

Eigene Methoden in Processing¹

Wozu benötigen wir eigene Methoden?

Bislang haben wir in unseren Processing-Programmen Methoden auf zwei verschiedene Arten verwendet. Die Methoden *setup()*, *draw()*, *mouseClicked()*, *keyPressed()* usw. geben unserem Programm eine Grundstruktur und werden von Processing bei den jeweiligen Ereignissen ausgeführt. Die Namen der Methoden sind vorgegeben. Den Inhalt der Methoden bestimmen wir beim Programmieren. Andere Methoden, wie z. B. *fill()*, *rect()*, *random()* usw. werden ebenfalls von Processing bereitgestellt und können von uns innerhalb einer der Methoden der ersten Sorte aufgerufen werden, um die entsprechende Aktion ausführen zu lassen. Manche Funktionen haben Sie vielleicht schon vermisst. Zum Beispiel eine Methode, die prüft, ob es sich bei einem Zeichen um eine Ziffer, einen Groß- oder einen Kleinbuchstaben handelt. Oder eine Methode, mit der Sie eine Figur, z. B. einen Schneemann, die sie aus vielen verschiedenen Formen gezeichnet haben, an anderen Positionen erneut zeichnen können. Oder eine Methode, die überprüft, ob sich der Mauszeiger innerhalb eines bestimmten Bereichs befindet. Im Folgenden schauen wir uns daher an, wie wir solche Methoden selbst erstellen können.

Eine Methode zu erstellen lohnt sich immer dann, wenn man die Lösung eines Teilproblems häufiger benötigt. Anstatt die Lösung, die ja in der Regel aus mehreren Anweisungen und ggf. auch Schleifen und Verzweigungen besteht, jedes Mal neu aufzuschreiben, fasst man den entsprechenden Teil in einer Methode zusammen. Diese kann dann wie andere Processing-Methode jedes Mal aufgerufen werden, wenn man die entsprechende Funktion benötigt. Das hat außerdem den Vorteil, dass man eine Änderung nur einmal in der Methode vornehmen muss, wenn man z. B. später noch einen Fehler oder eine Verbesserungsmöglichkeit entdeckt. Bei komplexeren Problemen kann das Zusammenfassen von Teillösungen auch einfach der Strukturierung dienen. Das gesamte Programm wird dann übersichtlicher.

Bei den Methoden können wir drei Arten unterscheiden. Im einfachsten Fall führt der Aufruf einer Methode einfach nur zur Ausführung einer bestimmten Anweisung oder Anweisungsfolge. Ein Beispiel für eine solche Methode wäre *noFill()* oder auch die Methoden *setup()*, *draw()* usw. Dann gibt es Methoden, denen Werte als Parameter übergeben werden müssen. Die Methode *rect()* beispielsweise benötigt die x- und die y- Koordinate sowie die Höhe und Breite des gewünschten Rechtecks als Parameter. Die dritte Variante führt nicht nur zur Ausführung von Anweisungen, sondern liefert auch einen Wert zurück. Die Methode *length()* für Zeichenketten beispielsweise liefert uns die Länge der Zeichenkette als Ganzzahl. Die zweite und dritte Variante lassen sich natürlich auch kombinieren. So erhält die Methode *random()* die Grenzen, zwischen denen eine Zufallszahl erzeugt werden soll, als Parameter. Und als Rückgabewert erhalten wir die Zufallszahl als Gleitkommazahl.

¹ Die Programmierumgebung Processing wurde 2001 von Ben Fry und Casey Reas initiiert. Nähere Informationen finden Sie unter <https://processing.org/>

Methoden ohne Parameter und Rückgabe

Wir beginnen mit der einfachsten Form der Methoden, die lediglich eine Anweisungsfolge ausführt. Schauen wir uns als Beispiel ein Programm an, das jedes Mal, wenn der Anwender einen Mausklick ausführt, an die Position der Maus eine Blume zeichnet. Die Blume besteht aus mehreren geometrischen Formen, so dass wir das Zeichnen der Blume in eine Methode *blumeZeichnen()* auslagern. Wir können der Methode auch einen beliebigen anderen Namen geben, er sollte aber Aufschluss darüber geben, was die Methode macht. Ansonsten verliert man beim Programmieren schnell den Überblick.

```
1 void setup(){
2   size(400, 300);
3   background(0, 55, 255);
4 }
5 void draw(){
6 }
7 void mouseClicked(){
8   blumeZeichnen();    //Methodenaufruf
9 }
10 void blumeZeichnen() { //Definition der Methode
11   stroke(0, 255, 0);
12   line(mouseX, mouseY, mouseX, mouseY+60);
13   stroke(255, 0, 0);
14   fill(255, 0, 0);
15   circle(mouseX, mouseY, 30);
16   stroke(0, 0, 0);
17   fill(255, 255, 0);
18   circle(mouseX, mouseY, 10);
19 }
```

Beispiel 1: Erstellen und Verwenden einer Methode ohne Parameter und Rückgabewert

Die Methode *blumeZeichnen()* wird in Zeile 10 definiert. Wie auch bei den vorgegebenen Methoden, die wir mit Inhalt füllen, wird das Schlüsselwort `void` vorangestellt. Dieses steht immer dann vor dem Methodennamen, wenn die Methode keinen Rückgabewert hat. Hinter dem Methodennamen folgt zunächst ein rundes Klammerpaar. Da die Methode keine Parameter als Eingabe erhält, bleibt die Klammer leer. In den geschweiften Klammern steht schließlich, welche Befehle ausgeführt werden sollen, wenn die Methode aufgerufen wird (s. Zeile 11 bis 19).

Der Aufruf der Methode erfolgt in Zeile 8. Wie bei anderen Processing-Methoden schreiben wir dazu einfach den Methodennamen gefolgt von einem runden Klammerpaar und einem Semikolon als Abschluss auf.

Wir finden im ganzen Programm bis jetzt nur einen Aufruf der Methode *blumeZeichnen()*, so dass man die Zeilen 11 bis 18 natürlich auch direkt in die Methode *mouseClicked()* hätte schreiben können. Unser Programm soll aber noch wachsen und wir sehen bereits jetzt, dass wir durch die Zusammenfassung in einer eigenen Methode viel schneller einen Überblick über das Programm gewinnen.

Aufgabe 1: Ergänzen Sie in Beispiel 1 eine weitere Methode *hausZeichnen()*, die an die Position der Maus ein Haus zeichnet. Wenn der Anwender die linke Maustaste klickt, soll die Blume, beim Klicken der rechten Maustaste das Haus gezeichnet werden.

Methoden mit Parameter

Anstatt auf die Systemvariablen `mouseX` und `mouseY` zuzugreifen, können wir der Methode *blumeZeichnen()* die x- und y- Koordinate für die Position auch als Parameter übergeben. Dann haben wir die Möglichkeit gleich zu Beginn in der Methode *setup()* ein oder zwei Blumen an eine von uns vorgegebene Position zu zeichnen. Der Anwender kann dann später mithilfe der Maus weitere Blumen ergänzen. Für die Parameter müssen wir den Datentyp festlegen. Eine Ganzzahl ist hier ausreichend genau, so dass sich der Datentyp *int* anbietet.

```
1 void setup(){
2   size(400, 300);
3   background(0, 55, 255);
4   blumeZeichnen(150, 100);           //Methodenaufruf
5   blumeZeichnen(190, 100);           //Methodenaufruf
6 }
7 void draw(){
8 }
9 void mouseClicked(){
10  blumeZeichnen(mouseX, mouseY);      //Methodenaufruf
11 }
12 void blumeZeichnen(int xPos, int yPos){ //Definition der Methode
13  stroke(0, 255, 0);
14  line(xPos, yPos, xPos, yPos + 60);
15  stroke(255, 0, 0);
16  fill(255, 0, 0);
17  circle(xPos, yPos, 30);
18  stroke(0, 0, 0);
19  fill(255, 255, 0);
20  circle(xPos, yPos, 10);
21 }
```

Beispiel 2: Erstellen und Verwenden einer Methode mit Parametern

Bei der Definition der Methode in Zeile 12 stehen die Parameter der Methode in den runden Klammern. Dabei wird zuerst der Datentyp, hier *int*, und dann der Name des Parameters, hier *xPos* bzw. *yPos* angegeben. Überall wo innerhalb der Methodendefinition vorher die Systemvariablen `mouseX` und `mouseY` verwendet wurden, stehen nun die Parameternamen *xPos* bzw. *yPos*. Beim Aufruf der Methode müssen die entsprechenden Werte übergeben werden. Dies können entweder feste Werte sein, wie wir es in Zeile 4 und 5 sehen, oder Variablen. So werden in Zeile 10 beim Klicken der Maus weiterhin die Koordinaten der Maus zum Zeichnen verwendet, indem sie als Parameter übergeben werden. Die Parameter einer Methode kann man sich daher genauso wie eine Variable vorstellen. Beim Aufruf der Methode wird der entsprechende Wert in die Variable geschrieben. In Zeile 4 wird der Wert 150 in die Variable *xPos* geschrieben, in Zeile 10 wird die aktuelle x-Koordinate der Maus in die Variable *xPos* geschrieben. Jedes Mal, wenn innerhalb der Methodendefinition auf *xPos* zugegriffen wird, wird der entsprechende Wert ausgelesen.

Im Falle der Positionierung mit der Maus hat dies noch einen Vorteil. Wenn die Methode zum Zeichnen sehr viele geometrische Figuren enthält, könnte sich die Position der Maus während des Zeichnens ändern. Wird jedes Mal direkt auf die Systemvariable `mouseX` zugegriffen, könnte die Figur also durcheinandergeraten, wenn sich die Position der Maus und damit der Inhalt der Variablen zwischenzeitlich verändert. Wenn die Position hingegen wie in Zeile 12 beim Methodenaufruf übergeben wird, ist sie in der Variablen für den Parameter gespeichert und eine Mausbewegung hat keinen Einfluss mehr auf die Ausführung der Methode.

Aufgabe 2:

- Verändern Sie Ihre Methode `hausZeichnen()` aus Aufgabe 1 so, dass die Position des Hauses wie bei der Methode `blumeZeichnen()` mithilfe von Parametern übergeben wird. Ergänzen Sie in der Methode `setup()` einen Methodenaufruf der Methode `hausZeichnen()`, so dass gleich beim Programmstart ein Haus gezeichnet wird.
- Ergänzen Sie bei den Methoden `blumeZeichnen()` und `hausZeichnen()` jeweils einen weiteren Parameter für die Größe der Figur.

Methoden mit Rückgabewert

Wenn wir in Processing einen Button simulieren möchten, zeichnen wir ein Rechteck und müssen überprüfen, ob der Mausklick im Bereich dieses Rechtecks erfolgt ist. Wir müssen also für die x- und die y- Koordinate der Maus jeweils überprüfen, ob sie sich innerhalb der Grenzen des Rechtecks befinden. Das sind insgesamt vier Vergleiche. Wenn wir an mehr als einer Stelle wissen möchten, ob der Anwender den Button angeklickt hat, lohnt es sich auf jeden Fall, dafür eine Methode einzurichten. Unsere Methode sollte einen Wahrheitswert zurückliefern: `true`, wenn der Mausklick im Bereich des Buttons erfolgt ist und ansonsten `false`.

```
1 void setup(){
2   size(400, 300);
3   background(0, 55, 255);
4   fill(255, 0, 0);           //Button zeichnen
5   rect(30, 50, 80, 35);
6 }
7 void draw(){
8 }
9 void mouseClicked(){
10  if(roterButtonAngeklickt()){ //Aufruf der Methode
11    System.out.println("Treffer");
12  }else{
13    System.out.println("Kein Treffer");
14  }
15 }
16 boolean roterButtonAngeklickt(){ //Definition der Methode
17  if(mouseX > 30 && mouseX < 110 && mouseY > 50 && mouseY < 85){
18    return true; //gib true zurück
19  }else{
20    return false; //gib false zurück
21  }
22 }
```

Beispiel 3: Erstellen und Verwenden einer Methode mit Rückgabewert

Da die Methode einen Wahrheitswert zurückgibt, wurde in Zeile 16 bei der Methodendefinition das Schlüsselwort `void` durch den Datentyp des Rückgabewertes, hier also `boolean`, ersetzt. Die Rückgabe des entsprechenden Wertes erfolgt innerhalb der Methodendefinition mithilfe des Befehls `return`. So wird in Zeile 17 die Bedingung überprüft, ob sich die Mauskoordinaten im Bereich des roten Rechtecks, das in Zeile 5 gezeichnet wurde, befinden. Ist die Bedingung wahr, so wird in Zeile 18 `true` zurückgegeben, andernfalls in Zeile 20 der Wert `false`.

Der Aufruf der Methode erfolgt in Zeile 10. Da es sich um eine Methode handelt, die `true` oder `false` zurückgibt, kann sie die Bedingung für die Verzweigung ersetzen. Die Methode wird ausgeführt und anhand des Rückgabewertes erfolgt die Verzweigung in den `if`- oder `else`-Zweig. Damit das Programm nicht zu unübersichtlich wird, wird hier nur Treffer bzw. kein Treffer ausgegeben. Hier sind aber natürlich viele sinnvollere Reaktionen denkbar.

Aufgabe 3: Ergänzen Sie in Beispiel 3 einen weiteren Button (Rechteck) und eine entsprechende Methode, die überprüft, ob der Button angeklickt wurde. Statt der Textausgabe können Sie über die Buttons beispielsweise die Farbe des Hintergrunds wählen lassen.

Methoden mit Parametern und Rückgabewert

So richtig viel Arbeit haben wir uns mithilfe unserer Methoden für die Buttons noch nicht erspart, da wir für jeden Button eine neue Methode erstellen müssen. Das ändern wir jetzt, indem wir eine Methode erstellen, die für alle Buttons funktioniert. Dazu müssen wir der Methode allerdings die Position des Buttons und seine Höhe und Breite als Parameter übergeben. Also die Werte, die auch die Methode `rect()` zum Zeichnen des Rechtecks als Parameter erhalten hat. Dies macht es uns relativ einfach auch noch weitere Buttons zu ergänzen und auszuwerten.

```
1 void setup(){
2     size(400, 300);
3     background(0, 255, 0);
4     fill(255, 0, 0);          //roten Button zeichnen
5     rect(30, 50, 80, 35);
6     fill(0, 0, 255);         //blauen Button zeichnen
7     rect(130, 50, 80, 35);
8 }
9
9 void draw(){
10 }
11
11 void mouseClicked(){
12     if(buttonAngeklickt(30, 50, 80, 35)){
13         background(255, 0, 0);
14     }
15     if(buttonAngeklickt(130, 50, 80, 35)){
16         background(0, 0, 255);
17     }
18     //Da der Hintergrund die Buttons übermalt, müssen diese neu gezeichnet werden
19     fill(255, 0, 0);
20     rect(30, 50, 80, 35);
21     fill(0, 0, 255);
22     rect(130, 50, 80, 35);
23 }
```

```
24 boolean buttonAngeklickt(float x, float y, float breite, float hoehe){
25     if(mouseX > x && mouseX < x + breite &&
        mouseY > y && mouseY < y + hoehe){
26         return true;
27     }else{
28         return false;
29     }
30 }
```

Beispiel 4: Erstellen und Verwenden einer Methode mit Parametern und Rückgabewert

In Zeile 24 sehen wir, dass die Methoden *roterButtonAngeklickt()* und *blauerButtonAngeklickt()* durch eine Methode *buttonAngeklickt()* ersetzt wurden, die nun entsprechend der Methode *rect()* vier Parameter erwartet. Der Rückgabewert ist weiterhin vom Typ *boolean*.

In Zeile 12 wird die Methode *buttonAngeklickt()* mit den Parameterwerten für das rote Rechteck und in Zeile 15 mit den Parameterwertem für das blaue Rechteck aufgerufen. Beim Anklicken eines Buttons wird der Hintergrund in der jeweils angeklickten Farbe gefärbt. Da der Hintergrund alles übermalt, müssen anschließend die Rechtecke für die Buttons neu gezeichnet werden. Auch dafür würde sich daher das Erstellen einer Methode anbieten.

Aufgabe 4: Erweitern Sie das Programm aus Beispiel 4 um einen gelben und einen grünen Button (Rechteck). Beim Anklicken der Buttons soll die jeweilige Farbe als Hintergrund übernommen werden. Ergänzen Sie daher auch eine Methode für das Zeichnen der vier Buttons.

Aufgabe 5: Beim Bearbeiten von Zeichenketten ist es manchmal hilfreich zu wissen, um welche Art von Zeichen es sich handelt. Erstellen Sie für die folgenden Fragestellungen jeweils eine Methode, die ein Zeichen vom Typ *char* als Parameter erhält und genau dann *true* zurückgibt, wenn das Zeichen zu der jeweiligen Gruppe gehört.

- a) Ist das Zeichen eine Ziffer?
- b) Ist das Zeichen ein Vokal?
- c) Ist das Zeichen ein Kleinbuchstabe?
- d) Ist das Zeichen ein Großbuchstabe?
- e) Ist das Zeichen ein Buchstabe?

Aufgabe 6:

- a) Erstellen Sie eine Methode *ersetze()*, die als Parameter eine Zeichenkette *text* vom Typ *String*, eine Position *i* vom Typ *int* und ein Zeichen *c* vom Typ *char* erhält. Die Methode soll eine Kopie der Zeichenkette *text* zurückgeben, wobei das Zeichen an Position *i* durch das Zeichen *c* ersetzt wurde.
- b) Verwenden Sie die Methode, um bei einem Text, den der Anwender eingibt, jedes vierte Zeichen durch ein Fragezeichen zu ersetzen. Ist der Text danach noch lesbar? Wie sieht es aus, wenn gezielt alle Vokale ersetzt werden?
- c) Verwenden Sie die Methode, um bei einem Text, den der Anwender eingibt, alle Großbuchstaben durch die entsprechenden Kleinbuchstaben zu ersetzen.

Aufgabe 7: Wählen Sie ein Programm aus, das Sie bereits implementiert haben, das Ihnen jedoch relativ umfangreich und unübersichtlich erschienen ist. Strukturieren sie dieses Programm durch das Erstellen und Verwenden geeigneter Methoden.

Aufgabe 8: Das beiliegende Programm zu Aufgabe 8 simuliert einen Ball, der sich durch das Fenster bewegt. Dazu wird die aktuelle Position des Balls in zwei Variablen für die x und die y- Koordinate des Mittelpunktes gespeichert. Bei jedem Aufruf der Methode *draw()* wird die Position um fünf Einheiten in x- und y- Richtung verändert. Leider verlässt der Ball dabei irgendwann das Fenster. In den grafischen Programmiersprachen gibt es häufig eine Methode *pralle vom Rand ab*, mit der das Verlassen des Fensters verhindert werden kann.

Ergänzen Sie in dem Programm eine entsprechende Methode *pralleVomRandAb()* und rufen Sie diese an geeigneter Stelle in der Methode *draw()* auf, um den Ball am Verlassen des Fensters zu hindern.

Lizenz

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

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.