

PROCESSING

SCHUBLADEN UND ZEICHEN

Created by Michael Kirsch & Beat Rossmly

INHALT

1. Rückblick

1. Processing Basics
2. float
3. for
4. else
5. Mouse Input

2. Theorie

1. Wir brauchen mehr Ordnung!
2. Array
3. Nicht mathematische Datentypen?
4. Characters & Strings
5. Handlungen zusammenfassen!
6. Funktionen
7. Zauberwort: void
8. Andere Rückgabetypen
9. Processing Basics

3. Anwendung

1. Array
2. String
3. Funktionen

4. Verknüpfung

1. KeyBoard Visuals
2. Was ist eine Animation?
3. Reagieren auf den Input
4. Trennen von Input und Animation
5. Strukturieren durch Funktionen

5. Ausblick

1. Nächste Sitzung
2. Übung

RÜCKBLICK

PROCESSING BASICS

Einzeiliger Kommentar

```
// das ist ein einzeiliger Kommentar
```

Mehrzeiliger Kommentar

```
/* das ist  
ein mehrzeiliger  
Kommentar */
```

Inkrementieren

```
x = x+1;  
x += 1;  
x++;
```

Dekrementieren

```
x = x-1;  
x -= 1;  
x--;
```

Fensterbreite

```
w = width;
```

Fensterhöhe

```
h = height;
```

FLOAT

- **float** zeichnet Variablen aus, die Gleitkommazahlen enthalten.
- Achtung: diese markieren - anders als im deutschsprachigen Raum üblich - ihre Nachkommastellen mit einem Punkt!
- Vorsicht beim Mischen von Datentypen ist geboten!

```
int a = 5;
int b = 2;

float z = a/b;
println(z); // -> 2.0 !!!

float c = 5;
float d = 2;

z = c/d;
println(z); // -> 2.5

int e = 5;
float f = 2;

z = e/f;
println(z); // -> 2.5 !!!

float g = 5;
int h = 2;

z = g/h;
println(z); // -> 2.5 !!!
```

FOR

- **for** leitet eine Schleife ein.
- In den () wird die Wiederholung des Rumpfes beschrieben. Angefangen bei **n** bis (hier) **5** im angegebenen Intervall (**++**).
- **{ }** enthalten die auszuführenden Befehle.

```
void setup () {  
    size(400,400);  
  
    for (int n=0; n<5; n++) {  
        float x = random(width);  
        float y = random(height);  
        ellipse(x,y,5,5);  
    }  
}
```

ELSE

if bedingt das Ausführen einzelner oder mehrerer Befehle abhängig von einer Aussage.

Die Befehle im Rumpf von **else** treten nur ein, ist die Aussage von **if** falsch.

```
if (x == 100) {  
    println("x ist genau 100");  
}
```

```
else {  
    println("x ist nicht 100");  
}
```

MOUSE INPUT

mouseX und **mouseY**
enthalten die aktuelle
Position des Cursors.

```
// Kreis an Cursor Position  
ellipse(mouseX, mouseY, 100, 100);
```

mousePressed gibt
den Status
"gedrückt/ungedrückt"
der linken Maustaste
wieder.

```
if (mousePressed) {  
    println("Linke Maustaste gedrückt!");  
}  
else {  
    println("Linke Maustaste nicht gedrückt!");  
}
```


THEORIE

WIR BRAUCHEN MEHR ORDNUNG!

- Je komplexer unsere Sketche werden, desto mehr Variablen benötigen wir, desto unübersichtlicher wird das ganze.
- Wenn es so etwas wie Schubladen gäbe, in denen man alles zusammengehörige verstauen könnte...



<http://nos.twinsnd.co/image/117523960441>

WIR BRAUCHEN MEHR ORDNUNG!

- Stellen wir uns vor, wir würden nun 6 sich bewegende Kreise zeichnen wollen...
- Wir bräuchten 6 Variablen für X-Koordinaten, 6 für Y-Koordinaten, ...

```
int x1, x2, x3, x4, x5, x6;
int y1, y2, y3, y4, y5, y6;

void setup () {
    size(400,400);

    x1 = 0;
    x2 = 0;
    x3 = 0;
    x4 = 0;
    x5 = 0;
    x6 = 0;

    y1 = 0;
    y2 = 0;
    y3 = 0;
    y4 = 0;
    y5 = 0;
    y6 = 0;
}

void draw () {...}
```

ARRAY

- Ein Array wird mit [] markiert.
- Vor den [] steht der Datentyp den das Array *ausschließlich* enthält.
- Nach den [] steht der Name mit dem wir das Array deklarieren.

```
int[] x;
int[] y;

void setup () {
    size(400,400);

    x = new int[6];
    y = new int[6];

    x[0] = 0;
    x[1] = 0;
    x[2] = 0;
    x[3] = 0;
    x[4] = 0;
    x[5] = 0;

    y[0] = 0;
    y[1] = 0;
    y[2] = 0;
    y[3] = 0;
    y[4] = 0;
    y[5] = 0;

    ...
}
```

ARRAY

- Um das *neue* Array zu initialisieren, verwenden wir den Operator **new**, wiederum den Datentyp und anschließend in den **[]** die Länge unseres Speichers.
- In diesem Fall können wir also genau die 6 X-Koordinaten im Array **x** ablegen.

```
int[] x;
int[] y;

void setup () {
    size(400,400);

    x = new int[6];
    y = new int[6];

    x[0] = 0;
    x[1] = 0;
    x[2] = 0;
    x[3] = 0;
    x[4] = 0;
    x[5] = 0;

    y[0] = 0;
    y[1] = 0;
    y[2] = 0;
    y[3] = 0;
    y[4] = 0;
    y[5] = 0;

    ...
}
```

ARRAY

- Um nun auf die einzelnen im Array enthaltenen Werte zuzugreifen, verwenden wir wiederum [] und darin die Stelle von der wir lesen oder in die wir schreiben wollen.
- Wichtig: Die erste Stelle des Speichers hat stets die Nummer **0**!
- Aber das ist ja jetzt noch länger?

```
int[] x;
int[] y;

void setup () {
    size(400,400);

    x = new int[6];
    y = new int[6];

    x[0] = 0;
    x[1] = 0;
    x[2] = 0;
    x[3] = 0;
    x[4] = 0;
    x[5] = 0;

    y[0] = 0;
    y[1] = 0;
    y[2] = 0;
    y[3] = 0;
    y[4] = 0;
    y[5] = 0;

    ...
}
```

ARRAY

- Verknüpft mit unserem restlichen Wissen können wir nun einen Vorteil aus dieser neuen Struktur gewinnen.
- **for** hilft uns nun den wiederkehrenden Zugriff auf Teile des Arrays (hier die Initialisierung) zu verkürzen.

```
int[] x;
int[] y;

void setup () {
    size(400,400);

    x = new int[6];
    y = new int[6];

    for (int n=0; n<6; n++) {
        x[n] = 0;
        y[n] = 0;
    }
}
```

ARRAY

Initialisierung ohne konkrete Werte

```
int[] a = new int[3];
```

Initialisierung mit konkreten Werten

```
int[] a = new int[] {1,2,3};
```

Tipp: formatierte Ausgabe von Arrays

```
int[] a = new int[] {1,2,3};  
printArray(a);
```


NICHT MATHEMATISCHE DATENTYPEN?

- Natürlich gibt es neben `int` und `float` auch andere nicht mathematische Datentypen.
- Wir haben diese schon bei der Ausgabe von Text in der Konsole verwendet!



CHARACTERS & STRINGS

- Jedes Zeichen unseres Alphabets, unsere Ziffern und Sonderzeichen werden im Computer als Character gespeichert. Intern sind diese als Zahlen repräsentiert. Diese Codierung nennt sich Unicode.
- Variablen des Typen Character deklarieren wir mit **char** und initialisieren in ' '.

```
char c = 'a';  
println(c);  
// -> a  
  
int i = 'a';  
println(i);  
// -> 97
```

CHARACTERS & STRINGS

- Wörter und Sätze sind Reihen von Zeichen. Strings enthalten alle Zeichen eines Worts oder eines Satzes.
- Wir deklarieren mit **String** und initialisieren in `" "`.
- Java/Processing unterscheidet zwischen Groß- und Kleinschreibung. Also **String** immer mit großem **S**.

```
String h = "Hello";  
println(h);  
// -> Hello  
  
String w = "World!";  
println(w);  
// -> World!  
  
String hw = h+w;  
println(hw);  
// -> HelloWorld!  
  
hw = h+" "+w;  
println(hw);  
// -> Hello World!  
  
hw = "Hello World!";  
println(hw);  
// -> Hello World!
```

HANDLUNGEN ZUSAMMENFASSEN!

- Angenommen wir wollen nun nicht immer identische Höhe und Breite in **ellipse** eingeben um einen Kreis zu erhalten.
- Könnten wir nicht einen eigenen Befehl bestimmen der einfach einen Kreis zeichnet?

```
void setup () {...}  
  
void draw () {  
  stroke(255,0,0);  
  fill(255,0,255);  
  ellipse(33,44,55,55);  
  
  stroke(0,255,0);  
  fill(0,255,255);  
  ellipse(22,77,33,33);  
  
  stroke(0,0,255);  
  fill(255,255,0);  
  ellipse(55,11,22,22);  
}
```

HANDLUNGEN ZUSAMMENFASSEN!

- Können wir einen Befehl **kreis(x,y,d)** bestimmen?
- Vielleicht könnte man diesem neben Koordinaten und Durchmesser auch Umriss und Füllfarbe übergeben?

```
void setup () {...}

void draw () {
  stroke(255,0,0);
  fill(255,0,255);
  kreis(33,33,55);

  stroke(0,255,0);
  fill(0,255,255);
  kreis(22,77,33);

  stroke(0,0,255);
  fill(255,255,0);
  kreis(55,11,22);
}
```

HANDLUNGEN ZUSAMMENFASSEN!

- Das würde die Möglichkeit eröffnen, sich wiederholende Strukturen verkürzt schreiben zu können.
- Aber wie können wir das erreichen?

```
void setup () {...}  
  
void draw () {  
  kreis(33,33,55,255,0,0,255,0,255);  
  
  kreis(22,77,33,0,255,0,0,255,255);  
  
  kreis(55,11,22,0,0,255,255,255,0);  
}
```

FUNKTIONEN

- Außerhalb von **setup** und **draw** können wir sogenannte Funktionen definieren.
- Mit einem Namen **kreis** und in () festgelegten Variablen.
- Diese Variablen können nun im Rumpf verwendet werden.

```
void setup () {...}

void draw () {...}

void kreis (int x, int y, int d, int sR, int sG, int sB, int fR, int fG, int fB) {
  stroke(sR, sG, sB);
  fill(fR, fG, fB);
  ellipse(x,y,d,d);
}
```

ZAUBERWORT: VOID

- Was bedeutet jetzt eigentlich immer dieses **void**?
- Funktionen können an unterschiedlichen Orten aufgerufen werden.
- Auch z.B. nach Zuweisungsoperatoren.

```
maleEinenKreis(25,25,25);  
  
int x = dasDoppelteVon(3);
```


ZAUBERWORT: VOID

- Also müssen Funktionen auch in der Lage sein Werte auszugeben, die dann weiterverarbeitet werden können!
- **maleEinenKreis()** soll keinen Wert ausgeben. Also schreiben wir **void** um den Computer mitzuteilen, dass diese Funktion nichts zurückgibt.

```
maleEinenKreis(25,25,25);  
  
int x = dasDoppelteVon(3);  
  
void maleEinenKreis(int x, int y, int r)  
{  
    ellipse(x,y,2*r);  
}
```

ANDERE RÜCKGABETYPEN

- Wie sagen wir nun dem Computer, dass er bei **dasDoppelteVon()** einen Wert zurückgeben soll?
- Vor dem Namen der Funktion steht immer der returntype (**int**, **char**, ...)!
- Der letzte Befehl einer Funktion ist immer **return ...;** (außer bei **void**).

```
maleEinenKreis(25,25,25);

int x = dasDoppelteVon(3);

void maleEinenKreis(int x, int y, int r) {
    ellipse(x,y,2*r);
}

int dasDoppelteVon (int n) {
    return n*2;
}
```

PROCESSING BASICS

Farben sind Datentypen

```
color a = color(255,0,0); // rgb  
color b = color(0); // schwarz  
color c = color(255); // weiß  
color d = color(100); // grauton
```

Füllfarbe

```
fill(a);
```

Umrissfarbe

```
stroke(b);
```

ohne Füllfarbe

```
noFill();
```

ohne Umrissfarbe

```
noStroke();
```

Stärke des Umrisses

```
strokeWeight(5);
```

ANWENDUNG

ARRAY

```
// Arrays zum Speichern von Maus Koordinaten

void setup () {
  size(800,800);
  // initialisiere Arrays
}

void draw () {
  // vergangene Mauspositionen im Array verschieben

  // neue Position speichern

  // Linien zwischen Positionen zeichnen
}
```

STRING

```
// Arrays: Subjekt Prädikat Objekt

void setup () {
  size(800,800);
  // initialisiere Arrays
}

void draw () {
  // gebe zufällig erzeugte Sätze in der Konsole aus
}
```

FUNKTIONEN

```
void setup () {  
    size(800,800);  
}  
  
void draw () {  
    // rufe Zeichenfunktion auf  
}  
  
// deklariere Zeichenfunktion  
  
// deklariere Funktion "erzeuge Y-Koordinate" abhängig von "X-Position"  
// deklariere Funktion "erzeuge Größe" abhängig von "X-Position"
```

VERKNÜPFUNG

KEYBOARD VISUALS

- Wir entwickeln über die nächsten Sitzungen ein Programm, dass auf Tastendruck kleine Animationen abspielt.
- Zunächst entwerfen wir das Konzept für eine Taste und erweitern dann nach und nach das Programm.

```
void setup () {  
    size(600,400);  
}  
  
void draw () {  
    background(0);  
    // reagieren auf Input  
    // zeichnen der Animation  
}
```

WAS IST EINE ANIMATION?

- Für uns besteht im wesentlichen eine Animation aus einer Abfolge von Bildern.
- Diese Abfolge ist abhängig von der Zeit, wie die Frames eines Videos.
- Der **animationCounter** repräsentiert das zu zeichnende Frame und zählt von 0 bis zu einer beliebig großen Zahl.

```
int animationCounter;

void setup () {
  size(600,400);
}

void draw () {
  background(0);
  // reagieren auf Input
  // zeichnen der Animation
}
```

REAGIEREN AUF DEN INPUT

- Zunächst dient uns die Maustaste als Input.
- Wenn diese gedrückt ist inkrementieren wir den counter.
- Ansonsten setzen wir auf 0 zurück.
- Wenn die Maustaste gedrückt ist, wird ebenfalls gezeichnet.

```
int animationCounter;

void setup () {
  size(600,400);
  animationCounter = 0;
}

void draw () {
  background(0);
  // reagieren auf Input
  // zeichnen der Animation
  if (mousePressed) {
    animationCounter++;
    int r = animationCounter;
    ellipse(300,200,r,r);
  }
  else {
    animationCounter = 0;
  }
}
```

TRENNEN VON INPUT UND ANIMATION

- Wir trennen die Logik der Input-Überprüfung von den Zeichenbefehlen.
- Wir erhalten eine Struktur, die leicht vervielfacht und auf mehrere Tasten/Inputs angewandt werden kann.

```
int animationCounter;

void setup () {
  size(600,400);
  animationCounter = 0;
}

void draw () {
  background(0);
  // reagieren auf Input
  if (mousePressed) {
    animationCounter++;
  }
  else {
    animationCounter = 0;
  }

  // zeichnen der Animation
  if (animationCounter > 0) {
    int r = animationCounter;
    ellipse(300,200,r,r);
  }
}
```

STRUKTURIEREN DURCH FUNKTIONEN

- Nun teilen wir die uns zuvor überlegte Struktur in die Funktionen **handleInput** und **drawAnimation** auf.
- Wir können auch auf Tastendrücke hören!
- Was ist der Nachteil von **keyPressed**?

```
int animationCounter;
void setup () {...}

void draw () {
  background(0);
  // reagieren auf Input
  handleInput();
  // zeichnen der Animation
  drawAnimation();
}

void handleInput () {
  if (keyPressed) {
    animationCounter++;
  }
  else {animationCounter = 0;}
}

void drawAnimation () {
  if (animationCounter > 0) {
    int r = animationCounter;
    ellipse(300,200,r,r);
  }
}
```

AUSBLICK

NÄCHSTE SITZUNG

- Klassen und Objekte

ÜBUNG

QUELLEN

- <http://nos.twnsnd.co/image/117523960441>
- <http://nos.twnsnd.co/image/124161898747>