

Reihungen in Processing¹

Bislang haben Sie vermutlich Variablen verwendet, die genau einen Wert speichern können, z. B. einen Wahrheitswert, eine Zahl oder ein Zeichen. Um beispielsweise den RGB-Wert eines Farbtons zu speichern, benötigt man drei Variablen vom Typ Ganzzahl, eine für den Rotwert, eine für den Grünwert und eine für den Blauwert. Für drei Werte ist das noch überschaubar. Möchte man sich noch mehr Werte eines Typs merken, z. B. zehn verschiedene Quizfragen mit den jeweiligen Antworten, wird es schnell unübersichtlich. Daher schauen wir uns im Folgenden eine Datenstruktur an, in der wir mehrere Werte eines Datentyps zusammenfassen können.

Eindimensionale Reihungen

Die einfachste Form einer Datenstruktur, die mehrere Werte eines Datentyps enthält, ist in Processing die **Reihung** (eng. Array). Der entsprechende Datentyp wird im Zusammenhang mit Reihungen auch als Inhaltstyp bezeichnet. In einer Reihung vom Inhaltstyp Ganzzahl (`int`) der Länge drei können beispielsweise drei Ganzzahlen gespeichert werden. In einer Reihung vom Inhaltstyp Zeichenkette (`String`) der Länge zehn können zehn verschiedene Zeichenketten abgelegt werden.

Erzeugen von Reihungen und Zugriff auf einzelne Werte

Anders als bei den Listen in Scratch² oder Snap!³ muss die Länge der Reihung bereits beim Erzeugen festgelegt werden. Dabei müssen nicht direkt beim Start des Programms auch alle Speicherplätze der Reihung gefüllt werden, dies kann noch geschehen, während das Programm läuft. So können z. B. die Antworten des Anwenders auf Quizfragen nach und nach in eine Reihung eingetragen werden. Entscheidend ist, dass nach dem Erzeugen eine **feste Anzahl** an Speicherplätzen in der Reihung zur Verfügung steht und diese Anzahl während der Laufzeit des Programms nicht mehr geändert werden kann. Man spricht daher auch von **statischen Reihungen**.

Betrachten wir ein erstes Beispiel zum Erzeugen und Verwenden einer Reihung.

```
1  int[] rgb = new int[3];
2  void setup() {
3      size(400, 300);
4  }
5  void draw() {
6      rgb[0] = (rgb[0]+1)% 256;
7  }
8  void mouseDragged() {
9      stroke(rgb[0], rgb[1], rgb[2]);
10     line(pmouseX, pmouseY, mouseX, mouseY);
11 }
```

Beispiel 1: Zeichenprogramm mit Farbwechsel

Haben Sie schon eine Idee, wie sich das Programm in Beispiel 1 beim Ausführen verhalten wird?

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

² Scratch ist ein Projekt der Scratch Foundation in Zusammenarbeit mit der Lifelong Kindergarten Group des MIT Media Lab. Es ist kostenlos unter <https://scratch.mit.edu> erhältlich.

³ Snap! wird von der University of California, Berkeley zur Verfügung gestellt: <https://snap.berkeley.edu>

Wir schauen uns den Code einmal etwas genauer an. In Zeile 1 wird eine Reihung mit dem Namen `rgb` erzeugt, um den Rot-, den Grün- und den Blauwert eines Farbtons in einer Variablen zu speichern. Als Inhaltstyp der Reihung wird daher `int` angegeben. Mithilfe der eckigen Klammern wird festgelegt, dass es sich um eine Reihung und nicht um eine einfache Variable vom Typ `int` handelt. Mit dem Ausdruck `new int[3]` wird eine Reihung vom Inhaltstyp Ganzzahl der Länge drei erzeugt. Da noch keine Werte für die einzelnen Felder der Reihung festgelegt wurden, enthalten sie zunächst alle den Wert 0⁴.

In Zeile 9 wird auf die Werte der Reihung zugegriffen. Dazu wird hinter dem Namen der Reihung die Position des gewünschten Wertes in eckigen Klammern angegeben. Dabei beginnt die Nummerierung bei 0, wie wir es bereits von der Nummerierung der Zeichen einer Zeichenkette kennen.

In Zeile 6 wird der Wert der Reihung an der Position 0 verändert. Auf der rechten Seite des Zuweisungsoperators wird der Wert an der Position 0 ausgelesen, 1 addiert und die Summe modulo 256 genommen, so dass wir einen neuen Wert zwischen 0 und 255 erhalten. Auf der linken Seite des Zuweisungsoperators steht, dass der neue Wert in der Reihung an Position 0 eingetragen wird.

Sollen die Felder einer Reihung beim Erzeugen direkt mit individuellen Startwerten belegt werden, können diese in geschweiften Klammern angegeben werden. Die Länge der Reihung ergibt sich dann aus der Anzahl der Werte. Die Anweisung zum Erzeugen der Reihung in Zeile 1 sähe dann z. B. so aus:

```
int[] rgb = {78, 107, 158};
```

Aufgabe 1: Geben Sie für die folgenden Codefragmente an, wie sie in Bezug auf Abbildung 1 ausgewertet werden.

- a) `rgb[1]`
- b) `rgb[3]`
- c) `rgb[2] = rgb[1];`
- d) `rgb[1] = rgb[0] + rgb[1];`
- e) `rgb[2] = rgb[2] - 10;`

| | | | |
|-----|----|-----|-----|
| rgb | 78 | 107 | 158 |
| | 0 | 1 | 2 |

Abbildung 1: Reihung `rgb` mit individuellen Startwerten

Aufgabe 2:

- a) Verändern Sie das Programm in Beispiel 1 so, dass eine Zeichnung mit wechselnden Grüntönen bzw. mit wechselnden Blautönen entsteht.
- b) Verwenden Sie in Beispiel 1 statt der Standardbelegung eine individuelle Anfangsbelegung wie in Abbildung 1.

Aufgabe 3: Verändern Sie den Inhalt der `draw`-Methode in Beispiel 1 so, dass für den Rot-, den Grün- und den Blauwert jeweils eine Zufallszahl in die Reihung `rgb` eingetragen wird.

Systematisches Durchlaufen einer Reihung

In Beispiel 1 und den Aufgaben 1 und 2 haben wir nur einzelne Werte der Reihung verändert. Vielleicht haben Sie auch in Aufgabe 3 für jede Zufallszahl, die eingetragen werden muss, eine neue Anweisung geschrieben. Bei drei Werten ist das kein Problem. Möchte man jedoch in einer längeren Reihung systematisch alle Werte der Reihung betrachten oder verändern, wäre es bei langen Reihungen sehr umständlich für jedes Feld der Reihung eine eigene Anweisung anzugeben. Stattdessen

⁴ Bei Reihungen, die Werte eines primitiven Datentyps wie z. B. `int`, `float` oder `boolean` enthalten, wird jedes Feld zu Beginn automatisch mit dem Standardwert belegt, also 0, 0.0 bzw. `false`. Für komplexere Datentypen (z. B. `String`) muss hingegen jedes Feld vor dem ersten Zugriff explizit mit einem Wert belegt werden, da es ansonsten zu Fehlermeldungen kommt.

bietet es sich daher an, in einer Zählschleife alle Positionen der Reihung zu durchlaufen. In Beispiel 2 wird so jeder Wert der Reihung um 1 erhöht.

```
1 for(int i = 0; i < 3; i++){
2     rgb[i] = (rgb[i] + 1) % 256;
3 }
```

Beispiel 2: Durchlaufen einer Reihung mit einer Zählschleife

Aufgabe 4:

- Wählen Sie in Beispiel 1 die Werte aus Abbildung 1 als Startbelegung der Reihung `rgb`. Ersetzen Sie anschließend Zeile 6 durch die Zeilen 1 bis 3 in Beispiel 2. Testen Sie das Programm. Geben Sie für die ersten vier Aufrufe der `draw`-Methode die Belegung der Reihung `rgb` an.
- Lösen Sie Aufgabe 3 unter Verwendung einer Zählschleife.

Aufgabe 5: Die Datei *Aufgabe5* enthält eine Methode `blumeZeichnen(int x, int y)`, die eine Blume an der übergebenen Position zeichnet.

- Es sollen zehn Blumen an zufälligen Positionen gezeichnet werden. Legen Sie dazu zwei Reihungen vom Typ `int` an. Eine für die x-Koordinaten und eine für die y-Koordinaten der Blumen.
- Über das Blumenbeet weht nun der Wind und lässt die Blumen hin und her tanzen. Verändern Sie dazu in der `draw`-Methode bei jedem Zeichenzyklus die Koordinaten geeignet.

Hinweis: Setzen Sie die Bildwiederholrate in der `setup`-Methode z. B. auf 4, damit die Bewegung nicht zu hektisch wird: `frameRate(4);`

- Verändern Sie das Programm so, dass es die Blumen in unterschiedlichen Farben zeichnet.

Verwendung des Attributs `length` beim systematischen Durchlaufen einer Reihung

Beim Programmieren stellt man gelegentlich fest, dass die Länge einer Reihung anders gewählt werden sollte. So könnte man in Aufgabe 5 das Programm so verändern, dass 20 oder 30 statt 10 Blumen gezeichnet werden. Auch wenn man die Länge einer Reihung zur Laufzeit nicht mehr anpassen kann, ist das im Quelltext vor der Ausführung des Programms natürlich jederzeit möglich. Ändert man aber nur die Länge der Reihung und vergisst beispielsweise die Schleifenbedingung für das Durchlaufen der Reihung anzupassen, arbeitet das Programm nicht mehr wie gewünscht oder es kommt zu Fehlermeldungen. Um dies zu verhindern, ist **das Attribut `length`** einer Reihung hilfreich, das die Länge einer Reihung enthält. Der Aufruf `rgb.length` in Beispiel 1 würde z. B. den Wert 3 liefern, da die Reihung `rgb` die Länge 3 hat.

Aufgabe 6:

- Ändern Sie in Ihrem Programm aus Aufgabe 5 die Länge der Reihungen zunächst auf 20 und anschließend auf 5, ohne die Schleifenbedingung beim Durchlaufen der Reihung zu verändern. Erläutern Sie Ihre Beobachtungen.
- Verwenden Sie in der Schleifenbedingung für das Durchlaufen der Reihung das Attribut `length`, so dass Sie die Länge der Reihung beim Programmieren beliebig ändern können.

Aufgabe 7: Die Datei *Aufgabe7* enthält eine Methode `windmuehleZeichnen(int x, int y, int winkel)`, die eine Windmühle an der übergebenen Position mit dem übergebenen Drehwinkel in Grad zeichnet.

Verwenden Sie die Methode, um mehrere Windmühlen zu zeichnen, die sich im Wind drehen. Gestalten Sie Ihr Programm so, dass die Anzahl der Windmühlen mit möglichst wenig Aufwand verändert werden kann.

Verwendung einer *for-each*-Schleife zum systematischen Durchlaufen einer Reihung

Da es sich bei einer Reihung um ein sogenanntes iterierbares Objekt handelt, kann für das systematische Durchlaufen aller Elemente auch eine spezielle *for*-Schleife verwendet werden, die allgemein als *for-each*-Schleife bezeichnet wird.

Beispiel 3 zeigt, wie mithilfe einer *for-each*-Schleife das Maximum aller Werte einer global definierten Reihung `messwerte` bestimmt werden kann. Bei den Werten der Reihung könnte es sich z. B. um die regelmäßig gemessenen Temperaturwerte eines Tages handeln. Abgebildet ist nur die Operation `bestimmeMax`. Das vollständige Beispiel liegt als Processing-Programm bei.

```
1 float bestimmeMax() {  
2     float max = 0;  
3     for(float wert: messwerte) {  
4         if(wert > max) max = wert;  
5     }  
6     return max;  
7 }
```

*Beispiel 3: Durchlaufen einer Reihung mithilfe einer *for-each*-Schleife*

Bei der Definition einer *for-each*-Schleife wird eine Variable definiert, die temporär nacheinander die Elemente der Reihung aufnimmt. In Beispiel 3 ist das die Variable `wert`. Der Datentyp der Variablen muss dem Inhaltstyp der Reihung entsprechen. Hinter dem Doppelpunkt wird die Reihung angegeben, die durchlaufen werden soll. In Beispiel 3 ist das die global definierte Reihung `messwerte`.

Hinweis: Eine *for-each*-Schleife ermöglicht den Zugriff auf jedes Element einer Reihung, ohne dass explizit eine Zählvariable für die Position benötigt wird. Da die Elemente nur als temporäre Variable bereitgestellt werden, eignet sich diese Schleife nur zum Lesen der Werte. Eine Veränderung der Werte der Reihung ist mit diesem Schleifenkonstrukt nicht möglich.

Aufgabe 8: Erweitern Sie Beispiel 3 um eine Operation `bestimmeMinimum` und `bestimmeDurchschnitt`, welche den kleinsten Messwert bzw. den Durchschnitt aller Messwerte bestimmen. Verwenden Sie bei der Implementierung eine *for-each*-Schleife.

Aufgabe 9: Erweitern Sie Beispiel 3 um

- eine Operation `istEnthalten(suchWert: Fließkommazahl): Wahrheitswert`, die `wahr` zurückgibt, wenn der als Parameter übergebene Wert `suchWert` in der global definierten Reihung `messwerte` enthalten ist.
- eine Operation `suche(suchWert: Fließkommazahl): Ganzzahl`, die eine Position zurückgibt, an der die global definierte Reihung `messwerte` den als Parameter übergebenen Wert `suchWert` enthält. Wenn der Wert nicht enthalten ist, wird `-1` zurückgegeben.
- Erläutern Sie, welche Art der *for*-Schleife jeweils geeignet ist, um die Operationen `istEnthalten` und `suche` zu implementieren.

Einsatz von Reihungen und Zeichenketten in Anwendungsaufgaben aus der Kryptologie

Aufgabe 10: Monoalphabetische Substitutionsverfahren ersetzen jedes Zeichen im Klartext durch ein zugehöriges Geheimtextzeichen. Ein solches Ersetzungsverfahren soll hier implementiert werden.

- a) Erzeugen Sie eine Reihung vom Inhaltstyp Zeichen, die Sie mit 26 unterschiedlichen Zeichen füllen. Das erste Zeichen soll später das 'a' ersetzen, das zweite das 'b' usw.
- b) Implementieren Sie ein Programm, das einen Klartext vom Anwender erfragt, alle Ersetzungen vornimmt und anschließend den Geheimtext ausgibt.

Hinweis1: Die Methode `toLowerCase()` gibt die Zeichenkette, für die die Methode aufgerufen wird, in Kleinbuchstaben zurück.

Hinweis2: Die Umwandlung eines Zeichens in den ASCII-Code ist hilfreich, um die zugehörige Position in der Reihung der Geheimtextzeichen zu bestimmen.

- c) Erläutern Sie, warum eine Dechiffrierung in den Klartext anhand der vorhandenen Reihung aufwändiger ist. Wie könnte man dieses Problem lösen?

Aufgabe 11: Um Geheimtexte, die mit einem Ersetzungsverfahren erzeugt wurden, zu knacken, ist es hilfreich die jeweilige Anzahl der einzelnen Zeichen zu kennen und vergleichen zu können. Dazu soll in den folgenden Teilaufgaben ein Analyseprogramm entstehen.

- a) Erstellen Sie ein Programm, das die jeweilige Anzahl der Buchstaben von A bis Z in einem beliebigen Text, den der Anwender eingibt, ermittelt und in einer Reihung vom Inhaltstyp Typ Ganzzahl speichert. Groß- und Kleinschreibung sollen dabei nicht unterschieden werden.
- b) Implementieren Sie die Umrechnung der absoluten Häufigkeiten in relative Häufigkeiten.
- c) Stellen Sie die Häufigkeitsverteilung der Zeichen in geeigneter Form dar, z. B. in einem Histogramm.
- d) Verwenden Sie die in der Reihung gespeicherten Werte, um das häufigste Geheimtextzeichen zu bestimmen.
- e) Sortieren Sie die Zeichen nach ihrer Häufigkeit.

Aufgabe 12: Vergleichen Sie den Aufbau einer Zeichenkette (Datentyp `String`) mit einer Reihung vom Inhaltstyp Zeichen (`char[]`). Erläutern Sie, welche Gemeinsamkeiten und welche Unterschiede es gibt.

Zweidimensionale Reihungen

Für die Wertepaare der Koordinaten x und y in Aufgabe 5 und 7 haben wir zwei Reihungen angelegt und darauf vertraut, dass die Werte, die zusammengehören, an der gleichen Position in den Reihungen stehen. Stattdessen können wir auch eine zweidimensionale Reihung anlegen, die wir uns wie eine Tabelle oder eine Matrix mit Zeilen und Spalten vorstellen können. Im Falle der x- und y-Koordinate für verschiedene Blumen oder Windmühlen hätten wir eine zweidimensionale Reihung mit einer Zeile für jede Figur. In jeder Zeile gibt es zwei Spalten. Spalte 1 für die x-Koordinate und Spalte 2 für die y-Koordinate.

| | | | |
|-------------|---|-----|-----|
| koordinaten | 0 | 30 | 34 |
| | 1 | 70 | 87 |
| | 2 | 29 | 113 |
| | 3 | 99 | 44 |
| | 4 | 178 | 209 |
| | 5 | 312 | 340 |
| | 6 | 112 | 233 |
| | 7 | 76 | 122 |
| | 8 | 362 | 370 |
| | 9 | 77 | 205 |
| | | 0 | 1 |

```
int[][] koordinaten = new int[10][2];
```

Beispiel 4: Erzeugen einer zweidimensionalen Reihung

```
int[][] koordinaten = {{30,34}, {70,87},  
                       {29, 113}, {99, 44}, {178, 209}, {312,  
                       340}, {112, 233}, {76, 122}, {362,370},  
                       {77, 205}};
```

Beispiel 5: Erzeugen einer zweidimensionalen Reihung mit Initialwerten

```
koordinaten[4][1]
```

Beispiel 6: Zugriff auf einen Wert in der zweidimensionalen Reihung

Abbildung 2: Zweidimensionale Reihung mit zehnmal zwei Einträgen

Beispiel 4 zeigt, wie das Gerüst für die zweidimensionale Reihung erzeugt wird. Wir legen hier fest, dass wir die erste Zahl in eckigen Klammern als Anzahl der Zeilen und die zweite Zahl in eckigen Klammern als Anzahl der Spalten interpretieren.⁵ In Beispiel 4 wird eine Reihung mit lauter Nullen erzeugt. Möchten wir beim Erzeugen hingegen gleich die Werte aus Abbildung 2 eintragen, müssten wir den Code aus Beispiel 5 verwenden. Beispiel 6 zeigt, wie auf einen einzelnen Wert der zweidimensionalen Reihung zugegriffen werden kann.

Aufgabe 13: Die folgenden Aufgaben beziehen sich auf die zweidimensionale Reihung in Abbildung 2.

- Geben Sie an, welchen Wert der Aufruf `koordinaten[4][1]` zurückgibt.
- Ergänzen Sie die Indizes in dem Ausdruck `koordinaten[][]` so, dass der Wert 70 aus der Reihung ausgelesen wird.
- Erläutern Sie anhand des Codes in Beispiel 4, dass es sich bei einer zweidimensionalen Reihung genau genommen um eine Reihung von Reihungen handelt.
- Stellen Sie eine Vermutung auf, welche Werte die Aufrufe `koordinaten.length` bzw. `koordinaten[0].length` zurückgeben.

⁵ Diese Interpretation ist auch die im niedersächsischen Abitur ab 2027 übliche, wenn keine andere Definition explizit gegeben ist (vgl. Niedersächsisches Kultusministerium (Hrsg.) (2025) *Ergänzende Hinweise zum Kerncurriculum Informatik für die gymnasiale Oberstufe am Gymnasium, an der Gesamtschule sowie für das Kolleg*. <https://cuvo.nibis.de/index.php?p=download&upload=736> [Datum des Zugriffs: 13.08.2025])

Beispiel 7 zeigt, wie die Werte einer zweidimensionalen Reihung systematisch ausgelesen bzw. durchlaufen werden können:

```
1 int[][] koordinaten = {{30,34}, {70,87}, {29, 113}, {99, 44}, {178,
2   209}, {312, 340}, {112, 233}, {76, 122}, {362,370}, {77, 205}};
3 void setup(){
4   for(int i = 0; i < koordinaten.length; i++){
5     for(int j = 0; j < koordinaten[i].length; j++){
6       System.out.print(" | " + koordinaten[i][j]);
7     }
8     System.out.println();
9   }
```

Beispiel 7: Systematisches Auslesen der Werte einer zweidimensionalen Reihung.

Aufgabe 14:

- Geben Sie der Reihe nach an, welche Indexpaare die Variablen `i` und `j` in Beispiel 7 bilden.
- Stellen Sie eine Vermutung auf, wie die Ausgabe des Programms aussieht. Überprüfen Sie Ihre Vermutung anschließend, indem Sie Beispiel 7 ausführen.

Auch für das systematische Durchlaufen einer zweidimensionalen Reihung kann mithilfe von *for-each*-Schleifen erfolgen. Beispiel 8 zeigt die Implementierung von Beispiel 7 mit zwei geschachtelten *for-each*-Schleifen.

```
1 int[][] koordinaten = {{30,34}, {70,87}, {29, 113}, {99, 44}, {178,
2   209}, {312, 340}, {112, 233}, {76, 122}, {362,370}, {77, 205}};
3 void setup(){
4   for(int[] r : koordinaten){
5     for(int j : r){
6       System.out.print(" | " + j);
7     }
8     System.out.println();
9   }
```

Beispiel 8: Systematisches Auslesen der Werte einer zweidimensionalen Reihung mit for-each-Schleifen

Aufgabe 15: Erläutern Sie den Aufbau der äußeren und der inneren *for-each*-Schleife in Beispiel 8.

Aufgabe 16: Nehmen Sie für diese Aufgabe noch einmal die Methoden `blumeZeichnen(int x, int y)` aus Aufgabe 5 oder `windmuehleZeichnen(int x, int y, int winkel)` aus Aufgabe 7 zur Hilfe.

- Erstellen Sie ein Programm, das 15 Blumen oder Windmühlen an zufälligen Positionen zeichnet. Verwenden Sie dafür eine zweidimensionale Reihung.
- Erweitern Sie Ihre Reihung um weitere Spalten, um für die Figuren weitere individuelle Werte wie Farben oder den Drehwinkel der Windmühle zu speichern.

Aufgabe 17:

- Erstellen Sie eine zweidimensionale Reihung mit mehreren englischen Vokabeln und ihrer deutschen Übersetzung. Die Vokabeln sollen nacheinander abgefragt werden. Speichern Sie für jede Vokabel, ob sie richtig oder falsch beantwortet wurde. Fragen Sie die falsch übersetzten Vokabeln so lange erneut ab, bis der Anwender alle Vokabeln richtig übersetzt hat.

- b) Erweitern Sie Ihr Programm um die Möglichkeit, sich zu einem Wort die Übersetzung ausgeben zu lassen.
- c) Diskutieren Sie, ob es für Aufgabenteil b) von Vorteil ist, wenn die Vokabeln alphabetisch sortiert in der Reihung vorliegen.
- d) Erweitern Sie Ihr Programm um die Möglichkeit, dass zufällig drei unterschiedliche Vokabeln aus der Reihung abgefragt werden.

Aufgabe 18: Erstellen Sie ein Programm für ein Tic-Tac-Toe-Spiel. Unterteilen Sie den Bildschirm dazu in 9 Felder, die die Spieler*innen abwechselnd mit der Maus anklicken können. Speichern Sie die aktuelle Belegung des Spielfeldes in einer zweidimensionalen Reihung.

Aufgabe 19: In Aufgabe 11 wurde die Häufigkeit der einzelnen Zeichen in einem Text bestimmt. Eine zweidimensionale Reihung ermöglicht uns nun, den ASCII-Code des Zeichens und die Anzahl gemeinsam in einer zweidimensionalen Reihung zu speichern.

Legen Sie eine entsprechende Reihung an und sortieren Sie diese anschließend nach der Häufigkeit wie in Aufgabe 11 e.

Aufgabe 20: Bei der Verschlüsselung eines Textes durch Transposition geht es darum, die Reihenfolge der Zeichen zu verändern, um den Text unlesbar zu machen. Eine Möglichkeit, die Zeichen eines Textes systematisch zu vertauschen, besteht darin, den Text zeilenweise in eine zweidimensionale Reihung zu schreiben und ihn anschließend spaltenweise auszugeben. Leerzeichen werden aus dem Text entfernt. Freie Felder werden mit x oder zufälligen Zeichen aufgefüllt.

Beispiel: Der Klartext „Achtung hier kommt eine geheime Nachricht. Der Schatz wurde gefunden.“ wird in eine zweidimensionale Reihung mit 8 Zeilen und 8 Spalten zeilenweise ohne Leerzeichen eingetragen. Die letzten vier Felder werden mit x aufgefüllt. Anschließend wird der Text spaltenweise ausgelesen, so dass sich die verschlüsselte Nachricht *AieiiSrdceimccdehrrnehhentkeNtag.uoga.texnmecDzfxgmhhewuxhterrunx* ergibt.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | c | h | t | u | n | g | h |
| i | e | r | k | o | m | m | t |
| e | i | n | e | g | e | h | e |
| i | m | e | N | a | c | h | r |
| i | c | h | t | . | D | e | r |
| S | c | h | a | t | z | w | u |
| r | d | e | g | e | f | u | n |
| d | e | n | . | x | x | x | x |

Abbildung 3: 8x8 Reihung mit Beispielttext

- a) Erstellen Sie ein Programm, das zu einem Text, den der Anwender eingibt, den Geheimtext nach dem beschriebenen Verfahren erstellt. Verwenden Sie zunächst eine zweidimensionale Reihung fester Größe mit 8 Zeilen und 8 Spalten.
- b) Begründen Sie, dass in diesem Fall eine erneute Verschlüsselung des Geheimtextes wieder den Klartext ergibt.
- c) Verändern Sie Ihr Programm so, dass als Schlüssel die Anzahl der Zeilen festgelegt werden kann. Die Anzahl der Spalten ergibt sich dann aus der Länge des Textes und der Anzahl der Zeilen.
- d) Ergänzen Sie die Möglichkeit einen Geheimtext wieder zu entschlüsseln. Auch hier muss als Schlüssel zusätzlich die Zeilenzahl eingegeben werden.

Aufgabe 21: Das Programm *Aufgabe21* bietet dem Anwender die Möglichkeit Pixelgrafiken mit schwarzen und weißen Pixeln zu erstellen. Die Pixel im linken Bild ändern ihre Farbe, wenn sie angeklickt werden. Auf der rechten Seite erscheint eine Kopie des Bildes.

Für das Bild, das auf der linken Seite erstellt wird, sollen verschiedene Transformationen durchgeführt werden können. Das Ergebnis wird jeweils auf der rechten Seite angezeigt. Auch die Hintereinanderausführung mehrerer Transformationen soll möglich sein.

- Testen Sie das Programm und erläutern Sie den bereits vorhandenen Quellcode.
- Implementieren Sie die Methoden `spiegelBildZeichnen` und `drehungRechts` so, dass das in der zweidimensionalen Reihung `kopie` gespeicherte Bild vertikal gespiegelt bzw. um 90° gedreht wird. Abbildung 4 und 5 zeigen jeweils ein Beispiel.
- Ergänzen Sie weitere Transformationen, z. B. horizontal spiegeln, um 90° nach links drehen, an der Diagonalen spiegeln usw.
- Bei einem Rechtsklick auf ein Pixel des linken Bildes sollen das ausgewählte Pixel und die acht umliegenden Pixel die Farbe wechseln.

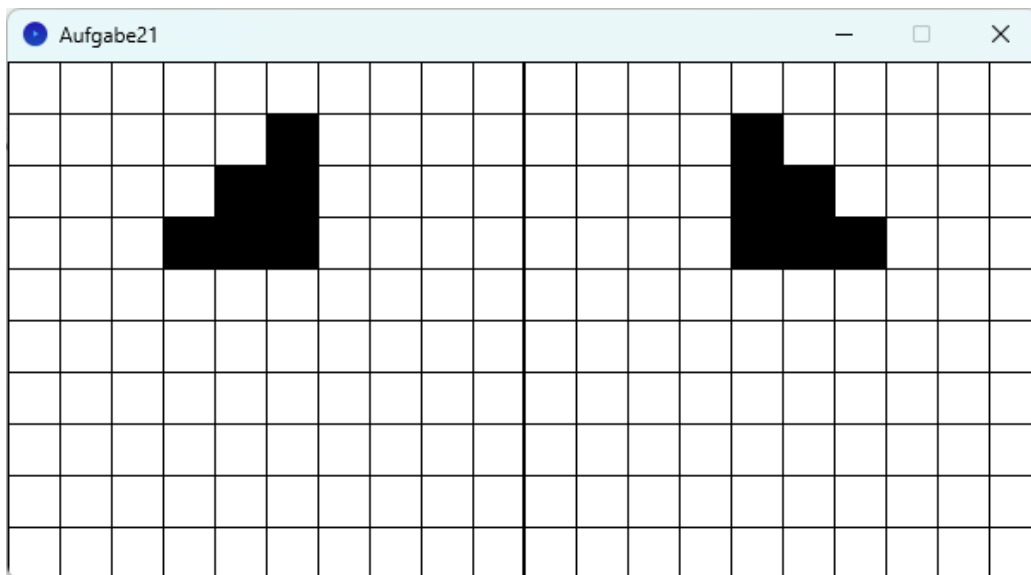


Abbildung 4: vertikal gespiegeltes Bild

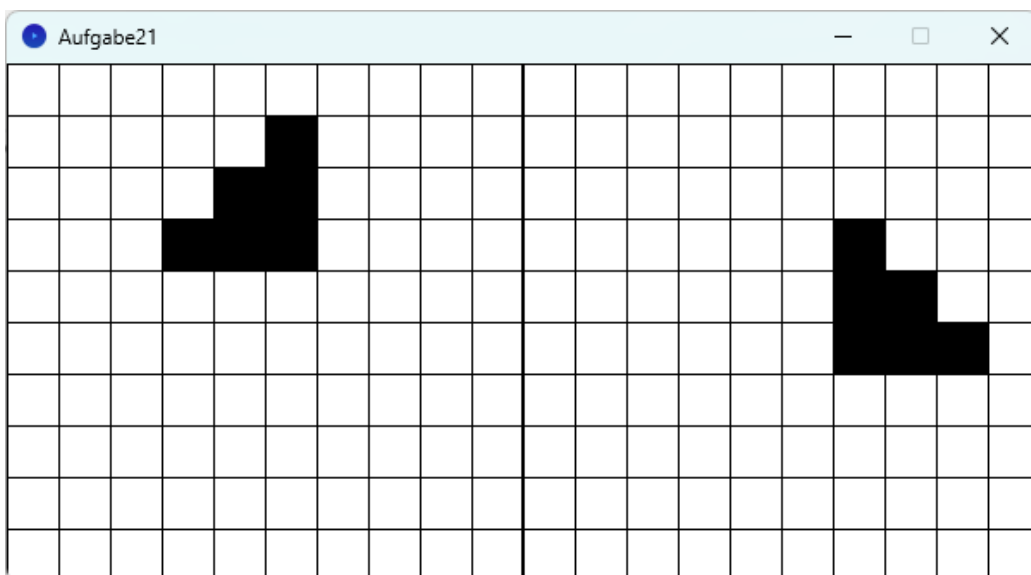
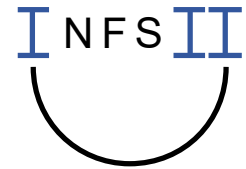


Abbildung 5: Um 90° gedrehtes Bild



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 Arbeitsblatt 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.