Luigi LO IACONO
Stephan WIEFLING
Michael SCHNEIDER

3. Auflage

PROGRAMMIEREN TRAINIEREN

Mit über 150 Workouts in Java und Python

Im Internet:
GitHub-Repository zum Buch

Lösungen unter
plus.hanser-fachbuch.de

HANSER

Lo Iacono / Wiefling / Schneider

Programmieren trainieren

Ihr Plus – digitale Zusatzinhalte!

Auf unserem Download-Portal finden Sie zu diesem Titel kostenloses Zusatzmaterial.

Geben Sie auf **plus.hanser-fachbuch.de** einfach diesen Code ein:

plus-23nr4-W2umK

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:

www.hanser-fachbuch.de/newsletter

Luigi Lo Iacono
Stephan Wiefling
Michael Schneider

Programmieren trainieren

Mit über 150 Workouts in Java und Python

3., überarbeitete Auflage

HANSER

Die Autoren:
Prof. Dr.-Ing. Luigi Lo Iacono, Bonn
Stephan Wiefling, Köln
Michael Schneider, Daaden

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Lektorat: Sylvia Hasselbach
Copy editing: Sandra Gottmann, Wasserburg
Layout: le-tex publishing services GmbH
Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München
Umschlagrealisation und Titelmotiv: © Max Kostopoulos
Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen
Printed in Germany

Print-ISBN: 978-3-446-47766-7
E-Book-ISBN: 978-3-446-47821-3
E-Pub-ISBN: 978-3-446-47999-9

Inhalt

Vorwort

Der Messenger auf deinem Smartphone, der Bluetooth-Lautsprecher, mit dem du deine Lieblingsmusik abspielst, der Algorithmus, der auf Netflix die neueste Serie vorschlägt – jemand hat all das programmiert. Fast nichts in der Welt kommt noch ohne Code aus. Du bist grade auf dem besten Weg, ein Teil davon zu werden und die Zukunft mitzugestalten. Wie aufregend!

Klar, es gibt auch frustrierende Momente – wenn du nicht herausfinden kannst, wo der Fehler in deinem Code liegt, oder wenn du das Kapitel über Objektorientierung in deinem Lehrbuch nicht verstehst. Aber keine Sorge, wir alle haben diese Erfahrung gemacht. Es braucht Zeit und Übung, um wirklich gut im Programmieren zu werden.

Ich selbst habe angefangen, Informatik zu studieren, ohne jemals auch nur eine Zeile Code gesehen zu haben. Und oh boy, war das eine Herausforderung. Am Anfang fiel es mir echt schwer. Aber irgendwas hat mich gepackt, und mit jeder Übungsaufgabe wurde es ein bisschen einfacher. Und je mehr ich gelernt habe, desto mehr habe ich gemerkt, wie viel Spaß programmieren macht. Es ist einfach das beste Gefühl, wenn man den Fehler im Code dann irgendwann doch findet und das Programm endlich genau so funktioniert, wie man es sich vorgestellt hast.

Und irgendwann siehst du dann, was du mit diesem Handwerk alles anstellen und wie kreativ du mit Code werden kannst. In meiner Arbeit als Spieleprogrammiererin kann ich mit nur wenigen Zeilen Code die Steuerung für eine Spielfigur festlegen, eine Unterhaltung zwischen zwei Charakteren auf dem Bildschirm anzeigen oder ein ganzes Level automatisch generieren lassen. Die Möglichkeiten sind fast endlos! Und das Beste: Es gibt selten nur eine richtige Lösung für ein Problem. Das gibt uns beim Programmieren die Chance, unsere eigenen Ideen und Visionen umzusetzen. Und nein, wir sitzen nicht den ganzen Tag allein im Keller vor dem Bildschirm, wie Kriminelle beim Hacken in einem Hollywoodfilm. Wir arbeiten im Team an spannenden Projekten, um gemeinsam ein Ziel zu erreichen.

Mit den Übungsaufgaben in diesem Buch (in dieser Auflage übrigens mit 20 brandneuen Aufgaben) kannst du deine Fähigkeiten im Programmieren Schritt für Schritt ausbauen. Damit auch du dich bald an eigene und komplexere Projekte wagen kannst und mit deinem Code die Zukunft gestaltest!

Kathrin Radtke (Spellgarden Games), im Januar 2023

Neues in der 3. Auflage

Im Vergleich zu den ersten beiden Auflagen hat sich in der 3. Auflage erneut einiges geändert.

Wie du dir vorstellen kannst, sind **neue Aufgaben** hinzugekommen. Dieses Mal sind es zwanzig Aufgaben mehr im Vergleich zur 2. Auflage. Wir haben die neuen Aufgaben gekennzeichnet, damit du sie schnell finden kannst. Sowohl im Inhaltsverzeichnis als auch in der Aufgabenüberschrift befindet sich der Hinweis „Neu“, an dem du die neuen Aufgaben direkt erkennen kannst. Ansonsten sind die Aufgaben in den Kapiteln nach unserer subjektiven Einschätzung der Schwierigkeit, des Zeitaufwands und der erforderlichen Kreativität aufsteigend sortiert. Du kannst dich also auch gut von vorne nach hinten durcharbeiten.

Außerdem haben wir **Fehler korrigiert**, die uns von Leserinnen und Lesern zuletzt zur 2. Auflage zurückgemeldet wurden (siehe auch Danksagung). Es versteht sich von selbst, dass wir zudem alles Erforderliche **auf den aktuellen Stand** gebracht haben. Dies betrifft insbesondere alle Kapitel, in denen die dem Buch zugrunde gelegte Programmierumgebung „Processing“ behandelt wird. Auch hier haben wir für die 3. Auflage das Feedback unserer Leserschaft aufgegriffen und genauer beschrieben, wie die Beispiellösungen auch mit einer anderen Programmierumgebung verwendet werden können. Für die Aufgaben, für die die Verwendung außerhalb von Processing ohne größere Änderungen an der Musterlösung möglich ist, haben wir eine Kennzeichnung eingeführt, die dies ersichtlich macht. Das Icon, das an jeder Aufgabe zur visuellen Unterstützung steht, ist dann eingerahmt und mit dem Text „Auch ohne Processing lösbar“ versehen.

Eine weitere wichtige Neuerung der 3. Auflage ist, dass wir uns entschlossen haben, die Beispiellösungen nicht mehr im Buch mit abzudrucken. Der Grund dafür ist, dass wir wertvolles **Papier einsparen** möchten. Wie du es vermutlich schon von Hanser-Büchern und natürlich auch von unseren ersten beiden Auflagen gewohnt bist, hast du viele verschiedene Zugänge zu den Beispiellösungen. Diese sind z. B. im E-Book enthalten oder können als Programmtexte von GitHub oder direkt von Hanser heruntergeladen werden. Also wundere dich nicht, wenn bei den Anhängen C und D die Lösungen, die im E-Book enthalten sind, im gedruckten Buch fehlen. Jetzt weißt Du, warum dies der Fall ist.

Darüber hinaus erhältst du als Käufer:in des gedruckten Buches einen Code, mit dem du das komplette E-Book herunterladen kannst. Diesen Code findest du ganz vorne im Buch unter der Überschrift „E-Book inklusive“.

Danksagung

Ein Buchprojekt ist harte Arbeit. Ohne die Unterstützung vieler helfender Hände geht es nicht. Wir können uns gar nicht genug bei euch allen bedanken, versuchen es aber dennoch, so gut wir können.

Zeit ist wohl das Kostbarste, was wir haben. Darum bin ich umso dankbarer, dass meine liebe Familie mir diese für derartige und andere Projekte einräumt. Danke, Barbara, Giuliana, Antonio und Fabio.

Danke an Brigitte, Frank, Christian und alle anderen wundervollen Menschen, die mich bei der Arbeit an diesem Buch unterstützt haben. Außerdem danke ich allen Förderinnen und Förderern, besonders meinem damaligen Informatiklehrer „Herr Schepanek“.

Ich danke allen Kolleginnen und Kollegen, die mich im Laufe der Jahre begleitet haben. Besonders möchte ich mich bei Katrin für ihre Geduld und ihren Support sowie bei Juli und Lana für ihre Motivation bedanken.

Gemeinsam möchten wir uns bei Christian Ullenboom bedanken. Er hat das Buch kritisch durchgearbeitet und uns viele wertvolle Anmerkungen dazu gegeben. Gleiches gilt für Dirk Louis, der uns zudem freundlicherweise das Vorwort zur Erstauflage des Buches geschrieben hat. Bei Kathrin Radtke und Patrick Stenzel möchten wir uns für die Vorworte zur dritten bzw. zweiten Auflage bedanken. Dem Hanser Verlag und insbesondere Sylvia Hasselbach möchten wir für die viele Geduld mit uns und die ungebrochene Unterstützung bedanken. Ein derartiger Rückhalt ist unverzichtbar.

Außerdem bedanken wir uns bei Remo Lötscher, Lothar Massing, Lars Mühlbauer, Jens Schönbohm und Lars Wildeshaus, die uns auf kleine Fehler in den vorherigen Auflagen aufmerksam gemacht haben, welche wir in dieser Auflage korrigieren konnten.

Schließlich wollen wir uns an dieser Stelle bei den vielen Freiwilligen da draußen bedanken, die ihre wertvolle Zeit dafür aufwenden, um der Allgemeinheit viele nützliche Dinge kosten- und diskriminierungsfrei zur Verfügung zu stellen. Unser Buch bedient sich viel aus der Public Domain, wofür wir uns gerne durch Benennung der wesentlichen Bausteine bedanken wollen. Zunächst sind hierzu die beiden zugrunde gelegten Programmiersprachen zu nennen. **Java** (https://www.java.com/de/) wird von der Oracle Corporation und **Python** (https://www.python.org/) von der Python Software Foundation bereitgestellt. Beide gehören aktuell zu den am meisten eingesetzten Sprachen und können auf vielfältigste Weise verwendet werden. Um die Hürden gerade für den (fachfremden) Einstieg in die Programmierung weitestgehend zu eliminieren, stellt die Processing Foundation die gleichnamige Entwicklungsumgebung zur Verfügung (https://processing.org/). **Processing** basiert dabei von Hause aus auf Java. Im Laufe der Zeit sind weitere Programmiersprachen hinzugekommen, darunter neben JavaScript auch Python (https://py.processing.org/). Wir verwenden im Buch zudem **Piktogramme**, um die Aufgaben durch kleine Icons visuell zu unterstreichen. Iconify.it stellt eine Sammlung von 650 freien Glyphicons bereit, aus der wir uns hierzu bedient haben. Schließlich verwenden wir in einigen Programmieraufgaben Bilder, die durch das Programm verarbeitet werden sollen. Hier haben wir auf die Public-Domain-Cliparts von clker.com (https://clker.com/) zurückgegriffen. Ebenso haben wir das Public-Domain-Bild „squirrel“ von Lola Williams in zwei Aufgaben verwendet.

Last, but not least wollen wir uns **bei dir bedanken**. Wir freuen uns, dass wir dein Interesse geweckt und es schon mal bis in deine Hände geschafft haben. Jetzt bleibt uns nur noch, dir beim Programmierentrainieren viel Erfolg und auch Spaß zu wünschen.

Luigi Lo Iacono, *Stephan Wiefling* und *Michael Schneider*
August 2023

Vorwort zur zweiten Auflage

> *„Jede hinreichend fortschrittliche Technologie ist von Magie nicht zu unterscheiden.“*
> — Arthur C. Clarke

Wir sind Programmierer. Wir sind Magier. Das MIT ist unser Hogwarts, der Google Campus ist unsere unsichtbare Universität, Cupertino ist unser Narnia. Steve Jobs ist unser David Copperfield, Larry Page und Sergey Brin sind unsere Ehrlich Brother, und Frank Thelen ist immerhin vielleicht noch so was wie unser Vincent Raven. Wir sind Siegfried und Roy, und aus Versehen programmierte Endlosschleifen sind unsere weißen Tig. . . strapazieren wir die Allegorie mal nicht über. Jedenfalls: Wir sind Magier.

Oder wenigstens wirken wir für unser Umfeld so. Der Onkel dritten Grades, der im Atomkraftwerk arbeitet, würde uns selbst dann anrufen, wenn AssetWise mal hakt, weil wir eben Informatiker sind und uns dementsprechend mit allem auskennen, was irgendwie mit Computern zusammenhängt. Dabei ist es auch egal, wie komplex oder unterkomplex die Aufgabe ist. Wir werden angerufen, wenn ein Teilchenbeschleuniger angesteuert werden muss, aber auch, wenn es im Fachgeschäft für Strickzubehör „Woll im Leben“ einer Freundin des Freundes deiner Tante väterlicherseits in der alten Fußgängerzone der Kleinstadt, in der du geboren wurdest, nicht mehr ganz so gut läuft und sie jetzt „mal eben“ einen Shop braucht, um den Globalisierungseffekt besser für sich zu nutzen und das Wollgeschäft im Sturm zu erobern. Gestrickt wird ja wohl überall, und sie ist sogar bereit, dir für die drei Wochen Arbeit noch 100 € in die Hand zu drücken. Dafür müsstest du dann aber auch die nächsten drei Jahre zu jeder Tages- und Nachtzeit für Support zur Verfügung stehen.

Doch wir steuern nicht nur die Stromversorgung und das weltweite Woll-Business. Wir halten Banken am Laufen, das Transportwesen und die Kommunikation, ohne uns läuft gar nichts mehr heutzutage. Wir können Welten erschaffen, und wir können sie auch zerstören, je nachdem, ob wir Harry Potter oder Lord Voldemort sein wollen.

Welchen Weg du einschlagen willst – und jeder, der schon mal programmiert hat und behauptet, niemals auf die dunkle Seite geschaut zu haben, lügt – entscheidest du selbst, und auf dem Weg zu deiner Entscheidung ist dieses Buch dein Hogwarts Express, und du musst nicht mal gegen eine Mauer rennen, um hineinzugelangen. Du hast die erste Seite aufgeschlagen und das Vorwort gelesen und die Richtung, in der du weiterblätterst, ist deine rote und deine blaue Pille.

Schlag es wieder zu – dann endet die Geschichte hier, du wachst auf in deinem Bett und glaubst, was immer du glauben möchtest. Blätter weiter, bleib im Wunderland, und das Buch zeigt dir, wie tief der Kaninchenbau geht. Nerd today, boss tomorrow.

Patrick Stenzel (@rock_galore), im Januar 2020

Vorwort zur ersten Auflage

Nerds sind in. Diese liebenswerten Zeitgenossen mit dem vielen Spezialwissen und den kindlichen Vorlieben für Superhelden werden lange nicht mehr nur komisch beäugt. Im Gegenteil. Sie selbst sind nunmehr Stars in vielen Fernsehserien, und ihr modischer Stil ist allgemein akzeptiert. Diese Entwicklung kommt auch der Programmierung zugute. Lange Zeit galt diese Fertigkeit als ein Gebiet, das den Nerds vorbehalten ist. Dem ist nicht so! Es muss nur der Mut aufgebracht werden, sich damit auseinanderzusetzen. Dann wird schnell klar, was mit der Programmierung alles umgesetzt werden kann. Die Bandbreite ist groß und wird durch aktuelle Trends stetig befeuert. Insbesondere durch die Digitalisierung und Vernetzung vieler Alltagsgegenstände finden sich Softwareprogramme vermehrt jenseits gängiger Anwendungsfälle im betrieblichen Kontext von Unternehmen wieder. Also, keine Scheu und ran ans Programmieren!

Mir selbst bereitet das Programmieren viel Freude. Zudem ist es mir eine Herzensangelegenheit, mein Programmier-Know-how und meine Erfahrung an andere weiterzugeben. Ich weiß aus vielen Schulungen sehr genau, was es für Hürden und Stolpersteine beim Programmierenlernen gibt und wie diesen zu begegnen ist. **Gutem Trainingsmaterial kommt dabei eine zentrale Rolle zu**.

Die Autoren Lo Iacono, Wiefling und Schneider schließen hier eine wichtige Lücke. Sie versorgen dich mit vielen Trainingsaufgaben, die dir helfen werden, die wesentlichen Programmierkonzepte wirklich zu verstehen. Und mehr noch. Du kannst und solltest so lange mit den vielen Aufgaben trainieren, bis der Groschen tatsächlich gefallen ist. Das ist wichtig. Denn erst dann wirst du in der Lage sein, mit dem erlernten Handwerkszeug auch selbstständig Entwicklungsaufgaben bewältigen und lösen zu können. Genau da sollst du hin. Viele Lehrformate gehen hier nicht weit genug. Die falsche Annahme ist dabei häufig, dass ein Beispiel und eine Übungsaufgabe zum Verständnis ausreichen. Weit gefehlt. Es fängt schon damit an, dass nicht jeder mit dem gegebenen Beispiel oder der gestellten Übungsaufgabe etwas anfangen kann. Hier schafft das vorliegende Buch Abhilfe, und es gehört damit in die „*Einstieg in die Programmierung*“-Ecke deines Bücherregals.

Dirk Louis, im Januar 2018

1 Einleitung

1.1 Wozu sollte ich programmieren lernen?

Weil du es kannst und weil die Programmierung **das Werkzeug des 21. Jahrhunderts** ist. Die frühere Bundeskanzlerin Angela Merkel hat in einem Interview mit YouTuberinnen und YouTubern das Programmieren auf eine Stufe mit den Grundfertigkeiten Lesen, Schreiben und Rechnen gestellt (https://youtu.be/Uq2zIzscPgY?t=12m18s). Programmieren ist lange nicht mehr nur etwas für Fachleute, die das studiert haben. Durch den Einzug des Digitalen in alle Branchen und den Alltag können viele andere als nur Informatikfachkräfte von der Programmierung profitieren und damit ihre Ideen erproben und verwirklichen. Beispiele kannst du unzählige finden. Lass uns hier nur einige zur Verdeutlichung kurz anreißen. Dir fallen dann bestimmt selbst viele weitere Beispiele ein.

Angenommen, du schaffst **Kunst** und hast bisher mit den klassischen Materialien und Techniken deiner Disziplin gearbeitet. Für deine neueste Projektidee möchtest du mit regelmäßigen Formen und Farben experimentieren, wie es z. B. Sol LeWitt in seinem künstlerischen Schaffen getan hat (https://de.wikipedia.org/wiki/Sol_LeWitt). Das erfordert viel Fleiß, Geduld und Präzision. Da du deine Zeit lieber damit verbringen möchtest, an spannenden neuen Konstruktionen und deren Wirkung zu experimentieren, anstatt diese in langwierigen und teils monotonen Arbeitsschritten erst erstellen zu müssen, wünschst du dir einen Automatismus dafür, der das für dich erledigt. Dies kann ein eigens geschriebenes Computerprogramm leisten. Ist ein solches geschrieben, liegen die Vorteile auf der Hand. Veränderungen an den Farben, der Größe sowie Anordnungen der Formen usw. sind umgehend gemacht. Auch das Ausgabeformat kann leicht angepasst werden, um das Kunstwerk in vielfältiger Art und Weise zu drucken oder aus einem Rohling zu fräsen. Schlüsselfiguren der computergenerierten Kunst sind z. B. Manfred Mohr, Joseph Nechvatal, Olga Kisseleva und John Lansdown.

Als **Fachkraft für Veranstaltungstechnik** sieht man sich heute immer stärker mit Anforderungen von Kundinnen und Kunden konfrontiert, die nach noch nicht da gewesenen Hinguckern verlangen. Hierfür gibt es naturgemäß keine fertigen Lösungen, die man aus dem Regal ziehen kann. Somit siehst du dich auf der einen Seite immer mit neuen spannenden Entwicklungsaufgaben konfrontiert, musst dafür aber auf der anderen Seite adäquate Lösungen entwickeln. Diese bedingen eigentlich immer auch Software, die es zu programmieren gilt.

Im letzten fiktiven Szenario wollen wir ins **Internet der Dinge** abtauchen. Mit diesem Schlagwort wird der allgemeine Trend bezeichnet, mit dem die Digitalisierung und die Vernetzung im Gewand des Internets stetig in Gegenstände des alltäglichen Gebrauchs diffundieren. Der smart gewordene Fernseher ist ein Paradebeispiel hierfür. Einige neue Anwendungen findest du toll, willst aber noch nicht in neue Produkte investieren. Die alten tun es ja noch. So findest du es z. B. praktisch, im Supermarkt einen Blick in deinen Kühlschrank werfen zu können, um zu sehen, ob es genügend Frühstückseier fürs Wochenende gibt. Der Kühlschrank ist schnell für diesen Anwendungsfall erweitert. Mit einer batteriebetriebenen Kamera, einem LED-Licht und etwas Programmierung kannst du bald via Smartphone-App in deinen Kühlschrank gucken.

Das soll zeigen, was dir alles an Möglichkeiten offensteht, wenn du die Programmierung als ein Werkzeug verstehst und dich dessen bemächtigst.

1.2 Wie kann mir dieses Buch dabei helfen?

Vor den Erfolg haben die Gottheiten allerdings den Schweiß gesetzt. Diese Tatsache hat der griechische Dichter und Geschichtsschreiber *Hesiod* bereits vor langer Zeit festgestellt und dann so zutreffend formuliert. Dieser Ausspruch trifft unseres Erachtens kaum besser auf etwas zu als die Programmierung. Es gehört eine ordentliche Portion Arbeit dazu, bis der Groschen fällt und man die wesentlichen Programmierkonzepte verstanden hat. Erst dann wird man in der Lage sein, Aufgabenstellungen jeglicher Couleur selbstständig angehen und erfolgreich bewältigen zu können. Wir wollen dir diese notwendigen Mühen nicht verschweigen. Unserer Erfahrung nach scheitert so mancher Einstieg genau an dieser Hürde.

Unser Ansatz ist deshalb, durch **viele spannende Programmieraufgaben** das nötige Material zum Trainieren bereitzustellen. Du wirst in diesem Buch im Wesentlichen Aufgabenstellungen von uns bekommen, an denen du Aufgabe für Aufgabe alle relevanten Programmierkonzepte üben kannst. Damit dir dabei nicht die Laune vergeht, haben wir uns viel Mühe beim Zusammenstellen der Aufgaben gegeben. Es wird deutlich über die meisten „Lehrbuchaufgaben“ hinausgehen und sich, soweit möglich, erheblich näher an praktischen Anwendungsszenarien orientieren. Der klassische Lehrbuchstil hangelt sich meist an Aufgabenstellungen aus der Mathematik entlang. Das ist wichtig, und daher haben auch wir das ab und an mit dir vor. Wir können aber auch verstehen, wenn derartige Aufgaben nicht allen liegen, um etwas Neues zu lernen. Daher programmierst du eher Anwendungen, die etwas Nützliches tun oder etwas hübsch Anzuschauendes generieren. Als Appetitanreger haben wir in der nachfolgenden Bild 1.1 schon mal drei Beispiele aus dem Buch für dich.

Diese drei Bilder zeigen exemplarisch, was du mit unserem Trainingsprogramm programmieren sollst und auch können wirst, wenn du fleißig am Ball bleibst. Es lohnt sich!

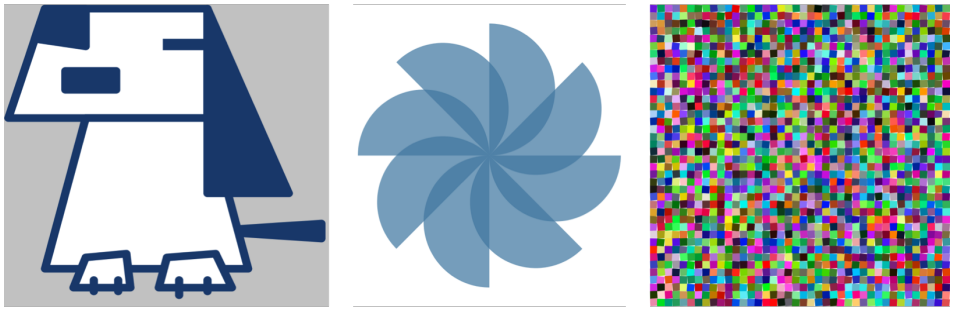


Bild 1.1 Drei Beispielbilder, die du programmieren wirst

1.3 Was muss ich mitbringen?

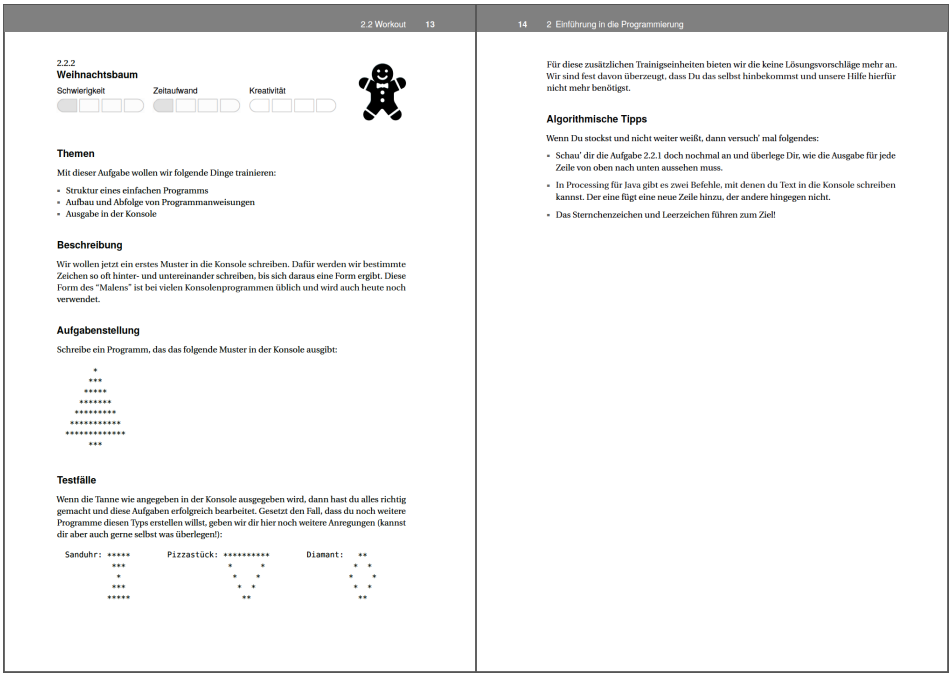
Nicht viel! Mit diesem Buch können wir es nicht leisten, dir die Basics beizubringen. Das musst du halt selbst tun, oder du bekommst es in irgendeiner Form gezeigt. Wir erklären zu Beginn eines jeden Kapitels noch mal kurz die im Fokus stehenden Übungsschwerpunkte. Das ist mehr eine Gedächtnisstütze und soll als Warm-up dienen, falls du es überhaupt brauchst. Wir gehen dabei nochmals kurz auf die wesentlichen Konzepte ein und erläutern Besonderheiten in den Programmierumgebungen, für die wir Beispiellösungen bereitstellen.

1.4 Wie geht das vonstatten?

Wie schon gesagt, ist das hier ein Trainingsbuch fürs Programmieren. Wir stellen dir **ein großes Angebot an Übungsaufgaben mit Lösungsvorschlägen** zum Training bereit. Die grundlegende Struktur gleicht dabei derer gängiger Ressourcen zur Einführung in die Programmierung. Es geht mit dem Aufbau erster einfacher Programme los und wird durch das Hinzukommen von Programmierkonzepten wie Variablen, Datentypen, Operatoren, Ausdrücken, bedingten Anweisungen, Wiederholungsanweisungen, Funktionen, Arrays, Strings bis hin zur Objektorientierung stetig erweitert. Damit wir uns – ohne unnötiges Störfeuer und Ablenkung – auf das Kernthema des jeweiligen Kapitels konzentrieren können, bestehen die ersten Programme der Kapitel 2, 3 und 4 zunächst aus Anweisungen. Erst in den nachfolgenden Kapiteln 5 bis 9 kommen dann Strukturelemente für den Programmcode in Form von Funktionen sowie Klassen und Objekten hinzu. Wir werden dir Kapitel für Kapitel Trainingsaufgaben stellen, für die du dir ein passendes Programm überlegen und dieses dann in einer Programmiersprache vollständig angeben sollst.

Jede Trainingsaufgabe ist nach einem **festen Schema** aufgebaut (siehe Bild 1.2). Um jede Aufgabe eindeutig identifizieren zu können, haben wir diese mit einem eindeutigen Namen, einer eindeutigen Nummer und einem Piktogramm versehen. Das wird dir insbesondere dabei helfen, dich mit Bezugspersonen, Leuten aus der Schule, Mitstudierenden, dem Arbeitsteam oder der Community über die Aufgaben auszutauschen. Auch unsere Lösungsvorschläge im Anhang des E-Books und online wirst du auf diese Weise spielend der jeweiligen Aufgabe zuordnen können. Die Identifizierungsnummer ist dem jeweiligen

Buchkapitel zugeordnet. Das einfache Zurückspringen zur Aufgabenstellung im Buch ist damit auch gewährleistet.



2.2 Workout 13

2.2.2
Weihnachtsbaum

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe in der Konsole

Beschreibung

Wir wollen jetzt ein erstes Muster in die Konsole schreiben. Dafür werden wir bestimmte Zeichen so oft hinter- und untereinander schreiben, bis sich daraus eine Form ergibt. Diese Form des "Malens" ist bei vielen Konsolenprogrammen üblich und wird auch heute noch verwendet.

Aufgabenstellung

Schreibe ein Programm, das das folgende Muster in der Konsole ausgibt:

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

Testfälle

Wenn die Tanne wie angegeben in der Konsole ausgegeben wird, dann hast du alles richtig gemacht und diese Aufgaben erfolgreich bearbeitet. Gesetzt den Fall, dass du noch weitere Programme diesen Typs erstellen willst, geben wir dir hier noch weitere Anregungen (kannst dir aber auch gerne selbst was überlegen!):

```
Sanduhr: *****      Pizzastück: **********      Diamant:   **
          ***                   *  *                    *  *
           *                   *    *                  *    *
          ***                   *  *                    *  *
         *****                   **                      **
```

14 2 Einführung in die Programmierung

Für diese zusätzlichen Trainigseinheiten bieten wir die keine Lösungsvorschläge mehr an. Wir sind fest davon überzeugt, dass Du das selbst hinbekommst und unsere Hilfe hierfür nicht mehr benötigst.

Algorithmische Tipps

Wenn Du stockst und nicht weiter weißt, dann versuch' mal folgendes:

- Schau' dir die Aufgabe 2.2.1 doch nochmal an und überlege Dir, wie die Ausgabe für jede Zeile von oben nach unten aussehen muss.
- In Processing für Java gibt es zwei Befehle, mit denen du Text in die Konsole schreiben kannst. Der eine fügt eine neue Zeile hinzu, der andere hingegen nicht.
- Das Sternchenzeichen und Leerzeichen führen zum Ziel!

Bild 1.2 Exemplarische Darstellung des Aufgaben-Templates

Die Aufgaben haben wir nach unserem Dafürhalten in den Kategorien **Schwierigkeit**, **Kreativität** und **Zeitaufwand** bewertet und sortiert. Einfachere Aufgaben, die im Vergleich eher wenig Zeit und kreative Eigenleistung erfordern, findest du meist am Anfang eines jeden Kapitels. Du bist aber völlig frei in deiner Entscheidung, welcher Aufgabe du dich in welcher Reihenfolge widmen möchtest. Du musst auch längst nicht alle Aufgaben durchackern. Wenn der Groschen in Bezug auf die in einem Kapitel fokussierten Programmierkonzepte gefallen ist, kann es ans nächste Kapitel gehen.

Wir haben schon eingangs gesagt, dass wir mit dir und den Aufgaben eines jeden Kapitels bestimmte Programmierkonzepte trainieren möchten. Auf welche **Themen** sich eine jeweilige Aufgabe dabei im Besonderen fokussiert, geben wir dir kurz stichwortartig an. Dies soll ein zusätzliches Kriterium sein, wonach du entscheiden kannst, ob du eine bestimmte Aufgabe bearbeiten möchtest.

Worum es sich in einer Aufgabe dreht, erläutert eine kurze **Beschreibung**. Hier wird der Kontext gesetzt und gegebenenfalls auch ein Praxisbezug hergestellt, damit du weißt, wo sich in der Praxis etwas Derartiges finden lässt. Was du dann tatsächlich tun musst, geben wir dir im anschließenden Abschnitt mit dem passenden Namen **Aufgabenstellung** an. Du solltest beide Abschnitte aufmerksam und eventuell mehrmals lesen, um sicherzustellen, dass du deinen Auftrag richtig verstanden hast.

Zur Überprüfung, ob dein Programm tatsächlich funktioniert, geben wir dir **Testfälle** mit entsprechenden Testdaten an die Hand. Zudem haben wir zu jeder Aufgabe eine ausführlich

kommentierte Lösung in den Programmiersprachen Java und Python beigefügt. Bevor du dir die aber anguckst, solltest du wirklich lange Zeit selbst an der Erarbeitung einer Lösung werkeln.

Wenn du völlig auf dem Schlauch stehst und gar keinen Zugang findest, sind am Ende einer jeden Aufgabe **algorithmische Tipps** aufgelistet, die dich einem möglichen Lösungsansatz näherbringen sollen. Bitte nutze diese Tipps und versuche unbedingt, selbst eine Lösung herzustellen, bevor du dir unsere Lösungsvorschläge im Anhang des E-Books oder online anschaust. Nur auf diese Weise kommst du genügend ins Schwitzen, um nachher wirklich behaupten zu können, das Programmieren auch selbstständig zu beherrschen.

1.5 Was muss ich sonst noch wissen?

Damit du möglichst effektiv und fokussiert trainieren kannst, haben wir die Trainingsumgebung für dich von unnötigem Ballast entschlackt. Du sollst nicht schon bei der Installation, Konfiguration und Verwendung der Programmierumgebung die Lust am Programmieren bzw. die Sicht auf das Wesentliche verlieren. Wir stützen uns daher auf ein einziges Werkzeug, mit dem du in **Java und Python** das Programmieren trainieren kannst. Es handelt sich dabei um das frei und kostenlos verfügbare Tool mit dem Namen **Processing**, das du unter der Webadresse https://processing.org/ abrufen kannst. Hier findest du auch viele weitere Informationen und Dokumentationen rund um Processing. Im Anhang A haben wir Installationsanleitungen für die gängigen Betriebssysteme Windows, macOS und Linux beigefügt.

Dass wir auf Processing abstellen, soll aber nicht heißen, dass du mit dem Erlernten in der Praxis nicht viel anfangen kannst. Ganz im Gegenteil! Die Programmiersprachen Java und Python gehören zu den am weitesten verbreiteten Sprachen, und schließlich kommt es im Wesentlichen auf die Programmierkonzepte an. Wenn du diese intensiv trainiert und dadurch verinnerlicht hast, dann bist du bereit, alle möglichen Aufgabenstellungen programmatisch zu lösen. Dann haben wir unser gemeinsames Ziel erreicht. Die Verwendung professionellerer Programmierumgebungen wie z. B. Eclipse, IntelliJ, PyCharm oder Visual Studio Code ist dann ein Klacks. Darüber müssen wir dann nicht mehr viele Worte verlieren. Wenn du so gar keine Lust auf Processing haben solltest und unsere Aufgaben lieber mit anderen Programmierumgebungen für Java oder Python bearbeiten möchtest, dann findest du im Anhang B Anleitungen für den Aus- und Umstieg.

Die Quelltexte zum Buch – unsere Lösungsvorschläge – haben wir auf den Onlinediensten **GitHub** und **Hanser Plus** für dich bereitgelegt. Du findest sie unter den Adressen https://github.com/protrain und https://plus.hanser-fachbuch.de. Wie du damit umgehst, erklären wir für alle gängigen Desktop-Betriebssysteme ab Anhang C.1 (für Java) bzw. ab Anhang D.1 (für Python).

Mit dem Kauf des Buches soll aber noch nicht Schluss sein. Wir würden uns sehr freuen, von dir zu hören. Über GitHub kannst du uns z. B. auf Fehler im Buch oder in den Lösungen aufmerksam machen. Wir tragen das dann in die Errata-Liste ein bzw. korrigieren die Programme. Außerdem kannst du uns auch deine Lösung(en) bereitstellen. Wenn diese einen eigenen Lösungsweg zeigen, nehmen wir sie mit in das Repository auf. Selbiges gilt

für Lösungen in anderen Programmiersprachen. Achte aber bitte hierbei darauf, dass es für die Sprache eine ähnlich einfache und umfangreiche Programmierumgebung gibt, wie es Processing für Java und Python ist. Wenn du sonstige Anregungen zur Verbesserung hast oder Ideen für weitere Aufgaben beisteuern möchtest, freuen wir uns, von dir zu hören.

Hoffentlich konnten wir dein Interesse wecken und dir unseren Ansatz zum Programmierenlernen schmackhaft machen. Jedenfalls würden wir uns sehr freuen, gemeinsam mit dir das Programmieren zu trainieren. Und wenn dir die mehr als 150 Aufgaben in diesem Buch nicht ausreichen sollten, möchten wir dich auf einige andere Ressourcen aufmerksam machen, die einen ähnlichen Ansatz wie wir verfolgen:

- *Advent of Code* (https://adventofcode.com/): Seit 2015 veröffentlicht der „Advent of Code" Programmieraufgaben, um die Adventszeit zu versüßen. Die Aufgaben sind oft algorithmisch anspruchsvoll und erfordern kreative Lösungsansätze. Du kannst die Aufgaben in der Programmiersprache deiner Wahl lösen. Dabei geht es nicht nur darum, die richtige Lösung zu finden (wie in diesem Buch), sondern auch um Effizienz und Eleganz des Codes. Es sind noch alle Aufgaben vergangener Adventskalender online verfügbar.
- *rustlings* (https://github.com/rust-lang/rustlings): Mit „rustlings" kannst du spielerisch und praxisnah deine Kenntnisse und Fähigkeiten in der Programmiersprache Rust verbessern. Es bietet interaktive Übungen und Aufgaben, um verschiedene Aspekte von Rust abzudecken. Du kannst den Code bearbeiten, Lösungen testen und Hinweise nutzen, um dein Verständnis zu vertiefen. Egal, ob du Anfänger oder erfahrener Entwickler bist, rustlings ist ein wertvolles Lernwerkzeug, um deine Kenntnisse in Rust zu erweitern und dich mit Best Practices vertraut zu machen.

1.6 Wie kann und sollte ich ChatGPT & Co verwenden?

Künstliche Intelligenz ist in aller Munde. Du wirst z. B. ChatGPT kennen und es gegebenenfalls auch schon verwendet haben. Wenn nicht, dann hier einige kurze Erläuterung dazu: **ChatGPT** ist ein intelligentes Sprachmodell, das entwickelt wurde, um dir in Konversationen zu helfen. Du kannst es nutzen, um Fragen zu stellen, Informationen anzufordern oder einfach nur eine Unterhaltung zu führen. Es basiert auf umfangreichem Training mit Textdaten und verwendet statistische Muster, um dir Antworten zu geben. Bitte beachte jedoch, dass es manchmal ungenaue oder falsche Antworten geben kann. Es ist wichtig, klare Fragen zu stellen und die erhaltenen Antworten kritisch zu überprüfen. ChatGPT ist ein nützliches Werkzeug für natürlichsprachliche Unterstützung, aber vergiss nicht, die Ergebnisse immer zu hinterfragen, um sicherzustellen, dass sie angemessen und korrekt sind.

Warum ist das für dieses Buch und die Programmierung relevant? Nun, weil Du ChatGPT und andere, stärker auf die Programmierung spezialisierte Sprachmodelle wie z. B. Codex, DeepCode oder CoPilot auch dazu verwenden kannst, dir Programmcodes generieren zu lassen. Du könntest also auf die Idee kommen, ChatGPT & Co für die Lösung der Aufgaben im Buch heranzuziehen. Damit betrügst du dich aber nur selbst. Du bekommst immer eine Antwort von diesen Systemen, und auf den ersten Blick mag die plausibel aussehen. Ob das aber funktionierende Programmcodes sind, die die Aufgabenstellung vollständig erfüllen,

ist keinesfalls sicher. Hier ist die programmierende Person weiterhin selbst gefordert, dies sicherzustellen. Ein Grund mehr, das Programmieren nachhaltig durch geeignetes Training zu erlernen. Sonst wirst du die generierten Codes nicht beurteilen und gegebenenfalls verbessern können. Also lass dir nicht stupide Programmcodes zu den Aufgaben generieren und vorsetzen, sondern nutze diese Tools vielmehr als Unterstützung in deinem Lernprozess. So könntest du dir z. B. deine eigenen Codes erklären lassen und so vielleicht Fehler selbst finden. Oder du könntest dir die Beispiellösungen erklären lassen, die du nicht selbst nachvollziehen kannst. Wenn das dann immer noch nicht hilft, frage gerne uns!

2 Einführung in die Programmierung

2.1 Warm-up

Dein Training beginnt in diesem Kapitel mit ersten einfachen Programmen. Dazu musst du wissen, wie der grundlegende Aufbau eines Programms sowie der Aufbau der Anweisungen in einer bestimmten Programmiersprache sind. Letzteres gehört zur sogenannten **Syntax** einer Programmiersprache. So wie z. B. die Syntax einer natürlichen Sprache Prinzipien und Regeln des Wort- und Satzbaus festlegt, so legt die Syntax einer Programmiersprache das Vokabular und den Aufbau von Anweisungen fest.

Für die allerersten Programme, die du entwickeln sollst, genügt zunächst die allereinfachste Struktur überhaupt. Hierbei werden Programme als eine lineare Abfolge von Anweisungen angegeben. Anweisungen verfügen immer über einen Namen und eine Liste von Parametern, die die Anweisung verarbeiten soll. Um den Anweisungsnamen von der Parameterliste unterscheiden zu können, werden die Parameter häufig eingeklammert und dem Anweisungsnamen nachgestellt.

```
nameAnweisung(parameter);
```

Verfügt die Parameterliste über mehrere Einträge, so werden diese mit Komma (`,`) voneinander getrennt.

```
nameAnweisung(parameter1, parameter2);
```

Parameterlose Anweisungen sind durch ein leeres Klammernpaar gekennzeichnet.

```
nameAnweisung();
```

Um mehrere Anweisungen voneinander unterscheiden zu können, wird dafür ein Trennzeichen in der Syntax einer Programmiersprache festgelegt. In Java ist das das Semikolon (`;`). Das folgende Beispiel zeigt ein abstraktes Programm, das sich aus sieben Anweisungen zusammensetzt, die in der angegebenen Reihenfolge ausgeführt werden. Die lineare Programmabfolge führt die programmierten Anweisungen zeilenweise von links nach rechts beginnend mit der obersten Zeile aus.

```
Anweisung1(); Anweisung2(); Anweisung3(); Anweisung4(); Anweisung5();
Anweisung6(); Anweisung7();
```

Durch diese Syntaxregel können die einzelnen Anweisungen separiert werden, unabhängig davon, wie du diese in die Quelltextdatei schreibst. Zur besseren Lesbarkeit empfehlen wir dir aber, dich auf eine Anweisung pro Zeile zu beschränken und die Anweisungen untereinander zu schreiben.

```
Anweisung1();
Anweisung2();
Anweisung3();
Anweisung4();
Anweisung5();
Anweisung6();
Anweisung7();
```

Die Programmiersprache Python legt in ihrer Syntax als Trennzeichen von Anweisungen den Zeilenumbruch fest. Ein Zeilenumbruch kann je nach Betriebssystem aus einem oder zwei Zeichen bestehen (`'\n'`, `'\r'` oder `'\r\n'`).

In der Programmierliteratur hat sich das `"Hello World!"`-Programm als einführendes Beispiel zur Darstellung der grundlegenden Syntax eines einfachen Programms in einer bestimmten Programmiersprache etabliert. Das Hello-World-Programm gibt in der Konsole einen einfachen Text aus, nämlich `Hello World!`. Wir wollen es zur Konkretisierung der einführenden Erläuterungen verwenden.

Java:

```
print("Hallo␣Welt!");
```

Python:

```
print("Hallo␣Welt!")
```

Die `print()`-Anweisung bekommt einen Parameter übergeben. Dieser enthält den Text, den die Anweisung in der Konsole ausgeben soll. Um den Text eingrenzen zu können, wird dieser von doppelten Anführungszeichen (`"`) eingerahmt.

Die Aufgaben dieses Kapitels drehen sich um derartige Programme. Deine Aufgabe wird es sein, die zur Lösung der Aufgabenstellung benötigten Anweisungen zu identifizieren und diese dann in einer geeigneten Abfolge zu platzieren. Welche Anweisungen eine Programmiersprache im Standardumfang bereitstellt, sind in der Referenzdokumentation aufgeführt. Die Referenz der von Processing bereitgestellten Anweisungen kann im Internet eingesehen werden:

- https://processing.org/reference/ (Java)
- http://py.processing.org/reference/ (Python)

Referenzen sind sehr umfangreich. Dies gilt auch für die von Processing. Es kann daher etwas dauern, bist du dich darin zurechtfindest. Für die in diesem Kapitel bereitgestellten Trainingsaufgaben sind insbesondere Funktionen zur Ausgabe von Texten in der Konsole und Funktionen zur Ausgabe elementarer geometrischer Formen im grafischen Ausgabefenster wichtig. Um dir das Auffinden dieser Anweisungen zu erleichtern, führen wir dir in der nachfolgenden Auflistung die relevanten auf.

- Konsolenausgabe
  - https://processing.org/reference/print_.html (Java)
  - http://py.processing.org/reference/print.html (Python)
- Linie
  - https://processing.org/reference/line_.html (Java)
  - http://py.processing.org/reference/line.html (Python)

- Dreieck
  - https://processing.org/reference/triangle_.html (Java)
  - http://py.processing.org/reference/triangle.html (Python)
- Rechteck
  - https://processing.org/reference/rect_.html (Java)
  - http://py.processing.org/reference/rect.html (Python)
- Viereck
  - https://processing.org/reference/quad_.html (Java)
  - http://py.processing.org/reference/quad.html (Python)
- Ellipse
  - https://processing.org/reference/ellipse_.html (Java)
  - http://py.processing.org/reference/ellipse.html (Python)
- Kreisausschnitt
  - https://processing.org/reference/arc_.html (Java)
  - http://py.processing.org/reference/arc.html (Python)

Um sich mit der Funktionsweise der Anweisungen vertraut zu machen, empfehlen wir dir, die Beschreibung in der Referenz aufmerksam zu lesen. Dies ist eine wichtige Grundfertigkeit, die zum Programmieren dazugehört.

Verwenden werden wir in diesem Buch die Entwicklungsumgebung Processing. Hiermit können wir Programme sowohl in Java als auch in Python schreiben. Processing bietet nicht nur den Vorteil der einfachen Installation auf nahezu allen Betriebssystemen. Wir können damit auch sehr einfach (grafische) Programme auf Basis von Anweisungen schreiben. Aber auch höherwertige Konzepte, wie wir sie in den späteren Kapiteln umsetzen werden, sind in Processing möglich; perfekte Voraussetzungen also zum Trainieren deiner Programmiertechniken mit diesem Buch.

Alle Installationsschritte von Processing findest du in Abschnitt A.1. Wie du an die digitalen Quelltexte unserer Lösungsvorschläge zu einzelnen Aufgaben kommst und wie du sie in Processing öffnest, steht im Anhang C.1.1 für Java und im Anhang D.1.1 für Python.

Dateien mit Quelltext können wir in Processing mit Klick auf *Datei → Öffnen …* laden. In Bild 2.1 haben wir zum Beispiel eine solche Datei geöffnet. Dort können wir gut die grafische Bedienoberfläche von Processing erkennen:

- Mit dem Start- und Stopp-Button *(1)* kannst du deinen Java- bzw. Python-Code ausführen.
- Um vom Java- auf den Python-Modus zu wechseln, kannst du den Modus-Auswahlreiter *(2)* verwenden. Wie du den Python-Modus in Processing installierst, steht in Anhang A.5. Links neben diesem Button ist der integrierte Debugger, den du zur Analyse von Java-Code verwenden kannst. Mehr dazu findest du in Anhang C.1.4.
- In der Mitte der Bedienoberfläche *(3)* steht der eigentliche Quelltext. In diesen Bereich kannst du deinen Java- bzw. Python-Code hineinschreiben.
- Entsprechende Ausgaben in der Konsole findest du im darunterliegenden Bereich *(4)*. Hier werden auch auftretende Fehler im Code angezeigt, sofern es welche gibt.

Nach der Einrichtung von Processing und dem Lesen der Einführung solltest du für dieses Kapitel ausgerüstet sein. In dem Sinne: Viel Spaß bei den ersten Aufgaben, und ran an die Workouts!

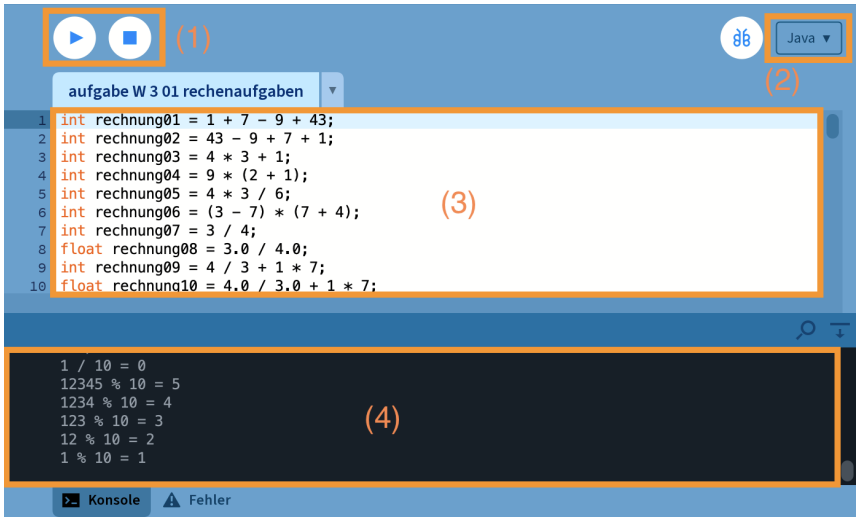
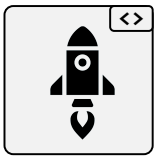


Bild 2.1 So sieht die grafische Bedienoberfläche von Processing aus.

2.2 Workout

W.2.1 Three-Two-One – Mein erstes Programm



Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau von Programmanweisungen
- Ausgabe in der Konsole

Beschreibung

Wir wollen ein erstes Programm schreiben. Der Klassiker hierfür ist die Ausgabe eines Texts – meist der Text `Hello World` – in der Konsole. Dazu braucht es in der Regel nur eine einzige Anweisung. An dieser kannst du aber bereits den Aufbau von Anweisungen und einfachen Programmen nachvollziehen und trainieren. Los geht's!

Aufgabenstellung

Schreibe ein Programm, das den Text `Three-Two-One - Takeoff!` in der Konsole ausgibt. Wenn dein Programm funktioniert, solltest du den angegebenen Text in der Konsole lesen können, so wie nachfolgend exemplarisch zu sehen ist:

```
Three-Two-One - Takeoff!
```

Wenn das geklappt hat, dann mach doch einfach weiter und modifiziere dein erstes Programm nach deinen Wünschen. Ändere z. B. den Text oder füge weitere Anweisungen zur Textausgabe hinzu. Reflektiere dabei, wie dein Programm auf die Änderungen reagiert. Wenn du das Resultat hast kommen sehen, und es ist nichts Unerwartetes bei der Ausführung deines Programms passiert, hast du es im Griff und verstanden, wie Anweisungen und einfache Programme aufgebaut sind.

Testfälle

Zum Testen deines Programms brauchst du in diesem Fall noch keine Testdaten. Starte dein Programm und prüfe, ob die geforderte Ausgabe in der Konsole ausgegeben wird.

(Algorithmische) Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Gib nicht auf. Du solltest es so lange probieren, bis es klappt. Das nennt man *Trial and Error* (Versuch und Irrtum). Versuch es weiter! Vermutlich bist du schon nah dran an der Lösung, denn der Fehler liegt sehr häufig im Detail.
- Wir benötigen eine passende Anweisung, die uns die Programmiersprache zur Ausgabe von Daten in der Konsole bereitstellt. Wie lautet diese?
- Anweisungen folgen einem festgelegten Aufbau. Hier schleichen sich schon mal Tippfehler ein. Was sagen denn die Fehlermeldungen, wenn du versuchst, dein Programm zu starten?

W.2.2 Weihnachtsbaum

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe in der Konsole

Beschreibung

Wir wollen jetzt ein erstes Muster in die Konsole schreiben. Dafür werden wir bestimmte Zeichen so oft hinter- und untereinander schreiben, bis sich daraus eine Form ergibt. Diese Form des „Malens“ ist bei vielen Konsolenprogrammen üblich und wird auch heute noch verwendet.

Aufgabenstellung

Schreibe ein Programm, das das folgende Muster in der Konsole ausgibt:

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

Testfälle

Wenn die Tanne wie angegeben in der Konsole ausgegeben wird, dann hast du alles richtig gemacht und diese Aufgaben erfolgreich bearbeitet. Gesetzt den Fall, dass du noch weitere Programme dieses Typs erstellen willst, geben wir dir hier noch weitere Anregungen (du kannst dir aber auch gerne selbst was überlegen!):

```
Sanduhr: *****        Pizzastück: **********       Diamant:   **
          ***                     *      *                  *  *
           *                       *    *                  *    *
          ***                       *  *                    *  *
         *****                       **                      **
```

Für diese zusätzlichen Trainingseinheiten bieten wir dir keine Lösungsvorschläge mehr an. Wir sind fest davon überzeugt, dass du das selbst hinbekommst und unsere Hilfe hierfür nicht mehr benötigst.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Schau’ dir die allererste Aufgabe dieses Buches doch noch einmal an und überlege dir, wie die Ausgabe für jede Zeile von oben nach unten aussehen muss.
- In Processing für Java gibt es zwei Befehle, mit denen du Text in die Konsole schreiben kannst. Der eine fügt eine neue Zeile hinzu, der andere hingegen nicht.
- Das Sternchen- und das Leerzeichen führen zum Ziel!

W.2.3 Haus

Schwierigkeit

Zeitaufwand

Kreativität

Themen

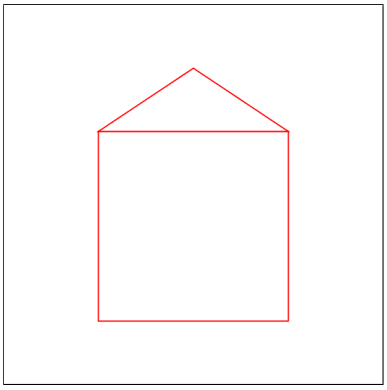
Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

Wir wollen uns ein virtuelles Haus bauen. Dafür haben wir ein 600×600 Pixel großes Fenster zur Verfügung gestellt bekommen.

Das Dach ist 300 Pixel breit und 100 Pixel hoch. Der Grundbau darunter ist 300 Pixel breit und 300 Pixel hoch.



Aufgabenstellung

Programmiere das angegebene Bild mithilfe der Processing-Grundelemente.

Testfälle

Siehe Aufgabenstellung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Schau' dir die Aufgabe zu den Grundelementen an.
- Das Bild hat ein Rechteck und ein Dreieck.
- Die Füllfarbe ist weiß und die Strichfarbe ist rot.

W.2.4 Perlenkette

Schwierigkeit

Zeitaufwand

Kreativität

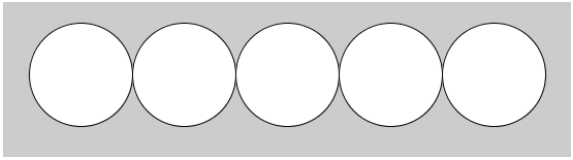
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

In dieser Aufgabe wollen wir die unten dargestellte Perlenkette programmieren:



Die Kette besteht aus fünf Perlen, die als Kreise mit schwarzer Linie und weißer Füllung dargestellt sind.

Aufgabenstellung

Programmiere das angegebene Bild mithilfe der grafischen Grundelemente von Processing.

Testfälle

Wenn die geforderten Grundformen in Art, Größe, Farbe und Lage wie in der Aufgabenstellung gefordert gezeichnet werden, dann hast du eine Lösung gefunden und die Aufgabenstellung richtig gelöst.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Alle Processing-Befehle kannst du auf der offiziellen Homepage nachlesen (Java: https://processing.org/reference/, Python: http://py.processing.org/reference/). Hier kannst du nachschauen, um die entsprechenden Befehle für das Programm zu finden.
- In Processing gibt es keine Funktion zum Zeichnen von Kreisen. Aber es gibt eine Funktion zum Malen von Ellipsen. Ab wann wird eine Ellipse zum Kreis?
- Wenn du die Ausmaße des Bildschirmfensters weißt, wo wird wohl die Mitte des Bildschirmfensters liegen?

W.2.5 Die erste Zeichnung

Schwierigkeit

Zeitaufwand

Kreativität

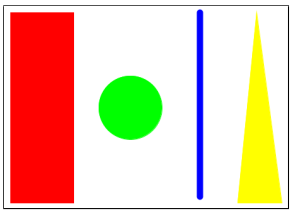
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

In dieser Aufgabe wollen wir die Processing-Grundelemente besser kennenlernen. Dazu wollen wir folgende Grafik programmieren:



Die Grafik hat unter anderem folgende Eigenschaften:

- Fenstergröße: 450 Pixel breit und 320 Pixel hoch
- Rechteck:
  - x-Position: 10
  - y-Position: 10
  - Größe: 100 Pixel breit und 300 Pixel hoch
  - Farbe: rot
- Kreis:
  - x-Position: 200
  - y-Position: Mitte des Bildschirmfensters
  - Radius: 100 Pixel
  - Farbe: grün
- Linie:
  - Breite: 10
  - Start: 310 (x), 10 (y)
  - Ziel: 310 (x), 300 (y)
  - Farbe: blau
- Dreieck:
  - Eckpunkte:
    - 400 (x), 10 (y)
    - 370 (x), 310 (y)
    - 440 (x), 310 (y)
    - Farbe: gelb

Aufgabenstellung

Programmiere das angegebene Bild mithilfe der grafischen Grundelemente von Processing.

Testfälle

Wenn die geforderten Grundformen in Art, Größe, Farbe und Lage wie in der Aufgabenstellung gefordert gezeichnet werden, dann hast du die Lösung gefunden und umgesetzt.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Alle Processing-Befehle kannst du auf der offiziellen Homepage nachlesen (Java: https://processing.org/reference/, Python: http://py.processing.org/reference/). Hier kannst du nachschauen, um die entsprechenden Befehle für das Programm zu finden.
- In Processing gibt es keine Funktion zum Zeichnen von Kreisen. Aber es gibt eine Funktion zum Malen von Ellipsen. Ab wann wird eine Ellipse zum Kreis?
- Wenn du die Ausmaße des Bildschirmfensters weißt, wo wird wohl die Mitte des Bildschirmfensters liegen?

W.2.6 Nachteule

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

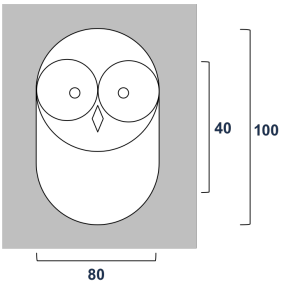
- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

In dieser Aufgabe wollen wir eine Eule nach dem folgenden Vorbild zeichnen:

Aufgabenstellung

Programmiere das angegebene Bild mithilfe der Processing-Grundelemente. Die folgenden Konstruktionsüberlegungen sollen dir dabei eine Hilfestellung bieten:



Testfälle

Wenn deine Eule grundsätzlich mit der abgebildeten Eule übereinstimmt, dann hast du die Lösung gefunden und die Aufgabe gelöst.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Bei Ellipsen und dem Spezialfall der Kreise wird immer der Mittelpunkt angegeben und nicht die linke obere Ecke.
- Achte auf die Reihenfolge!

W.2.7 Daumen

Schwierigkeit

Zeitaufwand

Kreativität

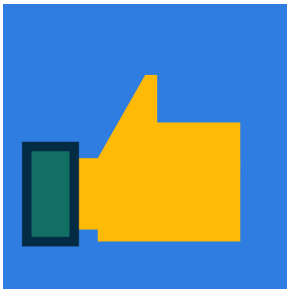
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

Wir wollen einen Daumen zeichnen. Das folgende Bild ist keine exakte Vorgabe, aber soll zeigen, wie er aussehen könnte:



Hierfür haben wir folgende Farben benutzt:

- Hintergrund: 47 (Rot), 125 (Grün), 225 (Blau)
- Daumen: 255 (Rot), 186 (Grün), 8 (Blau)
- Hemdkragen:
  - außen: 3 (Rot), 43 (Grün), 67 (Blau)
  - innen: 19 (Rot), 111 (Grün), 99 (Blau)

Aufgabenstellung

Programmiere einen Daumen wie in der oben stehenden Darstellung. Verwende zum Zeichnen die Processing-Grundelemente Rechteck und Dreieck.

Testfälle

Siehe Aufgabenstellung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Die Bestimmung der Pixelpositionen ist am einfachsten, wenn du auf kariertem Papier die Grafik einzeichnest.
- Den Hemdkragen kannst du mit nur einem Rechteck zeichnen.
- Den Daumen kannst du mit vier Rechtecken und einem Dreieck zeichnen.

W.2.8 Tasse

Neu

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

Wir wollen eine Tasse zeichnen. Das folgende Bild ist keine exakte Vorgabe, aber soll zeigen, wie sie aussehen könnte:

Hierfür haben wir folgende Farben benutzt:

- Hintergrund: 116 (Rot), 168 (Grün), 146 (Blau)
- Tasse: 251 (Rot), 242 (Grün), 196 (Blau)
- Beschriftung: 199 (Rot), 82 (Grün), 42 (Blau)
- Ränder: 125 (Grau)

Aufgabenstellung

Programmiere eine Tasse wie in der oben stehenden Darstellung. Verwende zum Zeichnen die Processing-Grundelemente.

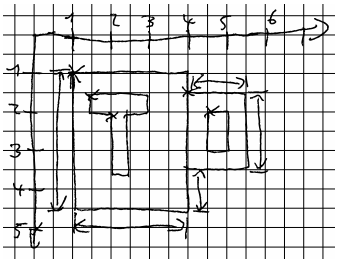
Testfälle

Siehe Aufgabenstellung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Schau dir die vorherigen Zeichenaufgaben an.
- Der Henkel der Tasse hat zwei verschiedene Füllfarben.
- Hier ein gezeichneter Hinweis:



W.2.9 Raupe Allzeitappetit

Schwierigkeit Zeitaufwand Kreativität

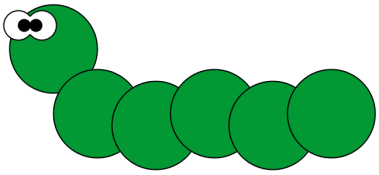
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

In dieser Aufgabe wollen wir eine Raupe zeichnen:



Aufgabenstellung

Programmiere das angegebene Bild mithilfe der Processing-Grundelemente.

Testfälle

Wenn deine Raupe grundsätzlich mit der abgebildeten Raupe übereinstimmt, dann hast du die Lösung gefunden und die Aufgabe gelöst.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Überlege dir zunächst, welche Grundelemente dieses Bild beinhaltet und wo diese platziert sind. Achte dabei auch auf eventuell „unsichtbare“ Grundelemente.
- Die Augen der Raupe bestehen entweder aus fünf (!) Kreisen oder zwei ganzen und zwei halben Kreisen. Beides ist möglich.
- Bei Ellipsen/Kreisen wird immer der Mittelpunkt angegeben und nicht die linke obere Ecke.

W.2.10 Klötzchen-Kunst

Schwierigkeit Zeitaufwand Kreativität

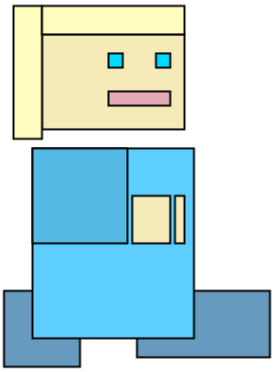
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

In dieser Aufgabe wollen wir einen Menschen aus Rechtecken programmieren:



Aufgabenstellung

Programmiere das angegebene Bild mithilfe der Processing-Grundelemente.

Testfälle

Wenn deine Klötzchen-Figur grundsätzlich mit dem abgebildeten Menschen übereinstimmt, dann hast du die Lösung gefunden und die Aufgabe gelöst.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Schaue dir die vorherigen Zeichenaufgaben noch einmal an.
- Bei der Höhe und Breite des Rechtecks kannst du auch negative Werte angeben, um das Rechteck in die umgekehrte Richtung zu zeichnen.

W.2.11 Ghettoblaster

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

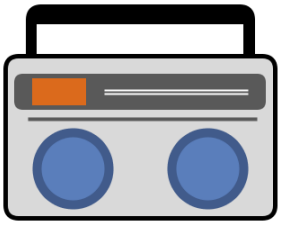
Beschreibung

Der sogenannte Ghettoblaster gilt quasi als der Vorgänger der Bluetooth-Box. Er bestand aus zwei Lautsprechern und meistens auch einem UKW-Radio, mit welchem man unterwegs Musik hören konnte.

Einen solchen Ghettoblaster wollen wir in dieser Aufgabe als Grafik realisieren.

Aufgabenstellung

Programmiere in Processing die Zeichnung eines Ghettoblasters. Er soll in dieser Form gestaltet werden:



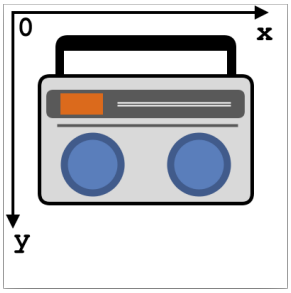
Testfälle

Siehe Aufgabenstellung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Bevor du dich an die Programmierung setzt, solltest du dir die Umsetzung überlegen. Am besten skizzierst du dir das Bild auf ein kariertes Blatt Papier. Danach zeichnest du das Koordinatensystem des Ausgabefensters ein. Wichtig hierbei ist, dass die y-Achse des Koordinatensystems von oben nach unten geht und der Nullpunkt in der linken oberen Ecke liegt:



Auf dem Blatt Papier kannst du anschließend bei allen Elementen die Höhe und Breite der einzelnen Elemente einzeichnen. Ebenso kannst du planen, wo die Koordinatenposition liegen wird.

- Nach der Planung kannst du mit der Programmierung beginnen. Hierbei wird es sehr helfen, wenn du die einzelnen Elemente deines Bildes mit entsprechenden Kommentaren versiehst. So behältst du immer den Überblick, an welcher Stelle welches Element gezeichnet wird. Das könnte in Java zum Beispiel so aussehen:

```
// Blaue Lautsprecherbox unten links
... (hier steht dann der entsprechende Code)
```

- Sollten Elemente nicht an der vermuteten Stelle gezeichnet werden: Prüfe die entsprechende Stelle im Code und schaue nach, ob sich nicht ein Gedanken- oder Tippfehler eingeschlichen hat. Probiere auch gerne verschiedene Werte in den Zeichenfunktionen aus. Das wird dir beim Verstehen der Funktionen sicher weiterhelfen.

Neu

W.2.12 Gesichtsmaske

Schwierigkeit Zeitaufwand Kreativität

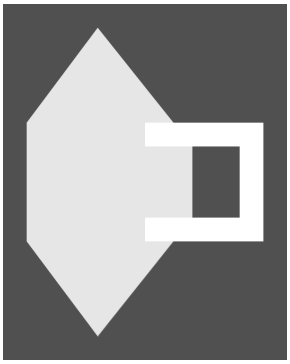
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

Wir wollen eine Gesichtsmaske auf unseren Bildschirm zeichnen. Das folgende Bild zeigt, wie sie aussehen könnte.



Aufgabenstellung

Schreibe ein Programm, welches eine Gesichtsmaske wie in der oben stehenden Darstellung ausgibt. Verwende zum Zeichnen die Processing-Grundelemente Dreieck und Rechteck.

Testfälle

Siehe Aufgabenstellung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Das Programmieren der Aufgabe könnte für dich einfacher sein, wenn du zunächst nur die Formen auf den Bildschirm zeichnest und sie danach erst einfärbst. So kannst du besser erkennen, wo noch Elemente fehlen.
- Insgesamt wären vier Dreiecke und fünf Rechtecke im Bild.

W.2.13 Hallo Bello!

Schwierigkeit Zeitaufwand Kreativität

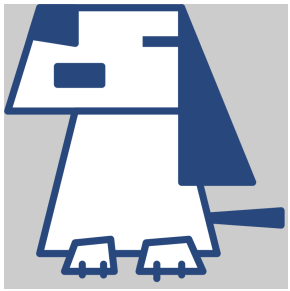
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe im grafischen Fenster

Beschreibung

Auch einen Hund können wir mit einfachen geometrischen Objekten selber programmieren. Folgendes Bild haben wir als Vorgabe bekommen:



Aufgabenstellung

Programmiere das angegebene Bild mithilfe der Processing-Grundelemente.

Testfälle

Siehe Aufgabenstellung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- In diesem Bild haben wir viele Elemente, die sich nur über die Eckpunkte beschreiben lassen (Trapez, Linien, Dreiecke). Deshalb solltest du hier besonders vor dem Programmieren die genauen Positionen der Elemente planen. Nimm dir dazu ein (kariertes) Blatt Papier und zeichne die Elemente und deren Position ein. Ist dann alles genau geplant, läuft die Programmierung wesentlich einfacher.
- Achte darauf, welches Element über welches andere Element gelegt werden soll. Dies kannst du über die Reihenfolge festlegen, mit der du die Elemente in das Ausgabefenster zeichnest.
- Es ist empfehlenswert, wenn du zunächst alle Formen einzeichnest. Stimmt die Zeichnung dann mit dem Ergebnis einigermaßen überein, kannst du die Eigenschaften der Elemente noch hinzufügen (Farbe, Liniendicke, Linienart etc.).

3 Variablen, Datentypen, Operatoren und Ausdrücke

3.1 Warm-up

Das Programm einer der vorangegangenen Aufgaben hatte zum Ziel, einen vorgegebenen Text auszugeben. Das war ziemlich statisch. Ähnlich verhält es sich mit einem Programm, welches die Fläche eines Rechtecks mit den Seitenlängen `a=3` und `b=4` berechnet. Gibt es der Person aber die Flexibilität, auch Flächen von Rechtecken mit *beliebigen* Seitenlängen zu berechnen, wird es schon interessanter.

Damit das Programm also dynamischer reagieren kann, benötigen wir etwas wie einen Platzhalter. Den kann das Programm anstatt eines konkreten Werts verwenden. In Programmen spricht man in diesem Zusammenhang von **Variablen**.

Bei einer Variablen handelt es sich um einen Speicherbereich, in den wir Datenwerte ablegen und auslesen können. Den Bezeichner, also den Namen der Variablen, spezifizieren wir unter Beachtung einiger Regeln selbst.

Da wir bei den hier behandelten Programmiersprachen keine Beschränkungen in der Länge eines Bezeichners haben, gibt uns das die Möglichkeit, *sprechende Namen* zu verwenden. Denn je treffender die Bezeichner benannt werden, umso einfacher fällt es dir, das Programm zu verstehen. Beachte aber, dass je nach verwendetem Programmiersystem ganz unterschiedliche Gesetzmäßigkeiten für die Benennung von Variablen und anderen Bezeichnern gelten.

Auch wenn wir jetzt noch ganz am Anfang stehen, solltest du dir darüber klar werden, in welcher Landessprache du die Variablen – und die später folgenden Bezeichner – schreibst. Beispielsweise verbieten einige Programmiersprachen die Verwendung von Umlauten und Sonderzeichen, ohne die ein entsprechender Bezeichner vielleicht nur umständlich oder schlecht zu begreifen ist.

Einige Programmiersprachen bieten ausgearbeitete *Naming Conventions* oder *Naming Guidelines* an, siehe z. B. http://www.oracle.com/technetwork/java/codeconventions-135099.html für Java oder https://www.python.org/dev/peps/pep-0008/#naming-conventions für Python. Darin werden Konventionen angegeben, die – durchgängig angewendet – dem Entwicklungsteam beim Erfassen eines Algorithmus helfen.

Die in diesem Buch behandelten Programmiersprachen sind beide **streng typisierte Sprachen**. Dies hat zur Folge, dass jeder Variablen eindeutig ein **Datentyp** zugeordnet wird. Der Datentyp gibt an, *was* für eine Art von Daten bzw. *wie* Daten gespeichert werden sollen. Es ist beispielsweise möglich, das Datum `4711` als eine Zahl oder eine Zeichenkette abzuspei-

chern. Außerdem legt der zugeordnete Datentyp auch den Wertebereich fest. Die möglichen Operationen auf den Daten dieses Typs werden ebenfalls durch den Datentyp definiert.

Nun existieren abhängig von dem jeweiligen Programmiersystem die unterschiedlichsten **Datentypen**. Im arithmetischen Umfeld werden verschiedene Datentypen vor allem durch ihren Wertebereich klassifiziert. Java definiert beispielsweise sechs verschiedene arithmetische Datentypen. Während eine Variable des Datentyps `byte` lediglich Zahlen von -128 bis +127 aufnehmen kann, speichert eine Variable des Datentyps `short` Werte im Bereich von -32768 bis +32767. Und auf die gleiche Weise beherbergen die nachfolgenden Datentypen immer größere Werte in der angegebenen Größenrelation:

```
byte < short < int < long < float < double
```

Sehr oft müssen wir Daten eines Datentyps in einen anderen umwandeln. In diesem Zusammenhang spricht man von der **Typumwandlung**. Es ist schnell einsehbar, dass eine Typumwandlung von einem kleineren in einen größeren Datentyp kein Problem darstellt. Dies wird auch als **implizite Typumwandlung** bezeichnet und geht automatisch vonstatten. Wollen wir dagegen in die entgegengesetzte Richtung, also von einem Datentyp mit höherem Wertebereich in einen Datentyp mit geringerem Wertebereich, konvertieren, sprechen wir von der **expliziten Typumwandlung**. Diese Umwandlung kann nur *manuell* erfolgen, da ansonsten ein Datenverlust eine mögliche Folge wäre. Dennoch kann es nötig sein, mit besonderer Vorsicht eine solche Konvertierung durchzuführen. Und zu diesem Zweck stellen Programmiersprachen wie z. B. Java einen sogenannten **Casting-Operator** zur Verfügung.

Während Python keine Deklaration der Variablen verlangt und damit der Datentyp bei der ersten Wertzuweisung vom Programmiersystem implizit festgelegt wird, fordert Java die Bekanntmachung jeder Variablen unter Angabe des zu verwendenden Datentyps *vor* der ersten Zuweisung. Hierbei spricht man auch von **expliziter Typisierung**.

Schritt	Java	Python
Deklaration	int diameter;	
Zuweisung	diameter = 80;	
In einem Schritt	int diameter = 80;	diameter = 80

In den oben aufgeführten Beispielen wird mehrfach der Zuweisungsoperator (=) verwendet. Er gilt für alle Datentypen und bewirkt die Zuweisung eines Werts an die Variable. Der **Ausdruck**, also die Verarbeitungsvorschrift, wird dementsprechend von rechts nach links abgearbeitet. Bei einem Ausdruck handelt es sich um ein Konstrukt, das einen Wert zurückliefert. Im einfachsten Fall ist ein Ausdruck ein Literal, also bereits der Wert, den der Ausdruck liefert. Ein Ausdruck kann aber auch eine Variable oder eine Funktion sein. Auch eine Verarbeitungsvorschrift, die sich aus Operanden und Operatoren zusammensetzt und einen Wert liefert, wird als *Ausdruck* bezeichnet. Hierbei können die Operanden wiederum Ausdrücke sein.

Beachte, dass es sich bei dem =-Operator nicht um die mathematische Vergleichsoperation handelt! Der Operator zum Bestimmen der Gleichheit zweier Operanden wird in beiden Programmiersprachen durch ein doppeltes Gleichheitszeichen ausgedrückt ==. Hinzu kommen noch weitere **Vergleichsoperatoren**, die in der folgenden Tabelle zu sehen sind.

Vergleich	Operator
Gleichheit	`==`
Ungleichheit	`!=`
Kleiner	`<`
Kleiner gleich	`<=`
Größer gleich	`>=`
Größer	`>`

Darüber hinaus sind dir die **arithmetischen Operatoren** `+`, `-`, `*`, `/` bekannt. Hinzu kommt noch der **Modulo-Operator** (`%`), der den ganzzahligen Rest einer Division zweier Zahlen angibt, also z. B. `5 % 3 = 2`.

Manchmal wird es nötig sein, Bedingungen zu verknüpfen. Das kann mit den sogenannten **logischen Operatoren** gemacht werden. Hier die tabellarische Übersicht:

Operand 1	Operand 2	UND	ODER	Exklusiv ODER
‘false‘	‘false‘	‘false‘	‘false‘	‘false‘
‘false‘	‘true‘	‘false‘	‘true‘	‘true‘
‘true‘	‘false‘	‘false‘	‘true‘	‘true‘
‘true‘	‘true‘	‘true‘	‘true‘	‘false‘

Zusätzlich gibt es noch die **Negation**, bei der das Ergebnis der Operation einfach umgekehrt wird.

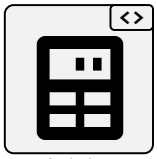
Wie bereits weiter oben beschrieben, definiert der Datentyp einer Variablen die möglichen Operatoren und damit Operationen auf diesen Daten. Der obige Abschnitt behandelt die typischen Operatoren der Arithmetik. Da wir uns aber auch anderer Datentypen (`char`, `String`) bedienen, müssen wir uns darüber im Klaren sein, dass hier gegebenenfalls andere Gesetzmäßigkeiten gelten. So gibt es bei Strings neben dem Zuweisungsoperator nur noch den +-Operator, der auch eine völlig andere Bedeutung gegenüber arithmetischen Datentypen hat: Er verknüpft zwei Zeichenketten miteinander. Operatoren besitzen also eine *Wandlungsfähigkeit* in Abhängigkeit vom Datentyp, auf den sie angewendet werden.

Und nun – ran an die Workouts, und viel Erfolg beim Training!

3.2 Workout

W.3.1 Einfache Rechenaufgaben

Schwierigkeit Zeitaufwand Kreativität



Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

In dieser Aufgabe wollen wir einfache Rechenaufgaben durchführen.

Aufgabenstellung

Schreibe ein Programm, das die bei den Testfällen stehenden Rechenaufgaben berechnet, die Ergebnisse in einer Variablen speichert und die Werte der Variablen in der Konsole ausgibt.

Testfälle

```
1 + 7 - 9 + 43 = 42
43 - 9 + 7 + 1 = 42
4 * 3 + 1 = 13
9 * (2 + 1) = 27
4 * 3 / 6 = 2
(3 - 7) * (7 + 4) = -44
3 / 4 = 0
3.0 / 4.0 = 0.75
4 / 3 + 1 * 7 = 8
4.0 / 3.0 + 1 * 7 = 8.333333
2 * 2 * 2 * 2 * 2 * 2 * 2 = 128
42 / 7 / 3 = 2
```

```
12 % 2 = 0
13 % 5 = 3
12345 / 10 = 1234
1234 / 10 = 123
123 / 10 = 12
12 / 10 = 1
1 / 10 = 0
12345 % 10 = 5
1234 % 10 = 4
123 % 10 = 3
12 % 10 = 2
1 % 10 = 1
```

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Solltest du ein Ergebnis wie `NaN` (Not a Number) auf der Konsole sehen, wurde eine Division mit der Zahl `0` durchgeführt. Anscheinend hast du hier statt einer Kommazahl eine Integer-Zahl für die Division verwendet. Achte daher darauf, dass bei der Division die korrekten Datentypen verwendet werden. Beispiel: 255 ist Integer, 255.0 ist Float.
- Wenn du Variablen mit Zahlenwerten in Python auf der Konsole ausgeben willst und der Fehler `TypeError: cannot concatenate 'str' and 'float' objects` auftaucht, musst du diese Variablen in Klammern mit einem davorstehenden **str** setzen. Beispiel: **str(Variable)**. Hiermit wandelst du die Zahl in ein String-Objekt um. Anstelle von `str` kannst du auch die Python-Methode `IrgendeinString.format(Variablen,...)` verwenden, mit der du einen String mit Platzhaltern `{}` mit Variableninhalten auffüllen kannst. Mehr zu Strings erfährst du später im Buch.

W.3.2 Perlenkette 2.0

Schwierigkeit Zeitaufwand Kreativität

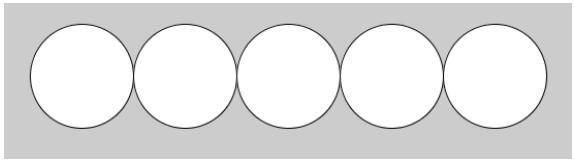
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

In dieser Aufgabe wollen wir die Perlenkette aus dem vorherigen Kapitel mit den neu hinzugekommenen Programmierkonzepten (Variablen und arithmetische Operatoren) umsetzen. Hier noch mal die angestrebte Perlenkette:



Aufgabenstellung

Programmiere die angegebene Zeichnung unter Verwendung der Processing-Grundelemente. Nutze dabei Variablen und Operatoren.

Testfälle

Wenn die Perlenkette wie in der Aufgabenstellung gefordert von deinem Programm gezeichnet wird, dann bist du schon auf einem guten Weg. Eine richtige Lösung hast du dann entwickelt, wenn dein Programm zudem mit Veränderungen der Perlengröße zurechtkommt. Wenn du diese durch eine entsprechende Variable vorgeben kannst und für verschiedene Werte immer fünf aneinandergereihte Perlen dargestellt werden, passt alles. Dann siehst du jetzt auch, wie sich durch den Einsatz von Variablen und Operatoren Programme flexibel an neue Gegebenheiten anpassen.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Speichere die Startposition (X-Koordinate), die Y-Koordinaten der Kreise sowie deren Radius in jeweils einer Variablen.
- Zeichne anschließend die Kreise auf den Bildschirm. Hierzu kannst du die Processing-Funktion **ellipse()** verwenden.
- Wenn du die Startposition und den Radius der Kreise kennst, kannst du die Position des nächsten Kreises durch das Addieren der X-Startposition mit dem Radius ermitteln.

W.3.3 Blutalkoholkonzentration

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

Mit der Widmark-Formel können wir die Blutalkoholkonzentration abschätzen. Sie wird mit dieser Formel berechnet:

$$c = \frac{A}{m * r}$$

mit

$$A = V * \epsilon * \rho$$

wobei

- c: Alkoholkonzentration im Blut in [g/kg]
- A: Aufgenommene Masse des Alkohols in [g]
- r: Verteilungsfaktor im Körper – Männer: $r \approx 0{,}7$ – Frauen: $r \approx 0{,}6$ – Kinder: $r \approx 0{,}8$
- m: Masse der Person in [kg]
- V: Volumen des Getränks in [ml]
- ϵ: Alkoholvolumenanteil in [%] (z. B. Bier $\approx 0{,}05$)
- ρ: Dichte von Alkohol [g/ml] $\rightarrow \rho \approx 0{,}8$ g/ml

Aufgabenstellung

Schreibe ein Programm, das die Blutalkoholkonzentration mit der Widmark-Formel berechnet. Die Eingabegrößen sollen dabei flexibel einstellbar sein.

Testfälle

Ein 80 kg ($m = 80$) schwerer Mann ($r \approx 0{,}7$) trinkt eine 0,5-l-Flasche Bier ($A = 500\,ml * 0{,}05 * 0{,}8\,g/ml = 20g$). Daraus ergibt sich nach der Widmark-Formel eine Blutalkoholkonzentration von ungefähr 0,35714 g/kg (≈ 0,36 Promille).

Überlege dir weitere Testfälle, die möglichst viele der möglichen Kombinationen aus Alkoholart, Geschlecht, Köpergewicht und Volumen abdecken. Hier noch eine Anregung von uns, wie das aussehen kann:

```
Bier (epsilon=0,05)      Wein (epsilon=0,12)      Scotch (epsilon=0,46)
- Mann (m=70, 80, ...)   - Mann (m=70, 80, ...)   - Mann (m=70, 80, ...)
  V=500, 1000, ...         V=200, 400, ...          V=20, 40, ...
- Frau (m=60, 70, ...)   - Frau (m=60, 70, ...)   - Frau (m=60, 70, ...)
  V=500, 1000, ...         V=200, 400, ...          V=20, 40, ...
- Kind (m=40, 50, ...)   - Kind (m=40, 50, ...)   - Kind (m=40, 50, ...)
  V=500, 1000, ...         V=200, 400, ...          V=20, 40, ...
```

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Achte bei der Programmierung auf den korrekten Datentyp bei der Berechnung.
- Für einen besseren Überblick könntest du die Variablennamen nach den entsprechenden Bezeichnungen in der Widmark-Formel verwenden. Dabei könnte es zusätzlich helfen, wenn du mit Kommentaren entsprechend dazuschreibst, wozu die einzelne Variable verwendet wird.
- Teile die Formel in mehrere einzelne Berechnungen auf, idealerweise mit dafür angelegten Variablen. So behältst du einfacher den Überblick.

W.3.4 Stoffwechselrate

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

Die Größe des Grundumsatzes (G, Kalorien pro Tag) können wir zur Charakterisierung des Stoffwechsels beim Menschen verwenden. Sie ist diejenige Energiemenge, die der Körper pro Tag bei völliger Ruhe, einer Temperatur von 28 Grad Celsius und leerem Magen zur Aufrechterhaltung seiner Funktionen benötigt.

Bereits 1918 veröffentlichten J. A. Harris und F. G. Benedict die nach ihnen benannte Harris-Benedict-Formel. Diese berechnet den Grundumsatz mithilfe von Körpergewicht (m in kg), Körpergröße (l in cm) und dem Alter (t in Jahre). Die Harris-Benedict-Formel stellt noch heute eine in der Ernährungsmedizin allgemein akzeptierte gute Näherung des gemessenen Grundumsatzes (G).

Sie lautet für Männer:

$$G = 66{,}47 + 13{,}7 * m + 5 * l - 6{,}8 * t$$

und für Frauen:

$$G = 655{,}1 + 9{,}6 * m + 1{,}8 * l - 4{,}7 * t$$

Aufgabenstellung

Schreibe ein Programm, welches den Grundumsatz nach der Harris-Benedict-Formel berechnet und die Ergebnisse für beide Geschlechter in der Konsole ausgibt.

Testfälle

- Eingabe:
  - Körpergewicht: 58 kg
  - Körpergröße: 180 cm
  - Alter: 25 Jahre
- Ergebnis:
  - Mann: 1591.07 Kalorien pro Tag
  - Frau: 1418.4 Kalorien pro Tag

- Eingabe:
  - Körpergewicht: 90 kg
  - Körpergröße: 160 cm
  - Alter: 45 Jahre
- Ergebnis:
  - Mann: 1793.47 Kalorien pro Tag
  - Frau: 1595.6 Kalorien pro Tag
- Eingabe:
  - Körpergewicht: 45 kg
  - Körpergröße: 176 cm
  - Alter: 17 Jahre
- Ergebnis:
  - Mann: 1447.37 Kalorien pro Tag
  - Frau: 1324.0 Kalorien pro Tag

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Speichere die Eingabegrößen als Variablen.
- Berechne die Formeln für Mann und Frau in unterschiedlichen Variablen und schreibe das Ergebnis in die Konsole.
- Achte bei der Berechnung auf den passenden Datentyp.

W.3.5 Baumstammvolumen

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

Nachdem ein Baum gefällt und der Stamm aufgearbeitet wurde, möchte man in der Holzwirtschaft wissen, wie viel Holz der Stamm hat (Volumen in Festmeter). Dazu messen wir die Länge (L in Meter) und den Mittendurchmesser (D in Zentimeter).

Nach folgender Formel können wir anschließend das Volumen berechnen:

$$Volumen = (\frac{\pi}{4} * D^2) * L/10000$$

Aufgabenstellung

Schreibe ein Programm, das nach der oben stehenden Formel das Volumen eines Baumstamms berechnet.

Testfälle

- 10 Meter lang und 30 cm Durchmesser: 0.70685834 Festmeter
- 15 Meter lang und 32 cm Durchmesser: 1.2063715 Festmeter

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Die Kreiszahl PI steht in Processing als Variable **PI** zur Verfügung.
- Definiere zuerst die Variablen für Länge und Durchmesser und berechne anschließend das Volumen.

W.3.6 Körperoberfläche

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

Die Körperoberfläche eines Menschen wird nach der Formel von Mosteller wie folgt errechnet:

$$KOF = \sqrt{\frac{h * w}{3600}}$$

In der angegebenen Formel bezeichnet h die Körpergröße in cm und w das Körpergewicht in kg. Das Ergebnis ist die Körperoberfläche in m^2.

Aufgabenstellung

Schreibe ein Programm, das die Körperoberfläche nach der Formel von Mosteller in m^2 berechnet.

Testfall

- 1,60 m und 55 kg → 1.5634719 m^2
- 1,80 m und 58 kg → 1.7029387 m^2
- 2,10 m und 80 kg → 2.1602468 m^2

Algorithmische Tipps

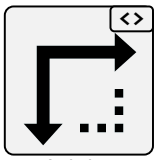
Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Zum Wurzelziehen kannst du die Processing-Methode **sqrt(float n)** nutzen.
- Achte bei Divisionen immer darauf, dass die Nenner aus Kommazahlen bestehen.
- Bei Divisionen von Integer-Werten folgt ein Integer-Ergebnis.

Neu

W.3.7 Schuhgröße

Schwierigkeit Zeitaufwand Kreativität



Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

Schuhgrößen beim fiktiven Schuhhersteller *Victoria* lassen sich für Deutschland nach der folgenden Formel berechnen:

$$S = l * 1{,}5 + 2{,}25$$

In der Formel ist S die Schuhgröße und l die Fußlänge in cm. Unser Schuhhersteller stellt allerdings nur Schuhe in ganzzahligen Größen her (also zum Beispiel 36 oder 39). Daher ignorieren wir die Nachkommastelle in diesem Fall.

Aufgabenstellung

Schreibe ein Programm, welches aus einer übergebenen Fußlänge in cm die Schuhgröße berechnet. Die Fußlänge soll dabei individuell einstellbar sein.

Der Einfachheit halber soll bei der Schuhgröße die Nachkommastelle abgeschnitten werden.

Testfälle

- 27.17 cm → 43
- 25,1675 cm → 40
- 15,2 cm → 25

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Mit welchem Datentyp könntest du die Nachkommastelle von einer Fließkommazahl abtrennen?
- Erst die Schuhgröße in einem Datentyp ausrechnen und dann in den anderen Datentyp konvertieren.

W.3.8 Haus mit Garage

Schwierigkeit

Zeitaufwand

Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

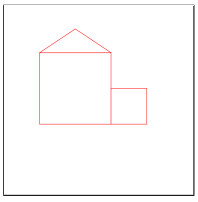
Beschreibung

Wir wollen ein Haus mit Garage malen. Dabei soll die Garage folgende Eigenschaften haben:

Eigenschaft	Wert	Berechnung
Höhe	Halbe Höhe des Hauses	hausHoehe / 2
Breite	Fünf Achtel der Hausbreite	hausBreite * 5/8
X-Position	Rechts neben dem Haus	hausX + hausBreite
Y-Position	Boden des Hauses	hausY + hausHoehe – garagenHoehe

Das Dach des Hauses ist 300 Pixel breit und 100 Pixel hoch. Der Grundbau darunter ist 300 Pixel breit und 300 Pixel hoch.

Das Ergebnis könnte dann so aussehen:



Aufgabenstellung

Male ein Haus mit Garage, wie in der Aufgabenstellung beschrieben.

Testfälle

Siehe Beschreibung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Im Kapitel davor haben wir ein Haus gemalt. Nehme diese Aufgabe als Vorlage und versuche erst einmal, die Zahlenwerte dort durch Variablen zu ersetzen (z. B. Höhe und Breite).
- Schau dir die Processing-Funktion `translate(x, y)` an. Damit kannst du den Startpunkt (Nullpunkt) des Koordinatensystems verschieben. Komplexe Überlegungen wie z. B. beim Zeichnen des Dachs oder der Garage kannst du damit dann wesentlich einfacher umsetzen.

W.3.9 RGB nach CMYK

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen
- Erstellung und Auswertung von Ausdrücken

Beschreibung

Wenn wir Farben für Computer beschreiben wollen, gibt es dafür verschiedene Formate. Bei Displays wird dafür oft das RGB-Format verwendet. Dieses beschreibt den Anteil der Farbwerte von Rot (R), Grün (G) und Blau (B). Diesen Anteil speichern wir jeweils als ganze Zahl im Zahlenbereich von 0 bis 255.

Bei Druckern wird üblicherweise das CMYK-Format verwendet. Hierbei werden die Farben durch die Farbwerte Cyan (C), Magenta (M), Gelb (Yellow, Y) und Schwarz (Key, K) beschrieben. Diese CMYK-Werte speichern wir als eine Kommazahl (reelle Zahl), die zwischen den ganzen Zahlen 0 und 1 liegt.

Mit diesen Berechnungen kannst du Farbwerte von RGB nach CMYK umwandeln:

$$C = \frac{w - \frac{r}{255}}{w}$$
$$M = \frac{w - \frac{g}{255}}{w}$$
$$Y = \frac{w - \frac{b}{255}}{w}$$
$$K = 1 - w$$

wobei

$$w = max\{\frac{r}{255}, \frac{g}{255}, \frac{b}{255}\}$$

Aufgabenstellung

Schreibe ein Programm, das einen RGB-Farbwert in einen CMYK-Farbwert umrechnet. Das Programm soll drei ganze Zahlen **r**, **g** und **b** in entsprechende Variablen speichern und die berechneten CMYK-Werte auf der Konsole ausgeben.

Testfälle

Testfall 1:

- RGB-Ausgangswerte:
  - R: 75
  - G: 0
  - B: 130
- CMYK-Zielwerte:
  - C: 0.42307693
  - M: 1.0
  - Y: 0.0

Testfall 2:

- RGB-Ausgangswerte:
  - R: 11
  - G: 11
  - B: 11
- CMYK-Zielwerte:
  - C: 0.0
  - M: 0.0
  - Y: 0.0
  - K: 0.95686275

Testfall 3:

- RGB-Ausgangswerte:
  - R: 20
  - G: 40
  - B: 60
- CMYK-Zielwerte:
  - C: 0.6666666
  - M: 0.3333333
  - Y: 0.0
  - K: 0.7647059

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Schau dir noch einmal die Aufgabe zu den einfachen Rechenaufgaben an.
- Das Maximum kannst du mit der Funktion **max(Zahl1, Zahl2, Zahl3)** berechnen.

W.3.10 Tic-Tac-Toe-Spielfeld

Schwierigkeit

Zeitaufwand

Kreativität

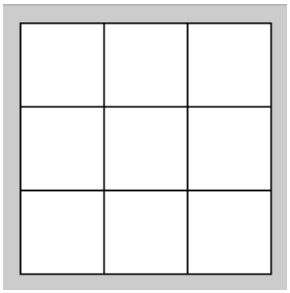
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

Wir wollen in dieser Aufgabe das Spielfeld des bekannten Spiels **Tic-Tac-Toe** grafisch darstellen. Dieses Spielfeld besteht aus neun meist quadratischen Feldern, die in einem 3-mal-3-Raster angeordnet sind:



Aufgabenstellung

Schreibe ein Programm, das ein leeres Spielfeld für Tic-Tac-Toe erzeugt und darstellt. Mache dabei Gebrauch von geeigneten Variablen!

Testfälle

Siehe Beschreibung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Überlege dir zunächst, wie die Elemente im Fenster platziert werden und welche Formen du dafür verwendest, idealerweise auf einem Blatt Papier. Erst danach solltest du mit der Programmierung anfangen.
- Rechteck oder Linie zeichnen? Beides ist möglich, aber ein Fall ist einfacher umzusetzen.
- Bei wiederholenden Zahlenwerten könntest und solltest du diese in einer Variablen speichern und im Programm verwenden.

W.3.11 Gamecontroller

Neu

Schwierigkeit

Zeitaufwand

Kreativität

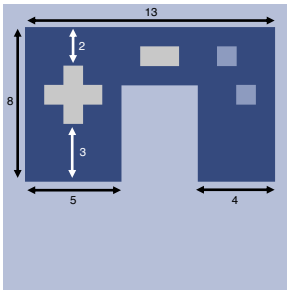
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

Wir wollen einen Gamecontroller zeichnen. Die folgende Grafik gibt uns einige Hinweise zu den einzelnen Abmessungen:



Farben:

- Hintergrund: 182 (Rot), 191 (Grün), 216 (Blau)
- Controller: 53 (Rot), 74 (Grün), 126 (Blau)
- Buttons links/mitte: 200 (Grau)
- Buttons rechts: 141 (Rot), 155 (Grün), 191 (Blau)

Aufgabenstellung

Schreibe ein Programm, welches den oben dargestellten Gamecontroller zeichnet. Die Größe des Gamecontrollers soll dabei mit einer Variablen einstellbar sein.

Testfälle

Kästchengröße 25 Pixel	Kästchengröße 40 Pixel	Kästchengröße 100 Pixel
		

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Mithilfe der Relationen solltest du den Controller bauen können.
- Du kannst den Nullpunkt vom Koordinatensystem mit `translate(x, y)` verschieben.
- Siehe auch die vorherigen Zeichenaufgaben für weitere Tipps.

W.3.12 Fußballtor

Schwierigkeit | Zeitaufwand | Kreativität

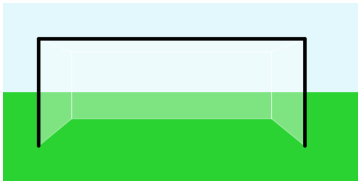
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Verwenden von Variablen und Datentypen
- Verwendung einfacher Grundrechenoperatoren
- Anpassung der Funktionsweise von Operatoren an Datentypen

Beschreibung

Wir wollen ein Fußballtor zeichnen. Dies könnte ungefähr so aussehen:



Das Tor hat dabei folgende Eigenschaften:

Eigenschaft	Wert
Höhe des Tores	300 Pixel
Breite des Tores	2.5-fache Torhöhe
Torposition	100 Pixel (X), 100 Pixel (Y)
Position des inneren Kastens*	1/8 der Torbreite (X), 1/8 der Torhöhe (Y)
Breite des inneren Kastens	3/4 der Torbreite
Höhe des inneren Kastens	5/8 der Torhöhe

* von innerhalb des Tores aus gesehen

Außerdem verwenden wir folgende Farben:

- Wiese: 42 (Rot), 211 (Grün), 50 (Blau)
- Himmel: 227 (Rot), 248 (Grün), 252 (Blau)
- Torpfosten: 0 (Rot), 0 (Grün), 0 (Blau)
- Torkasten: 255 (Rot), 255 (Grün), 255 (Blau), 100 (Alpha)

Die Torpfosten haben eine Liniendicke von 10 Pixel. Alle anderen Elemente haben eine Liniendicke von 3 Pixel. Die Fenstergröße ist 1000 x 500 Pixel.

Aufgabenstellung

Programmiere ein Fußballtor. Verwende zum Zeichnen Variablen und Operatoren sowie die Processing-Grundelemente. Gerne kannst du auch die `translate()`-Funktion verwenden.

Testfälle

Siehe Beschreibung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Der Alpha-Wert ist der vierte Parameter bei Farbwerten (u. a. auch beim Datentyp `color`). Damit kannst du die Transparenz (Durchsichtigkeit) der Farbe entsprechend einstellen.
- Male zuerst den Hintergrund, dann das Tornetz und danach die Torpfosten.
- Für die Netze am Rand bietet sich die Funktion `quad(...)` an, die eine vierseitige Fläche mit festgelegten Eckpunkten (Polygon) zeichnet.
- Falls du beim Tornetz nicht weiterkommen solltest: Probiere zunächst, die Torpfosten zu zeichnen, und füge später dann das Tornetz hinzu. Durch die bereits sichtbaren Torpfosten fällt dir die Orientierung für das Zeichnen der Netze eventuell leichter.

4 Kontrollstrukturen

4.1 Warm-up

Bis jetzt führen unsere Programme die Programmanweisungen eine nach der anderen aus. In diesem Kapitel wollen wir dem Ganzen etwas Würze verleihen, indem wir die Programme in die Lage versetzen, ihren Ablauf zu steuern. Damit ist gemeint, dass nicht grundsätzlich alle Anweisungen während der Ausführung eines Programms auch tatsächlich abgearbeitet werden sollen oder dürfen. Hierbei spricht man von **bedingten Anweisungen**, die nur unter einer bestimmten Voraussetzung abgearbeitet werden. Andere Anweisungen müssen dagegen *wiederholt* ausgeführt werden. Hierunter versteht man die sogenannten **Wiederholungsanweisungen**. Nimmt man beide Paradigmen zusammen, ist vom Einsatz von **Kontrollstrukturen** die Rede. Sie steuern die Abfolge der Ausführung von Anweisungen und geben an, in welcher Reihenfolge und wie oft Anweisungen ausgeführt werden.

Zu der Familie der **bedingten Anweisungen** gehören die `if-else`-Anweisung sowie die `switch`-Anweisung. Der große Unterschied zwischen diesen beiden bedingten Anweisungen besteht darin, dass die `if-else`-Anweisung lediglich zwei Fälle unterscheidet und damit auch nur diese beiden Fälle behandeln kann: den Teil, der ausgeführt wird, wenn die Bedingung (auch *boolescher Ausdruck* genannt) zutrifft, und den zweiten Teil, der alternativ abgearbeitet wird.

```
if(<booleschерAusdruck>) {
    // In diesen geklammerten Block kommen alle Anweisungen hinein,
    // die abgearbeitet werden, wenn die Bedingung wahr ist.
}
else {
    // In diesen geklammerten Block kommen alle Anweisungen hinein,
    // die abgearbeitet werden, wenn die Bedingung falsch ist.
}
```

Der `else`-Block ist optional. Wenn wir ihn nicht brauchen, kann er komplett entfallen.

Es kommt oft vor, dass mehr als zwei Fälle betrachtet werden müssen. Stellen wir uns beispielsweise vor, wir müssen die Lampen einer Ampel steuern. Hier haben wir drei Fälle zu unterscheiden, die wir zwar durch mehrere und vielleicht auch verschachtelte `if-else`-Konstrukte ausdrücken können, was aber schnell unübersichtlich wird. In Situationen, in denen drei oder mehr Fälle unterschieden werden, können wir in der Programmiersprache Java auf die `switch`-Anweisung zurückgreifen.

```
switch(<Ausdruck>) {
    case <Konstante_1>:
        <Anweisungsblock_1>;
        break;
    case <Konstante_2>:
        <Anweisungsblock_2>;
        break;
    ...
    case <Konstante_n>:
        <Anweisungsblock_n>;
        break;
    default:
        <Anweisungsblock_sonst>;
}
```

Besonderes Augenmerk müssen wir auf das Schlüsselwort `break` legen, welches die Ausführung der umgebenden Struktur unmittelbar beendet. Im obigen Beispiel wird also die `switch`-Struktur verlassen. Vergessen wir eine `break`-Anweisung, werden alle Anweisungen bis zum nächsten `break` bzw. dem Ende der `switch`-Anweisung ausgeführt. Optional beschreibt das Schlüsselwort `default` den Sonst-Fall.

Verwenden wir Java, darf es sich bei dem Bedingungsausdruck nur um ganzzahlige Werte handeln; in anderen Programmiersprachen wie zum Beispiel **C#** können dies z. B. auch Strings sein.

Sehr oft ist es in unseren Programmen notwendig, dass Anweisungen mehrfach ausgeführt werden. Hierbei kann es sich um eine gegebene *Anzahl* von Wiederholungen handeln oder darum, dass Anweisungen in Abhängigkeit von einer *Bedingung* wiederholt werden müssen. Ein geläufiger Begriff für derartige Konstrukte ist **Wiederholungsanweisungen** oder schlicht und kurz **Schleifen**.

Alle Schleifenarten können in zwei Typen unterteilt werden: In *kopfgesteuerte* und *fußgesteuerte* Schleifen. Der Unterschied besteht darin, dass eine *kopfgesteuerte* Schleife die Bedingung zur Ausführung prüft, bevor die erste Anweisung aus dem Inneren der Schleife ausgeführt wird. Es ist also durchaus möglich, dass die Anweisungen innerhalb der Schleife nie ausgeführt werden. Obwohl wir alle Probleme unabhängig von der verwendeten Schleifenart lösen können, bietet es sich schon an, die *richtige* Auswahl zu treffen. Denn hierdurch kann das Verständnis für einen Algorithmus verbessert werden.

Die *`for`-Schleife* ist eine Zählschleife, in der wir angeben, wie oft die Anweisungen im Schleifenrumpf ausgeführt werden sollen. Nach dem Schlüsselwort `for` wird in Klammern zunächst eine Initialisierung vorgenommen; häufig findet diese mit der Deklaration einer Zählvariablen statt. Durch ein Semikolon getrennt, folgen dann die Abbruchbedingung und letztlich die Aktualisierungsanweisung.

Im nachfolgenden Beispiel wird eine Integer-Variable deklariert und mit dem Wert 1 initialisiert. Anschließend folgt die Abbruchbedingung. Solange diese `true` zurückliefert, werden die Anweisungen innerhalb des Blocks ausgeführt, und im Anschluss daran wird die Aktualisierungsanweisung abgearbeitet.

```
for(int i=1; i<10; i++) {
    // In diesen geklammerten Block kommen alle Anweisungen hinein,
    // die wiederholt abgearbeitet werden sollen.
}
```

Hinweis: Die `for`-Schleife kann auch als Endlosschleife *missbraucht* werden, indem z. B. keine Parameter im Schleifenkopf angegeben werden: `for (;;){...}`. **Endlosschleifen** sind in den allermeisten Fällen unerwünscht. Ein Programm soll schon irgendwann mal zum Ende kommen. Anwendungen, in denen Endlosschleifen hingegen erwünscht sind, sind Programme, die tatsächlich so lange laufen sollen, bis sie jemand abbricht. Dazu gehören z. B. Server-Programme (z. B. Webserver, E-Mail-Server usw.).

In dieser Art wird die `for`-Schleife in Python nicht unterstützt. Das vorangegangene Beispiel sieht in Python folgendermaßen aus:

```
for i in range(1, 10):
    # Anweisungsblock, der wiederholt abgearbeitet werden soll
    # In diesem Beispiel wird der Anweisungsblock neunmal wiederholt
```

Neben der Syntax – die Lauf-Parameter werden nicht in Klammern angegeben – wird die Zählvariable `i` nicht initialisiert, und auch die Bedingung wird nicht explizit formuliert. Stattdessen werden pauschal alle Elemente im Bereich zwischen 1 und 10 nacheinander abgearbeitet. Verantwortlich hierfür ist die Funktion `range(1, 10)`, die die Menge an Zahlen von 1 bis 9 generiert.

Die ebenfalls zur Familie der *kopfgesteuerten* Schleifen gehörende **`while`-Schleife** wird eingesetzt, um Anweisungen so oft wiederholen zu lassen, bis der als Bedingung gesetzte boolesche Ausdruck den Wert `false` liefert.

```
while(i<10) {
    // Anweisungen, die abgearbeitet werden sollen
    i++;
}
```

Hinweis: Auch mit dieser Schleifenvariante lässt sich eine Endlosschleife bauen, z. B. durch den Ausdruck *while (true) {...}*.

Auch hier unterscheidet sich die Umsetzung in Python leicht von der bisher betrachteten Java-Variante:

```
while i < 10:
    # Anweisungen, die abgearbeitet werden sollen
    i += 1
```

Zuletzt betrachten wir noch diese Schleifenart, die im Prinzip der `while`-Schleife ziemlich ähnlich ist. Bei ihr findet die Überprüfung der Bedingung allerdings am Ende des Ausführungsblocks statt. Das bedeutet, dass die Anweisungen im Schleifenrumpf *mindestens* einmal abgearbeitet werden. Aus diesem Grund gehört sie zur Familie der fußgesteuerten Schleifen und ist unter dem Namen **`do-while`-Schleife** bekannt.

```
do {
    // Anweisungen, die abgearbeitet werden sollen
    i += 1;
} while(i<10)
```

Hinweis: Wie bei der `while`-Schleife wird durch eine geeignete Bedingung eine Endlosschleife realisiert, z. B. mit: *do {...} while (true)*.

Wir sollten uns darüber im Klaren sein, dass die Anweisung *break* auch innerhalb von Schleifenrümpfen angewendet werden kann. Wie auch bei den Bedingungsanweisungen führt sie dazu, dass die gegebene Struktur verlassen wird. Das bedeutet, das Programm fährt mit der nächsten Zeile außerhalb der Schleife fort.

Ein weiteres Schlüsselwort **`continue`** ist nur in Schleifenrümpfen erlaubt. Es bewirkt, dass der aktuelle Durchlauf zunächst beendet und die Bedingung neu geprüft wird. Bei der `for`-Schleife wird dazwischen noch die Aktualisierungsanweisung ausgeführt.

Nun haben wir das Rüstzeug für die Arbeit an den nächsten Aufgaben zusammen und können loslegen. Dabei sollten wir auch daran denken, dass wir die Kontrollstrukturen auch beliebig ineinander verschachteln können.

Und nun – ran an die Workouts, und viel Erfolg beim Training!

4.2 Workout

W.4.1 Maximum bestimmen

Schwierigkeit

Zeitaufwand

Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Bedingte Anweisungen

Beschreibung

Das Maximum bestimmt die größte Zahl aus einer Auswahl von vorliegenden Zahlen. Zur Ermittlung des Maximums gibt es in den gängigen Programmiersprachen eine Funktion. Wir wollen uns das aber in dieser Aufgabe selber schreiben.

Aufgabenstellung

Schreibe ein Programm, welches das Maximum von drei Integer-Variablen bestimmt und in der Kommandozeile ausgibt. Benutze bei den Vergleichen ausschließlich `if`- und `else`-Anweisungen sowie den >-Operator.

Testfälle

- Vorliegende Zahlen: 1, 2, 3 → Maximum davon: 3
- Vorliegende Zahlen: 42, 7, 13 → Maximum davon: 42
- Vorliegende Zahlen: -9, 4, 2 → Maximum davon: 4

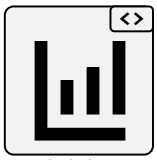
Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Initialisiere zuerst die Variablen und die Variable, die den maximalen Wert haben soll.
- Anschließend kannst du mithilfe einer Abfolge von `if`- und `else`-Anweisungen das Maximum bestimmen.
- Wenn Variable `a` größer ist als `b` und `c`, dann ist `a` das Maximum. Wenn `b` größer ist als `a` und `c`, dann ist `b` das Maximum. Trifft beides nicht zu, dann kann nur `c` das Maximum sein.

W.4.2 Summe berechnen

Schwierigkeit Zeitaufwand Kreativität



Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Wiederholungsanweisungen

Beschreibung

In dieser Aufgabe wollen wir die Summe von 24 aufeinanderfolgenden Zahlen berechnen. Da dies mit dem Taschenrechner ein sehr großer Tippaufwand wäre, stellt sich die Frage: Warum wollen wir das nicht einfach den Computer erledigen lassen?

Aufgabenstellung

Schreibe ein Programm, das mit einer `for`-Schleife die Summe der Zahlen von 3 bis 27 berechnet und das Ergebnis in der Konsole ausgibt.

Testfälle

- Das Ergebnis der Summe von 3 bis 27 ist 375.
- Das Ergebnis der Summe von 1 bis 100 ist 5050.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

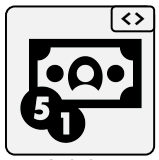
- Deklariere und initialisiere zunächst die Variable, die die Summe beinhalten soll.
- Gehe die Zahlen von 3 bis 27 nacheinander durch.
- Addiere zur Summenvariable die aktuelle Zählerzahl.

W.4.3 Tippspiel

Schwierigkeit

Zeitaufwand

Kreativität



Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Bedingte Anweisungen

Beschreibung

Bei Fußballwettspielen werden die erzielten Wettpunkte nach folgenden Regeln ermittelt:

Exakter Tipp (3 Punkte):

- Sieg (z. B. Ergebnis: 3:2, Tipp: 3:2)
- Niederlage (z. B. Ergebnis: 0:1, Tipp: 0:1)
- Unentschieden (z. B. Ergebnis: 2:2, Tipp: 2:2)

Richtige Tendenz (1 Punkt):

- Sieg (z. B. Ergebnis: 3:2, Tipp: 2:1)
- Niederlage (z. B. Ergebnis: 0:1, Tipp: 1:2)
- Unentschieden (z. B. Ergebnis: 2:2, Tipp: 1:1)

Sonst 0 Punkte

Aufgabenstellung

Schreibe ein Programm, das für ein Fußballtippspiel die erzielten Punkte berechnet und das Ergebnis in der Konsole ausgibt. Dazu werden das getippte Ergebnis und das tatsächlich erzielte Ergebnis des Fußballspiels benötigt.

Testfälle

- Ergebnis: **3:2**, Tipp: **3:2** → Punkte: **3**
- Ergebnis: **0:1**, Tipp: **0:1** → Punkte: **3**
- Ergebnis: **2:2**, Tipp: **2:2** → Punkte: **3**
- Ergebnis: **3:2**, Tipp: **2:1** → Punkte: **1**
- Ergebnis: **0:1**, Tipp: **1:2** → Punkte: **1**
- Ergebnis: **2:2**, Tipp: **1:1** → Punkte: **1**
- Ergebnis: **3:2**, Tipp: **2:0** → Punkte: **1**
- Ergebnis: **0:1**, Tipp: **1:1** → Punkte: **0**
- Ergebnis: **2:2**, Tipp: **2:4** → Punkte: **0**

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Deklariere und initialisiere Variablen für den Tipp und das Ergebnis des Fußballspiels.
- Prüfe anhand der Variablen, welche Punktzahl zutrifft.
- Schreibe diese Punktzahl in die Konsole.

W.4.4 PIN-Code-Generator

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Wiederholungsanweisungen

Beschreibung

In dieser Aufgabe wollen wir alle vierstelligen PINs einer Bankkarte oder eines Passcodes erzeugen und in der Konsole ausgeben.

Aufgabenstellung

Schreibe ein Programm, das die oben beschriebene Aufgabe umsetzt. Setze diese mit einer `for`-Schleife um.

Testfälle

- Ausgabe:

```
0000
0001
0002
0003
0004
0005
0006
0007
...
9993
9994
9995
9996
9997
9998
9999
```

Alle PINs sind vierstellig!

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Alle PINs durchgehen.
- Prüfe, ob die Zahl ein-, zwei- oder dreistellig ist, und füge gegebenenfalls voranstehende Nullen hinzu.
- Da das Processing-Konsolenfenster maximal 500 Einträge anzeigen kann, sind bei einem schnellen Prozessor eventuell nur die letzten 500 Einträge sichtbar. Du kannst die Funktionalität aber überprüfen, indem du die Schleifenanweisung etwas langsamer ablaufen lässt. Der Processing-Befehl `delay(1)` fügt beispielsweise eine künstliche Pause von einer Millisekunde in dein Programm ein.

W.4.5 Ladevorgang-Rädchen

Schwierigkeit

Zeitaufwand

Kreativität

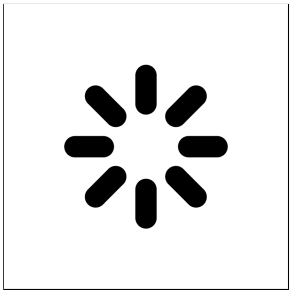
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Wiederholungsanweisungen

Beschreibung

In dieser Aufgabe wollen wir diese Figur programmieren:



Der Ausgabebereich hat dabei folgende Eigenschaften:

- Fenstergröße: 400 x 400 Pixel
- Hintergrundfarbe: Weiß
- Weichzeichnen: aktiv

Die Linien haben dazu diese Einstellungen:

- Linienstärke: 30 Pixel
- Linienende: abgerundet

Damit wir die Formen in der dargestellten Art und Weise anordnen können, benötigen wir zwei neue Anweisungen:

- Die Anweisung `translate()` verschiebt den Nullpunkt des Koordinatensystems an die in der Funktion definierte Stelle.
- Die Anweisung `rotate()` rotiert das Koordinatensystem in einer kreisförmigen Bewegung um den (verschobenen) Ursprung. Die Funktion erwartet als Parameter die Angabe des Winkels im Bogenmaß.

Vor dem Einsatz sollten wir die Dokumentation der Anweisungen auf der Processing-Website durchlesen (https://www.processing.org/reference/).

Der Winkel, mit dem wir die Rotation in der Figur durchführen, beträgt 45 Grad (= PI/4.0 in Bogenmaß).

Aufgabenstellung

Programmiere die in der Beschreibung dargestellte Figur mit den Processing-Grundelementen. Verwende dafür eine oder mehrere Wiederholungsanweisungen.

Testfälle

Siehe Beschreibung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

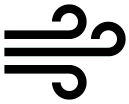
- Verschiebe den Nullpunkt des Koordinatensystems von der linken oberen Ecke in die Mitte des Ausgabebereichs. Um den Mittelpunkt zu bestimmen, kannst du die reservierten Variablen **width** und **height** von Processing verwenden. Diese enthalten die Breite bzw. Höhe des aktuellen Ausgabebereichs.
- Anschließend kannst du eine Linie zeichnen und danach das Koordinatensystem um 45 Grad drehen. Die Koordinaten der Linie bleiben immer gleich, da das Koordinatensystem gedreht wird. Könntest du also diese Schritte in einer Schleife verarbeiten?
- Versuche doch mal, die Linie etwas weiter vom Nullpunkt entfernt zu starten. Dann wird mit der Rotation die Linie mit dem geforderten Abstand zu den anderen Linien rotieren.

W.4.6 Windrad

Schwierigkeit

Zeitaufwand

Kreativität



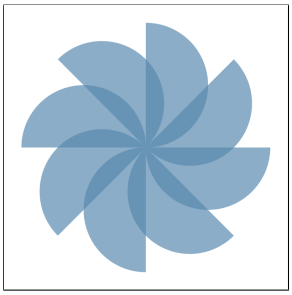
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Wiederholungsanweisungen

Beschreibung

Dieses Mal wollen wir ein Windrad mit Processing grafisch darstellen:



Aufgabenstellung

Programmiere das dargestellte Bild mithilfe von Wiederholungsanweisungen.

Testfälle

Siehe Beschreibung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Schau dir die vorherige Aufgabe noch einmal an.
- Theoretisch kannst du das Bild mit jeder Schleifenart realisieren. Probiere doch mal aus, ob du die Aufgabe mit allen Schleifenarten schaffst.
- Halbkreise kannst du mit der Funktion `arc()` zeichnen. Weitere Informationen findest du in der Processing-Dokumentation.

W.4.7 Rotierte Dreiecke

Schwierigkeit Zeitaufwand Kreativität

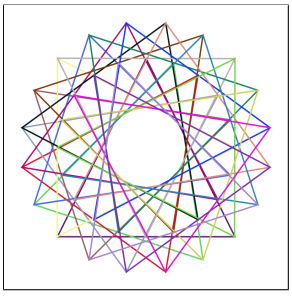
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Wiederholungsanweisungen

Beschreibung

In dieser Aufgabe wollen wir eine Zeichnung mit rotierenden Dreiecken realisieren:



Zur Umsetzung der Grafik sind hier noch zwei Hinweise:

- Es handelt sich um 20 gleichschenklige Dreiecke. Gleichschenklig bedeutet, dass zwei Seiten der Dreiecke gleich lang sind.
- Die Farben der Dreiecke sind zufällig.

Aufgabenstellung

Schreibe ein Programm, das die dargestellte Form erzeugt.

Testfälle

Siehe Beschreibung.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Die vorherigen Zeichenaufgaben in diesem Kapitel könnten dir weiterhelfen.
- Male ein Dreieck um den Nullpunkt herum. Um die Koordinaten herauszufinden, könnte dir eine eigene Zeichnung auf Papier weiterhelfen.
- Drehe das Koordinatensystem anschließend um $\frac{1}{20}$ von 360 Grad. 360 Grad sind im Bogenmaß `2*PI`.

W.4.8 Moderne Kunst

Schwierigkeit

Zeitaufwand

Kreativität

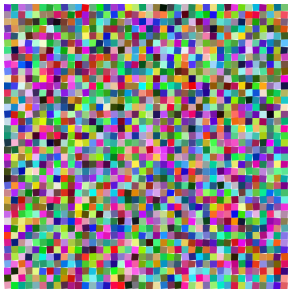
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Verschachtelte Wiederholungsanweisungen

Beschreibung

Moderne Kunstwerke können wir auch mit dem Computer schaffen. Beispielsweise können wir eine Art Mosaik mit Processing realisieren:



Für die Realisierung wichtig ist:

- Die Farben sind zufällig generiert.
- Alle Vierecke sind gleich groß.
- Die Vierecke sind teilweise leicht gedreht.

Aufgabenstellung

Realisiere das oben gezeigte moderne Kunstwerk als Processing-Programm.

Testfälle

Siehe Beschreibung.

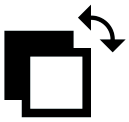
Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Setze die Größe der Kästchen als Variable fest. Dann kannst du in Abhängigkeit von den Fensterabmessungen die Schrittweiten pro Zeile bzw. Spalte berechnen (`height/size` oder `width/size`).
- Ebenso kannst du die Rotationswinkel in einer Variablen speichern. Dann kannst du erst rotieren, dann das Kästchen zeichnen und dann wieder zurückrotieren. So kannst du das Koordinatensystem mit jedem Kästchen weiterwandern lassen. Dies ist einfacher umzusetzen im Vergleich zur manuellen Koordinatenbestimmung für jedes einzelne Kästchen.
- Wenn du das Koordinatensystem an das Ende einer Zeile bewegt hast: Vergiss nicht, für die nächste Spalte wieder die gesamte Fensterbreite zurückzuwandern.

W.4.9 Schachbrett

Schwierigkeit Zeitaufwand Kreativität



Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Verschachtelte Wiederholungsanweisungen

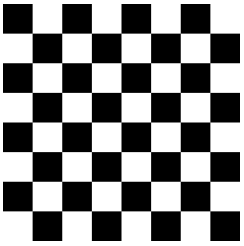
Beschreibung

Wir wollen in dieser Aufgabe das Spielfeld für das Strategiespiel Schach programmieren. Ein Schachfeld besteht aus einem quadratischen Raster mit insgesamt 64 Feldern, die abwechselnd in dunklen (meist Schwarz) und hellen (meist Weiß) Farben gehalten sind. Im Kapitel „Testfall“ ist ein Schachfeld abgebildet, das dir als visuelle Referenz dienen kann.

Aufgabenstellung

Programmiere ein Schachbrett. Benutze dabei Auswahlanweisungen und Wiederholungsanweisungen.

Testfall



Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Gehe jede Zeile und Spalte durch und male dabei jeweils erst ein weißes und dann ein schwarzes Viereck. Die Farbe könntest du zum Beispiel als `boolean`-Variable speichern, die mit jedem Schritt zwischen `true` und `false` wechselt.
- Wie schon in den vorherigen Aufgaben bietet sich das Verschieben des Koordinatenursprungs zum Zeichnen an.
- Achte darauf, dass jede Zeile des Schachbretts nicht mit der gleichen Farbe der vorherigen Zeile beginnt.

W.4.10 Ebbe und Flut berechnen

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Kontrollstrukturen
- Bedingte Anweisungen

Beschreibung

Für den nächsten Urlaub an der Nordsee wollen wir sichergehen, dass wir an einer Wattwanderung am Tag teilnehmen können. Daher wollen wir die Ebbe- und Flutzeiten der kommenden Tage berechnen. Eine Uhrzeit, zu der Ebbe ist, kennen wir bereits.

Grob geschätzt wiederholt sich eine Ebbe oder Flut alle zwölf Stunden und 25 Minuten. Diese Zeit wird auch Tide genannt. In der Mitte der Tide folgt auf eine Ebbe eine Flut und umgekehrt.

Aufgabenstellung

Schreibe ein Programm, das basierend auf einer Ebbe-Uhrzeit die fünf nächsten Ebbe- und Flut-Uhrzeiten berechnet. Das Programm soll außerdem den Tag, von der ersten Ebbe aus gesehen, zu den Uhrzeiten anzeigen. Verwende bei der Umsetzung des Programms Schleifen und Operatoren.

Testfälle

Testfall 1, erste Ebbe um 4:47 Uhr:

```
Tag 0 - Ebbe: 4 Uhr und 47 Minuten
Tag 0 - Flut: 10 Uhr und 59 Minuten
Tag 0 - Ebbe: 17 Uhr und 12 Minuten
Tag 0 - Flut: 23 Uhr und 24 Minuten
Tag 1 - Ebbe: 5 Uhr und 37 Minuten
Tag 1 - Flut: 11 Uhr und 49 Minuten
Tag 1 - Ebbe: 18 Uhr und 2 Minuten
Tag 2 - Flut: 0 Uhr und 14 Minuten
Tag 2 - Ebbe: 6 Uhr und 27 Minuten
Tag 2 - Flut: 12 Uhr und 39 Minuten
Tag 2 - Ebbe: 18 Uhr und 52 Minuten
```

Testfall 2, erste Ebbe um 16:12 Uhr:

```
Tag 0 - Ebbe: 16 Uhr und 12 Minuten
Tag 0 - Flut: 22 Uhr und 24 Minuten
Tag 1 - Ebbe: 4 Uhr und 37 Minuten
Tag 1 - Flut: 10 Uhr und 49 Minuten
Tag 1 - Ebbe: 17 Uhr und 2 Minuten
Tag 1 - Flut: 23 Uhr und 14 Minuten
Tag 2 - Ebbe: 5 Uhr und 27 Minuten
Tag 2 - Flut: 11 Uhr und 39 Minuten
Tag 2 - Ebbe: 17 Uhr und 52 Minuten
Tag 3 - Flut: 0 Uhr und 4 Minuten
Tag 3 - Ebbe: 6 Uhr und 16 Minuten
```