

Programmieren!

Meine ersten Schritte als ProgrammiererIn!

Prolog 2016

Stefan Podlipnig, TU Wien

Ziele



- Kennenlernen einer einfachen Programmiersprache
- Verständnis für einfache Programmierkonzepte entwickeln

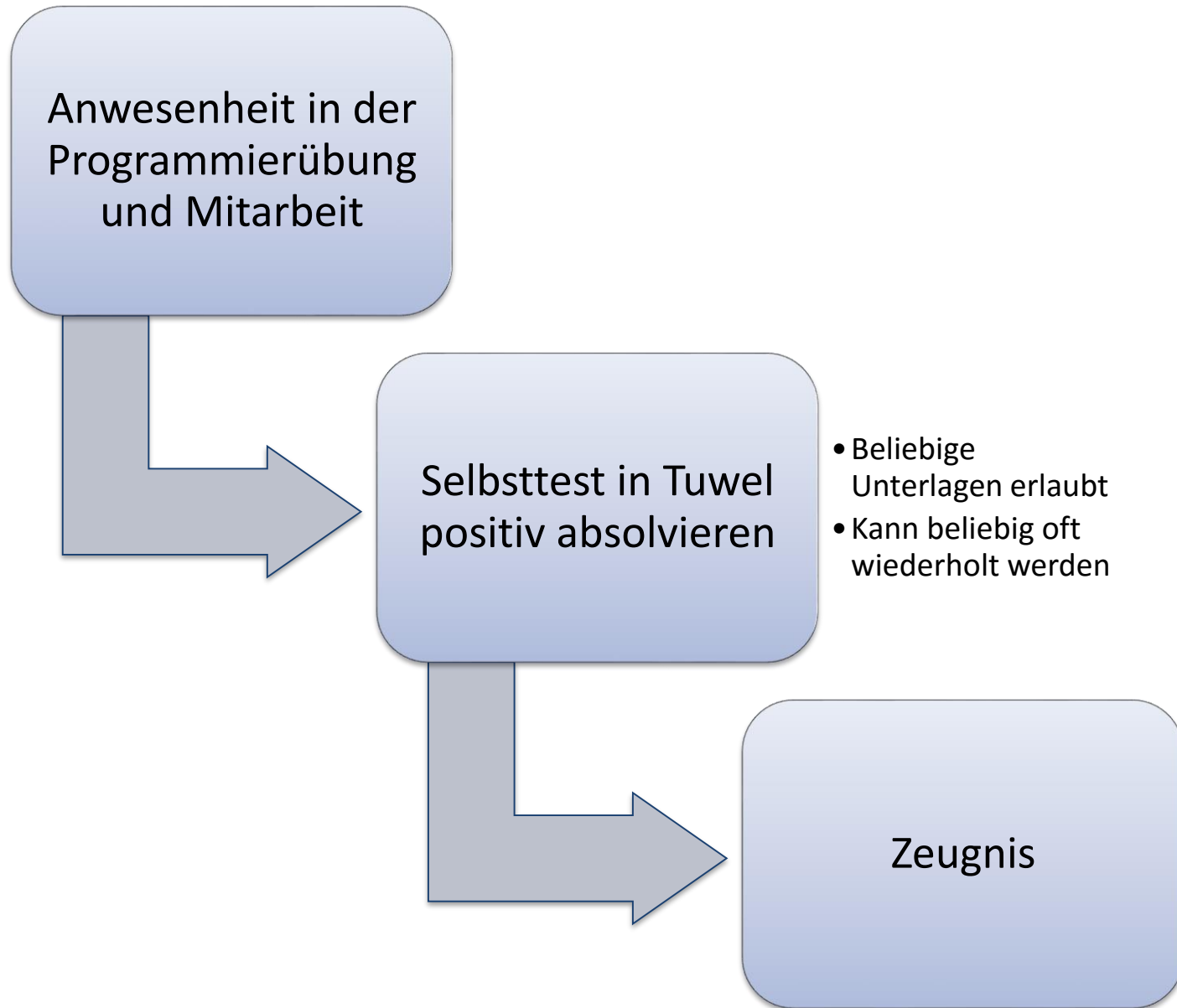
Diese Veranstaltung soll Ihnen einen **einfachen Einstieg** in die Programmierung ermöglichen

Organisation

- Vorlesungen
 - 7 Einheiten (Audi-Max)
- Übungen im Computerraum
 - HG EG 05 (Frogger Raum), Favoritenstrasse 11
 - 28.09. – 02.10.
 - Termine auf der TISS-Homepage des Prologs
 - Anmeldung zu **einem** Termin
 - TISS-Homepage (Gruppen-Anmeldung)
 - Extra vier Gruppen für „Fortgeschrittene“

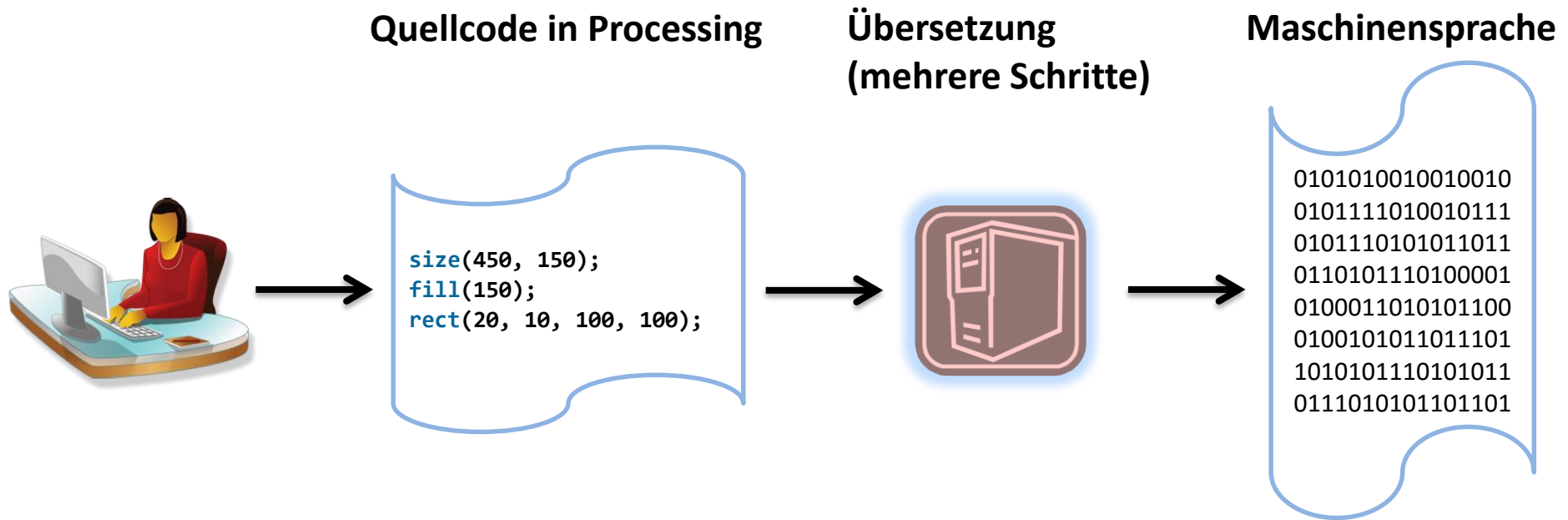


Zeugnis für gesamten Prolog-Kurs



PROCESSING – GRUNDLAGEN

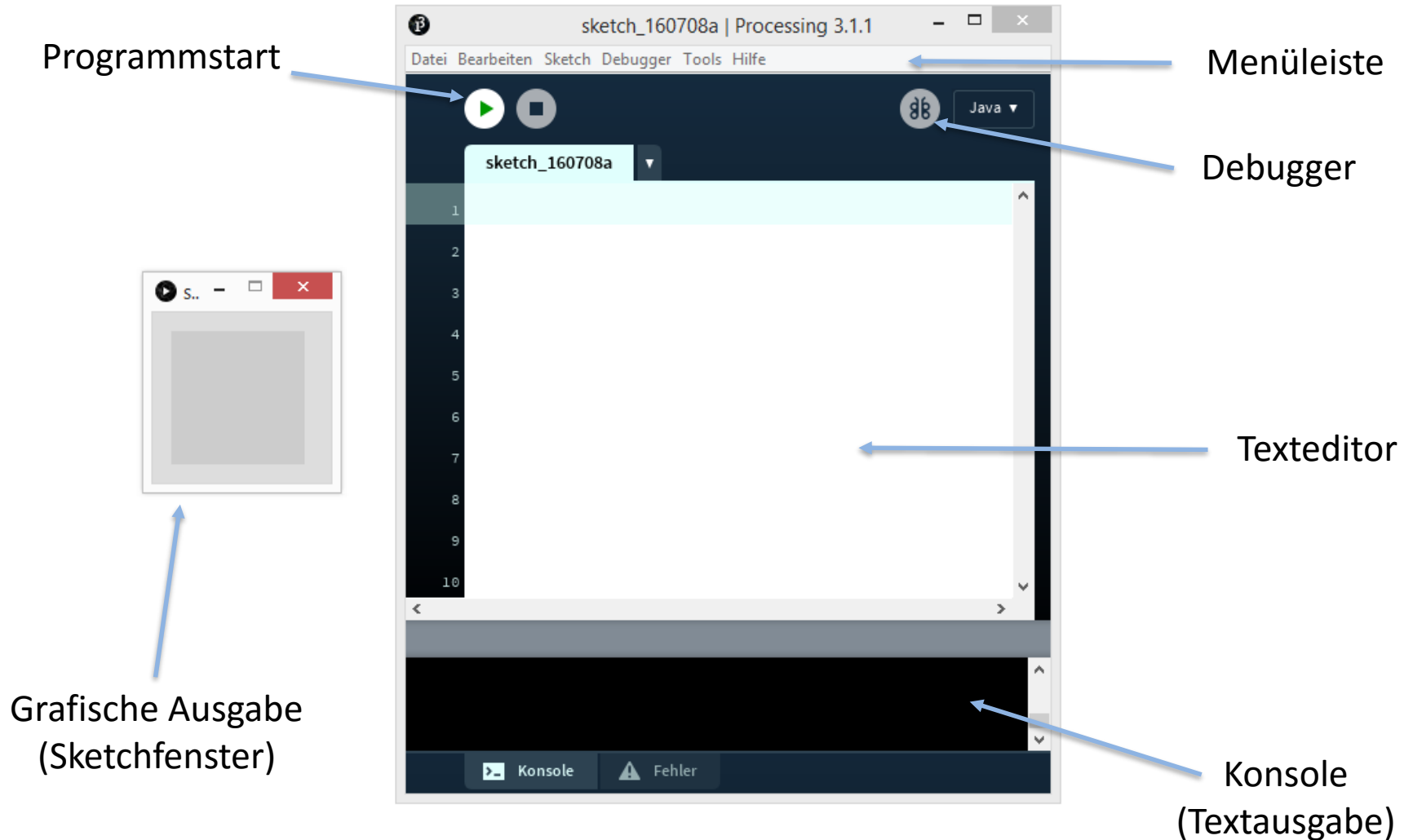
Programmierung (in Processing)



- Maschinensprache = Befehle, die ein Prozessor versteht
- Höhere Programmiersprache (z.B. Processing)
 - Für Menschen leichter
 - Muss bestimmten Regeln folgen (Unterstützung der Übersetzung)

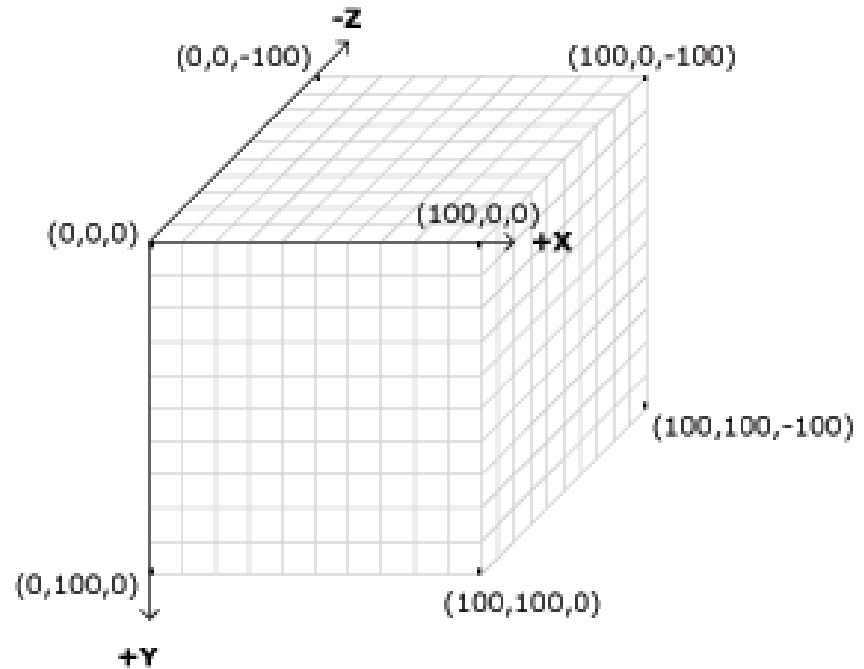
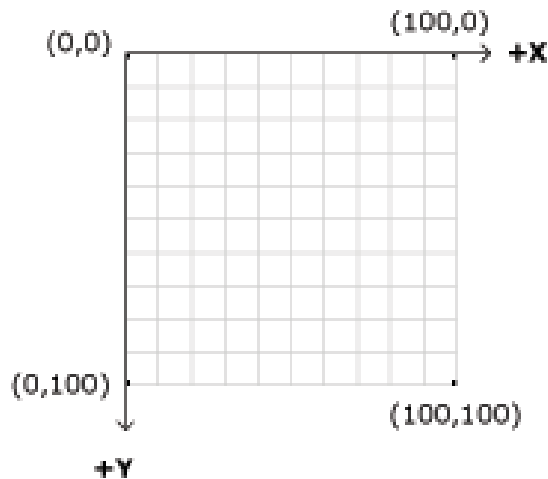
- Einfache Programmierumgebung
 - Visuelle Elemente
 - Interaktionen
- Keine eigene Programmiersprache
 - Stark vereinfachte Version der Programmiersprache Java
- Webseite
 - <http://processing.org/>

Entwicklungsumgebung



Sketches und Sketchfenster

- Skizzen (Sketches)
 - Für ein neues Programm
- Koordinatensystem im Sketchfenster
 - Positive Y-Werte gehen nach unten



Zeichnen mit Funktionen

- Funktion in Processing
 - Kleines „Programm“ für eine bestimmte Aufgabe
- Aufbau eines Aufrufs
 - Name der Funktion
 - Öffnende Klammer
 - Argumente (durch Beistriche getrennt)
 - Dienen zum Anpassen
 - Schließende Klammer
 - Strichpunkt
 - Schließt die gesamte Anweisung ab

```
size(300, 150);
```

Beispiele

- Sketchfenster mit der Größe 200×200 (Breite \times Höhe) zeichnen

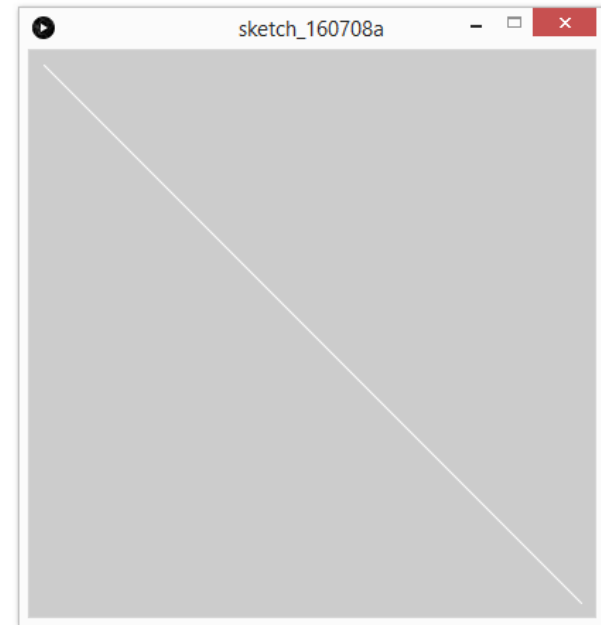
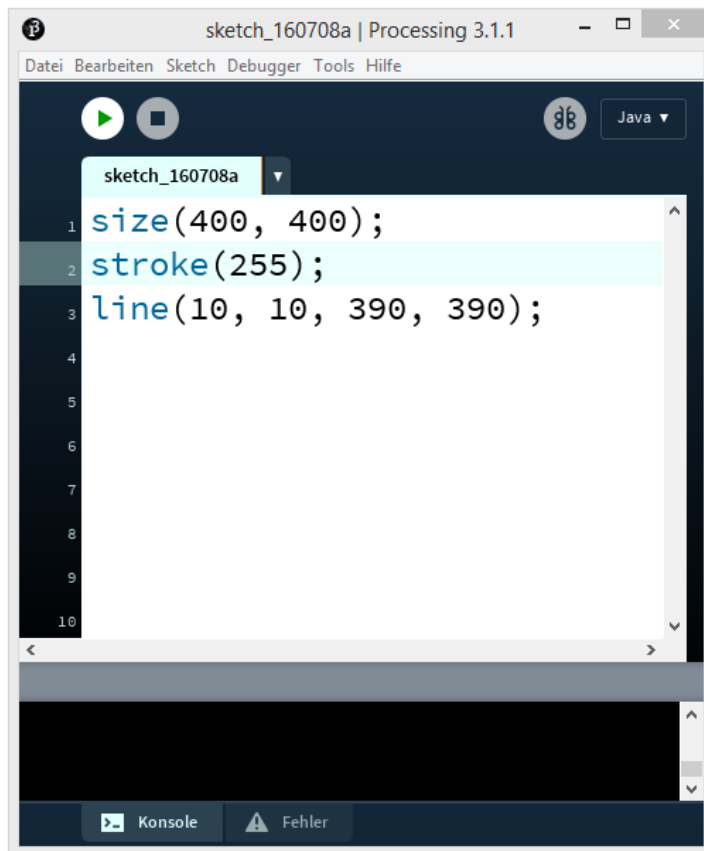
```
size(200, 200);
```

- Sketchfenster mit der Größe 200×200 und danach Punkt (Koordinaten $x=100$ und $y=50$) zeichnen

```
size(200, 200);  
point(100, 50);
```

Beispiel

- Beispiel
 - Sketchfenster mit der Größe 400 × 400
 - Zeichenfarbe für Linien auf weiß setzen
 - Linie zwischen Startpunkt (10, 10) und Endpunkt(390, 390) zeichnen



Informationen zu den Funktionen

- Auflistung von Funktionen in Processing
 - <http://www.processing.org/reference/>
- Beispiel stroke

Name stroke()

Examples



```
stroke(153);  
rect(30, 20, 55, 55);
```



```
stroke(204, 102, 0);  
rect(30, 20, 55, 55);
```

Description

Sets the color used to draw lines and borders around shapes. This color is either specified in terms of the RGB or HSB color depending on the current `colorMode()` (the default color space is RGB, with each value in the range from 0 to 255).

When using hexadecimal notation to specify a color, use "#" or "0x" before the values (e.g. #CCFFAA, 0xFFCCFFAA). The # syntax uses six digits to specify a color (the way colors are specified in HTML and CSS). When using the hexadecimal notation starting with "0x", the hexadecimal value must be specified with eight characters; the first two characters define the alpha component and the remainder the red, green, and blue components.

Beispiel ohne grafische Ausgabe

- ggt-Berechnung (klassisch)
 - Eingabe - zwei nichtnegative ganze Zahlen a und b
 - Ablauf
 - Wenn a gleich 0 ist, dann ist $\text{ggt} = b$
 - Sonst wiederhole so lange b nicht gleich 0 ist
 - Wenn $a > b$ ist, dann bekommt a den Wert von $a - b$
 - Sonst bekommt b den Wert von $b - a$
 - $\text{ggt} = a$
- Beispiel: ggt von $a=12$ und $b=44$
 1. $a = 12, b = 32$
 2. $a = 12, b = 20$
 3. $a = 12, b = 8$
 4. $a = 4, b = 8$
 5. $a = 4, b = 4$
 6. $a = 4, b = 0$
 - $\text{ggt} = 4$

Beispiel ggt-Berechnung

- Zahlen merken (**speichern**)
- Auf Zahlen Operationen anwenden (**rechnen**)
- Abhängig von einem Wert eine Entscheidung treffen (**vergleichen und verzweigen**)
- Ablauf möglicherweise öfter ausführen (**wiederholen**)
- Ablauf einen Namen geben und mit unterschiedlichen Werten aufrufen (**benennen und parametrisieren**)

Processing

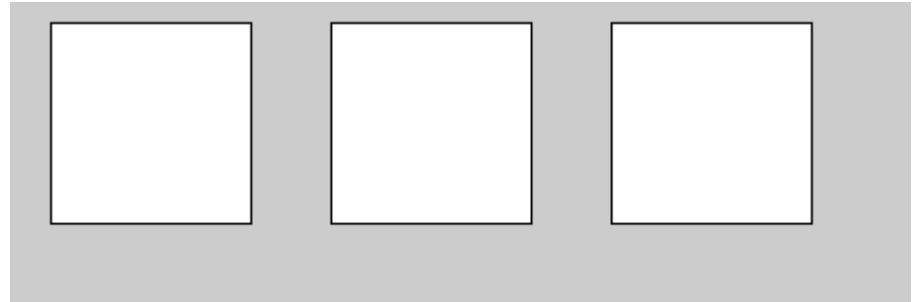


VARIABLEN

Motivation

- Ausgangsbeispiel

```
size(450, 150);  
rect(20, 10, 100, 100);  
rect(160, 10, 100, 100);  
rect(300, 10, 100, 100);
```



- Änderungswünsche (Beispiel)
 - Jedes Rechteck 120×120 Bildpunkte groß
 - Jedes Rechteck mit y-Koordinate 20
- Änderung
 - Alles händisch?
 - Was passiert bei neuen Anforderungen?

Lösung? - Variablen!

Variable

- Benannte Speicherstelle
- Wird einmal angegeben (deklariert)
 - Name
 - Datentyp
- Danach Zugriff über Name
- Wert im Speicher kann sich im Laufe des Programms ändern

Beispiel für Variable – Integer (ganze Zahlen)

- Variable für eine ganze Zahl mit dem Namen number
- Deklaration

```
int number;
```

- Bedeutung
 - `int` = Datentyp (steht für ganze Zahlen)
 - `number` = Name
- Datentyp für ganze Zahlen
 - 4 Bytes (32 Bits) für ganze Zahlen verwendet
 - Wertebereich (2^{32} Möglichkeiten = 4 294 967 296 Zahlen)
 - -2 147 483 648 bis +2 147 483 647
 - Beispiel: 2678
 - 0000 0000 0000 0000 0000 1010 0111 0110

- Der Datentyp legt fest
 - Wertebereich
 - Repräsentation im Speicher (wie viele Bytes werden belegt)
 - Erlaubte Operationen
- Beispiele für einfache (primitive) Datentypen
 - Ganze Zahlen (z.B. 10, 200)
 - Kommazahlen (z.B. 20.5, 73.23451)
 - Zeichen (z.B. 'a', 'C')
 - Wahrheitswert (true, false)
- Zeichenketten (kein einfacher Datentyp)
 - Beispiele
 - "Hallo"
 - "Eingabe:"

Wert zuweisen

- Form
 - Variable = „Wert“;
 - Bedeutung
 - Die rechte Seite (Wert) wird der linken Seite (Variable) zugewiesen
- Beispiele (Integer Variable)
 - Einfache Deklaration, danach Zuweisung (ohne Datentyp)
`int number;`
...
`number = 10;`
 - Deklaration mit Initialisierung
`int number = 10;`
 - Deklaration mit Initialisierung, Deklaration mit Zuweisung aus anderer Variable
`int number1 = 10;`
`int number2 = number1;`

Ausgangsbeispiel mit Variablen

- Ausgangsbeispiel ohne Variablen -> Ausgangsbeispiel mit Variablen

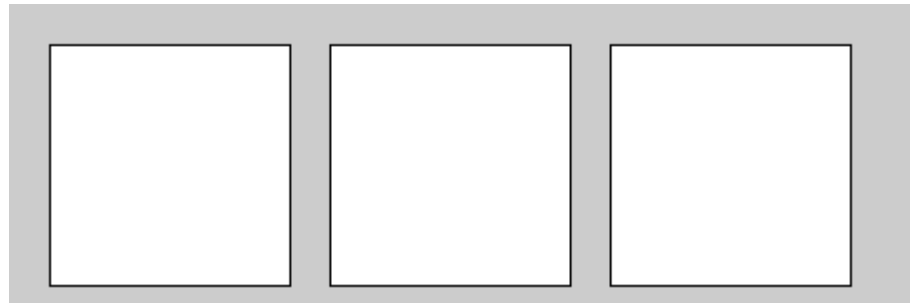
```
size(450, 150);  
rect(20, 10, 100, 100);  
rect(160, 10, 100, 100);  
rect(300, 10, 100, 100);
```



```
size(450, 150);  
int y = 10;  
int s = 100;  
rect(20, y, s, s);  
rect(160, y, s, s);  
rect(300, y, s, s);
```

- Beispiel mit anderer Variableninitialisierung

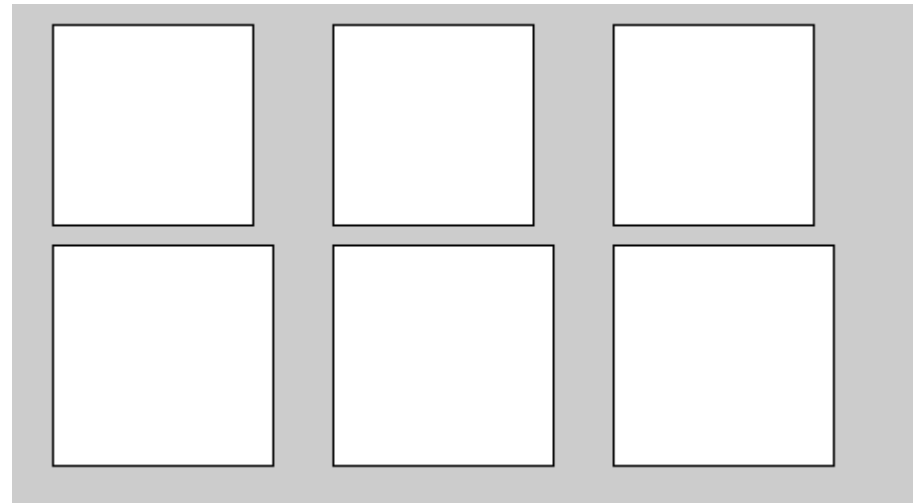
```
size(450, 150);  
int y = 20;  
int s = 120;  
rect(20, y, s, s);  
rect(160, y, s, s);  
rect(300, y, s, s);
```



Verändern von Variableninhalten

- Beispiel

```
size(450, 250);  
int y = 10, s = 100;  
rect(20, y, s, s);  
rect(160, y, s, s);  
rect(300, y, s, s);  
y = 120;  
s = 110;  
rect(20, y, s, s);  
rect(160, y, s, s);  
rect(300, y, s, s);
```



- Hinweise

- Variablen des gleichen Typs können in einer Zeile vereinbart werden
- Zuweisung verändert das Bitmuster im Speicher
 - Der alte Wert ist nicht mehr vorhanden

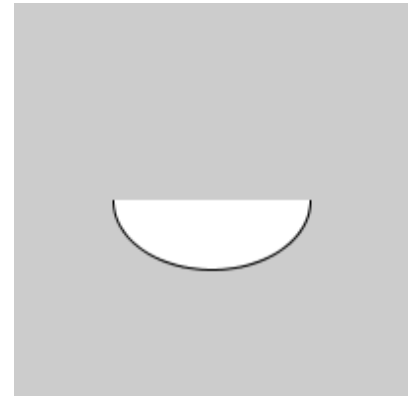
Primitive Datentypen in Processing

Typ	Größe in Bits	Wertebereich
boolean	meist 8	true oder false
char	16	<ul style="list-style-type: none">• Enthält u.a. Buchstaben (z.B. 'A'), Zahlen, weitere Alphabete, Sonderzeichen• Kann als Zahl aufgefasst werden (0 bis 65535)
byte	8	−128 bis +127 (-2^7 bis $2^7 - 1$)
short	16	−32768 bis +32767 (-2^{15} bis $2^{15} - 1$)
int	32	−2147483648 bis +2147483647 (-2^{31} bis $2^{31} - 1$)
long	64	−9223372036854775808 bis +9223372036854775807 (-2^{63} bis $2^{63} - 1$)
float	32	ca. -3.4×10^{38} bis 3.4×10^{38} (spezielle Darstellung für Kommazahlen)
double	64	ca. -1.8×10^{308} bis 1.8×10^{308} (spezielle Darstellung für Kommazahlen)
color	32	2^{24} Farben + Alphakanal

Processing-Variablen

- Spezielle Variablen
 - Informationen über das ablaufende Programm
 - Beispiele
 - `width` = Breite des Sketchfensters
 - `height` = Höhe des Sketchfensters
- Beispiele für Konstanten
 - `PI` entspricht Kreiszahl π
 - `HALF_PI` entspricht $\pi/2$
- Beispiel

```
size(200, 200);  
arc(width/2, height/2, 100, 70, 0, PI);
```



Zuweisungskompatibilität (1)

- Beispiel

```
int x;  
short y = 10;  
x = y;
```

- Kompatibilitätsbeziehung
 - \rightarrow = kann konvertiert werden in
 - `byte` \rightarrow `short` \rightarrow `int` \rightarrow `long` \rightarrow `float` \rightarrow `double`
 - `char` \rightarrow `int` ...

Zuweisungskompatibilität (2)

- Umkehrung
 - Das muss beim Programmieren explizit gesagt werden (**Cast**)
 - Processing bietet dafür auch eigene Funktionen an
 - **Achtung:** Kann zu Datenverlusten führen
- Beispiel (mit Processing-Funktion)

```
float x = 10.25;
```

```
int y;
```

```
...
```

```
y = int(x); // y = (int) x;
```



Cast



Welche der folgenden Deklarationen sind korrekt?

- a) `int x;` ☐
- b) `float y;` ☐
- c) `i float;` ☐
- d) `long huge;` ☐
- e) `double d;` ☐
- f) `large num;` ☐
- g) `char c;` ☐



- Sie haben folgende Variablen gegeben

`int i; float f; long l; double d; char c;`

Welche der folgenden Zuweisungen sind korrekt?

- | | |
|----------------------------|--------------------------|
| a) <code>i = 10;</code> | <input type="checkbox"/> |
| b) <code>f = 12;</code> | <input type="checkbox"/> |
| c) <code>d = 12.25;</code> | <input type="checkbox"/> |
| d) <code>f = d;</code> | <input type="checkbox"/> |
| e) <code>l = i;</code> | <input type="checkbox"/> |
| f) <code>c = 'A';</code> | <input type="checkbox"/> |
| g) <code>i = c;</code> | <input type="checkbox"/> |

OPERATOREN

Motivation

- Wir möchten auch rechnen und dann das Ergebnis einer Variable zuweisen

Lösung? - Operatoren!

Wichtige Operatoren

- Addition (+)
- Subtraktion (-)
- Multiplikation (*)
- Division (/)
- Modulo (%)
- Zuweisung (=)

Ausdruck und Anweisung

- Beispiele für Ausdrücke

$3 + 4$

$2 + 4 * 5$

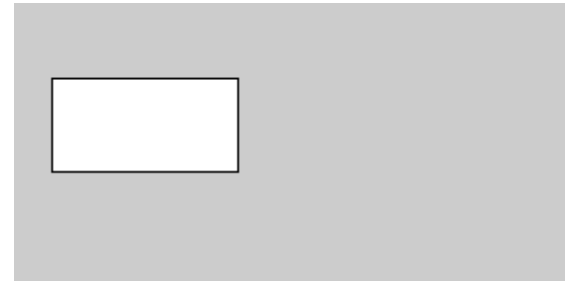
- Beispiele für Anweisungen (durch Anhängen eines Semikolons)

$x = 3 + 4;$

$x = y + 5 - 2;$

- Beispiel

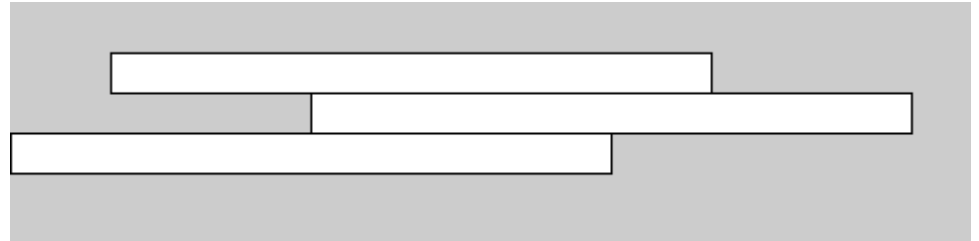
```
size(300, 150);  
int x = 20;  
int y = x + x;  
rect(x, y, 100, 50);
```



- Vorrang bei unterschiedlichen Operatoren
 - Zum Beispiel „Punkt- vor Strichrechnung“
- Beispiel
 - $x = 2 + 4 * 5;$
 - Operatoren: $*$, $+$, $=$ (geordnet nach Vorrang)
 - $4 * 5$ auswerten
 - $2 + 20$ berechnen
 - 22 der Variable x zuweisen

Beispiel für Operatoren mit Präzedenz

```
size(480, 120);  
int x = 50;  
int h = 20;  
int y = 25;  
int y2;  
rect(x, y, 300, h);  
x = x + 100;  
y2 = y + h;  
rect(x, y2, 300, h);  
x = x - 150;  
y2 = y + h * 2;  
rect(x, y2, 300, h);
```



```
size(480, 120);  
int x = 50;  
int h = 20;  
int y = 25;  
rect(x, y, 300, h);  
rect(x + 100, y + h, 300, h);  
rect(x - 50, y + h * 2, 300, h);
```

Ausdrücke wie $y + h * 2$ werden nur ausgewertet (keine Variable verändert) und **das Ergebnis** als Argument übergeben

Kürzere Schreibweise

- Kürzere Schreibweise für Operatoren

Operation	Bezeichnung	entspricht
Op1 += Op2	Additionszuweisung	Op1 = Op1 + Op2
Op1 -= Op2	Subtraktionszuweisung	Op1 = Op1 - Op2
Op1 *= Op2	Multiplikationszuweisung	Op1 = Op1 * Op2
Op1 /= Op2	Divisionszuweisung	Op1 = Op1 / Op2
Op1 %= Op2	Modulo-Zuweisung	Op1 = Op1 % Op2

Inkrement und Dekrement

- Inkrementoperator (++) bzw. Dekrementoperator (--)
 - Wert einer Variable um 1 erhöhen bzw. verringern
- ++
 - `a++`; entspricht `a += 1`; entspricht `a = a + 1`;

Operator	Benennung	Beispiel	Erklärung
++	Präinkrement	<code>++a</code>	a wird vor seiner weiteren Verwendung um 1 erhöht
++	Postinkrement	<code>a++</code>	a wird nach seiner weiteren Verwendung um 1 erhöht
--	Prädecrement	<code>--b</code>	b wird vor seiner weiteren Verwendung um 1 erniedrigt
--	Postdecrement	<code>b--</code>	b wird nach seiner weiteren Verwendung um 1 erniedrigt

- Beispiel (Folge von drei Anweisungen)

```
a = 3;  
b = ++a;  
c = a++;
```

Werte nach der dritten Anweisung:
a hat den Wert **5**
b und c haben den Wert **4**

Ausdrücke und unterschiedliche Datentypen

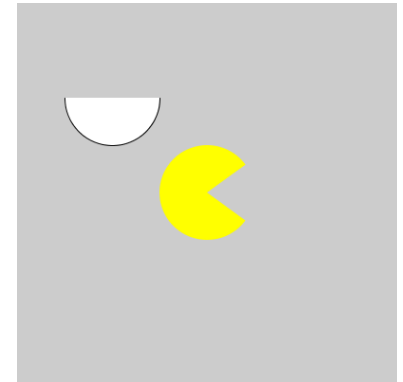
- Variablen unterschiedlichen Typs in einem Ausdruck
- Typ des Ausdrucks = „größter“ Typ im Ausdruck
 - Beispiel

```
int x = 10;  
float y = 20.0;  
float z = x + y;
```

Summe ist vom Typ float

Kommentare

- Für größeren Programmcode
 - Werden beim Ausführen ignoriert
 - `//` bei einzeiligen Kommentaren
 - `/* ... */` realisieren mehrzeilige Kommentare
- Beispiel



```
/* Simple
   program
   with
   output */

size(400, 400);
arc(100, 100, 100, 100, 0, PI); // semi circle

// Draw a Pac-Man
noStroke();
fill(255, 255, 0); // yellow
arc(width/2, height/2, 100, 100, 0.63, PI * 1.8);
```


Namenswahl

- Variablen
 - Kurze aber aussagekräftige (sprechende) Namen
- Englisch bevorzugt
- „lowerCamelCase“-Schreibweise
 - Beispiele
 - total**S**um
 - number**O**f**V**alues
 - line**W**idth
- Hilfsvariable
 - Kurze Namen oder nur Buchstaben (z. B: x, y, i)



Welche der folgenden Zuweisungen sind korrekt?

- a) `a = 2 + 3;` ☐
- b) `b = b + ;` ☐
- c) `c + 1 = 2 + 3;` ☐
- d) `d = +2 + a;` ☐
- e) `+e = 2 + 3;` ☐
- f) `f = + 2 3;` ☐



- Sie haben folgende Deklarationen gegeben

```
int a = 3, b = 5, c = 0, d = 0;  
float x = 1.75, y = 2.5, z = 1.0;
```

Welche Werte haben die Variablen c, d, z, a, c und b nach der Ausführung der folgenden Anweisungen?

```
c = 3 + 2 * -2 + a * 3;  
d = d * a - 2 * b + 4;  
z = x + y * a + 2;  
a++;  
c = a++ + 2;  
b = a + 2;
```

VERZWEIGUNGEN

Motivation

- Wir möchten an bestimmten Punkten im Programm Entscheidungen treffen
- Beispiel
 - Hat Variable x einen Wert < 10
 - Wenn ja, dann bestimmten Codeabschnitt ausführen
 - Ansonsten Codeabschnitt nicht ausführen

Lösung? – Verzweigungen!

Verzweigungen in Processing

- Einfache Form

```
if (test) {  
    statements  
}
```
- Schlüsselwort `if`
- `test` = Ausdruck (in Klammern), der ausgewertet wird
 - Wahrheitswert (muss `true` oder `false` ergeben)
 - Falls wahr (`true`)
 - Anweisungen (`statements`) im Block zwischen `{` und `}` ausführen
 - Eine Anweisung kann auch wieder eine `if`-Anweisung sein (Verschachtelung)
 - Wenn nur eine Anweisung
 - Klammern `{ }` können weggelassen werden

Vergleichsoperatoren

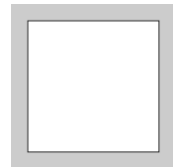
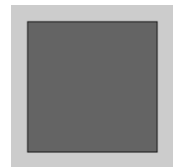
- Vergleichsoperatoren

Notation	Mathematische Notation
<code>a < b</code>	$a < b$
<code>a > b</code>	$a > b$
<code>a <= b</code>	$a \leq b$
<code>a >= b</code>	$a \geq b$
<code>a == b</code>	$a = b$
<code>a != b</code>	$a \neq b$

- Beispiel

```
size(200, 200);  
int rand = int(random(10));  
if (rand > 5) {  
    fill(100);  
}  
rect(20, 20, 160, 160);
```

Funktion random
liefert Zufallszahl

rand = 5	
rand = 9	

Logische Werte

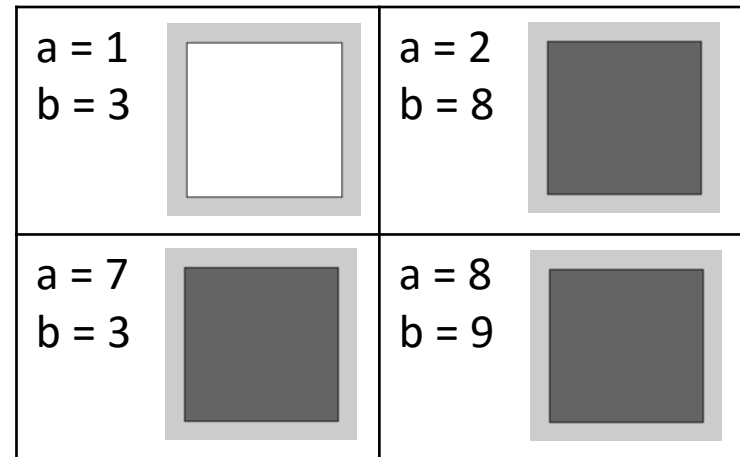
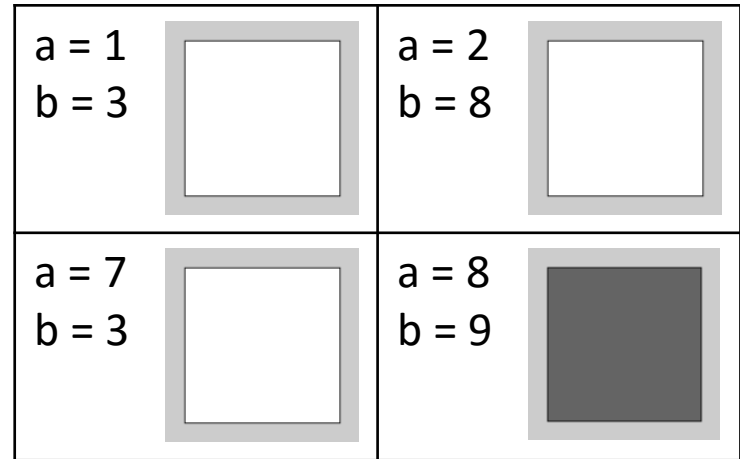
- Wertebereich umfasst 2 Werte
 - true und false
- Operationen
 - Negation: !
 - Oder: ||
 - Und: &&
 - XOR: ^

a	b	!a	a && b	a b	a ^ b
false	false	true	false	false	false
false	true		false	true	true
true	false	false	false	true	true
true	true		true	true	false

Beispiele

```
size(200, 200);  
int a = int(random(10));  
int b = int(random(10));  
if ((a > 5) && (b > 5)) {  
  fill(100);  
}  
rect(20, 20, 160, 160);
```

```
size(200, 200);  
int a = int(random(10));  
int b = int(random(10));  
if ((a > 5) || (b > 5)) {  
  fill(100);  
}  
rect(20, 20, 160, 160);
```



Präzedenz in Processing (Auswahl)

- Präzedenz (Vorrang)
 - Höchste Präzedenz zuerst, innerhalb einer Zeile gleiche Präzedenz

Symbole	Beispiel
()	a * (b + c)
++ -- !	a++, --b, !b
* / %	a * b
+ -	a + b
> < <= >=	if (a < b) { ... }
== !=	if (a == b) { ... }
&&	if ((a < c) && (b > c)) { ... }
	if (a (b > c)) { ... }
= += -= *= /= %=	a += 10

- Auswertung bei Operatoren auf gleicher Stufe (Assoziativität)
 - Meist von links nach rechts
 - Manchmal von rechts nach links (z.B. Zuweisung)

Komplexere Formen der Verzweigung (1)

- Mit else-Zweig

```
if (test) {  
    statements1  
} else {  
    statements2  
}
```

- Wenn test auf true ausgewertet
 - Anweisungen in statements1 ausführen
 - Andernfalls die Anweisungen in statements2 ausführen

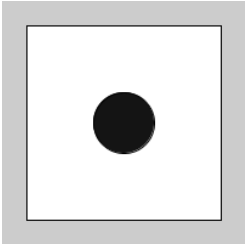
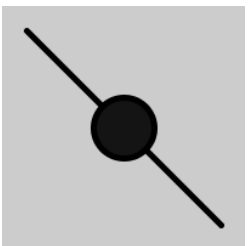
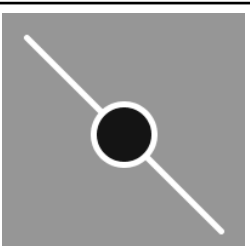
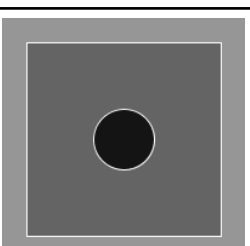
Komplexere Formen der Verzweigung (2)

- Mehrere Alternativen

```
if (test1) {  
    statements1  
} else if (test2) {  
    statements2  
} else if (test3) {  
    ...  
} else {  
    statementsX  
}
```

Beispiel

```
size(200, 200);
int rand = int(random(10));
if (rand > 5) {
  background(150);
  stroke(255);
  fill(100);
}
if (rand % 2 == 0) {
  rect(20, 20, 160, 160);
} else {
  strokeWeight(5);
  line(20, 20, 180, 180);
}
fill(20);
ellipse(100, 100, 50, 50);
```

rand = 2	
rand = 5	
rand = 7	
rand = 8	



- Sie haben folgenden Processing-Code gegeben

```
if (x > 5 || y < 4 && z > 6) {  
  fill(100);  
  rect(10, 10, 100, 100);  
}
```

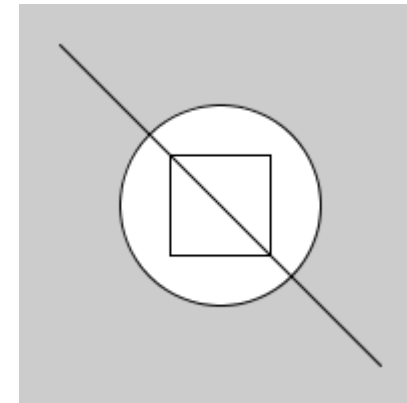
Bei welchen Kombinationen von Werten für x, y und z werden die Anweisungen im Block ausgeführt?

- a) x=2, y=4, z=5 ☐
- b) x=6, y=5, z=8 ☐
- c) x=5, y=3, z=2 ☐
- d) x=3, y=3, z=8 ☐



- Sie haben folgenden Processing-Code und eine dazugehörige Ausgabe gegeben

```
size(200, 200);  
int rand = int(random(10));  
if (rand < 5) {  
    rect(50, 50, 100, 100);  
    ellipse(100, 100, 100, 100);  
} else {  
    ellipse(100, 100, 100, 100);  
    rect(75, 75, 50, 50);  
}  
if (rand % 2 == 0) {  
    line(180, 20, 20, 180);  
} else {  
    line(20, 20, 180, 180);  
}
```



Welche Werte für rand erzeugen die obige Ausgabe?

SCHLEIFEN

Motivation

- Ausgangsbeispiel

```
size(480, 120);  
strokeWeight(8);  
line(20, 40, 80, 80);  
line(80, 40, 140, 80);  
line(140, 40, 200, 80);  
line(200, 40, 260, 80);  
line(260, 40, 320, 80);  
line(320, 40, 380, 80);  
line(380, 40, 440, 80);
```



- Problem

- Anweisung wird sehr oft mit kleinen Änderungen bei den Argumenten wiederholt

Lösung? - Schleifen!

for-Schleife

- Aufbau

```
for (init; test; update) {  
    statements  
}
```
- Schlüsselwort `for`
- `init` = Initialisierung vor dem Start der Schleife
 - Z.B. Laufvariable für Schleife vereinbaren und initialisieren
 - Eine Laufvariable gilt dann nur **innerhalb** der Schleife
- `test` = Abbruchtest für Beenden der Schleife
- `update` = Veränderung von Schleifenvariablen
 - Wird nach den Anweisungen ausgeführt
- `statements` = ein oder mehrere Anweisungen in einem Block
 - Bei einer einzigen Anweisung können die Klammern `{ }` weggelassen werden
- `init`, `test` oder `update` können leer bleiben

Beispiel

- Beispiel: Die Zahlen von 0 bis 9 ausgeben

```
for (int i = 0; i < 10; i++) {  
    println(i);  
}
```

- Ablauf

- Betreten der Schleife - i wird mit 0 initialisiert;
- Test auf $i < 10$ ergibt true
- `println(i)` gibt 0 aus
- i wird um 1 erhöht - i hat den Wert 1
- Test auf $i < 10$ ergibt true
- `println(i)` gibt 1 aus
- i wird um 1 erhöht - i hat den Wert 2
- ...
- i wird um 1 erhöht - i hat den Wert 10
- Test auf $i < 10$ ergibt **false** - **Ende der Schleife**



Ausgangsbeispiel angepasst

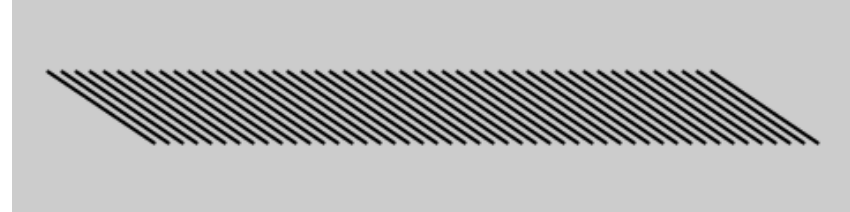
- Ausgangsbeispiel mit Schleife

```
size(480, 120);  
strokeWeight(8);  
for (int i = 20; i < 400; i += 60) {  
    line(i, 40, i + 60, 80);  
}
```

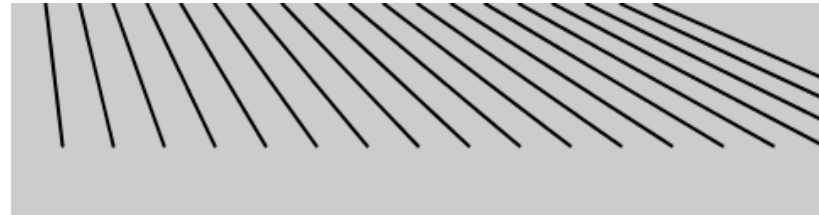


Weitere Beispiele

```
size(480, 120);
strokeWeight(2);
for (int i = 20; i < 400; i += 8) {
  line(i, 40, i + 60, 80);
}
```

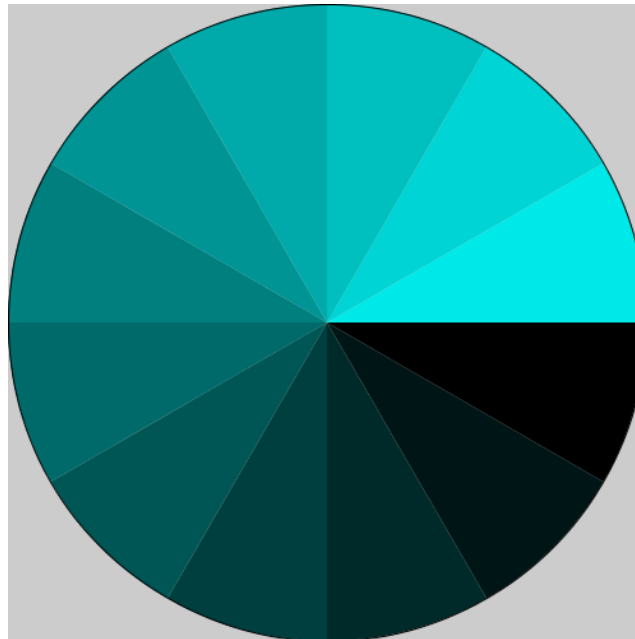


```
size(480, 120);
strokeWeight(2);
for (int i = 20; i < 400; i += 20) {
  line(i, 0, i + i/2, 80);
}
```



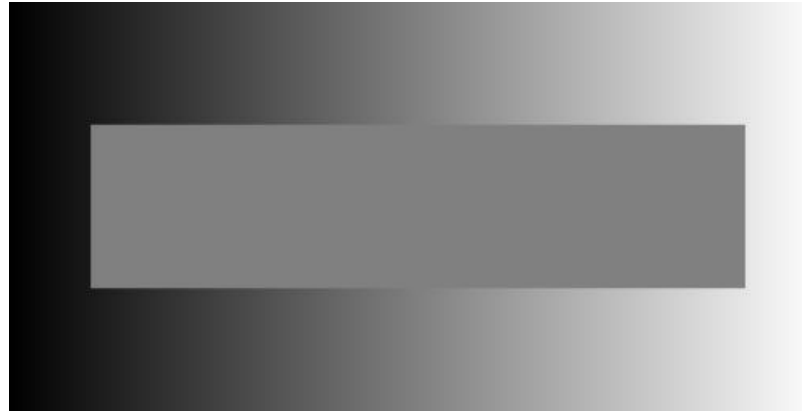
Weitere Beispiele

```
size(500, 500);
int parts = 12;
int degree = 360;
int c = 0;
for (int i = 0; i < degree; i += degree/parts) {
  c = int(map(i, 0, degree, 0, 255));
  fill(0, c, c);
  arc(width/2, height/2, width, height, radians(i), radians(i + degree/parts));
}
```



Komplexeres Beispiel 1 - Beschreibung

- Optische Täuschung

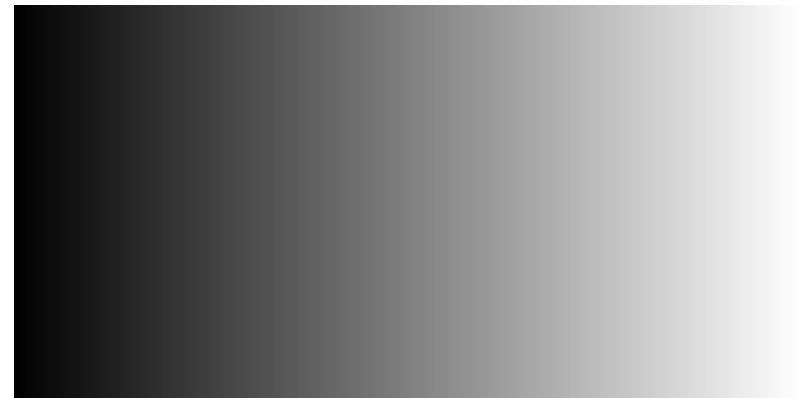


- Vorgehensweise
 - Hintergrund
 - Alle Graustufen von links nach rechts durchlaufen
 - 256 Graustufen
 - Breite des Fensters sollte ein Vielfaches der Anzahl der Graustufen sein
 - Höhe kann beliebig gewählt werden (hier Anzahl der Graustufen)

Komplexeres Beispiel 1 - Hintergrund

- Wir legen fest
 - Anzahl der Graustufen (shades)
 - Faktor für Vielfaches (factor)
 - Größe des Sketchfensters (shades * factor, shades)
- Zeichnen
 - Eine Graustufe mit einer Breite von factor Pixel mit einem Rechteck
 - Mit entsprechender Graustufe gefüllt
 - Keine Umrandungslinien (noStroke())
- Ergebnis

```
int shades = 256;
int factor = 2;
size(512, 256);
noStroke();
for (int i = 0; i < shades; i++) {
  fill(i);
  rect(i * factor, 0, factor, height);
}
```



Komplexeres Beispiel 1 - Abschluss

- Einfarbiger Balken über den Hintergrund
 - Größe möglichst flexibel
 - Startpunkt, Breite und Höhe mit Hilfe von width und height realisieren
 - Neue Farbe
 - Mittlere Graustufe gewählt
- Ergebnis

```
int shades = 256;
int factor = 2;
size(512, 256);
noStroke();
for (int i = 0; i < shades; i++) {
  fill(i);
  rect(i * factor, 0, factor, height);
}
fill(shades/2);
rect(width*0.1, height*0.3, width*0.8, height*0.4);
```



Komplexeres Beispiel 1 – Alternative Lösung

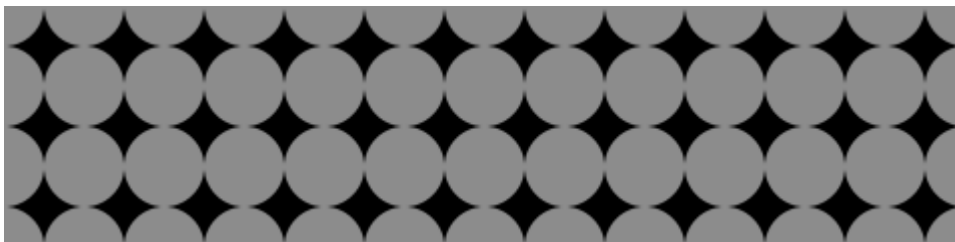
- Alternative
 - Zeichenfläche beliebig
 - Von links nach rechts pro X-Koordinate eine vertikale Linie
 - Graustufe ergibt sich aus der X-Koordinate der Linie
 - width Punkte und 256 Graustufen
 - Aktuelle X-Koordinate auf den Bereich 0 – 255 abbilden (map)
- Ergebnis

```
size(600,300);
for (int x = 0; x < width; x++) {
  stroke(map(x, 0, width - 1, 0, 255));
  line(x, 0, x, height - 1);
}
noStroke();
fill(128);
rect(width*0.1, height*0.3, width*0.8, height*0.4);
```

Verschachtelte Schleifen

- Schleifen können ineinander geschachtelt werden
 - Anzahl der Durchläufe erhöht sich entsprechend
- Beispiel

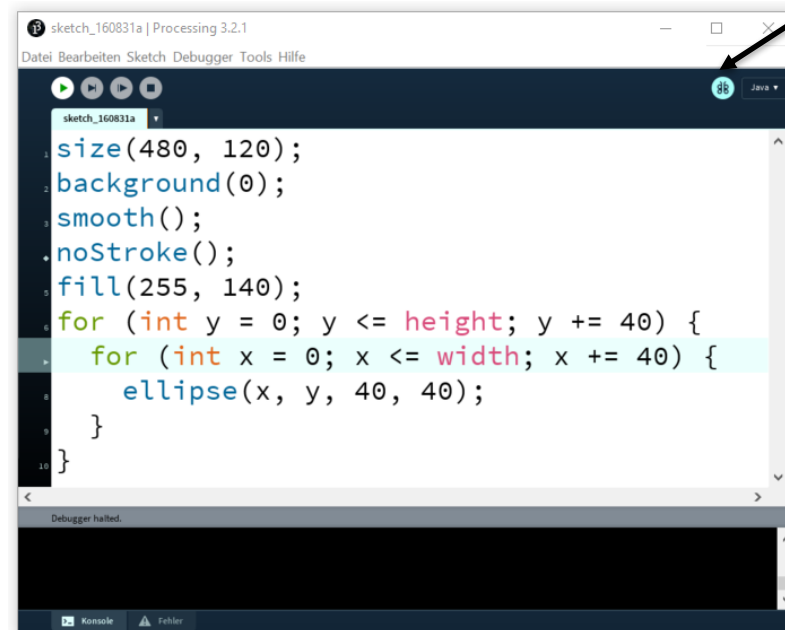
```
size(480, 120);  
background(0);  
noStroke();  
fill(255, 140);  
for (int y = 0; y <= height; y += 40) {  
  for (int x = 0; x <= width; x += 40) {  
    ellipse(x, y, 40, 40);  
  }  
}
```



Durchlauf	x	y
1	0	0
2	40	0
3	80	0
4	120	0
5	160	0
6	200	0
7	240	0
8	280	0
9	320	0
10	360	0
11	400	0
12	440	0
13	480	0
14	0	40
15	40	40
16	80	40
...
52	480	120

Debugging

- Debugger
 - Werkzeug zum Diagnostizieren und Auffinden von Fehlern in Programmen
- Typische Funktionen
 - Steuerung des Programmablaufs (Haltepunkte, engl. Breakpoints)
 - Schrittweise Durchführung von Programmen
 - Inspizieren und modifizieren von Variablen
- Debugger in Processing



Debugger
einschalten

A screenshot of the 'Variables' window in the Processing IDE. It shows a table with two columns: 'Name' and 'Value'. The table contains two rows: one for 'y' with a value of 40, and one for 'x' with a value of 120. Below the table, there is a section labeled 'Processing' which is currently empty. An arrow points from the text 'Variablen Inspizieren' to this window.

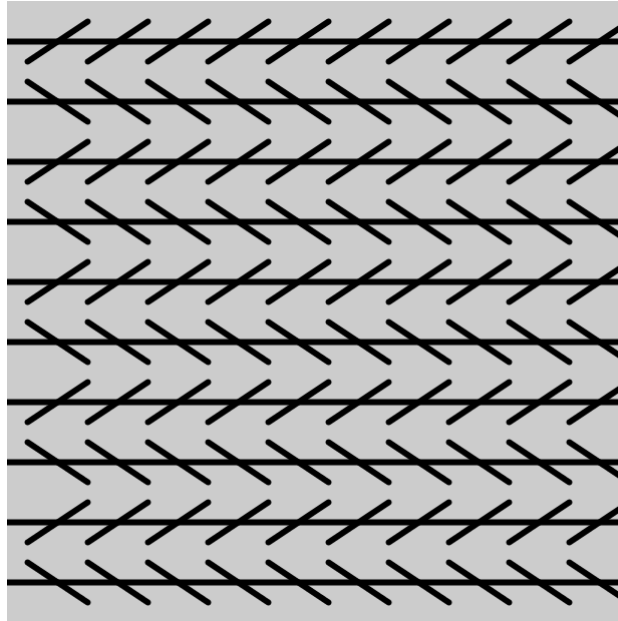
Name	Value
y	40
x	120

Processing

Variablen
Inspizieren

Komplexeres Beispiel 2 - Beschreibung

- Optische Täuschung

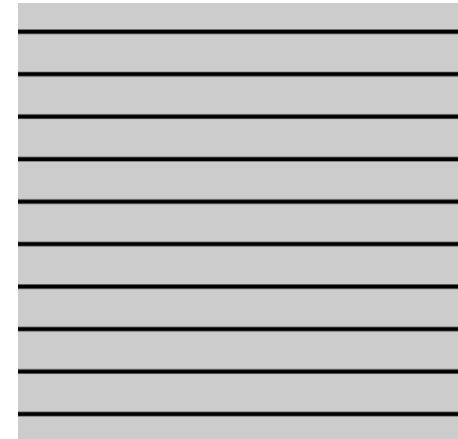


- Vorgehensweise
 - Schleife für horizontale Linien
 - Schleife für kürzere Linien
 - Abwechselnd nach links oder nach rechts geneigt

Komplexeres Beispiel 2 – Horizontale Linien

- Horizontale Linien
 - X-Achse
 - Start bei 0, Länge entspricht Breite des Fensters
 - Y-Achse (Werte frei gewählt)
 - Start bei 40, Inkrement von 60
- Ergebnis

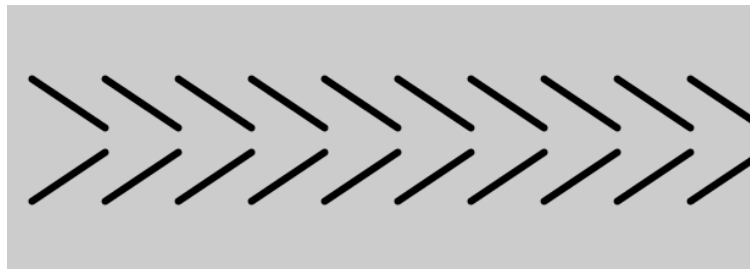
```
size(620, 620);  
strokeWeight(6);  
for (int y = 40; y < height; y += 60) {  
    line(0, y, width - 1, y);  
}
```



Komplexeres Beispiel 2 – Kürzere Linien

- Für jede horizontale Linie
 - Mehrere kürzere Linien
 - Abwechselnd nach links oder nach rechts geneigt
- Beispiel (noch einzeln realisiert)

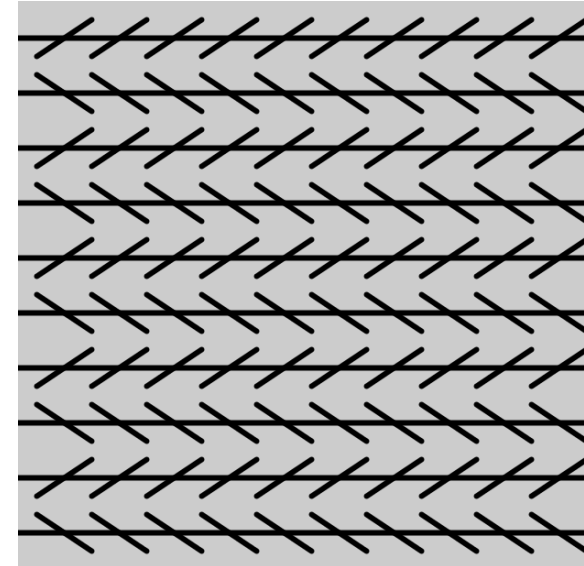
```
size(620, 220);  
strokeWeight(6);  
int increment = 60;  
int step = 40;  
for (int x = 20; x < width; x += increment) {  
    line(x, 60, x + increment, 60 + step);  
    line(x + increment, 120, x, 120 + step);  
}
```



Komplexeres Beispiel 2 – Abschluss

- Schleifen verschachteln
- Abwechselnd kürzere Linien nach links oder nach rechts zeichnen
- Ergebnis (einfacher Ansatz)

```
size(620, 620);
strokeWeight(6);
int increment = 60;
int step = 40;
boolean even = false;
for (int y = 20; y < height; y += increment) {
  line(0, y + step/2, width - 1, y + step/2);
  for (int x = 20; x < width; x += increment) {
    if (even) {
      line(x, y, x + increment, y + step);
    } else {
      line(x + increment, y, x, y + step);
    }
  }
  even = !even;
}
```





Welche Ausgabe wird durch folgende Schleifen erzeugt?

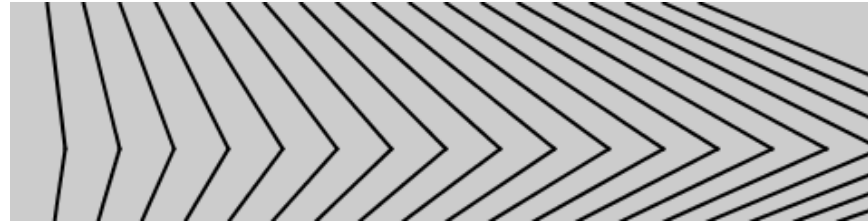
```
for (int i = 0; i < 5; i++) {  
    print(i);  
}
```

```
for (int i = 1; i <= 5; i += 2) {  
    print(i);  
}
```

```
for (int i = 5; i > 0; i--) {  
    print(i);  
}
```

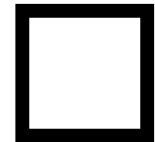


- Sie haben folgende Ausgabe gegeben (Größe 480 × 120)



Durch welche Schleife wurde diese Ausgabe erzeugt?

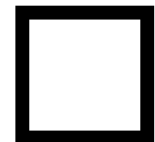
```
for (int i = 20; i < 400; i += 20) {  
    line(i, 0, i + i/2, 80);  
    line(i + i/2, 120, i * 1.2, 80);  
}
```



```
for (int i = 20; i < 400; i += 20) {  
    line(i, 0, i + i/2, 80);  
    line(i + i/2, 80, i * 1.2, 120);  
}
```



```
for (int i = 20; i < 400; i += 20) {  
    line(i - i/2, 80, i * 1.2, 120);  
    line(i, 0, i - i/2, 80);  
}
```



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 Aufrufer nur wichtig
 - Welchen Input (Parameter) kann man übergeben
 - Welche Auswirkung hat der Aufruf (Output, Rückgabewert)?
- Solche Funktionen kann man auch selbst schreiben

Beispiel

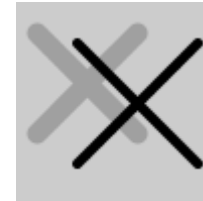
- Ein Kreuz zeichnen

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



- Zwei Kreuze zeichnen

```
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);
```



} Duplizierter Code mit veränderten Werten

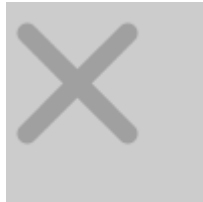
Beispiel mit Funktion (erster Schritt)

- Code wird in eine Funktion verpackt

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

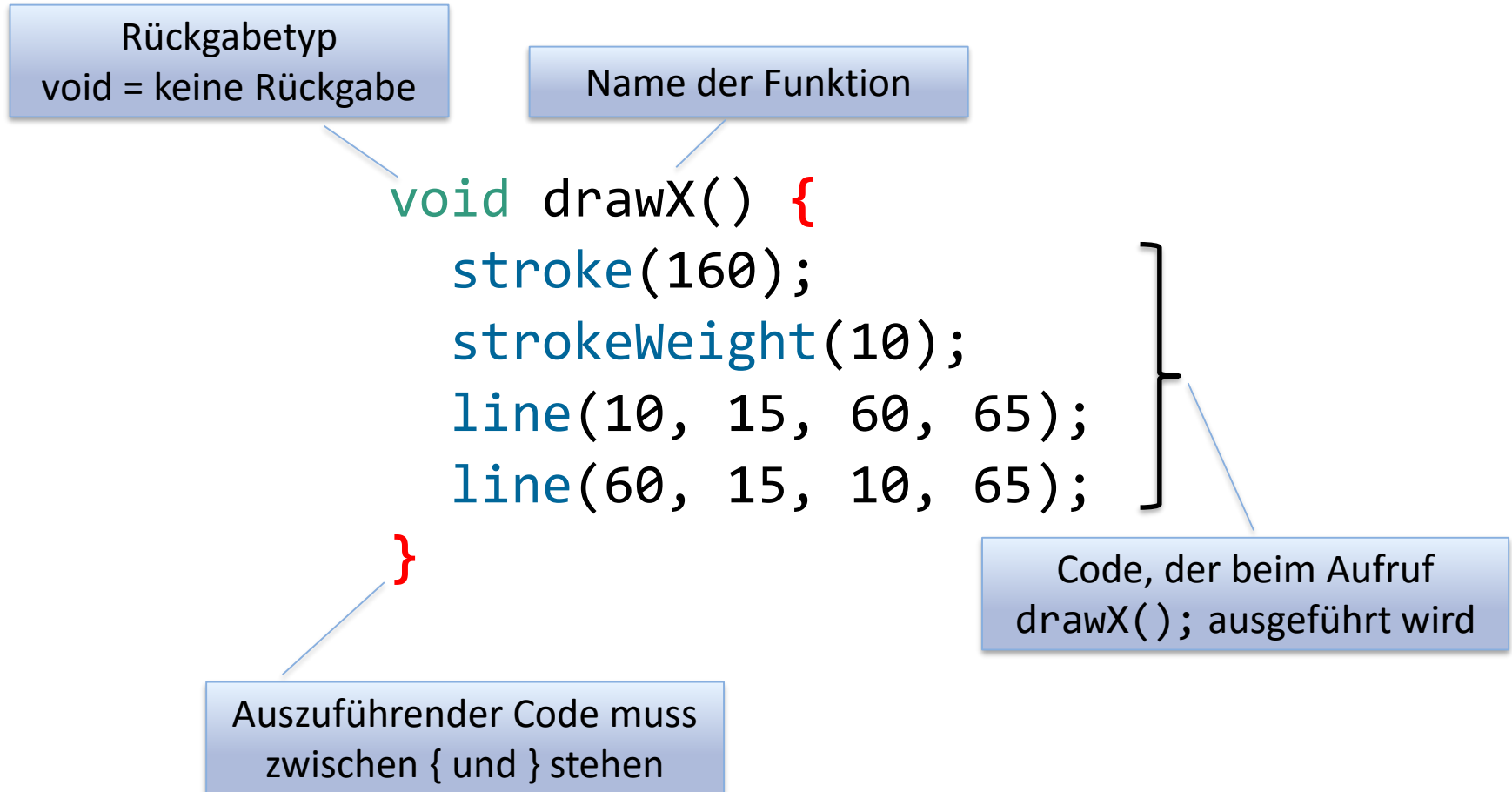


```
void setup() {  
  size(100, 100);  
  background(200);  
  drawX();  
}  
  
void drawX() {  
  stroke(160);  
  strokeWeight(10);  
  line(10, 15, 60, 65);  
  line(60, 15, 10, 65);  
}
```



- Eigene Funktion kann nur mehr aus einer anderen Funktion (z.B. setup) aufgerufen werden
- Funktion setup dient als Startpunkt für Processing-Programme

Anatomie der drawX-Funktion



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

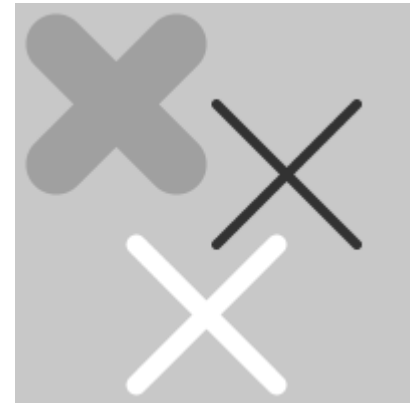
Parameter (wenn mehrere, dann durch Beistrich getrennt)
Form für jeden einzelnen Parameter: Datentyp Name

```
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)
 - Bis zum Beenden des Programms

- 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 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;
```

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

```
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);  
}
```

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

The diagram illustrates function return types using two code snippets. The first snippet, `void setup() { float f = average(12.0, 6.0); println(f); }`, shows a function that returns `void`. The second snippet, `float average(float num1, float num2) { float av = (num1 + num2)/2.0; return av; }`, shows a function that returns a `float`. Annotations include: 'Typangabe' (Type specification) with arrows pointing to the `void` and `float` return types; and 'Beim Rücksprung wird Wert zurückgeliefert' (When returning, a value is delivered) with an arrow pointing to the `return av;` statement.

```
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;  
}
```

Typangabe

Beim Rücksprung wird Wert zurückgeliefert

Beispiel (Maximum zweier Zahlen, mehrere Versionen)

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

Mehrere return-
Anweisungen möglich

```
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

- 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)

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

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

- Hinweis
 - Rekursion und Iteration sind hier gleich mächtig
 - Man kann hier die iterative Berechnungen in eine rekursive umwandeln und umgekehrt

Beispiel

```
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;  
    }  
}
```

Ablauf der Rekursion bei sum

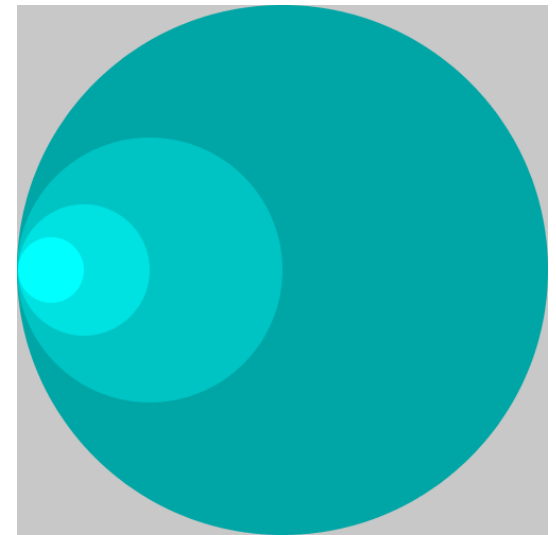
```
...  
int sumRecursive(int num) {  
    if (num > 0) {  
        return num + sumRecursive(num - 1);  
    } else {  
        return 0;  
    }  
}
```

Rekursion sumRecursive(3)

```
sumRecursive(3) = 3 + sumRecursive(2)  
sumRecursive(2) = 2 + sumRecursive(1)  
sumRecursive(1) = 1 + sumRecursive(0)  
sumRecursive(0) = 0  
sumRecursive(1) = 1 + 0 = 1  
sumRecursive(2) = 2 + 1 = 3  
sumRecursive(3) = 3 + 3 = 6
```

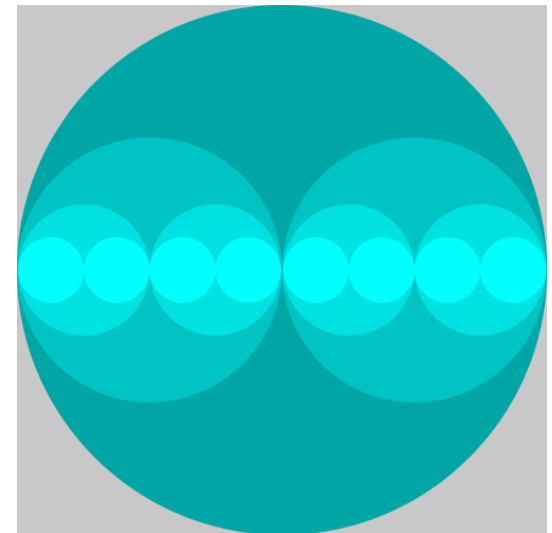
Ein visuelles Beispiel ...

```
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);  
  }  
}
```

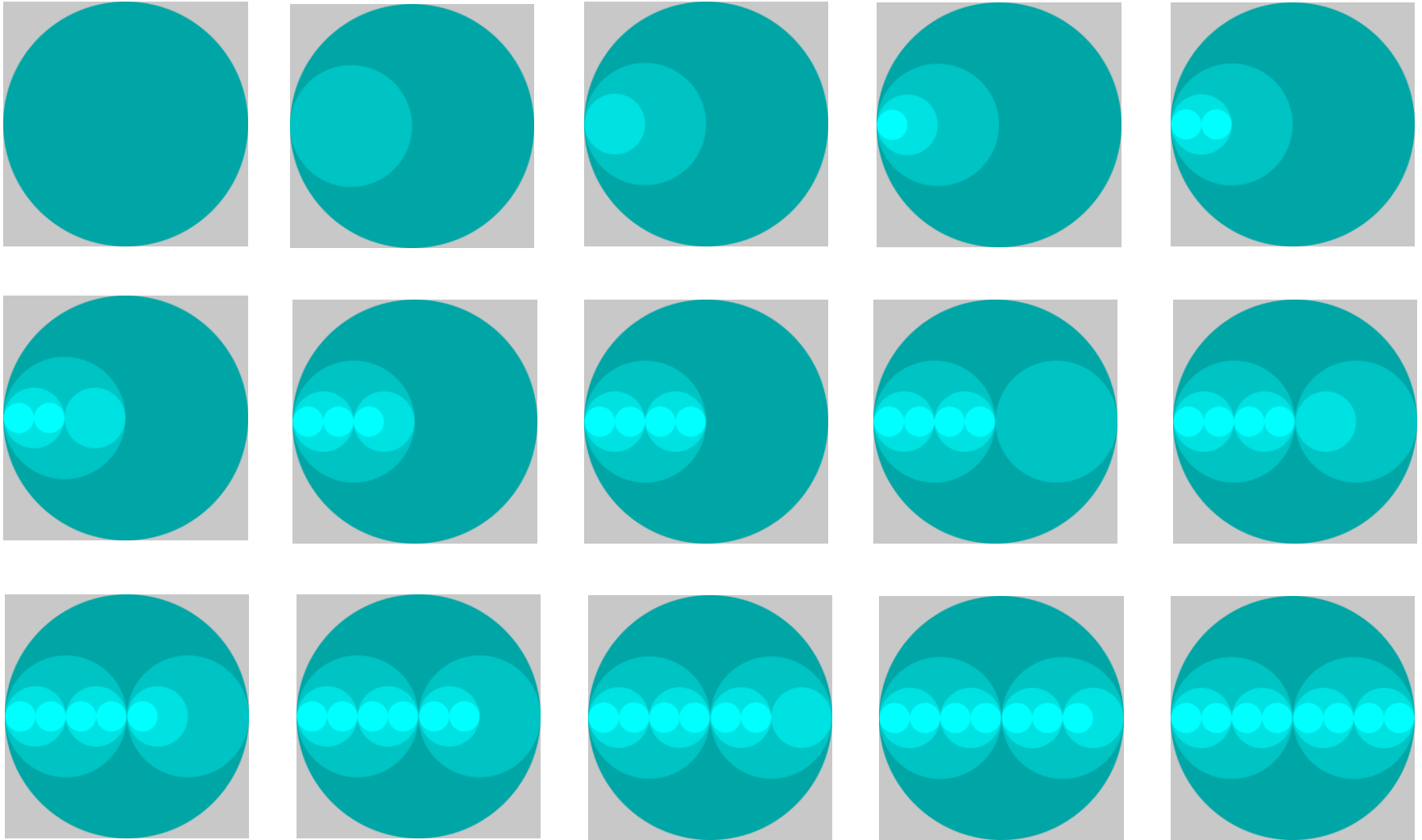


... und was wird jetzt gezeichnet?

```
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?





- Eine Funktion sollte die Summe von drei ganzen Zahlen zurückgeben

Welche der folgenden Implementierungen sind korrekt?

```
int sum(int a, int b, int c) {  
    int s = a + b + c;  
    return s;  
}
```

```
int sum(int a, int b, int c) {  
    return a + b + c;  
}
```

```
void sum(int a, int b, int c) {  
    int s = a + b + c;  
}
```

```
short sum(int a, int b, int c) {  
    return a + b + c;  
}
```

```
int sum(int a, b, c) {  
    int s = a + b + c;  
    return s;  
}
```

```
float sum(int a, int b, int c) {  
    return a + b + c;  
}
```



- Sie haben folgende rekursive Funktion gegeben

```
int x(int n) {  
    return n==0 ? 1 : x(n-1) * n;  
}
```

Welche Werte geben folgende Aufrufe aus?

x(3)
x(4)
x(2 + 3)
x(0)
x(-1)

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 bei Deklaration
`int[] arr = new int[10];`
- Späteres Anlegen
`int[] arr;`
...
`arr = new int[10];`

Anlegen von Arrays (Beispiel int-Arrays)

- Beispiel

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

- Die Arrayelemente haben zunächst 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)

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

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

Anlegen von Arrays – mit Initialisierung

- Mit Initialisierung

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

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

Ausgabe:

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);  
}
```



Beispiel (Anwendung bei draw)

- Mauszeiger verfolgen
- Aktuelle Mausposition
 - mouseX
 - mouseY
- 2 Arrays
 - Die letzten 80 Werte

```
int max = 80;
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++){
    stroke(255, 0, 0, 100 - i);
    strokeWeight(i*1/3.0);
    line(x[i], y[i], x[i + 1], y[i + 1]);
  }
}
```

Zuweisung

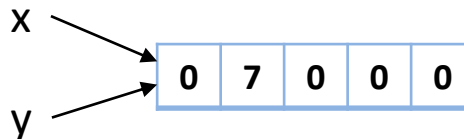
- Arrayvariable ist eine Referenz auf das eigentliche Array
- Einer Arrayvariable vom Typ x kann immer nur ein Array vom Typ x zugewiesen werden
 - Bei der Zuweisung wird nur die Adresse im Speicher kopiert, **nicht** der Inhalt
- Beispiel

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

Ausgabe:

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

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

Kopie eines
Arrays anlegen

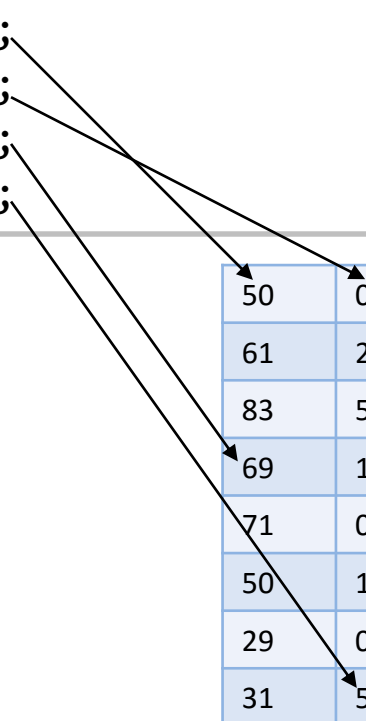
Ausgabe:

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

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]);
```



The diagram shows four arrows originating from the code snippets and pointing to specific cells in the table below:

- An arrow from `x[0][0]` points to the cell containing 50.
- An arrow from `x[0][1]` points to the cell containing 0.
- An arrow from `x[3][0]` points to the cell containing 69.
- An arrow from `x[7][1]` points to the cell containing 51.

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

Ausgabe:
50
0
69
51



Welche der folgenden Deklarationen erzeugt ein Array mit 3 Elementen?

- | | |
|--|--------------------------|
| <code>int[] array;</code> | <input type="checkbox"/> |
| <code>int[] array = new int[3];</code> | <input type="checkbox"/> |
| <code>int[3] array;</code> | <input type="checkbox"/> |
| <code>int[] array = new array[3];</code> | <input type="checkbox"/> |
| <code>int[] array = {1, 2, 3};</code> | <input type="checkbox"/> |
| <code>int[] array = 3;</code> | <input type="checkbox"/> |



Welche der folgenden Schleifen geben alle Elemente eines Arrays array aus?

```
for (int i = 0; i < array.length; i++) {  
    print(array[i]);  
}
```

☐

```
for (int i = 0; i < array.length;) {  
    print(array[i++]);  
}
```

☐

```
for (int i = 1; i <= array.length; i++) {  
    print(array[i]);  
}
```

☐

```
for (int i = array.length-1; i >= 0; i--) {  
    print(array[i]);  
}
```

☐

AUSBLICK

Was haben Sie in diesem Prolog-Teil kennengelernt ?

- 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, ...)

Was man mit Processing programmieren kann

Präsentation von Beispielen aus **Processing: Creative Coding and Generative Art in Processing 2** (Ira Greenberg et. al)



- 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 VO Programmkonstruktion

- 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