Einführung in die Programmierung mit Processing

Aufgabe 1: "Hello World" Sketch

Im Rahmen dieses Übungsblatts werden wir zunächst ein einfaches Programm "Hello World" erstellen, das nur diese Meldung ausgibt und danach *terminiert*, d. h. beendet wird.
Der Begriff »Hello World!« markiert für viele Entwickler den Start ihres Lernprozesses beim Programmieren. Das »Hallo Welt!«-Programm wurde erstmals im Lernbuch für die Programmiersprache C der Bell Laboratories "Programming in C – A Tutorial" (Autor Brian Kernighan im Jahr 1974) verwendet.

Die Programmentwicklung in Processing findet in einer integrierten Programmentwicklungsumgebungen statt (engl.: **integrated development environment** oder kurz: **IDE**). Neben einem Editor, d. h. einem Textverarbeitungsprogramm für das Eingeben des Programm- bzw. *Quellcodes* (engl. *source code*) eines Programms, umfasst diese alle Werkzeuge, um den Quellcode in ein ausführbares Programm zu übersetzen und um dieses schließlich auszuführen.

Starten Sie zunächst bitte Processing ...

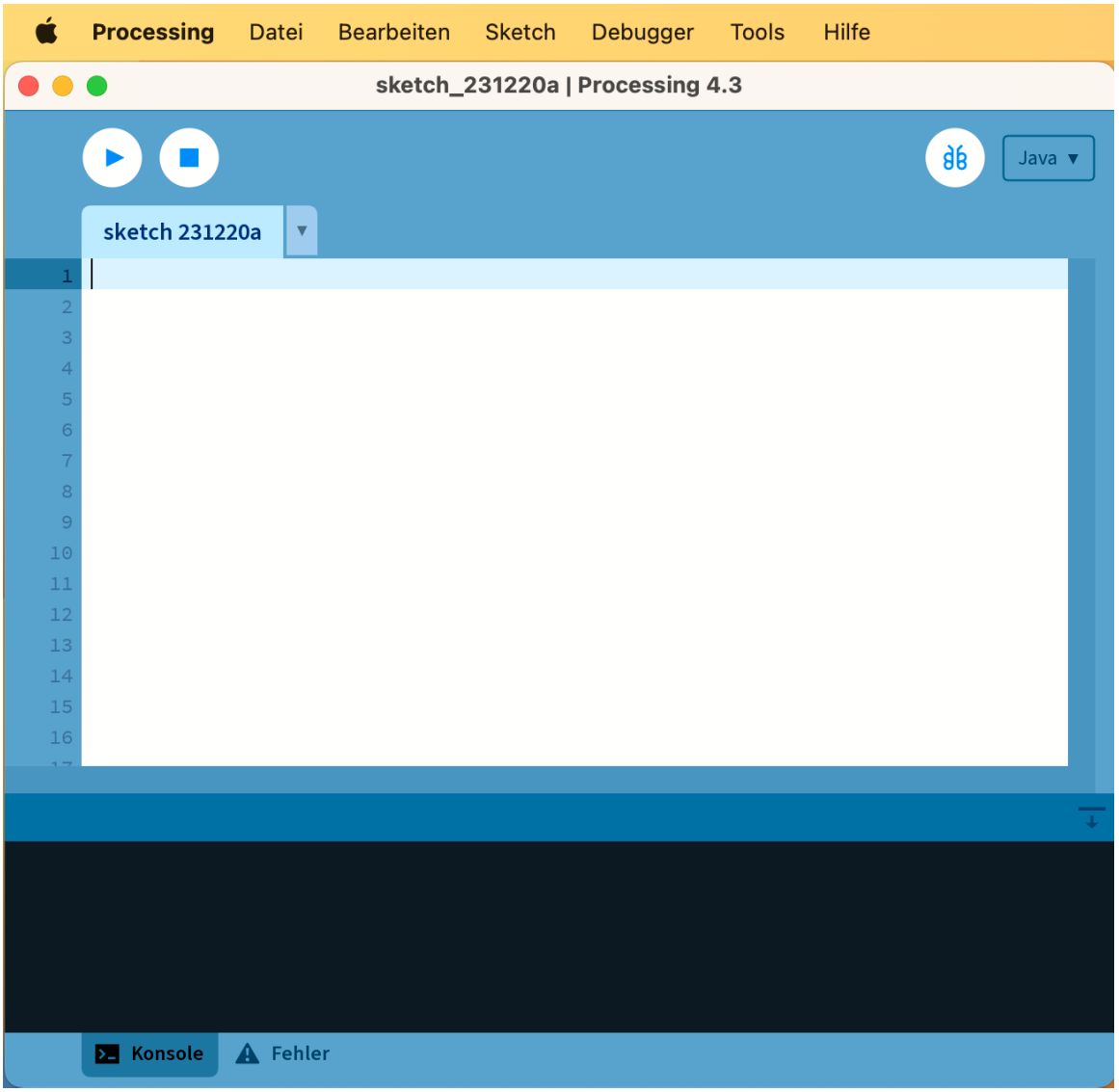


Abb. 1: Processing starten

.. und tippen Sie dann in den Editor bitte folgendes ein:

```
println("Hello World");
```

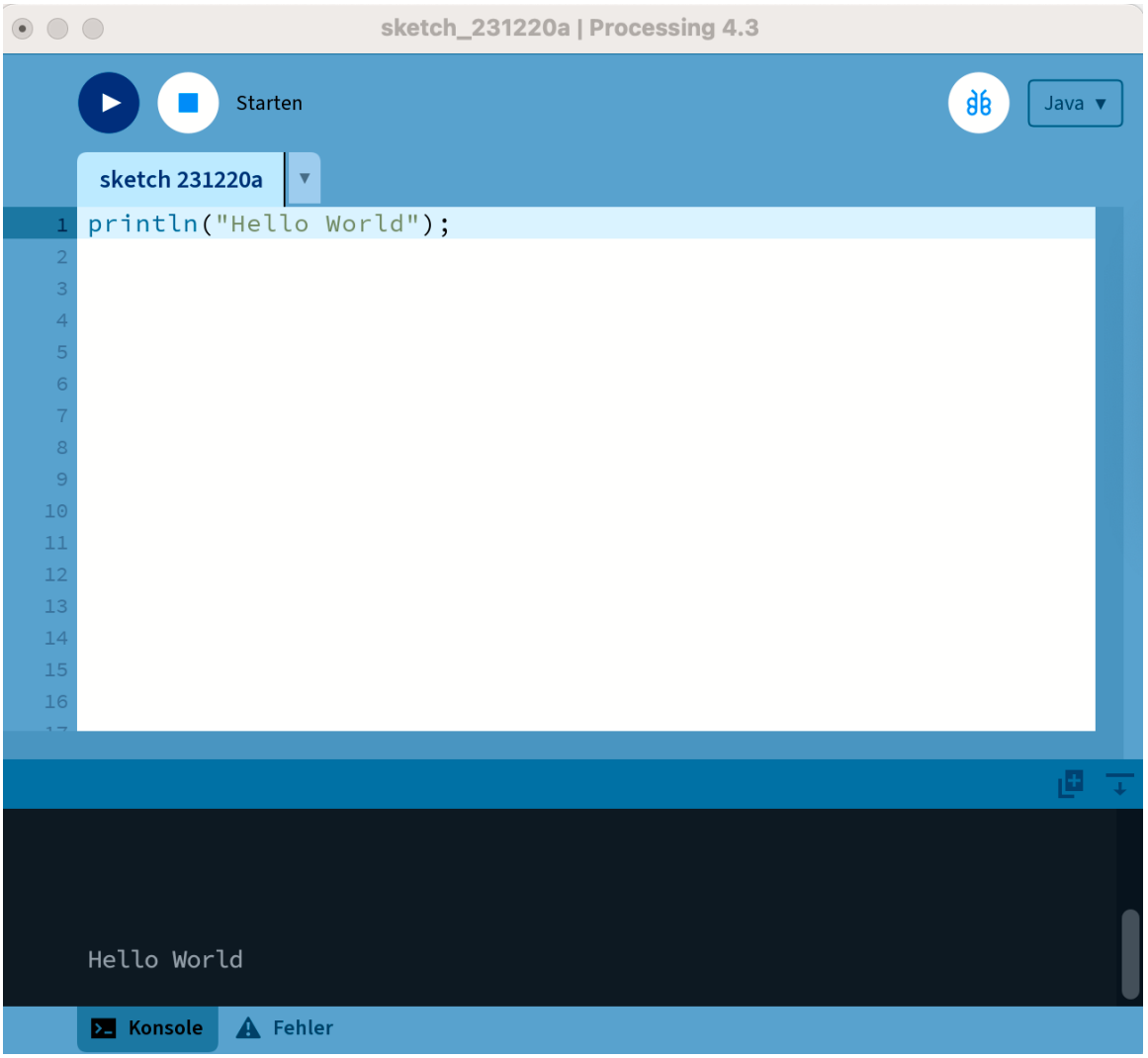


Abb. 2: Unsere erste Sketch

Durch Drücken auf den Start-Button  können Sie das Programm ausführen. Hierfür wird es von der IDE in ausführbaren Maschinencode übersetzt. Der im Programm enthaltene Befehl `println` gibt die angegebene Zeichenkette `"Hello World"` in das Konsolenfenster bzw. einfach "auf die Konsole" aus. Dieser Begriff kommt von den frühesten Computern, bei denen die Benutzer nur über eine einfache Tastatur-Monitor-Kombination - die sogenannte "Konsole" - an einen Zentralrechner angeschlossen waren und mit diesem arbeiteten.

Wir haben damit unser erstes Programm (in Processing nennt man das Sketch ("Skizze")) erstellt und ausgeführt.

Während Sie Inhalte eingeben, startet die IDE regelmäßig im Hintergrund den Übersetzungsvorgang, bei dem der Dateiinhalt zu ausführbarem Programmcode übersetzt wird. Dieser Vorgang wird als *Kompilieren* (engl.: *compile*) bezeichnet. Das Kompilieren der Programmdatei in ausführbaren Programmcode geschieht bei Processing also im Hintergrund, ohne dass hierfür ein bestimmter Knopf zu drücken wäre.

Solange Ihr Programm grammatikalische Fehler oder Schreibfehler enthält, d. h. Fehler der **Syntax**, signalisiert dies die IDE, indem sie die betreffenden Zeilen, wie in der Abbildung unten, rot unterkringelt und darunter eine rot markierte Fehlerbeschreibung ausgibt. In der Abbildung unten wurde z. B. das Wort `println` falsch geschrieben. Die IDE signalisiert mit diesem Fehler ebenfalls, dass das Programm nicht übersetzt und damit auch nicht ausgeführt werden kann. In Zukunft müssen Sie also solange den Programmcode überarbeiten, bis die IDE keine Fehler mehr anzeigt. Erst dann wird das Programm übersetzt und kann schließlich auch ausgeführt werden.

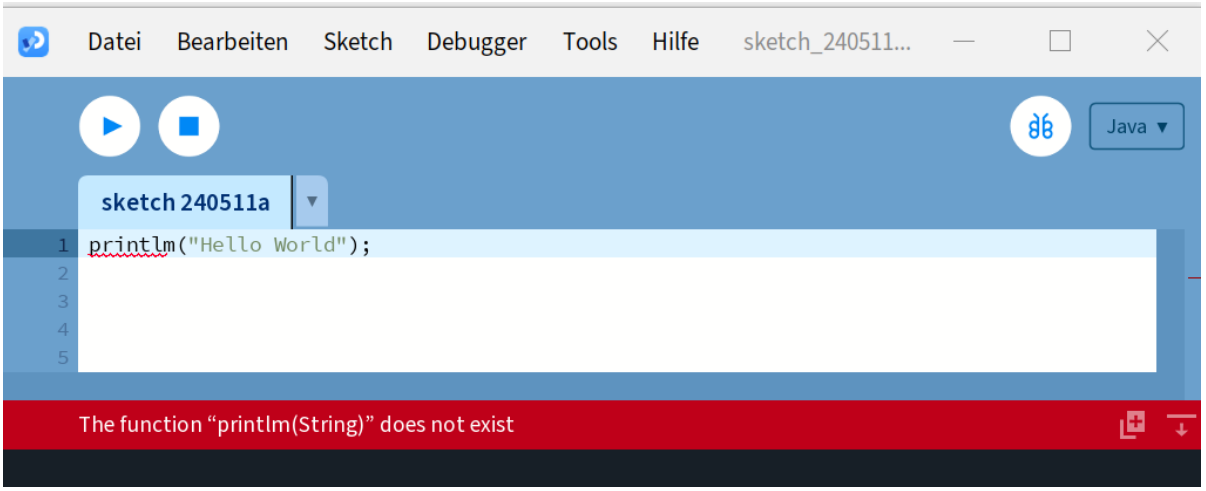


Abb. 3: Fehlermeldung

Neben der Möglichkeit einfache Textausgaben auf die Konsole durchzuführen, bietet Processing als multimediales Werkzeug auch vielfältige Möglichkeiten zu grafischen Ausgaben bis hin zu Animationen. Diese Themen werden wir erst später behandeln, wenn wir thematisch so weit sind. Für die anstehenden Übungen reichen uns vorerst einfache Textausgaben "auf die Konsole".

Um die Sketch schließlich abzuspeichern wählen Sie *Datei* > *Speichern*. Als Name können Sie z. B. "Uebungsblatt1" wählen. Unter macOS und Windows legen Sie Ihre Sketch-Dateien dieses Kurses am besten alle in einen Ordner unter

```
/Benutzer/Dokumente/Processing
```

Processing erstellt für jede neue Sketch automatisch einen Unterordner. Im weiteren Verlauf der Vorlesung, werden wir Processing noch um **Libraries** und sogenannte **Tools** erweitern, die ebenfalls in einem Unterordner "Libraries" bzw. "Tools" dieses Verzeichnisses gespeichert werden bzw. dort auch gesucht werden können.

Aufgabe 2: Zahlen in Variablen speichern und ausgeben

In den Vorlesungsfolien (Seite 51) haben wir die Ausgabe des Beispielprogramms ebenfalls mit einer Textausgabe über die Konsole realisiert. Versuchen wir nun das ganze praktisch umzusetzen.
Verwenden Sie dafür den folgenden Code:

```
//Uebungsblatt 1

/*
//Aufgabe 1
println("Hello World");
*/

//Aufgabe 2
int hoursWorked = 40;
println("Hours Worked: " + hoursWorked);
```

In diesem ist der bereits vorhandene Programmcode in Kommentarzeichen gesetzt, um diesen zu deaktivieren. Man spricht in diesem Zusammenhang von "Auskommentieren". Darunter steht als neuer aktiver Code der Programmcode von der Folie.
Speichern Sie das Programm (Uebungsblatt1) und führen Sie es aus. Es sollte folgende Ausgabe im Konsolenfenster erscheinen:

Abb. 4: Hours Worked: 40

Um in einem Programm Zahlenwerte zwischenspeichern zu können, werden Variablen benötigt. Hier dient die Variable `hoursworked` dazu, den Wert 40 zwischenzuspeichern.

Definieren Sie nun in ihrer Sketch zwei weitere Variablen für den Stundenlohn ("hourly rate") und die Anzahl der Arbeitstage ("workdays"), sodass z. B. folgende Ausgabe erscheint, wenn der Stundenlohn 35 ist und die Anzahl der Arbeitstage gleich 5:

Abb. 5: Ausgabe

Erweitern Sie ihre Sketch danach in der Weise, dass die Variablen für den Stundenlohn etc., nachdem ihr Wert auf die Konsole ausgegeben wurde, neue Werte zugewiesen bekommen. Danach geben Sie die Variablen erneut aus, sodass sich z. B. folgende Ausgabe ergibt:

Abb. 6: Ausgabe

Zeilenwechsel innerhalb einer Ausgabe lassen sich auch durch Hinzufügen von \n am Ende der auszugebenden Zeichenkette auslösen.

Aufgabe 3: Wertebereiche von Datentypen

Kommentieren Sie den vorhandenen Code wieder aus und ergänzen Sie die Sketch durch die

folgenden zwei Anweisungen:

```
//Aufgabe 3
short val = 32;
println("A value of data type short: " + val);
```

Wie sie sehen, verwenden wir pro Übungsblatt nur eine einzige Sketch. Die einzelnen Aufgabenlösungen kommentieren wir im Quellcode jeweils aus, wenn die betreffende Aufgabe fertig ist, um immer nur die aktuelle Aufgabe aktiv zu haben.

Führen Sie die Sketch wieder aus. In der Konsole sollte nun 32 erscheinen. Erhöhen Sie den Wert danach auf z. B. 1000 und wiederholen Sie das Ausführen. Es sollte immer noch alles wie gehabt funktionieren.

Probieren Sie danach den folgenden Wert: 35000. Nun wird es beim Kompilieren ein Problem geben und die Sketch wird nicht starten. Um eine Erläuterung zu dem Problem zu erhalten, schauen wir uns unten wieder die Fehlermeldung an. Für Fehlermeldungen gibt es zusätzlich einen Reiter "Fehler", der aktiviert werden kann und gegebenenfalls weitere Informationen zu den aktuellen Fehlern anzeigt.

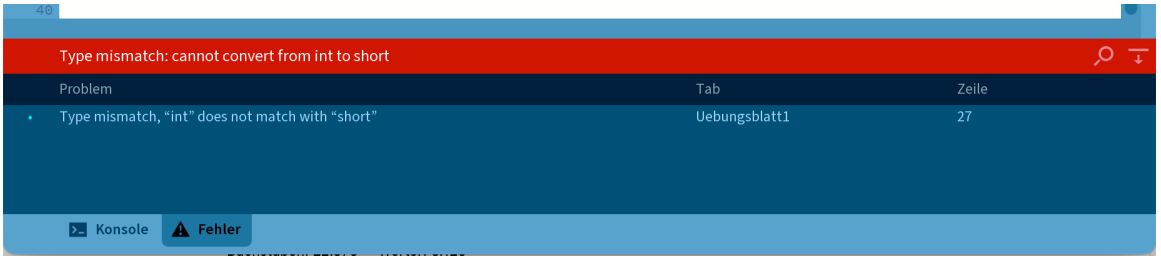


Abb. 7: Fehlermeldung

Wie lässt sich das Problem mit dem Fehler bei dem vorhandenen Zahlenwert lösen?

Diese Aufgabe war ein Beispiel für Aufgaben aus dem Online-Kurs von Bradley Kjell für die Programmiersprache Java. Die Aufgabe entstammte den Aufgaben zu Kapitel 8 des Kurses und entspricht dort inhaltlich Aufgabe 1. Da sich diese Aufgaben in adaptierter Form auch gut für das Lernen der Programmierung mit Processing eignen, werden wir im Rahmen dieser Veranstaltung - insbesondere zu Beginn - eine Anzahl der Aufgaben aus dem Online-Kurs bearbeiten.

Aufgabe 4: Schriftzeichen ausgeben

Hier eine weitere Aufgabe aus dem Kurs. Kommentieren Sie zuerst Aufgabe 3 aus und ergänzen Sie dann ihre Sketch zu Übungsblatt 1 (Uebungsblatt1.pde) um die folgenden zwei Befehle und führen diese aus:

```
//Aufgabe 4
char ch = 'A' ;
println("Ein Character: " + ch);
```

Führen Sie danach die folgenden Änderungen durch. Prüfen Sie nach jeder Änderung, ob das Programm erstellt und ausgeführt werden kann:

1. Ändern Sie 'A' in 'Z'.
2. Ändern Sie 'Z' in 'AA'. Nun gibt es ein Problem. Was könnte die Ursache hierfür sein?
3. Ändern Sie 'AA' in '␣' (␣ steht für ein Leerzeichen). Hier gibt es kein Problem. Warum?
4. Ändern Sie '␣' in ''.
   - Hinweis: Es befindet sich KEIN Zeichen zwischen den beiden Hochkommata.
   - Nun geht es wieder nicht. Warum?
5. Ändern Sie '' in "A".
   - Jetzt gibt es wieder ein Problem? Warum?

Aufgabe 5: Lohnprogramm

Fügen Sie Ihrem Programm folgenden Programmcode hinzu:

```
int gearbeiteteStunden = 40;
int stundenLohn = 10, steuerSatz = 10;
println("Arbeitszeit: " + gearbeiteteStunden);
println("Bruttolohn: " + (gearbeiteteStunden * stundenLohn));
println("Steuerbetrag: " + (gearbeiteteStunden * stundenLohn / steuerSatz));
```

- Führen Sie das Programm aus.
- Entfernen Sie dann eine der Variableninitialisierungen, z. B. die Zeichen `= 40` für die Variable `gearbeiteteStunden`. Wird das Problem von der Entwicklungsumgebung erkannt? Kann das Programm ausgeführt werden?
- Im Programm werden die Variablen definiert. Modifizieren Sie das Programm in der Weise, dass die Variablen deklariert und - getrennt davon - initialisiert werden. Führen Sie danach das Programm wieder aus und betrachten Sie die Ausgabe.
- Nun wollen wir einen Fehler produzieren. Entfernen Sie eine der Deklarationen. Kann das Programm noch kompiliert werden?

Aufgabe 6: Wert einer Quadratgleichung

Erweitern Sie ihre vorhandene Sketch um Programmcode zur Berechnung der Quadratgleichung:

```
x² - 7x + 10
```

Deklarieren Sie hierzu eine Variable x vom Typ `int`. Weisen Sie der Variablen einen Wert zu. Schreiben Sie eine Anweisung, die den Wert der Quadratgleichung berechnet und das Ergebnis in eine andere Variable vom Typ `int` schreibt. Lassen Sie das Ergebnis ausgeben, ungefähr in der Form:

Der Aufbau der auszugebenden Zeichenkette ist wie folgt:

Teilzeichenkette + Variable + Teilzeichenkette + Variable

Mit Hilfe des + Operators können also beliebig komplexe Zeichenketten konstruiert werden.

Führen Sie das Programm für verschiedene Werte von x aus und betrachten Sie das Ergebnis. Verwenden Sie große und kleine Werte, negative Werte und Nullwerte.

Aufgabe 7: Erneute Wertzuweisung an eine Variable

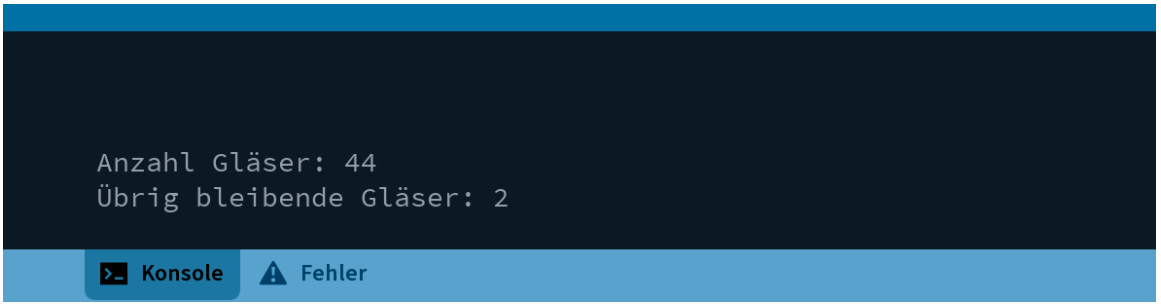
Die Quadratgleichung ergibt bei `x = 2` und bei `x = 5` Null. Verändern Sie das Programm aus der vorigen Aufgabe, so dass es die verschiedenen Ergebnisse für die Werte `6`, `2` und `5` von `x` während des gleichen Programmlaufs nacheinander berechnet und ausgibt.

Verwenden Sie für dieses Programm NUR zwei Variablen mit den Namen `x` und `wert`. Hieraus folgt, dass Sie den Variablen verschiedene Werte an verschiedenen Stellen des Programms zuweisen müssen.

Nutzen Sie beim Schreiben von Programmen die Kopieren- und Einfügen-Funktionen des Editors.

Aufgabe 8: Gläser verpacken

Gläser sollen in Kartons verpackt werden. Schreiben Sie ein Programm, das für eine gegebene Anzahl Gläser ausrechnet, wie viele Gläser übrig bleiben, wenn in jeden Karton 6 Gläser passen und dieses Ergebnis wie folgt ausgibt:



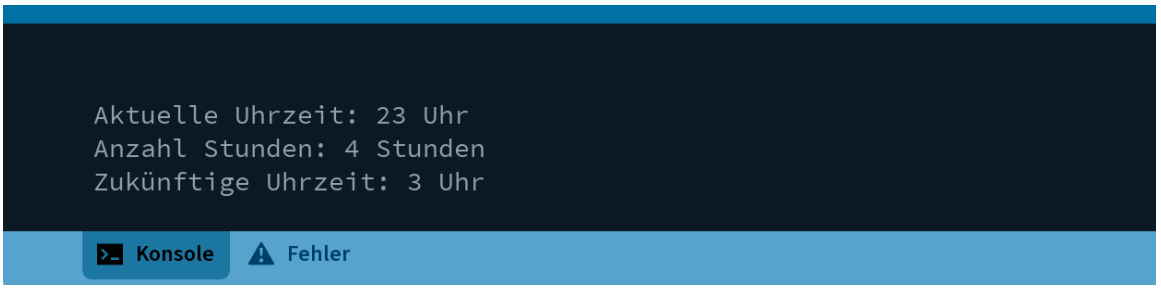
Legen Sie für die Anzahl Gläser eine Variable an und initialisieren Sie diese mit einem Wert, z. B. dem Wert `44`. Verwenden Sie **Ganzzahl-Arithmetik** und den **Modulo-Operator**. Ändern Sie den Wert der Variable für die Anzahl Gläser einige Male. Es werden immer mindestens 6

Gläser verpackt. Kompilieren Sie das Programm jeweils, führen Sie es aus und kontrollieren Sie das Ergebnis.

Es müssen keine Werte über die Programmoberfläche eingegeben werden. Das Eingeben von Werten werden wir erst später behandeln.

Aufgabe 9: Uhrzeit

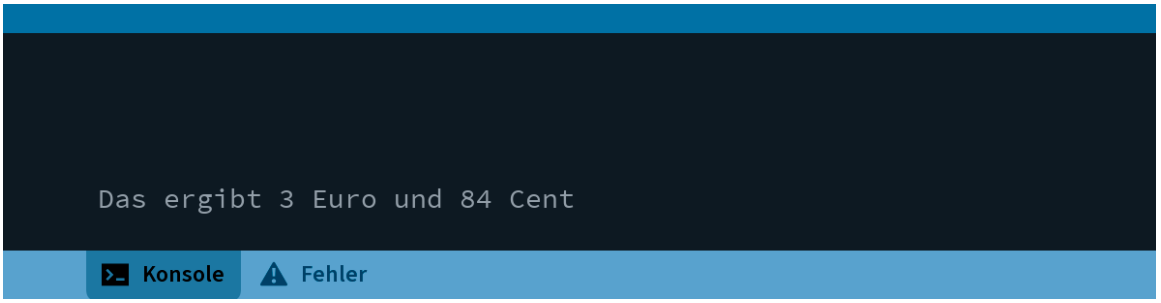
Schreiben Sie ein Programm, das mit Hilfe des **Modulo-Operators** für eine gegebene Uhrzeit ausrechnet, wie spät es nach einer gegebenen Anzahl von Stunden sein wird und das Ergebnis wie folgt ausgibt:



Verwenden Sie unterschiedliche Variablen für die Uhrzeiten und die Anzahl an Stunden. Es müssen nur ganzzahlige Werte für die Stunden berücksichtigt werden. Weisen Sie auch hier den Variablen für die Uhrzeit und die Anzahl an Stunden unterschiedliche Werte zu, kompilieren Sie das Programm jeweils, führen Sie es aus und kontrollieren Sie das Ergebnis.

Aufgabe 10: Cent zu Dollar

Schreiben Sie ein Programm, in dem eine `int`-Variable mit dem Namen `cent` deklariert wird. Initialisieren Sie die Variable z. B. mit dem Wert `384`. Schreiben Sie dann Programmcode, der diesen Wert in Euro und Cent zerlegt und das Ergebnis wie folgt ausgibt:



Verwenden Sie für dieses Programm **Ganzzahl-Arithmetik**. Schauen Sie sich ggf. noch einmal den Modulo-Operator an.

Aufgabe 11: Karussell

Ein Karussell besitzt eine gegebene Anzahl von Armen, auf denen jeweils eine Gondel montiert ist. In jeder Gondel finden eine gegebene Anzahl von Fahrgästen Platz. Schreiben Sie ein Programm, das für eine anstehende Karussellfahrt und eine gegebene Anzahl von Fahrgästen mit Hilfe des **Modulo-Operators** ausrechnet, wie viele Fahrgäste auf die nächste Fahrt warten müssen und das Ergebnis wie folgt ausgibt:

Sie können davon ausgehen, dass die Karussellfahrt so begehrt ist, dass die Anzahl der Interessenten bzw. Fahrgäste immer größer gleich der Anzahl der Plätze im Karussell ist.

Aufgabe 12: Wechselgeld

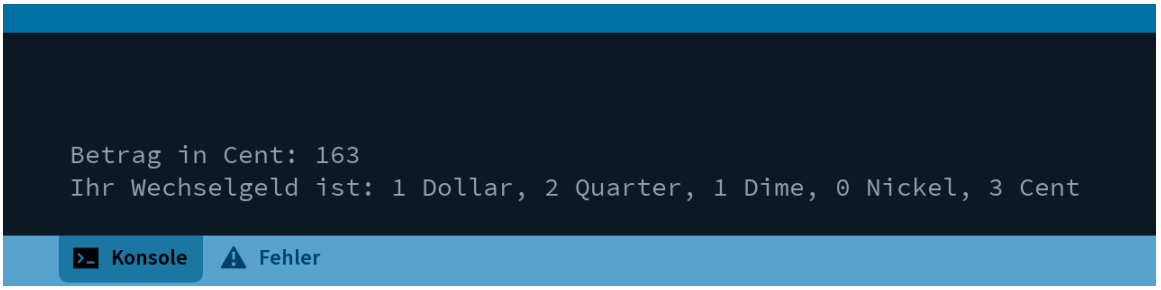
Wenn Sie in einem amerikanischen Geschäft an der Kasse Ihr Wechselgeld bekommen, gibt Ihnen der Kassenmitarbeiter zuerst Dollar, dann Quarter (25 Cent), dann Dime (10 Cent), dann Nickel (5 Cent) und schließlich Cent zurück. 163 Cent teilen sich dabei z. B. wie folgt auf:

- Ein Dollar passt in 163 Cent, Rest 63 Cent.
- Zwei Quarter passen in 63 Cent, Rest 13 Cent.
- Ein Dime passt in 13 Cent, Rest 3 Cent.
- Es wird kein Nickel gebraucht.
- Drei Cent bleiben übrig.

Ihr Wechselgeld ist dann also: 1 Dollar, 2 Quarter, 1 Dime, 0 Nickel, 3 Cent

Schreiben Sie ein Programm, das für einen in einer Variable gegebenen Cent-Wert die darin enthaltene Anzahl an Dollar, Quarter, Dime, Nickel und Cent ausgibt. Alle mathematischen Operationen sind ganzzahlig. Wenn Sie nicht mehr weiter wissen, hilft es oft, das Problem zunächst exemplarisch mit Papier und Stift anzugehen.

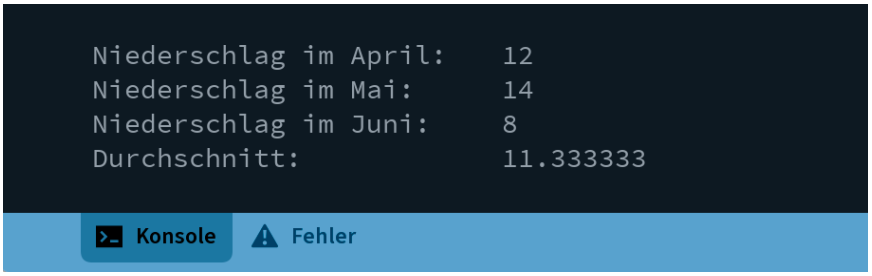
Programmausgabe:



Übungsaufgaben zu Gleitkommazahlen

Aufgabe 1: Durchschnittliche Niederschlagsmenge

Legen Sie für das Übungsblatt eine neue Sketch an und schreiben Sie dann in der Quellcodedatei Programmcode für die Berechnung der durchschnittlichen Niederschlagsmenge für die drei Monate April, Mai und Juni. Deklarieren und initialisieren Sie zu diesem Zweck für jeden Monat eine `int`-Variable und weisen Sie dieser einen Wert zu.
Berechnen Sie dann den Durchschnittswert als Gleitkommazahl (`float`) und speichern diesen in einer weiteren Variable. Geben Sie schließlich in der folgenden Weise alle Variablenwerte auf die Konsole aus:



Link auf die Processing Reference zu Gleitkommazahlen: float

Verwenden Sie das Tabulatorzeichen `\t`, um die Zahlen auszurichten. Zeilenwechsel innerhalb von Zeichenketten können mit `\n` erzwungen werden. Dies ist hier allerdings nicht erforderlich.

Aufgabe 2: Fläche eines Kreises

Schreiben Sie ein neues Programm, in dem zunächst eine Variable für den Radius r eines Kreises deklariert wird, aus diesem Wert dann die Fläche (`float`) $f = \pi \cdot r^2$ des Kreises berechnet und dieser Wert schließlich in der unten dargestellten Weise auf die Konsole ausgegeben wird.

Die Konstante π steht durch die Mathematikbibliothek bereits zur Verfügung. Benutzt wird die Konstante, indem der Name benutzt wird: `PI`

Beispielablauf für einen Radius von 10:

Was passiert wenn wir für die Fläche anstatt eines `float` ein `double` verwenden?

Aufgabe 3: Exponentialfunktion

Kommentieren Sie den Code in Ihrer Sketch aus und übernehmen Sie dann den folgenden Code in Ihre Sketch:

```
float wert = 32;
println("e hoch " + wert + " ergibt " + exp(wert));
```

Nachdem in der vorigen Aufgabe die Konstante π bzw. `PI` zum Einsatz kam, benutzen wir jetzt die Funktion bzw. *Methode* `exp` der Mathematikbibliothek. Im Rahmen unserer Programmiersprache werden mathematische Funktionen als Methoden bezeichnet. Warum das so ist wird später geklärt, wenn wir zu dem Thema objektorientierte Programmierung kommen. Die Methode `exp` dient zur Berechnung der Exponentialfunktion e^x. Benutzt bzw. *aufgerufen* wird die Methode, indem man ihren Namen verwendet: `exp(wert)`

Die Methode `exp` liefert einen Zahlenwert, der einer Variable zugewiesen werden kann oder direkt in einem Ausdruck oder einer Ausgabeanweisung benutzt werden muss. Anderenfalls verschwindet dieser. Der Wert, den die Methode liefert, ist eine Gleitkommazahl und wird im Beispiel direkt in der Ausgabeanweisung benutzt.

Die Zeichenkette für die Ausgabe in der Methode `println` oben hat den folgenden Aufbau:

`Zeichenkette1 + Variable + Zeichenkette2 + Ausdruck`

Dabei besteht der Bestandteil "Ausdruck" nur aus dem Aufruf der Methode `exp`. Mit Hilfe des Pluszeichens können Zeichenketten nicht nur mit Variablen, sondern auch mit weiteren Zeichenketten und Ausdrücken zu einer Gesamtzeichenkette verbunden werden. Ein Methodenaufruf repräsentiert für sich allein also schon einen Ausdruck, kann aber auch Teil eines größeren Ausdrucks sein. Eine Gesamtzeichenkette kann aus beliebig vielen Teilzeichenketten und Variablen bzw. Ausdrücken bestehen.

Beachten Sie, wie die obige Programmausgabe in mehreren Schritten aus Zeichenketten und Variablen- sowie Methodenwerten zusammengesetzt wird.

Führen Sie die Sketch aus und vergleichen Sie die Programmausgabe mit dem Ausgabebefehl oben.

Programmausgabe:

```
e hoch 32.0 ergibt 7.8962957E13
```

Erhöhen Sie dann schrittweise den Wert des Exponenten `wert` und führen Sie das Programm aus, bis ein Problem auftritt. Wann tritt das Problem auf und wie äußert sich dies? Woran könnte dies liegen?

Aufgabe 4: Trigonometrie

Um den **Sinus** eines Gleitkommawertes als Gleitkommawert zu berechnen, verwenden Sie die Sinusmethode aus der Mathematikbibliothek:

```
sin(wert)
```

Der `wert`, d. h. das Argument der Methode, wird in Bogenmaß angegeben und nicht in Grad.
Der **Cosinus** wird folgendermaßen berechnet:

```
cos(wert)
```

Schreiben Sie ein neues Programm, in dem ein Bogenmaßwert als Gleitkommazahl (float) in einer Variable gespeichert und folgendes berechnet wird:

1. Der Sinus von 0.3 Bogenmaß. Das Ergebnis wird in einer Variablen gespeichert.
2. Der Cosinus von 0.3 Bogenmaß. Das Ergebnis wird in einer anderen Variablen gespeichert.
3. Die Summe des Quadrats von jedem dieser Werte (verwenden Sie die Variablen). Das Ergebnis wird in einer dritten Variablen gespeichert.
4. Geben Sie die Werte der drei Variablen aus.

Die Ausgabe sollte in etwa so aussehen.

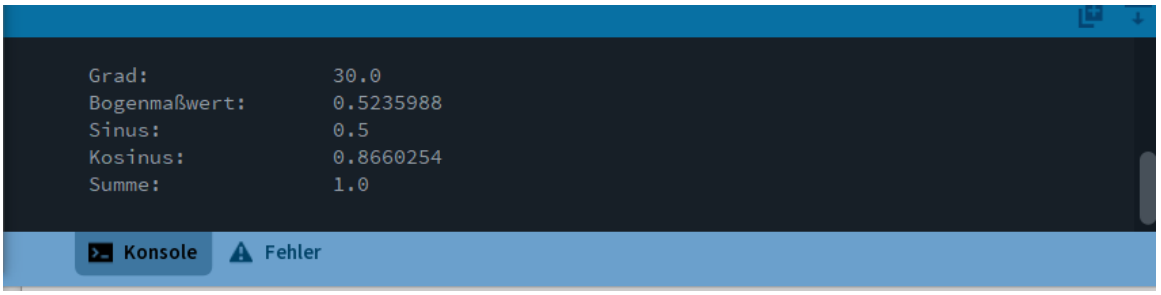
Testen Sie neben 0.3 ein paar weitere Werte. Was stellen Sie jeweils fest?

Aufgabe 5: Grad in Bogenmaß

Es ist manchmal schwierig in Bogenmaß zu denken, weil wir daran gewöhnt sind, mit Grad zu arbeiten. In den grundlegenden Mathematikfunktionen der Computer wird aber i. d. R. Bogenmaß verwendet. Erinnern Sie sich, dass 180 Grad in Bogenmaß dem Wert von π entspricht. Um einen in Grad gegebenen Winkel in Bogenmaß umzurechnen, benutzen Sie die folgende Anweisung:

```
bogenmass = grad * PI/180
```

Editieren Sie die vorherige Methode in der Weise, dass sie nun einen Winkel von 30 Grad verarbeiten kann.



Aufgabe 6: Das Ohmsche Gesetz

Das Ohmsche Gesetz setzt den **Widerstand** eines elektrischen Geräts in Beziehung zu dem elektrischen **Strom**, der durch das Gerät fließt und der angelegten **Spannung**. Das Gesetz ist:

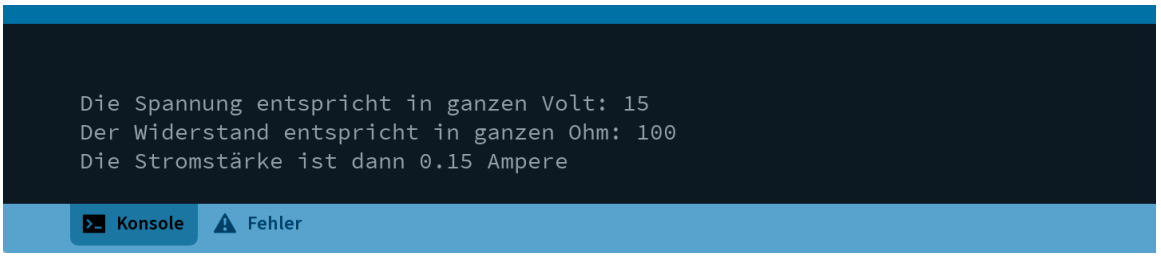
```
R = U/I
```

Hierbei ist U die Spannung (gemessen in Volt), I die Stromstärke (gemessen in Ampere) und R ist der Widerstand (gemessen in Ohm).

Schreiben Sie ein Programm, in dem Integer-Variablen für die Spannung und den Widerstand eines Gleichstromgeräts deklariert und initialisiert werden.

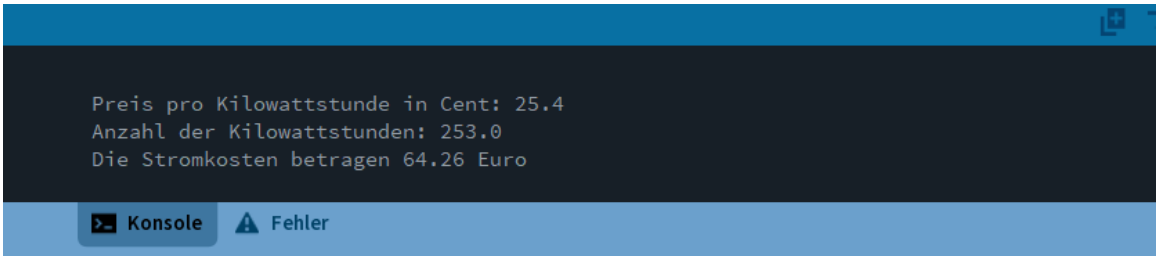
Das Programm soll dann die Stromstärke, die durch das Gerät fließt, berechnen und ausgeben. Verwenden Sie hierfür **Gleitkomma-Arithmetik**.

Ausgabe des Programms:



Aufgabe 7: Stromkosten

Schreiben Sie ein Programm, das die Stromkosten eines elektrischen Geräts berechnet. Im Programm werden zuerst Variablen für die Kosten je Kilowattstunde in Cent und die Anzahl der verbrauchten Kilowattstunden definiert. Tragen Sie jeweils `float`-Gleitkommawerte ein.



Sie erhalten bestimmt eine andere Ausgabe:

```
Die Stromkosten betragen 64.26199 Euro
```

Da es nichts kleineres als Cent gibt, muss nach der zweiten Kommastelle gerundet werden.

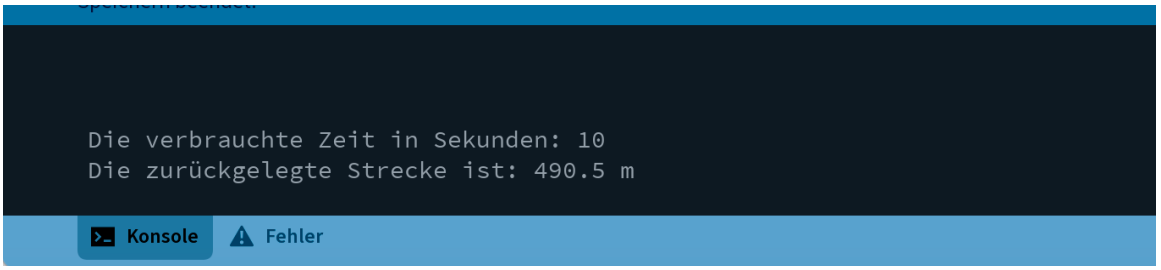
```
println("Die Stromkosten betragen " + round(kosten*100)/100.0 + " Euro");
```

Aufgabe 8: Fallender Stein

Wenn ein Stein von einem Turm fällt, nimmt er an Geschwindigkeit zu, bis er die Erde erreicht hat. Die dabei zurückgelegte Strecke wächst quadratisch über der Zeit nach dem "Weg-Zeit-Gesetz":

$$s = \frac{1}{2} g t^2$$

Hier ist *s* die Strecke in m, die in der Zeit *t* in Sekunden zurückgelegt wurde. *g* ist die Standard-Fallbeschleunigung: *g* = 9,80665 ≈ 9.81 m/s². Schreiben Sie ein Programm, in dem eine `float`-Variable für die verbrauchte Zeit definiert und mit deren Wert die zurückgelegte Strecke berechnet und ausgegeben wird:



Die Methode `pow(x, y)` aus der Mathematikbibliothek liefert die Potenz x^y.

Aufgabe 9: Mittelwertberechnung

Wenn es darum geht, zu einer Menge von Zahlen den Mittelwert zu bestimmen, so wird zumeist das *arithmetische* Mittel berechnet:

$$m_a = \frac{1}{n} (a_1 + a_2 + ... + a_n)$$

Es gibt allerdings Situationen, in denen andere Mittelwertbildungen sinnvoller sind. Ist man z. B. an dem durchschnittlichen Wachstum einer Größe interessiert, wobei sich die einzelnen Wachstumsschritte gegenseitig beeinflussen, wie z. B. beim Zinseszins, so muss das *geometrische* Mittel benutzt werden:

$$m_g = \sqrt[n]{a_1 \cdot a_2 \cdot \ldots \cdot a_n}$$

Daneben gibt es aber z. B. auch das *harmonische* Mittel, welches z. B. bei der Berechnung von Durchschnittsgeschwindigkeiten angewendet wird:

$$m_h = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}, \text{ mit } x_i > 0.$$

Schreiben Sie ein Programm, das drei Gleitkommazahlen in `float`-Variablen speichert und für diese das arithmetische, geometrische und harmonische Mittel berechnet und ausgibt:

```
Sie verwenden die drei Gleitkommazahlen: 2.0, 3.0, 10.0
Arithmetisches Mittel: 5.0
Geometrisches Mittel: 3.9148679
Harmonisches Mittel: 3.2142856
```

Konsole Fehler

Lernvideo zu geometrischem und harmonischem Mittelwert (im Labor sind Kopfhörer erforderlich).

Aufgabe 10: Logarithmus

Schreiben Sie ein Programm, in dem eine Gleitkommazahl als `float`-Wert und eine Basis *b* als `int`-Wert gespeichert und dann der Logarithmus zur Basis *b* berechnet und ausgegeben wird.

```
Gegeben ist die Gleitkommazahl: 65536
Verwendet wird die Basis: 2
Der Logarithmus von 65536.0 zur Basis 2 beträgt: 16.0
```

Konsole Fehler

```
Gegeben ist die Gleitkommazahl: 65536
Verwendet wird die Basis: 8
Der Logarithmus von 65536.0 zur Basis 8 beträgt: 5.3333335
```

Konsole Fehler

Für die Logarithmusberechnung kennt Processing den *natürlichen* Logarithmus $\log_e$ mit der Zahl *e* als Basis und umgesetzt mit der Methode `log(x)` (mit $x > 0$).

Die Umrechnung von Logarithmen unterschiedlicher Basen geschieht an Hand folgender Formel:

$$\log_{\mathrm{b}}(x) = \frac{\log_e(x)}{\log_e(\mathrm{b})}$$

Der Logarithmus zur Basis 2 einer Zahl x ist beispielsweise definiert als: $\log_2(x) = n$ wenn $2^n = x$.

Beispiele:

- $2^5 = 32$; also ist $\log_2(32) = 5$
- $2^{10} = 1024$; also ist $\log_2(1024) = 10$

Übungsaufgaben zum Thema einfache Fallunterscheidungen

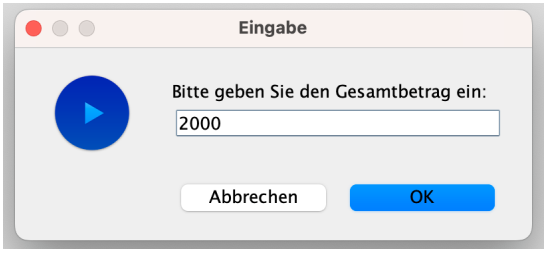
Ab jetzt brauchen wir für die Lösung der Übungsaufgaben auch eine Dateneingabe (Vorlesungsfolien ab Folie "Dateneingabe"). Dazu verwenden wir eine eigene Library: InOut in Processing. Bitte laden Sie die ZIP-Datei unter dem vorstehend angegebenen Link herunter und entpacken Sie diese in den Unterordner "libraries" in Ihrem "Processing"-Ordner.

Aufgabe 1: Discountpreise

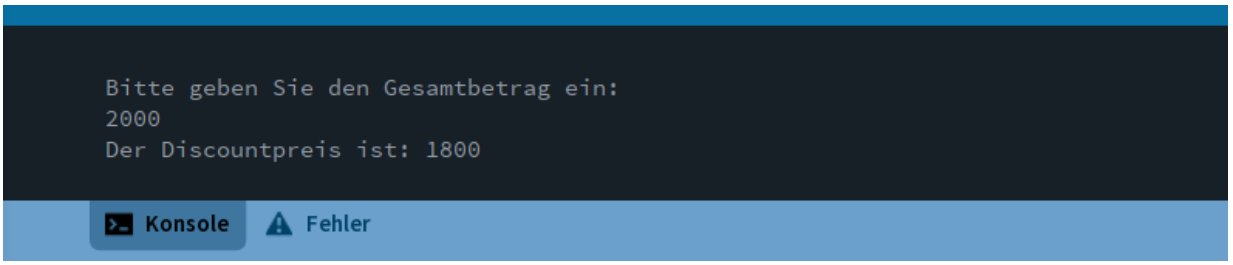
Während einer Sonderaktion wird ein Rabatt von 10% auf alle Einkäufe mit einem Gesamtbetrag von mehr als €10.00 gewährt. Schreiben Sie ein Programm, das nach dem Gesamtbetrag fragt und den Discountpreis berechnet. Der Gesamtbetrag wird in Cent (als Ganzzahl) eingegeben.

Verwenden Sie Ganzzahl-Arithmetik. In Ihrem Programm dürfen dann u. a. keine Kommazahlen stehen.

Lesen Sie zunächst - wie auf den Vorlesungsfolien beschrieben - mit Hilfe der InOut-Library den Wert für den Gesamtbetrag ein:



Die Bibliothek protokolliert den Eingabevorgang auf der Konsole. Berechnen Sie dann den Discountpreis und geben Sie diesen aus, sodass insgesamt die folgende Ausgabe für den Beispielwert 2000 auf der Konsole erscheint:

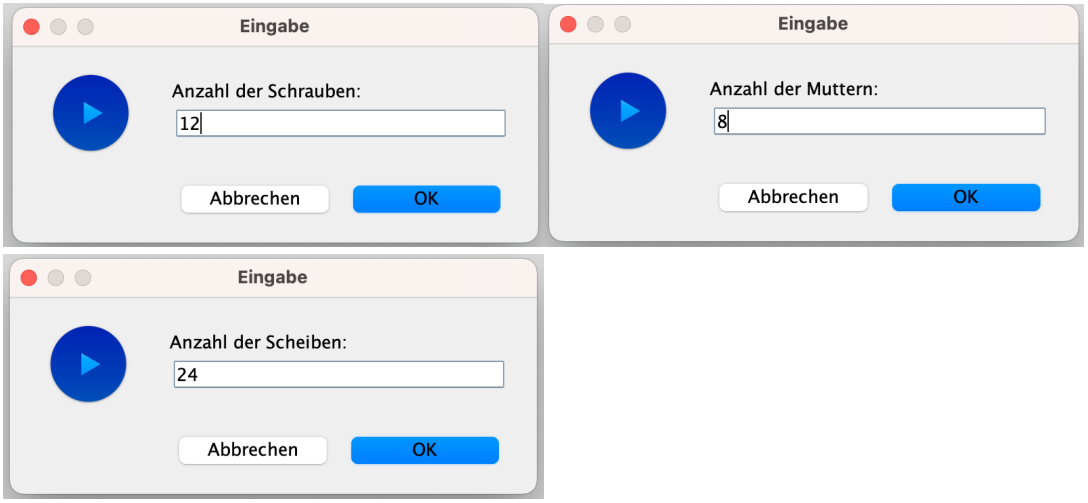


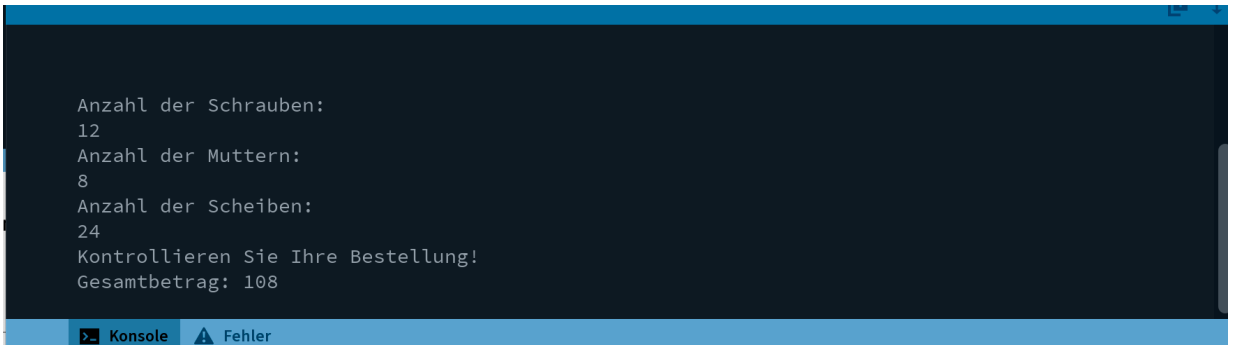
Aufgabe 2: Überprüfung der Bestellung

Bob's Schraubendiscount verlangt folgende Preise:

- 5 Cent pro Schraube
- 3 Cent pro Mutter
- 1 Cent pro Unterlegscheibe

Schreiben Sie ein Programm, das nach der Anzahl der Schrauben, Muttern und Unterleg- (auch: Beileg-) scheiben fragt, und dann den Gesamtbetrag berechnet und ausgibt. Zusätzlich überprüft das Programm die Bestellung. Wenn die Anzahl der bestellten Schrauben von der Anzahl der bestellten Muttern abweicht, gibt das Programm die Meldung "Kontrollieren Sie Ihre Bestellung!" aus. Anderenfalls gibt das Programm "Die Bestellung ist in Ordnung" aus. In jedem Fall wird der Gesamtbetrag ausgegeben.





Verwenden Sie für die Artikelpreise Konstanten. Konstanten werden in Großbuchstaben geschrieben.

Für den Ein- und Ausgabevorgang werden wir bei den Beispielabläufen ab jetzt häufig die folgende, kompaktere Darstellung nutzen:

```
Anzahl der Schrauben: 12
Anzahl der Muttern: 8
Anzahl der Scheiben: 24
Kontrollieren Sie Ihre Bestellung!
Gesamtbetrag: 108
```

Aufgabe 3: Letzte Tankmöglichkeit vor Death Valley

Al's Last Chance Gas Station befindet sich an der Straße 190 am Rande des **Death Valley**. Die nächsten 200 Meilen gibt es keine weitere Tankstelle. Schreiben Sie ein Programm, das den Fahrern hilft, zu entscheiden, ob getankt werden sollte oder nicht. Das Programm liest folgende Daten ein:

- Die Tankkapazität in Gallonen (3,78l).
- Die Benzinanzeige in Prozent (voll = 100, 3/4-voll = 75 usw.).
- Den Benzinverbrauch des Fahrzeugs in Meilen pro Gallone.

Die Ausgabe des Programms ist "Tanken!", oder "Weiterfahren." Je nachdem, ob das Fahrzeug genug Benzin für 200 Meilen hat oder nicht.

Verwenden Sie für die Eingabe und die Arithmetik **ganze Zahlen**.

Beispielablauf 1:

Beispielablauf 2:

Aufgabe 4: Dampfmaschinenwirkungsgrad

Der thermische Wirkungsgrad einer Dampfmaschine hängt ab vom Temperaturunterschied zwischen der Umgebungstemperatur und der des Dampfs. Die Obergrenze des thermischen Wirkungsgrads ist der Carnot-Wirkungsgrad:

$$\eta_c = 1 - \frac{T_n}{T_h}$$

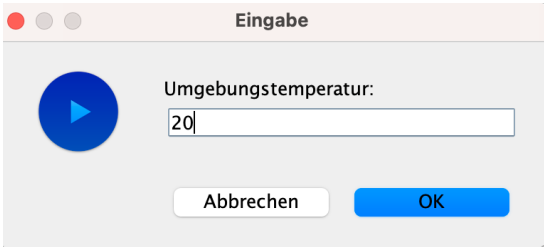
Dabei ist T_n die niedrigste (Umgebung) und T_h die höchste (Dampf) im Prozess auftretende Temperatur in Kelvin (K). 0°C entsprechen 273,15K.

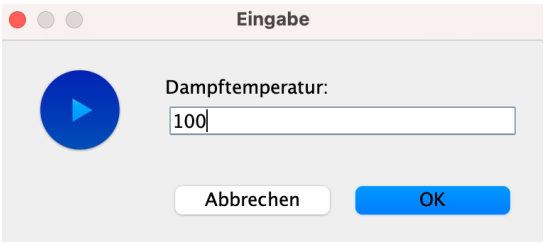
Schreiben Sie ein Programm, dass die Umgebungs- und Dampftemperatur in Grad Celsius einliest und hieraus die Obergrenze für den thermischen Wirkungsgrad in Prozent berechnet.

Unterhalb von 373,15K versiegt die Dampfbildung, sodass der Wirkungsgrad 0 ist.

Führen Sie die Berechnungen in Gleitkommaarithmetik durch.

Beispielablauf 1:

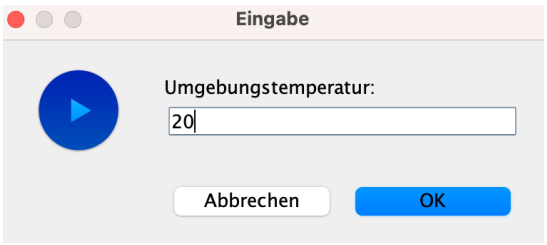


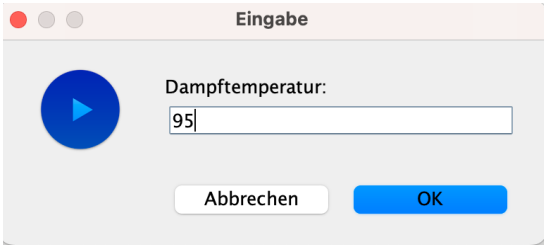



```
Umgebungstemperatur:
20
Dampftemperatur:
100
Maximaler thermischer Wirkungsgrad: 21.439098 %
```

Konsole Fehler

Beispielablauf 2:






```
Umgebungstemperatur:
20
Dampftemperatur:
95
Keine Dampfbildung, Wirkungsgrad gleich 0
```

Konsole Fehler

Aufgabe 5: Fantasy Game

In einem Rollenspiel können dem Spielercharakter Punktwerte für drei verschiedene Charaktereigenschaften zugewiesen werden:

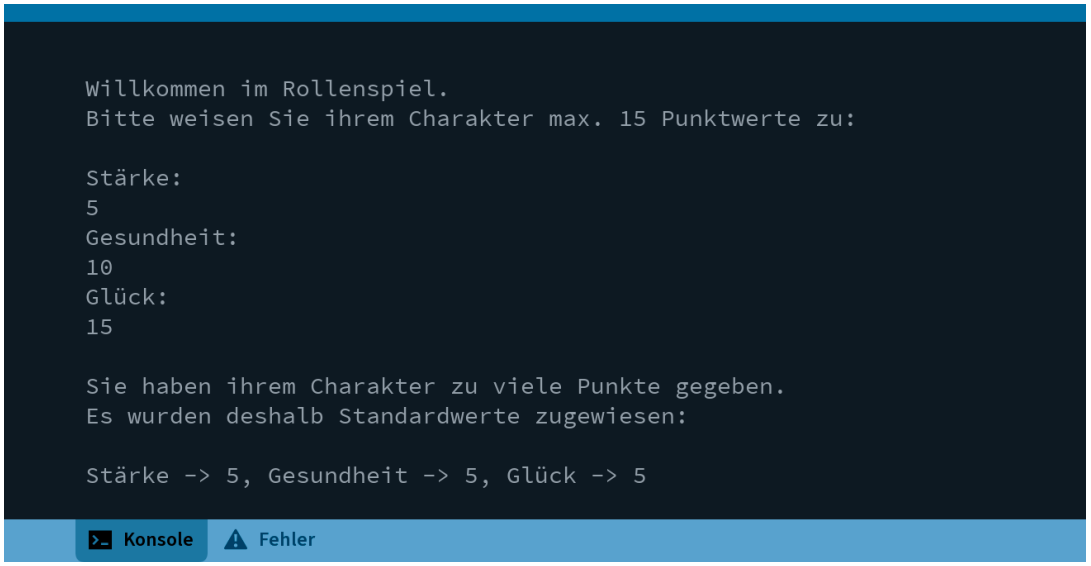
- Stärke
- Gesundheit
- Glück

Schreiben Sie ein Programm, das zu Beginn Werte für die drei Charaktereigenschaften erfragt. Die Gesamtsumme der vergebenen Punkte darf allerdings 15 nicht überschreiten. Anderenfalls

bekommen alle drei Charaktereigenschaften die Standardwerte 5 zugewiesen.

Beispielablauf:

```
Willkommen im Rollenspiel.
Bitte weisen Sie ihrem Charakter max. 15 Punktwerte zu:

Stärke: 5
Gesundheit: 10
Glück: 15
Sie haben ihrem Charakter zu viele Punkte gegeben.
Es wurden deshalb Standardwerte zugewiesen:

Stärke -> 5, Gesundheit -> 5, Glück -> 5
```



Aufgabe 6: Jahrhundertwechselproblem

Schreiben Sie ein Programm, das die Eingabe des Geburtsjahrs einer Person erfragt sowie das aktuelle Jahr und diese Angaben jeweils in zweistelliger Form - ohne Jahrhundertangabe - erwartet. Das Programm berechnet dann das Alter der Person und gibt dieses aus:

Beispielablauf 1:

```
Geburtsjahr: 01
Aktuelles Jahr: 13

Ihr Alter: 12
```

Beispielablauf 2:

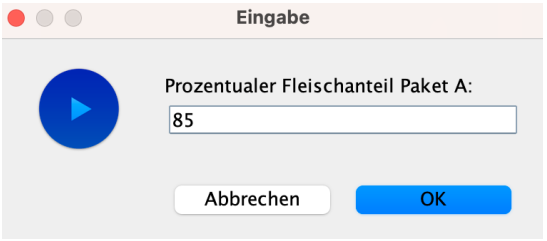
```
Geburtsjahr: 89
Aktuelles Jahr: 07

Ihr Alter: 18
```

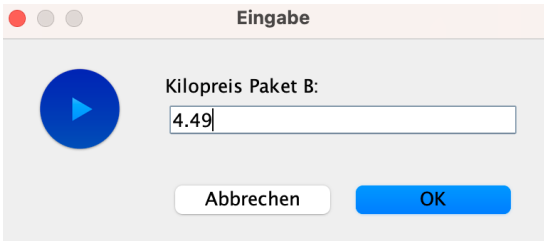
Aufgabe 7: Hackfleischangebote vergleichen

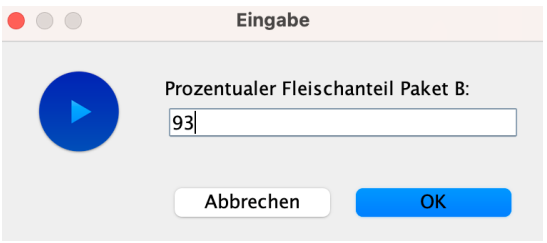
Die Hackfleischangebote unterschiedlicher Hersteller besitzen einen unterschiedlichen Fleisch- und Fettanteil. Je höher der Fettanteil eines Kilos Fleisch ist, umso geringer sein Wert.

Schreiben Sie ein Programm, das den Benutzer nach dem Kilopreis und dem Fettanteil für zwei unterschiedliche Fleischangebote fragt und das bessere Angebot bestimmt.

Beispielablauf:








```
Kilopreis Paket A:
3.99
Prozentualer Fleischanteil Paket A:
85
Kilopreis Paket B:
4.49
Prozentualer Fleischanteil Paket B:
93
Kilopreis des Fleischanteils von Paket A: 4.69

Kilopreis des Fleischanteils von Paket B: 4.83

Paket A besitzt den besseren Preis
```

Konsole Fehler

Die Situation, dass zwei Angebote exakt gleich gut sind, brauchen Sie nicht zu berücksichtigen.

Die Eingabe von Gleitkommazahlen geschieht in Processing **nicht** in der landesüblichen Weise, d. h. in Deutschland mit Komma, sondern in der angelsächsischen Form mit einem Punkt. Bei der Ausgabe wird von der Laufzeitumgebung ebenfalls die angelsächsische Form mit Punkt verwendet.

Wird der Punkt bei der Eingabe durch ein Komma ersetzt, so erfolgt üblicherweise ein

Syntaxfehler mit der Fehlermeldung: `Missing Operator ....`

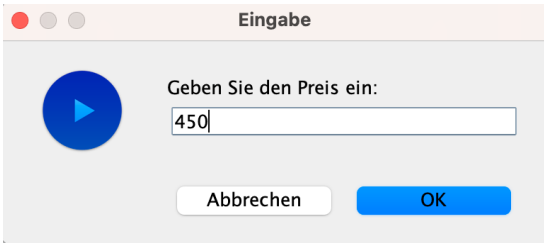
Übungsblatt zu logischen Ausdrücken sowie einzweigigen und geschachtelten Fallunterscheidungen

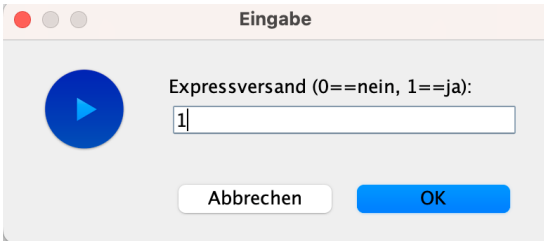
---

Aufgabe 1: Internet Delikatessen

Ein Delikatessengeschäft benötigt ein Programm zur Rechnungserstellung. Das Programm erfragt den Preis in Cent des gewünschten Artikels und ob der Artikel per Express verschickt werden soll. Für Artikel unter €10 betragen die Versandkosten €2.00. Kostet der Artikel €10 oder mehr betragen sie €3.00. Der Expresszuschlag beträgt €5.00.

Beispielablauf:






```
Geben Sie den Preis ein:
450
Expressversand (0==nein, 1==ja):
1

Rechnung:
  Preis: 4.5
  Versand: 7.0
  Gesamt: 11.5
```

Konsole Fehler

Aufgabe 2: Kontogebühr

Eine Bank verfährt nach der folgenden Regel: Wenn ein Kunde auf seinem Girokonto einen monatlichen Geldeingang von mehr als €2000 oder auf seinem Sparkonto ein Guthaben von

mehr als €15000 hat, wird keine Kontogebühr erhoben. Andernfalls wird eine Gebühr von €4,50 pro Monat erhoben. Schreiben Sie ein Programm, das nach dem Geldeingang und Guthaben fragt und dann ausgibt, ob eine Gebühr erhoben wird oder nicht.

Beispielablauf 1:

```
Willkommen bei der Geldbank.
Bitte geben Sie Ihren Geldeingang und Ihr Guthaben ein.

Geldeingang: 2050
Guthaben: 1000

Glückwunsch! Wir erheben keine Kontogebühren!
```

Beispielablauf 2:

```
Willkommen bei der Geldbank.
Bitte geben Sie Ihren Geldeingang und Ihr Guthaben ein.

Geldeingang: 1950
Guthaben: 5000

Wir müssen bei Ihnen leider Kontogebühren erheben.
```

Aufgabe 3: Reifendruck

Die beiden vorderen Reifen eines Autos sollten beide denselben Reifendruck haben. Ebenso sollten die beiden hinteren Reifen denselben Reifendruck haben, aber nicht unbedingt denselben wie die Vorderreifen. Schreiben Sie ein Programm, das den Reifendruck der vier Reifen einliest und dann eine Meldung ausgibt, ob der Reifendruck in Ordnung ist oder nicht.

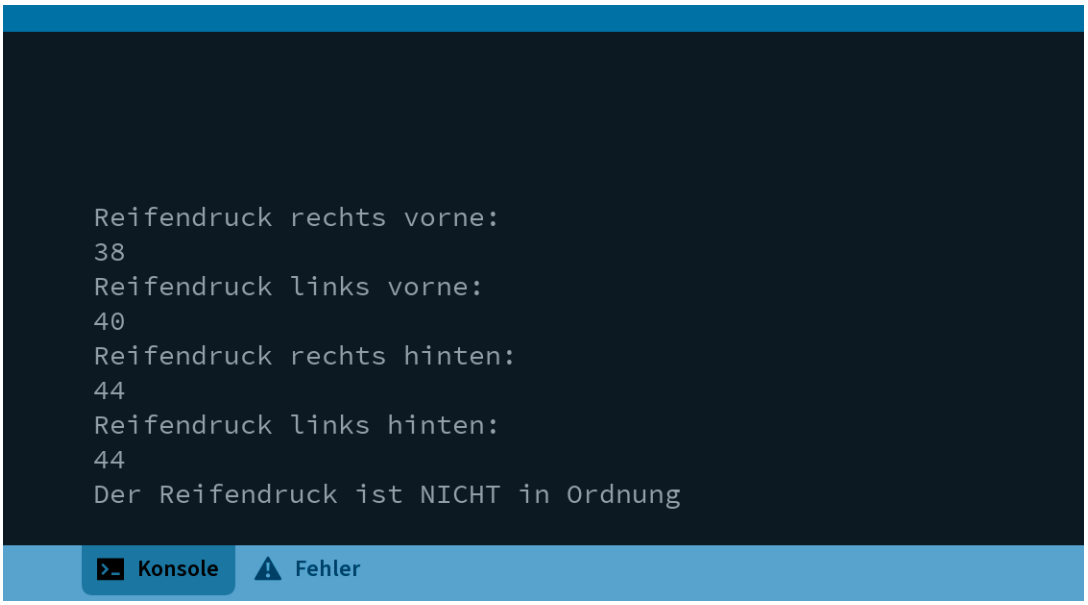
Beispielablauf 1:

```
Reifendruck rechts vorne: 38
Reifendruck links vorne: 38
Reifendruck rechts hinten: 42
Reifendruck links hinten: 42

Der Reifendruck ist in Ordnung
```

Beispielablauf 2:

```
Reifendruck rechts vorne: 38
Reifendruck links vorne: 40
Reifendruck rechts hinten: 44
Reifendruck links hinten: 44

Der Reifendruck ist NICHT in Ordnung
```

Zum Vergleich nochmal die Ausgabe unter Processing:



Aufgabe 4: Reifendruck zum Zweiten

Es genügt nicht, dass der Reifendruck der beiden Vorder- und Hinterräder derselbe ist. Die Drücke der beiden Reifen müssen sich ebenfalls jeweils innerhalb eines bestimmten Bereichs befinden. Erweitern Sie das Programm aus der vorigen Aufgabe in der Weise, dass zusätzlich überprüft wird, ob der Druck jedes Reifens sich zwischen 35 und 45 befindet. Befindet sich ein Reifen außerhalb dieses Bereichs, wird sofort eine Warnmeldung ausgegeben. Danach fährt das Programm mit dem Einlesen und Verarbeiten der Werte fort.

Kommt es zu einer Warnmeldung auf Grund des nicht eingehaltenen Druckbereichs, dann gibt das Programm am Ende zusätzlich eine letzte Warnmeldung aus. Diese Warnmeldung wird auch ausgegeben, wenn sich die Reifendrücke der beiden Vorderreifen oder der beiden Hinterreifen voneinander unterscheiden.

Deklarieren Sie für die Umsetzung dieser unterschiedlichen Situationen eine boolesche Variable `goodPressure`.

Initialisieren Sie die Variable `goodPressure` mit `true` und setzen Sie den Wert auf `false`, wenn sich ein Reifen außerhalb des gültigen Bereichs befindet.

Beispielablauf 1:

```
Reifendruck rechts vorne: 32
Warnung: Der Reifendruck ist außerhalb des erlaubten Bereichs

Reifendruck links vorne: 32
Warnung: Der Reifendruck ist außerhalb des erlaubten Bereichs

Reifendruck rechts hinten: 42
Reifendruck links hinten: 42

Der Reifendruck ist NICHT in Ordnung!
```

Beispielablauf 2:

```
Reifendruck rechts vorne: 38
Reifendruck links vorne: 40
Reifendruck rechts hinten: 44
Reifendruck links hinten: 44

Der Reifendruck ist NICHT in Ordnung!
```

Beispielablauf 3:

```
Reifendruck rechts vorne: 38
Reifendruck links vorne: 38
Reifendruck rechts hinten: 40
Reifendruck links hinten: 40

Der Reifendruck ist in Ordnung!
```

Aufgabe 5: Reifendruck zum Dritten

Reifen müssen nicht genau denselben Druck haben. Erweitern Sie Ihr Programm aus der vorigen Aufgabe dahingehend, dass sich zusätzlich die Drücke der beiden Vorder- und Hinterräder in einem Toleranzbereich von 3 psi bewegen dürfen.

Beispielablauf 1:

```
Reifendruck rechts vorne: 35
Reifendruck links vorne: 37
Reifendruck rechts hinten: 41
Reifendruck links hinten: 41

Der Reifendruck ist in Ordnung!
```

Beispielablauf 2:

```
Reifendruck rechts vorne: 35
Reifendruck links vorne: 37
Reifendruck rechts hinten: 40
Reifendruck links hinten: 45

Der Reifendruck ist NICHT in Ordnung!
```

Aufgabe 6: Mikrowellenherd

Ein Hersteller von Mikrowellen empfiehlt, bei der Erwärmung von Lebensmitteln 50% zur Erwärmungszeit zu addieren, wenn die Mikrowelle mit zwei Produkten gefüllt wird. Werden drei Produkte gleichzeitig erwärmt, so soll die Erwärmungszeit sogar verdoppelt werden. Mehr als drei Produkte gleichzeitig zu erwärmen wird nicht empfohlen.

Schreiben Sie ein Programm, dass nach der Anzahl zu erwärmender Produkte und der Erwärmungszeit für ein Produkt fragt und daraus die empfohlene Erwärmungszeit für alle Produkte berechnet.

Beispielablauf 1:

```
Bitte geben Sie die Erwärmungszeit des Produkts in Sekunden ein: 180
Bitte geben Sie die Anzahl der Produkte ein: 2

Die Erwärmungszeit beträgt: 270s
```

Beispielablauf 2:

```
Bitte geben Sie die Erwärmungszeit des Produkts in Sekunden ein: 180
Bitte geben Sie die Anzahl der Produkte ein: 3

Die Erwärmungszeit beträgt: 360s
```

Beispielablauf 3:

```
Bitte geben Sie die Erwärmungszeit des Produkts in Sekunden ein: 180
Bitte geben Sie die Anzahl der Produkte ein: 4

Fehler: Mehr als drei Produkte
```

Beispielablauf 4:

```
Bitte geben Sie die Erwärmungszeit des Produkts in Sekunden ein: 180
Bitte geben Sie die Anzahl der Produkt ein: -2

Falsche Produktanzahl
```

Entsprechend für Anzahl Produkte gleich 0 oder Erwärmungszeit <= 0.

Aufgabe 7: Torten essen

Beim "State Fair Pie Eating Contest" müssen die Teilnehmer in der Schwergewichtsklasse zwischen 235 und 265 Pfund wiegen. Schreiben Sie ein Programm, das nach dem Gewicht des Teilnehmers fragt und dann ausgibt, ob er oder sie zum Wettbewerb in der Schwergewichtsklasse zugelassen ist oder nicht.

Beispielablauf 1:

```
Bitte geben Sie ihr Gewicht in Pfund ein: 230
Sie sind leider zu leicht für die Schwergewichtsklasse
```

Beispielablauf 2:

```
Bitte geben Sie ihr Gewicht in Pfund ein: 250
Sie sind für die Schwergewichtsklasse zugelassen
```

Beispielablauf 3:

```
Bitte geben Sie ihr Gewicht in Pfund ein: 300
Sie sind leider zu schwer für die Schwergewichtsklasse
```

Aufgabe 8: Entscheidungsbaum

Implementieren Sie den in der Vorlesung erläuterten Entscheidungsbaum für das Sortieren dreier Zahlen (S.180). Das Programm soll die drei Zahlenwerte vom Benutzer erfragen und die sortierten Zahlenwerte in aufsteigender Reihenfolge ausgeben. (Dieses Programm werden wir in einer der folgenden Übungen wieder benötigen!)

Beispielablauf:

```
Erste Zahl: 100
Zweite Zahl: -6
Dritte Zahl: 14

Ergebnis: -6, 14, 100
```

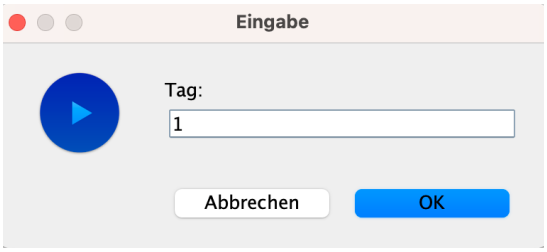
Aufgabe 9: `switch`-Anweisung

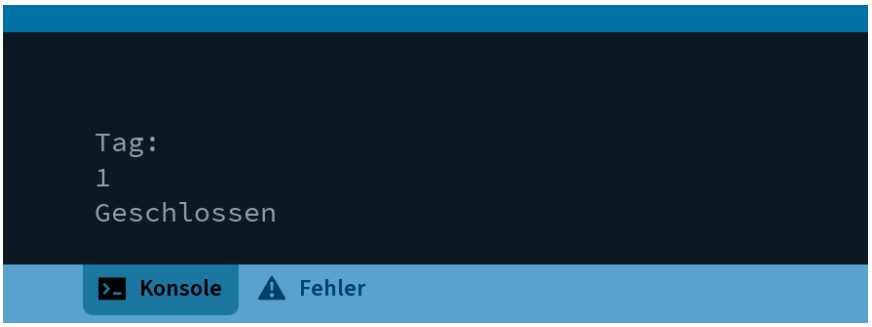
Die Wochentage Montag bis Sonntag seien durch die Zahlenwerte 1 bis 7 codiert. Ein Sekretariat hat Montag, Samstag und Sonntag geschlossen. Von Dienstag bis Donnerstag ist es von 09:00-15:30 geöffnet. Freitags hingegen nur von 09:00-13:00.

Schreiben Sie ein Programm, welches zunächst den Wochentag einliest und dann die dazu passenden Öffnungszeiten des Sekretariats ausgibt. Wird ein Zahlenwert außerhalb des Bereichs 1 bis 7 eingegeben, so erfolgt eine Fehlermeldung des Programms.

Beispielablauf 1:

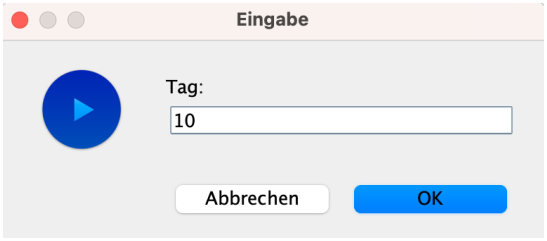
```
Tag: 1
Geschlossen
```

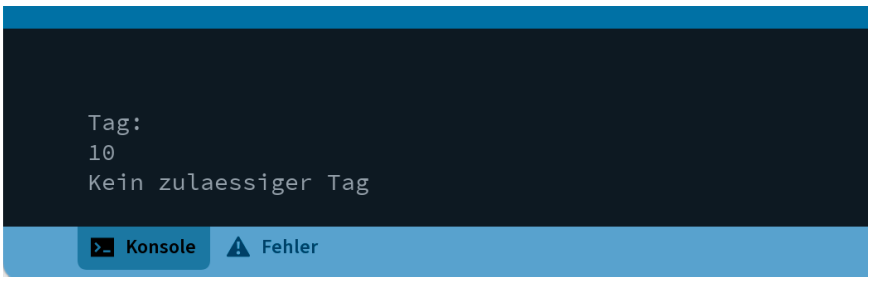




Beispielablauf 2:

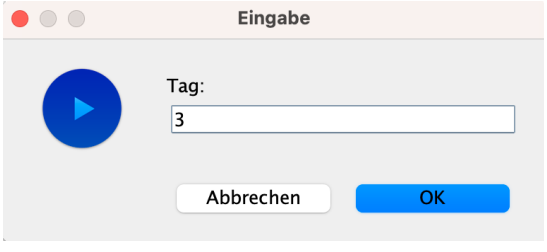
```
Tag: 10
Kein zulaessiger Tag
```





Beispielablauf 3:

```
Tag: 3
09:00-15:30
```



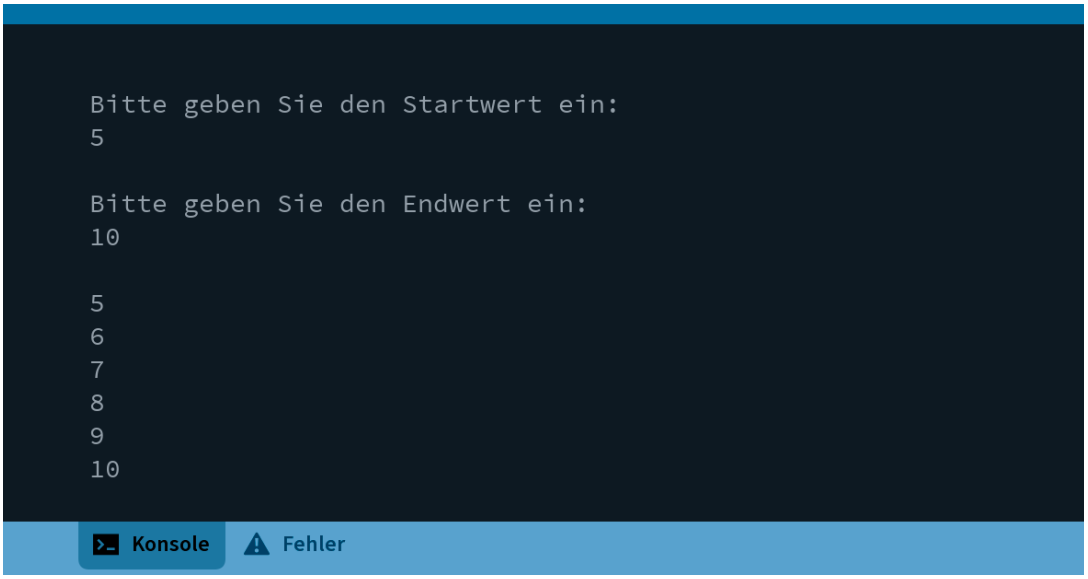

Tag:
3
09:00-15:30
Konsole
Fehler

Übungsaufgaben zum Thema Zählschleifen

Aufgabe 1: Zahlenbereich ausgeben

Schreiben Sie ein Programm, das nach einem Start- und Endwert fragt und dann alle Zahlen (Integer) inklusive der eingegebenen mit Hilfe einer `while`-Schleife ausgibt.

Beispielablauf :

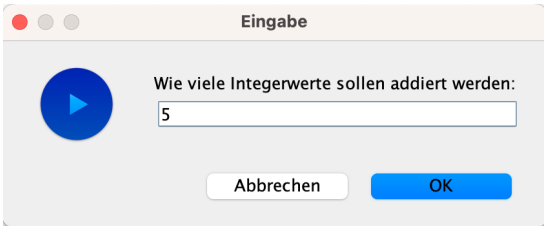


Aufgabe 2: Integer addieren

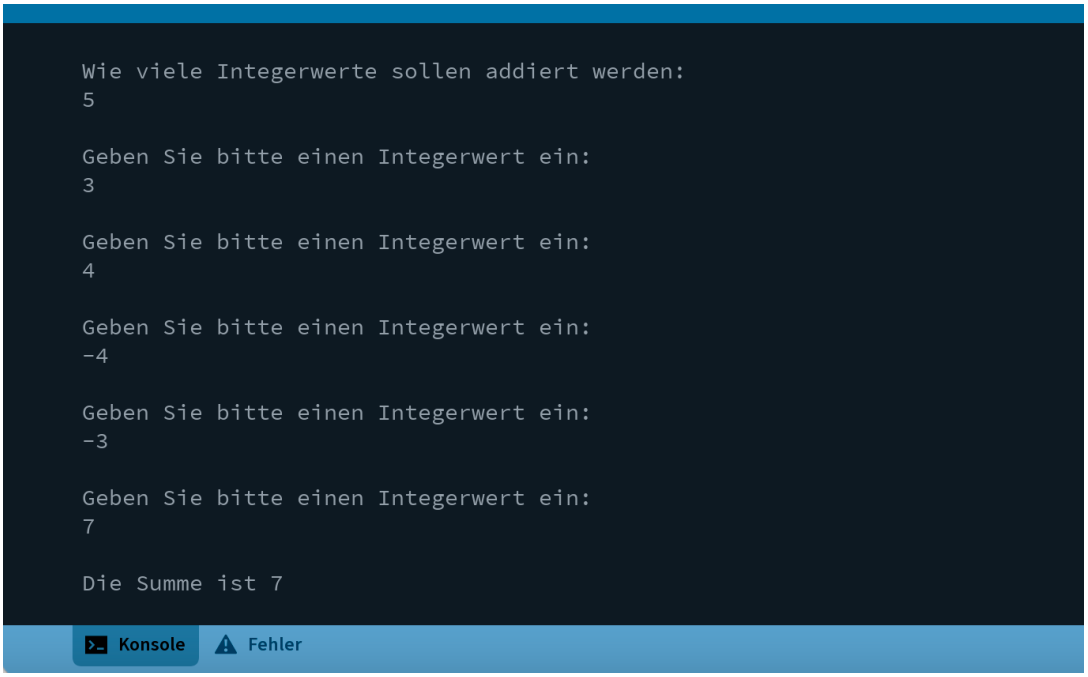
Schreiben Sie ein Programm, das mit Hilfe einer `for`-Schleife Integer addiert, die eingegeben werden. Das Programm fragt zuerst, wie viele Zahlen addiert werden sollen. Danach fordert das Programm auf, die Zahlen nacheinander einzugeben. Schließlich gibt es das Ergebnis auf den Bildschirm aus.

Beispielablauf:

```
Wie viele Integerwerte sollen addiert werden: 5
```




```
Geben Sie bitte einen Integerwert ein: 3

Geben Sie bitte einen Integerwert ein: 4

Geben Sie bitte einen Integerwert ein: -4

Geben Sie bitte einen Integerwert ein: -3

Geben Sie bitte einen Integerwert ein: 7
```



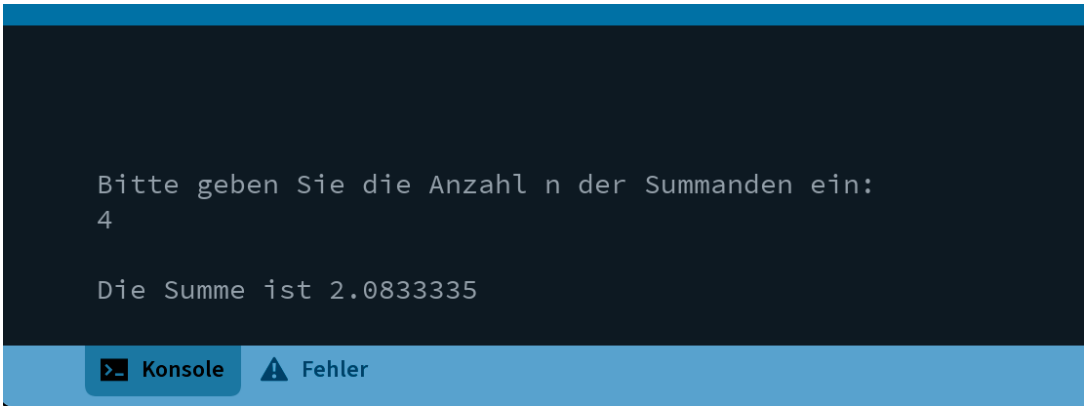
Aufgabe 3: Reihe berechnen

Schreiben Sie ein Programm, das mit Hilfe einer `for`-Schleife die folgende Summe berechnet:

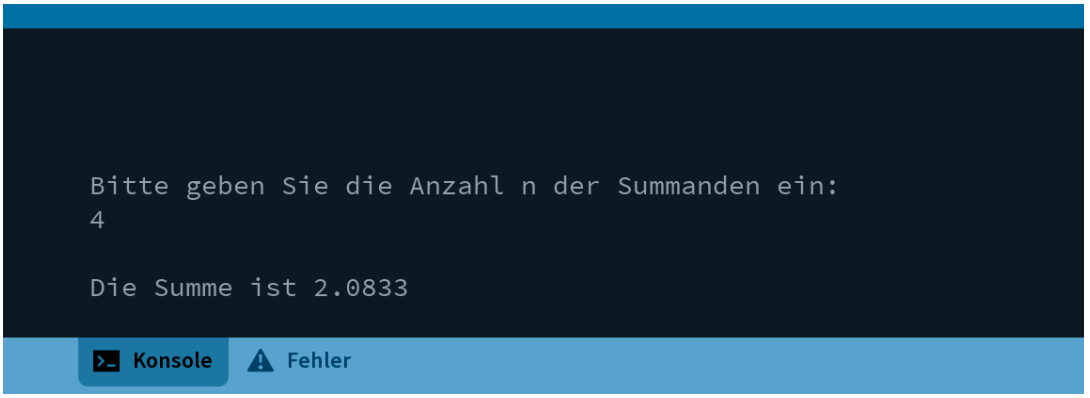
```
summe = 1.0/1 + 1.0/2 + 1.0/3 + 1.0/4 + 1.0/5 + … + 1.0/n
```

Die Variable n ist ein über die InOut-Bibliothek einzulesender Integerwert, bis zu dem die Reihe berechnet werden soll.

Beispielablauf :



Um einen Wert *val* auf eine Anzahl von *n* Stellen nach dem Komma zu runden, können Sie folgenden Ausdruck verwenden:
round(val * pow(10, n)) / pow(10, n)



Aufgabe 4: Addition von Quadrat- und Kubikzahlen

Schreiben Sie ein Programm, das Quadrat- und Kubikzahlen (als Integerwerte) von 1 bis n addiert. Dabei wird `n` wieder über die InOut-Bibliothek eingelesen.

Verwenden Sie nur eine einzelne `for`-Schleife.

Beispielablauf:

```
Bitte geben Sie n ein: 5

Die Quadratsumme ist: 55
Die Kubiksumme ist: 225
```

Aufgabe 5: Potenz einer Zahl

Schreiben Sie ein Programm, das mit Hilfe einer `for`-Schleife x^n berechnet, wobei `x` eine Gleitpunktzahl und `n` eine positive Integerzahl ist. `x` und `n` werden zu Beginn eingegeben.

Wird eine negative Zahl für `n` eingegeben, so wird vom Programm darauf hingewiesen, dass `n` positiv sein muss.

```
x^n = x * x * x * … * x
```

Die Operation `* x` wird `n`-mal durchgeführt.

Beispielablauf 1:

```
Bitte geben Sie x ein: 1.3

Bitte geben Sie n ein: 5

1.3 hoch 5 ergibt 3.7129
```

Beispielablauf 2:

```
Bitte geben Sie x ein: 5.6

Bitte geben Sie n ein: -3

n muss eine positive Integerzahl sein!
```

Beispielablauf 3:

```
Bitte geben Sie x ein: 7.25

Bitte geben Sie n ein: 0

7.25 hoch 0 ergibt 1.0
```

Aufgabe 6: Standardabweichung einer Menge von Eingabewerten

Schreiben Sie ein Programm, dass für eine Anzahl von Gleitkommazahlen, die über die InOut-Bibliothek eingelesen werden, mit Hilfe einer einzelnen `for`-Schleife die Standardabweichung berechnet.

Die Standardabweichung ist die Quadratwurzel aus der Varianz. Die *empirische* Varianz s^2 einer Menge von Eingabewerten ist die Summe der Abweichungsquadrate aller Eingabewerte von ihrem arithmetischen Mittel bzw. Durchschnittswert $\bar{x}$.

Sie wird nach der folgenden Formel berechnet:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2, \; n \geq 2.$$

Diese Formel ist für unsere Zwecke indes nicht brauchbar, da sie das Vorhandensein des Mittel-(Durchschnitts-)werts voraussetzt. Die eingegebenen Zahlen müssten also entweder zwischengespeichert werden, wofür wir bislang keine Möglichkeit besitzen, oder zweimal hintereinander eingegeben werden. Einmal für die Berechnung des Mittelwerts und einmal für die Berechnung der quadratischen Abweichung vom Mittelwert.

Eine bessere Alternative ist die folgende Formel für die Varianz, in die man die obige Formel umformen kann, und bei der der Mittelwert zeitgleich mit der quadratischen Abweichung berechnet wird:

$$s^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - n\bar{x}^2\right],\ n \geq 2.$$

Zunächst wird über die Konsole die Anzahl *n* der einzulesenden Zahlen eingegeben. Danach liest das Programm sukzessive die Zahlen ein und berechnet zwei Summen. Zum einen die Summe der eingegebenen Zahlenwerte, zum anderen die Summe der Quadrate der eingegebenen Zahlenwerte.

Aus der Summe der eingegebenen Zahlenwerte wird dann durch Teilen durch *n* der arithmetische Mittelwert berechnet und mit dessen Hilfe der in der eckigen Klammer stehende Teil der Formel.

Das Programm gibt schließlich die Standardabweichung aus.

Für die Berechnung der Quadratwurzel aus s^2 gibt es die Funktion `sqrt()` in der Mathematikbibliothek.

Beispielablauf:

```
Bitte geben Sie die Anzahl n der Eingabewerte ein: 4

Bitte geben Sie die 1. Zahl ein: 2

Bitte geben Sie die 2. Zahl ein: 3

Bitte geben Sie die 3. Zahl ein: 1

Bitte geben Sie die 4. Zahl ein: 2

Die Standardabweichung ist: 0.8165
```

Aufgabe 7: Zahlenfolgen

1. Schreiben Sie ein Programm, welches die Länge einer Zahlenfolge einliest und die Länge der Abschnitte, aus denen diese bestehen soll. Die Zahlenfolge wird dann mit Hilfe einer einzelnen `for`-Schleife ausgegeben. Dabei besteht jeder Abschnitt aus den Zählwerten von 1 bis zur Länge des Abschnitts. Ist das Ende der Zahlenfolge erreicht, bevor der letzte Abschnitt voll ist, so wird nur noch der in die Folge passende Teil des Abschnitts ausgegeben.

   Beispielablauf 1:

   ```
   Laenge der Zahlenfolge: 20

   Abschnittslaenge: 5

   Ergebnis:
   1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
   ```

Beispielablauf 2:

```
Laenge der Zahlenfolge: 14

Abschnittslaenge: 3

Ergebnis:
1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2,
```

2. Schreiben Sie Variationen Ihrer vorhandenen Lösung, mit denen sich die folgenden Zahlenfolgen generieren lassen:

   Beispielablauf 3:

```
Laenge der Zahlenfolge: 20

Abschnittslaenge: 5

Ergebnis:
+1, -2, +3, -4, +5, -1, +2, -3, +4, -5, +1, -2, +3, -4, +5, -1, +2, -3, +4, -5,
```

   Beispielablauf 4:

```
Laenge der Zahlenfolge: 20

Abschnittslaenge: 5

Ergebnis:
1, 2, 3, 4, 5, 4, 3, 2, 1, 2, 3, 4, 5, 4, 3, 2, 1, 2, 3, 4,
```

   Beispielablauf 5:

```
Laenge der Zahlenfolge: 20

Abschnittslaenge: 5

Ergebnis:
+1, -2, +3, -4, +5, -4, +3, -2, +1, -2, +3, -4, +5, -4, +3, -2, +1, -2, +3, -4,
```

Aufgabe 8: Fibonacci-Folge

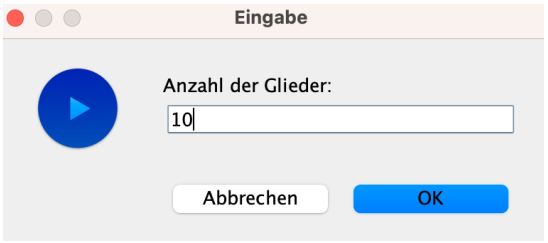
Die Fibonacci-Folge ist eine seit Jahrtausenden bekannte Folge von Zahlen mit starkem Bezug zu Wachstumsvorgängen in der Natur. Sie beschreibt z. B. die Anzahl von Spiralen bei den Fruchtständen vieler Pflanzen, bei Honigbienen die Anzahl der Ahnen einer Drohne oder kann auch zur Beschreibung des Wachstums einer Population von Kaninchen herangezogen werden. Die Glieder a_i der Fibonacci-Folge werden so gebildet, dass ausgehend von den Startwerten 1 und 1 jedes weitere Glied der Zahlenfolge durch Summieren der beiden vorherigen Glieder gebildet wird. Insgesamt ergibt sich so die folgende Zahlenfolge:

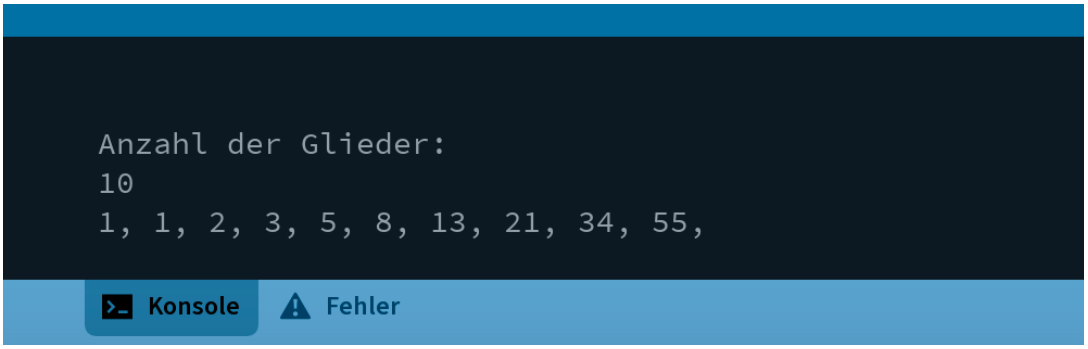
`1, 1, 2, 3, 5, 8, 13, usw.`

1. Schreiben Sie ein Programm, dass die Anzahl der gewünschten Glieder der Fibonacci-Folge einliest, diese dann in einer `for`-Schleife berechnet und in das Ausgabefenster ausgibt.

   Programmausgabe:

   ```
   Anzahl der Glieder: 10
   1, 1, 2, 3, 5, 8, 13, 21, 34, 55,
   ```

2. Die Fibonacci-Folge besitzt auch einen Bezug zum Goldenen Schnitt, der für das Seitenverhältnis `1,618033…` steht. Je mehr Folgenglieder a_i berechnet werden, um so mehr nähert sich der Quotient $q_i = \frac{a_i}{a_{i-1}}$ zweier aufeinander folgender Glieder diesem Seitenverhältnis an.

   Erweitern Sie Ihr Programm dahingehend, dass zusätzlich zu den Gliedern der Fibonacci-Folge der Wert dieses Quotienten ausgegeben wird.

   ```
   Anzahl der Glieder: 20
   1: a_i=1, q_i=./.
   2: a_i=1, q_i=1.0
   3: a_i=2, q_i=2.0
   4: a_i=3, q_i=1.5
   5: a_i=5, q_i=1.666667
   6: a_i=8, q_i=1.6
   7: a_i=13, q_i=1.625
   8: a_i=21, q_i=1.615385
   9: a_i=34, q_i=1.619048
   10: a_i=55, q_i=1.617647
   11: a_i=89, q_i=1.618182
   12: a_i=144, q_i=1.617978
   13: a_i=233, q_i=1.618056
   14: a_i=377, q_i=1.618026
   15: a_i=610, q_i=1.618037
   16: a_i=987, q_i=1.618033
   17: a_i=1597, q_i=1.618034
   ```

```
18: a_i=2584, q_i=1.618034
19: a_i=4181, q_i=1.618034
20: a_i=6765, q_i=1.618034
```

Da die Fibonacci-Folge sehr schnell wächst, werden Folgen mit mehr als 92 Gliedern den Bereich der Standard-Integertypen sprengen. Allerdings ist in diesen Bereichen der Wert für q_i längst ausreichend genau.

Übungsblatt zu den verschachtelten Zählschleifen

Allgemeine Hinweise zu den Aufgaben:

Lösen Sie die folgenden Aufgaben durch den Einsatz von `for`-Schleifen, dort wo es sinnvoll ist. Vermeiden Sie dabei `for`-Schleifen mit unvollständigen Schleifenköpfen. Überlegen Sie sich gegebenenfalls, wie diese sinnvoll gefüllt werden könnten oder nutzen Sie `while`-Schleifen.

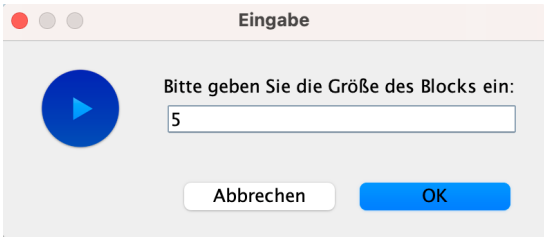
Verwenden Sie für die Erstellung der Grafiken in der Konsole nur die folgenden Befehle: `print("*")`, `println()` und `print(" ")`.

Aufgabe 1: Block von Sternen

Schreiben Sie ein Programm, das einen Block von Sternen ausgibt. Das Programm liest über die InOut-Bibliothek die Größe des Blocks ein und gibt danach eine entsprechende Anzahl von Zeilen von Sternen aus, wobei jede Zeile ebenfalls eine der Eingabe entsprechende Anzahl von Sternen hat.

Beispielablauf:

```
Bitte geben Sie die Größe des Blocks ein: 5

*****
*****
*****
*****
*****
```



Aufgabe 2: Verschobener Block von Sternen

Schreiben Sie ein Programm, das einen vom linken Rand verschobenen Block von Sternen ausgibt. Das Programm liest die Größe des Blocks ein und gibt danach eine korrespondierende Anzahl von Zeilen von Sternen auf der Konsole aus, wobei jede Zeile ebenfalls eine zur Eingabe passende Anzahl von Sternen hat. Auch der Abstand der Zeilen vom linken Rand entspricht dem eingegebenen Wert.

Beispielablauf:

```
Bitte geben Sie die Größe des verschobenen Blocks ein: 5

     *****
     *****
     *****
     *****
     *****
```

Aufgabe 3: Verschobener Rahmen von Sternen

Schreiben Sie ein Programm, das einen vom linken Rand verschobenen Rahmen von Sternen ausgibt. Das Programm liest die Größe des Rahmens ein und gibt danach einen Rahmen dieser Größe aus. Der Abstand des Rahmens vom linken Rand entspricht dem eingegebenen Wert.

Beispielablauf:

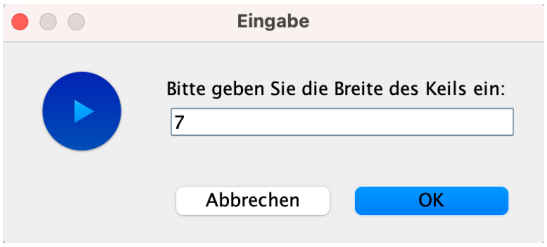
```
Bitte geben Sie die Größe des verschobenen Rahmens ein: 5

     *****
     *   *
     *   *
     *   *
     *****
```

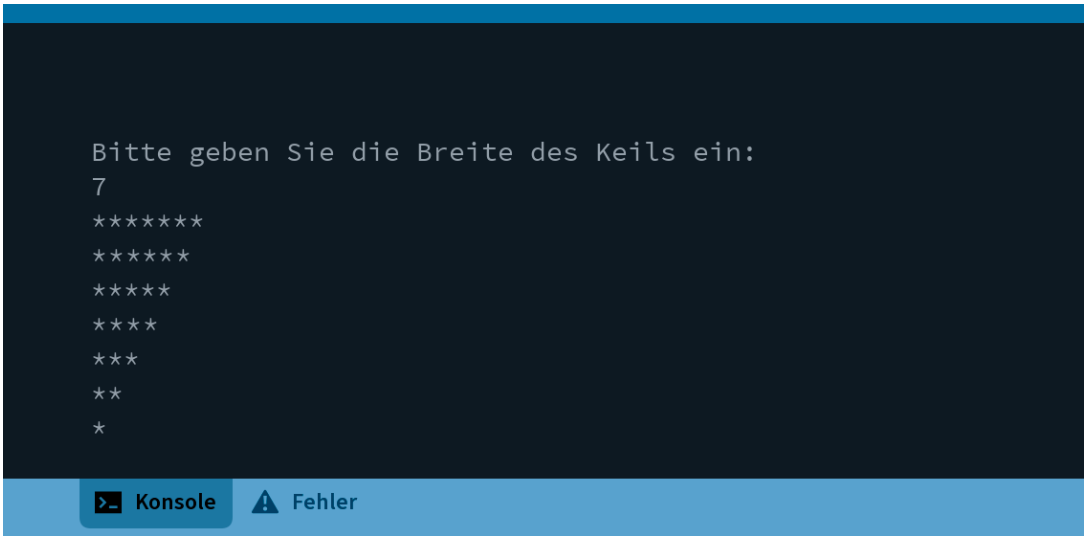
Aufgabe 4: Keil von Sternen

Schreiben Sie ein Programm, das Sterne in Keilform ausgibt. Das Programm liest die Anfangszahl für die Sterne ein und gibt danach Zeilen von Sternen aus, wobei jede Zeile einen Stern weniger hat, als die vorhergehende.

Beispielablauf :

```
Bitte geben Sie die Breite des Keils ein: 7

*******
******
*****
****
***
**
*
```





Aufgabe 5: Breiter Keil von Sternen

Schreiben Sie ein Programm, das Sterne in Keilform ausgibt. Das Programm liest die Höhe des Keils ein und gibt danach entsprechend viele Zeilen von Sternen aus, wobei jede Zeile zwei Sterne mehr hat, als die vorhergehende.

Beispielablauf:

```
Bitte geben Sie die Höhe des breiten Keils ein: 7
```

```
*
***
*****
*******
*********
***********
*************
```

Aufgabe 6: Baum

Schreiben Sie ein Programm, das einen Baum aus Sternen auf die Konsole ausgibt.

Die Größe des Baums soll durch Eingabe der folgenden drei Parameterwerte festgelegt werden können:

- Untere Breite der Baumkrone
- Breite und Höhe des Baumstamms

Für die Breitenangaben können ungeradzahlige Werte vorausgesetzt werden.

Beispielablauf :

```
Bitte geben Sie die Breite der Baumkrone ein: 15
Bitte geben Sie die Breite des Baumstamms ein: 5
Bitte geben Sie die Höhe des Baumstamms ein: 4
```

Wenn die Aufgabe zunächst zu schwierig erscheint, ist es sinnvoll, diese schrittweise zu lösen, wobei im ersten Schritt zunächst mit einem Baum fester Größe gearbeitet und erst später die Eingabe der Baumparameter hinzugefügt und die Lösung damit verallgemeinert wird.

Aufgabe 7: Weihnachtsbaum

Schreiben Sie ein Programm, das einen hübsch verzierten Weihnachtsbaum mit Kerzen aus i-Zeichen auf der Konsole ausgibt.

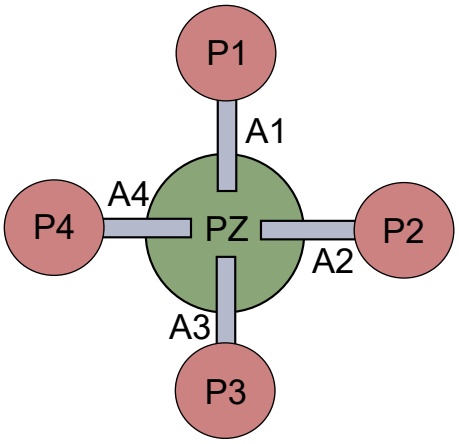
```
Bitte geben Sie die Breite der Baumkrone ein: 15
Bitte geben Sie die Breite des Baumstamms ein: 5
```

```
Bitte geben Sie die Höhe des Baumstamms ein: 4

       i
      i*i
     i***i
    i*****i
   i*******i
  i*********i
 i***********i
i*************i
     *****
     *****
     *****
     *****
```

Aufgabe 8: Karussell

Ein Karussell bestehe aus einer zentralen Plattform PZ, an der vier Auslegerarme (A1...A4) montiert sind. An jedem Auslegerarm ist wiederum eine Plattform (P1...P4) montiert.

Zu Beginn befinden sich die Arme 1 und 3 in hochgefahrener Stellung, die Arme 2 und 4 hingegen in der heruntergefahrenen Stellung. Nach jeweils zwei Drehungen der zentralen Plattform PZ wechseln sie ihre Stellung, d. h. die Arme 1 und 3 fahren herunter, die Arme 2 und 4 hingegen hinauf und umgekehrt.



Draufsicht aufs Karussel

Im Steuerhaus des Karussels kann die bedienende Person die Länge einer Karusselfahrt über die Anzahl der durchzuführenden Drehungen von PZ vorgeben.

Zusätzlich lassen sich die Anzahl der Drehungen D1 der Plattformen P1 und P3 sowie D2 der Plattformen P2 und P4 einstellen, die während einer Drehung von PZ durchzuführen sind.

Auf diese Weise lässt sich jede Karusselfahrt recht flexibel gestalten.

Schreiben Sie ein Programm, welches die genannten Parameter zu Beginn einliest und eine entsprechende Karusselfahrt mit Hilfe von Textausgaben in die Konsole simuliert.

Beispielablauf :

```
Karusselldrehungen: 9
```

```
D1: 2
D2: 3

Karusselldrehung 1:
A1A3 Drehung 1 oben, A2A4 Drehung 1 unten,
A1A3 Drehung 2 oben, A2A4 Drehung 2 unten,
A2A4 Drehung 3 unten,
Karusselldrehung 2:
A1A3 Drehung 1 oben, A2A4 Drehung 1 unten,
A1A3 Drehung 2 oben, A2A4 Drehung 2 unten,
A2A4 Drehung 3 unten,
Karusselldrehung 3:
A1A3 Drehung 1 unten, A2A4 Drehung 1 oben,
A1A3 Drehung 2 unten, A2A4 Drehung 2 oben,
A2A4 Drehung 3 oben,
Karusselldrehung 4:
A1A3 Drehung 1 unten, A2A4 Drehung 1 oben,
A1A3 Drehung 2 unten, A2A4 Drehung 2 oben,
A2A4 Drehung 3 oben,
Karusselldrehung 5:
A1A3 Drehung 1 oben, A2A4 Drehung 1 unten,
A1A3 Drehung 2 oben, A2A4 Drehung 2 unten,
A2A4 Drehung 3 unten,
Karusselldrehung 6:
A1A3 Drehung 1 oben, A2A4 Drehung 1 unten,
A1A3 Drehung 2 oben, A2A4 Drehung 2 unten,
A2A4 Drehung 3 unten,
Karusselldrehung 7:
A1A3 Drehung 1 unten, A2A4 Drehung 1 oben,
A1A3 Drehung 2 unten, A2A4 Drehung 2 oben,
A2A4 Drehung 3 oben,
Karusselldrehung 8:
A1A3 Drehung 1 unten, A2A4 Drehung 1 oben,
A1A3 Drehung 2 unten, A2A4 Drehung 2 oben,
A2A4 Drehung 3 oben,
Karusselldrehung 9:
A1A3 Drehung 1 oben, A2A4 Drehung 1 unten,
A1A3 Drehung 2 oben, A2A4 Drehung 2 unten,
A2A4 Drehung 3 unten,
```

Übungsblatt zu überwachungsgesteuerten Schleifen

Aufgabe 1: Zufallszahl aus einem Bereich ziehen

Implementieren Sie das Beispiel für eine überwachungsgesteuerte Schleife von der Vorlesungsfolie "Zufallszahl aus einem Bereich ziehen".

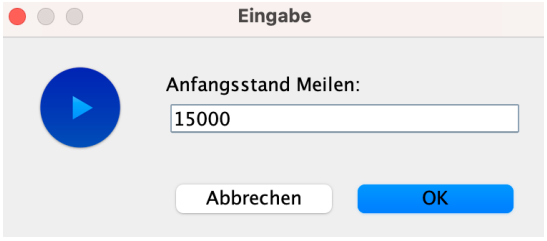
Rufen Sie das Programm ein paar Mal auf und werten Sie die Resultate aus.

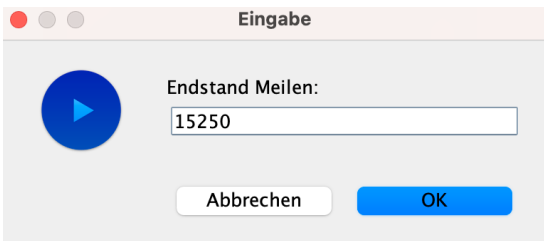
Aufgabe 2: Meilen pro Gallone

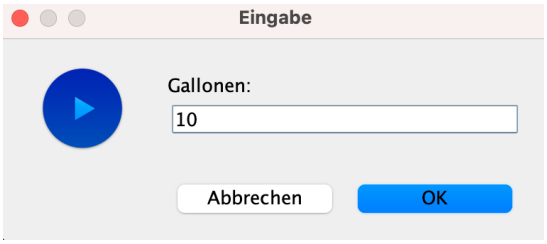
Schreiben Sie ein Programm, das die gefahrenen Meilen pro Gallone für eine Reihe von Autos berechnet. Die Daten für jedes Auto bestehen aus dem Anfangs- und Endstand in Meilen und der Anzahl der Gallonen. Das Programm wird beendet, sobald eine negative Zahl als Anfangsstand eingegeben wird.

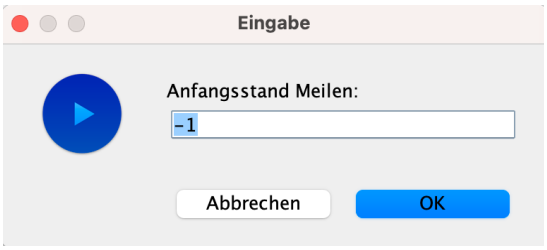
Beispielablauf:

```
Anfangsstand Meilen: 15000
Endstand Meilen: 15250
Gallonen: 10

Meilen pro Gallone: 25.0


Anfangsstand Meilen: -1

Ade.
```










```
Anfangsstand Meilen:
15000
Endstand Meilen:
15250
Gallonen:
10
Meilen pro Gallone: 25.0
Anfangsstand Meilen:
-1

Ade.
```

Konsole Fehler

Die I/O-Bibliothek haben wir bis jetzt nur zur Eingabe verwendet.
Es besteht aber auch eine Ausgabemöglichkeit (deswegen der
Name IOLib), die alternativ oder zusätzlich zur Konsolenausgabe genutzt werden kann:

```
IOLib.putString("Meilen pro Gallone: " + ..... ;
```


```
IOLib.putString("Ade.");
```

Aufgabe 3: Bereiche addieren

Schreiben Sie ein Programm, welches die Unter- und Obergrenze für einen Bereich von ganzen Zahlen einliest. Das Programm liest danach ganze Zahlen ein, die addiert werden sollen und berechnet zwei Summen:

- Die Summe der Integerzahlen, die im Bereich sind (inklusive).
- Die Summe der Integerzahlen, die außerhalb des Bereichs sind.

Das Abfragen weiterer Eingabewerte wird durch Eingabe einer 0 für die nächste zu addierende Zahl terminiert. Das Programm gibt dann die beiden Ergebnissummen aus.

Beispielablauf:

```
Untergrenze des Bereichs: 20
Obergrenze des Bereichs: 50
Ganze Zahl eingeben: 21
Ganze Zahl eingeben: 60
Ganze Zahl eingeben: 49
Ganze Zahl eingeben: 30
Ganze Zahl eingeben: 91
Ganze Zahl eingeben: 0

Summe der Werte innerhalb des Bereichs: 100
Summe der Werte außerhalb des Bereichs: 151
```

Aufgabe 4: Versandkosten-Kalkulator

Ein Paketdienst berechnet $3.00 Versandkosten bis zu einem Gewicht von 10 Pfund (inklusive). Darüber sind für jedes Pfund zusätzlich $0.25 zu bezahlen. Schreiben Sie ein Programm, dass nach dem Gewicht einer Sendung in ganzen Pfund fragt und dann die Versandkosten in Dollar ausgibt. Das Programm endet, wenn ein Gewicht von 0 oder weniger eingegeben wird.

Beispielablauf:

```
Gewicht der Sendung: 5
Versandkosten: $3.0

Gewicht der Sendung: 20
```

```
Versandkosten: $5.5

Gewicht der Sendung: 0
Ade.
```

Aufgabe 5: Fläche von Rechtecken

Ein CAD-Programm erwartet als Eingabe die Koordinaten von zwei Eckpunkten für eine Reihe von Rechtecken (siehe Abbildung 1). Dabei wird vorausgesetzt, dass die Seiten der Rechtecke parallel zur X- und Y-Achse verlaufen.

Die Koordinaten eines jeden Eckpunkts werden als Paar von ganzen Zahlen eingegeben, zuerst die X-Koordinate und dann die Y-Koordinate. Der Ursprung des Koordinatensystems (0,0) ist die linke obere Ecke, so dass Y abwärts zunimmt und X nach rechts zunimmt.

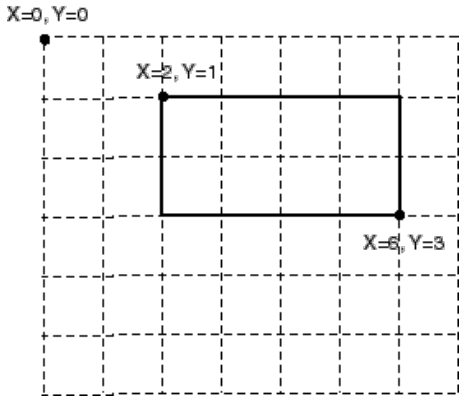


Abb.1: Beispiel CAD-Oberfläche

Das Programm berechnet für jedes Rechteck die Höhe, die Breite und die Fläche. Die zwei eingegebenen Eckpunkte müssen sich diagonal gegenüber liegen (links oben und rechts unten, oder rechts oben und links unten). Welche Wahl für jedes Rechteck getroffen wird, ist Sache der das Programm nutzenden Person. Die Person kann die Eckpunkte in beliebiger Reihenfolge eingeben. Höhe und Breite sind immer positive Integerzahlen.

Das Programm wird beendet, wenn die Person Eckpunkte eingibt, die kein Rechteck bilden können, da die Höhe 0 ist und/oder die Breite 0 ist.

Beispielablauf:

```
Erster Eckpunkt X-Koordinate: 100
Erster Eckpunkt Y-Koordinate: 100
Zweiter Eckpunkt X-Koordinate: 250
Zweiter Eckpunkt Y-Koordinate: 200

Breite: 150, Höhe: 100, Fläche: 15000

Erster Eckpunkt X-Koordinate: 50
Erster Eckpunkt Y-Koordinate: 100
Zweiter Eckpunkt X-Koordinate: 50
Zweiter Eckpunkt Y-Koordinate: 200

Ade.
```

Aufgabe 6: Grafikobjekt selektieren

In einem CAD-Programm soll geprüft werden, ob die das Programm nutzende Person das in der Abbildung 2 angegebene, grau getönte Pfeilobjekt selektieren möchte. Das Objekt besteht aus

einzelnen Bildpunkten bzw. Pixels (Abkürzung für *Picture Elements*) und gilt als selektiert, wenn die Koordinaten eines zum Objekt gehörenden Pixels eingegeben werden. Pixel werden durch die Koordinate ihrer linken oberen Ecke adressiert.

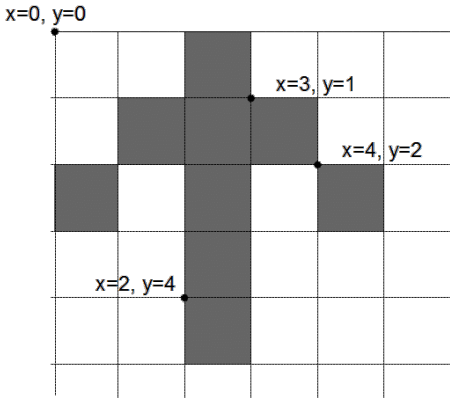


Abb.2: Pfeilobjekt

Das Programm wird beendet, wenn mindestens eine negative Koordinate eingegeben wird.

Beispielablauf:

```
X-Koordinate: 2
Y-Koordinate: 2

Objekt selektiert

X-Koordinate: 3
Y-Koordinate: 1

Objekt selektiert

X-Koordinate: 4
Y-Koordinate: 4

Objekt deselektiert

X-Koordinate: -1

Ade.
```

Übungsblatt zu ergebnisgesteuerten Schleifen

Aufgabe 1: Ratenzahlung

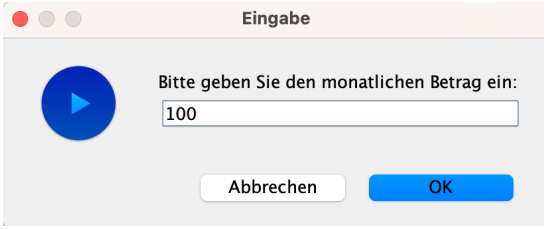
Angenommen, Sie schulden Ihrer Kreditkartengesellschaft €1000.00. Die Gesellschaft berechnet Ihnen monatlich 1.5% Zinsen auf den gewährten Kredit. Sie entscheiden sich, die Kreditkarte nicht mehr zu verwenden und Ihren Kredit monatlich mit einem bestimmten Betrag `n` zurückzuzahlen.

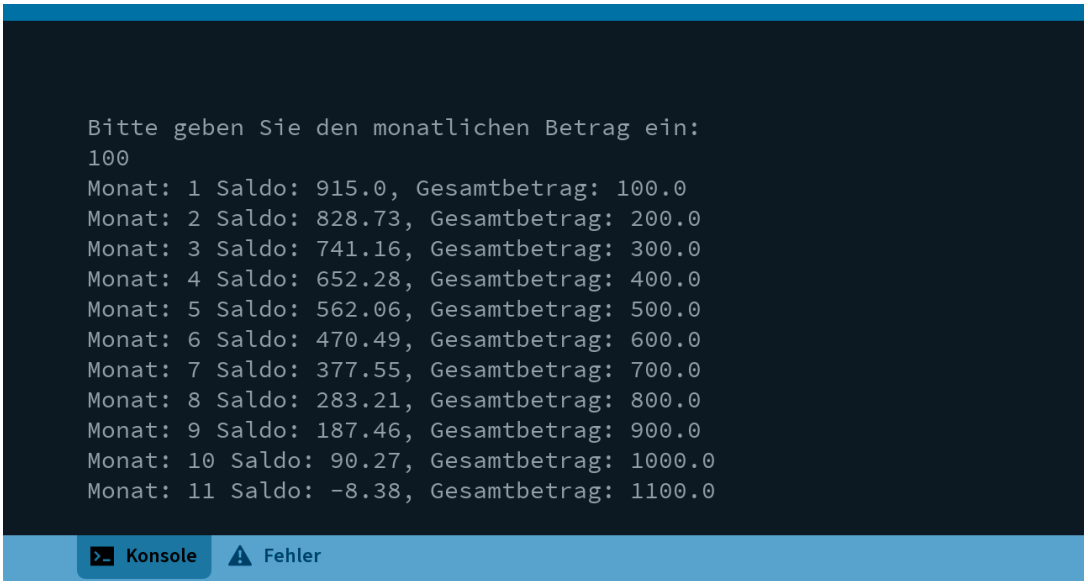
Schreiben Sie ein Programm, das nach dem monatlichen Rückzahlungsbetrag in ganzen Euros fragt, dann den Saldo und den aktuellen Gesamtbetrag der Zahlungen für jeden Monat ausgibt, bis der Saldo Null oder weniger beträgt.

Berechnen Sie die monatlichen Zinsen auf Grund des Saldos. Berechnen Sie dann den neuen Saldo, in dem Sie die Zinsen addieren und den zurückgezahlten Betrag subtrahieren.

Beispielablauf:

```
Bitte geben Sie den monatlichen Betrag ein: 100

Monat: 1, Saldo: 915.0, Gesamtbetrag: 100.0
Monat: 2, Saldo: 828.73, Gesamtbetrag: 200.0
Monat: 3, Saldo: 741.16, Gesamtbetrag: 300.0
Monat: 4, Saldo: 652.27, Gesamtbetrag: 400.0
Monat: 5, Saldo: 562.06, Gesamtbetrag: 500.0
Monat: 6, Saldo: 470.49, Gesamtbetrag: 600.0
Monat: 7, Saldo: 377.55, Gesamtbetrag: 700.0
Monat: 8, Saldo: 283.21, Gesamtbetrag: 800.0
Monat: 9, Saldo: 187.46, Gesamtbetrag: 900.0
Monat: 10, Saldo: 90.27, Gesamtbetrag: 1000.0
Monat: 11, Saldo: -8.38, Gesamtbetrag: 1100.0
```





Verwenden Sie für die Berechnungen der Einfachheit halber **Gleitkommaarithmetik** mit `float`.

Aufgabe 2: Wirksamkeit von Medikamenten

Jeden Monat, den ein bestimmtes Medikament gelagert wird, verliere es 4% seiner Wirksamkeit. Wenn seine Wirksamkeit unter 50% fällt, wird es als unbrauchbar eingestuft und sollte entsorgt werden. Schreiben Sie ein Programm, welches für jeden Monat den Verlust an Wirksamkeit ausgibt und beim Unterschreiten von 50% Wirksamkeit, dass das Medikament entsorgt werden sollte.

Programmausgabe:

```
Monat: 0, Effektivität: 100.0%
Monat: 1, Effektivität: 96.0%
Monat: 2, Effektivität: 92.16%
Monat: 3, Effektivität: 88.47%
Monat: 4, Effektivität: 84.93%
Monat: 5, Effektivität: 81.54%
Monat: 6, Effektivität: 78.28%
Monat: 7, Effektivität: 75.14%
Monat: 8, Effektivität: 72.14%
Monat: 9, Effektivität: 69.25%
Monat: 10, Effektivität: 66.48%
Monat: 11, Effektivität: 63.82%
Monat: 12, Effektivität: 61.27%
Monat: 13, Effektivität: 58.82%
Monat: 14, Effektivität: 56.47%
Monat: 15, Effektivität: 54.21%
Monat: 16, Effektivität: 52.04%
Monat: 17, Effektivität: 49.96%
Entsorgen!
```

Aufgabe 3: e^x

Der Grund, warum Rechner den Wert von Funktionen wie e^x berechnen können, liegt darin begründet, dass für diese Funktionen Reihenentwicklungen existieren.

Die Exponentialfunktion e^x besitzt z. B. die folgende Potenzreihenentwicklung:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} + ...$$

Dabei steht $n!$ für die Fakultät von n, d. h.:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 1$$

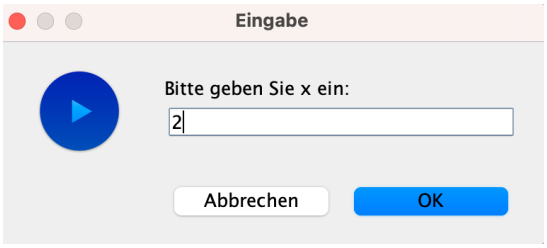
Schreiben Sie ein Programm, welches einen Wert für x eingeben lässt und ebenfalls die gewünschte Genauigkeit y der Berechnung, in Stellen hinter dem Komma, und dann e^x mit Hilfe einer Schleife berechnet, solange der in jeder Iteration zur Summe addierte Term größer ist als 10^{-y} . Vergleichen Sie schließlich den von Ihrem Programm berechneten Wert mit dem von der offiziellen Mathematikmethode `exp(x)` berechneten Wert.

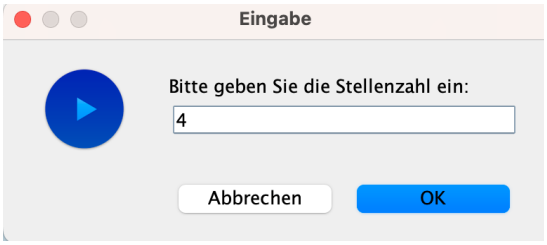
Zwischen dem n-ten und $(n-1)$-ten Term besteht die folgende Beziehung:

$$\frac{x^n}{n!} = \frac{x^{n-1}}{(n-1)!} \cdot \frac{x}{n}$$

Mit Hilfe dieser Beziehung können viele unnötige Berechnungen vermieden werden, sodass die Lösung dieser Aufgabe am Ende überraschend einfach ist.

Beispielablauf:





```
Bitte geben Sie x ein:
2
Bitte geben Sie die Stellenzahl ein:
4
n: 1, term: 2.0, sum: 3.0
n: 2, term: 2.0, sum: 5.0
n: 3, term: 1.3333334, sum: 6.3333335
n: 4, term: 0.6666667, sum: 7.0
n: 5, term: 0.26666668, sum: 7.266667
n: 6, term: 0.08888889, sum: 7.355556
n: 7, term: 0.025396826, sum: 7.380953
n: 8, term: 0.0063492064, sum: 7.387302
n: 9, term: 0.0014109347, sum: 7.388713
n: 10, term: 2.8218696E-4, sum: 7.388995
n: 11, term: 5.1306717E-5, sum: 7.3890467

Eigenes e^x:     7.3890467
exp(x):          7.389056
Iterationen:     11
```

Konsole Fehler

Aufgabe 4: sin(x)

Nicht nur für die Funktion e^x existiert eine Reihenentwicklung. Auch für die trigonometrische Funktion sin(x), die schon Teil der Übungsaufgaben war, existiert eine Potenzreihe:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ... + (-1)^n \frac{x^{2n+1}}{(2n+1)!} \pm ...$$

Schreiben Sie ein Programm für sin(x) entsprechend der Lösung zur vorigen Aufgabe.

Beispielablauf:

```
Bitte geben Sie x ein:
0.7
Bitte geben Sie die Stellenzahl ein:
4
Eigenes sin(x):          0.6442176
Processing sin(x):       0.64421767
Iterationen:             3
```

Konsole Fehler

Wie Sie sehen, besitzt diese Reihe eine weitaus höhere Konvergenzgeschwindigkeit, als die Reihe der vorigen Aufgabe, was sich in der deutlich geringeren Anzahl an Iterationen äußert, die zur Erreichung der geforderten Genauigkeit erforderlich sind.

Übungsaufgaben zum Thema Felder

Kopieren Sie jeweils den unten angegebenen Code in Ihren Editor und vervollständigen Sie diesen wie in den Aufgaben vorgegeben. Kompilieren und starten Sie danach das jeweilige Programm.

Aufgabe 1: Feldelementsumme

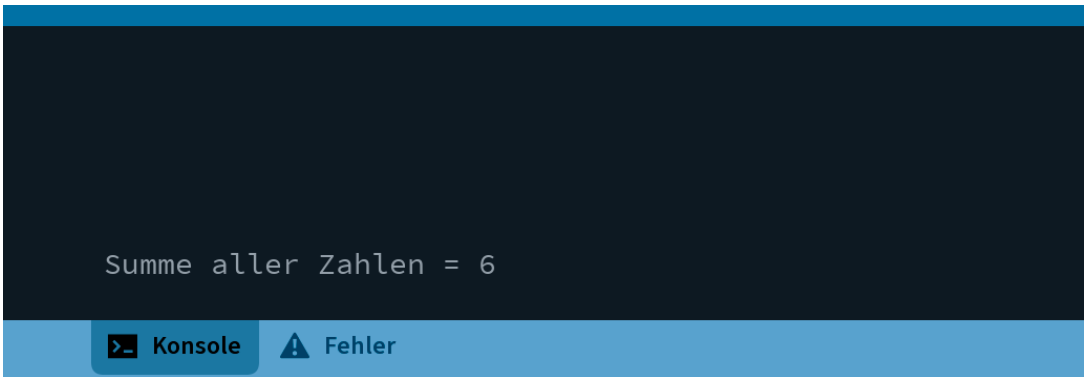
Betrachten Sie das folgende Codesegment:

```
int[] val = {0, 1, 2, 3};
int sum = 0;
println("Summe aller Zahlen = " + sum);
```

Vervollständigen Sie den Code, sodass die Summe der Zahlen im Array mit Hilfe einer Schleife berechnet wird.

Ausgabe des Programms:

```
Summe aller Zahlen = 6
```



Aufgabe 2: Zwei Felder

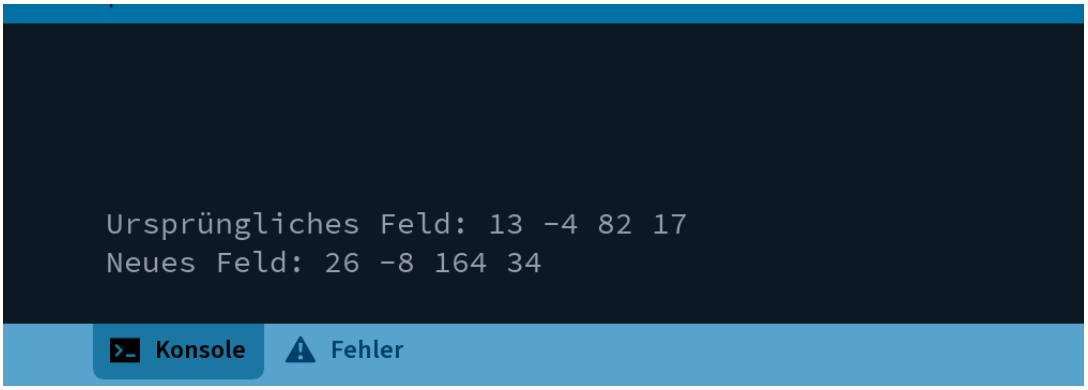
Gegeben sei ein Feld `val` mit vier Integerzahlen:

```
int val[] = {13, -4, 82, 17};
```

- Geben Sie die Feldelemente von `val` mit Hilfe einer Schleife aus
- Definieren Sie ein zweites Array mit Namen `twice`, welches die gleiche Anzahl an Feldelementen besitzt, wie `val`.
- Schreiben Sie mit Hilfe einer weiteren Schleife Werte in `twice`, die jeweils doppelt so groß sind wie die korrespondierenden Werte in `val`
- Geben Sie die Feldelemente von `twice` mit Hilfe einer zusätzlichen Schleife aus.

Programmausgabe:

```
Ursprüngliches Feld: 13 -4 82 17
Neues Feld: 26 -8 164 34
```



Aufgabe 3: Drei Felder

Gegeben seien drei Felder:

```
int valA[] = {13, -22,  82,  17};
int valB[] = {-12,  24, -79, -13};
int sum[]  = {  0,   0,   0,   0};
```

- Addieren Sie mit Hilfe einer Schleife die Werte der Feldelemente von `valA` und `valB`, die den gleichen Index besitzen, und schreiben Sie das Ergebnis in das korrespondierende Feldelement von `sum`.
- Geben Sie die Feldelemente von `sum` mit Hilfe einer weiteren Schleife aus.

Programmausgabe:

```
Summe: 1 2 3 4
```

Aufgabe 4: Dieselbe Summe

Gegeben sei das folgende Feld:

```
int valA[] = {13, -22,  82,  17};
```

Definieren Sie ein neues Feld `valB` und beschreiben Sie dessen Feldelemente in einer Schleife mit Werten, so dass die Summe der korrespondierenden Feldelemente von `valA` und `valB` 25 ergibt.

Programmausgabe:

```
valA:   [13] [-22] [82] [17]
valB:   [12] [47] [-57] [8]
Sum:    [25] [25] [25] [25]
```

Aufgabe 5: Mittelwert und unterschiedliche Summen der Feldelemente

Gegeben sei das folgende Feld:

```
int data[] = {3, 2, 5, 7, 9, 12, 97, 24, 54};
```

Schreiben Sie Programmcode, in dem mit Hilfe einer einzelnen Schleife folgendes berechnet wird:

- der Mittelwert (auch: Durchschnittswert) der Feldelemente als Integerwert,
- die Summe der Feldelemente, die einen geraden Wert (nicht Indexwert) besitzen,
- die Summe der Feldelemente mit ungeradem Wert.

Hinweis: Eine gerade Zahl lässt sich ohne Rest ganzzahlig durch zwei teilen.

Programmausgabe:

```
Mittelwert: 23, Summe gerader: 92, Summe ungerader: 121
```

Aufgabe 6: Die zwei größten Elemente

Gegeben sei das folgende Feld:

```
int data[] = {3, 1, 5, 7, 4, 12, -3, 8, -2};
```

Schreiben Sie ein Programm, dass mit Hilfe einer einzelnen Schleife, d. h. einer einzelnen Maximumsuche, die zwei größten Elemente des Felds findet und ausgibt.

Programmausgabe:

```
Größtes Element:        12
Zweitgrößtes Element:   8
```

Prüfen Sie Ihr Programm mit unterschiedlichen Feldern, wie z. B.:

- `{4, 1, 2, 3}`
- `{4, 5, 1, 2}`

Bei dieser Aufgabe steckt die größte Schwierigkeit in einer geeigneten Initialisierung der Variablen.

Aufgabe 7: Tonfilter

Ein Tonsignal wird manchmal als Liste von `int`-Werten gespeichert. Die Werte repräsentieren die Intensität des Signals in aufeinander folgenden Zeitintervallen.

```
int signal[] = {1, 5, 4, 5, 7, 6, 8, 6, 5, 4, 5, 4};
```

Oft ist in dem Signal ein kleiner Anteil von Störgeräuschen enthalten. Störgeräusche sind üblicherweise kleine, momentane Änderungen der Tonhöhe. Ein Beispiel hierfür ist das Rauschen, das zusätzlich zum Ton eines AM Radios zu hören ist.

Das Glätten des Tons unterdrückt das Störgeräusch und verbessert die Tonqualität.

In dieser Aufgabe werden Sie die Werte eines Integerarrays glätten. Angenommen, dass die ursprünglichen Werte in dem Array ein "Signal" repräsentieren. Berechnen Sie zu diesem Signal geglättete Feldelemente, indem Sie folgendes tun: Definieren Sie ein neues Feld `smooth`, mit der gleichen Anzahl an Feldelementen wie `signal`. Mit Ausnahme des ersten und letzten Feldelements in `smooth`, ist jeder Wert `smooth[N]` der Durchschnitt von drei Werten: `signal[N-1]`, `signal[N]` und `signal[N+1]`.

Berechnen Sie für das erste Element von `smooth` den Durchschnitt der ersten zwei Elemente von `signal`. Berechnen Sie für das letzte Element von `smooth` den Durchschnitt der letzten zwei Elemente von `signal`.

Verwenden Sie dafür Integerarithmetik, so dass die Werte in `smooth` Integer sind.

Beziehen Sie sich in Ihrem Code auf die Länge des Felds `signal`. Verwenden Sie keine festen Zahlenwerte für die Feldlänge, damit der Code unabhängig von der Länge des real bearbeiteten Felds ist.

Programmausgabe für das Feld oben:

```
Original signal: 1 5 4 5 7 6 8 6 5 4 5 4
Smoothed signal: 3 3 4 5 6 7 6 6 5 4 4 4
```

Denken Sie daran, dass bei der Integerdivision der Rest verworfen und kein gerundeter Wert berechnet wird.

Aufgabe 8: Datenoptimierung

Angenommen, Sie möchten den durchschnittlichen Säuregehalt von Kaffee bestimmen, der in den Cafés Ihrer Heimatstadt serviert wird. Zu diesem Zweck besuchen Sie eine Reihe von Cafés und tauchen Ihr PH-Messgerät in Kaffeeproben.

Leider liefert Ihr PH-Messgerät manchmal falsche Ergebnisse. Sie schließen daher den am weitesten vom Durchschnitt entfernten Messwert als Fehlmessung von der Durchschnittsberechnung aus.

Berechnen Sie den Durchschnitt aller Daten im unten angegebenen Feld `samples`. Durchsuchen Sie danach das Feld, um den Wert zu finden, der am weitesten (in jeder Richtung) vom Durchschnitt entfernt ist. Korrigieren Sie danach den Durchschnittswert, indem Sie diese Fehlmessung aus der Summenberechnung für den Durchschnittswert ausschließen. Berechnen Sie danach den korrigierten Durchschnittswert und geben Sie ihn aus.

```
double samples[] = {5.6, 6.2, 6.0, 5.5, 5.7,
                    6.1, 7.4, 5.5, 5.5, 6.3,
                    6.4, 4.0, 6.9};
```

Programmausgabe:

```
Durchschnitt:           5.930769
Entferntester Wert:     4.0
Neuer Durchschnitt:     6.091667
```

Definieren Sie für die Anzahl der Feldelemente wieder eine Konstante, damit Ihr Code unabhängig von der Länge des real bearbeiteten Felds ist.

Aufgabe 9: Histogramm

Als Histogramm wird die grafische Darstellung der Häufigkeitsverteilung einer Menge von Werten bezeichnet. Dies kann z. B. die Häufigkeit sein, mit der Bitfehler der Größe 0 Bit, 1 Bit, 2 Bit usw. bei einem bestimmten Übertragungskanal vorkommen oder die Anzahl von Pixel bzw. Bildelementen mit einem bestimmten Grauwert in einem Bild oder die Anzahl von Studierenden eines bestimmten Alters in einem Vorlesungsraum.

Gegeben sei das folgende Array,

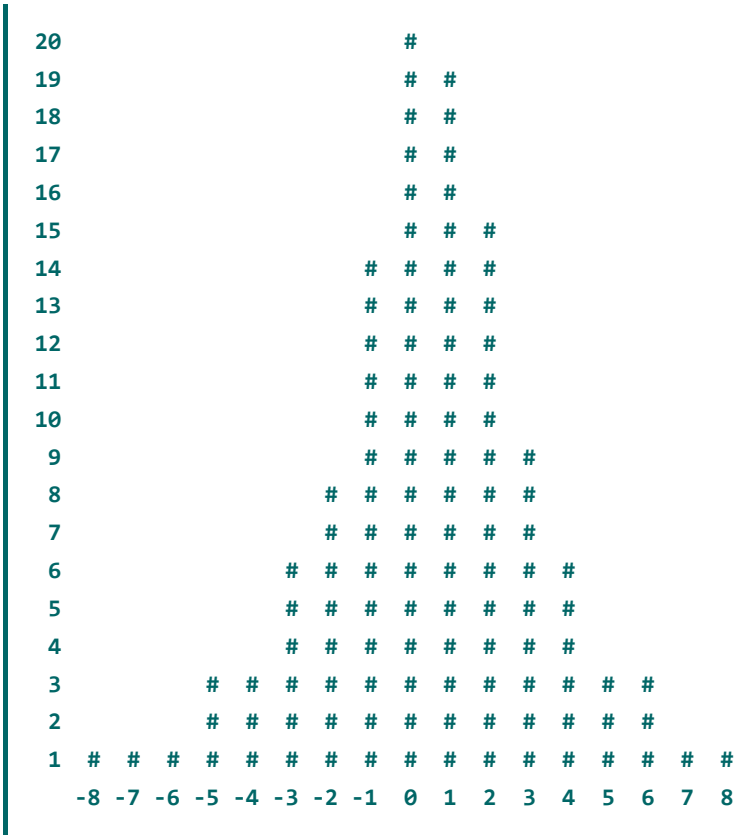
```
int samples[] = {1, 1, 1, 3, 3, 6, 8, 14, 20,
                 19, 15, 9, 6, 3, 3, 1, 1};
```

welches für die gezählte Häufigkeit bestimmter Abweichungen von einem erwarteten Wert steht. Der erwartete Wert sei hier 0. Der Wertebereich der Werte in `samples` reicht dabei von -8 bis 8, d. h.:

```
 1   1   1   3   3   6   8  14 20 19 15  9   6   3   3   1   1

-8  -7  -6  -5  -4  -3  -2  -1  0   1   2   3   4   5   6   7   8
```

Der Wert 0 kommt dabei z. B. mit 20 Zählungen am häufigsten vor. Der Wert 1 mit 19 Zählungen fast genauso oft. Es könnte sich bei diesen Werten z. B. um die Häufigkeit von Mehrbitfehlern auf einem bestimmten Übertragungskanal handeln, wobei Wechsel von 1 nach 0 z. B. negative Vorzeichen besitzen und Wechsel von 0 nach 1 positive Vorzeichen. Die Güte dieses Übertragungskanals wäre dann nicht besonders hoch, da Mehrbitfehler recht häufig auftreten. 0 würde einer fehlerfreien Übertragung entsprechen. ±1 einem einzelnen Bitfehler.

Schreiben Sie ein Programm, welches mit Hilfe einer geschachtelten `for`-Schleife das oben gegebene Feld in der folgenden Weise grafisch im Ausgabefenster darstellt:



Bestimmen Sie die Anzahl der erforderlichen Zeilen mit Hilfe einer Maximumsuche. Geben Sie die Beschriftung [-8;8] auf der x-Achse mit Hilfe einer weiteren `for`-Schleife aus.

Übungen zum Thema Sortieren und zum Debugging

Aufgabe 1: Elemente eines Felds umkehren

Gegeben sei folgendes Feld:

```
int data[] = {0, 1, 2, 3, 4};
```

Kehren Sie die Reihenfolge der Zahlen im Feld `data` um

1. Tun Sie dies im ersten Schritt mit einzelnen Zuweisungen
2. Entwickeln Sie diese Lösung im zweiten Schritt zu einer verbesserten Lösung weiter, die eine Schleife benutzt
3. Geben Sie das umgekehrte Feld jeweils mit einer Schleife aus

Benutzen Sie für das Umkehren kein zweites Feld, sondern vertauschen Sie die Feldelemente im Feld `data` selbst, durch Einsatz einer Hilfsvariable `temp`.

Programmausgabe:

Aufgabe 2: Elemente umgekehrt in ein zweites Feld schreiben

Gegeben sei folgendes Feld:

```
int data[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14};
```

Schreiben Sie Programmcode, der die Reihenfolge der Elemente in `data` umkehrt. Verwenden Sie hierfür zwei Felder und keine Hilfsvariable, wie bei der Lösung der Aufgabe oben.

Das erste Feld `data` wird nicht geändert. Das zweite Feld `result` erhält die Elemente von `data` in umgekehrter Reihenfolge eingeschrieben.

Geben Sie am Ende die Elemente des Felds `result` mit Hilfe einer Schleife aus.

Programmausgabe:

> [14] [13] [12] [11] [10] [9] [8] [7] [6] [5] [4] [3] [2] [1]

Aufgabe 3: Maximumsuche

In unserer Entwicklungsumgebung existiert ein wichtiges Werkzeug, welches die Fehlersuche in Programmen immens vereinfachen kann: Der **Debugger**.

Der Debugger gehört zu den grundlegenden Werkzeugen einer integrierten Entwicklungsumgebung.

In dieser Übung werden wir die Funktion des Debuggers in Processing näher kennenlernen.

Implementieren Sie zunächst unter Processing den in der Vorlesung behandelten Algorithmus zur Maximumsuche mit Feld (Seite 266), z. B. als neue Sketch, und führen Sie ihn danach aus.

Als Programmausgabe sollte folgendes erscheinen:

Diesen Algorithmus werden wir nun mit Hilfe des Debuggers untersuchen.

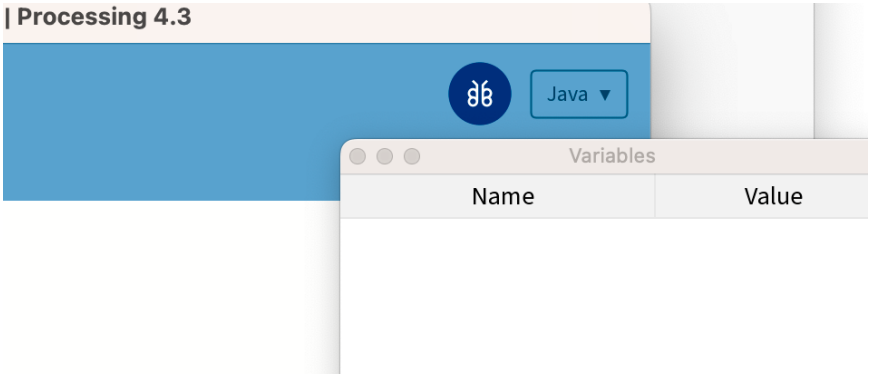
Debugging

Der Debugger gestattet es, ein Programm schrittweise auszuführen, um seine Funktion zu analysieren und dabei Variablenwerte einzusehen und diese während der Laufzeit des Programms ggf. auch zu ändern. Auf diese Weise lassen sich schwer zu findende Programmfehler leichter lokalisieren, als mit einfachen Textausgaben auf die Konsole.

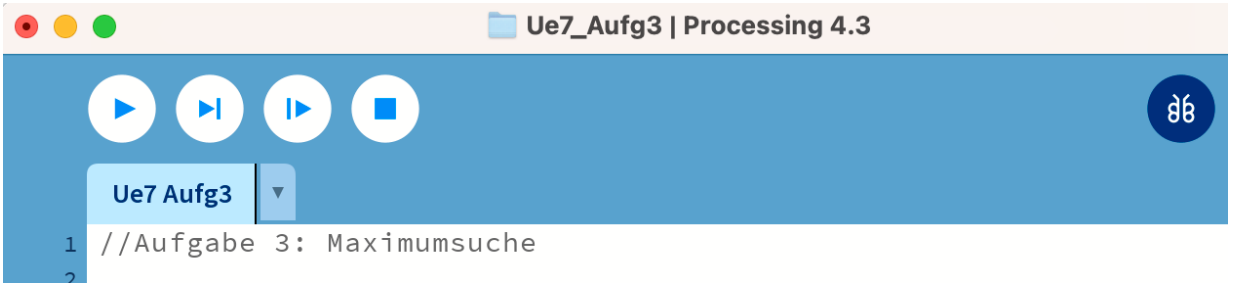
Hierfür benötigt der Debugger Informationen darüber, wie die in der Quelldatei definierten Symbole, d. h. hier zur Zeit in erster Linie die Variablen, in Beziehung stehen zum erzeugten Maschinencode. Diese Informationen werden bei der Übersetzung des Quellcodes von Processing automatisch erzeugt.

Aktivieren Sie den Debugger in Processing durch einen Klick auf folgendes Icon:

Der Debugger ist nun aktiv geschaltet (blaues Icon) und das Überwachungsfenster für die Variablen öffnet sich zusätzlich. Sollte diese Fenster verloren gehen, einfach den Debugger deaktivieren und erneut aktivieren.



Links sind neue Buttons dazugekommen. Neben den bekannten "Starten" und "Stoppen", sehen wir jetzt auch noch "Schritt" und "Weiter". Anstatt "Starten", heisst der Button jetzt "Debug".



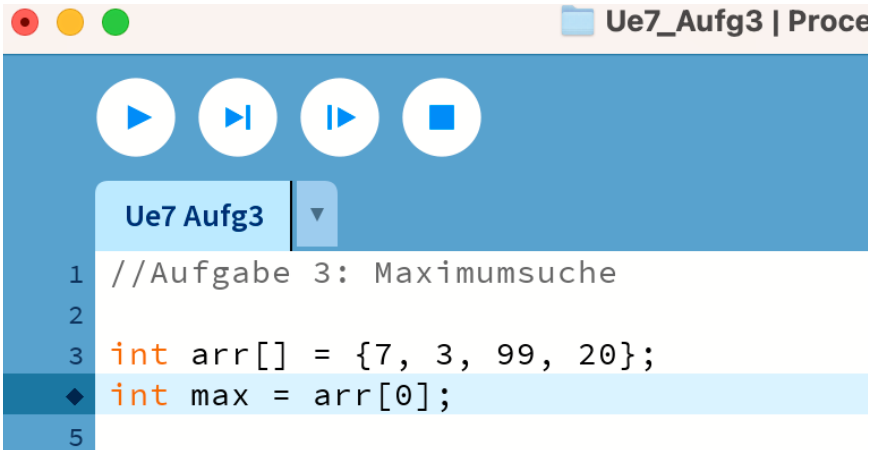
Klicken Sie zuerst auf den Button "Debug" ... Das Programm startet nun - wie gewünscht - im Debugger und produziert die folgende Ausgabe:

Programmausführungen unter dem Debugger unterscheiden sich also zunächst nicht von normalen Programmausführungen. Um den Programmcode untersuchen zu können, muss zuvor mindestens ein Unterbrechungs- bzw. Haltepunkt (engl.: *breakpoint*) definiert werden.

1. Definieren Sie nun einen Haltepunkt in der Befehlszeile

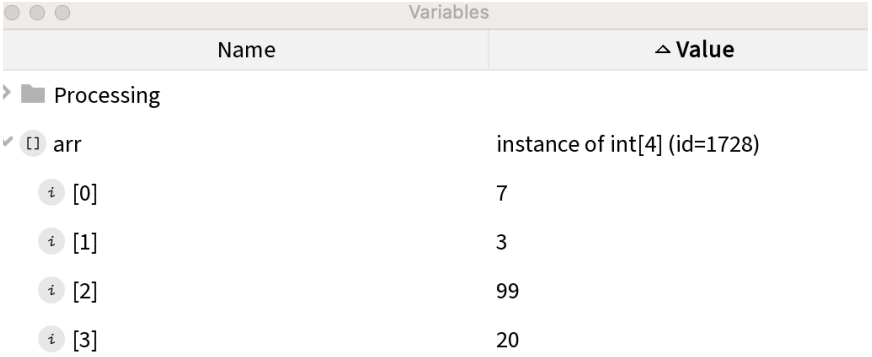
   ```
   int max = arr[0];
   ```

   indem Sie auf die Zeilennummer klicken.

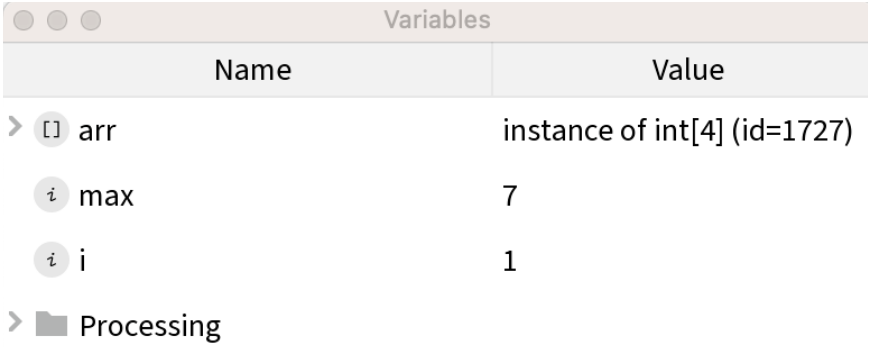


Es erscheint eine kleine blaue Raute, die für den eingefügten Haltepunkt steht. Durch einen weiteren Klick auf die blaue Raute kann man den Haltepunkt wieder entfernen. Lassen Sie diesen aber stehen und starten Sie das Programm im Debugger. Nun hält der Programmlauf am angegebenen Haltepunkt an, was durch einen kleinen Pfeil im Haltepunkt visualisiert wird. Zusätzlich ist die Programmzeile blau markiert, an der angehalten wurde.

Sobald das Programm am Haltepunkt wartet, sehen Sie im Überwachungsfenster für die Variablen, die Werte des zugehörenden Arrays. Mit dem Debugger lassen sich also Variablenwerte kontrollieren.



2. Gehen Sie jetzt im Programm durch Drücken des *Einzelschrittbuttons* ("Schritt") eine Befehlszeile weiter voran. Im Variablenfenster wird nun automatisch der neue Wert der Variable `max` angezeigt und nach einem weiteren Drücken die neu deklarierte Variable `i` hinzugefügt.



3. Drücken Sie einige Male den Einzelschrittbutton. Bei jedem Drücken wird genau ein Programmbefehl ausgeführt. Achten Sie dabei darauf, wie sich im Variablenfenster die Werte der Variablen `i` und `max` automatisch ändern. Auf diese Weise lässt sich kontrollieren, ob der Programmablauf und die Werte der Variablen den Erwartungen entsprechen. Durch fortgesetztes Drücken des Einzelschrittbuttons lässt sich bei Bedarf das Programm vollständig

schrittweise abarbeiten.

4. Sind im Programmcode mehrere Breakpoints platziert, so kann der Code zwischen diesen Breakpoints durch Drücken des Buttons *"Weiter"* in einem Zug durchlaufen werden. Beginnend bei einem Breakpoint, läuft das Programm bis zum nächsten Breakpoint oder bis zum Ende, wenn kein weiterer Breakpoint bzw. Haltepunkt mehr existiert. Probieren Sie dies aus, indem Sie in Ihrem Programmcode zwei Breakpoints in einigem Abstand voneinander platzieren und die Befehle zwischen ihnen mit *"Weiter"* durchlaufen.

Nach dieser kurzen Einführung sind Sie jetzt mit den wichtigsten Funktionen des Debuggers vertraut.

Aufgabe 4: Maxsort und Minsort

Implementieren Sie den in der Vorlesung erläuterten `Maxsort`-Sortieralgorithmus.

Ändern Sie den `Maxsort`-Sortieralgorithmus zu einem `Minsort` ab, d. h. die Zahlen sollen jetzt aufsteigend sortiert werden.

Untersuchen Sie auch diesen Algorithmus mit dem Debugger, um dessen Ablauf schrittweise zu studieren. Der Debugger ist auch ein hervorragendes Instrument, um unbekannte Algorithmen zu analysieren und zu verstehen.

Aufgabe 5: Spielkarten mischen und sortieren

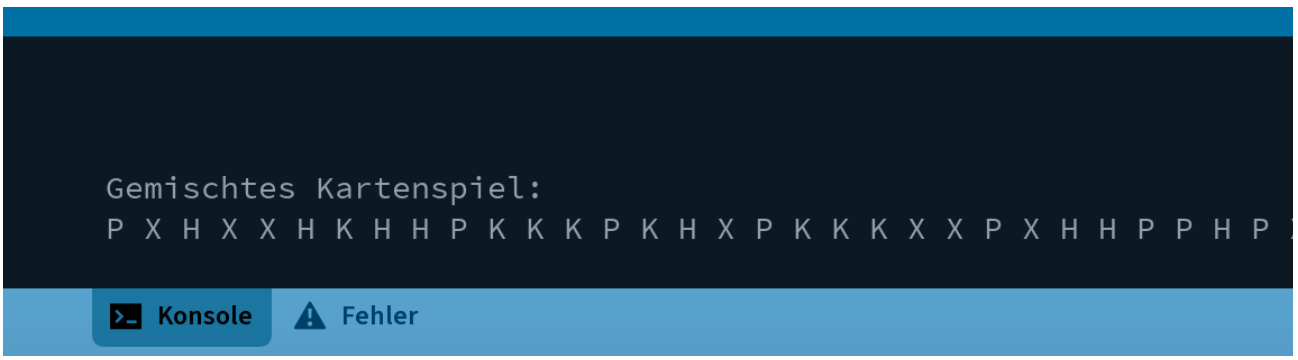
Ein Satz Spielkarten bestehe aus 32 Karten, mit jeweils 8 Herz-, Karo-, Kreuz- und Pik-Karten.

1. Schreiben Sie ein Programm, welches ein char-Feld von 32 Elementen mit 32 zufällig ausgewürfelten Karten belegt, wobei von jeder Farbe genau 8 Karten im Feld verteilt sind. Die Farben seien z. B. wie folgt codiert:

   Herz: 'H'
   Karo: 'K'
   Kreuz: 'X'
   Pik: 'P'

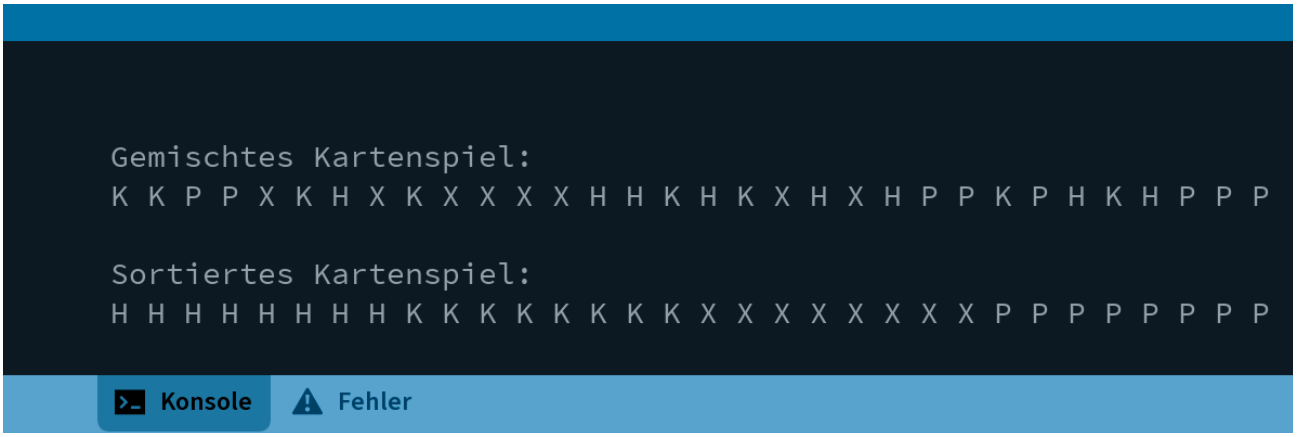
   Geben Sie das gemischte Kartenspiel auf die Konsole aus.

   Beispielprogrammausgabe:



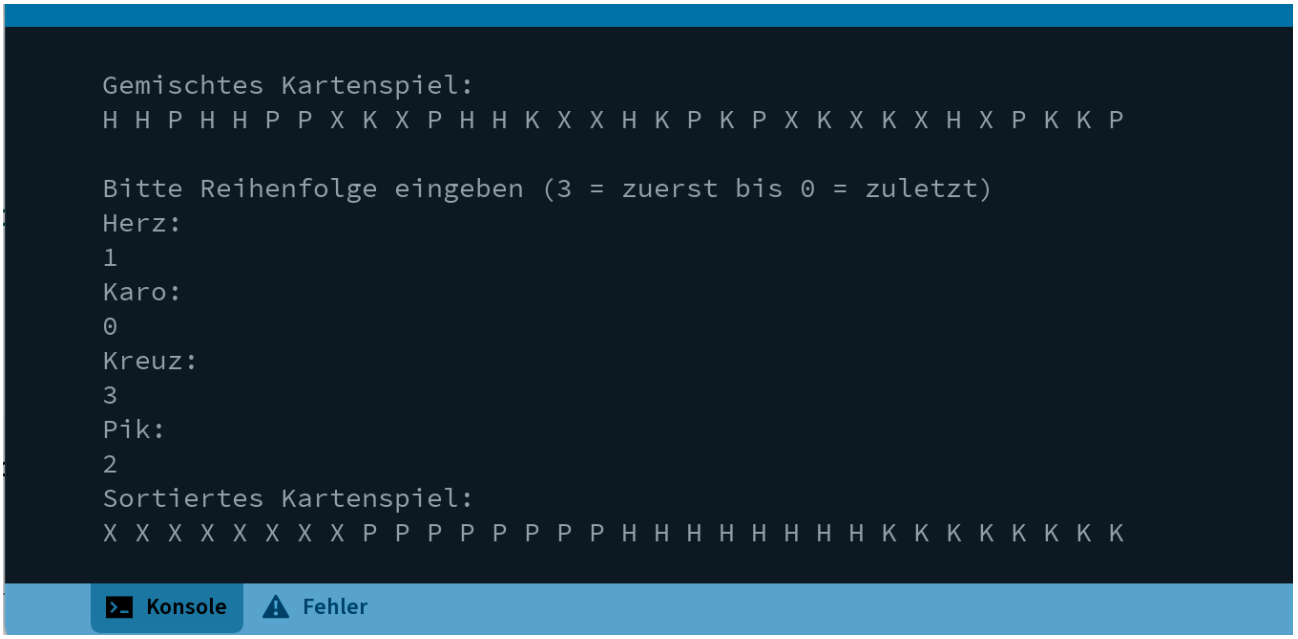
2. Erweitern Sie Ihr Programm um eine Variante des Max- bzw. Minsort-Algorithmus, die das gemischte Kartenspiel so sortiert, dass am Ende von links nach rechts zunächst die Herz-Karten im Feld liegen, dann die Karo-Karten, danach die Kreuz-Karten und schließlich die Pik-Karten. Das Sortieren geschieht auf dem ursprünglichen Feld, ohne Hilfsfelder zu diesem Zweck zu benutzen. Geben Sie zur Kontrolle das sortierte Feld aus.

   Programmausgabe:



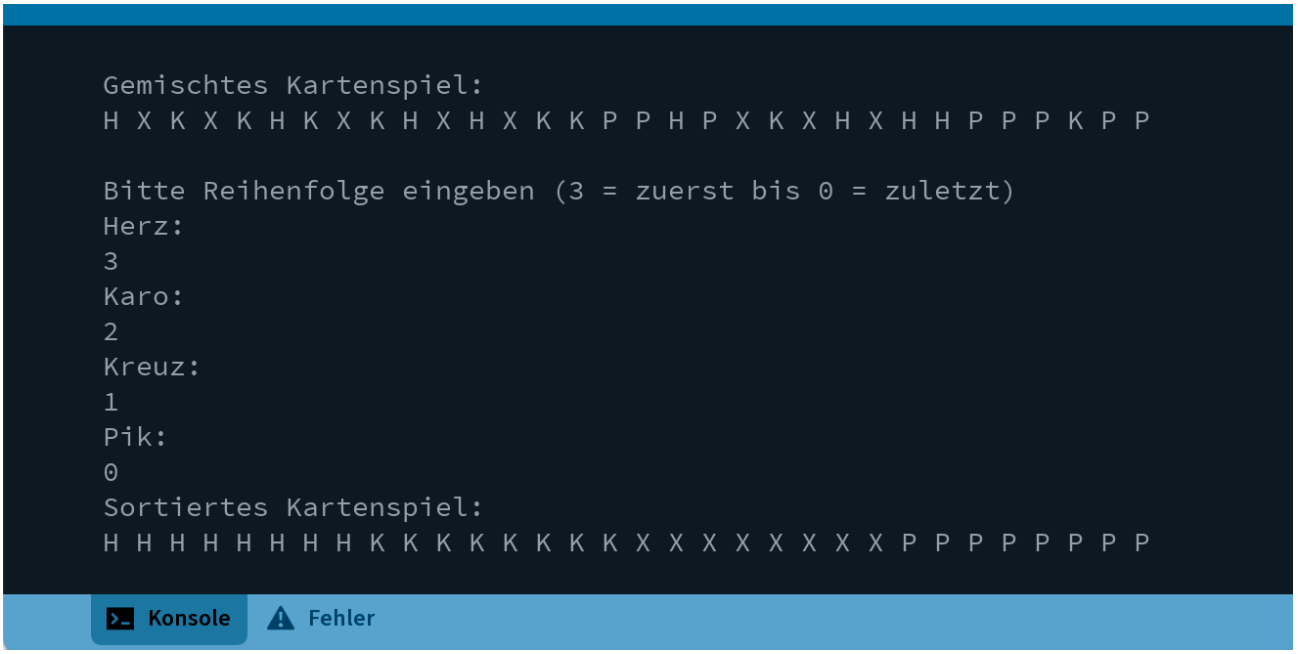
3. Erweitern Sie Ihr Programm dahingehend, dass die Reihenfolge, in der die Karten zu sortieren sind, eingegeben werden kann:

   Beispielprogrammausgabe 1:



   Kreuz zuerst, dann Pik, dann Herz und zum Schluss Karo.

   Beispielprogrammausgabe 2:



Aufgabe 6: Zeichenhäufigkeit

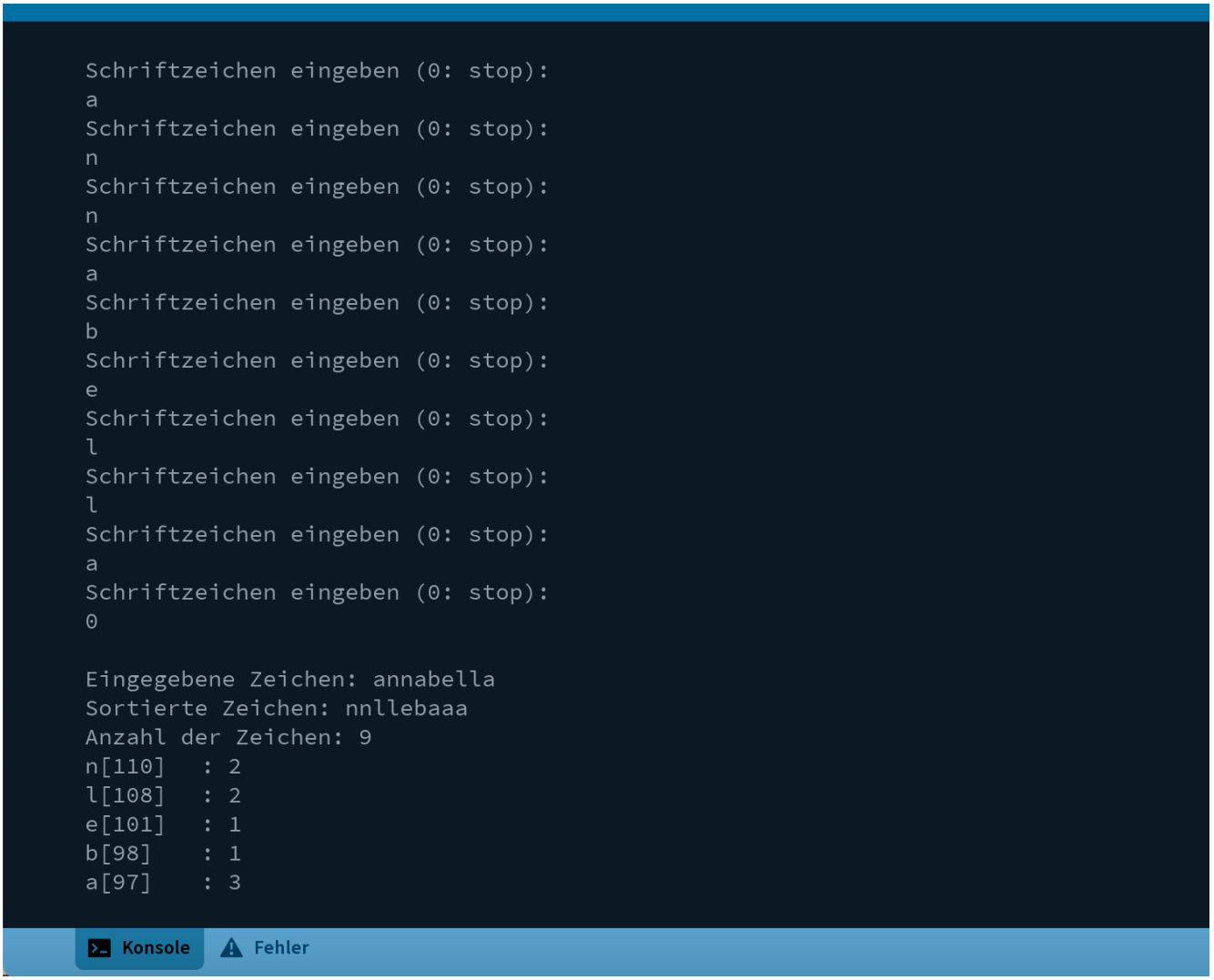
Schreiben Sie ein Programm, welches in einer Schleife über die Konsole nach einer Reihe von max. 50 Schriftzeichen fragt und diese in einem Feld vom Typ `char` abspeichert. Die Eingabe wird durch das Zeichen `0` beendet oder wenn mehr als 50 Zeichen eingegeben werden. Groß- und Kleinschreibung wird vernachlässigt. Das Feld wird danach absteigend, entsprechend der Codenummer der Zeichen in der Zeichentabelle sortiert (siehe der Beispielablauf unten), sodass das Zeichen mit der größten Codenummer im ersten Feldelement steht.

Nachfolgend wird die Häufigkeit, mit der die Zeichen im Feld vorkommen, gezählt und zusammen mit der Codenummer des Zeichens auf die Konsole ausgegeben.

Zeichen können Sie mit dem Befehl **getChar** der IOLib einlesen:

```
char ch = IOLib.getChar("Schriftzeichen eingeben (0: stop): ");
```

Beispielablauf:



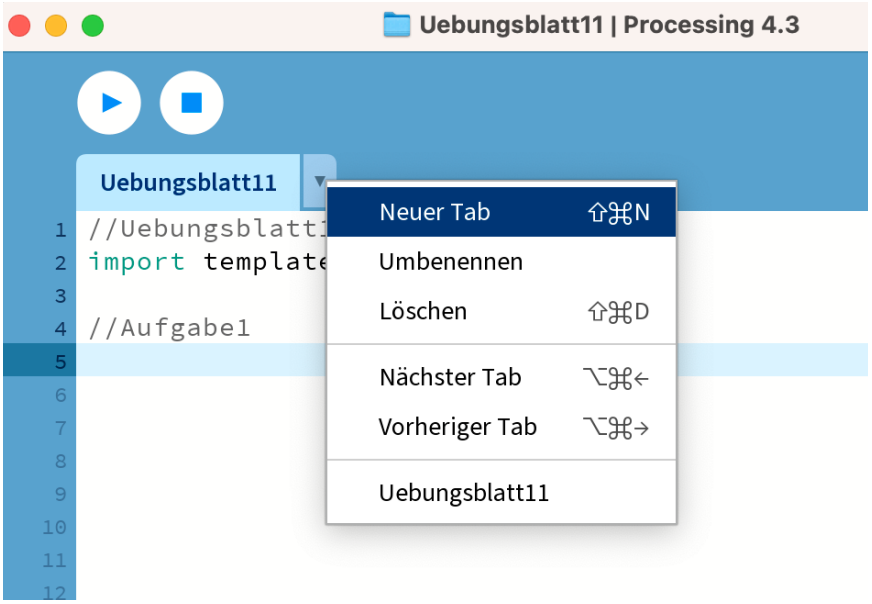
Die Codenummer des Zeichens steht jeweils in eckigen Klammern. Die hier abgebildeten Codenummern entsprechen der von Windows im europäischen Raum auf der Konsole eingesetzten Codepage 850.

Übungsaufgaben zum Thema Klassen und Objekte

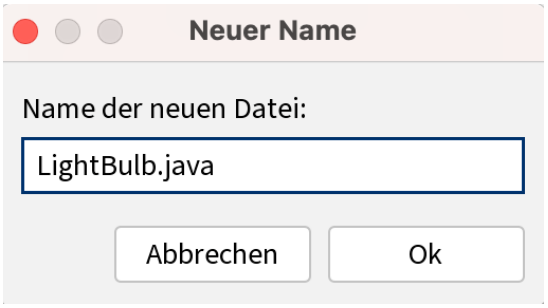
Aufgabe 1: Einführung in Klassen

1. Implementieren Sie das Programm mit dem LightBulb-Objekt aus der Vorlesung. Legen Sie unter Processing zuerst eine neue Sketch an (Uebungsblatt11).

Öffnen Sie jetzt einen neuen Tab, indem Sie auf den kleinen Pfeil klicken und dort "Neuer Tab" anklicken ..



.. und die Datei LightBulb.java erstellen.



Füllen Sie die neu angelegte Klassendatei (LightBulb.java) mit dem Code der Vorlesungsfolie, die die Überschrift "Vollständige LightBulb-Klasse" trägt und speichern Sie danach die Datei. Die öffentlichen Eigenschaften bzw. Funktionalitäten dieser Klasse können im Hauptprogramm durch Deklaration von Variablen, die diese Klasse als Datentyp besitzen, benutzt werden. Wir können Variablen des Typs `LightBulb` deklarieren, weil die neu erstellte Klasse Teil des aktiven Projekts und damit der Entwicklungsumgebung bekannt ist.

2. Ergänzen Sie Ihr Hauptprogramm (Uebungsblatt11.pde) um den Programmcode der Folie "Beispiel: Programm mit LightBulb-Objekt", um die neue Klasse auszuprobieren. Im Code wird ein neues Objekt vom Typ `LightBulb` erzeugt und dieses durch Aufruf der entsprechenden Methoden des Objekts "ein"- und "ausgeschaltet". Führen Sie den Code mit aktiviertem `lb.switchOff();` aus. Speichern Sie danach den Code mit aktiviertem `lb.switchOn();` und führen Sie ihn erneut aus. Was stellen Sie jeweils fest?

3. Kommentieren Sie `lb.switchOff();` und `lb.switchOn();` in Ihrem Hauptprogramm aus. Speichern Sie den Code und rufen Sie das Programm auf. Angezeigt wird jetzt der Initialisierungswert der Variablen `light`. Setzen Sie die Variable auf die beiden möglichen Anfangswerte (welche sind dies?), kompilieren Sie jeweils die Datei und führen Sie das Programm danach aus. (Es ist guter Stil Variablen nur initialisiert zu benutzen.)

4. Versuchen Sie nun den Wert der Variablen `light` durch eine Zuweisung `lb.light = true;` im Hauptprogramm zu ändern. Gibt es dabei ein Problem? Falls ja: Wie ließe sich dieses lösen?

5. Ändern Sie Ihr Hauptprogramm so ab, dass für die Überprüfung des logischen Zustandes der Variable `light` keine Hilfsvariable `lightState` mehr benötigt wird.

6. Ändern Sie das Attribut der Methode `isOn()` zu `private` und speichern Sie die Datei. Was passiert jetzt?

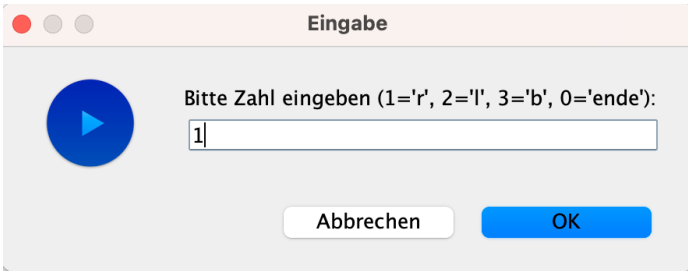
7. Nehmen Sie die Änderung der letzten Übung wieder zurück. Übernehmen Sie jetzt den Code des Hauptprogramms von der Folie "Mehr als ein Objekt erzeugen" in Ihr eigenes Hauptprogramm. Speichern Sie es und führen Sie es aus. Probieren Sie unterschiedliche Kombinationen der `switchOn()`-`switchOff()`-Methoden-Aufrufe und ihre Auswirkung auf die Ausgaben des Programms.

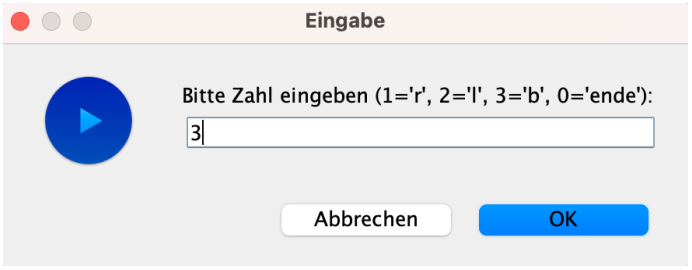
Aufgabe 2: Zwei Glühlampen ein- und ausschalten

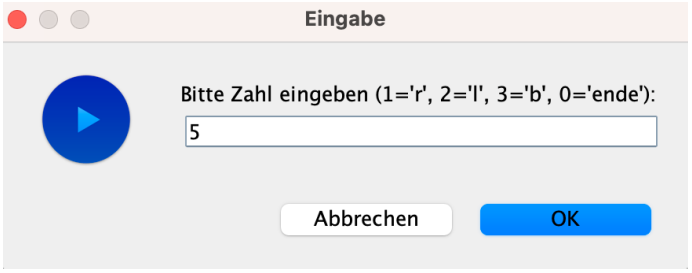
Erweitern Sie Ihr Hauptprogramm derart, dass mit einer überwachungsgesteuerten `do-while`-Schleife zwei Glühlampen, eine rechte und eine linke, ein- und ausgeschaltet werden. Hierzu werden mit Hilfe der IOLib Integer-Zahlenwerte eingelesen. Wird "1" eingegeben, so wird die rechte Glühlampe eingeschaltet, bei "2" die linke, bei "3" beide Glühlampen. Bei "0" wird das

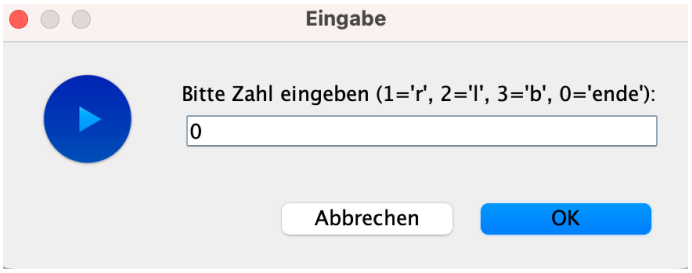
Programm beendet. Wird irgendeine andere Zahl eingegeben, so werden beide Glühlampen ausgeschaltet. Geben Sie den Zustand der Glühlampen jeweils nach dem Einlesen des Zeichens aus. Dies kann entweder grafisch mit `IOLib.putString("Ausgabestring");` in Form von Meldungen geschehen, wie im Beispiel unten, oder über die Konsole. Achten Sie darauf, dass Sie den Zustand der Glühlampen nur mit den Methoden der Klasse ändern und auslesen.

Beispielablauf:









Aufgabe 3: Lichterkette

1. Klassen können als Typ eines Felds benutzt werden. Probieren Sie dies an Hand eines Felds von `LightBulb`-Objekten aus. Kommentieren Sie den vorhandenen Code aus und erweitern Sie Ihr Hauptprogramm dann um ein Feld von fünf Glühlampen. Die Erstellung des Felds geschieht in zwei Schritten:

    1. Zunächst wird ein Feld erzeugt, welches fünf Elemente des Typs `LightBulb` aufnehmen kann:

        ```
        LightBulb[] lbarr = new LightBulb[5];
        ```

    2. Dieser Befehl erzeugt keine Glühlampen, sondern nur Speicherplätze, die Glühlampen aufnehmen können. Die eigentlichen Glühlampen werden in einem zweiten Schritt durch mehrfaches Instanzieren der Klasse `LightBulb` erzeugt und jeweils den einzelne Feldelementen zugewiesen. Dies sieht für das erste Feldelement mit dem Indexwert 0 z. B. so aus:

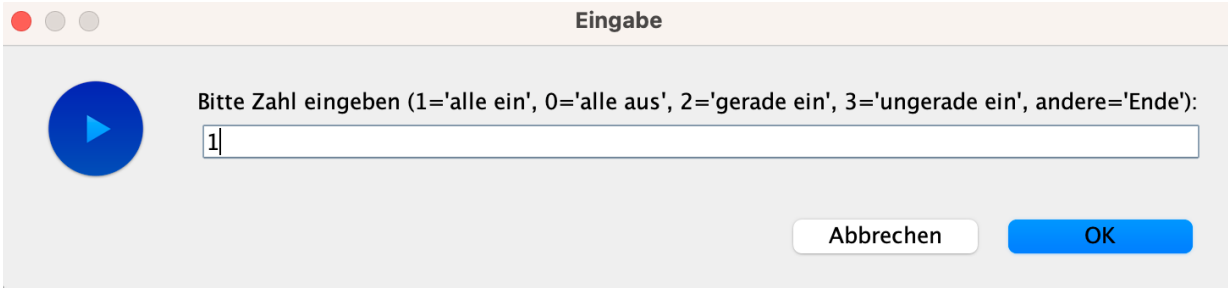
        ```
        lbarr[0] = new LightBulb(); // usw.
        ```

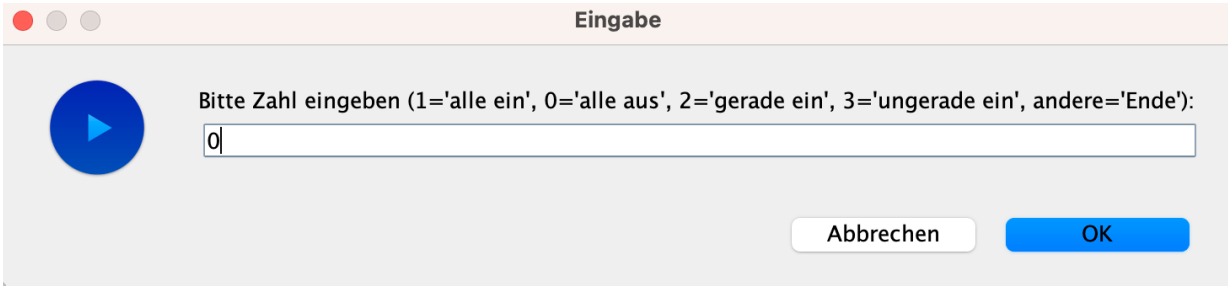
        Das Zuweisen der einzelnen Glühlampen an die Feldelemente kann auch mit Hilfe einer for-Schleife geschehen, um unnötigen Programmcode zu vermeiden.

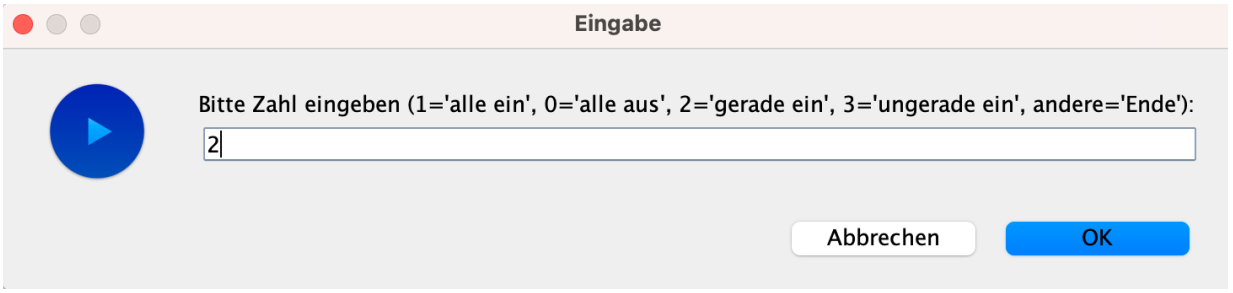
Bei der Erstellung von Feldern, die einen abstrakten Datentyp besitzen, muss immer unterschieden werden zwischen den Feldelementen, die das jeweilige Objekt aufnehmen sollen und den gesondert zu erzeugenden Objekten.

Die Glühlampen sollen durch Benutzereingaben ein- und ausgeschaltet werden können. Wird im Programm eine 1 eingegeben, so werden alle Lampen der Lichterkette eingeschaltet. Bei Eingabe einer 0 werden alle Lampen ausgeschaltet. Durch Eingabe einer 2 werden die geradzahligen Lampen eingeschaltet, durch Eingabe einer 3 die ungeradzahligen Lampen. Danach wird jeweils der Status der Lampen ausgegeben. Die Eingabe eines anderen Zahlenwerts beendet das Programm.

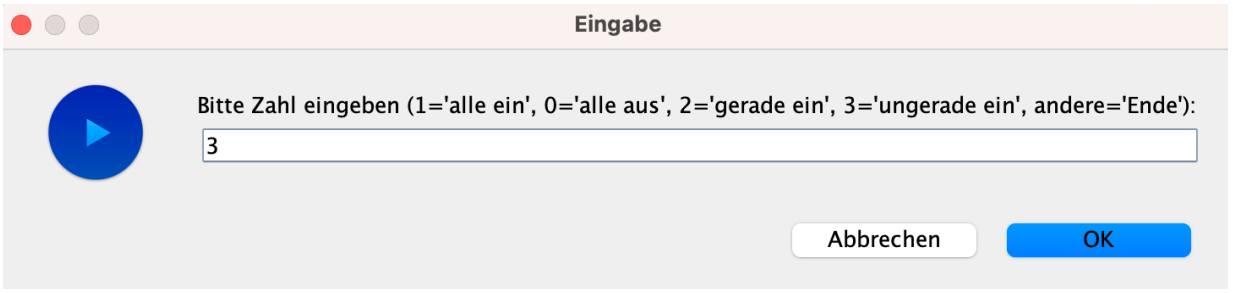
Beispielablauf:

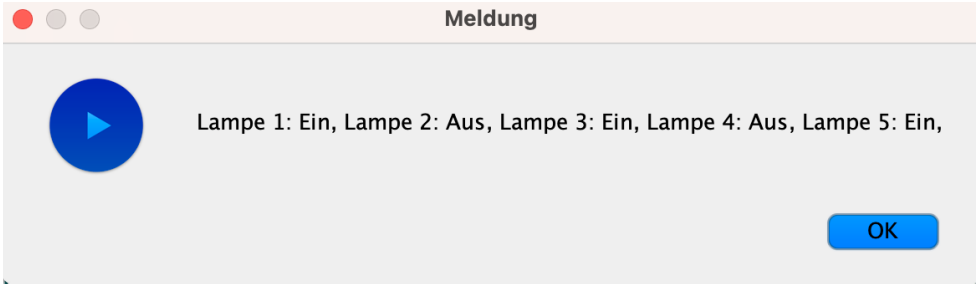


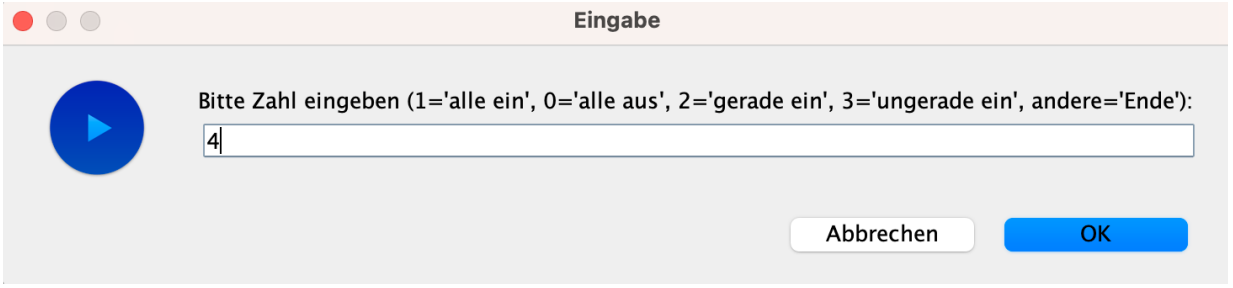



Eingabe
Bitte Zahl eingeben (1='alle ein', 0='alle aus', 2='gerade ein', 3='ungerade ein', andere='Ende'):
2
Abbrechen
OK


Meldung
Lampe 1: Aus, Lampe 2: Ein, Lampe 3: Aus, Lampe 4: Ein, Lampe 5: Aus,
OK


Eingabe
Bitte Zahl eingeben (1='alle ein', 0='alle aus', 2='gerade ein', 3='ungerade ein', andere='Ende'):
3
Abbrechen
OK


Meldung
Lampe 1: Ein, Lampe 2: Aus, Lampe 3: Ein, Lampe 4: Aus, Lampe 5: Ein,
OK

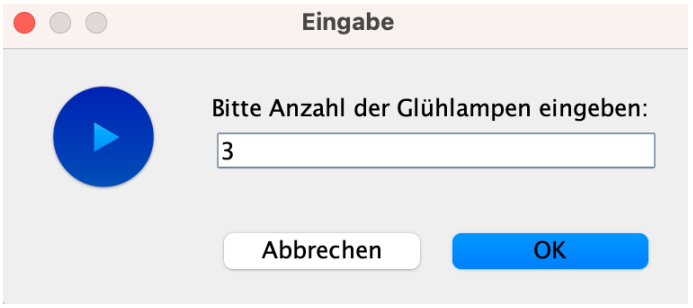

Eingabe
Bitte Zahl eingeben (1='alle ein', 0='alle aus', 2='gerade ein', 3='ungerade ein', andere='Ende'):
4
Abbrechen
OK

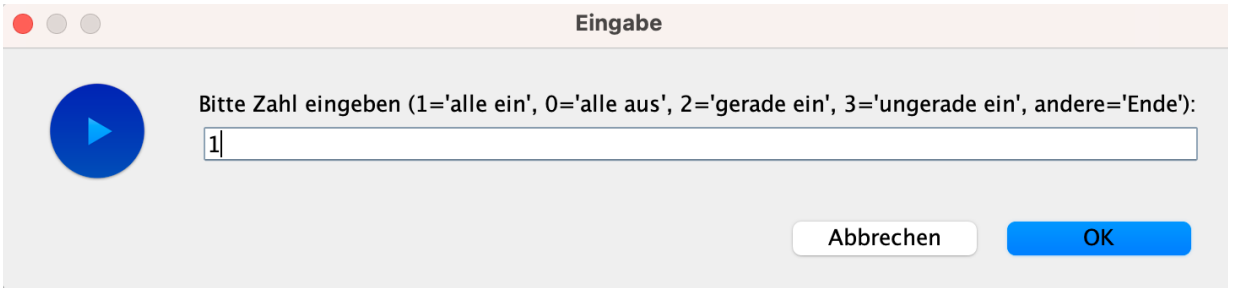

Meldung
Ade
OK

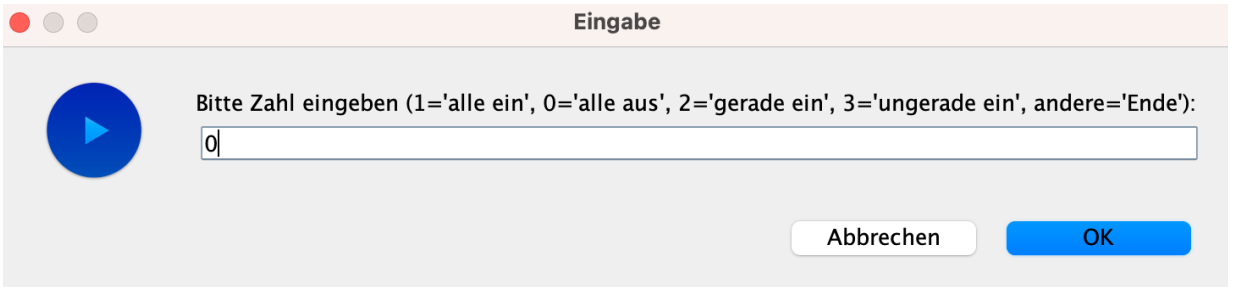
2. Erweitern Sie Ihre Lösung dahingehend, dass jetzt eine beliebige Anzahl von Glühlampen verwendet werden kann.

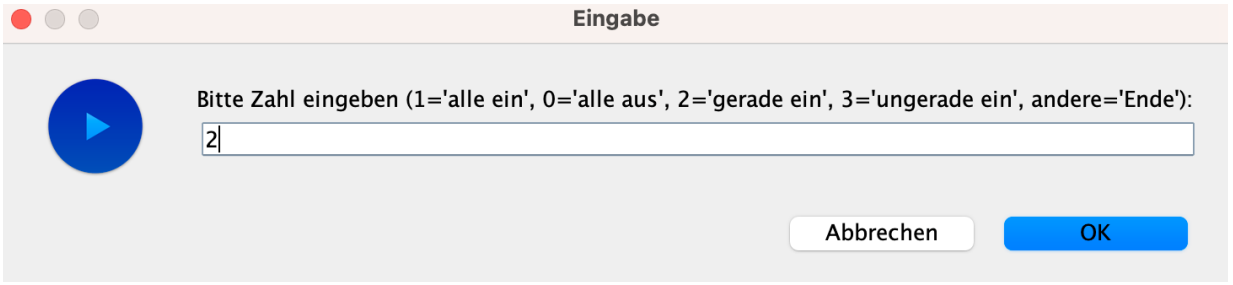
   Das Programm erfragt zunächst die gewünschte Anzahl an Glühlampen, erzeugt diese und geht dann in die Schleife, in der die Glühlampen in ihrer Gesamtheit mit Hilfe von Benutzereingaben ein- und ausgeschaltet werden können.

   Beispielablauf:

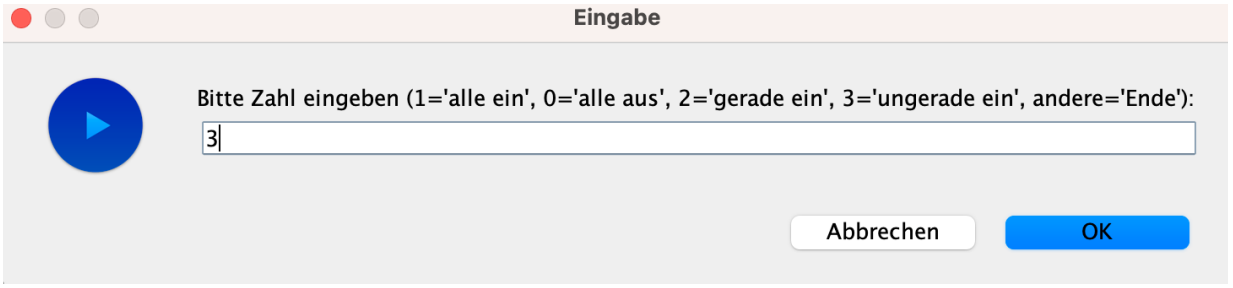




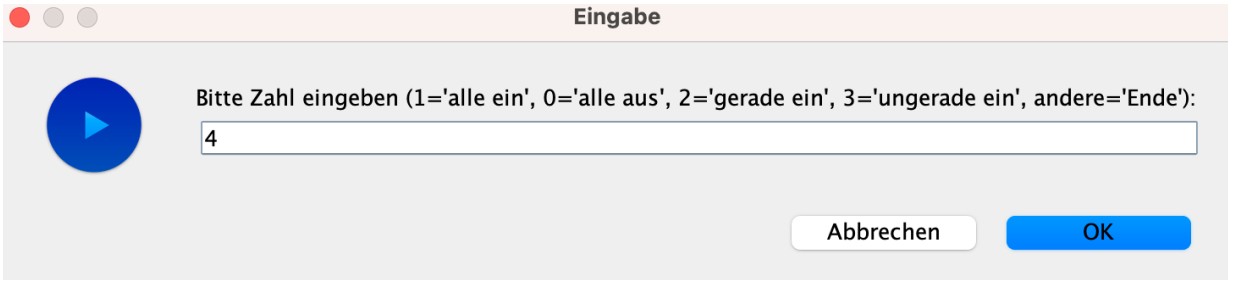



Eingabe
Bitte Zahl eingeben (1='alle ein', 0='alle aus', 2='gerade ein', 3='ungerade ein', andere='Ende'):
2
Abbrechen
OK


Meldung
Lampe 1: Aus, Lampe 2: Ein, Lampe 3: Aus,
OK


Eingabe
Bitte Zahl eingeben (1='alle ein', 0='alle aus', 2='gerade ein', 3='ungerade ein', andere='Ende'):
3
Abbrechen
OK


Meldung
Lampe 1: Ein, Lampe 2: Aus, Lampe 3: Ein,
OK


Eingabe
Bitte Zahl eingeben (1='alle ein', 0='alle aus', 2='gerade ein', 3='ungerade ein', andere='Ende'):
4
Abbrechen
OK


Meldung
Ade
OK

Übungsaufgaben zum Entwurf von Methoden und Klassen

Aufgabe 1: Polynomauswertung

- Erzeugen Sie ein neues Programmierprojekt. Erzeugen Sie in diesem eine neue Klasse `Polynom.java` und füllen Sie diese mit dem Code von der Folie "Parametrierte Methode: Vollständiges Beispiel". Kopieren Sie das Hauptprogramm von dieser Folie in die Sketch ihres Programmierprojekts. Führen Sie dann das Programm aus, sodass dieses die Ausgabe von der Folie liefert.
- Ändern Sie die Parameterwerte aller Parameter **innerhalb** der Methode, wie auf der Folie mit dem Titel "Zuweisungen an Parameter (1)" beschrieben. Geben Sie in Ihrem Hauptprogramm, wie auf der Folie mit dem Titel "Zuweisungen an Parameter (2)", die Werte der Variablen, die den Parametern zugewiesen wurden, vor und nach dem Methodenaufruf sowie wieder den Rückgabewert der Methode aus und überprüfen Sie auf diese Weise, dass in der Tat, als Folge der **Wertübergabe**, Zuweisungen an die Parameter im Methodenrumpf keine Auswirkungen auf die Werte der beim Methodenaufruf übergebenen Variablen haben.

Aufgabe 2: Überladene Methoden

1. Implementieren Sie die beiden Varianten der Methode `polynomEvaluation` von der Folie "Überladene Methode" und probieren Sie deren Aufruf aus, wie auf der Folie "Überladene Methode: Aufruf in der Sketch" beschrieben.
2. Legen Sie in Ihrem Hauptprogramm eine neue Klasse `Summation` an und in dieser zwei Varianten einer Methode

   ```
   static int sum(...)
   ```

   für das Summieren von Zahlenwerten. Dabei stehen die Punkte `...` im Methodenkopf oben für die unterschiedlichen Parametermengen der beiden Varianten.

   1. **Variante**: Diese besitzt zwei Parameter: `z1` und `z2` vom Typ `int`. Die Aufgabe der Methode besteht darin, die in diesen Parametern übergebenen Werte zu addieren und das Ergebnis mit `return` an die *Aufrufumgebung*, d. h. den Programmcode, in den der Aufruf der Methode eingebettet ist, zurückzugeben.
   2. **Variante**: Diese besitzt drei Parameter: `z1`, `z2` und `z3` vom Typ `int`. Entsprechend zur ersten Variante besteht die Aufgabe der Methode wieder darin, die in diesen Parametern übergebenen Werte zu addieren und das Ergebnis mit `return` an die Aufrufumgebung zurückzugeben.

   Wie oben angegeben, besitzen beide Varianten den gleichen Methodennamen.

   Testen Sie die Varianten der Methode in Ihrem Hauptprogramm durch das Übergeben einiger Zahlenwerte, wie z. B. 1, 2, 3 usw. als Parameterwerte und das Ausgeben der jeweils

berechneten Methodenwerte.

Aufgabe 3: Zirkus

Im Zirkus sind der Clown Beppo und der Seehund Charlie eine große Attraktion. Immer, wenn Beppo einen Ball wirft, balanciert Charlie diesen auf der Nase. Geworfene Ringe lässt Charlie um seinen Hals rotieren. Wirft Beppo zur Belohnung einen Fisch, so frisst Charlie diesen.

1. Definieren Sie eine Klasse `Clown`. Clowns können einen Namen haben, der bei der Instanzierung dem Konstruktor der Klasse als Zeichenkette übergeben wird. Zeichenketten werden in Variablen vom Typ `String` gespeichert.

   Das einzige Verhalten eines Clowns ist, dass er Bälle, Ringe und Fische wirft. Dieses Verhalten lässt sich mit Hilfe folgender Methode implementieren:

   ```
   public void werfen (Seehund s) {...}
   ```

   Die Wurfobjekte lassen sich in der Methode durch die Integerwerte 1, 2, und 3 codieren. Die zum Werfen gehörende Handlung wird in der Methode durch Textausgaben auf die Konsole simuliert, entsprechend zur Beispielausgabe unten. Bei jedem Aufruf der Methode `werfen` wirft der Clown nur ein Objekt. Das Werfen des jeweiligen Objekts löst bei dem als Parameterwert übergebenen Seehund eine Fangreaktion aus.

2. Definieren Sie eine Klasse `Seehund`. Auch Seehunde können einen Namen haben. Dieser wird wieder bei der Instanzierung der Klasse angegeben. Daneben besitzen Seehunde das von außen auslösbare Verhalten, d. h. die Methode

   ```
   public void fangen (int objekt) {...}
   ```

   und die *privaten* Verhaltensweisen `balancieren`, `rotieren` und `fressen`, die wieder für Methoden stehen.
   Die hierzu gehörenden Handlungen werden wieder durch Textausgaben simuliert. Das Verhalten `fangen` wird vom Clown aufgerufen und löst im Seehund das Balancieren, Rotieren oder Fressen aus.

3. Erzeugen Sie in Ihrem Hauptprogramm ein Clown-Objekt mit dem Namen `Beppo` und ein Seehund-Objekt mit dem Namen `Charlie`. Danach wirft `Beppo` in einer Schleife abwechselnd einige Male einen Ball, einen Ring und einen Fisch zu `Charlie`.

Nutzen Sie bei der Textausgabe die in dem jeweiligen Objekt gespeicherte Information über den Objektnamen. Beachten Sie, dass Sie in `.java`-Dateien nicht die abgekürzte Formen `println` bzw. `print` für die Ausgabe benutzen können, sondern nur die Java-Varianten `System.out.println` und `System.out.print`. Alle Programmausgaben finden bei dieser Aufgabe in den Java-Dateien statt und nicht aus der Sketch-Datei heraus.

Beispielausgabe:

```
Beppo wirft Ball
Charlie balanciert Ball
Beppo wirft Ring
Charlie rotiert Ring
Beppo wirft Fisch
Charlie frisst Fisch
```

```
Beppo wirft Ball
Charlie balanciert Ball
Beppo wirft Ring
Charlie rotiert Ring
Beppo wirft Fisch
Charlie frisst Fisch
usw.
```

Aufgabe 4: Autovermietung

Eine Autovermietung möchte die Bestellungen der Kunden zukünftig effizient über ein Kundenterminal abwickeln. Das Terminal soll alle hinsichtlich der Bestellung relevanten Daten in Objekten vom Typ `Car` speichern, in denen die konkreten Bestellwünsche der Kunden codiert sind. Im Einzelnen können die Kunden am Terminal ihre Wunschfarbe als Zeichenkette frei wählen (z. B. "rot") und angeben, ob das Auto eine Automatik und/oder ein Schiebedach haben soll oder nicht.

1. Konstruieren Sie eine Klasse `Car`, die in ihren *privaten* Klassenvariablen die gewünschte Farbe sowie die gewählten Werte für die Automatik und das Schiebedach codiert. Die Kenndaten werden mit Hilfe eines Konstruktors

   ```
   Car(String color, boolean automatic,
       boolean slidingRoof) {...}
   ```

   erfasst und im Objekt abgelegt.

   Schreiben Sie drei Methoden, mit deren Hilfe Sie die Daten auslesen können:

   - `public String getColour() {...}`
   - `public boolean hasAutomatic() {...}`
   - `public boolean hasSlidingRoof() {...}`

2. Schreiben Sie ein Programm, in dem Kundenwünsche zunächst, wie im Beispiel unten, mit Hilfe von Eingaben abgefragt werden. Speichern Sie die erfassten Daten in einem Objekt vom Typ `Car`. Die gespeicherten Daten werden danach aus dem Objekt ausgelesen und wie in dem folgenden Beispiel auf die Konsole ausgegeben.

   Sinngemäßer Ablauf der Programmeingabe über `IOLib`:

   ```
   Welche Farbe wünschen Sie? weiß
   Wünschen Sie eine Automatik? (false: nein, true: ja) true
   Wünschen Sie ein Schiebedach? (false: nein, true: ja) false
   ```

   Programmausgabe (ohne Ausgaben aus der `IOLib`):

   ```
   Das von Ihnen bestellte Auto ist wie gewünscht weiß,
   besitzt eine Automatik und kein Schiebedach.
   ```

   Berücksichtigen Sie bei der Programmausgabe alle möglichen Eingabefälle.

   Für die Programmausgabe werden die Daten ausgelesen, die im `Car`-Objekt gespeichert

wurden. Alle Programmausgaben geschehen bei dieser Aufgabe über die Sketch. Die Objekte dienen nur als Daten-Container.
