

## Einstieg in die Programmierung mit Processing

Processing<sup>1</sup> 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

Abbildung 1 zeigt das Grundgerüst eines Processing-Programms. In die geschweiften Klammern hinter `void setup()` werden alle Befehle geschrieben, die beim Starten des Programms einmalig ausgeführt werden sollen. Hier können z. B. die Größe des Fensters und die Hintergrundfarbe festgelegt werden.

**Aufgabe 1:** Erstellen Sie Ihr erstes Programm, indem Sie die Code-Zeilen aus Abbildung 1 eingeben und das Programm mit Klick auf den grünen Pfeil starten:

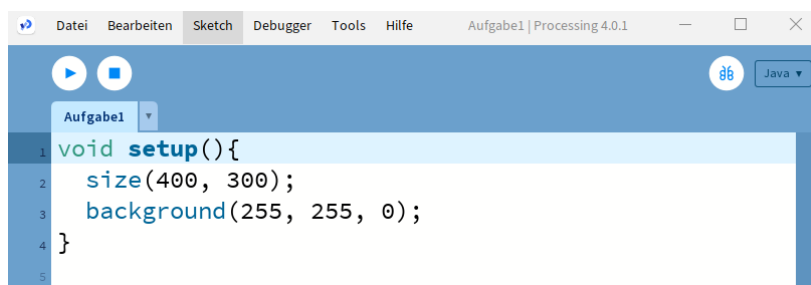


Abbildung 1: Ein erstes Programm in Processing

Es sollte ein 400 Pixel breites und 300 Pixel hohes Fenster mit gelbem Hintergrund angezeigt werden (s. Abbildung 2). Die Farbwerte werden als RGB-Werte eingegeben, so steht (255, 255, 0) z. B. für Gelb. Tabelle 1 enthält einige Beispiele für RGB-Werte.



Abbildung 2: 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

Tabelle 1: Beispiele für RGB-Werte

**Aufgabe 2:** Probieren Sie unterschiedliche Größen und Farben für das Fenster aus.

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

<sup>1</sup> 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 Fenster können wir nun verschiedene Figuren und Linien zeichnen.

**Aufgabe 3:** Ergänzen Sie den folgenden Code in Ihrem Programm. 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 void setup() {
2   size(400, 300);
3   background(255, 255, 0);
4   fill(255, 0, 0);
5   rect(30, 60, 50, 50);
6   fill(0, 255, 0);
7   rect(180, 100, 20, 40);
8   noFill();
9   circle(200, 250, 60);
10  strokeWeight(5);
11  ellipse(250, 150, 30, 60);
12 }
```

*Beispiel 1: Figuren zeichnen*

Die folgende Tabelle zeigt einige Beispiele für Methoden, die zum Zeichnen zur Verfügung stehen. Eine vollständige Übersicht erhalten Sie, wenn Sie im Menü *Hilfe* den Punkt *Referenz* aufrufen.

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 dritten Parameter als Durchmesser
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

*Tabelle 2: Befehle zum Zeichnen*

## Das Koordinatensystem in Processing

Die Position der Figuren wird mithilfe der x- und y-Koordinate im Zeichenfenster 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:** Testen Sie die Befehle zum Zeichnen von Figuren, indem Sie z. B. einen Schneemann aus verschiedenen Formen zeichnen. Verwenden Sie dazu das Grundgerüst eines Processing-Programms aus Abbildung 1.

## Zufallsgrafiken und Muster

Mithilfe von Zufallsgrafiken und Wiederholungen lassen sich schöne grafische Effekte erzielen. Dazu benötigen wir verschiedene algorithmische Konzepte wie z. B. Zufallszahlen, Variablen und Schleifen. Deren Verwendung in Processing schauen wir uns im Folgenden an.

### 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 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 20.

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 Systemvariablen<sup>2</sup> *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 6:

a) Beschreiben Sie, was die folgende Codezeile bewirkt:

```
circle(200, 150, random(10, height-10));
```

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.

## Variablen

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

---

<sup>2</sup> Systemvariablen sind Variablen, die Processing beim Starten eines Programms automatisch erzeugt und zur Verfügung stellt.

zufälligen Seitenlänge haben, sich aber an unterschiedlichen Positionen befinden. Abbildung 3 zeigt, wie man in Processing eine Variable erzeugt, die einen Wert speichert:

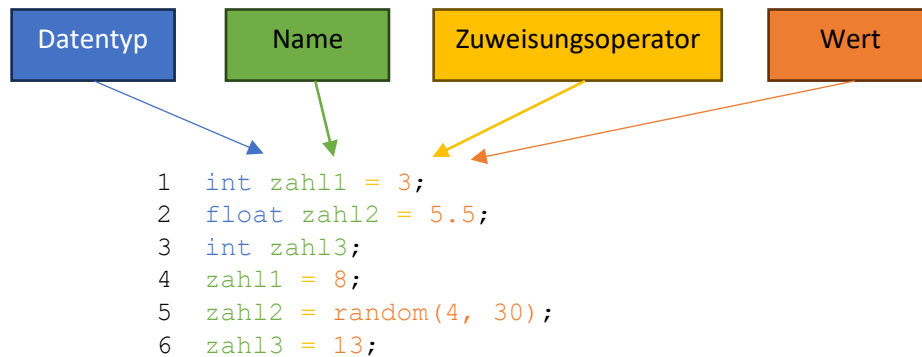


Abbildung 3: Beispiele für das Erzeugen und Arbeiten mit Variablen

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 beliebig gewählt werden kann. Mithilfe des Gleichheitszeichens kann der Variablen optional noch ein **Wert** zugewiesen werden. Soll der Wert der Variablen später verändert werden, wird der Datentyp nicht mehr angegeben, sondern nur noch der Name der Variablen, der Zuweisungsoperator `=` und der neue Wert (s. Zeile 4).

**Aufgabe 7:** Erweitern Sie Ihr Programm aus Aufgabe 6b so, dass drei Quadrate mit der gleichen zufälligen Seitenlänge an drei verschiedenen Positionen gezeichnet werden.

**Aufgabe 8:** Öffnen Sie das beiliegende Programm *Blume* (s. Abbildung 4) und führen Sie das Programm einmal aus.

- Beschreiben Sie, aus welchen Figuren die Blume zusammengesetzt wird und markieren Sie die entsprechenden Codezeilen in Abb. 4.
- 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. 4 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  void setup(){
2      size(400, 300);
3      background(0, 100, 255);
4      //Stängel
5      stroke(0, 255, 0);
6      strokeWeight(5);
7      line(100, 80, 100, 80+60);
8      //Blüte außen
9      stroke(255, 0, 0);
10     strokeWeight(2);
11     fill(255, 0, 0);
12     circle(100, 80, 30);
13     //Blüte innen
14     stroke(0, 0, 0);
15     fill(255, 255, 0);
16     circle(100, 80, 10);
17 }

```

Abbildung 4: Quelltext des Programms *Blume*

- Verändern Sie das Programm so, dass die Blume an einer zufälligen Position gezeichnet wird.

**Aufgabe 9:** Erläutern Sie den Unterschied zwischen den Codezeilen in Abb. 5 und in Abb. 6.

float zufallszahlX = random(50, 350);	float zufallszahlX = random(50, 350);
float zufallszahlY = random(50, 350);	float zufallszahlY = zufallsZahlX;

Abbildung 6

Abbildung 5

**Aufgabe 10:**

- Beschreiben Sie die drei Codezeilen in Abb. 7 unter Verwendung der folgenden Fachbegriffe: *Datentyp, zuweisen, Variable, Wert*
- Begründen Sie, dass in Zeile 3, die Angabe `int` nicht benötigt wird.
- Geben Sie für jede Zeile in Abb. 7 die aktuellen Werte der Variablen `z1` und `z2` an.

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

Abbildung 7

Zeile	z1	z2

Abbildung 8: Tabelle für Aufgabe 10c)

## 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 Blumen an zufälligen Positionen zeichnen lassen. Nun wäre es aber ziemlich umständlich 100-mal die Anweisungen für das zufällige Zeichnen eines Quadrats oder gar einer Blume einzugeben. Deshalb verwenden wir dafür eine Zählschleife, die Befehle so oft wie angegeben wiederholt. Eine Zählschleife lässt sich in Processing auf zwei Arten realisieren, mithilfe einer *while*-Schleife oder mithilfe einer *for*-Schleife:

### While-Schleife als Zählschleife

Die Anweisungen in einer *while*-Schleife werden wiederholt, bis die **Schleifenbedingung** nicht mehr erfüllt ist. Für eine Zählschleife muss eine Variable definiert werden, die z. B. mit dem Wert 0 startet und deren Wert innerhalb der Schleife fortlaufend erhöht wird, bis sie einen bestimmten Wert erreicht hat.

In Beispiel 2 startet die 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 99 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  }
```

Beispiel 2: While-Schleife als Zählschleife



### For-Schleife

Die *for*-Schleife arbeitet im Prinzip genauso wie die oben definierte *while*-Schleife. Es stehen jedoch alle Angaben (die **Definition der Variablen**, die **Initialisierung mit einem Startwert**, das **Erhöhen des Variablenwertes** und die **Schleifenbedingung**) innerhalb des Schleifenkopfes:

```
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 }
```

#### Beispiel 3: For-Schleife

In Zeile 1 in Beispiel 3 finden wir somit im Schleifenkopf zunächst die Anweisung zum Erstellen der Variablen *i* 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 um 1 zu erhöhen.

Noch ein wenig abwechslungsreicher können wir die Zufallsgrafik gestalten, indem wir die Zählvariable *i* beim Erzeugen der Zufallszahlen mit einbeziehen. So können wir zum Beispiel in Zeile 3 für die Seitenlänge eine Zufallszahl zwischen 5 und *i*+6 erzeugen lassen. 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 }
```

#### Beispiel 4: Verwendung der Zählvariablen in der for-Schleife

### Aufgabe 11:

- Ordnen Sie die Bestandteile der *for*-Schleife in Beispiel 3, den entsprechenden Codeteilen in Beispiel 2 zu.
- Verändern Sie das Programm aus Beispiel 4, indem Sie die Zählvariable auch an anderer Stelle miteinbeziehen. Verändern Sie auch die Anzahl der erzeugten Quadrate, den Startwert oder die Schrittweite beim Hochzählen. Sie können natürlich auch noch andere Figuren hinzunehmen.

**Aufgabe 12:** Verändern Sie Ihr Programm *Blume* aus Aufgabe 8 so, dass

- fünf Blumen an zufälligen Positionen gezeichnet werden (s. Abb. 9 links).
- fünf Blumen in einer Reihe gezeichnet werden (s. Abb. 9 mitte).
- eine Blumenreihe vom linken bis zum rechten Fensterrand gezeichnet wird (s. Abb. 9 rechts).

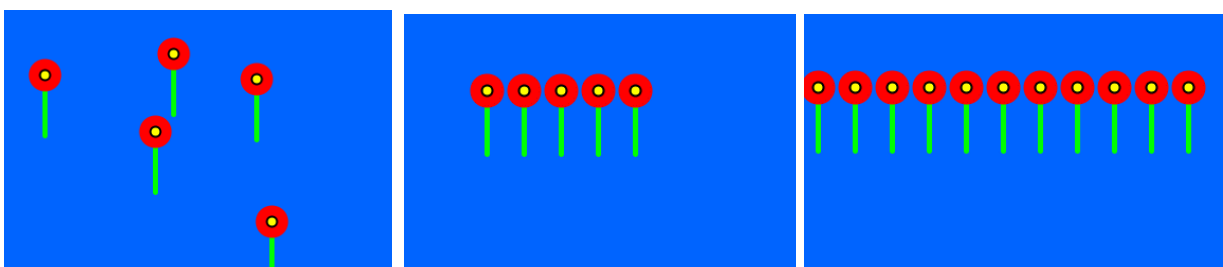


Abbildung 9: Vorschau für Aufgabe 12a, b und c

### Aufgabe 13:

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

- a) Stellen Sie eine Vermutung auf, was das Programm in Abbildung 10 für eine Ausgabe erzeugt. Überprüfen Sie Ihre Vermutung, indem Sie das Programm *Aufgabe13* öffnen und ausführen.
- b) Ersetzen Sie in dem Programm *Aufgabe13* die *while*-Schleife durch eine *for*-Schleife.
- c) Verändern Sie das Programm so, dass
  - [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.

```
1 void setup() {  
2     size(256, 100);  
3     int i = 0;  
4     int rotwert = 0;  
5     while(i < 256){  
6         stroke(rotwert, 0, 0);  
7         line(i, 0, i, 100);  
8         i = i+1;  
9         rotwert = rotwert + 1;  
10    }  
11 }
```

Abbildung 10: Quellcode zu Aufgabe 13

**Tipp:** Je komplexer deine 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 Abbildung 10 die schließende Klammer in Zeile 10 unter *while*... in Zeile 5 und die schließende Klammer in Zeile 11 unter *void*... in Zeile 1. Mit der Tastenkombination Strg + T nimmt Processing diese Ausrichtung automatisch vor.

### Variation der Figuren mithilfe von Verzweigungen

Noch ein bisschen mehr Abwechslung können wir in unsere Bilder bringen, wenn wir z. B. einige der zufällig gezeichneten Quadrate 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.

Ob eine Zahl gerade ist, erkennen wir daran, dass der Rest beim Teilen durch 2 Null ist. In Beispiel 5 verwenden wir daher die Bedingung „*i* modulo 2 gleich Null“. Der Operator Modulo ist in Processing das %-Zeichen. Für einen Vergleich wird ein doppeltes Gleichheitszeichen verwendet, da das einfache Gleichheitszeichen schon für die Wertzuweisung bei einer Variablen belegt ist. 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 }
```

Beispiel 5: Verzweigung, die zwischen Zeile 1 und 2 in Beispiel 4 eingefügt werden kann

**Aufgabe 14:** Erweitern Sie Ihr Programm aus Aufgabe 11 um verschiedene Verzweigungen.

Beispielsweise können abhängig von der Zählvariablen oder einem Zufallswert, die Formen oder die Farben variiert werden.



**Aufgabe 15:** Verändern Sie Ihr Programm aus Aufgabe 12b oder 12c so, dass sich die Farben in der Blumenreihe abwechseln. Wechseln Sie zunächst zwischen zwei, anschließend zwischen drei Farben.

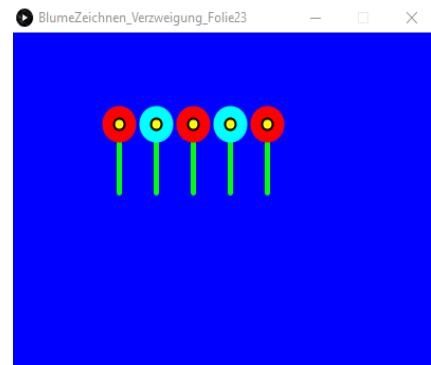


Abbildung 11: exemplarische Ausgabe zu Aufgabe 15

**Aufgabe 16:**

- a) Erstellen Sie ein Programm, das einen Turm aus Quadraten zeichnet. Die Farben der Quadrate sollen dabei wechseln (s. Abbildung 12).
- b) Erstellen Sie ein Programm, das eine dreifarbige Raupe zeichnet (s. Abbildung 13). Ergänzen sie gerne weitere Details wie z. B. Fühler und Augen.

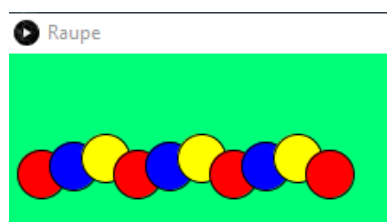


Abbildung 13: Dreifarbige Raupe

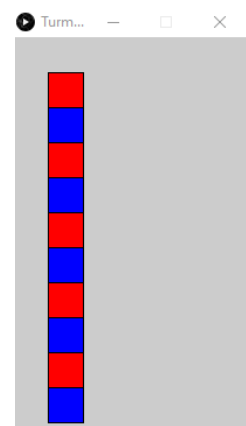


Abbildung 12: Turm aus roten und blauen Quadraten.

**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.

## Interaktion mit dem Anwender

Bislang werden alle Anweisungen in unserem Programm direkt beim Start ausgeführt und danach erhalten wir das fertige Bild, das wir nur durch ein erneutes Starten des Programms neu zeichnen können. Schön wäre es, wenn der Anwender das Programm beeinflussen kann, während es läuft. Dazu müssen wir das Grundgerüst unserer Programme erweitern.

### Grundgerüst eines Processing-Programms mit Interaktionsmöglichkeiten

Neben der Methode `void setup(){...}` können weitere vordefinierte Methoden in das Programm aufgenommen werden, die z. B. bei erneutem Zeichnen des Fensters, einem Mausklick oder einer Tastatureingabe ausgeführt werden. Die Methoden haben dabei immer die Form `void methodenname() {...}`. In die geschweiften Klammern wird der Programmcode eingefügt, der ausgeführt werden soll, wenn das entsprechende Ereignis eintritt. Ein Programm, das entsprechend aufgebaut ist, zeigt Beispiel 6. Es gibt somit Methoden, die selbst mit Inhalt füllen können und andere, die nur zum Aufrufen und Ausführen zur Verfügung stehen, wie z. B. `rect`, `fill` usw.

Wir ergänzen zunächst die Methode `void draw(){...}` in unserem Programm. Diese Methode wird nicht nur einmal beim Starten des Programms ausgeführt, sondern in kurzen regelmäßigen Abständen, solange das Programm läuft, so dass die Anzeige ständig aktualisiert wird. Hier könnten wir also Anweisungen platzieren, die in einer Endlosschleife immer neu ausgeführt werden sollen. Auch wenn wir hier nichts in die geschweiften Klammern schreiben, sorgt diese Methode dafür, dass das Fenster regelmäßig neu gezeichnet wird und damit Veränderungen, die wir in weiteren Methoden vornehmen, sichtbar werden. Deshalb ist es wichtig, dass unser Programm diese Methode enthält, auch wenn wir sie im Moment noch nicht mit Inhalt füllen.



## Reaktion auf die Maus

Beispiel 6 enthält neben den Methoden *setup* und *draw* die Methode *void mouseClicked(){...}*. Diese wird immer dann aktiviert, wenn der Anwender mit einer Maustaste einen Mausklick innerhalb des Programmfensters ausführt. Das heißt, der Inhalt der geschweiften Klammern dieser Methode, wird nur dann ausgeführt, wenn der Anwender eine Maustaste betätigt hat. In Beispiel 6 wird in diesem Fall ein Quadrat gezeichnet.

```
1 void setup() { //diese Methode wird nur einmalig beim Programmstart ausgeführt
2   size(400, 300);
3   background(255, 255, 0);
4   fill(255, 0, 0);
5 }
6 void draw() {} //diese Methode wird immer wieder von vorne durchlaufen
7 void mouseClicked() { //diese Methode wird jedes Mal ausgeführt, wenn eine Maustaste gedrückt
                        //wurde
8   square(mouseX, mouseY, 30); //mouseX und mouseY liefern die aktuelle Position der Maus
9 }
```

### Beispiel 6: Reaktion auf Mausklick

Um noch besser mit dem Anwender interagieren zu können, stellt Processing uns verschiedene Systemvariablen zur Verfügung. Das Programm in Beispiel 6 verwendet die Systemvariablen *mouseX* und *mouseY*, um bei jedem Mausklick ein rotes Quadrat an die aktuelle Position der Maus zu zeichnen. Weitere Beispiele für Systemvariablen im Zusammenhang mit der Maus zeigt Tabelle 3.

Systemvariable	Bedeutung
<i>mouseX</i>	enthält die aktuelle x-Koordinate der Maus
<i>mouseY</i>	enthält die aktuelle y-Koordinate der Maus
<i>pmouseX</i>	enthält die vorherige x-Koordinate der Maus
<i>pmouseY</i>	enthält die vorherige y-Koordinate der Maus
<i>mouseButton</i>	enthält die Werte LEFT, RIGHT oder CENTER, je nachdem welche Maustaste gedrückt wurde

Tabelle 3: Systemvariablen zum Zustand der Maus

Neben *mouseClicked()* gibt es noch weitere Methoden, die auf Mausereignisse reagieren. Einen Überblick gibt Tabelle 4. Ausführliche Erläuterungen finden Sie in der Referenz<sup>3</sup>.

Methode	wird ausgeführt, ...
<i>mousePressed()</i>	sobald eine Maustaste gedrückt wird, unabhängig vom Zeitpunkt des Loslassens.
<i>mouseReleased()</i>	wenn eine Maustaste wieder losgelassen wird.
<i>mouseDragged()</i>	wenn die Maus mit gedrückter Maustaste bewegt wird.
<i>mouseMoved()</i>	wenn die Maus ohne gedrückte Maustaste bewegt wird.

Tabelle 4: Methoden für die Reaktion auf Ereignisse, die von der Maus ausgelöst werden

<sup>3</sup> Ben Fry & Casey Reas. Processing – Reference. <https://processing.org/reference/> [letzter Zugriff: 10.04.2025]

**Aufgabe 17:** Erweitern Sie das Programm in Beispiel 6 so, dass beim Drücken der linken Maustaste ein Quadrat und beim Drücken der rechten Maustaste ein Kreis an die Position der Maus gezeichnet wird.

**Aufgabe 18:** In Aufgabe 8 haben Sie ein Programm erstellt, das beim Starten eine Blume an eine festgelegte Position zeichnet. Erweitern Sie das Programm so, dass an der Position, an die mit der Maus angeklickt wird, jeweils eine weitere Blume gezeichnet wird.

**Aufgabe 19:** Implementieren Sie ein Programm, das eine Spur des Mauszeigers auf dem Bildschirm hinterlässt, wenn der Anwender die Maus mit gedrückter Maustaste über den Bildschirm bewegt.

*Beispiel: Rechtecke mit der Maus zeichnen*

Betrachten wir nun als weiteres Beispiel ein Programm, das es dem Anwender erlaubt, Rechtecke mit der Maus auf den Bildschirm zu zeichnen bzw. diese aufzuziehen. Die Maustaste wird dazu an der Position der linken, oberen Ecke gedrückt und die Maus dann mit weiterhin gedrückter Maustaste in die rechte, untere Ecke des Rechtecks gezogen und dort losgelassen. Wir benötigen dazu zwei Variablen, in denen wir uns die Startposition für das Rechteck merken. Daher werden in Beispiel 7 in Zeile 1 und Zeile 2 eine Variable `x1` und eine Variable `y1` für die x- und für die y-Koordinate der Startposition angelegt. Sobald die Maustaste gedrückt wird, sorgt die Methode `mousePressed()` in Zeile 13 bis 16 dafür, dass in den Variablen `x1` und `y1` die aktuelle Position der Maus gespeichert wird. Da die Methode nach dem Drücken nur einmal ausgeführt wird, bleiben die Werte bis zum Loslassen und erneuten Drücken einer Maustaste unverändert. Das Zeichnen eines Rechtecks zwischen Startposition als linker, oberer Ecke und aktueller Mausposition als rechter, unterer Ecke, geschieht nun innerhalb der Methode `draw()`. Da diese Methode immer wieder ausgeführt wird, passt sich das Rechteck kontinuierlich an die aktuelle Position der Maus an. Damit nur Rechtecke gezeichnet werden, solange der Anwender die Maustaste gedrückt hält, fragen wir in Zeile 9 mithilfe der Systemvariablen `mousePressed` noch ab, ob die Maustaste gerade gedrückt wird. Nur dann wird der Befehl zum Zeichnen des Rechtecks in Zeile 10 ausgeführt. Die Breite und Höhe des Rechtecks ergeben sich dabei aus dem Betrag der Differenz zwischen der x-Koordinate der Startposition (`x1`) und der x-Koordinate der aktuellen Position (`mouseX`) bzw. zwischen der y-Koordinate der Startposition (`y1`) und der y-Koordinate der aktuellen Position (`mouseY`). Abbildung 14 veranschaulicht diese Rechnung an einem Beispiel.

**Aufgabe 20:** Testen Sie, ob und ggf. wie sich das Verhalten des Programms in Beispiel 7 verändert, wenn Sie ...

- a) das Zeichnen des Rechtecks in die Methode `mouseDragged()` verlagern und die Methode `draw()` leer bleibt.
- b) das Zeichnen des Rechtecks in die Methode `mouseReleased()` verlagern und die Methode `draw()` leer bleibt.
- c) Zwischen Zeile 14 und Zeile 15 die Anweisung `background(255, 255, 0);` einfügen.

**Aufgabe 21:** Ändern Sie das Programm aus Beispiel 7 so, dass der Anwender einen Kreis aufziehen kann. Die Startposition legt dabei den Mittelpunkt des Kreises fest und die aktuelle Position der Maus, soll sich immer am Kreisrand befinden.

```

1  int x1;
2  int y1;

3  void setup(){
4      size(400, 300);
5      background(255, 255, 0);
6      fill(255, 0, 0);
7  }

8  void draw(){
9      if(mousePressed){
10         rect(x1, y1, abs(mouseX - x1),
11             abs(mouseY - y1));
12     }

13 void mousePressed(){
14     x1 = mouseX;
15     y1 = mouseY;
16 }

```

Beispiel 7: Aufziehen von Rechtecken

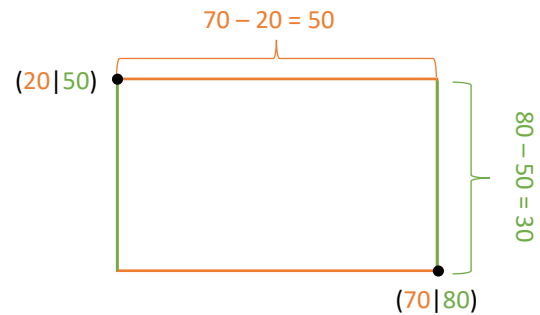


Abbildung 14: Berechnung der Höhe und Breite eines Rechtecks anhand der Koordinaten zweier gegenüberliegender Ecken

## Exkurs: Lokale und globale Variablen

Vielleicht ist Ihnen aufgefallen, dass wir die Variablen in Beispiel 7 außerhalb der Methoden in den ersten beiden Zeilen des Programms definiert haben, während wir die Variablen in den vorherigen Beispielen meist innerhalb der Methoden, dort wo wir sie benötigen, angelegt haben. Es stellt sich somit die Frage nach dem Unterschied.

Die Werte der Variablen `x1` und `y1` in Beispiel 7 werden in der Methode `mousePressed()` verändert und in der Methode `draw()` verwendet. Die Werte der Variablen werden also nicht nur während der Ausführung einer Methode, sondern während der gesamten Laufzeit des Programms benötigt. Variablen, die außerhalb der Methoden definiert werden und während der gesamten Laufzeit des Programms zur Verfügung stehen, bezeichnet man als **globale Variablen**. Ob sie oben, unten oder zwischen den Methoden definiert werden, ist dabei eigentlich egal. Damit wir den Überblick leichter behalten, sollten wir uns aber angewöhnen, sie immer an den Anfang des Programms zu setzen.

Eine Variable, die innerhalb einer Methode definiert wird, bezeichnet man hingegen als **lokale Variable**. Diese steht nur so lange zur Verfügung, bis die Ausführung der Methode beendet ist. Danach wird der Speicher für andere Zwecke wieder freigegeben. Eine solche lokale Variable haben wir in den Beispielen 2 und 3 als Zählvariable für die Schleifen verwendet. Wenn wir den Quellcode aus Beispiel 2 in die Methode `setup()` einfügen, steht die Variable `i` nach dem Beenden der `setup`-Methode nicht mehr zur Verfügung. In Beispiel 3 steht die Variable `i` bereits nach dem Beenden der `for`-Schleife nicht mehr zur Verfügung, da sie speziell für die Schleife definiert wurde.

Beim Entwerfen eines Programms sollte man also für jede Variable überlegen, ob eine Definition als globale oder als lokale Variable sinnvoller ist. Dadurch wird das Programm übersichtlicher und der Speicherplatz für lokale Variablen, die nicht mehr benötigt werden, kann freigegeben werden.

**Aufgabe 22:** Im Ordner zu Aufgabe 22 finden Sie fünf kurze Programmbeispiele. Beispiel 1a und 1b sowie 2a und 2b unterscheiden sich jeweils in der Definition der verwendeten Variable als globale bzw. lokale Variable.

- a) Erläutern Sie, wie sich die Verwendung einer lokalen bzw. globalen Variablen auf das Verhalten des Programms auswirkt.
- b) Begründen Sie für Beispiel 1 und 2 jeweils, ob die Verwendung einer lokalen oder einer globalen Variablen sinnvoller ist.
- c) Erläutern Sie, weshalb in Beispiel 3 alle gezeichneten Quadrate die gleiche Größe haben.  
Verändern Sie das Programm so, dass die Quadrate jeweils um 10 Einheiten größer werden.

### Simulation von Buttons

In Aufgabe 19, 20 und 21 müssen die Farbe und die Dicke der gezeichneten Linien vom Programmierenden festgelegt werden. Da wir auf Mausklicks reagieren können, können wir aber auch die Funktion von Buttons simulieren, um dem Anwender weitere Eingaben zu ermöglichen. Um einen Button zu simulieren, können wir einfach ein Rechteck zeichnen. Erfolgt ein Mausklick, kann mithilfe der Koordinaten des Rechtecks und der aktuellen Koordinaten der Maus überprüft werden, ob der Mausklick innerhalb des Button-Rechtecks erfolgt ist. Um zwischen den Linienfarben Schwarz, Rot und Blau zu wählen, könnten wir beispielsweise einfach ein schwarzes, ein rotes und ein blaues Rechteck zeichnen, das der Anwender zur Auswahl der jeweiligen Farbe anklicken kann. Um die Stiftdicke über einen Button zu verringern oder zu erhöhen, wäre aber vielleicht eine Beschriftung der Rechtecke hilfreich. Dazu steht uns die Methode `text()` zur Verfügung. Dieser Methode übergeben wir einen Text und eine Position, an der dieser Text erscheinen soll. Als Schriftfarbe wird die Farbe, die mit der Methode `fill()` festgelegt wurde, verwendet. Die Schriftgröße kann mit der Methode `textSize()` festgelegt werden. In Beispiel 8 wird in der Methode `setup()` ein graues Rechteck mit der Beschriftung „dicker“ erzeugt.

```
17 void setup() {  
18   size(400, 300);  
19   background(255, 255, 0);  
20   fill(200, 200, 200);  
21   rect(10, 10, 50, 30);  
22   fill(0,0,0);  
23   text("dicker", 15, 30);  
24 }
```

Beispiel 8: Zeichnen eines Buttons

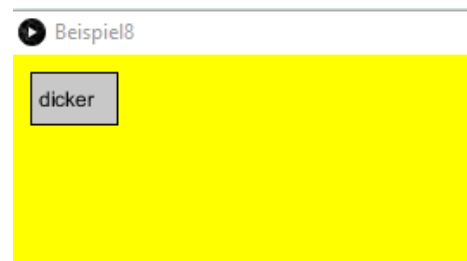


Abbildung 15: Programmfenster zu Beispiel 8

**Aufgabe 23:** Erweitern Sie Ihr Programm aus Aufgabe 19 um zwei Buttons, mit denen man die Dicke der gezeichneten Linie regulieren kann. Erzeugen Sie dazu zwei Rechtecke mit geeigneter Beschriftung. Beim Anklicken des einen Rechtecks soll die Dicke der Linie erhöht, beim Anklicken des anderen Rechtecks die Dicke der Linie verringert werden.

**Vorsicht:** Achten Sie darauf, dass die Dicke nicht negativ wird.

### Eingaben über die Tastatur

Im Folgenden schauen wir uns an, wie wir auf das Drücken einer Taste auf der Tastatur reagieren können, so dass auch auf diese Art Eingaben gemacht werden können.

In Aufgabe 17 wurde über die linke bzw. rechte Maustaste ausgewählt, ob ein Quadrat oder ein Kreis gezeichnet wird. Alternativ können wir unser Programm so erstellen, dass der Anwender vor dem Zeichnen mithilfe der Tasten auswählt, welche Figur gezeichnet werden soll, so dass bei einem Mausklick mit der linken Maustaste dann die passende Figur erscheint. In Beispiel 9 erfolgt die Auswahl über die Tasten 'q' für Quadrat bzw. 'k' für Kreis. Die Methode `keyPressed()`, wird immer

dann einmalig ausgeführt wird, wenn eine Taste auf der Tastatur gedrückt wurde<sup>4</sup>. Außerdem benötigen wir die Systemvariable `key`, die das Zeichen der zuletzt gedrückten Taste enthält. Das Programm in Beispiel 9 speichert die zuletzt gedrückte Taste in der Variablen `figur`, indem in Zeile 19 innerhalb der Methode `keyPressed()` der Variablen `figur` der Wert der Systemvariablen `key` zugewiesen wird. Innerhalb der Methode `mouseClicked()` wird nun ein Quadrat gezeichnet, wenn die Bedingung in Zeile 10 erfüllt ist, also der Anwender vorher Taste 'q' für Quadrat gedrückt hat, oder ein Kreis, wenn die Bedingung in Zeile 13 erfüllt ist, weil der Anwender Taste 'k' gedrückt hat.

```
1 char figur = 'x';
2 void setup(){
3     size(400, 300);
4     background(255, 255, 0);
5     fill(255, 0, 0);
6 }
7 void draw(){}
8
9 void mouseClicked(){
10     if(figur == 'q'){
11         square(mouseX, mouseY, 30);
12     }
13     if(figur == 'k'){
14         circle(mouseX, mouseY, 30);
15     }
16 }
17 void keyPressed(){
18     if(key == 'q' || key == 'k'){
19         figur = key;
20     }
21 }
```

#### Beispiel 9: Reaktion auf Tastatureingaben

Das Programm in Beispiel 9 würde auch funktionieren, wenn wir in den Bedingungen innerhalb der Methode `mouseClicked()` direkt die Systemvariable `key` abfragen würden, vorausgesetzt es werden nach 'q' oder 'r' keine weiteren Tasten gedrückt. Das Vorgehen, in der Methode `keyPressed()`, den Wert zunächst in einer Variablen zu speichern, ist jedoch hilfreich, wenn wir dem Anwender noch andere Eingaben über die Tastatur erlauben wollen. Deshalb wird der Wert von `key` nur in der Variablen `figur` gespeichert, wenn die Taste 'q' oder die Taste 'r' gedrückt wurde.

Wir könnten dem Anwender z. B. zusätzlich die Steuerung der Dicke der Umrandung über die Tastatur anbieten. Hierfür gibt es verschiedene Möglichkeiten. Wir könnten z. B. fünf Werte für die Dicke zur Verfügung stellen, die mithilfe der Tasten 1 bis 5 direkt ausgewählt werden. Wir könnten die Dicke aber auch ähnlich wie bei den beiden Buttons bei Drücken der Plus-Taste um 1 erhöhen und bei Drücken der Minus-Taste um 1 verringern, sofern sie aktuell größer als 1 ist. Wir schauen uns erst einmal die Umsetzung der zweiten Variante an.

Wir benötigen eine globale Variable `dicke`, in der wir uns die aktuelle Dicke merken. Diese ist zu

<sup>4</sup> Alternativ können wir die Methode `keyTyped()` verwenden. Anders als `keyPressed()` wird die Methode `keyTyped()` nicht ausgeführt, wenn Tasten gedrückt werden, die Steuerungsfunktionen übernehmen, also keine Zeichen codieren. Das ist hilfreich, wenn z. B. Klein- und Großbuchstaben unterschieden werden sollen.

Beginn 1, deshalb weisen wir der Variablen `dicke` zu Beginn den Wert 1 zu. Schauen wir uns zunächst Beispiel 10 an. Die Methode `keyPressed()` enthält nun drei Bedingungen: in Zeile 4, Zeile 7 und Zeile 11. Die gedrückte Taste speichern wir nur in der Variablen `figur`, wenn es sich um die Taste 'q' oder 'k' handelt. Wenn die Plus-Taste gedrückt wurde, wird die Variable `dicke` um 1 erhöht und die Methode `strokeWeight()` zum Setzen der Dicke der Umrandung mit dem neuen Wert als Parameter ausgeführt. Wenn die Minus-Taste gedrückt wurde und die Variable `dicke` nicht schon beim Wert 1 angelangt ist, wird der Wert um 1 verringert und ebenfalls die Methode `strokeWeight()` mit dem aktuellen Wert ausgeführt.

```
1  int dicke = 1;
2  void keyPressed() {
3
4      if(key == 'q' || key == 'k') {
5          figur = key;
6      }
7      if(key == '+') {
8          dicke++;
9          strokeWeight(dicke);
10     }
11     if(key == '-' && dicke > 1) {
12         dicke --;
13         strokeWeight(dicke);
14     }
15 }
```

*Beispiel 10: Unterscheiden der Tastatureingaben mithilfe mehrerer if-Konstruktionen*

```
1  int dicke = 1;
2  void keyPressed() {
3      switch(key) {
4          case 'k':
5          case 'q': figur = key;
6                      break;
7          case '+': dicke++;
8                      strokeWeight(dicke);
9                      break;
10
11         case '-': if(dicke > 1) {
12                     dicke --;
13                     strokeWeight(dicke);
14                 }
15                 break;
16     }
17 }
```

*Beispiel 11: Unterscheiden der Tastatureingaben mithilfe eines switch-case Blocks*

Je mehr Tasten wir in der Methode `keyPressed()` berücksichtigen wollen, desto mehr Bedingungen kommen zusammen. Deshalb lohnt es sich, dafür eine etwas einfachere Schreibweise einzuführen, die in der Variante rechts in Beispiel 11 zu sehen ist. Hinter dem Befehl `switch` in Zeile 3 steht in Klammern die Variable, die in jeder Bedingung überprüft werden soll. Jeder Wert, mit dem die Variable verglichen werden soll, steht hinter dem Befehl `case`. Wir haben also den Fall, dass die Systemvariable `key` die Werte 'k', 'q', '+' oder '-' annimmt. Hinter dem Doppelpunkt stehen dann die Anweisungen, die in dem jeweiligen Fall ausgeführt werden sollen. Es fällt auf, dass in Zeile 4 für den Fall 'k' kein Befehl angegeben ist. Das liegt daran, dass ab dem Fall, der zutrifft, auch die Anweisungen, die bei allen nachfolgenden Fällen stehen, ausgeführt werden, bis die Anweisung `break` gefunden wird. Da für den Fall 'k' und 'q' die gleiche Aktion ausgeführt werden soll, muss diese nur einmal hinter dem zweiten Fall angegeben werden. Da bei '+' und '-' unterschiedliche Anweisungen ausgeführt werden sollen, muss in Zeile 9 und Zeile 15 die Anweisung `break` stehen.

**Aufgabe 24:** Erweitern Sie das Programm aus Beispiel 10 oder 11 um weitere Optionen, die der Anwender mithilfe der Tasten auswählen kann, z. B:

- weitere Figuren, die gezeichnet werden können
- ob die Form gefüllt sein soll oder nicht
- die Farbe der Füllung (z. B. können fünf Farben zur Auswahl mit Tasten belegt werden)
- die Größe der Figur

## Eingaben über ein Eingabefeld

Spätestens bei der Wahl der Farbe oder Größe in Aufgabe 24 fällt auf, dass es relativ umständlich ist, komplexere Tastatureingaben, die aus mehr als einem Zeichen bestehen, mithilfe der Methode `keyPressed()` zu erfassen. Wie wir aus einzelnen Zeichen eine Zeichenkette aufbauen können, müssen wir uns erst noch anschauen. Aber man kann sich vorstellen, dass dies zum Erfassen einer Tastatureingabe, die aus mehr als einem Zeichen besteht, relativ mühselig ist. Wesentlich einfacher ist es, dem Anwender für längere Eingaben ein Eingabefeld anzubieten.

Beispiel 12 zeigt, wie der Anwender über ein Eingabefeld die Größe des zu zeichnenden Quadrats eingeben kann. Um die entsprechende Methode zum Anzeigen des Eingabefeldes verwenden zu können, müssen wir in die erste Zeile unseres Programms den folgenden Befehl einfügen:

```
import static javax.swing.JOptionPane.*;
```

Nun können wir in einer unserer Methoden z. B. `draw()`, `mouseClicked()` oder `keyPressed()` die Methode `showInputDialog()` aufrufen (s. Zeile 9). Als Parameter wird der Text übergeben, der dem Anwender als Erklärung oder Aufforderung zu dem Eingabefeld angezeigt werden soll. Der Rückgabewert der Methode ist die Eingabe des Anwenders als Zeichenkette. Mit der folgenden Codezeile würde beispielsweise die Eingabe des Anwenders in der Variablen `antwort` vom Typ Zeichenkette gespeichert:

```
String antwort = showInputDialog("Wie heiß du?");
```

Wenn der Anwender eine Zahl eingeben soll, kann diese ggf. auch in eine Ganzzahl oder Gleitkommazahl umgewandelt werden. Dies geschieht in Beispiel 12 in Zeile 9 mithilfe der Methode `Integer.parseInt()`. Die Breite, die der Anwender eingegeben hat, wird in der Variablen `breite` gespeichert und dann in Zeile 10 in der Methode zum Zeichnen des Quadrates verwendet.

```
1  import static javax.swing.JOptionPane.*;
2  void setup(){
3      size(400, 300);
4      background(255, 255, 0);
5      fill(255, 0, 0);
6  }
7  void draw(){}
8  void mouseClicked(){
9      int breite = Integer.parseInt(showInputDialog("Bitte gib die Breite
              des Quadrates ein!"));
10     square(mouseX, mouseY, breite);
11 }
```

*Beispiel 12: Benutzereingaben über ein Textfeld*

**Aufgabe 25:** Erweitern Sie eines Ihrer Programme aus Aufgabe 16 so, dass der Anwender über ein Eingabefeld die Anzahl der Quadrate bzw. Kreise bestimmen kann, aus denen der Turm bzw. die Raupe bestehen sollen.



## Projekte

Sie haben nun verschiedene Konzepte kennengelernt. Verwenden Sie diese, um ein etwas umfangreicheres Programm selbständig zu erstellen.

**Projektidee 1:** Kombinieren Sie das Gelernte zu einem etwas komplexeren Zeichenprogramm, das weitere Funktionen enthält. Folgende Funktionen könnten beispielsweise noch ergänzt werden:

- Ein Radiergummi, mit dem Teile der Zeichenfläche wieder gelöscht werden können.
- Vollständiges Löschen der bisherigen Zeichnung
- Weitere Buttons zur Auswahl der Farben, Formen oder anderer Optionen
- Mit der Maus angeklickte Positionen werden mit Linien verbunden, so dass ein Polygon entsteht.

**Projektidee 2:** Verwenden Sie die Zeichenfunktionen von Processing, um ein Pong-Spiel zu erstellen. Zwei schwarze Rechtecke, dienen als Schläger. Der linke Schläger kann z. B. mithilfe zweier Tasten auf der Tastatur hoch und runter bewegt werden, der linke Schläger mithilfe der linken und der rechten Maustaste. Die Spielenden müssen die Schläger so positionieren, dass der Ball vom Schläger abprallen kann. Fliegt der Ball aus dem linken oder rechten Spielfeldrand, hat der jeweilige Spielende, der den Ball nicht aufhalten konnte, verloren. Am oberen und unteren Rand prallt der Ball ab.

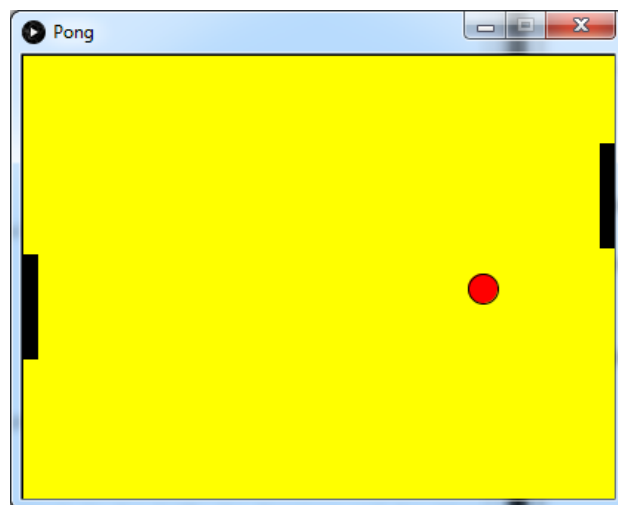


Abbildung 16: Exemplarisches Programmfenster zur Projektidee Pong-Spiel

Ergänzen Sie Ihr Spiel um weitere

Funktionalitäten. Wünschenswert wäre beispielsweise eine Anzeige des Punktestands.

**Hinweis:** Damit der Ball nicht flackert und sich nicht zu schnell bewegt, kann es hilfreich sein mithilfe der Methode `frameRate()` die Anzahl der pro Sekunde gezeichneten Bilder herunterzusetzen.

Probieren Sie z. B. `frameRate(4)`; Diese Anweisung muss einmalig zu Beginn in der Methode `setup()` ausgeführt werden.

**Projektidee 3:** Gestalten Sie ein Quiz mit mehreren Fragen nach dem Prinzip von „Wer wird Millionär?“. Dem Anwender wird jeweils eine Frage mit 4 Antwortmöglichkeiten angezeigt. Die Auswahl der Antwort kann über Buttons oder über die Tastatur erfolgen.

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.