# Programmieren lernen mit processing

## Ein Arbeitsheft

Autor: Ingo Bartling

Inhaltsverzeichnis

# KAPITEL 1: Allgemeines

## Was ist Programmieren?

*„Programmieren bedeutet ein Problem zu zerlegen, Algorithmen und Abläufe zu definieren und in eine Computersprache zu übersetzten."*

## Was ist ein Algorithmus?

Ein Algorithmus ist eine Vorschrift, die folgende Eigenschaften erfüllt:

1. **Eindeutigkeit**
   Ein Algorithmus muss in der Beschreibung eindeutig sein.

2. **Ausführbarkeit**
   Jeder Einzelschritt muss ausführbar sein.

3. **Finitheit (Endlichkeit)**
   Der Algorithmus muss in endlicher Zeit augeschrieben werden (und damit auch endlich lang sein)

4. **Terminierung**
   Nach endlich vielen Schritten liefert der Algorithmus immer ein Ergebnis

5. **Determiniertheit**
   Bei gleichen Startwerten kommt immer das gleiche Ergebnis heraus

6. **Determinismus**
   Zu jedem Zeitpunkt gibt es nur genaue 1 Möglichkeit, wie fortgesetzt werden kann

***Aufgabe 1.1.1: Rezept***

Schreibe einen Rezept aus mindestens 5 Schritten auf (oder drucke es aus und klebe es in dein Heft ein). Überprüfe nachvollziehbar anhand der obigen 6 Eigenschaften, ob ein Algorithmus vorliegt.

## Was ist eine Computersprache?

Computersprache gibt es in verschiedenen Abstraktionsstufen („01010100101110“ bis Scratch) und dienen der Kommunikation mit einem Computer. Anders als sogenannte natürliche Sprachen, wie Englisch, Französisch oder Deutsch, sind diese Sprache vor allem eindeutig. Das Schlüsselwort „class“ oder „for“ in der Computersprache bedeutet immer das gleiche. Im Deutschen kann das durchaus anders sein. So kann „Mutter“ mal Mama bedeuten oder auch das Gegenstück bei einer Schraube sein. (Mehr dazu in der 12. Klasse)

Unsere Programmiersprache wird Java sein. Man könnte auch eine andere Sprache nehmen, da es fast egal ist mit welcher Programmiersprache man anfängt. Da aber das bayerische Abitur an Java angelehnt ist, ist dies für Schüler am sinnvollsten.

## Was ist Lernen?

Etwas Neues zu lernen erzwingt in der Regel zwei Schritte:

1. Neues integrieren
   Zunächst muss das neue Wissen in Form von Fakten gelernt werden
2. Neues festigen
   Wiederholen, wiederholen, wiederholen

Wenn ich Gitarre lernen möchte, dann muss ich zunächst die Griffe und Anschlag- oder Zupfmuster lernen. Im zweiten Schritt muss das schnelle, automatische Spielen durch viel Üben trainiert werden.

Möchte ich Portraits zeichnen, so muss ich erst die Proportionen genau lernen. Dann übe ich durch viele Wiederholungen.

Beim Programmieren lernen ist dies ähnlich. Zunächst vermittelt der Lehrer die Fakten bevor der Student oder Schüler, angeleitet durch die Lehrperson, selbstständig übt.

# Was ist processing?

Mit Java lassen sich die unterschiedlichsten Arten von Software entwickeln: Büroanwendungen, Apps für Android-Handys, Spezial-Programme für Physik, Bank-Software, etc. Für den unerfahrenen Programmierer macht es meiner Erfahrung nach aber am meisten Freude, wenn er Interaktionen, Spiele und Grafik-Effekte erzeugen kann. Hier erzeugen selbst kleinste Anpassungen am selbst geschriebenen Quelltext sicht- und erlebbare Veränderungen.

Weitere Informationen finden sich bei processing.org.

# KAPITEL 1.2: Datentypen und Datenstrukturen

Ein Programm macht nur Sinn, wenn das Programm Daten hat, die verarbeitet werden. Dies können über vielfältige Arten ein Programm zugeführt werden. Die vorliegenden Daten, aber auch das Programm selbst, muss im Speicher des Computers liegen. Damit der Computer weiß, wie viel Speicher er zur Verfügung stellen muss, muss in den meisten Hochsprachen wie Java der Datentyp einer Variablen angegeben werden.

## Primitive Datentypen

Im Themenbereich Datenbanken hast du bereits verschiedene Datentypen kennengelernt. Auch in Java gibt es sogenannten primitive Datentypen, also Datenstrukturen, die aus keinen anderen bestehen.

### *Aufgabe 1.2.1: Datentypen bei DBs*

Erstelle eine tabellarische Auflistung von mindestens 4 verschiedene Datentypen, die du beim Thema Datenbanken kennengelernt hast, gib jeweils ein Beispiel an und erkläre knapp die Datentypen.

### *Aufgabe 1.2.2: Datentypen in Java*

Erstelle eine tabellarische Auflistung von der für uns wichtigsten primitiven Datentypen in Java. Übertrage hierzu die Tabelle in dein Heft und ergänze - mit Bleistift.

| Datentyp | Beispiel | Erklärung | Wertebereich / Beispiel |
|---|---|---|---|
| | true | | |
| float | 3.141f | Kommazahl | +/-1,4E-45 bis +/-3,4E+38 |
| | | Kommazahlen | |
| int | -2 | ganze Zahlen | -2.147.483.648 bis 2.147.483.647 |
| | | Ein einzelnes Zeichen | |
| String | "Hallo" | Mehrere Zeichen | - |

Auch wenn es kein primitiver Datentyp ist, so ist der Datentyp „String" dennoch so wichtig, dass ich ihn als grundlegend erachte und an dieser Stelle hier einführen möchte.

Starte processing und lösen folgende beiden Aufgaben.

### *Aufgabe 1.2.3: Superhelden-Klasse*

Ziehe eine der Superhelden-Karten und wähle für jede Eigenschaft einen passenden Datentyp. Definiere sodann Variablen für die Superhelden-Eigenschaften und gib Variablenwerte wieder aus.

### *Aufgabe 1.2.4: Schüler-Klasse*

Gib Eigenschaften von dir selbst so an, dass jeder wichtiger primitiver Datentyp (boolean, int, float/double, String) mindestens 1 mal benutzt wird und gib alle Werte wieder aus.

## Referenz-Datentypen

### *Aufgabe 1.2.5: Klasse-Definition*

Strukturiere die Superhelden-Attribute so, dass eine Klasse „Superheld" entsteht.

### *Aufgabe 1.2.6: Methode definieren*

Definiere eine Methode „ausgeben( )", die alle Attribute der Klasse Superheld ausgibt.

### *Aufgabe 1.2.7: Standardkonstruktor*

Definiere einen Standardkonstruktor und einen Konstruktor mit Parametern, so dass alle Attribute der Klasse „Superheld" mit Startwert belegt werden können. Achte auf eine ordentliche Strukturierung.

### *Aufgabe 1.2.8: Methodenkopf & -rumpf*

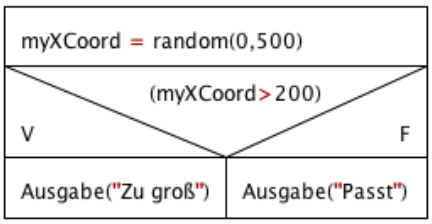
Markiere bei folgenden Methoden den Methodenkopf in Blau und den Methodenrumpf in Grün.

```
void draw() {
   rect(10,10,100,200);
}
```

```
void print(String x) {
     println(x);
}
```

### *Aufgabe 1.3.1: Struktogramme*

a) Übersetze das folgende Struktogramm in Java!




b) Zeichne das Struktogramm zu folgendem Java-Quelltext!

```
if (newXCoord<0) {
   xCoord = 10;
} else {
   xCoord = newXCoord;
}
```

### *Aufgabe 1.3.2: Klasse mit Nicht-Standardkonstruktor*

Definiere einen Klasse für Rechtecke: xCoord, yCoord, laenge, breite inkl. 2 Konstruktoren und ausgeben()-Methode.

### *Aufgabe 1.3.3:if-else*

Definiere eine Klasse für Rechtecke: xCoord, yCoord, laenge, breite inkl. 2 Konstruktoren und ausgeben()-Methode.

Ergänze den Nicht-Standardkonstruktor um if-else-Kontrollstrukturen.

### *Aufgabe 1.3.4: Die Klasse Kreis*

a) Definiere eine Klasse für Kreise, wobei die Klasse folgenden Attributen und Konstruktoren umfasst: xCoord, yCoord, durchmesser, inkl. 2 Konstruktoren und ausgeben()-Methode.

b) Ergänze den nicht-Standardkonstruktor um if-else-Kontrollstrukturen.

c) Verändere den Standardkonstruktor so, dass die Startwerte per Zufall belegt werden. Bediene dich dabei der processing-Funktion

„random(float unterWert, float obererWert)“.

# KAPITEL 1.4: Graphik in Processing

### *Aufgabe 1.4.1: Farben*

In welcher Einheit wird der Bildschirm unterteilt? _____

Erläutere die Bedeutung des Befehls strokeWeight(x) und fill(x). Aus welchen Wertebereich darf x gewählt werden? _____

_____

_____

Zur Darstellung von Farben wird häufig das RGB-Modell genommen. Erläutere dies am Beispiel „fill(255,0,255)“. _____

_____

_____

### *Aufgabe 1.4.2: setup() und draw()*

Viele processing-Projekte werden vor allem mit Hilfe zweier Methoden gesteuert. Erläutere!

| Befehl | Fragen | Deine Erläuterung |
|---|---|---|
| void setup() {<br><br>} | Wie oft wird diese Methode aufgerufen? | |
| void draw() {<br><br>} | Wie oft wird diese Methode aufgerufen?<br>_____ | |

Mit Hilfe welchen Befehls kann ein mehrfaches Wiederholen der Methode draw() verhindert werden?_____

### *Aufgabe 1.4.3: Das Koordinatensystem*

Erläutere das Koordinatensystem (Ursprung, Achsen) in processing!

_____

_____

### *Aufgabe 1.4.4: Grundlagen des Zeichnens*

Zur Darstellung von Objekten in processing musst du ein paar wichtige Methoden und Funktionen kennen. Ergänze die Tabelle jeweils um eine knappe Erklärung oder gib den Befehl an. Beantworte auch die Fragen. Informationen findest du auf processing.org.

| Befehl | Frage | Erläuterung |
|---|---|---|
| size(600,400) | | |
| | | Das Zeichenfenster soll genauso groß sein wie der Bildschirm. |
| frameRate(60) | Wie hoch ist der Standardwert? | |
| background(0) | | |
| | Auf welche Ecke des Objekts bezieht sich die Positionsangabe? | Es wird ein Rechteck an der Position (100\|200) mit der Breite 300 und Höhe 50 gezeichnet. |
| ellipse(20,50,100,100) | | |
| | | Es soll eine Linie vom Punkt (100\|200) zum Punkt (300\|400) gezogen werden |
| mouseX<br>mouseY | | |
| height<br>width | | |

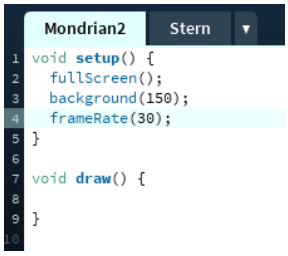
# Komplexerer Programmieraufgaben

1. Lese die ganze Aufgabenstellung.
2. Lies die Aufgabenstellung nochmals und markiere alle Substantive, Adjektive und Verben unter folgendem Aspekt:

   Substantiv – mögliche Klasse
   Adjektiv – mögliches Attribut einer Klasse
   Verb – mögliche Methode einer Klasse
3. Lege ein neues processing-Projekt an und definiere die Methoden setup() und draw() im ersten Reiter des Projekts.
4. Ähnlich zum Kochen erfolgt nun das sogenannte "Mise-en-place". Ergänze die setup()-Methode so, dass die äußeren Rahmenbedingungen für dein Projekt gegeben sind:

   - Fenstergröße
   - (Hintergrund)-Farbe(n)
   - frameRate(30) etc.

   Lege für jeden Referenz-Datentyp eine neue, aber noch leere Klasse in einem eigenen Reiter an.




5. Beginne nun die eigentliche Aufgabe zu lösen, in dem du möglichst einfach beginnst und schrittweise dein Programm erweiterst, verallgemeinerst und um weitere Funktionen ergänzt.

   Beachte:
   - Wechsle erst in eine neue Zeile wechselst, wenn die aktuelle Zeile korrekt ist.
   - Achte darauf, dass dein Programm immer funktioniert.

# Beispiel an der Aufgabe „Mondrian 2"

1. Definiere die Methode setup() mit Methodenrumpf und draw(). Lege eine Klasse „Kreuz" an.
2. Erzeuge innerhalb der Methode draw() ein Kreuz an einer speziellen Stelle z.B. (100|100)
3. Lass dieses Kreuz an der Stelle eines Mausklicks erzeugen.
4. Passe nun die Klasse Kreuz so an, dass mit Hilfe des Standardkonstruktors „Kreuz()" ein Kreuz an der Stelle (100|100) erzeugt wird. Ergänze hierzu die Klasse um die benötigten Attribute, den Standardkonstruktor und eine Methode show(). show() ist dabei nahezu identisch zu dem Programmcode aus Punkt 2.
5. Passe nun die draw()-Methode an:

```
void draw(){
    Kreuz k1 = new Kreuz();
    k1.show();
}
```

### *Aufgabe 1.5.1: Abschlussaufgabe 1(Strichcode)*

a) Lege ein neues processing-Projekt mit dem Namen „Strichcode“ an.

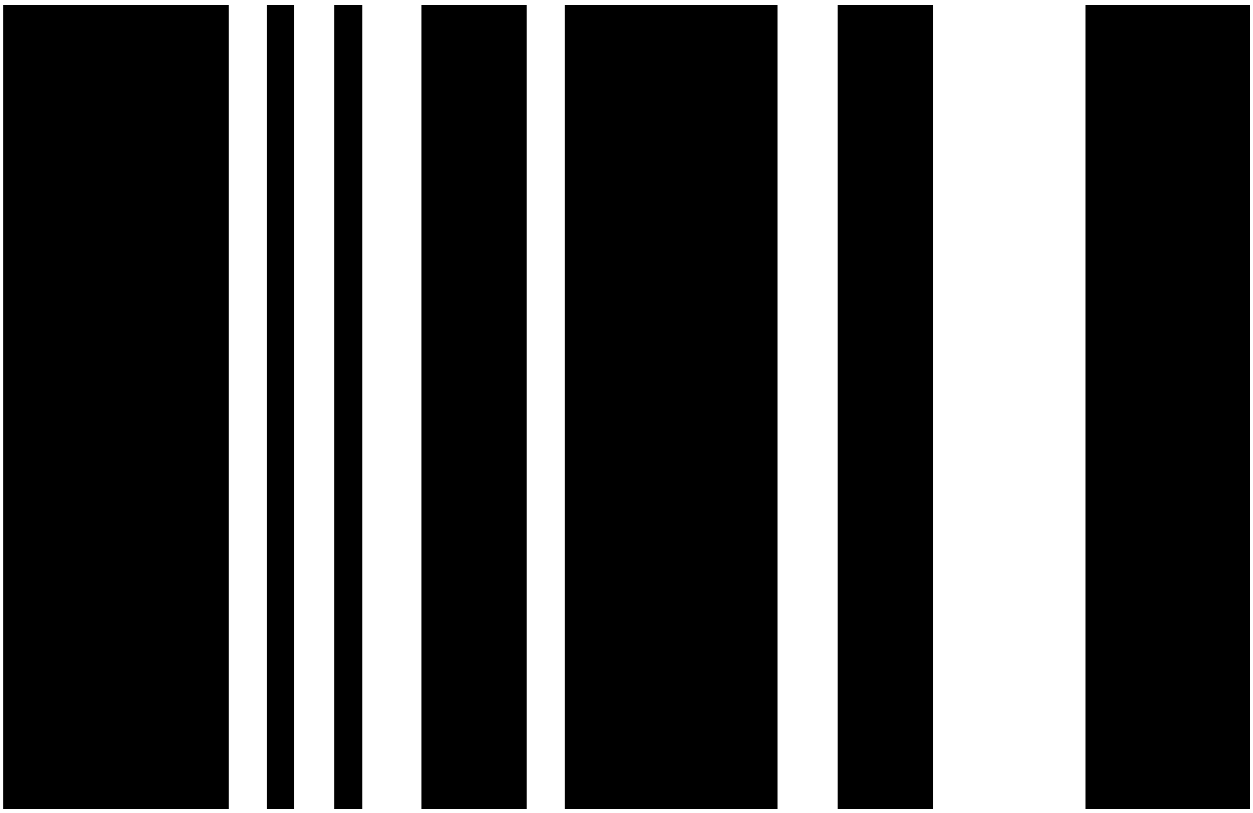
b) Erzeuge ein Fenster in Bildschirmgröße mit schwarzem Hintergrund.

c) Zeichen ein weißes Rechteck über die gesamte Bildschirmhöhe, das 100px breit ist und einen 10px breiten schwarzen Rand besitzt sowie die x-Position 50% von der Bildschirmbreite besitzt.
Warum ist das Rechteck dennoch nicht genau in der Mitte des Bildschirms?

d) Reduziere die frameRate auf 5 und lasse bei jedem neuen Bildaufbau ein bildschirmhohes, zufällig x-positioniertes, weißes Rechteck zeichnen.

e) Erhöhe die frameRate auf 10 und verändere das Programm so, dass das die Rechtecke mit Hilfe der Maus positioniert werden können, aber immer noch Bildschirm hoch sind.

f) Verändere das Programm so, dass mit 80%iger Wahrscheinlichkeit ein weißes, sonst aber ein schwarzes Rechteck gezeichnet wird.
*Tipp: random(1,10)<9*



### *Aufgabe 1.5.2: Abschlussaufgabe 2 (Mondrian 1)*

a) Lege ein neues processing-Projekt mit dem Namen „Mondrian1“ an.

b) Erstelle ausschließlich mit den Methoden setup() und draw() ein Programm mit dem Bilder ähnlich zu Mondrians Bildern erzeugt werden können. Dabei soll ein Fenster der Größe 800x400 mit Rechtecken gefüllt werden, die eine Zufallsfüllfarbe und einen Zufallsbreite sowie –höhe haben. Die Rechtecke besitzen einen 10px breiten, schwarzen Rand.

c) Die Rechtecke sollen automatisch erzeugt werden, wobei ungefähr pro Sekunde nur 1 Rechteck erzeugt wird.

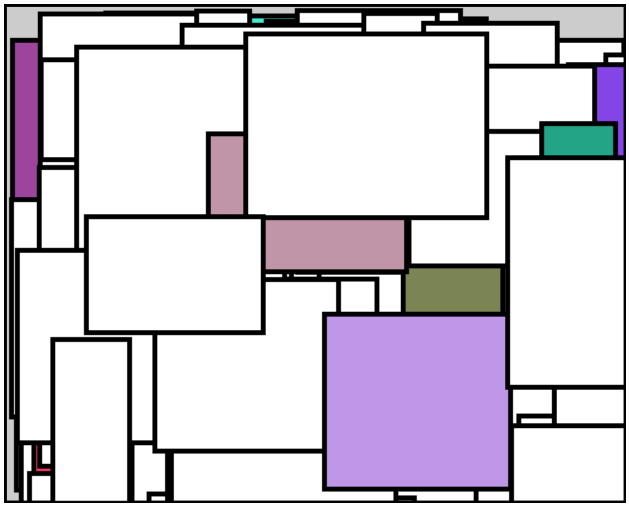
d) Durch Linksklick mit der Maus soll das Bild gelöscht werden.

e) Durch Klicken mit der linken Maustaste werden Rechtecke an der Mausposition erzeugt.

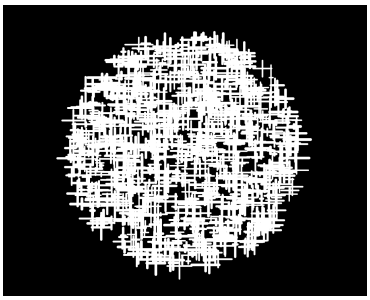
f) Statt Rechtecken werden Quadrate mit einer Zufallsgröße erzeugt.

Sternchenaufgaben dienen dazu dein Problemlösungsdenken zu testen und zu schulen. Diese Aufgaben haben ihre Schwierigkeit oftmals in der Frage: In welchen Schritten löse ich das Problem am einfachsten? Oftmals hilft es sich zu vorzustellen, wie man in der Realität das Problem lösen könnte: „Erst dann Rand malen und dann die Rechtecke. Oder erst die Rechtecke und dann zum Schluss den Rand übermalen.“

g*) Die Position der Rechtecke muss so eingeschränkt werden, dass am Rand ein 10px-breiter schwarzer Rand bleibt.



### *Aufgabe 1.5.3: Abschlussaufgabe 3 (Mondrian2)*



Überlege zunächst welche Information du zum Zeichnen eines Kreuzes benötigst. Programmieren beginnt sollte immer mit Papier und Bleistift beginnen und nicht am Computer.

a) Lege ein neues processing-Projekt mit dem Namen „Mondrian2“ an.

b) Lege eine neue Klasse „Kreuz“ an. Jedes Kreuz soll als Attribute unter anderem xPos und yPos haben.
Jedes Kreuz besteht aus 10px-breiten schwarzen Linien, deren Länge (horizontal wie vertikal) zwischen 10 und 50 Pixeln liegt.

c) Implementiere einen Standardkonstruktor und einen Nicht-Standardkonstruktor.

d) Implementiere auch eine Methode gibAus(), die alle Attribute-Werte in der Konsole ausgibt.

e) Ebenso wird eine Methode show() benötigt, die das Kreuz auf Basis der Attribute-Werte zeichnet.

f) Lege in der Projekt-Klasse die Methode setup() und draw() an. Bei Klick mit der linken Maustaste soll ein Kreuz gezeichnet werden.

g**) Die Position der Kreuze muss so eingeschränkt werden, dass ähnlich zu nebenstehendem Bild ein Kreis gebildet wird.

h*) Mache die Kreuze transparent. Je weiter sie von Mittelpunkt des Fensters entfernt sind, desto durchsichtiger sollen die Kreuze erscheinen. Benutze hierfür den sogenannten Alpha-Kanal einer Farbe, also z.B. stroke(255,100). 100 bestimmt in diesem Fall die Transparenz. Der Wert geht von 0 bis 255.

i**) Verändere die Transparenz oder Farbe mit der Anzahl der durchlaufenen Frames.

Benutze die processing-Funktion „line(x1,y1,x2,y2)“ mit der eine Linie vom Punkt (x1|y1) zum Punkt (x2|y2) gezogen wird.

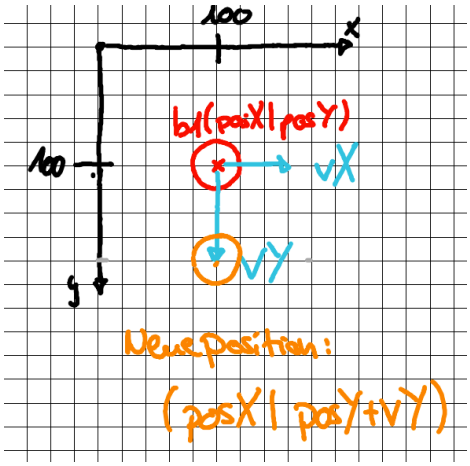
Für die Teilaufgabe g**) kann die Funktion „dist(x1,y1,x2,y2)“ benutzt werden. Diese berechnet den Abstand zwischen dem Punkt (x1|y1) und dem Punkt (x2|y2).
Für die Teilaufgabe h*) kann zusätzlich die Funktion map() benutzt werden.
Bei der Teilaufgabe i**) kann mit dem Modulooperator % gearbeitet werden, der den Rest einer Division zurückliefert: zum Beispiel:15%256 = 15, 255%256 = 255, 256%256 = 0.

## Kapitel 2.1: Physik der Bewegung

Nach den „statischen“ Effekten werden nun einzelnen Figuren wie Kreise und Linien animiert. In einem späteren Kapitel werden dann passend zu Spielen Grafiken animiert.

Anders als in der Physik, wird Bewegung damit nicht in m/s angegeben sondern in „Schrittweite in Pixel“ pro Frame. Die Bewegungsgeschwindigkeit lässt sich zum einen über die „frames per second“ steuern, aber auch über die Schrittweite pro Frame. Das Problem im ersten Fall ist, dass die framerate nicht zwingend gewährleistet werden kann, da diese auch von der Hardware abhängt und nicht beliebig nach oben geregelt werden kann. Wenn nichts anderes angegeben wurde, ist eine framerate von 30 eine gute Wahl. Die Bewegung ist flüssig und lässt sich ganz gut auf m/s umrechnen.

Im Gegensatz zur Realität ist die Bewegung in processing ruckartig: Pro Frame wird beispielsweise ein Kreis um 10px bewegt. Ist ein Kollisionsobjekt nur 1 Pixel entfernt, überlagern sich die Objekte. Wird also die „Schrittweite in Pixel“ zu hoch eingestellt, so wird diese sogenannte „collision detection“ immer komplexer und eine zugehörige immer aufwändiger. In der Regel hat sich ein Wert aus [-5;5] bis maximal |10| bewährt.




Basis der Bewegungsberechnungen bildet die Physik der (beschleunigten) Bewegung.

Im einfachsten Fall ist die Beschleunigung a=0.
Damit ergibt sich die neue y-Position des Balls aus der Berechnung:

„Alte y-Position + Geschwindigkeit“ ergibt die neue „y-Position“.

Übersetzt in Programmcode würde dies lauten:

_____ oder in Kurzform _____

und analog für die x-Koordinate _____ .

### *Aufgabe 2.1.1: Klasse Ball*

a) Lege ein neues processing-Projekt mit dem Namen „Ballphysik“ an. Lege einen weiteren Reiter „Ball“ in diesem Projekt an. Implementiere die Klasse „Ball“ mit einem Standardkonstruktor, so dass oben abgebildete Situation dargestellt wird: Der Ball hat einen Radius von 20px und ist an der Position (100|100).

b) Ergänze die Klasse „Ball“ um die Attribute „float vX;“ und „float vY“. Die Startwerte für die Geschwindigkeitsattribute sollen zunächst auf „vX=0“ und „vY=2;“ festgelegt sein, um eine eindeutige Ausgangssituation zu definieren.

### *Aufgabe 2.1.2: Bewegung mit konstanter Geschwindigkeit*

a) Implementiere in der Klasse „Ball“ eine Methode „void update()“, welche die neue Position des Balls berechnet.
Auf der Karte „Ballphysik“ soll nebenstehender Inhalt dabei umgesetzt.

```
Ball b1 = new Ball();
void setup() {
  background(0);
  framerate(30);
}
void draw() {
  b1.update();
  b1.show();
}
```

b) Erläutere die Zeile

```
Ball b1 = new Ball();
```

b) Erläutere weiter, warum die Zeile „Ball b1 = new Ball();“ außerhalb jeglicher Methode stehen kann.

_____

_____

f) Welchen Effekt hätte es, wenn die Zeile „Ball b1 = new Ball();“ innerhalb der Methode draw() stehen würde.

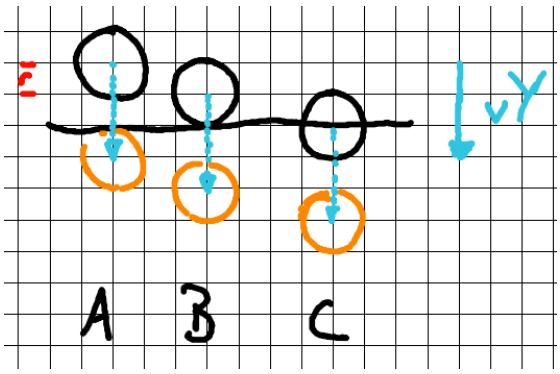
_____

_____

_____

g)

Damit der Ball am unteren Bildschirmrand abprallt, müssen die nebenstehenden drei Situation A, B, C erfasst werden. Hierbei wird ausgehend von der y-Position zunächst der Radius r (rot) hinzuaddiert, um den untersten Punkt des Balls zu berechnen.




Nun wird geprüft, ob beim nächsten Animationsschritt, der unterste Punkt des Balls den Bildschirmrand berührt oder sogar darüber hinzugeht.

### *Aufgabe 2.1.3: Abprallen am Rand*

a) Setze die Formulierung des letzten Absatzes in Programmcode um:

_____

Wenn die Bedingung für das Abprallen erfüllt ist, wird der Wert von vY einfach mit „-1“ multipliziert. Anstatt also 100+2 und damit 102 als neue y-Koordinate zu erhalten, wird 100+(-2) = 98 berechnet. Der Ball bewegt sich damit wieder nach oben.

b) Gib die vollständige bedingte Verzweigung für ein korrektes Abprall-Verhalten am unteren Bildschirmrand an.

_____

_____

_____

c) Gib analog zur Teilaufgabe b) den Programmcode für das Abprallen am linken und rechten Bildschirmrand an.

d) Wie du vielleicht gemerkt hast, hast du zwei Mal fast den gleichen Programmcode eingegeben. Nur die Bedingung hat sich geändert. Umgangssprachlich würde man sagen: „Wenn der Ball am linken Rand ODER der Ball am rechten Rand ist, tue...“. Dieses „ODER“ gibt es auch beim Programmieren: _____ . Das logische UND würde lauten: _____. Fülle nun die folgenden Tabellen aus:

| ODER | true | false |
|---|---|---|
| true | | |
| false | | |

| UND | true | false |
|---|---|---|
| true | | |
| false | | |

e) Fasse die Verzweigungen nun zusammen. Eine für die Vertikal-Bewegung und eine für die Horizontal-Bewegung.

_____

_____

_____

### *Aufgabe 2.1.4: Bewegung mit konstanter Beschleunigung*

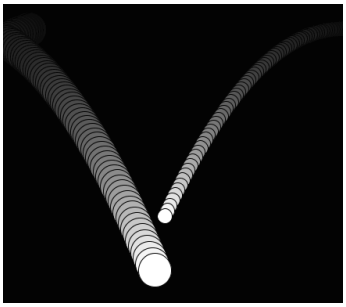
Definiere in der Projekt-Klasse „Ballphysik" eine Variable „g"als Simulation für die Erdbeschleunigung. Initialisiere g in der Methode setup() auf 0.1 . Passe nun die Geschwindigkeit innerhalb der update()-Methode an. Dabei soll vY solange um g erhöht werden, bis der maximale Wert 10 erreicht wurde, denn Körper können aufgrund der Luftreibung nicht beliebig schnell fallen.
Achte in der Methode update() darauf, dass die Schritte Beschleunigen, Abprallen, neue Position in dieser Reihenfolge abgearbeitet werden.

### *Aufgabe 2.1.5: „Energieverlust"*

Um ein realistischeres Verhalten des Balls zu erhalten (u.a. eine abnehmende Sprunghöhe), soll noch ein „Energieverlust" simuliert werden. Hierzu wird vX (Gleit- bzw. Rollreibung) und vY (Verformungsarbeit) bei Kollision mit dem Boden auf 90% verringert werden.
Gib die zugehörigen Programmierzeilen an:

_____

_____



### *Aufgabe 2.1.6: „Schlieren"-Effekt*

Ein schöner Effekt ist der nebenstehende Effekt, den ich Schlieren-Effekt nenne. Dieser wird dadurch erreicht, dass bei jedem Durchgang von draw() das gesamte Bild leicht transparent übermalt wird:

```
//Dunkles Grau (12) das fast vollständig durchsichtig ist (10)
fill(12,10);
//Fenstergroßes Rechteck
rect(0,0,width,height);
//Füllfarbe wieder zurücksetzen
fill(255);
```

### *Aufgabe 2.1.7: für einen Bildschirmschoner*

Lege ein ein neues Projekt „BildschirmSchoner“ an. Erstelle zwei neue Dateien „Linie“ und „Punkt“. Der Anfangs- und Endpunkt der Linie wird durch einen „Punkt“ definiert.

a) Übertrage die Animation aus der Aufgabe 2.1.3 auf die Punkte, so dass die Punkte sich über den Bildschirm bewegen und abprallen würden.

b) Animiere nun die Linie. Benutze hierzu das beistehende Grundkonstrukt für die Klasse Linie.

```
class Linie {
  Punkt anfangsPunkt;
  Punkt endPunkt;
  float r,g,b;
  float thickness;

  void update() {
    anfangsPunkt.update();
    endPunkt.update();
  }

  void show() {
    strokeWeight(thickness);
    stroke(r,g,b);
    line(anfangsPunkt.x,anfangsPunkt.y,endPunkt.x,endPunkt.y);
  }
}
```

### *Aufgabe 2.1.8*: Fliegender Text*

Ersetze die Linie aus Aufgabe 2.1.7 durch einen Text, der sich über den Bildschirm bewegt. Informiere dich dazu auf der Internetseite processing.org über Funktionen mit denen du die Darstellung und Bewegung des Textes steuern und beeinflussen kannst.
Hinweis: text(...), textSize(...), textWidth(...)