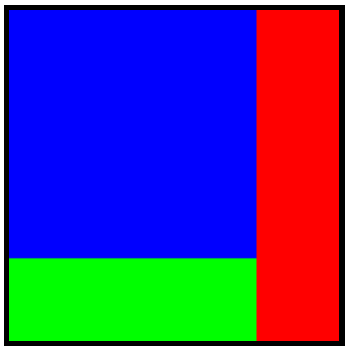
Aufgabenblatt 2 - Animationen und Schleifen

231
8 Z.

Aufgabe 1 – RGB-Boxen

Erzeugen Sie ein Bild, welches drei Boxen mit den Farben Rot, Grün und Blau enthält. So könnte Ihr Ergebnis aussehen:



- `noStroke()` - zeichnet alle darauffolgenden Formen ohne Umrandung
- `rect(x, y, w, h)` - zeichnet ein Rechteck mit der links-oberen Ecke an Position (x, y) mit einer Weite w und Höhe h.

232
5 Z.
int x

Aufgabe 2 – Summenformel

Berechnen Sie die Summe der Zahlen 1 bis 100 mithilfe einer Schleife. Verwenden Sie eine Variable `int sum;`, welche in jedem Schleifendurchlauf durch den Index erhöht werden soll. Lassen Sie sich in jedem Schleifendurchlauf `sum` in der Konsole ausgeben.

So sieht beispielsweise eine Schleife von 0 bis 20 (inkl.) aus, die in jedem Durchlauf den Index ausgibt. Tippen Sie den Code ab und führen Sie ihn aus.

```
1 for (int i = 0; i <= 20; i++) {
2    println(i);
3 }
```

So könnten die Konsolenausgabe Ihres Programms aussehen:

```
1 1
2 3
3 6
4 10
5 ...
6 4753
7 4851
8 4950
9 5050
```

233

6 Z.

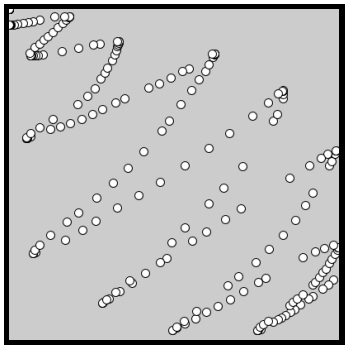
Aufgabe 3 – Zeichenprogramm

Schreiben Sie ein Programm, in welchem mit der Maus auf dem Zeichenbereich gemalt werden kann. Da dies ein interaktives Programm ist, müssen Sie mit `void setup(){ ... }` und `void draw(){ ... }` arbeiten. Zeichnen Sie an Position `mouseX` und `mouseY` einen Kreis der Größe 10.

So sieht ein Programm aus, dass in jedem `draw()`-Durchlauf die Position der Maus auf dem Zeichenfeld in der Konsole ausgibt. Probieren Sie es aus!

```
1 void setup(){
2    size(400, 400);
3 }
4 void draw() {
5    println("Mausposition: (" + mouseX + ", " + mouseY + ")");
6 }
```

Ist Ihr Programm richtig implementiert, erscheinen Kreise an den Stellen, an denen die Maus war:



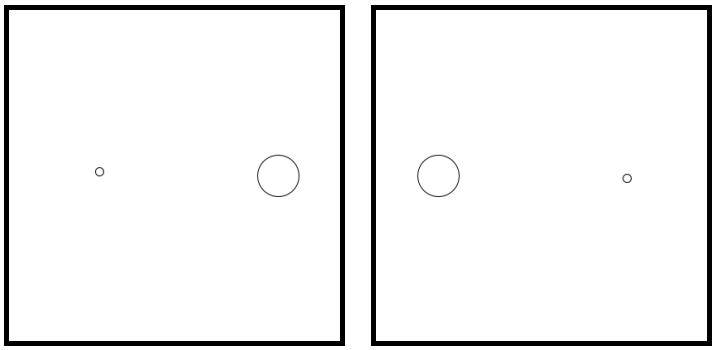
234

11 Z.

Aufgabe 4 – Ausweichen

Schreiben Sie ein Programm, in welchem ein Kreis der Maus ausweichen soll. Ist die Maus auf der linken Hälfte des Zeichenbereichs, soll der Kreis auf der rechten Seite erscheinen, und andersherum. Vergessen Sie nicht, den Hintergrund in der `draw`-Schleife zu zeichnen - sonst verschwinden vorher gezeichnete Kreise nicht.

Der kleine Kreis stellt in diesen Bildern die Position der Maus dar, und ist für Sie optional.



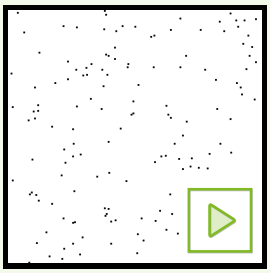
201

8 Z.
10 Z.
15 Z.
19 Z.
26 Z.

int x

Aufgabe 5 – Einstiegsaufgabe

5.1 Punkte im Raum

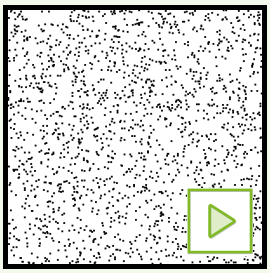


Mithilfe der `void draw()`-Funktion können Sie ihre Werke zum Leben erwecken lassen. Schreiben sie ein Programm, dass in jedem Durchlauf der `draw()`-Funktion einen Punkt an einer zufälligen Position auf den Zeichenbereich setzt. Verwenden Sie dazu die Methoden `random(i)` und `point(x, y)`.
Indem Sie den Hintergrund nicht immer wieder in der `draw()`-Funktion zeichnen, bleiben ältere Pixel erhalten.
Bei Bedarf können Sie auf diese Vorlage zurückgreifen:

```
1 void setup() {
2     // Einstellungen: Fenstergröße, Hintergrund, ...
3 }

4 void draw() {
5     // Zufällige Koordinaten wählen, Punkt zeichnen
6 }
```

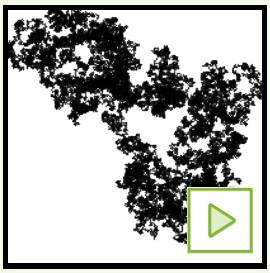
5.2 Mehr Punkte!



Derzeit erscheinen die Punkte relativ langsam im Zeichenbereich. Beschleunigen können Sie den Prozess, indem Sie in jeder Ausführung der `draw()`-Funktion nicht nur einen, sondern gleich mehrere Punkte zeichnen. **Verwenden Sie hierzu eine `for`-Schleife.**
Auch hier können Sie eine Vorlage verwenden, welche Sie in die `draw()`-Funktion einfügen - dabei müssen Sie die Anzahl maximaler Durchläufe an der richtigen Stelle einfügen:

```
1 for(int i = 0; i < maximaleDurchläufe; i++) {
2     // zu wiederholende Aktion
3 }
```

5.3 *Randomwalk*



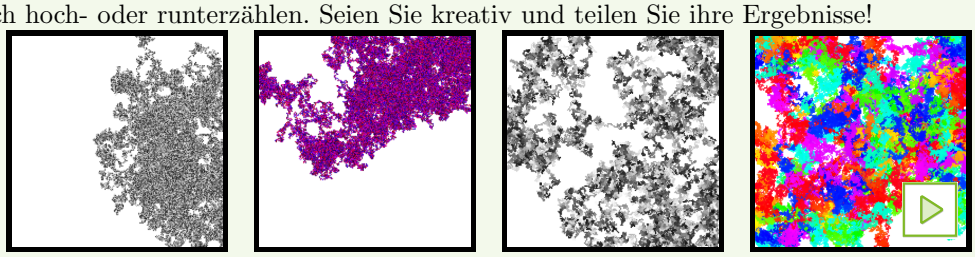
Sie sind nun in der Lage, mithilfe von Zufall und Schleifen ein bewegtes Bild zu erzeugen. Nun wollen wir einen *Randomwalk* erzeugen. Dies bezeichnet einen Pixel, der sich zufällig auf der Zeichenebene bewegt, und durch die hinterlassene Spur ein Bild erzeugt. In jedem `draw()`-Durchlauf soll sich der Pixel zufällig nach oben oder unten, und nach links oder rechts bewegen. Dabei soll es auch die Möglichkeit geben, dass sich der Pixel in einem Durchlauf auf der `x` bzw. `y`-Ebene nicht bewegt.
Hinweis: *Auch hier können Sie eine Schleife verwenden, um mehr Pixel pro* `draw()`*-Durchlauf zu erhalten.*

5.4 Absperrung

Falls Sie ihre Kreation mal für eine Weile beobachtet haben, ist Ihnen vielleicht aufgefallen, dass der Pixel sich irgendwann aus dem Bild bewegt und oft nie wieder auftaucht. Dies sollen Sie nun beheben: Befindet sich der Pixel außerhalb der Bildfläche, soll er sich sofort wieder in Richtung dessen bewegen.
Machen Sie dabei Gebrauch der `width`- und `height`-Variablen.

5.5 Farbmuster

Zu guter Letzt wollen wir nun dem Spektakel etwas Farbe verleihen: Entwickeln sie eine Möglichkeit, wie sich der Pixel mithilfe von `stroke(c)` in jedem Durchlauf eine Farbe generiert. Dabei können Sie auf den Zufall oder auf Variablen verlassen, welche nach und nach hoch- oder runterzählen. Seien Sie kreativ und teilen Sie ihre Ergebnisse!

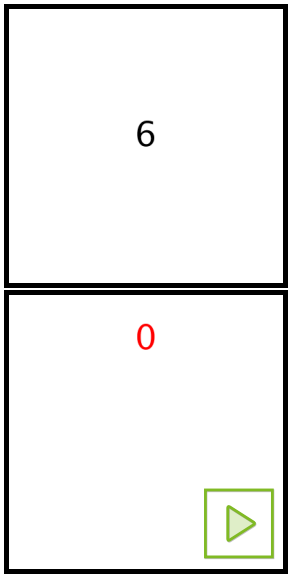


204

14 Z.
25 Z.

int x

Aufgabe 6 – Countdown



6.1 Das Konzept

Schreiben sie einen Algorithmus, der von 10 herunterzählt. Die derzeitige Zahl soll in der Mitte des Bildes angezeigt werden. Wenn der Zähler 0 erreicht, soll er nicht mehr weiter zählen (nicht -1, ...).
Lassen Sie eine Variable `counter` festlegen, bei welcher Zahl Ihr Countdown derzeit ist.
Mithilfe von `frameRate(i)` können Sie festlegen, wie oft die `draw()` -Schleife pro Sekunde aufgerufen werden soll.

6.2 Liftoff!

Lassen Sie bei 0 die Countdown-Zahl in die Luft schießen. Dafür können Sie, wenn Ihre `counter`-Zahl Null ist, die `frameRate(i)` wieder erhöhen.
Um den Raketenstart realistischer darzustellen, arbeiten Sie nicht nur mit Geschwindigkeit, sondern auch mit Beschleunigung. Können Sie eine gleichmäßig steigende Geschwindigkeit der Raketen-0 erzeugen?

224

11-25 Z.

Aufgabe 7 – Die Musterklasse

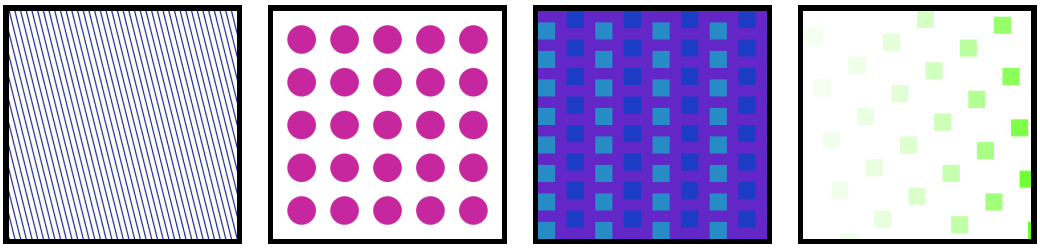
Schleifen sind, wie der Name schon sagt, wiederholend. Sie eignen sich deshalb perfekt, um Muster zu zeichnen. Lassen Sie Ihrer Kreativität freien Lauf!

7.1 Vorbereitung

Beginnen Sie mit ein paar simpleren Darstellungen, mit nur einem oder zwei Zeichenbefehlen und einer bis zwei Schleifen.

Hinweis: *Optional können Sie auch* `mouseX` *und* `mouseY` *verwenden, damit sich das Muster je nach Mausposition verändert.*
Hinweis: *Kennen Sie schon den* `%` *(„Modulo“)-Operator? Dieser könnte Ihnen hier auch helfen!*

Ein paar Möglichkeiten sind hier gezeigt, aber es gibt noch unendlich viele weitere!



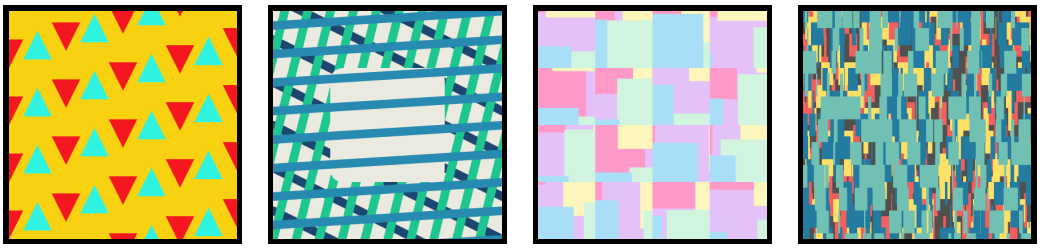
7.2 Muster-Designer:in

Wie Sie sehen, können Sie mit Farben, simplen Formen und allen möglichen Tricks viele schöne Kunstwerke erzeugen.

Können Sie ihre Muster noch komplexer machen? Wenden Sie ihr gesamtes Wissen an, um die Resultate so interessant wie möglich zu gestalten. Teilen Sie ihre Kunstwerke gern mit uns!

Hinweis: *Hier könnte Ihnen der* `translate(x, y)`*-Befehl Arbeit ersparen. Falls sie dies noch nicht getan haben, lesen Sie bitte mehr darüber in der Aufgabe zu Rotationen vom ersten Aufgabenzettel nach, und beachten Sie auch die Verwendung von* `pushMatrix()` *und* `popMatrix()`*.*

Hier einige Inspirationen:



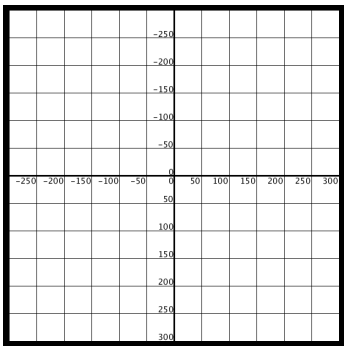
208

23 Z.
34 Z.

Aufgabe 8 – Funktionen zeichnen

Entwickeln Sie einen Algorithmus, der Funktionen visualisieren kann.

8.1 Koordinatensystem



Zeichnen Sie zunächst ein Koordinatensystem. Dabei soll der Punkt `0, 0` im Koordinatensystem nicht oben links, sondern in der Bildmitte liegen. Arbeiten Sie mit einem `translate(x, y)`-Befehl zu Beginn der `draw()`-Funktion.
Hinweis: *Falls Sie* `translate(x, y)` *noch nicht kennen sollten, lesen Sie darüber in der Processing Reference nach.*

Setzen Sie in dem Koordinatensystem mit `text(str, x, y)` sinnvolle Markierungen.

8.2 Funktionen

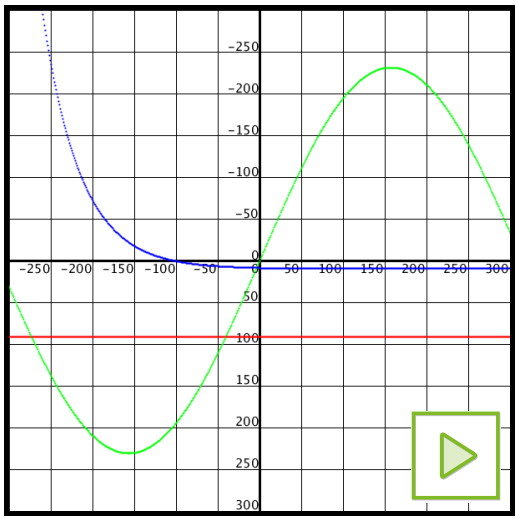
Zeichnen Sie auf ihrem Koordinatensystem nun die folgenden Funktionen in unterschiedlichen Farben, wobei die Parameter `a` und `b` mit `mouseX` und `mouseY` angepasst werden sollen. Zur Darstellung der Funktionen können Sie entweder `point(x, y)` oder `circle(x, y, d)` verwenden.

Hinweis: `mouseX` *und* `mouseY` *werden ihren* `translate(x, y)`*-Befehl nicht berücksichtigen, weshalb sie in jedem Durchlauf der* `draw()`*-Funktion zuerst die* `a`*- und* `b`*-Werte berechnen sollten.*

$$f(x) = a * 2 + b \quad (1)$$

$$g(x) = b * \sin(\frac{x}{100}) \quad (2)$$

$$h(x) = 10 - a^{x/b} \quad (3)$$



Alternativ können Sie auch Ihre eigenen Funktionen wählen und beobachten, wie sich die andere `a`- und `b`-Werte auf Ihre Funktionen auswirken.

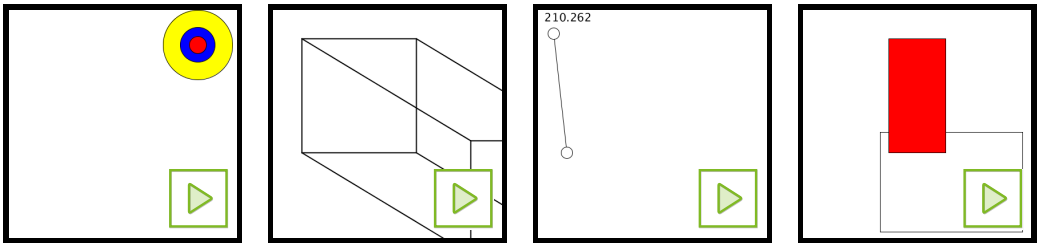
225
+0-5 Z.

Aufgabe 9 – Programme zum Leben erwecken

Gestern haben Sie eventuell einige Aufgaben bearbeitet, die Sie heute interaktiver gestalten können.

Suchen Sie sich unter denen unten dargestellten Aufgaben die aus, die Sie bearbeitet haben, und versuchen Sie, mithilfe von `draw()` und den `mouseX`- und `mouseY`-Variablen die dort gezeigten Elemente zu bewegen.

Selbstverständlich können Sie auch Ihren anderen Programmen mit den neuen Funktionen Interaktivität verleihen.



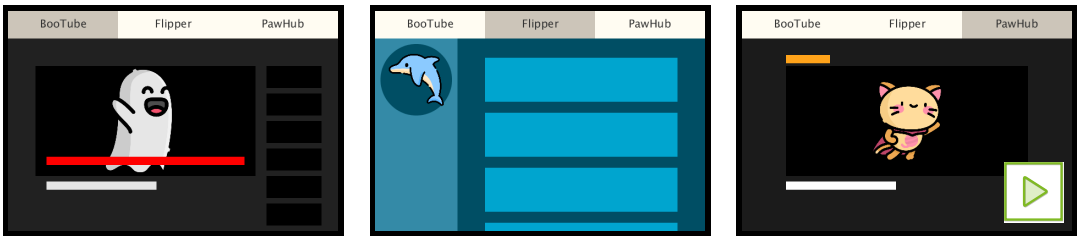
226
67 Z.

Aufgabe 10 – Browsertabs

Simulieren Sie mithilfe von `mouseX` und `mouseY` eine Browser-Applikation.

Je nachdem welchen Tab die Maus auf der oberen Leiste berührt, öffnet sich das entsprechende Fenster. Seien Sie kreativ mit den Website-Inhalten!

Bilddateien können Sie mithilfe der Klasse `PImage` einlesen. Dazu mehr in der *Processing Reference*.



Icons von Freepik via flaticon.

222

18 Z.
22 Z.

Aufgabe 11 – Fibonacci-Blumen

In der Natur taucht die Mathematik verblüffend oft auf - so auch in Blumen. Viele Blumen, wie bekannterweise die Sonnenblume, aber auch die meisten (!) anderen besitzen eine Anzahl von Blüten, welche einer Zahl in der Fibonacci-Sequenz entspricht:

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89, \ldots$$

Grund dafür ist die Evolution - diese Anzahlen von Blüten und auch Samen sind optimal, um an Platz und Ressourcen zu sparen.

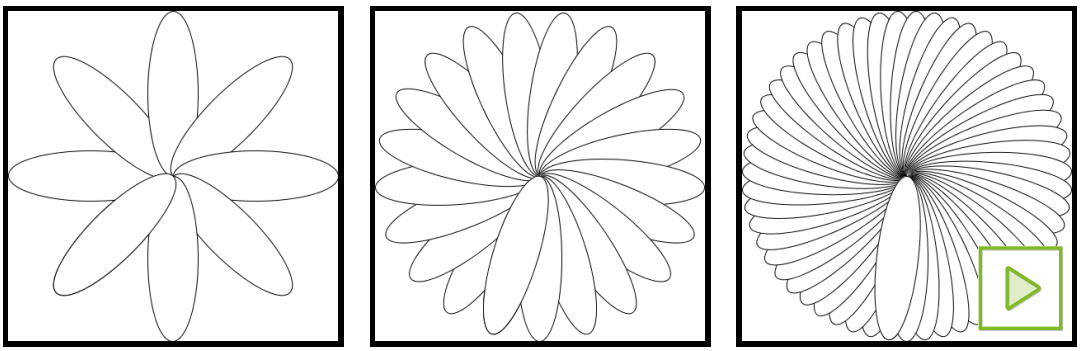
11.1 Blüten

Schreiben Sie zunächst einen Algorithmus, der mithilfe der Fibonacci-Nummer von `mouseX / 30` Blüten zeichnet. Wenden Sie dafür die folgende Funktion an (fügen Sie auf die selbe Ebene ein wie `void draw()` etc.):

```
1 int fibonacci(int n) {
2    if (n == 0 || n == 1)
3       return n;
4    return fibonacci(n -1) + fibonacci(n -2);
5 }
```

Dann können Sie diese in Ihrer `draw()`-Funktion z.B. so aufrufen: `int fib = fibonacci(mouseX / 30);`

Verwenden Sie diesen Zahlenwert, um die Anzahl Blüten festzulegen, und zeichnen Sie die entsprechende Blume.



Hinweis: *Hier werden die Befehle* `translate(x, y)` *und* `rotate(r)` *benötigt. Dazu können Sie bei Bedarf die Aufgabe des ersten Aufgabenzettels zu Rotationen bearbeiten.*
Hinweis: *Schwierigkeiten mit dem Rotationsschritt? Hier ein Tipp:* `TWO_PI / x` *würde ein x-tel der vollständigen Rotation als* `float`*-Wert ausgeben.*

Auf der nächsten Seite geht diese Aufgabe weiter...

11.2 Samen

Was wäre eine Blume ohne Samen? Vermutlich Ausgestorben.

Deshalb wollen wir unserer Blume noch Samen verleihen. Doch nicht einfach irgendwie, sondern mit einer weiteren besonderen Anordnung.

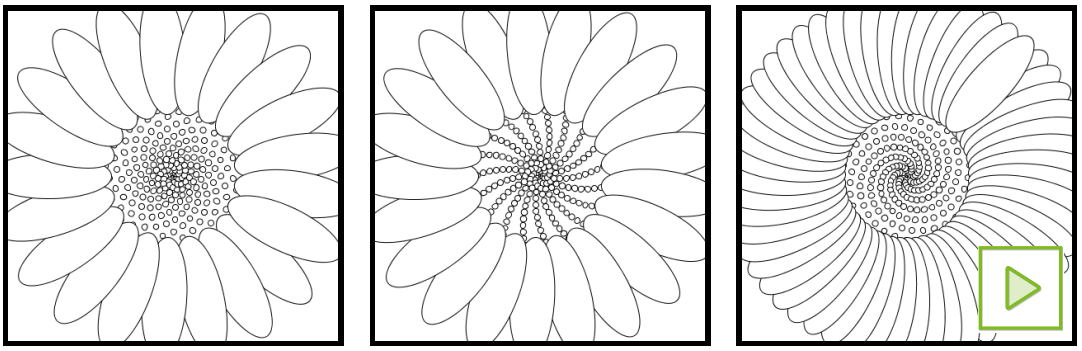
In der Natur greift hier übrigens auch oft die Fibonacci-Zahl. Wir wollen es uns aber etwas einfacher machen, und stattdessen die Mausposition über die Position der Samen entscheiden lassen.

Ändern Sie dazu die Zeichenfunktion so, dass ihre Blüten im Zentrum einen Leerraum lassen.

Schreiben Sie dann eine zweite Schleife, welche Ihnen abhängig von `mouseY` ein Muster bildet:

```
Beginne im Zentrum der Blume
Wiederhole für i = 1 bis n:
   Drehe dich um einen Radiant r
   Bewege dich ein Stück vorwärts
   Zeichne eine Samen
```

Probieren Sie herum, bis sie eine gute Berechnung von `r` und eine sinnvolle Anzahl für `n` gefunden haben.



Selbstverständlich können Sie die Blume auch einfärben (wie könnte man Farben Algorithmisch bestimmen..?) oder die Blüten mithilfe von Bildern zeichnen. Teilen Sie gerne ihre Kreationen!

207

26 Z.
44 Z.

Aufgabe 12 – Reaktionstests

In einigen Berufen werden Bewerber:innen auf ihre körperlichen und mentalen Fähigkeiten geprüft. Eine Möglichkeit, um die mentale Stärke einer Person zu messen, sind Reaktionstests. In dieser Aufgabe wollen wir zwei verschiedene dieser Art programmieren.

12.1 Positionswechsel

In diesem ersten Test sollen Bewerber:innen mit der Maus Kreise anwählen. Sobald sich die Maus in dem derzeit gezeichneten Kreis befindet, soll an einer anderen Position ein neuer Kreis erscheinen.

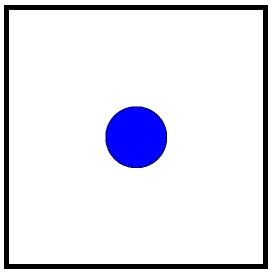
Können Sie mithilfe von `millis()` und einer eleganten Implementation die durchschnittliche Reaktionszeit in Sekunden ermitteln (also die Zeit zwischen Erscheinen des Kreises und dessen Anwahl)?
Hinweis: *Es kann hilfreich sein, zu zählen, wie oft ein Kreis getroffen wurde.*

12.2 Optionale Erweiterung: Farbwechsel

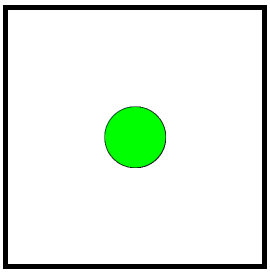
In diesem Test sollen Bewerber:innen mit der Maus auf einen Farbwechsel reagieren. Lassen Sie den Zufall über einen Zeitraum (ein paar Sekunden) entscheiden, nach welchem das Rechteck die Farbe wechselt. Ab diesem Zeitpunkt sollen Sie wieder die Zeit messen, die die Spieler:in benötigt, um den Kreis zu treffen.

Zeichnen Sie einen Kreis in die Bildmitte, und färben Sie ihn entsprechend ein.

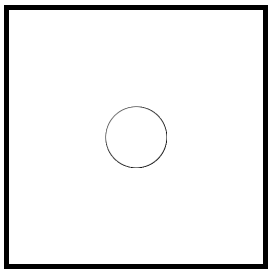
Sie können entweder das Ergebnis nach einer bestimmten Anzahl Runden anzeigen (und das Spiel beenden), oder wie im Aufgabenteil 1 den Durchschnitt immer anzeigen.



Der Kreis ist aktiv, wurde noch nicht angewählt.



Der Kreis war schon aktiv und wurde angewählt.



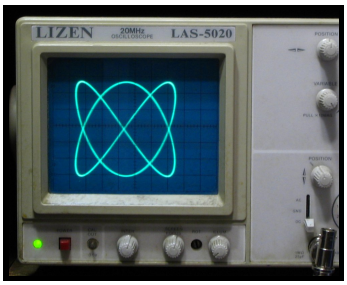
Der Kreis ist nicht aktiv und ignoriert die Maus.

Nach ein paar Runden wird das Ergebnis angezeigt.

223

10 Z.
10 Z.
10 Z.
14 Z.

Aufgabe 13 – Lissajous-Figuren



Quelle: Wikimedia, Omegatron (CC3)

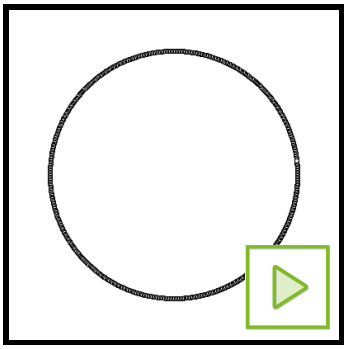
Lissajous-Figuren, benannt nach Physiker Jules Antoine Lissajous, sind Kurvengraphen, welche von zwei harmonischen Schwingungen ($x(t)$ und $y(t)$) erzeugt werden.

Processing eignet sich sehr gut, um diese Figuren zu zeichnen. Beginnen wir mit einer Eingewöhnung.

13.1 Kreis zeichnen

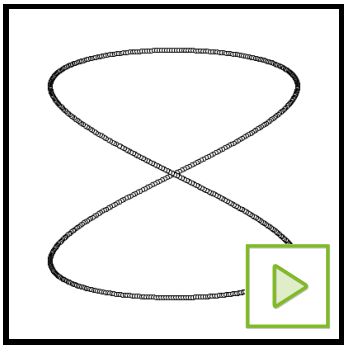
Zunächst sollen Sie einen Kreis mithilfe von `point(x, y)` oder (vielen, kleinen - nicht schummeln!) `circle(x, y, d)`-Befehlen zeichnen.

Funktionen wie `sin(f)` und Konstanten wie `PI` könnten Ihnen hierbei helfen.



13.2 Variationen

Vielleicht ist Ihnen bereits aufgefallen, dass Sie nun zwei `a *sin( b)` Funktionen haben.
Verändern oder beeinflussen Sie `b` und `a` von den beiden Funktionen, um die rechts unten gezeigten Figuren nachzustellen.

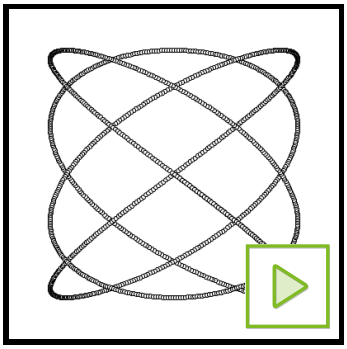


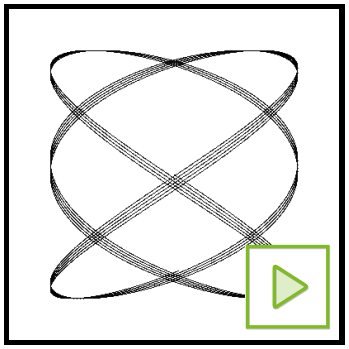
13.3 Figuren generieren lassen

Nun wird es interessant: Bauen Sie ihren Algorithmus so um, dass die gesamte Figur in einem Durchlauf der `draw()`-Funktion gezeichnet wird. Definieren Sie die beiden Funktionen, falls nicht schon erfolgt, wie folgt:

$$\begin{aligned} x(t) &= 150 * \sin(a * t) \\ y(t) &= 150 * \sin(b * t) \end{aligned}$$

Machen Sie nun die Parameter a und b von `mouseX` und `mouseY` abhängig ($a = \frac{\texttt{mouseX}}{500}$, $b = \frac{\texttt{mouseY}}{500}$) und zeichnen Sie die Lissajous-Figur von $t = 0$ bis $t = 100$ mit einer Schrittlänge von 0.1.





214

22 Z.
25 Z.

int x

Aufgabe 14 – Binäruhr

Die Binärcodierung ist die Umwandlung von Werten in Eine Sequenz aus Nullen und Einsen (bzw. „wahr“ und „falsch“). Möchte man eine Zahl binär Codieren, gilt folgendes System:

Kodierung	128	64	32	16	8	4	2	1
Binärwert	0	0	1	0	1	0	1	0

Die Abfolge `0010 1010` steht also für $32 + 8 + 2 = 42$. Für mehr Informationen, siehe Wikipedia.

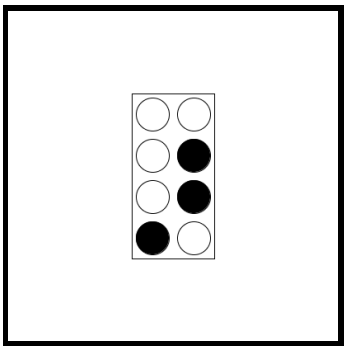
14.1 Binäres Modul

Zunächst sollen Sie eine Funktion schreiben, welches Ihnen an einer angegebenen Stelle ein Byte darstellt. Wenden Sie diese Vorlage an, um zu beginnen:

```
1  void setup() {
2     size(400, 400);
3  }

4  void draw() {
5     background(255);
6     drawModule(second(), 150, 100); //
          Funktionsaufruf
7  }

8  void drawModule(int value, int x, int y) {
9     rect(x, y, 100, 200);
10    // Konvertierung value -> Binär
11    // Repräsentation Binärwerte
12 }
```

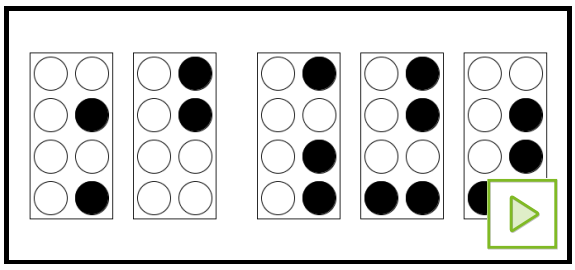


`0001 0110` = 22 in Binär.

Hinweis: *Der Befehl* `binary(i, digits)` *gibt Ihnen einen String des Wertes von* `i` *in Binär aus, mit der Länge* `digits`.
Hinweis: *Mit* `myString.charAt(i)` *erhalten Sie den Charakter des Strings* `myString` *am Index* `i`.

14.2 Die Zeit läuft

Wenden Sie diese Funktion nun an, um die gesamte Uhrzeit und Datum darzustellen. Die entsprechenden Funktionen finden Sie in der *Processing Reference*.



Es ist der 12. Mai um 11:29:22.

227

63 Z.

Aufgabe 15 – Mission-Control-Simulator

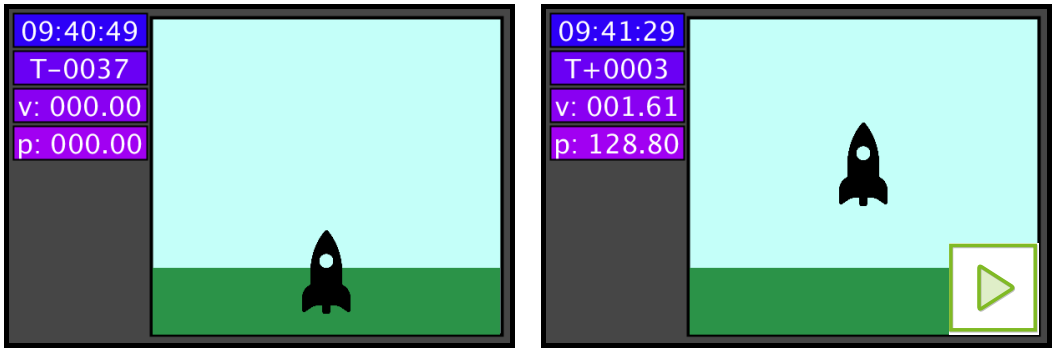
In dieser Aufgabe sollen Sie ein Programm entwerfen, welches eine (sehr grobe) Übersicht von einem Raketenstart zeigt. Entwerfen Sie ein Dashboard mit den folgenden Elementen:

1. Einer digitalen (oder analogen) Uhr mit der tatsächlichen Zeit (optional Datum)
2. Ein Panel, welches die Zeit bis zum- bzw. seit dem Raketenstart anzeigt
3. Ein Panel, welches die derzeitige Position und Geschwindigkeit der Rakete anzeigt
4. Eine visuelle Repräsentation der Rakete und dessen Position - seien Sie kreativ!

Lassen Sie den Raketenstart eine Minute nach Programmstart geschehen. Zählen Sie vom Programmstart ausgehend die Sekunden, welche das Programm am laufen ist, um bei Startzeit die Rakete starten zu lassen.

Diese Aufgabe können Sie ihren Fähigkeiten entsprechend erweitern. So könnten Sie Funktionen verwenden, um die Lesbarkeit zu verbessern. Auch könnten Sie der Rakete Feuer verleihen, oder diese tatsächlich ins Weltall verfolgen und dort einen Orbit einschlagen lassen.

Hinweis: *Für eine gleichmäßige Anzahl von Ziffern können Sie* `nf(num, left, right)` *und dessen Variationen verwenden (mehr dazu in der Processing Reference).*



Icons von Freepik via flaticon.