

Einstieg in die Programmierung mit Processing

Processing¹ ist eine Programmierumgebung, die sich an Programmieranfängerinnen und -anfänger richtet und auf Java basiert. Sie ist auf die Anwendungsbereiche Grafik und Animation spezialisiert.

Erste Schritte

Abb. 1 zeigt die Programmierumgebung von Processing. Hier können Sie verschiedene Befehle eingeben, die beim Starten des Programms ausgeführt werden sollen. In Zeile 1 wird z. B. die Größe des Fensters und in Zeile 2 die Hintergrundfarbe festgelegt.

Aufgabe 1: Erstellen Sie Ihr erstes Programm, indem Sie die Code-Zeilen aus Abb. 1 in die Programmierumgebung Processing eingeben und das Programm mit Klick auf den Pfeil starten:

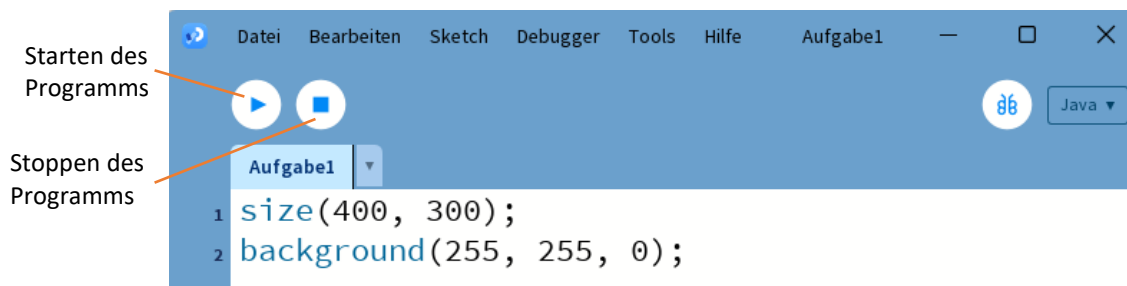


Abb. 1: Ein erstes Programm in Processing

Es sollte ein 400 Pixel breites und 300 Pixel hohes Fenster mit gelbem Hintergrund angezeigt werden (s. Abb. 2). Die Farbwerte werden als RGB-Werte eingegeben, so steht (255, 255, 0) z. B. für Gelb. Die Tabelle in Abb. 3 enthält einige Beispiele für RGB-Werte.



Abbildung 3: Programmfenster zu dem Beispiel aus Abbildung 1

Farbe	RGB-Wert
Weiß	255, 255, 255
Schwarz	0, 0, 0
Rot	255, 0, 0
Grün	0, 255, 0
Blau	0, 0, 255
Gelb	255, 255, 0
Cyan	0, 255, 255
Magenta	255, 0, 255

Abbildung 2: Beispiele für RGB-Werte

Aufgabe 2: Probieren Sie unterschiedliche Größen und Farben für das Programmfenster aus. Um den RGB-Wert für weitere Farben zu bestimmen, können Sie auch das Processing-Werkzeug *Farbauswahl* verwenden. Es wird über den Menüpunkt *Tools* → *Farbauswahl* geöffnet.

Hinweis: Speichern Sie die Programme, die Sie zu den Aufgaben erstellen, jeweils unter dem Namen *AufgabeNr* ab. Manche Programme benötigen Sie später ggf. noch einmal, um sie in einer anderen Aufgabe zu erweitern. Wählen Sie zum Speichern den Menüpunkt *Datei* → *Speichern unter ...*

¹ Die Programmierumgebung Processing wurde 2001 von Ben Fry und Casey Reas initiiert. Nähere Informationen finden Sie unter <https://processing.org/>. Für die Beispiele wurde Processing in der Version 4.0.1 verwendet.

Figuren zeichnen

In das Programmfenster können wir nun verschiedene geometrische Figuren und Linien zeichnen.

Aufgabe 3: Öffnen Sie das Programm *Beispiel_FigurenZeichnen* und beobachten Sie, was beim Starten des Programms gezeichnet wird. Überlegen Sie, welche Bedeutung die **Methoden** (so werden in Processing die Befehle genannt) und die **Parameter**, die in Klammern stehen, haben könnten.

```
1 size(400, 300);
2 background(255, 255, 0);
3 fill(255, 0, 0);
4 square(160, 200, 70);
5 fill(0, 255, 0);
6 rect(185, 160, 20, 40);
7 noFill();
8 circle(195, 130, 60);
9 strokeWeight(2);
10 stroke(0, 0, 200);
11 ellipse(185, 125, 10, 5);
12 ellipse(205, 125, 10, 5);
13 noStroke();
14 fill(255, 0, 0);
15 circle(195, 137, 10);
```

Abbildung 4: Programm-Beispiel Figuren zeichnen

Die Tabelle in Abb. 5 zeigt einige Beispiele für Methoden, die zum Zeichnen zur Verfügung stehen.

Methode (Befehl)	Bedeutung
rect(x, y, width, height)	zeichnet ein Rechteck mit der linken oberen Ecke an der Position x y und der angegebenen Breite und Länge
square(x, y, extent)	zeichnet ein Quadrat mit der linken oberen Ecke an der Position x y und der Seitenlänge <i>extent</i>
ellipse(x, y, width, height)	zeichnet eine Ellipse mit dem Mittelpunkt an der Position x y und der angegebenen Breite und Länge
circle(x, y, extent)	zeichnet einen Kreis mit dem Mittelpunkt an der Position x y und dem Durchmesser <i>extent</i>
triangle(x1, y1, x2, y2, x3, y3)	zeichnet ein Dreieck mit den Eckpunkten x1 y1, x2 y2 und x3 y3
line(x1, y1, x2, y2)	zeichnet eine Linie von Position x1 y1 zu Position x2 y2
point(x, y)	zeichnet einen Punkt an Position x y
stroke(r, g, b)	legt die Farbe der (Umrandungs-)Linien fest, die nachfolgend gezeichnet werden
strokeWeight(weight)	legt die Dicke der (Umrandungs-)Linien fest, die nachfolgend gezeichnet werden
fill(r, g, b)	legt fest, mit welcher Farbe die nachfolgend gezeichneten Figuren gefüllt werden
noFill()	legt fest, dass die nachfolgend gezeichneten Figuren nicht gefüllt werden
...	Eine vollständige Übersicht der Processing-Befehle erhalten Sie, wenn Sie im Menü <i>Hilfe</i> den Punkt <i>Referenz</i> aufrufen.

Abbildung 5: Befehle zum Zeichnen

Das Koordinatensystem in Processing

Die Position der Figuren wird mithilfe der x- und der y-Koordinate im Programmfenster angegeben. Dabei befindet sich die Position 0|0 in der linken oberen Ecke des Fensters. Die positive x-Achse verläuft nach rechts und die positive y-Achse (anders als in der Mathematik!) nach unten.

Aufgabe 4: Erkunden Sie das Koordinatensystem von Processing, indem Sie das Programm *KoordinatenTest* starten und verschiedene Positionen im Fenster mit der Maus anklicken. Das Programm zeigt Ihnen dann die Koordinaten der angeklickten Position.

Aufgabe 5: Probieren Sie verschiedene Befehle zum Zeichnen von Figuren aus, indem Sie z. B. einen Schneemann aus verschiedenen Formen zeichnen. Verwenden Sie dazu die Methoden aus der Tabelle in Abb. 5 und verwenden Sie ggf. ergänzend die Referenz, um weitere Methoden zu recherchieren.

Animationen

Bislang haben unsere Programme eine statische Zeichnung als Ausgabe erzeugt. In diesem Abschnitt lernen Sie, wie Sie Ihre Programme zu einer Animation erweitern können.

Aufgabe 6: Starten Sie das Programm *Beispiel_Animation*. Vergleichen Sie den Programmcode mit dem Aufbau der Programme, die Sie bislang erstellt haben. Welche Unterschiede fallen Ihnen auf?

```
1  int pos = 0;           //Erzeugen einer Variablen pos mit Startwert 0
2
3  void setup(){
4      size(400, 400);
5      background(0, 100, 200);
6      stroke(3);
7      line(0, 10, 400, 410);
8      circle(pos, pos, 20); //Verwenden des Werts der Variablen pos zum Zeichnen eines Kreises
9  }
10
11 void draw(){
12     background(0, 100, 200);
13     line(0, 10, 400, 410);
14     pos = pos + 2;       //Erhöhen des Wertes der Variablen pos um 2
15     circle(pos, pos, 20);
16 }
```

Abbildung 6: Beispiel für eine Animation

Strukturierung eines Programms mit setup und draw

Um eine Animation zu erzeugen, muss das Programm in die Abschnitte `setup` und `draw` unterteilt werden. `setup` und `draw` bezeichnet man wie die Zeichenbefehle als Methoden. Der Unterschied ist, dass die Zeichenbefehle bereits fest definierte Methoden sind, die Sie lediglich aufrufen und damit entscheiden, wann sie ausgeführt werden. Bei `setup` und `draw` ist es andersherum. Im Hintergrund ist festgelegt, dass die Methode `setup` einmal beim Starten des Programms ausgeführt wird, während die `draw`-Methode in regelmäßigen Abständen während der gesamten Laufzeit des Programms immer wieder ausgeführt und das Programmfenster entsprechend aktualisiert wird. Sie legen dabei den Inhalt der Methoden `setup` und `draw` fest, also welche Befehle beim Aufruf der Methoden jeweils ausgeführt werden sollen.

Aufgabe 7: Variieren Sie das Programm in Abb. 6 wie in a) bis c) beschrieben. Beobachten Sie, was die Veränderungen bewirken und erläutern Sie.

- Erhöhen Sie die Variable `pos` in Zeile 14, um einen anderen Wert z. B. 1 oder 5.
- Entfernen Sie den Befehl `background(0, 100, 200)` in Zeile 12 aus der `draw`-Methode und starten Sie das Programm. Was hat sich verändert? Erläutern Sie.
- Ergänzen Sie in der `setup`-Methode den Befehl `frameRate(5)`. Probieren Sie statt dem Wert 5 verschiedene andere Parameterwerte aus und vergleichen Sie die Animationen. Recherchieren Sie die Bedeutung der Methode `frameRate` in der Referenz von Processing.

Variablen

Erläuterungen zur Verwendung einer Variablen für eine Animation am Beispiel in Abb. 6

Der Eindruck einer Bewegung entsteht, wenn sich die Position einer Figur in schnell aufeinander folgenden Bildern ändert. In dem Beispiel in Abb. 6 entsteht der Eindruck eines Balls, der eine Rampe hinunterrollt, da der Kreis von Bild zu Bild ein Stückchen weiter unten entlang der Diagonalen gezeichnet wird. Um diese Positionsänderung im Programm umzusetzen, muss die aktuelle Position des Kreises gespeichert werden, um sie für das nächste Bild ein wenig zu verändern. Dazu wird in Zeile 1 eine Variable `pos` angelegt. Diese speichert die aktuelle x bzw. y-Koordinate des Kreises. Da sich der Kreis auf der Diagonalen bewegt, entspricht die x-Koordinate stets der y-Koordinate, so dass eine Variable für die Position ausreicht. In der `draw`-Methode wird der Wert der Variablen in Zeile 14 um 2 erhöht und als neue aktuelle Position gespeichert, bevor der Kreis mit dem Wert der Variablen `pos` in Zeile 15 erneut gezeichnet wird.

Umgang mit Variablen in Processing im Allgemeinen

Abb. 7 zeigt, wie man in Processing Variablen erzeugt und mit einem Wert belegt:

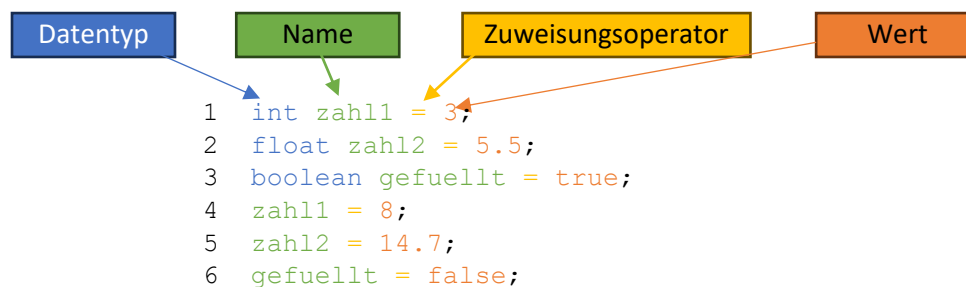


Abbildung 7: Beispiele für den Umgang mit Variablen

Eine Variable wird in Processing erstellt, indem zunächst der **Datentyp** der Variablen festgelegt wird, z. B. `int` für eine Ganzzahl, `float` für eine Fließkommazahl oder `boolean` für einen Wahrheitswert. Darauf folgt der **Name** der Variablen, der im Prinzip beliebig gewählt werden kann. Es ist sinnvoll sprechende Namen zu verwenden, die etwas über den Inhalt oder Zweck der Variablen aussagen. Mithilfe des Gleichheitszeichens kann der Variablen ein **Wert** zugewiesen werden. Wird der Wert der Variablen später verändert, wird der Datentyp nicht mehr angegeben, sondern nur noch der Name der Variablen, der Zuweisungsoperator `=` und der neue Wert (s. Zeile 4 bis 6 in Abb. 7). Das Erzeugen einer Variablen durch Angabe von Datentyp und Name bezeichnet man auch als **Deklaration** und die Zuweisung eines Startwerts als **Initialisierung**. Wird bei der Deklaration kein Startwert zugewiesen, sind Variablen von einem primitiven Datentyp mit einem Standardwert belegt, bis ihnen ein Wert zugewiesen wird. Variablen vom Typ `int` bzw. `float` haben den Standardwert 0 bzw. 0.0. Variablen vom Typ `boolean` haben den Standardwert `false`.

Aufgabe 8:

- Beschreiben Sie die drei Codezeilen in Abb. 8 unter Verwendung der folgenden Fachbegriffe: *Datentyp, Deklaration, Initialisierung, Variable, Wert, zuweisen*
- Begründen Sie, dass in Zeile 3, die Angabe `int` nicht benötigt wird.
- Geben Sie für jede Zeile in Abb. 9 die aktuellen Werte der Variablen `z1` und `z2` an.

```
1  int z1 = 5;
2  int z2 = z1 * 4;
3  z1 = z2 - z1;
```

Abbildung 8

Zeile	z1	z2
1		
2		
3		

Abbildung 9: Tabelle für Aufgabe 8c)

Aufgabe 9: Öffnen Sie das Programm *Blume* (s. Abb. 10) und führen Sie das Programm einmal aus.

- Beschreiben Sie, aus welchen Figuren die Blume zusammengesetzt wird und markieren Sie die entsprechenden Codezeilen in Abb. 10.
- Um die Figuren passend zueinander zu positionieren, sind die x- bzw. y-Koordinate der verschiedenen Figuren aufeinander abgestimmt. Markieren Sie die entsprechenden Stellen in Abb. 10 und erläutern Sie.
- Erzeugen Sie zwei Variablen: `x` und `y`. Verändern Sie das Programm so, dass die Zeichenbefehle die Variablen statt der festen Werte für die x- bzw. die y-Koordinate verwenden. Starten Sie das Programm mehrfach. Ändern Sie vor jedem Start die Werte der Variablen.
Erläutern Sie, welchen Vorteil die Verwendung von Variablen hier hat.

```
1  size(400, 300);
2  background(0, 100, 255);
3  //Stängel
4  stroke(0, 255, 0);
5  strokeWeight(5);
6  line(100, 80, 100, 80+60);
7  //Blüte außen
8  stroke(255, 0, 0);
9  strokeWeight(2);
10 fill(255, 0, 0);
11 circle(100, 80, 30);
12 //Blüte innen
13 stroke(0, 0, 0);
14 fill(255, 255, 0);
15 circle(100, 80, 10);
```

Abbildung 10: Quelltext des Programms *Blume*

- Erweitern Sie das Programm zu einer Animation, in der die Blume „wächst“. Dazu kann die Blume z. B. von einer Startposition aus in die Höhe wachsen oder der Durchmesser der Blüte schrittweise vergrößert werden. Denken Sie daran, Ihr Programm in die Abschnitte `setup` und `draw` aufzuteilen.

Hinweis: Im nächsten Abschnitt lernen Sie, wie Sie dabei verhindern, dass die Blume über das Programmfenster hinauswächst.

Verzweigungen

Betrachten wir noch einmal die Animation aus Abb. 6. Wenn der Ball in der rechten unteren Ecke angekommen ist, verschwindet er aus dem Bild, da die Variable `pos` kontinuierlich um 2 erhöht wird. Das können wir verhindern, indem wir vor dem Erhöhen des Werts prüfen, ob sich der Wert der Variablen noch im sichtbaren Bereich des Programmfensters befindet. Dazu benötigen wir die Kontrollstruktur **Verzweigung**. In Processing wird eine Verzweigung mit dem Schlüsselwort `if` eingeleitet. Nur wenn die Bedingung hinter `if` erfüllt ist, werden die darauffolgenden Befehle ausgeführt. Optional können alternative Befehle angegeben werden, die ausgeführt werden, wenn die Bedingung nicht erfüllt ist. Diese werden mit dem Schlüsselwort `else` eingeleitet.

Die Beispiele in Abb. 11 und Abb. 12 zeigen entsprechende Erweiterungen des Beispiels in Abb. 6.

```
1  int pos = 0;
2  void setup(){
3    size(400, 400);
4    ...
5  }
6
7  void draw(){
8    background(0, 100, 200);
9    line(0, 10, 400, 410);
10   if(pos < 390){
11     pos = pos + 2;
12   }
13   circle(pos, pos, 20);
14 }
```

Abbildung 11: Verzweigung ohne else-Zweig

```
1  int pos = 0;
2  void setup(){
3    size(400, 400);
4    ...
5  }
6
7  void draw(){
8    background(0, 100, 200);
9    line(0, 10, 400, 410);
10   if(pos < 390){
11     pos = pos + 2;
12   }else{
13     noLoop(); // stoppt die Ausführung von draw
14   }
15   circle(pos, pos, 20);
16 }
```

Abbildung 12: Verzweigung mit else-Zweig

Hinweis: Steht im if- bzw. else-Zweig nur eine Anweisung sind die geschweiften Klammern optional. Stehen im if- bzw. else-Zweig mehrere Anweisungen müssen diese in einem Block mit geschweiften Klammern eingeschlossen sein.

Tipp: Je komplexer Ihre Programme werden, desto mehr geschweifte Klammern müssen ineinander geschachtelt werden. Um hier nicht den Überblick zu verlieren, ist es sinnvoll jeden neuen Block, der von geschweiften Klammern umschlossen wird, ein wenig einzurücken und die schließende Klammer unter den Befehl zu schreiben, zu dem sie gehört. So steht in Abb. 11 die schließende Klammer in Zeile 12 unter if... in Zeile 10 und die schließende Klammer in Zeile 14 unter void draw... in Zeile 7. Mit der Tastenkombination *Strg + T* nimmt Processing diese Ausrichtung automatisch vor.

Aufgabe 10: Das Programm *Vorlage_Unterwasserwelt* enthält den Programmcode zum Zeichnen einer kleinen Unterwasserwelt. Das Programm soll zu einer Animation erweitert werden:

- Verändern Sie das Programm so, dass der Eindruck entsteht, dass die Luftblasen nach oben steigen. Setzen Sie eine Luftblase, die den oberen Rand erreicht, wieder an ihre Startposition zurück, so dass sie als „neue“ Luftblase wieder aufsteigen kann.
- Verändern Sie das Programm so, dass der rote Fisch nach rechts und der gelbe Fisch nach links „schwimmt“, bis er jeweils den Rand erreicht. Dann soll er „stehen bleiben“. Schauen Sie sich dazu die Koordinaten der Figuren, aus denen ein Fisch besteht, genau an. Sie sind so notiert, dass erkennbar ist, wie sie von einer x- und y-Position abhängig sind.
- Knobelaufgabe: Die Richtung, in die ein Fisch schaut bzw. schwimmt, ist davon abhängig, ob die x-Koordinate mit einer Plus- oder Minus-Operation verändert wird. Verändern Sie das Programm so, dass ein Fisch am Rand nicht stehen bleibt, sondern in die entgegengesetzte Richtung zurückschwimmt. Die Fische sollen also kontinuierlich von einem Rand zum anderen schwimmen.

Tipp: Aus einer Plus-Operation wird eine Minus-Operation, wenn man den zweiten Summanden mit -1 multipliziert.

Aufgabe 11: Erstellen Sie eine eigene Animation unter Verwendung der folgenden Konzepte: Aufteilung eines Programms in *setup*- und *draw*-Methode, Variablen, Verzweigungen.

Zufallsgrafiken und Muster

Durch das Zeichnen von geometrischen Figuren an zufälligen Positionen lassen sich schöne grafische Effekte erzielen. Dazu benötigen wir neben Variablen, eine Möglichkeit Zufallszahlen zu erzeugen. Außerdem erleichtert die Kontrollstruktur *Schleife* das Zeichnen mehrerer Figuren an verschiedenen Zufallspositionen oder in Mustern.

Zufallszahlen

Eine Zufallszahl erhalten wir mithilfe der Methode `random()`. Als Parameter können wir entweder nur eine obere Grenze oder eine untere und eine obere Grenze angeben.

`random(200)` ; liefert uns eine zufällige Zahl z zwischen 0 und 200 ($0 \leq z < 200$).

`random(50, 200)` ; liefert uns eine zufällige Zahl z zwischen 50 und 200 ($50 \leq z < 200$).

Die erzeugte Zufallszahl ist eine Fließkommazahl vom Datentyp `float`. Möchte man diese in eine Ganzzahl umwandeln, verwendet man den Befehl `int()` und übergibt die Zufallszahl als Parameter: `int(random(5, 20))` liefert eine zufällige Ganzzahl zwischen 5 und 20².

Möchten wir eine Figur an einer zufälligen Position zeichnen, sollte die x- Koordinate zwischen 0 und der Breite des Fensters und die y- Koordinate zwischen 0 und der Höhe des Fensters liegen. Dabei kann es hilfreich sein, die Systemvariablen³ `width` und `height` zu verwenden, die uns die Breite und Höhe des Fensters liefern. Wenn wir die Größe des Fensters ändern, werden die Werte so automatisch angepasst. Wenn das Fenster 400 Pixel breit ist, würde `random(width)` also eine zufällige Zahl zwischen 0 und 400 liefern.

Aufgabe 11:

a) Beschreiben Sie, was die folgenden Codezeilen jeweils bewirken:

1. `circle(random(width), 150, 30);`
2. `circle(100, random(30, height-30), 20);`
3. `circle(200, 150, random(10, height));`

b) Erstellen Sie ein Processing Programm, das ein Quadrat an einer zufälligen Position mit einer zufälligen Seitenlänge zwischen 2 und 100 Pixeln zeichnet. Achten Sie bei den Zufallszahlen für die x- und y- Koordinate darauf, dass sich das Quadrat innerhalb des Fensters befinden sollte.

Soll eine bestimmte Zufallszahl mehrfach verwendet werden, muss sich das Programm den Wert in einer Variablen merken. So könnten dann z. B. drei Quadrate gezeichnet werden, die die gleiche zufällige Seitenlänge haben, sich aber an unterschiedlichen Positionen befinden.

Aufgabe 12: Erläutern Sie den Unterschied zwischen den Codezeilen in Abb. 13 und in Abb. 14.

<pre>float zufallszahlX = random(50, 350); float zufallszahlY = random(50, 350);</pre>	<pre>float zufallszahlX = random(50, 350); float zufallszahlY = zufallsZahlX;</pre>
--	---

Abbildung 13

Abbildung 14

Aufgabe 13: Erstellen Sie ein Processing Programm, dass drei Quadrate mit der gleichen zufälligen Seitenlänge an drei verschiedenen zufälligen Positionen zeichnet.

² Alternativ kann die Umwandlung mit folgender Schreibweise erfolgen: `(int)random(5, 20)`

³ Systemvariablen sind Variablen, die Processing beim Starten eines Programms automatisch erzeugt und zur Verfügung stellt.

Viele Figuren mithilfe einer Schleife zeichnen

Eine einzelne zufällig gezeichnete Figur ist noch nicht besonders spannend. Hübsch wird es, wenn wir z. B. 100 Quadrate oder Kreise an zufälligen Positionen zeichnen lassen. Nun wäre es aber ziemlich umständlich 100-mal die Anweisungen für das zufällige Zeichnen eines Quadrats oder eines Kreises einzugeben. Deshalb verwenden wir dafür die Kontrollstruktur **Schleife**, die Befehle wiederholt ausführt. In Processing gibt es eine *while*-Schleife und eine *for*-Schleife. Beide eignen sich, um Figuren wiederholt zu zeichnen.

Die *while*-Schleife

Die Anweisungen in einer *while*-Schleife werden so lange wiederholt, bis die angegebene **Schleifenbedingung** nicht mehr erfüllt ist. Um eine bestimmte Anzahl an Figuren zu zeichnen, muss eine Variable definiert werden, die z. B. mit dem Wert 0 startet und deren Wert innerhalb der Schleife fortlaufend erhöht wird. Die Variable zählt also mit, wie viele Figuren bereits gezeichnet wurden. Die Schleifenbedingung prüft, ob der Wert der Variablen noch unter der gewünschten Anzahl an Figuren ist. Erreicht der Wert der Variablen die gewünschte Anzahl, bricht die Schleife ab.

In dem Beispiel in Abb. 15 startet die (Zähl-)Variable `i` vom Typ Ganzzahl (`int`) mit dem Wert 0 (siehe Zeile 1). Bei jedem Schleifendurchlauf wird der Wert um 1 erhöht. Die Anweisung `i++` in Zeile 6 ist dabei eine Kurzschreibweise für `i = i + 1`. Wenn die Variable `i` den Wert 100 erreicht hat, bricht die Schleife ab, da die Schleifenbedingung `i < 100` in Zeile 2 nicht mehr erfüllt ist.

Innerhalb der Schleife wird in Zeile 3 zunächst eine zufällige Füllfarbe ausgewählt. In Zeile 4 wird eine zufällige Seitenlänge zwischen 5 und 20 Pixeln erzeugt. In Zeile 5 wird das Quadrat dann an eine zufällige Position gezeichnet, wobei die Größe des Quadrates berücksichtigt wird. Insgesamt werden so 100 zufällige Quadrate gezeichnet.

```
1  int i = 0;
2  while(i < 100){
3      fill(random(255), random(255), random(255));
4      float seitenlaenge = random(5, 20);
5      square(random(width-seitenlaenge), random(height-seitenlaenge),
              seitenlaenge);
6      i++;
7  }
```

Abbildung 15: Beispiel für die Verwendung einer *while*-Schleife zum Zeichnen von 100 zufälligen Quadraten

Aufgabe 14: Erstellen Sie ein Programm, das mithilfe einer *while*-Schleife 150 Kreise mit einer zufälligen Farbe und einer zufälligen Größe zwischen 3 und 15 an einer zufälligen Position zeichnet.

Aufgabe 15: Ändern Sie das Beispiel in Abb. 15 so, dass so lange Quadrate gezeichnet werden, bis ein Quadrat mit einer Seitenlänge größer oder gleich 19 gezeichnet wurde.

Die *for*-Schleife

Eine *for*-Schleife ist speziell für den Fall vorgesehen, dass eine bestimmte Anzahl an Wiederholungen durchgeführt werden soll. Das wäre z. B. bei der *while*-Schleife in Abb. 15 der Fall. Hier sollen genau 100 Wiederholungen ausgeführt werden. Eine *for*-Schleife arbeitet dabei genauso wie die *while*-Schleife in Abb. 15. Die *for*-Schleife fasst allerdings alle notwendigen Angaben (die **Definition der (Zähl-)Variablen**, die **Initialisierung mit einem Startwert**, die **Schleifenbedingung** und das **Erhöhen des Variablenwertes**) innerhalb des Schleifenkopfes zusammen. Abb. 16 zeigt die Umsetzung der *while*-Schleife in Abb. 15 mithilfe einer *for*-Schleife.

```
1 for(int i = 0; i < 100; i++){
2     fill(random(255), random(255), random(255));
3     float seitenlaenge = random(5, 20);
4     square(random(width-seitenlaenge), random(height-seitenlaenge),
5             seitenlaenge);
6 }
```

Abbildung 16: Beispiel für die Verwendung einer *for*-Schleife zum Zeichnen von 100 zufälligen Quadraten

In Zeile 1 in Abb. 16 finden wir im Schleifenkopf zunächst die Deklaration der Variablen *i* und die Initialisierung mit dem Startwert 0, dann die Bedingung, die erfüllt sein muss, damit die Schleife ein weiteres Mal durchlaufen wird (*i* < 100) und zum Schluss die Anweisung *i++*, die am Ende von jedem Schleifendurchlauf ausgeführt wird, um die Variable *i* um 1 zu erhöhen. Für die Zählvariable einer *for*-Schleife wird häufig der Name *i* verwendet, der Name ist aber frei wählbar. Außerdem sind auch andere Schrittweiten möglich. Beispielsweise kann mit *i = i+5*, die Variable *i* bei jedem Schleifendurchlauf um 5 erhöht werden.

Aufgabe 16:

- Ordnen Sie die Bestandteile der *for*-Schleife in Abb. 16 den entsprechenden Codeteilen in Abb. 15 zu.
- Erstellen Sie ein Programm, das mithilfe einer *for*-Schleife 150 Kreise mit einer zufälligen Farbe und einer zufälligen Größe zwischen 3 und 15 an einer zufälligen Position zeichnet. Ersetzen Sie dazu ggf. in Ihrer Lösung aus Aufgabe 14 die *while*-Schleife durch eine *for*-Schleife.

Noch ein wenig abwechslungsreicher können wir die Zufallsgrafiken gestalten, indem wir die Zählvariable *i* beim Erzeugen zufälliger Werte für die Figuren miteinbeziehen. In Abb. 17 wird beispielsweise in Zeile 3 für die Seitenlänge eine Zufallszahl zwischen 5 und *i+6* erzeugt. Dadurch werden die Quadrate tendenziell immer größer:

```
1 for(int i = 0; i < 100; i++){
2     fill(random(255), random(255), random(255));
3     float seitenlaenge = random(5, i+6);
4     square(random(width-seitenlaenge), random(height-seitenlaenge),
5             seitenlaenge);
6 }
```

Abbildung 17: Verwendung der Zählvariablen der *for*-Schleife zur Variation der oberen Grenze einer Zufallszahl

Aufgabe 17: Öffnen Sie das Programm zu dem Beispiel in Abb. 17.

- Verändern Sie den Startwert, die obere Grenze für *i* oder die Schrittweite beim Hochzählen. Tauschen Sie sich untereinander aus, wie sich diese Veränderungen auf die erzeugten Zufallsgrafiken auswirken.
- Verändern Sie das Programm, indem Sie die Zählvariable *i* auch an anderer Stelle miteinbeziehen. Wenn Sie mögen, können Sie auch unterschiedliche Figuren erzeugen.

Aufgabe 18: Öffnen Sie das Programm *Blume*, das Sie in Aufgabe 9 bereits zu einer Animation erweitert haben. Wenn Sie noch nicht mit dem Programm gearbeitet haben, bearbeiten Sie zunächst die Aufgabenteile 9 a) bis c).

Erstellen Sie ein Processing-Programm, das

- fünf Blumen an zufälligen Positionen zeichnet (s. Abb. 18, links).
- fünf Blumen in einer Reihe zeichnet (s. Abb. 18, mitte).
- eine Blumenreihe vom linken bis zum rechten Fensterrand zeichnet (s. Abb. 18, rechts).

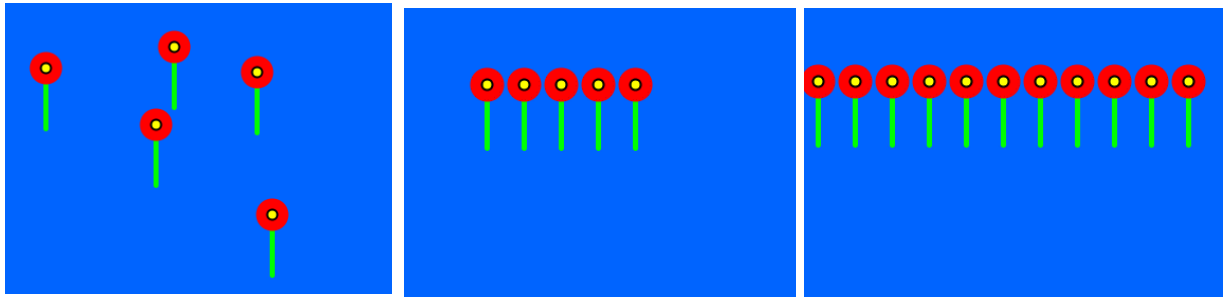


Abbildung 18: exemplarische Ausgabe für Aufgabe 18 a, b und c

Aufgabe 19:

Hinweis: Für diese Aufgabe werden Kenntnisse zur Codierung von Farben im RGB-Modell benötigt.

- a) Stellen Sie eine Vermutung auf, was das Programm in Abb. 19 für eine Ausgabe erzeugt. Überprüfen Sie Ihre Vermutung, indem Sie das Programm *Aufgabe19* öffnen und ausführen.
- b) Ersetzen Sie in dem Programm *Aufgabe19* die *while*-Schleife durch eine *for*-Schleife.
- c) Verändern Sie das Programm so, dass

```
1 size(256, 100);
2 int i = 0;
3 while(i < 256){
4     stroke(i, 0, 0);
5     line(i, 0, i, 100);
6     i = i+1;
7 }
```

- [1] ein Farbverlauf von Schwarz zu Hellgrün erzeugt wird.
- [2] ein Farbverlauf von Schwarz zu Cyan erzeugt wird.
- [3] ein Farbverlauf von Gelb zu Weiß erzeugt wird.
- [4] weitere Farbverläufe Ihrer Wahl erzeugt werden.

Abbildung 19: Quellcode zu Aufgabe 19

Fallunterscheidung mithilfe des Modulo-Operators

Noch ein bisschen mehr Abwechslung können wir in unsere Zufallsgrafiken bringen, wenn wir z. B. einige der zufällig gezeichneten Figuren nur als Umriss zeichnen. Zum Beispiel könnten wir ein Quadrat füllen, wenn unsere Zählvariable *i* gerade ist, und nicht füllen, wenn sie ungerade ist. Dazu benötigen wir eine Verzweigung, die die Fälle „*i* ist gerade“ und „*i* ist ungerade“ bzw. *sonst* unterscheidet.

Dass eine Zahl gerade ist, erkennen wir daran, dass der Rest beim Teilen durch 2 Null ist. Den Rest einer Division liefert der Operator Modulo, der in Processing mit dem Zeichen `%` dargestellt wird.

Für einen Vergleich wird ein doppeltes Gleichheitszeichen verwendet, da das einfache Gleichheitszeichen schon für die Wertzuweisung bei einer Variablen belegt ist.

Das Beispiel in Abb. 20 verwendet die Bedingung „*i* modulo 2 gleich Null“, um zwischen geraden und ungeraden Werten für *i* zu unterscheiden. Wenn die Bedingung erfüllt ist, also *i* gerade ist, wird die Anweisung `fill...` in Zeile 2 ausgeführt, ansonsten die Anweisung `noFill...` in Zeile 4.

```
1 if((i%2) == 0){
2     fill(random(255), random(255), random(255));
3 }else{
4     noFill();
5 }
```

Abbildung 20: Verzweigung, die zwischen gerade und ungeraden Werten für *i* unterscheidet

Aufgabe 20: Erweitern Sie eines Ihrer Programme zum Erzeugen von Zufallsgrafiken um eine Verzweigung, die die Figuren abwechselnd mit verschiedenen Eigenschaften zeichnet. Beispielsweise können abwechselnd verschiedene Formen oder verschiedene Wertebereiche für die Zufallszahlen verwendet werden.

Aufgabe 21: Verändern Sie Ihr Programm aus Aufgabe 18b oder 18c so, dass sich die Farben in der Blumenreihe abwechseln. Wechseln Sie zunächst zwischen zwei, anschließend zwischen drei Farben.

Tipp: Drei Kategorien können unterschieden werden, in dem geprüft wird, welcher Rest bei der Division der Zählvariablen durch 3 entsteht.

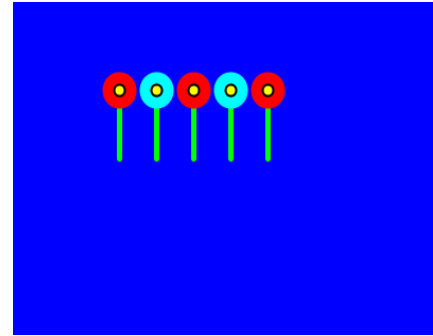


Abbildung 21: exemplarische Ausgabe zu Aufgabe 21

Aufgabe 22:

- Erstellen Sie ein Programm, das einen Turm aus Quadraten zeichnet. Die Farben der Quadrate sollen dabei wechseln (s. Abb. 22).
- Erstellen Sie ein Programm, das eine dreifarbige Raupe zeichnet (s. Abb. 23). Ergänzen sie gerne weitere Details wie z. B. Fühler und Augen.
- Knobelaufgabe: Vielleicht kennen Sie die optische Täuschung in Abb. 24. Erweitern Sie Ihr Programm aus Aufgabenteil a) zu einer entsprechenden Zeichnung. Entsteht die optische Täuschung auch, wenn Sie farbige Quadrate verwenden?

Hinweis: Halten Sie Ihren Quelltext für die Lösung dieser Aufgaben durch die Verwendung von Schleifen und Verzweigungen möglichst kurz. Im Idealfall werden die benötigten Zeichenbefehle nur einmal aufgeschrieben.

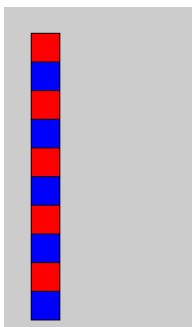


Abbildung 24: Turm aus roten und blauen Quadraten.

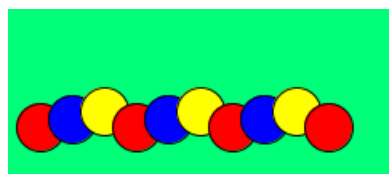


Abbildung 24: Dreifarbige Raupe

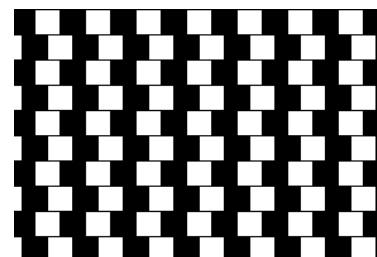


Abbildung 24: Optische Täuschung

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](#).

Für die korrekte Ausführbarkeit der Quelltexte in diesem Leitfaden und der beiliegenden Quelltexte zu den Beispielen und Aufgaben wird keine Garantie übernommen. Auch für Folgeschäden, die sich aus der Anwendung der Quelltexte oder durch eventuelle fehlerhafte Angaben ergeben, wird keine Haftung oder juristische Verantwortung übernommen.