

Programmieren! – Teil 2

Prolog 2019

Stefan Podlipnig, TU Wien

Überblick



Funktionen

Funktionen allgemein

- Funktion (wie z. B. line)
 - Unterstützt Modularisierung und Wiederverwendung
 - Einmal Code schreiben
 - Mehrfach an vielen Stellen mit unterschiedlichen Parametern verwenden
 - Unterstützt Abstraktion
 - Man muss den Ablauf nicht genau kennen
 - Für den Aufruf nur wichtig
 - Welchen Input (Parameter) kann man übergeben
 - Welche Auswirkung hat der Aufruf (Output, Rückgabewert)?
- Solche Funktionen kann man auch selbst schreiben

Beispiel

- Ein Kreuz zeichnen

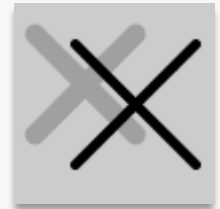
```
size(100,100);  
background(200);  
stroke(160);  
strokeWeight(10);  
line(10, 15, 60, 65);  
line(60, 15, 10, 65);
```



- Zwei Kreuze zeichnen

- Duplizierter Code mit ähnlichen Werten!

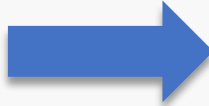
```
size(100, 100);  
background(200);  
stroke(160);  
strokeWeight(10);  
line(10, 15, 60, 65);  
line(60, 15, 10, 65);  
stroke(0);  
strokeWeight(5);  
line(30, 20, 90, 80);  
line(90, 20, 30, 80);
```



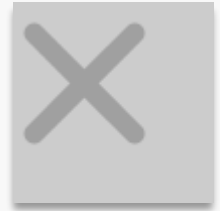
Beispiel mit Funktion (erster Schritt)

- Code wird in eine Funktion verpackt

```
void drawX() {  
  stroke(160);  
  strokeWeight(10);  
  line(10, 15, 60, 65);  
  line(60, 15, 10, 65);  
}
```



```
void setup() {  
  size(100, 100);  
  background(200);  
  drawX();  
}  
  
void drawX() {  
  stroke(160);  
  strokeWeight(10);  
  line(10, 15, 60, 65);  
  line(60, 15, 10, 65);  
}
```



- Funktion setup dient als Startpunkt für Processing-Programme (andere Funktionen aufrufen)

Anatomie der drawX-Funktion

Rückgabetyp
void = keine Rückgabe

Name der Funktion

```
void drawX() {  
    stroke(160);  
    strokeWeight(10);  
    line(10, 15, 60, 65);  
    line(60, 15, 10, 65);  
}
```

Code, der beim Aufruf
drawX(); ausgeführt wird

Auszuführender Code muss
zwischen { und } stehen

Ablauf (Beispiel)

```
void setup() {  
  size(100, 100);  
  background(200);  
  drawX();  
  line(40, 15, 100, 65);  
}
```

```
void drawX() {  
  stroke(160);  
  strokeWeight(10);  
  line(10, 15, 60, 65);  
  line(60, 15, 10, 65);  
}
```


Erweiterung der drawX-Funktion (1)

- Parameter für Grauwert

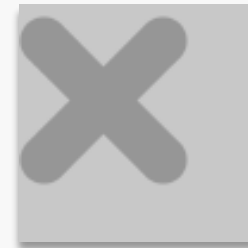
```
void setup() {  
  size(100, 100);  
  background(200);  
  drawX(100);  
}  
  
void drawX(int grayValue) {  
  stroke(grayValue);  
  strokeWeight(10);  
  line(10, 15, 60, 65);  
  line(60, 15, 10, 65);  
}
```



Erweiterung der drawX-Funktion (2)

- Parameter für Grauwert und Dicke

```
void setup() {  
  size(100, 100);  
  background(200);  
  drawX(150, 20);  
}  
  
void drawX(int grayValue, int weight) {  
  stroke(grayValue);  
  strokeWeight(weight);  
  line(10, 15, 60, 65);  
  line(60, 15, 10, 65);  
}
```



Anatomie der erweiterten drawX-Funktion

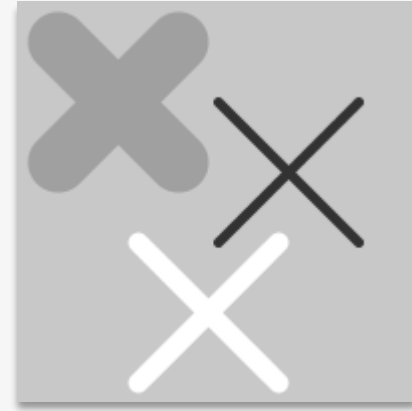
Parameter (wenn mehrere, dann durch Beistrich getrennt)
Form für jeden einzelnen Parameter: Datentyp Name

```
void drawX(int grayValue, int weight) {  
    stroke(grayValue);  
    strokeWeight(weight);  
    line(10, 15, 60, 65);  
    line(60, 15, 10, 65);  
}
```

Verwendung eines
Parameters

Erweiterung der drawX-Funktion (3)

```
void setup() {  
  size(200, 200);  
  background(200);  
  drawX(160, 30, 20, 20, 60);  
  drawX(50, 5, 100, 50, 70);  
  drawX(255, 10, 60, 120, 70);  
}
```



```
void drawX(int grayValue, int weight, int x, int y, int size) {  
  stroke(grayValue);  
  strokeWeight(weight);  
  line(x, y, x + size, y + size);  
  line(x + size, y, x, y + size);  
}
```

Processing-Funktion draw

- Code innerhalb von draw wird kontinuierlich ausgeführt
 - Ca. 60 mal pro Sekunde (kann mit `frameRate` eingestellt werden)
- Beispiel

```
void setup() {  
    size(200, 200);  
    frameRate(5);  
}  
  
void draw() {  
    background(200);  
    int grayValue = int(random(255));  
    int thickness = int(random(20));  
    int x = int(random(width / 10, width / 2));  
    int y = int(random(height / 10, height / 2));  
    drawX(grayValue, thickness, x, y, 100);  
}  
  
void drawX(int grayValue, int weight, int x, int y, int size) {  
    ...  
}
```

Globale und lokale Variablen – Sichtbarkeit

- Globale Variablen
 - Außerhalb von Funktionen
 - In jeder Funktion sichtbar
- Lokale Variablen
 - Innerhalb von Funktionen bzw. Blöcken
 - Block = Programmeinheit, in der lokale Deklarationen getroffen werden können (sind nur dort bekannt)
 - Ein Block entspricht einer Verbundanweisung von { bis }

Beispiel (lokale/globale Variablen)

```
int counter = 1;
int increment = 1;

void setup() {
  size(500, 500);
}

void draw() {
  background(counter);
  drawShape();
  if (counter == 255 || counter == 0) {
    increment *= -1;
  }
  counter += increment;
}

void drawShape() {
  int fillColour = 255 - counter;
  fill(fillColour);
  ellipse(height / 2, width / 2, counter + 100, counter + 100);
}
```

Globale Variablen -
sind in allen Funktionen sichtbar und
können überall verwendet werden

Lokale Variable -
nur in drawShape
sichtbar

Rückgabe

- Eine Funktion gibt immer etwas zurück
 - void, wenn „Nichts“ zurückgeliefert wird (z. B. nur Ausgabe produzieren)
 - Sonst muss der Typ vor der Funktion angegeben werden
 - Wert von diesem Typ wird mit return zurückgeliefert

- Beispiel

Typangabe

```
void setup() {  
    float f = average(12.0, 6.0);  
    println(f);  
}  
  
float average(float num1, float num2) {  
    float av = (num1 + num2) / 2.0;  
    return av;  
}
```

Beim Rücksprung wird Wert zurückgeliefert

Beispiele (Maximum zweier Zahlen)

Mehrere return-Anweisungen möglich

```
int max1(int a, int b) {  
    if (a > b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

```
int max2(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
    return b;  
}  
...
```

...

```
int max3(int a, int b) {  
    return a > b ? a : b;  
}
```

Bedingungsoperator (funktioniert wie einfaches if-else)

```
void setup() {  
    println(max1(10, 20));  
    println(max2(10, 20));  
    println(max3(10, 20));  
}
```

ggt-Berechnung (Wiederholung)

benennen und
parametrisieren

vergleichen und
verzweigen

wiederholen

```
int ggt(int a, int b) {  
    int first = a;  
    int second = b;  
    if (first == 0) {  
        return second;  
    } else {  
        while (second != 0) {  
            if (first > second) {  
                first -= second;  
            } else {  
                second -= first;  
            }  
        }  
    }  
    return first;  
}  
  
void setup() {  
    println(ggt(12, 44));  
    println(ggt(10, 20));  
    println(ggt(13, 1234));  
    println(ggt(2856, 12568));  
}
```

speichern

rechnen

while-Schleife:

Form:
while(test) {
 statements
}

Initialisierung: vor der Schleife
Weiterschalten: im Schleifenrumpf

ggt-Berechnung (kürzere Variante)

```
int ggt(int a, int b) {  
    if (a == 0) return b;  
    else  
        while (b != 0)  
            if (a > b) a -= b;  
            else b -= a;  
    return a;  
}  
  
void setup() {  
    println(ggt(12, 44));  
    println(ggt(10, 20));  
    println(ggt(13, 1234));  
    println(ggt(2856, 12568));  
}
```

Kürzere Variante:

- Parameter als Variablen benutzen und verändern (kein guter Stil)
- Keine Klammern, da immer nur eine Anweisung pro Schachtelungstiefe

Rekursion

- Eine Funktion heißt **rekursiv**, wenn sie sich selbst wieder aufruft
 - Dazu zählen auch indirekte Funktionsaufrufe (z. B. der Aufruf einer anderen Funktion, die wiederum die ursprüngliche Funktion aufruft)
- Grundprinzip der Rekursion
 - Zurückführen einer allgemeinen Aufgabe auf eine einfachere Aufgabe derselben Klasse

Rekursion (ein einfaches Beispiel)

- **Iterativ** ist die Summe von n Zahlen definiert durch
 - $\text{sum}(n) = 0 + 1 + 2 + \dots + n$
- **Rekursiv** ist die Summe definiert durch

$$\text{sum}(n) = \begin{cases} 0 & \text{falls } n = 0 & \text{Rekursionsanfang} \\ \text{sum}(n-1) + n & \text{sonst} & \text{Rekursionsschritt} \end{cases}$$

Beispiel (Code, Ablauf für $n = 3$)

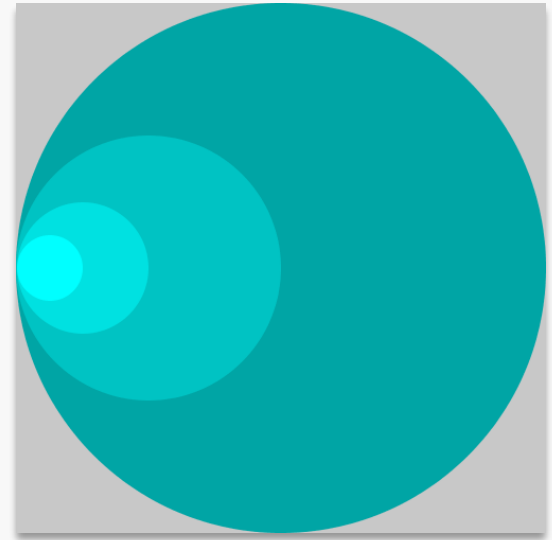
```
void setup() {  
    println(sumIterative(10));  
    println(sumRecursive(10));  
}  
  
int sumIterative(int num) {  
    int i, sum = 0;  
    for (i = 1; i <= num; i++) {  
        sum += i;  
    }  
    return sum;  
}  
  
int sumRecursive(int num) {  
    if (num > 0) {  
        return num + sumRecursive(num - 1);  
    } else {  
        return 0;  
    }  
}
```

Rekursion für sumRecursive(3)

$\begin{aligned} \text{sumRecursive}(3) &= 3 + \text{sumRecursive}(2) \\ \text{sumRecursive}(2) &= 2 + \text{sumRecursive}(1) \\ \text{sumRecursive}(1) &= 1 + \text{sumRecursive}(0) \\ \text{sumRecursive}(0) &= 0 \\ \text{sumRecursive}(1) &= 1 + 0 = 1 \\ \text{sumRecursive}(2) &= 2 + 1 = 3 \\ \text{sumRecursive}(3) &= 3 + 3 = 6 \end{aligned}$
--

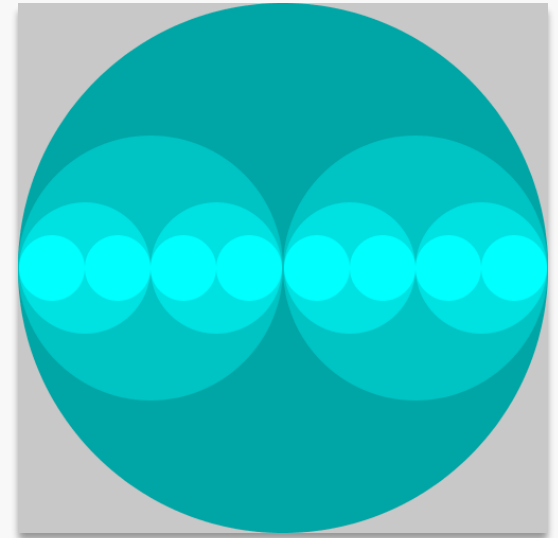
Ein visuelles Beispiel ...

```
void setup() {  
  size(500,500);  
  noStroke();  
}  
  
void draw() {  
  background(200);  
  drawCircle(width / 2, height / 2, 3);  
  noLoop();  
}  
  
void drawCircle(int x, int radius, int num) {  
  fill(0, 255 - num * 30.0, 255 - num * 30.0);  
  ellipse(x, height / 2, radius * 2, radius * 2);  
  if (num > 0) {  
    drawCircle(x - radius / 2, radius / 2, num - 1);  
  }  
}
```

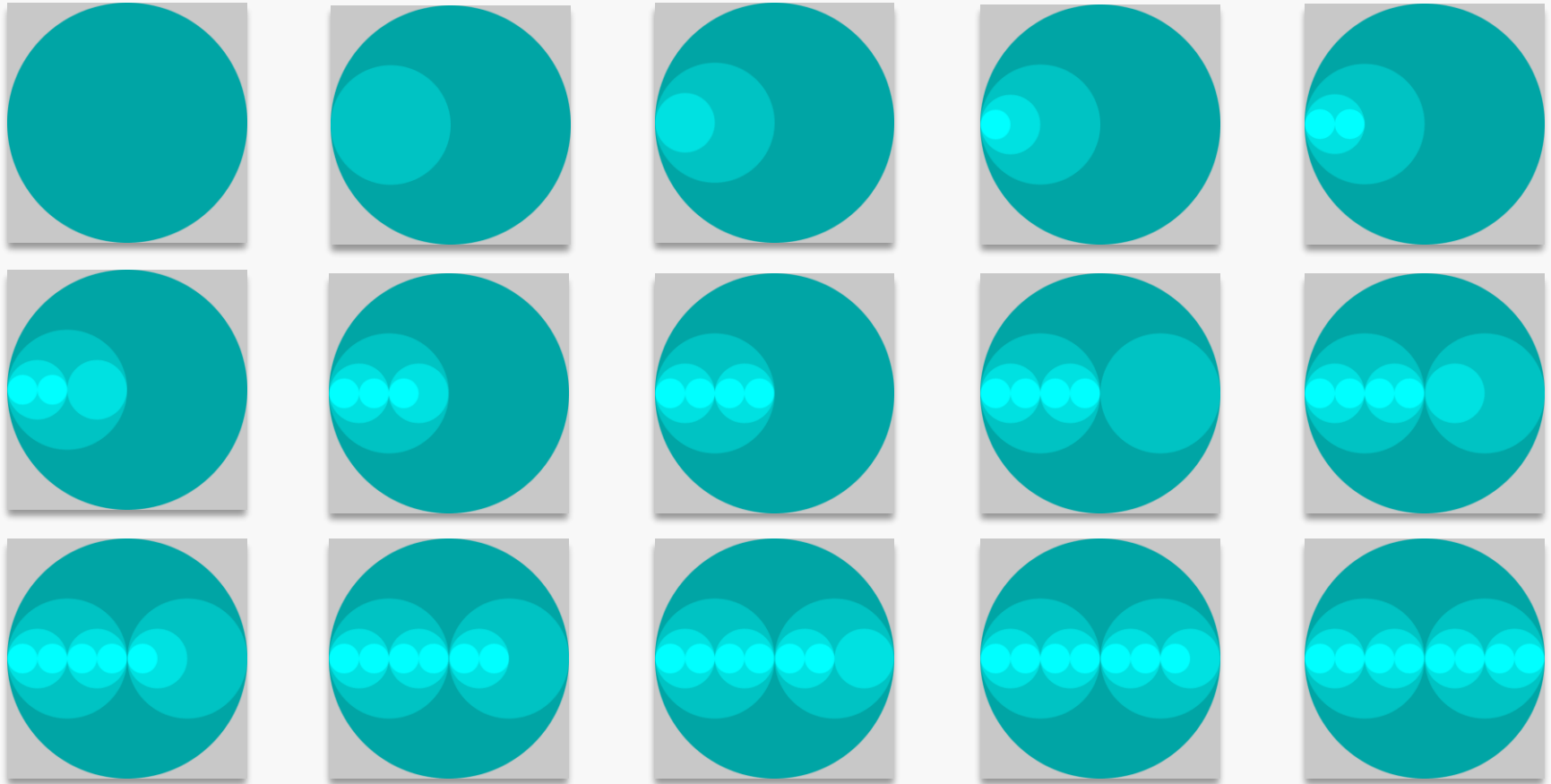


... und was wird jetzt gezeichnet?

```
void setup() {  
  size(500,500);  
  noStroke();  
}  
  
void draw() {  
  background(200);  
  drawCircle(width / 2, height / 2, 3);  
  noLoop();  
}  
  
void drawCircle(int x, int radius, int num) {  
  fill(0, 255 - num * 30.0, 255 - num * 30.0);  
  ellipse(x, height / 2, radius * 2, radius * 2);  
  if (num > 0) {  
    drawCircle(x - radius / 2, radius / 2, num - 1);  
    drawCircle(x + radius / 2, radius / 2, num - 1);  
  }  
}
```



Wie läuft diese Rekursion ab?



Arrays

Arrays

- Zusammenfassung von mehreren Elementen gleichen Typs
- Deklaration
 - Form: Datentyp[] Name
 - Beispiel: `int[] number;`
 - Achtung
 - Legt nur fest, dass number ein Array von ganzen Zahlen ist
 - Es wird noch keine Größe angegeben

Anlegen (Instanziierung) von Arrays

- Anlegen bei Deklaration

```
int[] arr = new int[10];
```

- Späteres Anlegen

```
int[] arr;
```

```
...
```

```
arr = new int[10];
```

- Hinweis

- Die Größe des Arrays kann mit `arr.length` abgefragt werden

Anlegen von Arrays (Beispiel int-Arrays)

- Beispiel

```
int[] arr = new int[10];
```

- Die Arrayelemente haben zunächst durch eine implizite Initialisierung alle den Wert 0 (bei int)
- Jedem Element ist ein Index vom Typ int zugewiesen
 - Indexzählung beginnt bei **0**
 - Indexzählung geht bis **Länge-1**
- Schematisch (nach dem Anlegen und Initialisieren)

Index	0	1	2	3	4	5	6	7	8	9
Inhalt	0	0	0	0	0	0	0	0	0	0

Anlegen von Arrays – mit expliziter Initialisierung

- Beispiel

```
int[] arr = { 3, 4, 5, 6, 7 };
```

// oder

```
int[] arr;
```

...

```
arr = new int[] { 3, 4, 5, 6, 7 };
```

- Array hat die Länge 5
- Array enthält die Elemente 3, 4, 5, 6, 7

Verwenden von Arrays

- Zugriff über Index
- Beispiel

```
int[] arr = new int[5];  
arr[0] = 3;  
arr[1] = arr[0] + 4;  
printArray(arr);
```

3
7
0
0
0

- Beispiel

```
int[] x = {50, 61, 83, 69, 71, 50, 29, 31, 17, 39};  
fill(0);  
for (int i = 0; i < x.length; i++) {  
    rect(0, i * 10, x[i], 8);  
}
```



Arrayzugriff – Exception

- Indexwert muss gültig sein
 - **Sonst Fehler, d.h. das Programm wird sofort unterbrochen**
- Beispiel (erzeugt `ArrayIndexOutOfBoundsException`)

```
int[] arr = new int[5];  
arr[0] = 3;  
arr[5] = arr[0] + 4;  
printArray(arr);
```

Index 5 ist nicht zulässig!

Beispiel – Anwendung bei draw

- Mauszeiger verfolgen
- Aktuelle Mausposition
 - mouseX
 - mouseY
- 2 Arrays
 - Die letzten 100 Werte

```
int max = 100;
int[] x = new int[max];
int[] y = new int[max];

void setup() {
  fullscreen();
}

void draw() {
  background(0);
  for (int i = max - 1; i > 0 ; i--) {
    x[i] = x[i - 1];
    y[i] = y[i - 1];
  }
  x[0] = mouseX;
  y[0] = mouseY;
  for (int i = 0; i < max - 1 ; i++) {
    float radius = i / 2.0;
    ellipse(x[i], y[i], radius, radius);
  }
}
```

Beispiel – Mauszeiger verfolgen (Variante 2)

- Lösung mit Ringpuffer
 - Daten „im Kreis“ einfügen
 - index ist aktuelle Position

```
int max = 100;
int[] x = new int[max];
int[] y = new int[max];
int index = 0;


void setup() {
  fullscreen();
}

void draw() {
  background(0);
  x[index] = mouseX;
  y[index] = mouseY;
  index = (index + 1) % max;
  for (int i = 0; i < max; i++) {
    int pos = (index + i) % max;
    float radius = (max - i) / 2.0;
    ellipse(x[pos], y[pos], radius, radius);
  }
}
```

Zuweisung

- Arrayvariable ist eine Referenz auf das eigentliche Array
- Einer Arrayvariable vom Typ x kann immer nur ein Array vom Typ x zugewiesen werden
 - Bei der Zuweisung wird nur die Adresse im Speicher kopiert, **nicht** der Inhalt
- Beispiel

```
int[] x = new int[5], y;  
y = x;  
y[1] = 7;  
println(y[0] + " " + x[0]);  
println(y[1] + " " + x[1]);
```



0	0
7	7

Zuweisung – Erklärung

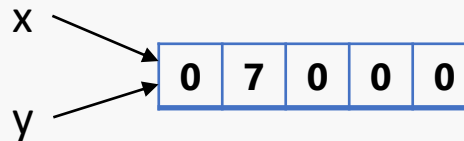
- Beispiel

```
int[] x = new int[5], y;  
y = x;  
y[1] = 7;  
println(y[0] + " " + x[0]);  
println(y[1] + " " + x[1]);
```

0	0
7	7

- Erklärung

- y und x sind Arrayvariablen und zeigen auf den gleichen Speicherbereich
- Die Zuweisung bei y[1] verändert auch x[1]



Beispiel (Array als Parameter, Rückgabetyp)

```
float[] data = {19.0, 40.0, 75.0, 76.0, 90.0};  
float[] halfData;
```

```
void setup() {  
    halfData = halve(data);  
    printArray(halfData);  
}
```

Es wird nicht der Inhalt des Arrays sondern ein Verweis darauf übergeben

```
float[] halve(float[] data) {  
    float[] numbers = new float[data.length];  
    arrayCopy(data, numbers);  
    for (int i = 0; i < numbers.length; i++) {  
        numbers[i] = numbers[i] / 2.0;  
    }  
    return numbers;  
}
```

Parametertyp,
Rückgabetyp:
float[]

Kopie eines
Arrays anlegen

[0]	9.5
[1]	20.0
[2]	37.5
[3]	38.0
[4]	45.0

Beispiel – Minimum in einem Array suchen

```
void setup() {  
    int[] numbers = new int[5];  
    for (int i = 0; i < numbers.length; i++) {  
        numbers[i] = int(random(100));  
    }  
    printArray(numbers);  
    print("Minimum = " + findMinimum(numbers));  
}
```

```
int findMinimum(int[] data) {  
    int min = data[0];  
    for (int i = 1; i < data.length; i++) {  
        if (data[i] < min) {  
            min = data[i];  
        }  
    }  
    return min;  
}
```

Achtung: Setzt voraus, dass
das Array data zumindest
1 Element enthält

```
[0] 21  
[1] 46  
[2] 29  
[3] 7  
[4] 86  
Minimum = 7
```

Beispiel – Sortieren eines Arrays

```
void setup() {  
    int[] numbers = new int[5];  
    for (int i = 0; i < numbers.length; i++) {  
        numbers[i] = int(random(100));  
    }  
    printArray(numbers);  
    sortArray(numbers);  
    println();  
    printArray(numbers);  
}
```

Einfacher (elementarer)
Sortieralgorithmus
(Selectionsort)

```
void sortArray(int[] data) {  
    int n = data.length;  
    for (int i = 0; i < n; i++) {  
        int min = i;  
        for (int j = i + 1; j < n; j++) {  
            if (data[j] < data[min]) {  
                min = j;  
            }  
        }  
        int swap = data[i];  
        data[i] = data[min];  
        data[min] = swap;  
    }  
}
```

[0]	12
[1]	45
[2]	77
[3]	37
[4]	56

[0]	12
[1]	37
[2]	45
[3]	56
[4]	77

Zweidimensionale Arrays

- Zweidimensionales Array (für Matrizen, Bilddaten etc.)
 - Array von Arrays
- Beispiel

```
int[][] x = { {50, 0}, {61, 204}, {83, 51}, {69, 102}, {71, 0},  
              {50, 153}, {29, 0}, {31, 51}, {17, 102}, {39, 204} };
```

```
println(x[0][0]);
```

```
println(x[0][1]);
```

```
println(x[3][0]);
```

```
println(x[7][1]);
```

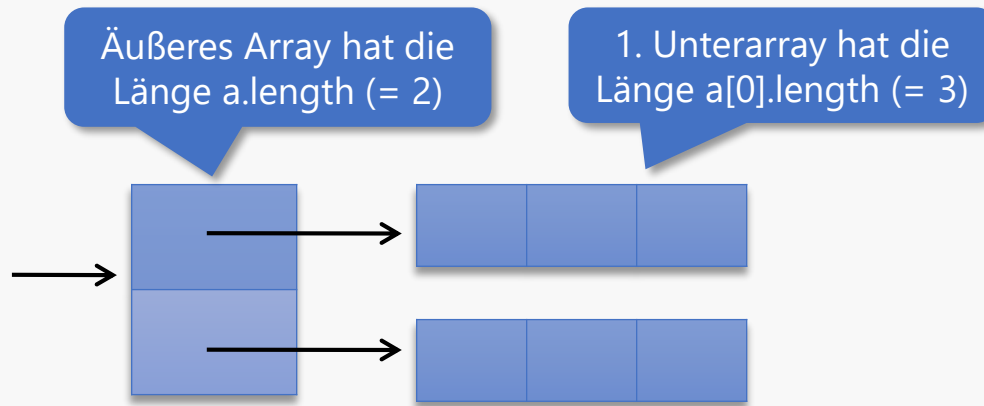
50	0
61	204
83	51
69	102
71	0
50	153
29	0
31	51
17	102
39	204

50
0
69
51

Programmieren!

Zweidimensionales Array – Aufbau

- Realität in Java
 - z. B. `int[][] a = new int[2][3];`




- Eindimensionale Arrays enthalten Elemente vom Basistyp
- Mehrdimensionale Arrays enthalten weitere Arrays

Beispiel – Matrix initialisieren und ausgeben

```
int[][] a = new int[3][4];
for (int i = 0; i < a.length; i++) {
    for (int j = 0; j < a[0].length; j++) {
        a[i][j] = (int) random(10);
    }
}

for (int i = 0; i < a.length; i++) {
    for (int j = 0; j < a[0].length; j++) {
        print(a[i][j] + " ");
    }
    println();
}
```

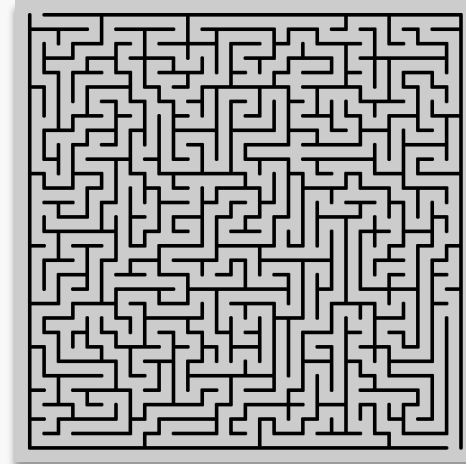
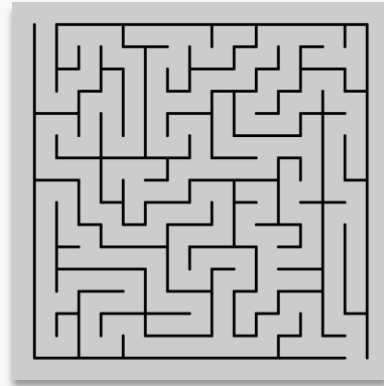
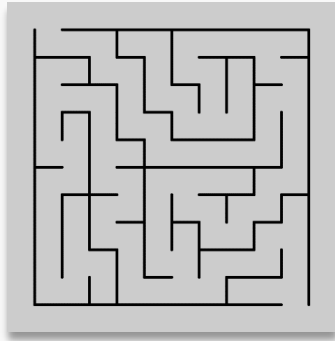


1	8	8	9
4	2	0	8
1	7	7	2

Beispiele für größere Programme

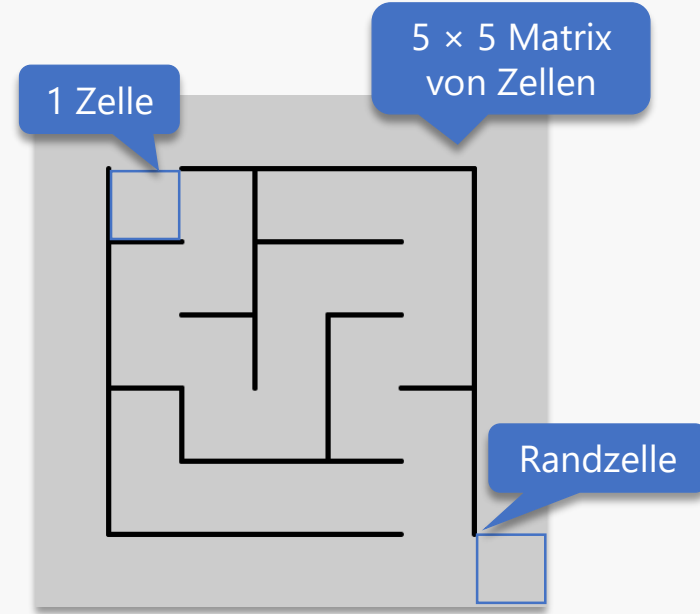
Beispiel – Labyrinth generieren

- Wie generiert man solche Labyrinth mit einem Programm?



Generierung – Ausgangslage

- Ausgangslage
 - Matrix von $n \times n$ Zellen
 - Insgesamt mehr (für Rand)
 - Jede Zelle hat zunächst vier Wände
 - Norden, Osten, Süden, Westen
 - Innere Zellen werden auf false gesetzt
 - Randzellen werden auf true gesetzt



Generierung – Algorithmus

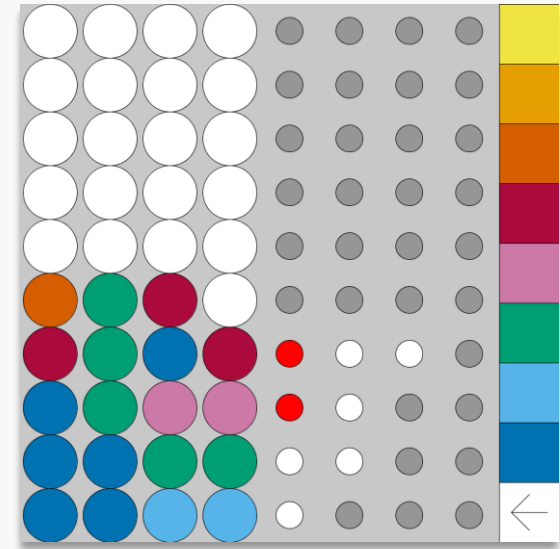
- Starte links oben
- Wiederhole, bis alle Zellen besucht wurden
 - Die aktuelle Zelle wird auf true gesetzt
 - Solange nicht alle Nachbarzellen der aktuellen Zelle true sind
 - Entferne eine Mauer zu einer zufälligen Nachbarzelle, die noch nicht besucht wurde
 - Diese Nachbarzelle wird die aktuelle Zelle
- Entferne am Ende zwei Mauern (z. B. links oben, rechts unten)

Praktische Überlegung

- Daten
 - 2D-Boolean-Arrays für die Himmelsrichtungen
 - 2D-Boolean-Array für die Markierung
 - Alle Arrays mit $n+2 \times n+2$ Zellen
 - Ränder!
- Details der Implementierung
 - In TUWEL

Weitere Beispiele

- www.openprocessing.org
- Wie könnte ein eigenes einfaches Spiel aussehen?
- Einfache Version von Mastermind
 - Spielfeld zeichnen
 - Programm „zieht“ zufällig Farben (Zahlen)
 - Farben setzen (durch anklicken von Flächen)
 - Programm generiert Feedback
 - Rot = Farbe richtig und Position richtig
 - Weiß = Farbe richtig, aber falsche Position
 - 10 Rateversuche möglich



Ausblick

Was wurde bisher besprochen?

- Beispiele für Funktionen in Processing
- Variablen
- Operatoren
- Verzweigungen
- Schleifen
- Eigene Funktionen schreiben
- Rekursion
- Arrays

Beispiele für weitere Aspekte in Processing

- Weitere Schleifen (do-while)
- Weitere Verzweigungen (switch)
- Viele weitere Funktionen für grafische Ausgaben (2D, 3D)
- Aufteilung von Programmcode in Klassen
- Vorgefertigte Klassen (für Bilder, Videos, ...)

Processing und Java

- Ähnlichkeiten zwischen Processing und Java (Beispiele)
 - Deklarationen und Datentypen
 - Verzweigungen
 - Schleifen
- Änderungen in Java (Beispiele)
 - Keine einfachen Sketches mehr
 - Mehr Schreibarbeit für lauffähiges Programm
 - Viele Programme erzeugen als Output keine Grafik 😊
 - Nicht mehr einfache Funktionen
 - Aufrufe werden komplexer

Mehr dazu in der VU Einführung in die Programmierung 1

- Ira Greenberg, Dianna Xu, Deepak Kumar: **Processing: Creative Coding and Generative Art in Processing 2**, 2. Auflage, friendsofED, 2013
- Casey Reas, Ben Fry: **Processing: A Programming Handbook for Visual Designers and Artists**, 2. Auflage, MIT Press, 2014
- Casey Reas, Ben Fry: **Getting Started with Processing**, 2. Auflage, O'Reilly & Associates, 2015