren Zahl) soll er wie folgt aussehen.

```
Wert von x: 18
False
```

Aufgabe A21 Manipulationen von Listen und Listen erzeugen:

(a) Welche Ausgabe liefert das folgende Programm?

```
a = [2, 3]
a.append(10)
print(a)
a.append(a[0]*a[1])
print(a)
a[1] = a[a[1]]
print(a)
```

(b) Erzeuge eine Liste `b`, die aus allen natürlichen Zahlen von 10 bis 100 in aufsteigender Reihenfolge besteht (jeweils einschliesslich).

(c) Erzeuge eine Liste `c`, die aus allen geraden natürlichen Zahlen von 10 bis 100 in aufsteigender Reihenfolge besteht (jeweils einschliesslich).

(d) Erzeuge eine Liste `d`, die aus den Quadraten aller natürlichen Zahlen von 1 bis 100 besteht (die letzte Zahl der Liste ist also $100^2 = 10'000$).

Aufgabe A22 Slicing bei Listen und Strings:

(a) Slicing bei Listen: Welche Ausgabe liefert das folgende Programm?

```
a = [3, 7, 2, 8, -4, 22, 5, 4]
print(a[2])
print(a[2:6])
print(a[2:])
print(a[:6])
print(a[2:7:3])
```

(b) Slicing bei Strings: Welche Ausgabe liefert das folgende Programm?

```
s = 'theaterredakteurin'
print(s[4])
print(s[4:9])
print(s[5::2])
```

2 Kontrollstrukturen

2.0.1. Kontrollstrukturen in der Informatik legen fest, in welcher Reihenfolge Anweisungen ausgeführt werden. Anweisungen sind grob gesagt Zeilen oder Blöcke aus Zeilen eines Computerprogramms (etwa eine Zuweisung an eine Variable wie `a = a+3` oder eine Befehl wie `print('Hello World')`).

Merke 2.0.2 Kontrollstrukturen

Moderne Programmiersprachen verwenden die folgenden **Kontrollstrukturen**:

- **Sequenz**: Anweisungen werden hintereinander (linear) ausgeführt (das kennen wir schon).
- **`if`-Statement = bedingte Anweisung**: Durch eine Bedingung wird gesteuert, dass gewisse Anweisungen einmalig ausgeführt werden.
  - **`if`-Statement**: Wenn eine Bedingung erfüllt ist, werden gewisse Anweisungen ausgeführt.
  - **`if-else`-Statement** = Verzweigung: Wenn eine Bedingung erfüllt ist, werden gewisse Anweisungen ausgeführt, sonst gewisse andere Anweisungen.
- **Schleife** = Loop = Iteration: Gewisse Anweisungen werden wiederholt ausgeführt, und zwar meistens gemäss einer der drei folgenden Arten:
  - solange eine Bedingung erfüllt ist (`while`-loop);
  - bis eine Bedingung erfüllt ist (`repeat until`-loop, in Python per `while True`-loop mit Abbruch durch `break`);
  - bis alle Elemente einer Liste abgearbeitet sind (`for`-loop).
- Blockstruktur = **Block**: Ein aus Anweisungen bestehender zusammengehöriger Abschnitt.
  In Python werden Blöcke durch Einrückungen (= Leerschläge am Zeilenbeginn) gebildet.
  Wenn in den vorherigen Kontrollstrukturen von «gewissen Anweisungen» die Rede war, ist damit stets ein Block gemeint.
- **Funktion**: Ein Unterprogramm, das einen Rückgabewert hat oder nicht.

Merke 2.0.3 Schlüsselwörter = key words

Schlüsselwörter (= **key words**) einer Programmiersprache sind fest vorgegebene Wörter, die in dieser Sprache eine gewisse Bedeutung haben und deswegen nicht als Namen von Variablen o. ä. verwendet werden dürfen.
Wichtige Schlüsselwörter in Python sind `if`, `else`, `while`, `for` zum Einleiten von Kontrollstrukturen.

2.1 `if`-(`else`-)Statements = bedingte Anweisungen

Beispiel 2.1.1. Ein Programm mit einem `if`-Statement.

```
x = float(input('Gib eine reelle Zahl ein: '))
if x >= 0:
    print(f'Die von dir eingegebene Zahl {x} ist positiv oder Null.')
    print(f'Deswegen kann ich ihre Wurzel berechnen. Sie beträgt {x**0.5}.')
print('Hier geht das Programm weiter.')
```

Das eigentliche `if`-Statement besteht nur aus den drei mittleren Zeilen.

```
if x >= 0:
    print(f'Die von dir eingegebene Zahl {x} ist positiv oder Null.')
    print(f'Deswegen kann ich ihre Wurzel berechnen. Sie beträgt {x**0.5}.')
```

- Die erste dieser Zeilen ist die sogenannte **Kopfzeile** (= **header**) des `if`-Statements und leitet dieses ein. Die Kopfzeile besteht aus dem **Schlüsselwort** (= **key word**) `if` am Anfang und einem Doppelpunkt `:` am Ende; dazwischen muss eine Bedingung stehen, also ein Ausdruck, der einen Wahrheitswert (= Boolean) liefert.
- Die nachfolgenden zwei Zeilen bilden einen Block; das sieht man daran, dass diese Zeilen um 4 Leerzeichen (= Leerschläge) eingerückt sind.
  Dieser Block ist der **Rumpf** (= **body**) des `if`-Statements. Er muss aus mindestens einer Zeile bestehen.
- Trifft die Bedingung in der Kopfzeile (zum Zeitpunkt des Programmablaufs) zu, so wird der Rumpf genau einmal abgearbeitet, sonst wird er übersprungen.

Beispiel 2.1.2. Eine Erweiterung des `if`-Statements ist das `if-else`-Statement:

```
x = float(input('Gib eine reelle Zahl ein: '))
if x >= 0:
    print(f'Die von dir eingegebene Zahl {x} ist positiv oder Null.')
    print(f'Deswegen kann ich ihre Wurzel berechnen. Sie beträgt {x**0.5}.')
else:
    print(f'Die von dir eingegebene Zahl {x} ist negativ.')
    print(f'Ihr Betrag ist {-x}.')
print('Hier geht das Programm weiter.')
```

Neu hinzugekommen sind drei Zeilen:

- die Zeile mit dem Schlüsselwort `else` samt Doppelpunkt;
- die beiden nachfolgenden, eingrückten Zeilen, die einen Block bilden.

Wenn die Bedingung nach dem Schlüsselwort `if` in der Kopfzeile zutrifft, wird der «`if`-Block» abgearbeitet, sonst der «`else`-Block».

⚒ Aufgabe A23

```
minuten = int(input("Wie viele Minuten Tageslicht hattest du heute schon? "))
#
# Hier ist Code zu ergänzen.
#
```

Ergänzen Sie das obige Programm so, dass es wie folgt mit dem Benutzer kommuniziert.

Falls der Benutzer 120 oder mehr antwortet, soll er gelobt werden. Sonst wird ihm gesagt, wie viele Minuten er noch draussen verbringen sollte, damit er mindestens 2 Stunden Tageslicht abbekommt.

Beispiel: Der Benutzer gibt ein, dass er 35 Minuten Tageslicht hatte. Dann soll der Computer Folgendes ausgeben:

```
Bitte gehe noch 85 Minuten nach draussen.
```

Testen Sie Ihr Programm mit Eingaben wie 30, 170, −10, 2400.

Hintergrundinformation: Zwei Stunden Tageslicht pro Tag minimieren laut wissenschaftlicher Studien das Risiko, kurzsichtig zu werden, siehe Deutschlandfunk Kultur: https://www.deutschlandfunkkultur.de/sehstoerung-des-auges-immer-mehr-kinder-werden-kurzsichtig-100.html

⚒ Aufgabe A24

Das folgende Programm erzeugt zwei Zufallszahlen zwischen 2 und 20.

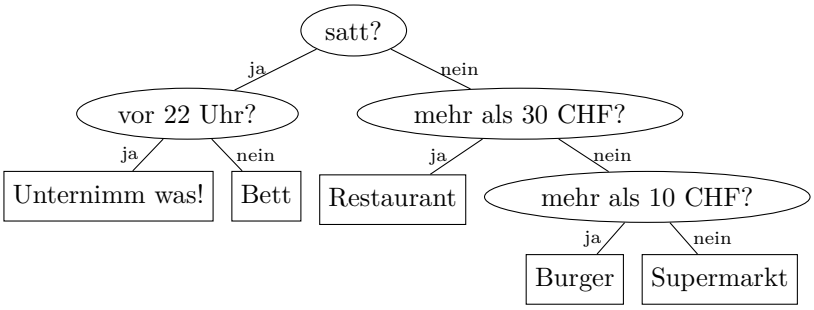
```
import random
x = random.randrange(2, 21)
y = random.randrange(2, 21)
print(x)
print(y)
```

Erweitere dieses Programm so, dass der Computer den Benutzer nach dem Produkt der Variablen `x` und `y` fragt und ihm mitteilt, ob er richtig gerechnet hat. Der Dialog mit dem Computer sollte sinngemäss so aussehen:

```
Was ist das Produkt von 5 und 7? 24
Dies ist leider falsch. Richtig wäre 35 gewesen.
```

Bonus (eine «sinnvolle» Lösung benötigt eine Schleife): Es gibt viele Erweiterungsmöglichkeiten. Etwa könnten dem Benutzer 10 Multiplikationsaufgaben gestellt werden und es wird ihm danach mitgeteilt, wie viel Prozent der Aufgaben er richtig gelöst hat. Es könnten auch zufällig Divisionsaufgaben (ohne Rest) eingestreut werden.

2.1.3. Entscheidungsbäume stellen Entscheidungsregeln graphisch dar. Hier ein Beispiel.



Solche Fallunterscheidungen kann man in Python mit Hilfe von ineinander verschachtelten `if-else`-Statements durchführen. Hier als Beispiel der obige Baum als Python-Programm – man beachte, dass die Einrückungen (= Leerschläge an den Zeilenanfängen) äusserst wichtig sind.

```
if satt:
    if uhrzeit < 22:
        print('Unternimm was.')
    else:
        print('Geh schlafen.')
else:
    if geld > 30:
        print('Geh ins Restaurant.')
    else:
        if geld > 10:
            print('Kauf einen Burger.')
        else:
            print('Geh in den Supermarkt.')
```

Bonus-Wissen: Die Finanzentscheidung in den letzten 7 Zeilen des obigen Programms kann man in Python auch etwas kürzer wie folgt mit Hilfe des Schlüsselworts `elif`, einer Verschmelzung von `else` und `if`, schreiben.

```
    if geld > 30:
        print('Geh ins Restaurant.')
    elif geld > 10:
        print('Kauf einen Burger.')
    else:
        print('Geh in den Supermarkt.')
```

Auch der mehrfache Einsatz von `elif` ist erlaubt (also etwa `if-elif-elif-elif-else`), was bei Fallunterscheidungen mit sehr vielen Alternativen das Programm deutlich übersichtlicher macht.

⚒ Aufgabe A25 Schreibe ein Programm, das vom Benutzer eine Temperaturangabe einliest und diesem dann mitteilt, ob es kalt oder kühl oder warm oder heiss ist. In welchem Temperaturbereich welches Temperaturempfinden eintritt, ist dir überlassen.

Das Programm soll sich an einem Entscheidungsbaum mit zwei Ebenen orientieren (also eine Ja-Nein-Entscheidung am Anfang und dann jeweils eine weitere Ja-Nein-Entscheidung).

Teste dein Programm: Für jeden Temperaturbereich ist mindestens ein Test durchzuführen.

⚒ Aufgabe A26 Ergänzen Sie das unten angegebene Programm so, dass es alle Lösungen der linearen Gleichung

$$ax + b = 0$$

ausgibt. Im Fall $a \neq 0$ ist die Gleichung leicht zu lösen, im Fall $a = 0$ muss man zwei weitere Fälle unterscheiden.

Testen Sie Ihr Programm zumindest mit den folgenden Gleichungen:

- $0x + 0 = 0$: alle reellen Zahlen sind Lösungen, Lösungsmenge $\mathbb{L} = \mathbb{R}$.
- $0x + 42 = 0$: keine Lösung, $\mathbb{L} = \varnothing$.
- $7x + 3 = 0$: genau eine Lösung $-\frac{3}{7}$, $\mathbb{L} = \left\{-\frac{3}{7}\right\}$.

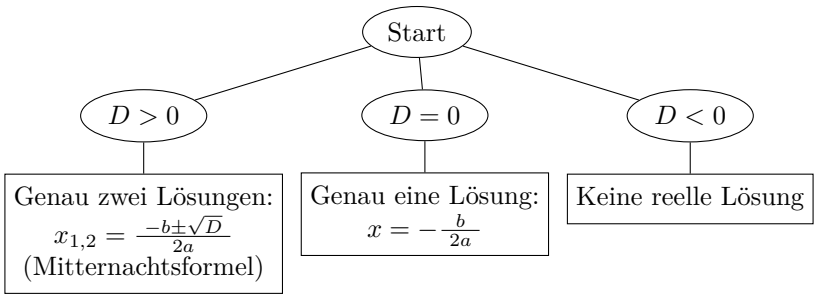
```
print("Ich kann jede lineare Gleichung ax+b=0 lösen.")
a = int(input("Gib eine Zahl a ein: "))
b = int(input("Gib eine Zahl b ein: "))
print(f"Die Lösungen der Gleichung {a}x+{b}=0 sind ...")
# Hier ist Code zu ergänzen.
```

Aufgabe A27 (Lösungen einer quadratischen Gleichung in allen Fällen angeben)

Alle Lösungen einer quadratischen Gleichung

$$ax^2 + bx + c = 0$$

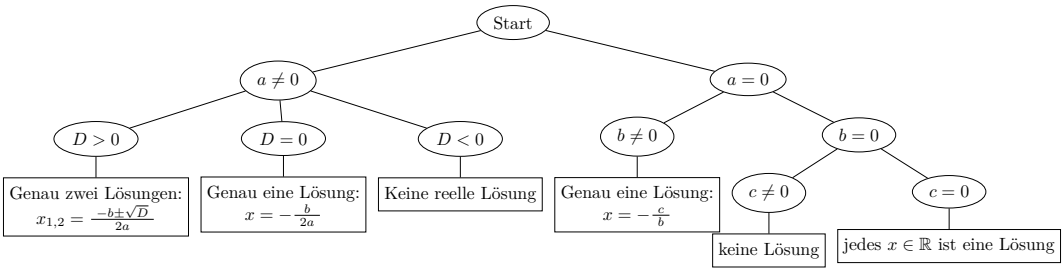
mit $a \neq 0$ kann man mit Hilfe des folgenden Entscheidungsbaumes bestimmen. Darin ist $D = b^2 - 4ac$ die sogenannte **Diskriminante**, die darüber entscheidet, wie viele relle Lösungen die Gleichung hat.



(a) Ergänzen Sie das folgende Programm so, dass es ausgibt, wie viele Lösungen die Gleichung hat. Ausserdem sollen alle Lösungen ausgegeben werden und zur Probe soll für jede Lösung der Wert von $ax^2 + bx + c$ zur Probe ausgerechnet und ausgegeben werden.
Testen Sie Ihr Programm mit mindestens drei quadratischen Gleichungen, die alle Fälle abdecken, etwa $(x-2)(x+3) = 0$ (genau zwei Lösungen), $(x+7)^2 = 0$ (genau eine Lösung), $(x-2)^2 = -3$ (keine Lösung).

```
print("Ich kann jede quadratische Gleichung ax^2+bx+c=0 lösen.")
a = int(input("Gib die Zahl a ein: "))
b = int(input("Gib die Zahl b ein: "))
c = int(input("Gib die Zahl c ein: "))
D = b**2 - 4*a*c                                  # Diskriminante D
print(f'Die Lösungen der Gleichung {a}x^2+{b}x+{c}=0 sind ...')
# Hier ist Code
# zu ergänzen.
```

(b) Bonus: Schreiben Sie das Programm nun so um, dass es auch im Fall $a = 0$ funktioniert. In diesem Fall ist also die lineare Gleichung $0x^2 + bx + c = bx + c = 0$ zu untersuchen.
Testen Sie Ihr Programm! Funktioniert es auch korrekt für die «neuen» Fälle, also etwa für die Gleichungen $3x - 2 = 0$ (genau eine Lösung), $0x - 2 = 0$ (keine Lösung), $0x + 0 = 0$ (unendliche viele Lösungen, Lösungsmenge $\mathbb{R}$)?



2.2 `while`-Schleifen

Beispiel 2.2.1. Ein Programm mit einer `while`-Schleife.

```
x = 39
while x > 0:
    print(x)
    x = x // 2
print('Hier geht das Programm weiter.')
```

Die formale Struktur einer `while`-Schleife ist nahezu identisch mit der Struktur eines `if`-statements. Der einzige Unterschied ist die Verwendung des Schlüsselworts `while` statt `if` in der Kopfzeile.

Bei der Programmausführung wird beim Erreichen der Kopfzeile einer `while`-Schleife die dortige Bedingung geprüft. Ist sie wahr, so wird der eingerückte Code-Block ausgeführt; danach wird erneut die Bedingung in der Kopfzeile geprüft. Ist sie wahr, wird der eingerückte Code-Block ausgeführt; danach wird erneut die Bedingung in der Kopfzeile geprüft usw.

Sobald die Bedingung in der Kopfzeile nicht wahr ist, so wird der eingerückte Code-Block nicht mehr ausgeführt und es geht weiter mit der Programmzeile nach dem eingerückten Code-Block.

⚒ Aufgabe A28

(a) Ohne Computer: Welche Ausgabe erzeugt das folgende Programm?

```
summe = 0
zwischenergebnisse = []
i = 1
while i < 10:
    print(i)
    summe = summe + i**2
    zwischenergebnisse.append(summe)
    i = i + 2
print(zwischenergebnisse)
print(summe)
```

(b) Schreibe ein Programm, dass die Summe aller natürlichen Zahlen von 1 bis 100 berechnet und ausgibt.

Lösung: 5050

(c) Schreibe ein Programm, dass alle ungeraden natürlichen Zahlen von 1 bis 20 aufsummiert. Ähnlich wie im obigen Beispielprogramm ist sowohl eine Liste aller Zwischenergebnisse auszugeben als auch das Endergebnis. Was fällt dir auf?

(d) Für welches n ist die Summe $1+4+9+16+25+\cdots+n^2$ der ersten Quadratzahlen erstmals grösser als $1'000'000$? Schreibe ein Programm, dass dieses n ermittelt.

⚒ Aufgabe A29

Die Collatz-Folge ist wie folgt definiert, ausgehend von einer beliebigen natürlichen Zahl $n > 0$.

- Solange $n >$ gilt, mache das Folgende:
  - Wenn n gerade ist: Ersetze n durch $\frac{n}{2}$.
    Sonst: Ersetze n durch $3n+1$.

Beispiel: Die mit 13 startende Collatz-Folge sieht so aus: $13, 40, 20, 10, 5, 16, 8, 4, 2, 1$

Passen Sie das folgende Programm so an, dass es ausgehend von einer Startzahl die zugehörige Collatz-Folge ausgibt.

```
n = int(input("Gib eine natürliche Zahl > 0 ein: "))
# Hier ist Code zu ergänzen.
```

Bemerkung: Die Collatz-Vermutung besagt, dass jede Collatz-Folge irgendwann bei 1 ankommt. Sie ist bisher nicht bewiesen.

⚒ Aufgabe A30

(Finanzmathematik) Für die Altersrente zahlt Frau Huber jedes Jahr am 1. Januar den Betrag 6000.- auf ein Sparkonto ein. Dem Konto wird am 31. Dezember jeweils ein Zins von 1% gutgeschrieben.

(a) Wie gross ist der Kontostand nach 35 Jahren?
Schreiben Sie ein Programm, das als erstes die drei Variablen `betrag=6000`, `zins=0.01` und `laufzeit=35` definiert und dann für jedes Jahr den Kontostand am 31. Dezember ausgibt.

(b) Fertige eine Kopie deines Programms an und verändere es so, das es herausfindet, wie lange Frau Huber sparen muss, um einen Betrag von 1'000'000.- auf dem Konto zu haben.

Aufgabe A31 (Wurzel einer Zahl berechnen, ohne die Wurzelfunktion von Python zu verwenden)

Die Wurzel $\sqrt{x}$ einer positiven Zahl x kann man näherungsweise berechnen, indem man zwei Variablen `links` und `rechts` so initialisiert, dass `links` $\leq \sqrt{x} \leq$ `rechts` gilt. Der gesuchte Wert $\sqrt{x}$ ist also «zwischen `links` und `rechts` eingesperrt». Zum Beispiel kann man `links` $= 0$ und `rechts` $= x$ setzen.

Nun berechnet man den Mittelwert `mittel` $= \frac{\texttt{links}+\texttt{rechts}}{2}$ und vergleicht sein Quadrat `mittel`^{2} mit x. Je nachdem, wie dieser Vergleich ausfällt, ersetzt man `links` oder `rechts` durch `mittel`, so dass nach der Ersetzung wieder `links` $\leq \sqrt{x} \leq$ `rechts` gilt.

Diesen Prozess wiederholt man solange, bis die Differenz `rechts` $-$ `links` kleiner als ein gewünschter Fehler ist.

Ergänzen Sie das folgende Programm so, dass der oben beschriebene Algorithmus realisiert wird!

```
x = 40
fehler = 0.000001
links = 0
rechts = x

while rechts - links > fehler:
    #
    # Hier ist Code zu schreiben.
    #
schaetzung = (links + rechts) / 2
print(f'{schaetzung} ist ungefähr die Wurzel aus {x}.')
print(f'{x**0.5} ist laut Python die Wurzel aus {x}.')
print(f'Der Fehler ist etwa {abs(mittel-x**0.5):.10f}.')
```

Bemerkung: Das beschriebene Verfahren heisst **Intervallschachtelung**, da $\sqrt{x}$ immer besser durch das kleiner werdende Intervall [`links`, `rechts`] eingeschachtelt wird.

Aufgabe A32 Das unten angegebene Programmgerüst ermittelt in der zweiten Zeile eine natürliche Zufallszahl zwischen 0 und 100 und speichert sie in der Variablen `zahl`.

```
from random import randrange
zahl = randrange(101)
print("Ich habe mir eine natürliche Zahl zwischen 0 und 100 ausgedacht.")
print("Welche Zahl ist es?")
versuche = 0
geratene_zahl = -100    # Absichtlich falsch, damit die while-Schleife
                        # mindestens einmal betreten wird.
while geratene_zahl != zahl:
    #
    # Hier Code ergänzen
    #
print("Benötigte Versuche:", versuche)
```

Ergänzen Sie die Vorlage so, dass es den Benutzer die Zahl raten lässt. Das Programm gibt bei jedem Tipp an, ob die Zahl zu klein, zu gross oder korrekt ist. Ein Beispieldialog könnte so aussehen:

```
Ich habe mir eine natürliche Zahl zwischen 0 und 100 ausgedacht.
Welche Zahl ist es?
Dein Tipp: 21
Zu klein.
Dein Tipp: 84
Zu gross.
Dein Tipp: 42
Korrekt. Herzlichen Glückwunsch!
Benötigte Versuche: 3
```

Beispiel 2.2.2. Geographie-Quiz gemeinsam geschrieben:

- Liste definieren, deren Einträge Listen der Form `[Land, Hauptstadt]` sind.
- `while True`-Endlosschleife, die bei Eingabe von `exit` per `break` beendet wird.
  (Auf diese Art schreibt man in Python Schleifen mit Abbruchbedingung am Ende (oder anderswo), die in anderen Sprachen bisweilen als `repeat-until`-Schleifen existieren.)
- Zufällig wird nach einer Hauptstadt zu einem Land gefragt oder umgekehrt.
- Die Antwort des Benutzers wird bestätigt oder korrigiert.
- Am Ende wird die Anzahl der Fragen und der korrekten Antworten ausgegeben.

2.3 `for`-Schleifen (`for` loops)

2.3.1. Oft werden in einem Programm gewisse Anweisungen nacheinander für alle Elemente einer Liste abgearbeitet. Eine komfortable Möglichkeit, dies zu erreichen, bieten `for`-Schleifen.
Beispiel 2.3.2. (`for`-Schleife über Liste)

Ein Programm mit einer `for`-Schleife über eine Liste.

```
stadtliste = ['Genf', 'Wien', 'Berlin']
for stadt in stadtliste:
    print(stadt)
    print(f'Die Einwohner von {stadt} heissen {stadt}er.')
print('Hier geht das Programm weiter.')
```

2.3.3. Die Kopfzeile jeder `for`-Schleife besteht zuerst aus dem Schlüsselwort `for`, danach folgt der Name einer noch nicht verwendeten Variablen, der sogenannten **Laufvariablen**, dann folgt das Schlüsselwort `in`, dann eine Liste und am Ende ein Doppelpunkt.

Der Rumpf jeder `for`-Schleife besteht aus einem eingerückten Code-Block, der aus mindestens einer Zeile besteht.

Die Laufvariable nimmt nacheinander alle Elemente der Liste als Wert an und führt jeweils den Rumpf der `for`-Schleife aus.
2.3.4. `for`-Schleifen können nicht nur über Listen gehen, sondern auch über andere listenartige Datenstrukturen («iterables») wie zum Beispiel

- Zahlenbereiche (`range`),
- Strings.

Beispiel 2.3.5. (`for`-Schleife über einen `range` (= Zahlenbereich))

Ein Programm mit einer `for`-Schleife über einen `range`.

```
summe = 0
n = 10
for i in range(n+1):
    print(i)
    summe = summe + i
print(f'Die Summe der Zahlen von 1 bis {n} ist {summe}.')
```

Der `range` darf auch durch Startwert, Endwert und Schrittweite angegeben werden.

```
for i in range(2, 15, 3):
    print(i)
```

2.3.6 (Mehrfaches Wiederholen desselben Code-Blocks)**.** Wenn man genau dieselben Anweisungen mehrfach ausführen möchte (aber eigentlich gar keine Laufvariable benötigt), bietet sich dafür ebenfalls eine `for`-Schleife über einen `range` an. Will man etwa 100 Mal `Hello World!` ausgeben, so geht das wie folgt.

```
for _ in range(100):
    print('Hello World!')
```

Da die Laufvariable hier überhaupt nicht verwendet wird, bekommt sie den unauffälligen Namen `_`. Jeder andere (neue) Variablenname wäre genausogut.

Beispiel 2.3.7. `for`-Schleife über String:

```
s = "regal"
for zeichen in s:
    print(zeichen)
```

2.3.8. Man kann jede `for`-Schleife relativ einfach durch eine meist weniger elegante `while`-Schleife ersetzen (umgekehrt ist das aber im Allgemeinen oft nicht sinnvoll möglich).

✎

⚒ Aufgabe A33

Schreibe jedes der folgenden Programme mit `while`-Schleife so um, dass es stattdessen eine elegantere `for`-Schleife verwendet, sonst aber dasselbe macht.

(a) Hier ist eine `for`-Schleife über einen geeigneten `range` zu vewenden.

```
i = 2
while i < 10:
    print(i)
    print(f'Das Quadrat von {i} ist {i**2}.')
    i = i+1
print('Hier geht das Programm weiter.')
```

(b) Hier ist eine `for`-Schleife über eine Liste (nämlich die Liste `wortliste`) zu vewenden.

```
wortliste = ['are', 'you', 'as', 'bored', 'as', 'I', 'am']
satz = ''
i = 0
while i < len(wortliste):
    wort = wortliste[i]
    satz = wort + ' ' + satz
    i = i+1
print(satz)
```

(c) Hier ist eine `for`-Schleife über eine Liste (nämlich die Liste `alt`) zu vewenden.

```
alt = [3, 6, 12, 50]
neu = []
i = 0
while i < len(alt):
    x = alt[i]
    neu.append(x**2)
    i = i+1
print(neu)
```

(d) Hier ist eine `for`-Schleife über einen String (nämlich den String `wort`) zu vewenden.

```
wort = 'Hellebarde'
neu = ''
i = 0
while i < len(wort):
    zeichen = wort[i]
    if zeichen == 'e':
        neu = neu + 'o'
    else:
        neu = neu + zeichen
    i = i+1
print(neu)
```

2.4 Lernkontrolle (und Vorbereitung auf die Prüfung)

Aufgabe A34 Schreibe die Ausgabe des folgenden Programms auf. (Die Ausgabe jedes `print`-Befehls gehört in eine neue Zeile.)

```
a = [2, 6, 1, 3, -7, -2, 1, -5]
i = 0
while True:
    print(i, a[i])
    if a[i] < 0:
        break
    i = i+1
```

Aufgabe A35 Schreibe die Ausgabe des folgenden Programms auf.

```
x = 15
while x > 1:
    print(x)
    if x % 3 == 0:
        x = x // 2
    else:
        x = x + 1
```

Aufgabe A36 Schreibe die Ausgabe des folgenden Programms auf.

```
buchstaben = ['a', 'o', 'e']
s = 't'
for x in buchstaben:
    s = s+x+s
    print(s)
```

Aufgabe A37 Schreibe die Ausgabe des folgenden Programms auf.

```
for i in range(5):
    print(i*str(i))
```

Aufgabe A38 Schreibe die Ausgabe des folgenden Programms auf.

```
p = 1
for i in range(2, 8, 2):
    p = p*i
    print(p)
```

Aufgabe A39 Schreibe die Ausgabe des folgenden Programms auf.

```
s = 'lager'
t = ''
for z in s:
    t = z+t+z
    print(t)
```

Aufgabe A40 Schreibe die Ausgabe des folgenden Programms auf und sage allgemein, welche Ausgabe es produziert, wenn `geheimcode` eine beliebige (in einem String gespeicherte) Folge von Ziffern ist.

```
geheimcode = '8304'
q = 0
for z in geheimcode:
    print(q, z)
    q = q + int(z)
print(q)
```

Aufgabe A41 Schreibe die Ausgabe des folgenden Programms auf.

```
text = 'drei chinesen mit dem kontrabass'
neu = ''
for b in text:
    if b in 'ei':
        if b == 'e':
            neu = neu+'i'
        else:
            neu = neu+'e'
    else:
        neu = neu + b
print(neu)
```

Aufgabe A42 Schreibe das folgende Programm mit einer `for`-Schleife statt mit einer `while`-Schleife.

```
x = []
j = 3
while j < 11:
    print(x, j)
    x.append(j**2)
    j = j+2
print(x)
```

Aufgabe A43

(a) Schreibe die Ausgabe des folgenden Programms auf.

(b) Schreibe in einem deutschen Satz auf, was das Programm allgemein leistet, wenn `text` ein beliebiger Text aus Kleinbuchstaben ist.

```
text = 'drei chinesen mit dem kontrabass'
z = 0
for b in text:
    if b in 'aeiou':
        z=z+1
print(z)
```

Aufgabe A44

(a) Schreibe die Ausgabe des folgenden Programms auf.

(b) Schreibe in einem deutschen Satz auf, was das Programm allgemein leistet, wenn `x` und `y` beliebige positive natürliche Zahlen sind.

```
x = 12
y = 20
v = x
while not (v % y == 0):
    v = v+x
print(v)
```

Aufgabe A45 Erweitere das folgende Programm so, dass es das kleinste Element der Liste `zahlen` bestimmt und die Position (= den Index), an dem dieses kleinste Element steht.

```
zahlen = [21, 7, 25, 9, 2, 12]
```

Die Ausgabe soll wie folgt aussehen. Das Programm soll für jede Liste von Zahlen mit mindestens einem Eintrag funktionieren.

```
Das kleinste Element der Liste [21, 7, 25, 9, 2, 12] ist 2.
Es steht an Position 4.
```

2.5 Lösungen

Hinweise zu den Symbolen:

⚒ Diese Aufgaben könnten (mit kleinen Anpassungen) an einer Prüfung vorkommen. Für die Prüfungsvorbereitung gilt: “If you want to nail it, you’ll need it”.

Diese Aufgaben sind wichtig, um das Verständnis des Prüfungsstoffs zu vertiefen. Die Aufgaben sind in der Form aber eher nicht geeignet für eine Prüfung (zu grosser Umfang, nötige «Tricks», zu offene Aufgabenstellung, etc.). **Teile solcher Aufgaben können aber durchaus in einer Prüfung vorkommen!**.

Diese Aufgaben sind dazu da, über den Tellerrand hinaus zu schauen und/oder die Theorie in einen grösseren Kontext zu stellen.

⚒ Lösung zu A1 ex-rechnen-mit-variablen

```
a = 7
a = 2*a
a = a+10
a = a**3
a = a // 7
a = a**10
a = a % 100
a = a+5
a = a**0.5
print(a)
```

⚒ Lösung zu A2 ex-fibonacci-per-moivre-binet-formel

```
n = 20
f = 1/5**0.5*(((1+5**0.5)/2)**n-((1-5**0.5)/2)**n)
print(f)

# Variante mit Speicherung von Zwischenergebnissen,
# was das Programm übersichtlicher macht.

g = (1+5**0.5)/2    # Das ist übrigens der goldene Schnitt.
h = (1-5**0.5)/2    # Das ist auch der negative Kehrwert von g (denn g*h=-1).
f = 1/5**0.5*(g**n-h**n)
print(f)
```

⚒ Lösung zu A3 ex-mitternachtsformel-elementar

(a) Korrigiertes Programm:

```
a = 2
b = 12
c = -182

x1 = (-b+(b**2-4*a*c)**0.5)/(2*a)
x2 = (-b-(b**2-4*a*c)**0.5)/(2*a)

print('Die Gleichung', a, 'x^2 +', b, 'x +', c, '=0 hat die Lösungen')
print('x1=', x1)
print('x2=', x2)
```

```
print('Probe (es sollte beide Male Null herauskommen):')
# Probe für x1
print(a*x1**2+b*x1+c)
# Probe für x2
print(a*x2**2+b*x2+c)
```

(b) (i) Genau eine Lösung, nämlich $x_1 = x_2 = 3.5$.
Angepasste ersten drei Zeilen:

```
a = 6
b = -42
c = 147
```

Ausgabe:

```
Die Gleichung 6 x^2 + -42 x + 147 hat die Lösungen
x1= (3.5+3.5j)
x2= (3.5-3.5j)
Probe (es sollte beide Male Null herauskommen):
0j
0j
```

(ii) Keine (reelle) Lösung, aber zwei komplexe Lösungen $x_{1,2} = 3 \pm i$.
Angepasste ersten drei Zeilen:

```
a = 1
b = -6
c = 10
```

Ausgabe:

```
Die Gleichung 1 x^2 + -6 x + 10 hat die Lösungen
x1= (3+1j)
x2= (3-1j)
Probe (es sollte beide Male Null herauskommen):
0j
0j
```

⚒ Lösung zu A4 ex-swap-and-cycle
Vertauschen:

```
h = a  # Wert von a sichern
a = b  # a überschreiben
b = h  # gesicherter Wert in b
```

Bonus:

```
a = a-b
b = a+b
a = b-a
```

oder

```
a = b-a
b = b-a
a = a+b
```

Bemerkung: In Python ist sogar die folgende «Doppelzuweisung» möglich:

```
a, b = b, a
```

⚒ Lösung zu A5 ex-vertauschen-mit-subtraktion-verstehen

Die Werte von `a` und `b` werden miteinander vertauscht.

⚒ Lösung zu A6 ex-rechnen-mit-strings

```
DreiSieben
3 + 7
10
SiebenSiebenSieben
21
3 * 7
3333333
777
Drei-Drei-Sieben!
Drei-Sieben! Drei-Sieben!
AAXXX! AAXXX! AAXXX!
(2+2*2*2)+2
```

⚒ Lösung zu A7 ex-strings-mit-zeilenumbruch

(a)
```
print(3 * "Ich liebe Informatik!\n")
```

(b)
```
print(6*((30*"+")+"\n"))
```

(c)
```
print(4 * (12 * "+--" + "+\n"))
```

⚒ Lösung zu A8 ex-hypotenuse-ausrechnen

```
# Einlesen von a in einer Zeile
a = float(input('Länge der Kathete a: '))
# Einlesen von b in zwei Zeilen
string_b = input('Länge der Kathete b: ')
b = float(string_b)
c = (a**2+b**2)**0.5
print(f'Die Hypotenuse hat die Länge {c}.')
```

⚒ Lösung zu A9 ex-kreis-umfang-und-flaeche

```
from math import pi
# print(pi)
r = float(input('Welchen Radius hat der Kreis? '))
U = 2*pi*r
A = pi*r**2
print(f'Der Kreis mit Radius {r} hat den Umfang {U}\nund die Fläche {A}.')
```

⚒ Lösung zu A10 ex-satz-einrahmen

```
s = input("Gib einen String ein! ")
laenge = len(s)
print((laenge+4)*"*")
print(f"* {s} *")
print((laenge+4)*"*")
```

⚒ Lösung zu A11 ex-schaltjahr

Nur die beiden Variablen `ist_schaltjahr_B` und `ist_schaltjahr_D` liefern stets die richtigen Werte.

Hier ist ein Programm, dass die vier Zuweisungen für die vier Jahreszahlen 2000, 2024, 2025 und 2100 berechnet und die Werte der Variablen ausgibt. (Das Programm enthält eine for-Schleife und muss deswegen noch nicht verstanden werden.)

```
for jahr in [2000, 2024, 2025, 2100]:
    ist_schaltjahr_A = (jahr % 4 == 0) and (jahr % 100 != 0) and (jahr % 400 == 0)
    ist_schaltjahr_B = (jahr % 4 == 0) and ((jahr % 100 != 0) or (jahr % 400 == 0))
    ist_schaltjahr_C = (jahr % 4 == 0) or (jahr % 100 != 0) or (jahr % 400 == 0)
    ist_schaltjahr_D = (jahr % 400 == 0) or ((jahr % 4 == 0) and (jahr % 100 != 0))
    print(f'{jahr=}: {ist_schaltjahr_A=}, {ist_schaltjahr_B=}, {ist_schaltjahr_C=}, {ist_schaltjahr_D=}, ')
```

Der Output dieses Programms ist

```
jahr=2000: ist_schaltjahr_A=False, ist_schaltjahr_B=True, ist_schaltjahr_C=True, ist_schaltjahr_D=True,
jahr=2024: ist_schaltjahr_A=False, ist_schaltjahr_B=True, ist_schaltjahr_C=True, ist_schaltjahr_D=True,
jahr=2025: ist_schaltjahr_A=False, ist_schaltjahr_B=False, ist_schaltjahr_C=True, ist_schaltjahr_D=False,
jahr=2100: ist_schaltjahr_A=False, ist_schaltjahr_B=False, ist_schaltjahr_C=True, ist_schaltjahr_D=False,
```

Dieser Output enthält dieselbe Information wie die Tabelle.

⚒ Lösung zu A12 ex-listen-elementar

(a)

```
[13, 8, 2, 4]
```

(b)

```
[0, 1, 1, 2, 3, 5]
```

⚒ Lösung zu A13 ex-listen-erzeugen-mit-range-und-list-comprehension

```
a = list(range(31))
b = list(range(-10, 21))
c = list(range(10, 26, 3))

# Lücken füllen!
aa = [i for i in range(31)]
bb = [-10+i for i in range(31)]
cc = [10+3*i for i in range(6)]

print(a)
print(aa)
print(b)
print(bb)
print(c)
print(cc)
```

```
w = [n**(1/n) for n in range(1, 11)]
print(w)

k = [n**3 for n in range(1, 21)]
print(k)

print("Ausgabe aller k-ten Potenzen der Zahlen von 1 bis n.")
n = int(input("Gib n ein: "))
k = float(input("Gib k ein: "))
p = [x**k for x in range(1, n+1)]
print(p)
```

⚒ Lösung zu A14 ex-graphen-zeichnen-mit-matplotlib

```
import matplotlib.pyplot as plt
from math import sin, cos, exp, log

plt.gca().set_aspect('equal')

xwerte = list(range(-3, 4))
ywerte = [x**2 for x in xwerte]
plt.plot(xwerte, ywerte)
plt.scatter(xwerte, ywerte)

xwerte = [-3+t*0.1 for t in range(61)]
ywerte = [x**2 for x in xwerte]
plt.plot(xwerte, ywerte)

ywerte = [1/2*x+2 for x in xwerte]
plt.plot(xwerte, ywerte)

xwerte = [-7+t*0.1 for t in range(141)]
sinuswerte = [sin(x) for x in xwerte]
plt.plot(xwerte, sinuswerte)
cosinuswerte = [cos(x) for x in xwerte]
plt.plot(xwerte, cosinuswerte)

xwerte = [-7+t*0.1 for t in range(96)]
expwerte = [exp(x) for x in xwerte]
plt.plot(xwerte, expwerte)
xwerte = [0.1+t*0.1 for t in range(69)]
logwerte = [log(x) for x in xwerte]
plt.plot(xwerte, logwerte)
plt.show()
```

⚒ Lösung zu A15 ex-string-slicing

```
K
e
urban
Kult
use
presse
erdkern
mieten
```

```
jesuit
satire
e
use
```

⚒ Lösung zu A16 ex-string-slicing-aktiv

Programm:

```
s = 'Donaudampfschiffahrt'
# dampf
print(s[5:10])

s = 'Versuchung'
# Versuch
print(s[:7])

s = 'kantonssschule'
# antonsschule
print(s[1:])

s = 'erbverzichtsvertrag'
# reiter
print(s[1::3])

s = 'kirchenstaatsvertrag'
# retter
print(s[2::3])

s = 'rohrfederzeichnung'
# hering
print(s[2::3])

s = 'kapitalanlagefirma'
# killer
print(s[0::3])
```

Output:

```
dampf
Versuch
antonssschule
reiter
retter
hering
killer
```

⚒ Lösung zu A17 ex-lernkontrolle-1-datentypen-operationen

(a)
- Integer
  `x=1`
- Float
  `x=1.2`
- String
  `x='Hallo'`
- Boolean

```
x=True
```

(b) `print(type(x))`

(c) Liste

```
a = [1, 2]
```

Lösung zu A18 ex-lernkontrolle-1-zahlen

(a)
- 5^{1234}: `5**1234`
- $\sqrt{2357}$: `2357**(1/2)` oder `2357**0.5`
- den Rest der Division von 1234567890 durch 235: `1234567890 % 235`
- den Ganzzahlquotient der Division von 1234567890 durch 235: `1234567890 // 235`
- das Ergebnis der Division von 1234567890 durch 235 (als Kommazahl): `1234567890 / 235`

(b) `gauss = (n*(n+1))/2`

(c) `V = 1/3*h*(G+(G*D)**0.5+D)`

(d)
```
5 7
7.0 25.0
```

Lösung zu A19 ex-lernkontrolle-1-strings

(a)
```
AAA
-AAA-
-AAA--AAA--AAA--AAA--AAA-
XXXXX
-AAA-
```

```
x = int(input('Wert von x: '))
y = int(input('Wert von y: '))
print(f'Der Rest der Division von {x} durch {y} ist {x % y}.')
```

Lösung zu A20 ex-lernkontrolle-1-booleans

(b)
```
print(True or False)
print(True and False)
x = 208
print(3 < x and x < 208)
print(x % 2 == 0)
```

```
x = int(input('Wert von x: '))
print(x % 2 == 0 and x % 3 != 0)
```

Lösung zu A21 ex-lernkontrolle-1-listen

(b)
```
[2, 3, 10]
[2, 3, 10, 6]
[2, 6, 10, 6]
```

(b) `b = list(range(10, 101))`

(c) `c = list(range(10, 101, 2))`
(d) `d = [k**2 for k in range(1, 101)]`

⚒ Lösung zu A22 ex-lernkontrolle-1-slicing

(a) Slicing bei Listen:

```
2
[2, 8, -4, 22]
[2, 8, -4, 22, 5, 4]
[3, 7, 2, 8, -4, 22]
[2, 22]
```

(b) Slicing bei Strings:

```
t
terre
erdkern
```

⚒ Lösung zu A23 ex-tageslicht

⚒ Lösung zu A24 ex-basics-if-kopfrechentrainer

⚒ Lösung zu A25 ex-temperatur-verbal

⚒ Lösung zu A26 ex-lineare-gleichung

⚒ Lösung zu A27 ex-quadratische-gleichung

⚒ Lösung zu A28 ex-zahlen-summieren-mit-while

⚒ Lösung zu A29 ex-collatz

```
n = int(input("Gib eine natürliche Zahl > 0 ein: "))
while n>1:
    print(n)
    if n gerade:
        n = n/2
    else:
        n = 3*n+1
print(n)
```

⚒ Lösung zu A30 ex-renten-sparen

(a)

```
betrag = 6000
zins = 0.01
```

```
laufzeit = 35
n = 1
kontostand = 0
while n<=laufzeit:
    kontostand = kontostand+betrag
    kontostand = kontostand*(1+zins)
    print("Kontostand Jahr ", n, ":", kontostand)
    n = n+1
```

```
Kontostand Jahr  1 : 6060.0
Kontostand Jahr  2 : 12180.6
Kontostand Jahr  3 : 18362.406
Kontostand Jahr  4 : 24606.03006
Kontostand Jahr  5 : 30912.090360600003
Kontostand Jahr  6 : 37281.21126420601
Kontostand Jahr  7 : 43714.02337684807
Kontostand Jahr  8 : 50211.163610616546
Kontostand Jahr  9 : 56773.27524672271
Kontostand Jahr  10 : 63401.00799918994
Kontostand Jahr  11 : 70095.01807918184
Kontostand Jahr  12 : 76855.96825997366
Kontostand Jahr  13 : 83684.52794257339
Kontostand Jahr  14 : 90581.37322199912
Kontostand Jahr  15 : 97547.18695421911
Kontostand Jahr  16 : 104582.6588237613
Kontostand Jahr  17 : 111688.48541199892
Kontostand Jahr  18 : 118865.3702661189
Kontostand Jahr  19 : 126114.0239687801
Kontostand Jahr  20 : 133435.1642084679
Kontostand Jahr  21 : 140829.5158505526
Kontostand Jahr  22 : 148297.81100905812
Kontostand Jahr  23 : 155840.7891191487
Kontostand Jahr  24 : 163459.1970103402
Kontostand Jahr  25 : 171153.7889804436
Kontostand Jahr  26 : 178925.32687024804
Kontostand Jahr  27 : 186774.58013895052
Kontostand Jahr  28 : 194702.32594034003
Kontostand Jahr  29 : 202709.34919974342
Kontostand Jahr  30 : 210796.44269174084
Kontostand Jahr  31 : 218964.40711865824
Kontostand Jahr  32 : 227214.05118984482
Kontostand Jahr  33 : 235546.19170174326
Kontostand Jahr  34 : 243961.6536187607
Kontostand Jahr  35 : 252461.27015494832
```

(b) Noch keine Musterlösung.

⚒ Lösung zu A31 ex-wurzel-annaehern

```
x = 40
fehler = 0.000001
links = 0
rechts = x

while rechts - links > fehler:
    mittel = (links + rechts) / 2
    if mittel**2 < x:
```

```
        links = mittel
    else:
        rechts = mittel

schaetzung = (links + rechts) / 2
print(f'{schaetzung} ist ungefähr die Wurzel aus {x}.')
print(f'{x**0.5} ist laut Python die Wurzel aus {x}.')
print(f'Der Fehler ist etwa {abs(mittel-x**0.5):.10f}.')
```

```
6.324555575847626 ist ungefähr die Wurzel aus 40.
6.324555320336759 ist laut Python die Wurzel aus 40.
Der Fehler ist etwa 0.0000000425.
```

⚒ Lösung zu A32 ex-zahl-erraten

⚒ Lösung zu A33 ex-while-schleifen-in-for-schleifen-umwandeln

(a)

```
for i in range(2, 10):
    print(i)
    print(f'Das Quadrat von {i} ist {i**2}.')
print('Hier geht das Programm weiter.')
```

```
wortliste = ['are', 'you', 'as', 'bored', 'as', 'I', 'am']
satz = ''
for wort in wortliste:
    satz = wort + ' ' + satz
print(satz)
```

(b)

```
alt = [3, 6, 12, 50]
neu = []
for x in alt:
    neu.append(x**2)
print(neu)
```

```
wort = 'Hellebarde'
neu = ''
for zeichen in wort:
    if zeichen == 'e':
        neu = neu + 'o'
    else:
        neu = neu + zeichen
print(neu)
```

⚒ Lösung zu A34 ex-while-und-if-1-ausgabe-bestimmen

```
0 2
1 6
2 1
3 3
4 -7
```

⚒ Lösung zu A35 ex-while-und-if-2-ausgabe-bestimmen

```
15
7
8
9
4
5
6
3
```

⚒ Lösung zu A36 ex-for-liste-1-ausgabe-bestimmen

```
tat
tatotat
tatotatetatotat
```

⚒ Lösung zu A37 ex-for-range-1-ausgabe-bestimmen

```

1
22
333
4444
```

⚒ Lösung zu A38 ex-for-range-3-ausgabe-bestimmen

```
2
8
48
```

⚒ Lösung zu A39 ex-for-string-1-ausgabe-bestimmen

```
ll
alla
gallag
egallage
regallager
```

⚒ Lösung zu A40 ex-for-string-2-ausgabe-bestimmen

Das Programm berechnet allgemein die Quersumme des Geheimcodes.

```
0 8
8 3
11 0
11 4
15
```

⚒ Lösung zu A41 ex-for-string-5-ausgabe-bestimmen

```
drie chenisin met dim kontrabass
```

⚒ Lösung zu A42 ex-while-schleife-in-for-schleife-umwandeln-1

```
x = []
for j in range(3, 11, 2):
    x.append(j**2)
print(x)
```

⚒ Lösung zu A43 ex-vokale-zaehlen

(a) Ausgabe: siehe unten
(b) Das Programm zählt, wie viele Vokale in `text` vorkommen.

```
10
```

⚒ Lösung zu A44 ex-kgv-bestimmen

(a) Ausgabe: siehe unten
(b) Das Programm ermittelt das kgV (= kleinste gemeinsame Vielfache) von x und y.

```
60
```

⚒ Lösung zu A45 ex-minimum-in-liste-samt-index

```
zahlen = [21, 7, 25, 9, 2, 12]
position = 0
minimumum = zahlen[position]
for i in range(len(zahlen)):
    if zahlen[i] < minimumum:
        position = i
        minimumum = zahlen[i]
print(f'Das kleinste Element der Liste {zahlen} ist {minimumum}.')
print(f'Es steht an Position {position}.')
```