# Wichtige Informationen

## Zu den Arbeitsblättern

Dies ist der erste von fünf Aufgabenzetteln, welche Sie begleitend zum Vorlesungsteil erhalten werden. **Lesen Sie sich dieses Blatt bitte vollständig durch, bevor Sie mit der Bearbeitung von Aufgaben beginnen.**
Jeder Aufgabenzettel hat eine große Auswahl an Aufgaben, welche Ihnen ermöglichen sollen, die Inhalte der Vorlesungen anzuwenden und zu vertiefen. Sie werden nicht annähernd in der Lage sein, alle vorhandenen Aufgaben in den kommenden Nachmittagen zu bearbeiten. Seien Sie also wählerisch! Suchen Sie sich die Aufgaben aus, welche Sie als sinnvoll ersehen.
Diese Dokumente sind in mehrere Kategorien von Aufgaben aufgeteilt. Neu für Sie wie für uns ist dieses Semester die *Byte*-Kategorie, welche kleine Aufgaben beinhaltet, die Sie innerhalb von 15 Minuten erledigen können sollten.
Haben Sie die *Bytes* erledigt (oder empfinden Sie keinen Bedarf dafür), können Sie entweder mit einer empfohlenen Aufgabe fortsetzen oder sich eine andere Aufgabe aussuchen.
Zu jeder Aufgabe existiert am linken Rand eine Tabelle mit Informationen zu Kategorie und Anforderungen (siehe rechts), sowie den Zeilenangaben der Musterlösungen. Diese können Sie als Richtwert verwenden.
Zusätzlich zu den Aufgabenblättern erhalten Sie auch die **Musterlösung** aller Aufgaben, sodass Sie bei Bedarf ihren Ansatz mit diesem vergleichen können.
Beachten Sie, dass es für jedes Problem sehr viele Ansätze gibt, und dass Ihrer nicht zwingend falsch ist, nur weil er das Problem anders als die Musterlösung löst.
Dabei gilt immer: **Kopieren Sie keinen Code! Lassen Sie sich (wenn überhaupt) nur inspirieren!**
Bitte arbeiten Sie mithilfe der Processing Reference (hier klicken). Dort finden Sie alle wichtigen Funktionen, Variablen und Anwendungsbeispiele.
Gerne können Sie auch die Tutor:innen um Hilfe beten.

## Kategorien

| Grafik | Beschreibung |
|---|---|
| Kreis (blau) | Byte |
| Stern (gelb) | Empfohlen |
| Dreieck (grün) | Leicht |
| Quadrat (orange) | Mittelschwer |
| Fünfeck (rot) | Schwer |
| Sechseck (lila) | Herausforderung |

## Anforderungen

| Grafik | Beschreibung |
|---|---|
| √x | Mathematik |
| int x | Variablen |
| | Verzweigungen |
| | Schleifen |
| | Arrays |
| | Geometrie |
| | 3D-Befehle |
| | Zufall |
| | Maus-Interaktion |
| A | Keyboard-Interaktion |
| | Zeit-Variablen |
| f(x) | Funktionen / Methoden |
| C | Klassen |
| v⃗ | Vektoren |

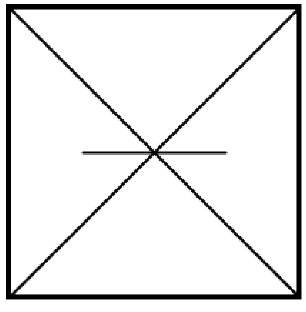
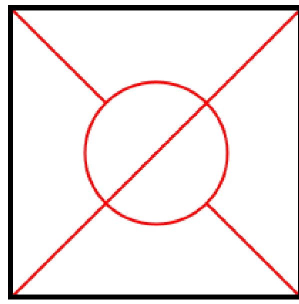
# Aufgabenblatt 1 - Variablen, Operatoren, Verzweigungen

135

**6 Z.**
**7 Z.**

## Aufgabe 1 – Linien I

Schreiben Sie jeweils ein Programm, welches diese Bilder nachstellt. Verwenden Sie dabei Ihnen bekannte Funktionen - eine Auswahl finden Sie unten.

- `size(x, y)` - setzt die Fenstergröße zu $x * y$ Pixel
- `background(r, g, b)` - zeichnet den Hintergrund mit den gegebenen rot, grün und blau-Werten
- `stroke(r, g, b)` - zeichnet alle darauffolgenden Linien und Umrandungen mit den gegebenen rot, grün und blau-Werten
- `fill(r, g, b)` - zeichnet alle darauffolgenden Füllungen mit den gegebenen rot, grün und blau-Werten
- `line(x1, y1, x2, y2)` - zeichnet eine Linie von $(x_1, y_1)$ nach $(x_2, y_2)$.
- `circle(x, y, d)` - zeichnet einen Kreis an Position $(x, y)$ mit Durchmesser $d$.

137

**7 Z.**

## Aufgabe 2 – Rechnungen

Deklarieren Sie zwei Variablen `int x; int y;` und setzen Sie sie auf $x = 3, y = 7$. Geben Sie zuerst die Variablen in der Konsole aus. Lassen Sie sich dann die Ergebnisse von Addition, Subtraktion, Multiplikation und Division zwischen den beiden Variablen ausgeben.

So könnte die Ausgabe der Addition aussehen: `println("x + y = "+ (x + y));`. Die Konsole ist in der Processing-IDE integriert. Führen Sie das Programm wie gewohnt aus, um die Konsolenausgaben zu sehen - die Zeichenbox können Sie in diesem Fall ignorieren.

Ändern Sie dann die Werte für $x$ und $y$ und beobachten Sie, wie sich auch die Ergebnisse der Rechnungen verändern.

Was passiert, wenn Sie `y` auf 0 setzen? Ändert sich die Antwort auf diese Frage, wenn `x` und `y` den Typ `float` statt `int` haben?

## Aufgabe 3 – Analysis

Schreiben Sie ein Programm, welches den Wert einer Variable `int x;` prüft. Geben Sie zunächst den Wert von $x$ in der Konsole aus. Dann sollen die Ergebnisse jeder Überprüfung ausgegeben werden:

- Ist $x$ größer/gleich oder kleiner als 100?
- Ist $x$ gerade oder ungerade (der Ausdruck `x % 2` ist genau dann 0, wenn $x$ gerade ist)?
- Ist $x * 5$ größer/gleich oder kleiner als 50?

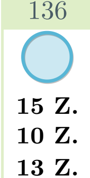
So sieht beispielhaft die Überprüfung aus, ob $x$ größer als 100 ist. Tippen Sie doch einmal den Code ab und führen Sie ihn aus - was passiert, wenn Sie die Variable $x$ ändern?

```
1  int x = 12;

2  if (x >= 100) {
3     println("x ist größer/gleich 100.");
4  } else {
5     println("x ist kleiner 100.");
6  }
```

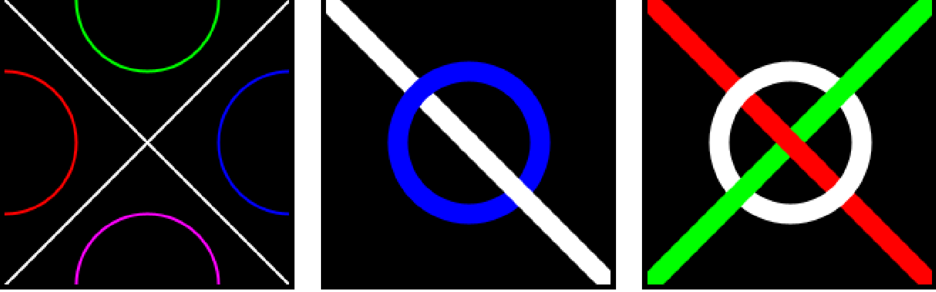
So könnte die Konsolenausgabe bei $x = 12$ aussehen:

```
1  x ist 12.
2  x ist kleiner als 100.
3  x ist gerade.
4  x*5 ist größer/gleich 50.
```




## Aufgabe 4 – Linien II

Wählen Sie eines dieser Bilder, und versuchen Sie es nachzustellen.



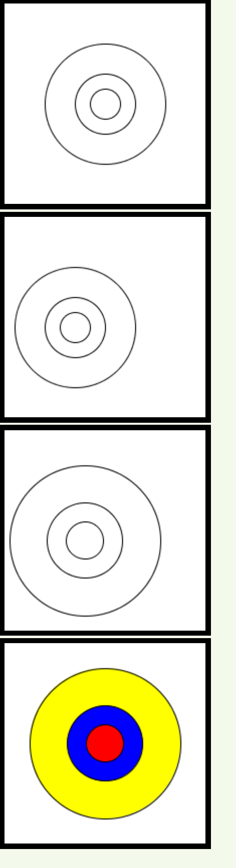
Zusätzliche Funktionen:

- `noFill()` - zeichnet alle darauffolgenden Formen ohne Füllung
- `strokeWeight(i)` - zeichnet alle darauffolgenden Linien und Umrandungen mit einer Pixeldicke von $i$

101

15 Z.

## Aufgabe 5 – Zielscheibe



### 5.1 Zeichnung

Zeichnen Sie eine Zielscheibe mithilfe von drei `circle(x, y, extent)`-Funktionen. Dabei soll der äußerste Kreis den doppelten Radius des mittleren, und den vierfachen Radius des kleinsten Kreis haben.

### 5.2 Positionierung

Deklarieren Sie zwei Variablen `int x` und `int y`, welche die Position der gesamten Zielscheibe festlegen. Verändert man eine der Variablen, verschiebt sich die Zielscheibe bei erneuter Ausführung.

### 5.3 Größeneinstellung

Deklarieren Sie eine weitere Variable `int r`, welche die Größe der Zielscheibe bestimmt. Mithilfe von Operatoren wie `*` und `/` können Sie den Wert einer Variable innerhalb der `circle(x, y, extent)`-Funktion verändern.
Aber Vorsicht: Die Variable `r` beschreibt den *Radius*. Die `circle`-Funktion benötigt jedoch den *Durchmesser*!

### 5.4 Anmalen

Nun sind Sie in der Lage, die Zielscheibe sowohl in der Größe als auch in der Position zu verändern. Als nächstes wollen wir der Scheibe etwas Farbe verleihen. Setzen Sie Ihre Zielscheibe dazu in die Mitte des Zeichenbereiches, und wählen Sie eine angemessene Größe.
Nun können Sie mithilfe von `fill(r, g, b)` die Zielscheibe anmalen. Setzen Sie dazu je vor eine `circle(x, y, extent)`-Funktion eine `fill(r, g, b)`-Funktion.

### 5.5 Passgenauigkeit

Vielleicht ist Ihnen schon aufgefallen, dass man die Zielscheibe mithilfe von `r` derzeit unendlich groß machen kann. Können Sie eine geeignete `if`-Bedingung formulieren, die sicherstellt, dass die Zielscheibe nur dann gezeichnet wird, wenn der gewählte Radius nicht aus der Bildebene ragt?

### 5.6 Flexible Passgenauigkeit

Ihre derzeitige Implementierung ist vermutlich davon ausgegangen, dass die Zielscheibe zentriert ist, während ermittelt wird, ob der Radius „erlaubt“ ist. Können Sie mithilfe von `width` und `height` einen Ausdruck bilden, der abhängig von `r`, sowie `x` und `y` ermittelt, ob der festgelegte Radius zulässig ist? Zeichnen Sie die Zielscheibe wieder nur wenn sie die vier Anforderungen erfüllt.

### 5.7 Größenanpassung

In den meisten Fällen wären Nutzer:innen Ihres Programms überrascht wenn die Zielscheibe plötzlich überhaupt nicht mehr gezeichnet wird, weil sie aus dem Zeichenbereich ragt.
Ihre letzte Herausforderung soll es sein, die Zielscheibe in solchen Fällen nun stattdessen so zu verkleinern, dass sie wieder in den Zeichenbereich passt (Sie sollen also `r` in solchen Fällen vor dem Zeichnen aktualisieren lassen).
Verwenden Sie hierfür die `min(int[] i)`-Funktion - diese gibt den kleinsten Wert eines *Arrays* aus. Was Arrays sind und wie man mit ihnen arbeitet wird später erläutert. Für dieses Programm können Sie zunächst den Code übernehmen: `r = min(new int[]{wert1, wert2, ...})`.
Dabei sind `wert1`, `wert2`, usw. Ihre Zahlenwerte (`x, y, ...`), durch Kommata getrennt. Auch hier können Sie mit Variablen und Operatoren arbeiten.
Hinweis: *Sie müssen Ihre `if`-Bedingung eventuell anpassen.*

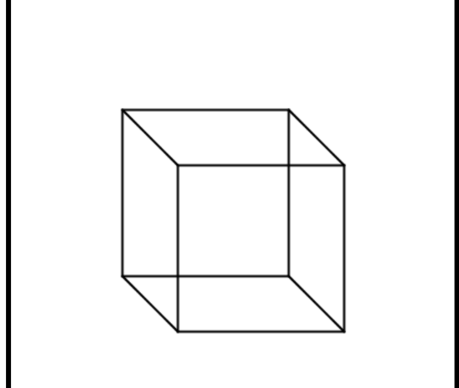
129

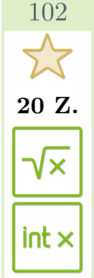
**15 Z.**

## Aufgabe 6 – Dreidimensionaler Würfel

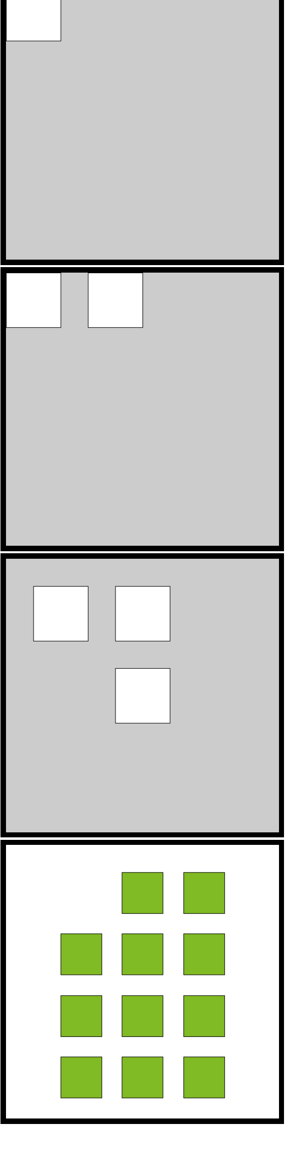
Zeichnen Sie einen scheinbar dreidimensionalen Würfel, mithilfe von zwei `square(x, y, extent)`- und vier `line(x1, y1, x2, y2)`-Funktionen. Legen Sie die Größe der Rechtecke (bzw. Seitenlänge des Würfels) sowie die Position in drei Variablen fest.

Können Sie mithilfe von zwei weiteren Variablen die Verschiebung der Vorderseite in Relation zur Hinterseite (die „Tiefe“) festlegen?






## Aufgabe 7 – THM-Logo



### 7.1 Würfel

Definieren Sie ein Quadrat mithilfe der `square(x, y, extent)`-Funktion. Dabei soll `extent` (bzw. die Höhe und Breite) durch eine Variable `float e` festgelegt werden. Wählen Sie eine angemessene Größe für `e` und platzieren Sie das Quadrat in die obere linke Ecke.

### 7.2 Zwei Würfel mit Abstand

Der Corporate Design Guide der THM (hier klicken) ist zu entnehmen, dass der Abstand zwischen den einzelnen Würfeln die Hälfte der Weite der Würfel beträgt. Zeichnen Sie anhand dieser Vorschrift ein weiteren Würfel rechts von dem ersten, der mithilfe von `e` auch die eigene `x`-Position ermittelt.
Hinweis: *Die Variable* `e` *sollte nun Position und Größe beider Würfel kontrollieren, und die Würfel sollten immer in der gleichen Relation zueinander stehen, egal welchen Wert Sie für* `e` *wählen.*

### 7.3 Vertikaler Nachbar und Offset

Wenden Sie das gleiche Prinzip an, um einen Würfel unter dem rechten Würfel zu zeichnen. Hier soll nun auch `y` mithilfe von `e` bestimmt werden.
Deklarieren Sie zusätzlich die Variablen `float offsetX` und `float offsetY`, welche alle Würfel um einen bestimmten Wert horizontal bzw. vertikal verschieben sollen.
Hinweis: *Um den Offset zu ermöglichen, müssen die Variablen* `offsetX` *und* `offsetY` *bei allen Würfeln zu dessen derzeitigen* `x` *und* `y` *addiert werden.*

### 7.4 Das Logo

Nun geht es ans Eingemachte: Sie haben alle Materialien, um nun das Logo gemäß der Vorschriften zu zeichnen. Es kann sein, dass Sie `e` anpassen müssen, damit alle Würfel mit Abständen in die Bildebene passen.
Verleihen Sie dem Logo auch einen Anstrich mit `fill(128, 186, 36)` und setzen Sie den Hintergrund weiß.
Hinweis: *Um Tipparbeit zu sparen, könnten Sie für jeden vertikalen Schritt einfach* `e + e / 2` *dem* `offsetY` *zufügen.*

128

29 Z.

## Aufgabe 8 – Geheimkombination

IT-Sicherheit ist ein heutzutage sehr relevantes Thema. Eine Möglichkeit, Daten und Anwendungen zu sichern, ist indem sie mit einem Passwort versehen werden.

Für Passwörter werden üblicherweise Zeichenketten oder Zahlenwerte verwendet, aber theoretisch könnte jede Art von Wert dafür benutzt werden. Schreiben Sie ein Programm, welches eine Reihe von bereits entgegengenommenen und gespeicherten Werten überprüft. **Nur wenn alle Werte richtig sind**, soll ein grünes Rechteck auf der Zeichenfläche sein - sonst ist es rot.

Derzeit enthalten die Variablen `secret...` die „richtigen" Werte. Schreiben Sie also Ihren Algorithmus so, dass er jeden der Werte einzeln prüft und bei einem falschen bzw. veränderten Wert die Variable `combinationWrong` auf `true` setzt.
Dabei soll der Wert `secretFloat` zwischen 4 und 5 liegen, überprüfen Sie bitte nicht, ob er genau 4.8 ist. Anhand `combinationWrong` können Sie dann am Ende entscheiden, ob das Rechteck grün oder rot werden soll.

```
size(400, 400);
background(255);

boolean secretBoolean = false;
char secretCharacter = 'g';
String secretString = "theSecretRecipe";
int secretInt = 42;
float secretFloat = 4.8;

boolean combinationWrong = false;

if(secretBoolean != false) {
   combinationWrong = true;
}

...
```

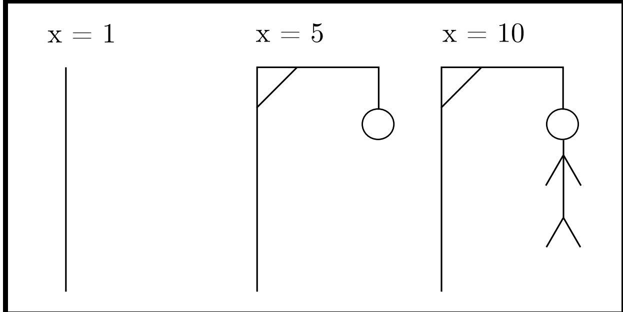
**Achtung**: Bei dem Datentyp `String` muss beim Vergleich auf die Methode `equals()` zurückgegriffen werden, weil in Processing bzw. Java der `==`-Operator in diesem Fall nur prüft, ob es sich um das selbe `String`-Objekt handelt. Die Aussage `"Hello".equals("Hello")` ist also `true`, und `"Hello".equals("World")` ist `false`. Diese Methode (mit dem Punkt nach dem String) kann auch auf Variablen angewandt werden.

103

26 Z.

## Aufgabe 9 – Galgenmännchen I

Schreiben Sie ein Programm, welches ein Galgenmännchen zeichnet. Dabei soll die Variable `x` entscheiden, welche bzw. wie viele der Striche gezeichnet werden sollen.




Für dieses Problem ist es sinnvoll, eine `switch`-Anweisung anstatt von einer Reihe von `if`-Anweisungen zu verwenden. Eine solche `switch`-Anweisung sieht so aus:

```
int x = 1;

switch(x) {
   case 10:
      // rechter Fuß
      line(a, b, c, d);
   case 9:
      // linker Fuß
      line(a, b, c, d);
   case ...:
      ...
   case 1:
      // linker Balken
      line(a, b, c, d);
}
```

Dabei setzt die `switch`-Anweisung dort ein, wo der Wert in den Klammern mit dem hinter `case` übereinstimmt, und führt alle darauf folgenden Anweisung aus, bis die `switch`-Anweisung zu Ende ist, oder bis sie auf ein `break` trifft.

Kommentieren Sie dabei auch immer, welches Element des Bildes sie mit einer `line(x1, y1, x2, y2)`- bzw. `circle(x, y, extent)`-Funktion zeichnen, damit Sie den Überblick nicht verlieren.

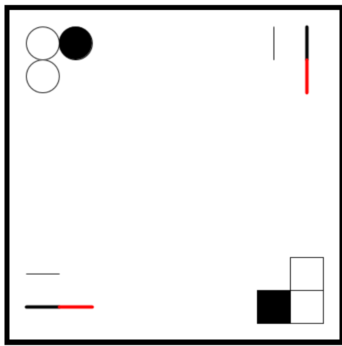
Hinweis: *Diese Aufgabe wird in Tag 3 fortgesetzt.*

105

**20 Z.**

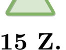
## Aufgabe 10 – Positionierung

Zeichnen Sie die folgende Grafik so genau wie möglich nach. Versuchen Sie dabei die richtigen Koordinaten nicht nur durch ausprobieren herauszufinden, sondern überlegen Sie vorher, wie die Werte in etwa aussehen müssten. Die Befehle für die Farbe und Dicke von Linien finden Sie in der *Processing Reference.*



Sortieren Sie ihre Zeichenbefehle nach (Stroke/Fill)-Farbe und Dicke, um an Codezeilen zu sparen.

130

**15 Z.**


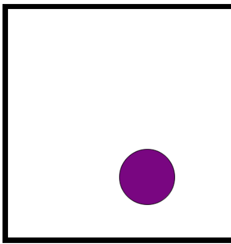
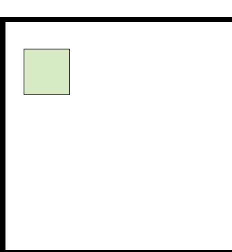
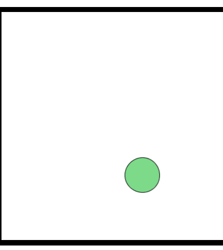
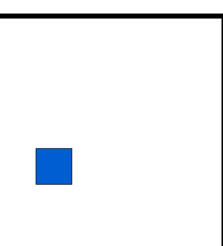
## Aufgabe 11 – Kunst des Zufalls

Generieren Sie moderne Kunst per Zufall. Dabei soll der Zufall entscheiden...

1. ...welche Form gezeichnet wird (Kreis oder Quadrat).
2. ...wo sie gezeichnet wird (`x` und `y`-Positionen).
3. ...wie groß sie sein soll (`extent`).
4. ...in welcher Farbe sie gezeichnet werden soll.

Verwenden Sie dazu die `random(i)`-Funktion, welcher Ihnen einen `float`-Wert zwischen `0` und `i` ausgibt. Mithilfe von `floor(i)` können Sie daraus abgerundete `int`-Werte erhalten.

Hinweis: *Bestimmen Sie erst alle Zufallsvariablen und speichern Sie sie als Variablen, bevor Sie mit dem Zeichnen beginnen.*

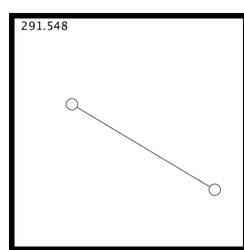
131

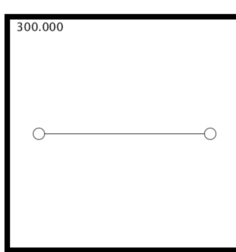
13 Z.

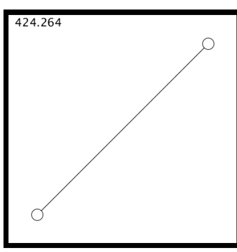
## Aufgabe 12 – Distanzberechnung

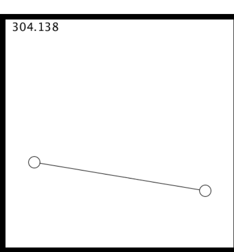
Oft kann es passieren, dass Sie z.B. bei der Entwicklung eines Spieles die Distanz zwischen zwei Punkten benötigen. Verwenden Sie zwei `circle(x, y, extent)`- sowie einer `text(str, x, y)`-Funktion, um zwei vordefinierte Punkte und dessen Distanz darzustellen. Zeichnen Sie zusätzlich eine Linie mit `line x1, y1, x2, y2`, welche die zu messende Distanz darstellt.

Berechnen Sie die Distanz der zwei Punkte mithilfe des Satz des Pythagoras.




Hinweis: *Die Wurzel eines Wertes erhalten Sie mit* `sqrt(i)`*, und um einen Wert mit sich selbst zu multiplizieren, können Sie* `pow(i, 2)` *verwenden.*
Hinweis: *Die Textfarbe ändern Sie mit* `fill(r, g, b)`*, die Textgröße mit* `textSize(i)`*.*

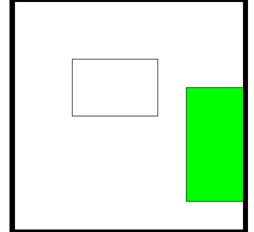
132

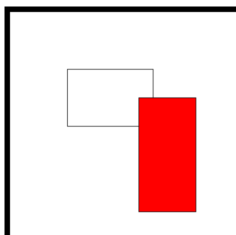
21 Z.

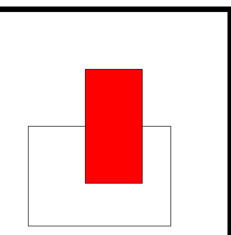
## Aufgabe 13 – Kollisionserkennung

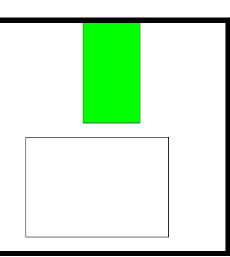
Möchten Sie z.B. ein Platformer oder ein Geschicklichkeitsspiel entwickeln, kann es sein, dass Sie ermitteln müssen, ob sich zwei Rechtecke irgendwo überschneiden, also ob sie kollidieren.

Legen Sie zwei Rechtecke fest. Wenn sie sich überschneiden, soll eines der Rechtecke rot eingezeichnet werden. Ansonsten soll es grün sein.

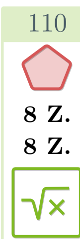








Ermitteln Sie die vier Bedingungen, die erfüllt werden müssen, um zwei Rechtecke als kollidierend zu beschreiben. Falls Sie nicht weiterkommen, kann eine Recherche helfen.

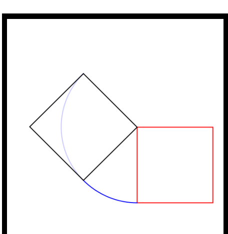

110

8 Z.
8 Z.

## Aufgabe 14 – Rotationen

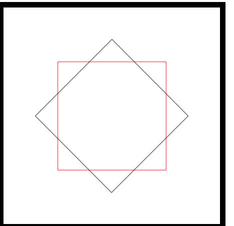
Eine der größeren Herausforderungen in Processing, zumindest was Zeichnungen betrifft, ist das Rotieren von Formen; also Text, Ellipsen, Rechtecke oder andere Formen um einen gewählten Winkel zu rotieren. Mit dieser Aufgabe sollen Sie dies erlernen.

### 14.1 Einleitung zu Rotationen



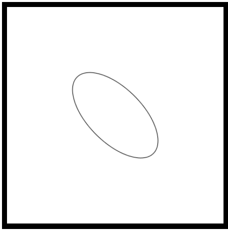
Ein Blick in die *Processing Reference* offenbart die `rotate(angle)`-Funktion. Leider funktioniert diese Funktion nicht ohne weiteres. Statt die darauf zu zeichnende Form zu rotieren (vgl. `fill(r, g, b)` - alle darauffolgenden Formen haben die Farbe `r, g, b`), rotiert `rotate(angle)` das Zeichenbrett um dessen Ursprung.

Im Normalfall befindet sich der Ursprung (Punkt `0, 0`) in der oberen linken Ecke. Das bedeutet, dass wenn man nun den Befehl z.B. für 135° verwendet, sich alle Punkte um 135° aus der Bildebene rotieren, vgl. erste Abbildung rechts (Zeichenbereich zu Beginn in rot dargestellt).



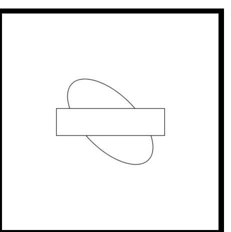
Um nach der Rotation mit sinnvollen Koordinaten arbeiten zu können, verwenden wir zusätzlich die Funktion `translate(x, y)`. Dabei wird durch die Koordinaten die neue Position des Ursprungs gewählt - nach dem Befehl ist also der Punkt `x, y` nun der Punkt `0, 0` - und alle darauf folgenden Zeichenbefehle funktionieren entsprechend.

Wie das dann aussieht, sehen Sie in der zweiten Abbildung rechts - hier wurde `translate(x, y)` auf die Mitte des Zeichenbereiches angewandt, und dann um 45° rotiert.



**Zeichnen Sie zur Eingewöhnung** zunächst eine Ellipse, welche um 45° rotiert ist, in die Mitte des Bildes. Vorsicht: der `rotate(angle)`-Befehl nimmt einen Radiant entgegen. Dabei entsprechen $360° = 2\pi$. Das Rechnen können Sie sich bei Bedarf durch die Verwendung von `radians(degree)` sparen. Das Ergebnis sollte so aussehen wie Abbildung 3 rechts.

### 14.2 push- und popMatrix()



Falls Sie eine komplexere Figur zeichnen möchten, wird es schnell unübersichtlich, wenn die Zeichenebene mehrfach verschoben werden muss, um eine neue Rotation zu ermöglichen. Processing stellt uns die Funktionen `pushMatrix()` und `popMatrix()` zu Verfügung, welche dies erleichtern:

`pushMatrix()` speichert die derzeitige Zeichenebene („Matrix“ - d.h. Rotation und Ort des Ursprungs), und `popMatrix()` ruft die zuletzt gespeicherte Matrix wieder ab und übernimmt sie.

**Zeichnen Sie das rechts abgebildete Bild** mit nur einer `rotate(angle)`-Funktion unter der Verwendung von `pushMatrix()` und `popMatrix()`.

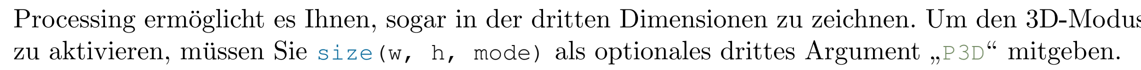
134

**10 Z.**

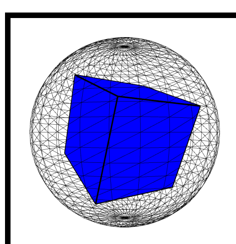
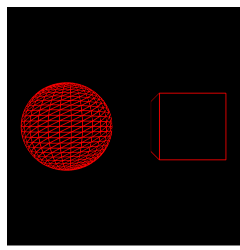
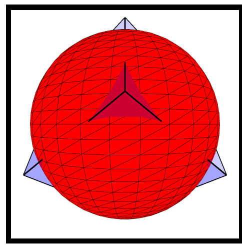
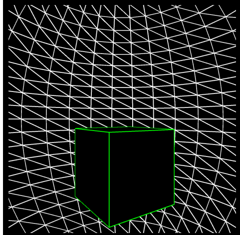
## Aufgabe 15 – Die dritte Dimension

Hinweis: *Es ist ratsam, vor dieser Aufgabe die Aufgabe zu Rotationen zu bewältigen.*

Processing ermöglicht es Ihnen, sogar in der dritten Dimensionen zu zeichnen. Um den 3D-Modus zu aktivieren, müssen Sie `size(w, h, mode)` als optionales drittes Argument „`P3D`“ mitgeben.



**Wenden Sie ihr derzeitiges Wissen über Geometrie und Rotationen an**, um ein Quader mit `box(size)` und eine Sphäre mit `sphere(radius)` zu zeichnen.

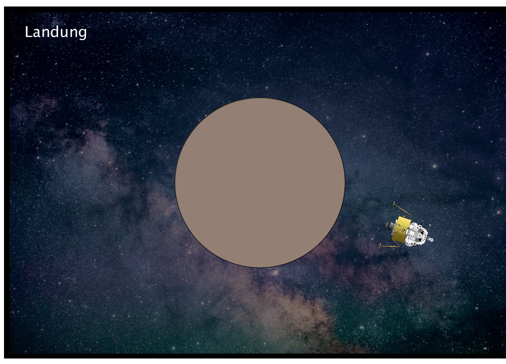
Mithilfe von `translate(x, y)`, sowie `rotateX(angle)`, `rotateY(angle)` und `rotateZ(angle)` können Sie den Zeichenbereich wie gewohnt anpassen. Methoden wie `fill(r, g, b)`, `noFill()` und `strokeWeight(i)` funktionieren wie auch im zweidimensionalen Raum.

133

**36 Z.**

## Aufgabe 16 – Mondlandung

Simulieren Sie eine Mondlandung. Ermitteln Sie den derzeitigen Status des Landeanflugs („im Orbit“, „Landung“, „gelandet“ oder „abgestürtzt“) anhand der Position des Moduls relativ zum Mittelpunkt des Mondes, und geben sie den Status via `text(str, x, y)` an.
Drehen Sie das Modul so, dass die Landebeine zum Mond gerichtet sind, und fügen Sie mit `PImage` Bilder als Hintergrund und für das Modul ein.



Hinweis: *für diese Aufgabe ist es sinnvoll, sich zusätzlich zur Klasse* `PImage` *mit der Klasse* `PVector` *und dessen Befehlen auseinanderzusetzen. Dazu mehr in der Processing Reference.*
Hinweis: *die Aufgabe zu Rotationen bildet eine gute Grundlage zur Darstellung des Moduls.*

Bildquellen: Space, Lunar Module