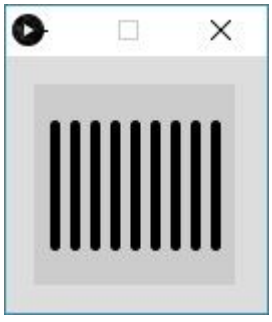
6. Schleifen

Mit den Fertigkeiten und dem Wissen aus den ersten fünf Kapiteln können bereits sehr interessante Programme geschrieben werden. Allerdings kann das Programmieren umständlich sein, wenn dieselben Befehle oftmals hintereinander ausgeführt werden sollen. Aktuell müssten nämlich Befehle so oft hingeschrieben werden, wie man sie eben zur Ausführung braucht. Das kann 5 mal sein, aber auch 100 mal. Und manchmal weiß man dies vor der Ausführung des Programms nicht einmal, da die Anzahl der Wiederholungen abhängig von anderen Parametern ist, welche vielleicht erst zur Laufzeit berechnet werden. Um sich Arbeit zu sparen, gleichzeitig aber auch flexibel zu sein, wird das Konzept der **Schleife** zur Umsetzung von Wiederholungen verwendet.

Die Wiederholung von Befehlen ist bereits aus Kapitel 3 bekannt: Durch die Wiederholung des `draw()` Bereichs ist es möglich, Animationen zu erstellen. Dazu wurde ein Bild gezeichnet und im nächsten Durchlauf von `draw()` mit einer geringfügigen Veränderung erneut gezeichnet. Auch mit Schleifen können Befehle einmal hingeschrieben werden und mehrmals wiederholt werden, ohne dabei mehr Schreibaufwand zu haben. Mit Hilfe von Variablen können innerhalb der Schleife außerdem kleine Änderungen an den Befehlen vorgenommen werden, um zum Beispiel interessante Muster zu erzeugen.

Starten wir mit einem einfachen Beispiel: Es sollen 10 Linien nebeneinander gezeichnet werden.

Eine Möglichkeit wäre es nun, für jede Linie den entsprechenden Befehl zu schreiben.



```
void setup() {
  strokeWeight( 5 );
  int num = 1;

  line( num * 10, 20, num * 10, 80 );
  num++;
  line( num * 10, 20, num * 10, 80 );
  num++;

  // weitere 8 mal
  // line( num * 10, 20, num * 10, 80 );
  // num++;

}
```

Das ist jedoch eine sehr mühsame Arbeit. Außerdem ist diese Lösung nicht sehr flexibel. Was ist, wenn man den Abstand zwischen den Linien ändern will? Oder die Anzahl der Linien? Oder die Länge der Linien? Jeder einzelne Befehl müsste schon bei kleinster Änderung angepasst oder erweitert werden. Mit einer Schleife lässt sich dieses Beispiel besser lösen.

6.1. While-Schleife

Sachverhalte, die sich umgangssprachlich mit *“Solange (eine Bedingung zutrifft) wiederhole … “* formulieren lassen, können elegant mit einer sogenannten `while`-Schleife abgebildet werden. Beim obigen Beispiel könnte dies vereinfacht formuliert werden mit: “Solange weniger als 10 Linien gezeichnet sind, zeichne eine weitere (vertikale) Linie (im selben Abstand, mit derselben Länge..)...”

Konkret wird das obige Beispiel dann in Processing mit einer `while`-Schleife folgendermaßen umgesetzt:

```
void setup() {
  strokeWeight( 5 );
  int num = 1;
  while( num < 10 ) {
    line( num * 10, 20, num * 10, 80 );
    num++;
  }
}
```

Sehen wir uns das Ganze nun nochmals Schritt für Schritt an:

Die `while`-Schleife wird mit dem Schlüsselwort `while` eingeführt. Danach folgt die Bedingung wie bei einer `if`-Verzweigung, gefolgt vom Code in geschwungenen Klammern, der wiederholt werden soll.

Der Aufbau der `while`-Schleife sieht allgemein wie folgt aus:

```
while (Bedingung) {
  // Code, der wiederholt werden soll
  // Anweisung 1;
  // Anweisung 2;
  // ...

}
```

Damit ist er dem Aufbau einer `if`-Anweisung sehr ähnlich. Genau wie bei der `if`-Verzweigung kann der Code einmal ausgeführt werden oder auch gar nicht. Aber mit einer Schleife kann der Code auch mehr als nur einmal ausgeführt werden, was mit einer `if`-Verzweigung nicht möglich war.

Bei der `while`-Schleife wird der Code solange ausgeführt, solange die Bedingung innerhalb der runden Klammern nach dem Schlüsselwort `while` wahr ist. Ist also die Bedingung wahr, dann werden die Anweisungen innerhalb der nachfolgenden geschwungenen Klammern ausgeführt. Bei der schließenden geschwungenen Klammer angekommen, wird dann

überprüft, ob die Bedingung noch immer wahr ist und der Code wird aufs Neue ausgeführt, wenn das der Fall ist. Dieser Vorgang wird solange wiederholt, bis die Bedingung nicht mehr wahr ist. Dann wird die Schleife beendet - die Anweisungen innerhalb der Schleife werden nicht mehr ausgeführt, und der Code nach der Schleife wird ausgeführt. Da das Beenden der Schleife von der Bedingung abhängt, die Schleife also abhängig von der Bedingung ausgeführt oder abgebrochen wird, wird die Bedingung auch als **Abbruchbedingung** bezeichnet.

Im Beispiel mit den senkrechten Linien wird eine sogenannte **Zählvariable** (wir nennen sie im Beispiel `num`) angelegt. Wir wählen den Namen so, weil die Variable `num` in der Schleife die Anzahl der Schleifendurchläufe mitzählt. Das bedeutet, diese Variable wird mit jedem Schleifendurchlauf um eins inkrementiert, d.h.hinauf gezählt (`num++;`).
Die Bedingung (`num < 10`) hängt nun von dieser Zählvariablen ab. Damit die Schleife irgendwann abgebrochen wird, muss gewährleistet sein, dass der Wert der Zählvariable so verändert wird, dass die Abbruchbedingung irgendwann nicht mehr zutrifft. In diesem Beispiel wird das mit dem Inkrementieren der Variable `num` erreicht (`num++;`). `num` wird bei jedem Schleifendurchlauf um 1 erhöht - so lange, bis `num` den Wert 10 enthält. Danach stimmt die Aussage `num < 10` nicht mehr und die Schleife wird abgebrochen.

Will man die Anzahl der Linien ändern, muss einfach die Abbruchbedingung entsprechend angepasst werden, z.B auf `num < 15` für 15 Linien.

Beispiel: Schleifen können auch innerhalb des `draw()` Bereichs verwendet werden. Die Schleife selbst zeichnet wiederum alle Linien. Mit einer kleinen Anpassung der Koordinaten können die Linien auch animiert werden. Testen Sie den Code aus!

```
float move = 0;

void setup() {
  strokeWeight( 5 );
}

void draw() {
  background(255);
  int num = 0;

  while( num < 5 ) {
    line( num * 20 + move, 20, num * 20 + move, 80 );
    num++;
  }

  move += 0.5;
  move = move % 20;
}
```

6.2. For-Schleife

Um jegliche Art von Wiederholungen in der Programmierung auszudrücken, reicht es grundsätzlich, nur die `while`-Schleife zu kennen und anwenden zu können. Es gibt jedoch noch einen weiteren Schleifentyp, welcher viel intuitiver anzuwenden ist, wenn die Anzahl der Wiederholungen bereits (direkt oder indirekt) bekannt ist - die `for`-Schleife.

Beispiel: Das Linien-Beispiel kann folgendermaßen mit einer `for`-Schleife umgesetzt werden:

```
void setup() {
  strokeWeight(5);
  for(int num = 1; num < 10; num++) {
    line(num * 10, 20, num * 10, 80);
  }
}
```

Die `for`-Schleife wird mit dem Schlüsselwort `for` eingeleitet. Die Initialisierung der Zählvariable (`int num = 1;`) wird im Vergleich zur Umsetzung mit einer `while`-Schleife jetzt in den ersten Teil innerhalb der runden Klammern der `for`-Schleife verschoben. Die Bedingung `num < 10;` steht im zweiten Teil (in der "Mitte"). Das Inkrementieren der Zählvariable (d.h. `num++`) wird nun nicht mehr (wie bei der `while`-Schleife) innerhalb des Anweisungsblockes geschrieben, sondern in den dritten Teil ("rechts") innerhalb der runden Klammern "vorgezogen". Die drei Teile werden voneinander mittels Strichpunkten getrennt.

Allgemein sieht der Aufbau einer `for`-Schleife damit wie folgt aus:

```
for(Initialisierung, Bedingung, Update){
  // Code, der wiederholt werden soll
  // Anweisung 1;
  // Anweisung 2;
  // ...
}
```

An erster Stelle steht die *Initialisierung*. Hier werden Variablen angelegt und initialisiert, welche für die Ausführung der Schleife relevant sind. In den meisten Fällen wird hier eine Zählvariable deklariert und initialisiert.
Dann folgt die *Bedingung*, wie sie von der `while`-Schleife bereits bekannt ist.
Als letztes wird im *Update*-Teil eine Änderung definiert, welche nach jedem Schleifendurchgang passieren soll, damit am Ende der Schleife die Abbruchbedingung nicht mehr erfüllt ist. Das ist in der Regel eine Beschreibung, wie die Zählervariable in der Initialisierung verändert wird, sodass dann die Bedingung irgendwann nicht mehr zutrifft.
In den geschwungenen Klammern folgt dann wieder der Code, der wiederholt werden soll.

Wenn man die `for`-Schleife und die `while`-Schleife einander gegenüberstellt, dann sieht man, dass die `for`-Schleife alle für die Schleife selbst relevanten Codeteile in einer Zeile zusammenfasst und in diesem Fall einen besseren Überblick verschafft.

6.3. For-Schleife vs. While-Schleife

Auch wenn es mehrere Arten von Schleifen gibt, kann für jedes Wiederholungsproblem sowohl eine `for`- als auch eine `while`-Schleife verwendet werden. Das Linien-Beispiel konnte sowohl mit der einen als auch mit der anderen Schleife gelöst werden

Es stellt sich dann die Frage, warum es mehrere Arten von Schleifen gibt?

Die `for`-Schleife wird gerne verwendet, wenn man weiß, wie oft die Schleife wiederholt wird, wenn Zählprobleme abgebildet werden sollen und wenn die Veränderung des Schleifenzählerwerts konstant ist. Auf einen Blick sieht man, welche Variablen angelegt werden, welche Bedingungen gelten müssen und wie die Variable verändert wird. Der Aufbau der Schleife hilft auch, das Hochzählen der Variable nicht zu vergessen und verhindert somit **Endlosschleifen** (siehe Unterkapitel Endlosschleife)

In der `while`-Schleife hingegen passiert schnell, dass auf die Zählvariable vergessen wird. Dafür sind `while`-Schleifen besonders geeignet, wenn die Anzahl der Wiederholungen nicht im vornherein bestimmt ist bzw. wenn keine Zählvariablen auftreten.

Ein abstraktes Beispiel ist: Solange der User das Programm nicht beendet, zeichne weiter. In diesem Fall gibt es keine Zählvariable. Die Bedingung ist abhängig von der Benutzerinteraktion.

Mit einer `while`-Schleife ließe sich das vereinfacht wie folgt darstellen:

```
boolean userPressed_ESC = false;

while( !userPressed_ESC ) {
  // draw things
}
```

Bei einer `for`-Schleife müsste man die Initialisierung und das Update auslassen:

```
boolean userPressed_ESC = false;

for( ; !userPressed_ESC; ) {
  // draw things
}
```

Beide Schleifen sind möglich, aber die `while`-Schleife ist für dieses Problem intuitiver.

6.4. Endlosschleifen

Schleifen sind sehr praktisch und ersparen viel Schreibarbeit. Allerdings kann es zu sogenannten Endlosschleifen kommen. Das heißt, dass die Abbruchbedingung nie falsch wird und die Schleife somit ewig weiterläuft. In den meisten Fällen ist das unerwünscht und ungeplant. Typische Zeichen für eine unerwünschte Endlosschleife ist, wenn das Programm nicht auf Eingaben reagiert oder bestimmte Aktionen, die nach der Schleife ausgeführt werden sollten, nicht ausgeführt werden. In Processing könnte ein Zeichen für eine Endlosschleife sein, dass nur eine oder keine Form gezeichnet wird, statt mehreren oder die Animation nicht funktioniert.

Damit die Bedingung nach einigen Wiederholungen nicht mehr zutrifft, muss sichergestellt sein, dass es innerhalb der Schleife die Möglichkeit gibt, die für die Bedingung relevanten Variablen zu ändern. Im Linien-Beispiel wurde die Variable `num` in der Abbruchbedingung verwendet. In der Schleife wurde mit dem Inkrementieren sichergestellt, dass die Abbruchbedingung irgendwann nicht mehr zutrifft.

Die zwei häufigsten Quellen für Endlosschleifen sind

- falsche Abbruchbedingungen und,
- dass darauf vergessen wurde, die entsprechenden Variablen in der Schleife zu ändern.

6.5. Verschachtelte Schleifen

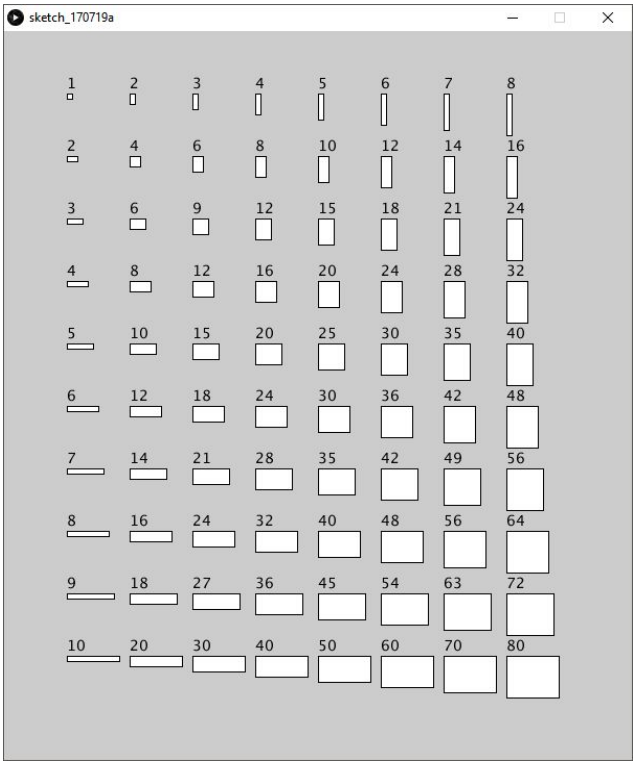
Einzelne Schleifen sind bereits sehr hilfreich und können sehr interessante Effekte erzeugen. Aber auch Schleifen können verschachtelt, d.h. Schleifen innerhalb von Schleifen, werden, um noch mehr interessante Muster zu zeichnen. Folgendes Beispiel ist eine Abwandlung des Linienbeispieles aus dem Video.

Kopieren Sie den Code in Processing und führen Sie ihn aus.

```
void setup() {
  size(600, 700);

  int xOffset = 60;
  int yOffset = 60;
  int scale = 5;
  for (int row = 1; row <= 10; row++) {
    for (int column = 1; column <= 8; column++) {
      fill(0);
      textSize(14);
      text(row*column, column*xOffset, row*yOffset-5);
      fill(255);
      rect(column*xOffset, row*yOffset, scale*row, scale*column);
    }
  }
}
```

Das Beispiel zeichnet im Sketch-Fenster eine Multiplikationstabelle, bei der die Ergebnisse nicht nur als Zahl, sondern auch als Fläche eines Rechtecks dargestellt sind.



Verändern Sie ein paar Werte (z.B. die Grenzen in den Bedingungen oder den Offset) und untersuchen Sie, wie sich die Änderungen auf das Ergebnis auswirken.

Um zu sehen, was die innere Schleife macht, ändern Sie die Obergrenze der äußeren Schleife in deren Bedingung von 10 auf 1. Dadurch hat die äußere Schleife nur einen Durchlauf und die innere Schleife wird genau einmal ausgeführt. Dann ändern Sie diese Obergrenze auf 2 usw. Dadurch kann man gut sehen, wie mit Hilfe der Schleifen die Tabelle zeilenweise von oben nach unten gezeichnet wird.

Überlegen Sie, ob die folgenden Aussagen stimmen oder nicht und versuchen Sie ihre Antwort zu begründen.

1. Die innere Schleife zeichnet eine Zeile in der Multiplikationstabelle

2. Die Obergrenze der inneren Schleife bestimmt die Anzahl der Spalten.

3. Das Ändern von `xOffset` bewirkt, dass der horizontale Abstand zwischen den Einträgen kleiner oder größer wird.

4. Die Obergrenze der äußeren Schleife bestimmt die Anzahl der Zeilen in der Tabelle

5. `scale` bestimmt die Größe der Rechtecke