

Programmieren – Teil 2

Propädeutikum (Prolog) 2025
Jürgen Kogler und Martin Riener
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

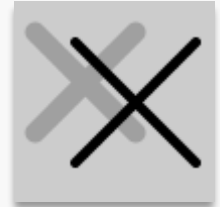
```
size(100,100);  
background(200);  
stroke(160);  
strokeWeight(10);  
line(10, 15, 60, 65);  
line(60, 15, 10, 65);
```



- Zwei Kreuze zeichnen

- Duplizierter Code mit ähnlichen Werten!

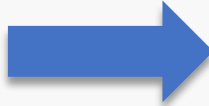
```
size(100, 100);  
background(200);  
stroke(160);  
strokeWeight(10);  
line(10, 15, 60, 65);  
line(60, 15, 10, 65);  
stroke(0);  
strokeWeight(5);  
line(30, 20, 90, 80);  
line(90, 20, 30, 80);
```



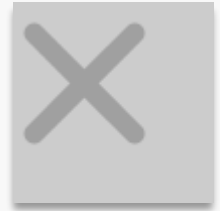
Beispiel mit Funktion (erster Schritt)

- 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

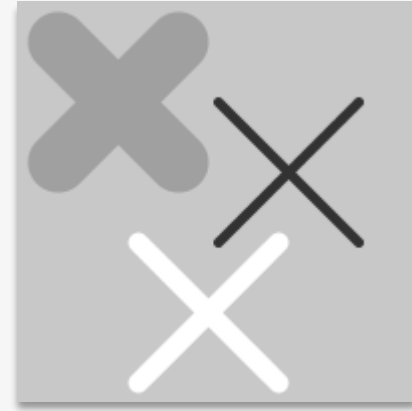
Parameter (wenn mehrere, dann durch Beistrich getrennt)
Form für jeden einzelnen Parameter: Datentyp Name

```
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 $n = 3$)

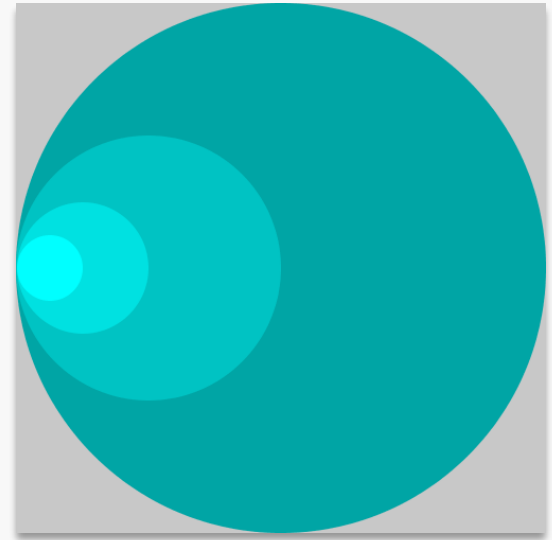
```
void setup() {  
    println(sumIterative(10));  
    println(sumRecursive(10));  
}  
  
int sumIterative(int num) {  
    int i, sum = 0;  
    for (i = 1; i <= num; i++) {  
        sum += i;  
    }  
    return sum;  
}  
  
int sumRecursive(int num) {  
    if (num > 0) {  
        return num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

Rekursion für sumRecursive(3)

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

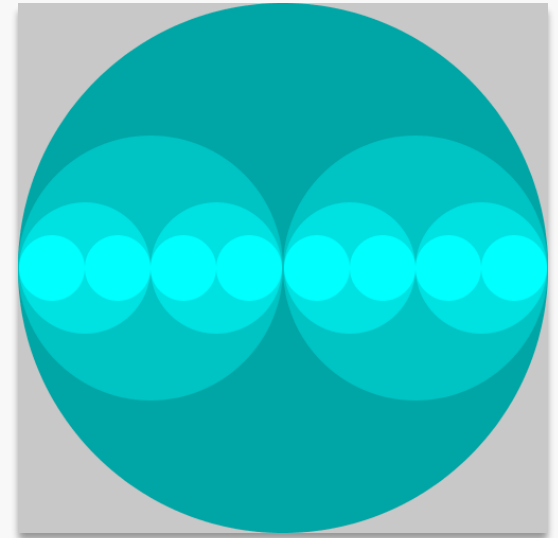
Ein visuelles Beispiel ...

```
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);  
  }  
}
```

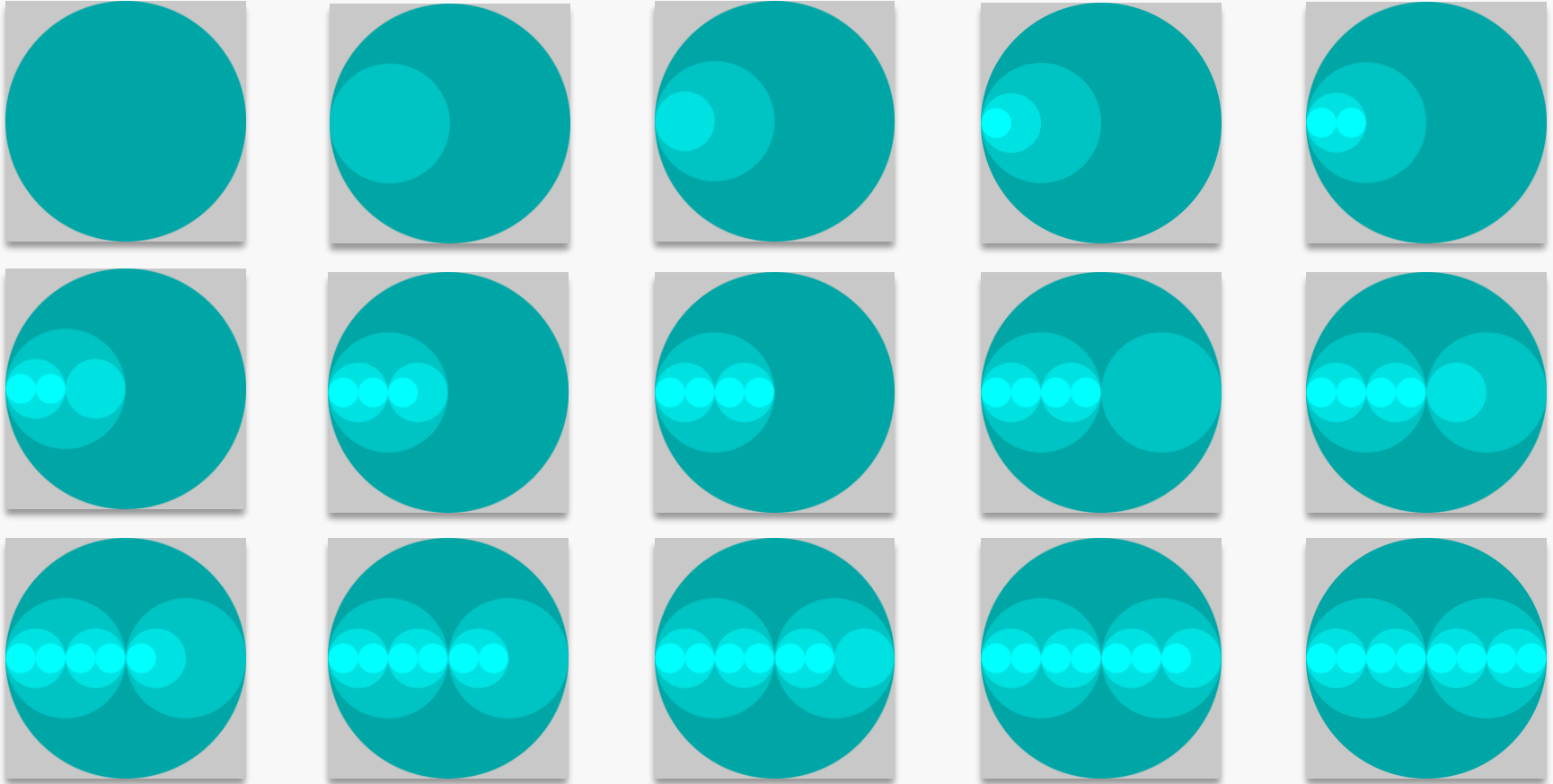


... und was wird jetzt gezeichnet?

```
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; 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);  
}
```

Parametertyp,
Rückgabetyp:
float[]

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

Kopie eines
Arrays anlegen

```
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;  
}
```

| | |
|-----|------|
| [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

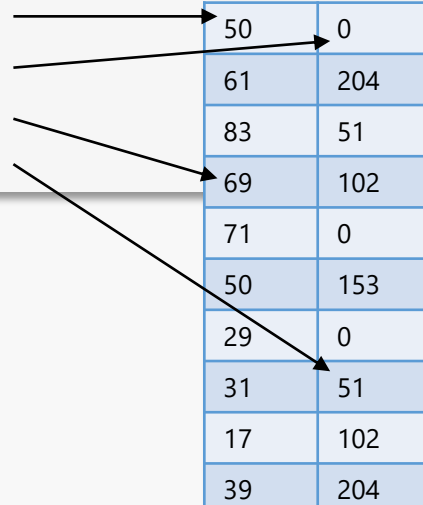
```
int[][] x = { {50, 0}, {61, 204}, {83, 51}, {69, 102}, {71, 0},  
              {50, 153}, {29, 0}, {31, 51}, {17, 102}, {39, 204} };
```

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

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

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

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

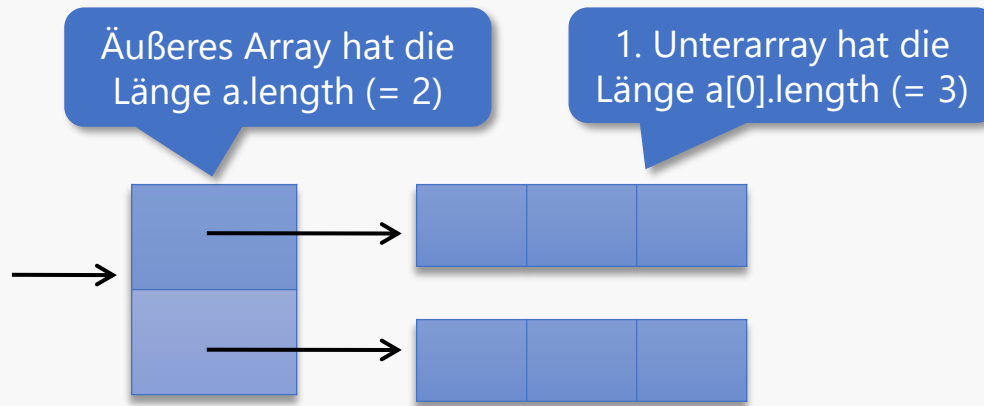


| | |
|----|-----|
| 50 | 0 |
| 61 | 204 |
| 83 | 51 |
| 69 | 102 |
| 71 | 0 |
| 50 | 153 |
| 29 | 0 |
| 31 | 51 |
| 17 | 102 |
| 39 | 204 |

50
0
69
51

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();
}
```

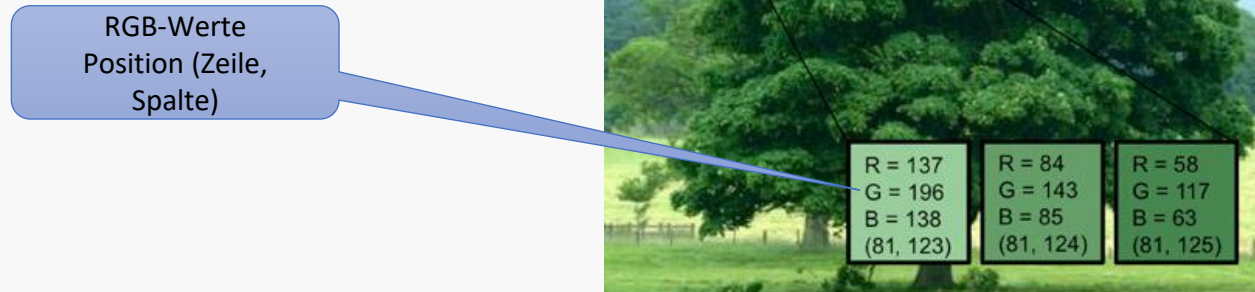


| | | | |
|---|---|---|---|
| 1 | 8 | 8 | 9 |
| 4 | 2 | 0 | 8 |
| 1 | 7 | 7 | 2 |

Beispiele für die Verwendung von Arrays → Bildverarbeitung

Digitale Bilder

- Rechteck von Bildpunkten (Pixel)
- Farbe eines Pixels
 - Typischerweise in mehreren Bytes codiert
 - Beispiel RGB (3 Bytes)
 - Jeweils 8 Bit für Rot-, Grün- und Blauanteil



Bilder in Processing

- Datentyp (PImage) für Bilder (PImage-Klasse)

```
PImage img;
```

- Einige Variablen wie z.B. width und height
 - Einige Funktionen wie z.B. resize (Anpassen an Fenstergröße)
- Laden eines Bildes

```
PImage img = loadImage("tree.jpg");
```

- Bild muss im Verzeichnis des Sketches liegen
- Allgemeine Funktionen
 - image – zur Darstellung eines Bildes

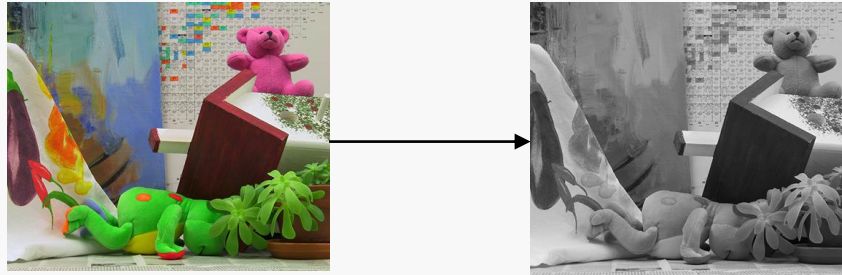
Beispiel – „Verpixeln“

```
size(640, 320);
PImage img = loadImage("tree.jpg");
img.resize(width, height);
int resolution = 50;
int xInc = width/resolution;
int yInc = height/resolution;
for (int y = 0; y < img.height; y += yInc) {
    for (int x = 0; x < img.width; x += xInc) {
        fill(img.get(x, y));
        rect(x, y, xInc, yInc);
    }
}
```

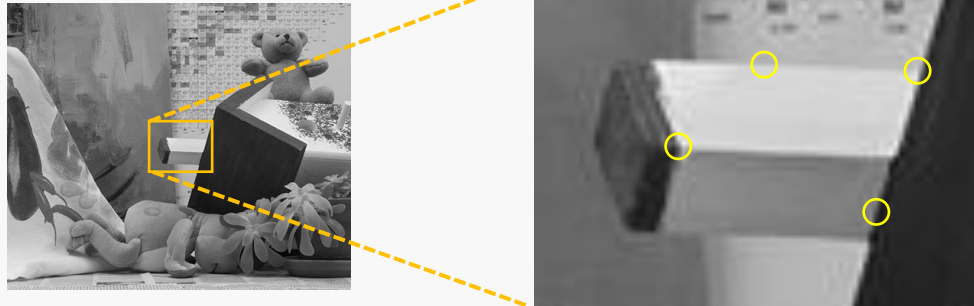


Beispiel – „Kantendetektion“ (1)

- Für die Detektion von Kanten in einem Bild wird das entsprechende Grauwertbild verwendet

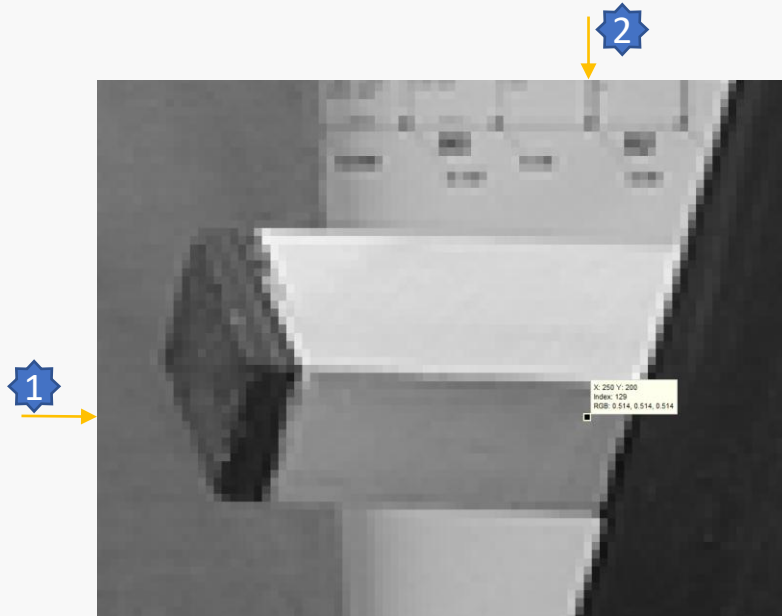


- Kanten sind dort zu finden wo der Gradient zwischen Nachbarpixel am größten ist

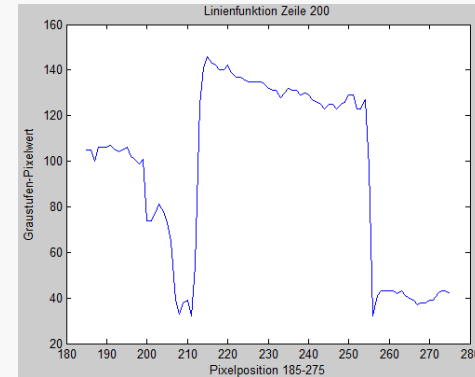


Beispiel – „Kantendetektion“ (2)

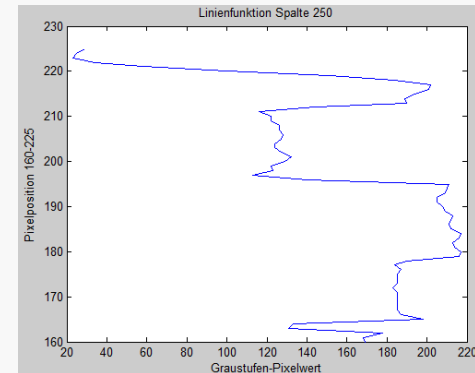
- Jede Zeile bzw. Spalte des Bildes kann als entsprechende Linienfunktion dargestellt werden



1



2



Beispiel – „Kantendetektion“ (3)

- Filter-Kernel
 - z.B. 3x3 Filter für die Detektion vertikaler Kanten im Bild

| | | |
|----|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |



Beispiel – „Kantendetektion“ (4)

- Ergebnis Kantenbild

| | | |
|----|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |



Beispiel – „Kantendetektion“ (5)

```
float[][] kernel1 = {{ -1, 0, 1},
                     { -1, 0, 1},
                     { -1, 0, 1}};

size(900, 750);
PImage img = loadImage("teddy.png");
PImage edgeImg = createImage(img.width, img.height, RGB);
image(img, 0, 0);
img.filter(GRAY);
image(img, width/2, 0);
img.filter(BLUR, 1);
image(img, 0, height/2);

for (int y = 1; y < img.height-1; y++) {
  for (int x = 1; x < img.width-1; x++) {
    float sum = 0;
    for (int ky = -1; ky <= 1; ky++) {
      for (int kx = -1; kx <= 1; kx++) {
        float val = red(img.get(x+kx,y+ky));
        sum += kernel1[ky+1][kx+1] * val;
      }
    }
    edgeImg.set(x,y,color(sum, sum, sum));
  }
}
image(edgeImg, width/2, height/2);
```

Beispiel – „Kantendetektion“ (6)

- Ergebnis Kantenbild

| | | |
|----|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |



| | | |
|----|----|----|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | 1 |



Beispiel – „Stereoskopie/Stereo Matching“ (1)

- Zwei zwei-dimensionale Bilder dienen als Grundlage für die drei-dimensionale Rekonstruktion

Linke Aufnahme

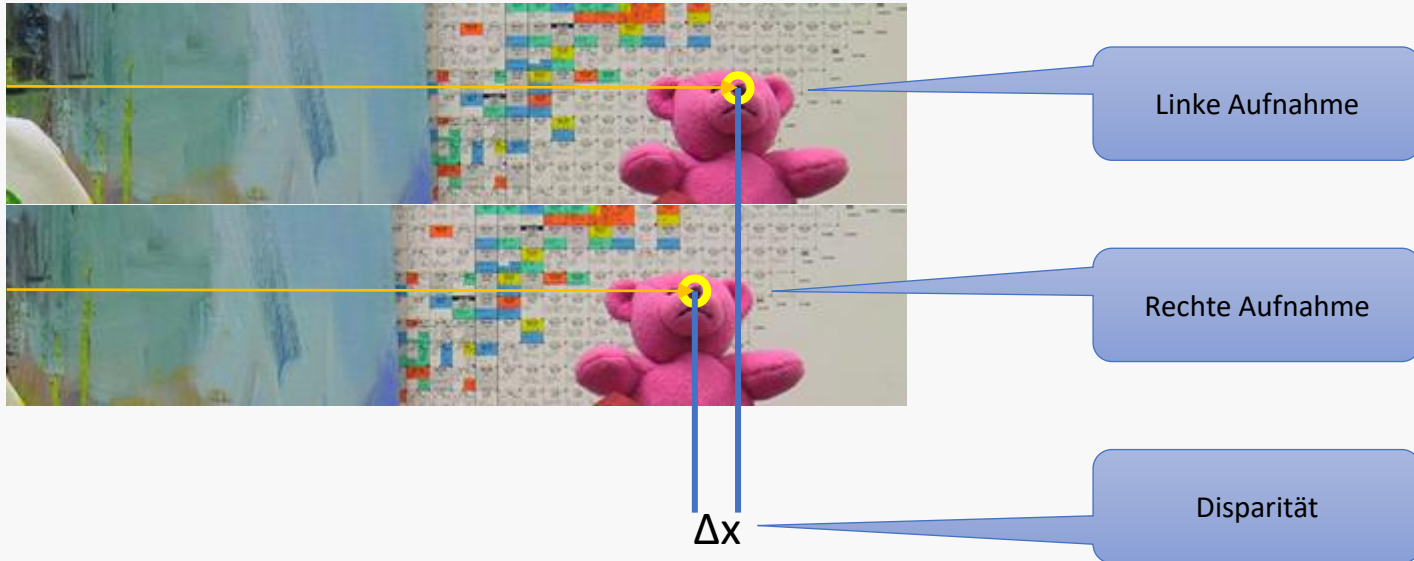


Rechte Aufnahme



Beispiel – „Stereoskopie/Stereo Matching“ (2)

- Jedes Pixelpaar finden. Zu jedem Pixel in der linken Aufnahme das korrespondierende Pixel in der rechten Abbildung suchen



Beispiel – „Stereoskopie/Stereo Matching“ (3)

```
PImage img_left;
PImage img_right;
int ws = 11; // window size
int dr = 50; // disparity range
```

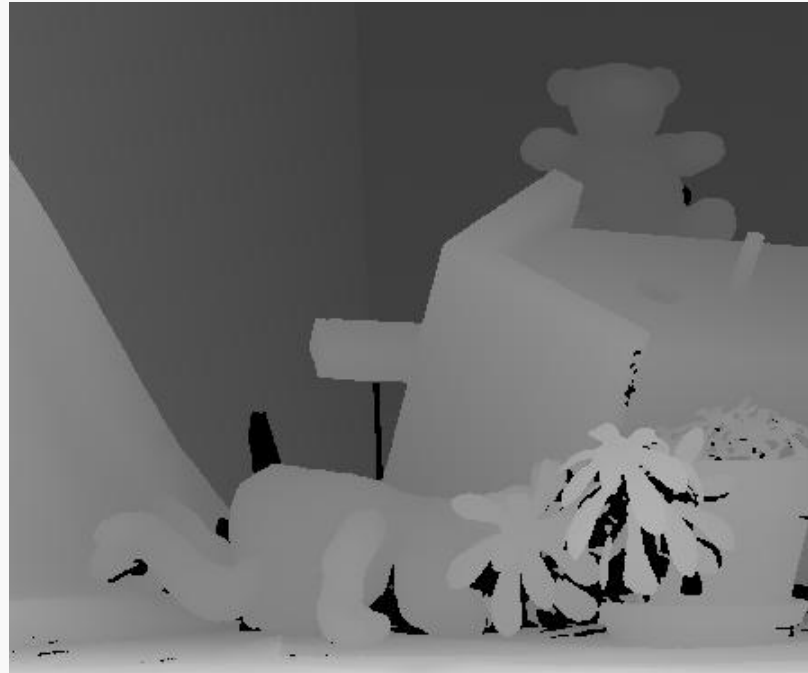
```
void setup() {
    size(900, 750);
    img_left = loadImage("teddy_l.png");
    img_right = loadImage("teddy_r.png");
    noLoop();
}
```

```
void draw() {
    image(img_left, 0, 0);
    image(img_right, width/2, 0);
    img_left.loadPixels();
    img_right.loadPixels();
    PImage disp_img = createImage(img_left.width, img_left.height, RGB);
    float sum_diff = 0.0;
    float sum_diff_min = 100000.0;
    int disp = 0;
    for (int y = ws/2; y < img_left.height-ws/2; y++) {
        for (int x = dr+ws/2; x < img_left.width-ws/2; x++) {
            sum_diff_min = 100000.0;
            for (int dr_idx = 1; dr_idx <= dr; dr_idx++){
                sum_diff = 0.0;
                for (int wy = y-ws/2; wy <= y+ws/2; wy++){
                    for (int wx = x-ws/2; wx <= x+ws/2; wx++){
                        sum_diff += abs(red(img_left.pixels[wy*img_left.width+wx])-
                                         red(img_right.pixels[wy*img_right.width+wx-dr_idx]));
                    }
                }
                if(sum_diff < sum_diff_min){
                    sum_diff_min = sum_diff;
                    disp = dr_idx;
                }
            }
        }
        int color_val = (255/dr)*disp;
        disp_img.pixels[y*img_left.width+x] = color(color_val, color_val, color_val);
    }
    image(disp_img, 0, height/2);
}
```


Beispiel – „Stereoskopie/Stereo Matching“ (4)

- Distanzen der gefundenen Pixel werden als Grauwert dargestellt und bilden somit eine Tiefenkarte

Linke Aufnahme



Rechte Aufnahme



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