

Programmieren – Teil 1

Propädeutikum (Prolog) 2025
Jürgen Kogler und Martin Riener
TU Wien

Allgemeines

Verwendete Plattformen

- TISS:

- <https://tiss.tuwien.ac.at/course/courseDetails.xhtml?dswid=3199&dsrid=177&semester=2025W&courseNr=180771>

- TUWEL:

- <https://tuwel.tuwien.ac.at/course/view.php?id=75192>

- Slido:

- <https://app.sli.do/event/jq4asgTqTEx86aAi3dizRG>
- Code: #Prolog2025W

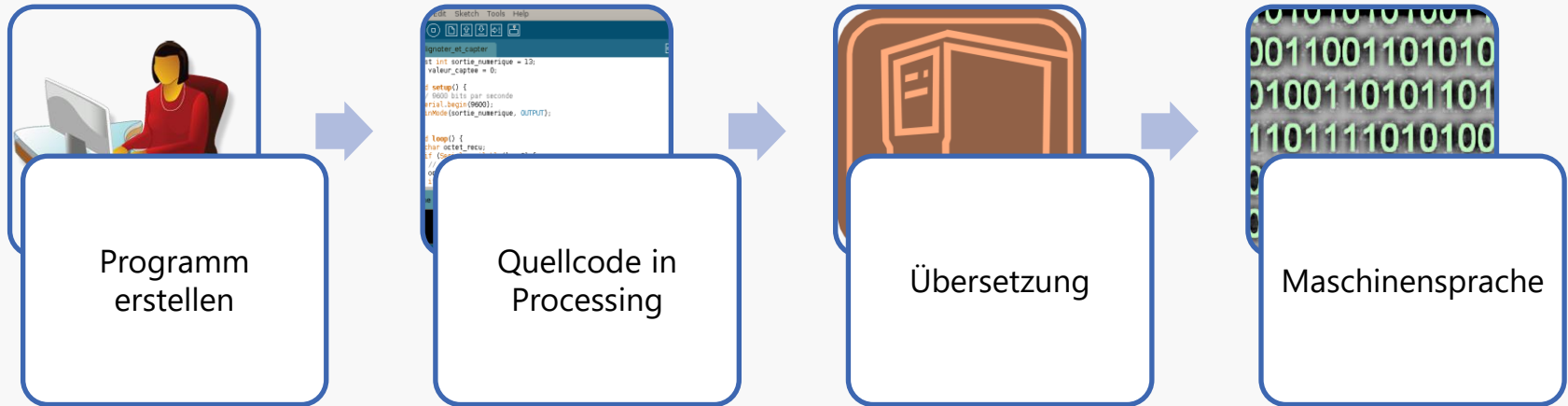
Processing – Grundlagen

Anweisungen für Roboter



Programmierung (in Processing)

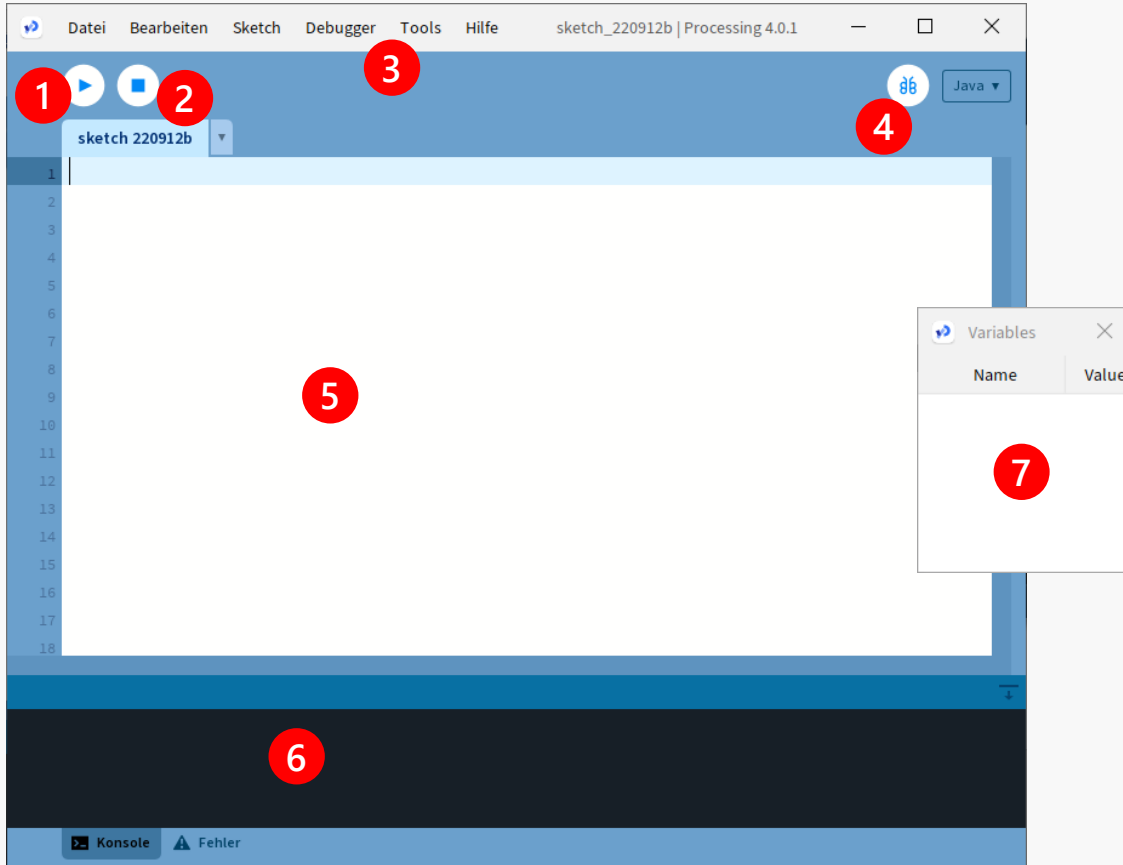
- Maschinensprache = Befehle, die ein Prozessor versteht
- Höhere Programmiersprache (z. B. Processing)
 - Für Menschen leichter
 - Muss bestimmten Regeln folgen (Unterstützung der Übersetzung)



Processing

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

Entwicklungsumgebung



1 Programm starten

2 Programm beenden

3 Menüleiste

4 Debugger

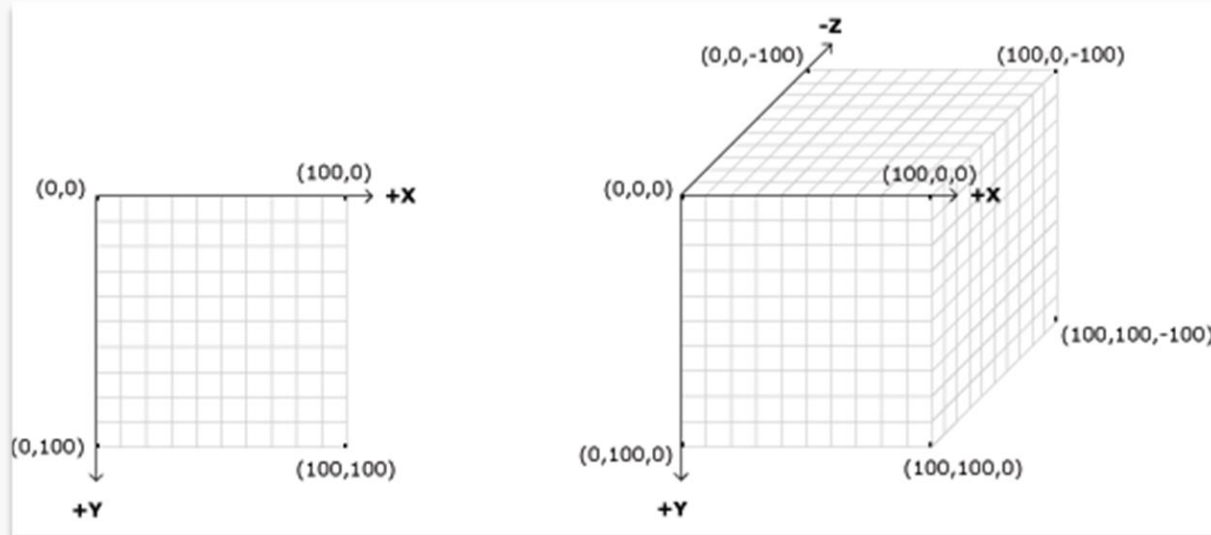
5 Texteditor

6 Konsole (Textausgabe)

7 Grafische Ausgabe
(Sketchfenster)

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

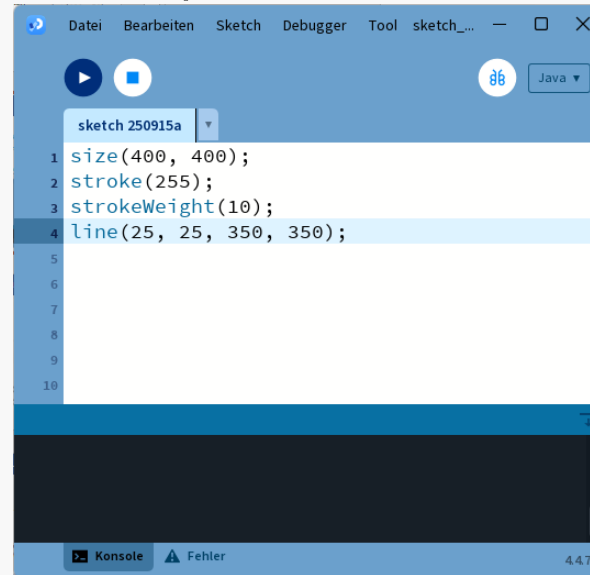
- Sketchfenster mit der Größe 200 × 200 (Breite × Höhe) zeichnen
- Sketchfenster mit der Größe 200 × 200 und danach Punkt (Koordinaten x=100 und y=50) zeichnen

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

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

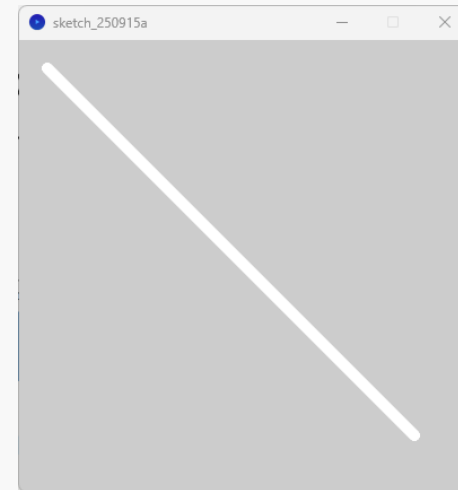
Beispiel

- Beispiel
 - Sketchfenster mit der Größe 400×400
 - Zeichenfarbe für Linien auf weiß setzen
 - Dicke für Linien auf 10 Pixel setzen
 - Linie zwischen Startpunkt (25, 25) und Endpunkt (350, 350) zeichnen

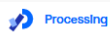


```
1 size(400, 400);  
2 stroke(255);  
3 strokeWeight(10);  
4 line(25, 25, 350, 350);  
5  
6  
7  
8  
9  
10
```

The screenshot shows the Arduino IDE interface. The code editor displays the following code: `size(400, 400);`, `stroke(255);`, `strokeWeight(10);`, and `line(25, 25, 350, 350);`. The line numbers 1 through 10 are visible on the left. The IDE has a blue theme and a menu bar with options like 'Datei', 'Bearbeiten', 'Sketch', 'Debugger', and 'Tool'.



Reference (<https://processing.org/reference/>)

[Processing](#)[Download](#)[Documentation](#)[Learn](#)[Teach](#)[About](#)[Donate](#)

Reference

Filter by keywords...

Shortcuts

Data
Rendering
Output
Structure

Input
Image
Color
Control

Constants
Shape
Lights Camera
Environment

Typography
Math
Transform

Data

Composite

Array
ArrayList
FloatDict
FloatList
HashMap
IntDict
IntList

An array is a list of data

An `ArrayList` stores a variable number of objects

A simple table class to use a `String` as a lookup for a float value

Helper class for a list of floats

A `HashMap` stores a collection of objects, each referenced by a key

A simple class to use a `String` as a lookup for an int value

Helper class for a list of ints

We need
your help!

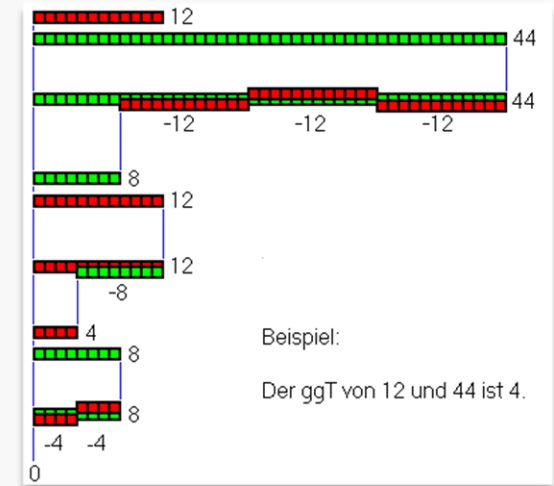


Help us continue with your
generosity!

[Donate](#)

Beispiel ohne grafische Ausgabe (1)

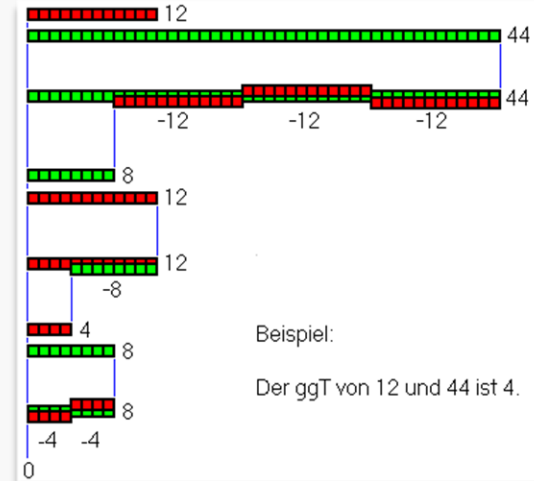
- ggT-Berechnung (klassisch)
- Vorgehen
 - Eingabe - zwei nicht negative ganze Zahlen a und b
 - Ablauf
 - Wenn a gleich 0 ist, dann ist $\text{ggT} = b$
 - Sonst wiederhole so lange b nicht gleich 0 ist
 - Wenn $a > b$ ist, dann bekommt a den Wert von $a - b$
 - Sonst bekommt b den Wert von $b - a$
 - $\text{ggT} = a$



https://de.wikipedia.org/wiki/Euklidischer_Algorithmus#/media/File:Euklidischer_Algorithmus.png

Beispiel ohne grafische Ausgabe (2)

- Praktisches 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 in dieser LVA

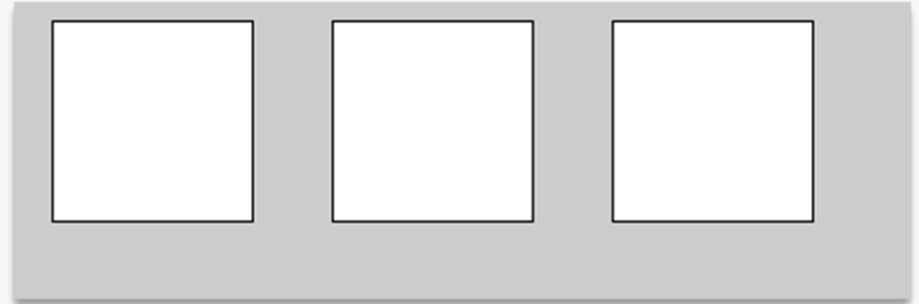


Variablen

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?

Variable

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

Beispiel für Variable

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

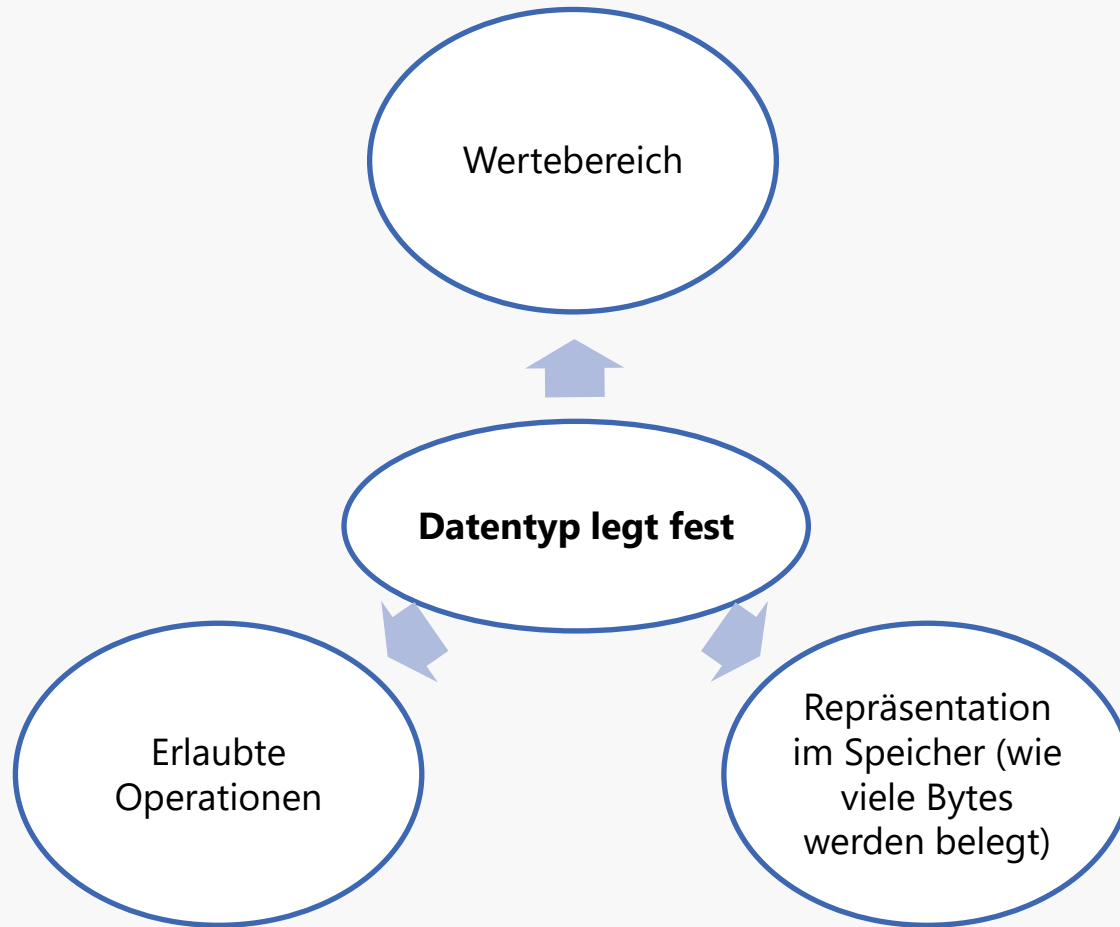
```
int number;
```

- Bedeutung
 - `int` = Datentyp (steht für ganze Zahlen)
 - `number` = Name

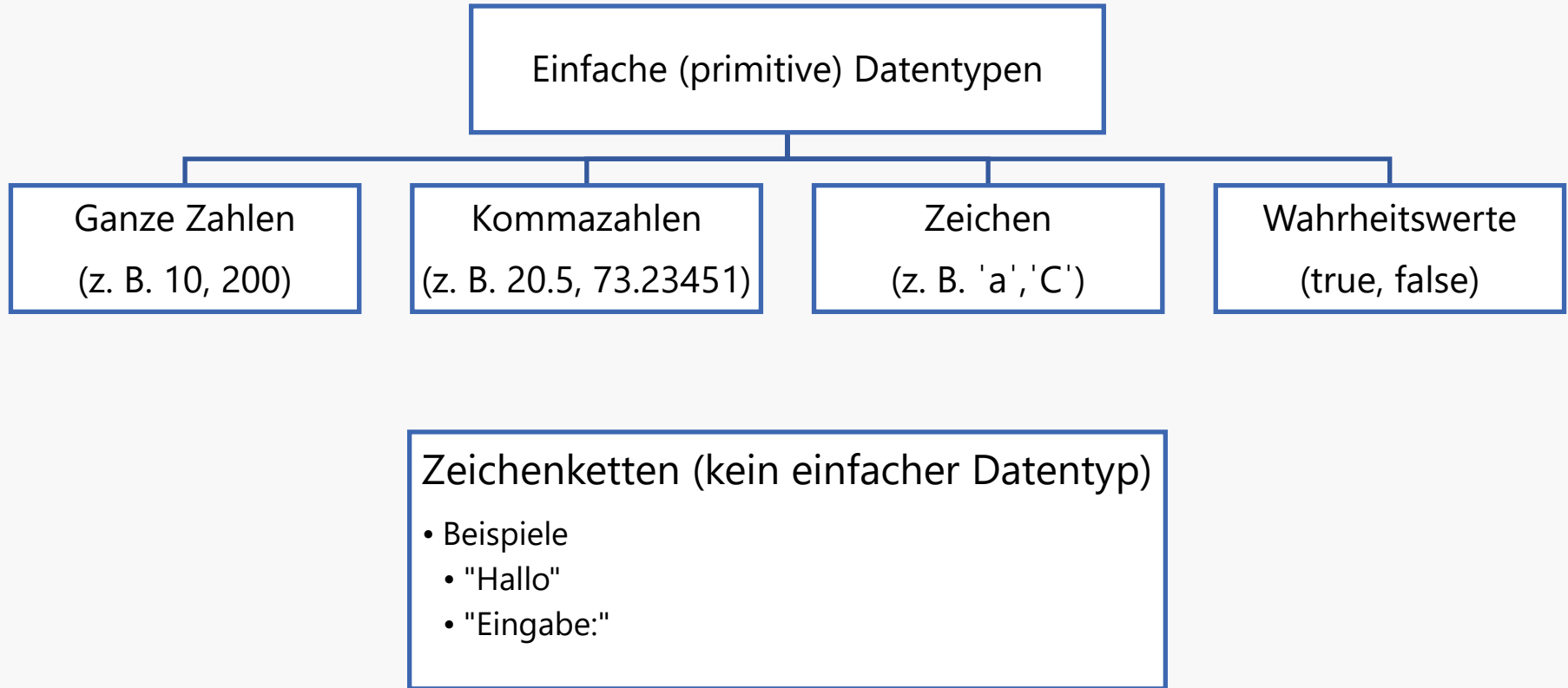
Datentyp Integer

- 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

Datentyp allgemein



Beispiele für Datentypen



Wert zuweisen

- Form
 - Variable = „Wert“;
- Ablauf
 - Die rechte Seite wird zuerst ausgewertet
 - Danach wird der Wert der linken Seite zugewiesen
- Aufbau
 - Links steht eine Variable
 - Rechts kann ein fixer Wert (Literal), eine Variable oder eine Berechnung stehen

Deklaration

- Einfache Deklaration, danach Initialisierung (ohne Datentyp)

```
int number;
```

```
...
```

```
number = 10;
```

- Deklaration mit Initialisierung

```
int number = 10;
```

- Deklaration mit Initialisierung (im zweiten Fall wird schon bekannte Variable benutzt)

```
int number1 = 10;
```

```
int number2 = number1;
```

Benutzung von Variablen – Beispiel (1)

- Ausgangsbeispiel ohne Variablen

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

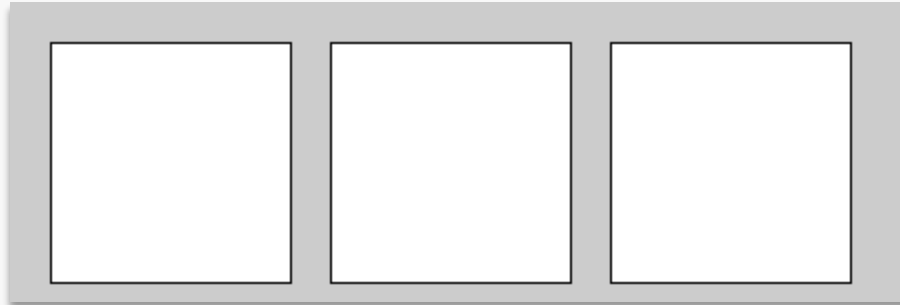
- Ausgangsbeispiel mit Variablen

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

Benutzung von Variablen – Beispiel (2)

- Beispiel mit anderer Variableninitialisierung

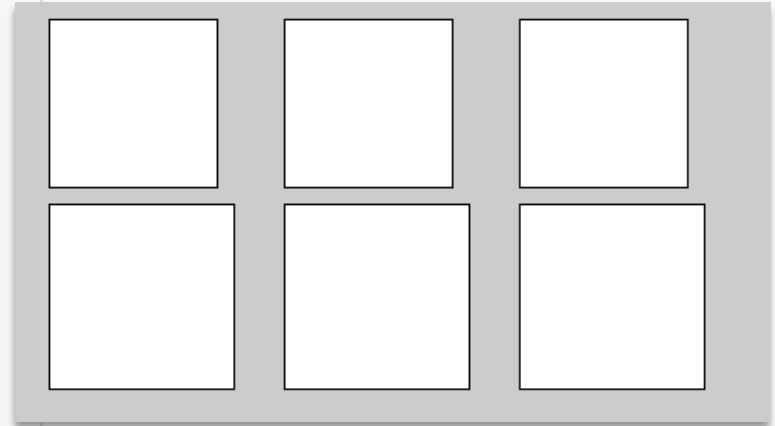
```
size(450, 150);  
int y = 20;  
int sideLength = 120;  
rect(20, y, sideLength, sideLength);  
rect(160, y, sideLength, sideLength);  
rect(300, y, sideLength, sideLength);
```



Verändern von Variableninhalten

- Beispiel

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



Variablen des gleichen Typs können in einer Zeile vereinbart werden!
Erneute Zuweisung nach der Initialisierung verändert das Bitmuster im Speicher -
der alte Wert ist **nicht** mehr vorhanden (destruktive Zuweisung)!

Verändern von Variableninhalten verhindern

- Schlüsselwort `final` bei der Deklaration benutzen
- Beispiele

```
final int number = 100;
```

```
final float value = 5.5;
```

- Fixer Wert, der nicht mehr verändert werden kann

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, SonderzeichenKann 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

Beispiele für Deklarationen

- long-Variable

```
long largeValue = 10000000;
```

- double-Variable

```
double fraction = 12.456;
```

- boolean-Variable

```
boolean check = true;
```

- char-Variable

```
char letter = 'a';
```

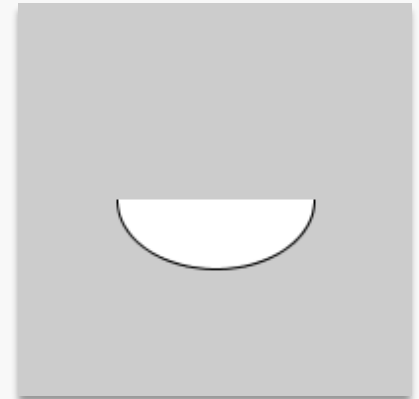
- String-Variable

```
String greeting = "hello";
```


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);
```

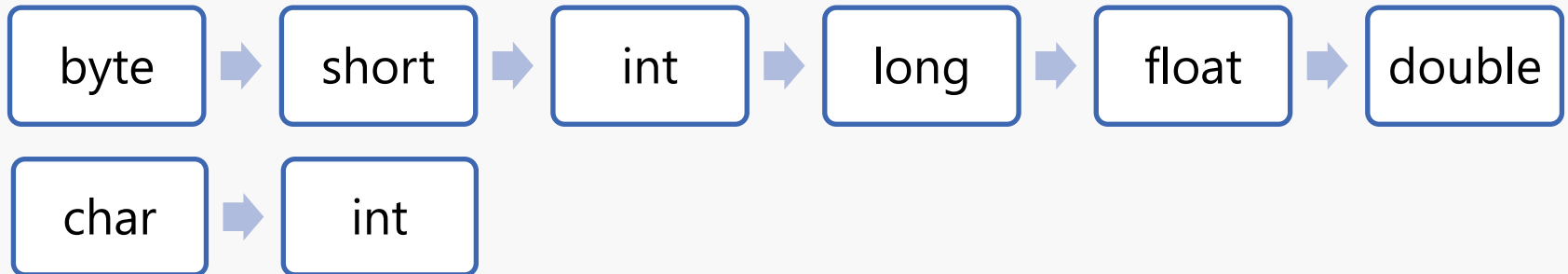


Zuweisungskompatibilität (1)

- Beispiel

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

- Kompatibilitätsbeziehung



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

Operatoren

Motivation

- Bis jetzt nur Werte zugewiesen
 - Wir möchten mit den Werten auch rechnen
 - Ergebnisse wieder Variablen 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);
```

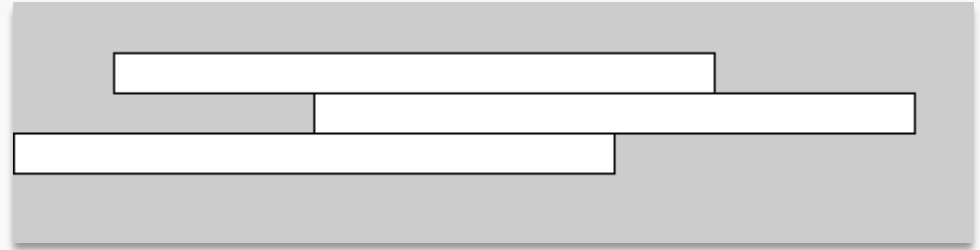


Operatorvorrang

- Vorrang bei unterschiedlichen Operatoren
 - Zum Beispiel „Punkt- vor Strichrechnung“
- Beispiel $x = 2 + 4 * 5;$
 - Operatoren: $*$, $+$, $=$ (geordnet nach Vorrang)
 - $4 * 5$ auswerten, dann $2 + 20$ berechnen
 - 22 der Variable x zuweisen
- Alternative
 - Auswertung durch Klammerung bestimmen
 - Beispiel
 - $x = (2 + 4) * 5;$
 - x wird der Wert 30 zugewiesen

Beispiel für Operatorvorrang

```
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);
```

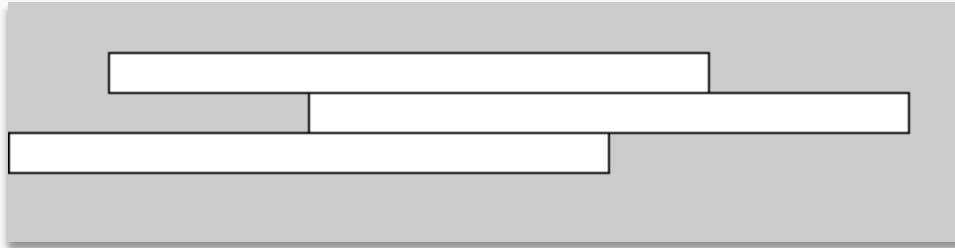


Ausdrücke bei Parameterübergabe

- Beispiel von vorheriger Folie verkürzt

```
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 wird verändert) und **das Ergebnis** als Argument übergeben



Operatorvorrang in Processing (Auswahl)

- Operatorvorrang
 - Höchste Stufe zuerst
 - Innerhalb einer Zeile gleiche Stufe
- Assoziativität
 - Auswertung auf gleicher Stufe
 - Meist von links nach rechts
 - Manchmal von rechts nach links
 - z. B. Zuweisung

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

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	++a	a wird vor seiner weiteren Verwendung um 1 erhöht
++	Postinkrement	a++	a wird nach seiner weiteren Verwendung um 1 erhöht
--	Prädekrement	--b	b wird vor seiner weiteren Verwendung um 1 verringert
--	Postdekrement	b--	b wird nach seiner weiteren Verwendung um 1 verringert

Inkrement und Dekrement – Beispiel

- 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 können in einem Ausdruck vorkommen
- Typ des Ausdrucks = „größter“ Typ im Ausdruck
 - Beispiel

```
int x = 10;
```

```
float y = 20.0;
```

```
float z = x + y;
```

Summe ist vom Typ float

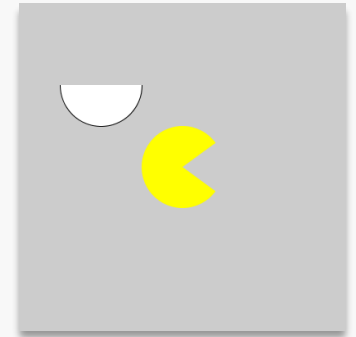
Kommentare

- 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)

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

```
if (test) {  
    statements  
}
```

Vergleichsoperatoren

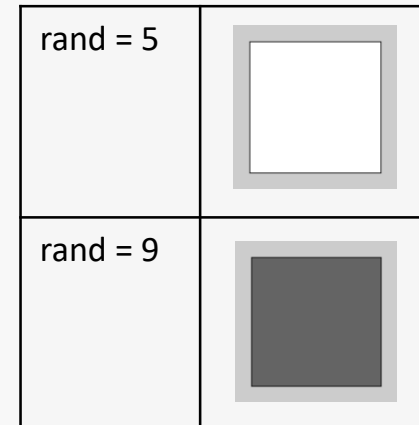
- Vergleichsoperatoren

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

- Beispiel

Funktion random liefert Zufallszahl

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



Logische Werte

- Wertebereich umfasst 2 Werte

- true und false

- Operationen




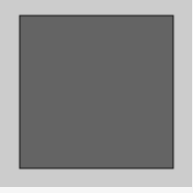

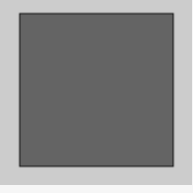


- Negation: !
 - Oder: ||
 - Und: &&
 - XOR: ^

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);
```

<p>a = 1 b = 3</p> 	<p>a = 2 b = 8</p> 
<p>a = 7 b = 3</p> 	<p>a = 8 b = 9</p> 
<p>a = 1 b = 3</p> 	<p>a = 2 b = 8</p> 
<p>a = 7 b = 3</p> 	<p>a = 8 b = 9</p> 

Komplexere Formen der Verzweigung (1)

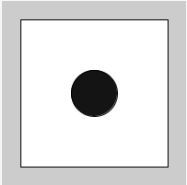
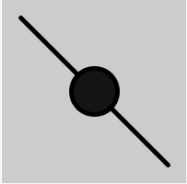
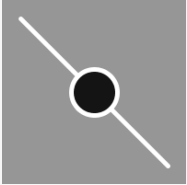
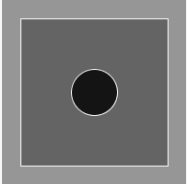
- Mit else-Zweig

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

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

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	

Komplexere Formen der Verzweigung (2)

- Mehrere Alternativen

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

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: Eine Anweisung wird oft mit kleinen Änderungen wiederholt

Lösung? – Schleifen!

for-Schleife

- Aufbau

- `init`

- Initialisierung vor dem Start der Schleife
 - Z. B. Variable für Schleife vereinbaren und initialisieren

- `test`

- Abbruchtest für Beenden der Schleife

- `update`

- Veränderung von Schleifenvariablen
 - Wird nach den Anweisungen ausgeführt

- Statements

- Ein oder mehrere Anweisungen
 - Bei einer einzigen Anweisung können die Klammern { } weggelassen werden

```
for (init; test; update) {  
    statements  
}
```

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

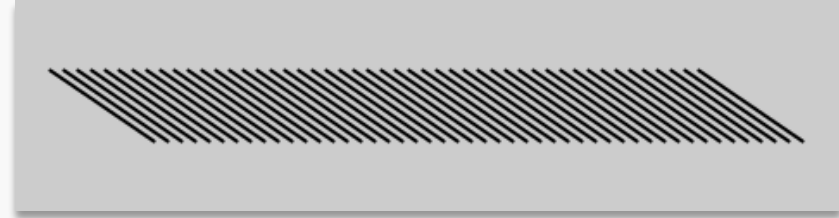
- Ausgangsbeispiel mit Schleife

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

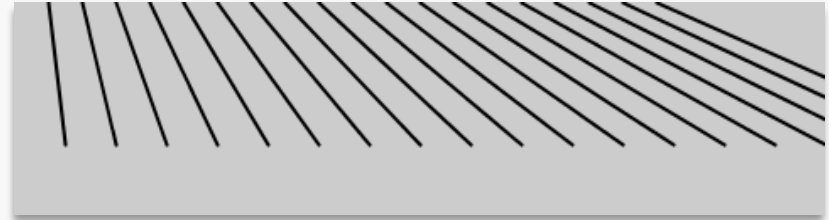


Weitere Beispiele

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

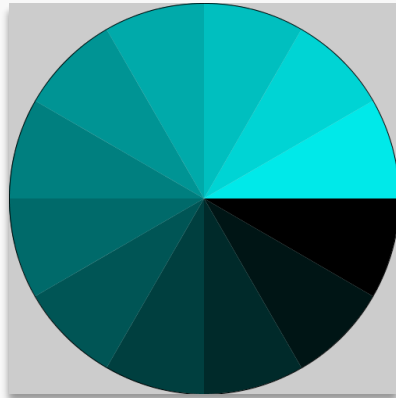


```
size(480, 120);  
strokeWeight(2);  
for (int x = 20; x < 400; x += 20) {  
  line(x, 0, x + x / 2, 80);  
}
```



Weiteres Beispiele

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



Komplexeres Beispiel 1 – Beschreibung

- Optische Täuschung
 - Beispiel siehe rechts
- Vorgehensweise
 - Hintergrund
 - Alle Graustufen von links nach rechts durchlaufen
 - 256 Graustufen
 - Breite des Fensters sollte ein Vielfaches der Anzahl der Graustufen sein
 - Höhe beliebig
 - Hier Anzahl der Graustufen



Komplexeres Beispiel 1 – Hintergrund

- Wir legen fest

- Anzahl der Graustufen (shades)
- Faktor für Vielfaches (factor)
- Größe des Sketchfensters
 - $\text{shades} * \text{factor} \times \text{shades}$

- Zeichnen

- Ein Rechteck mit einer Breite von factor Pixel
 - Mit entsprechender Graustufe gefüllt
 - Keine Umrandungslinien (noStroke())

- Ergebnis

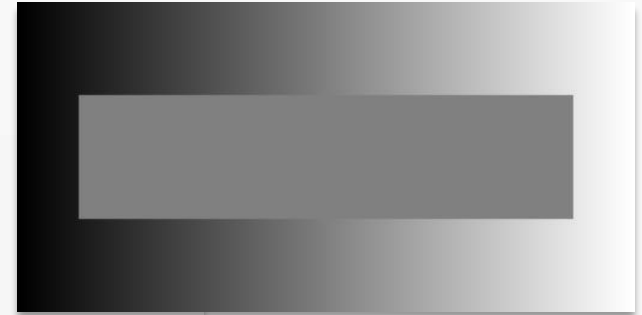
```
int shades = 256;
int factor = 2;
size(512, 256);
noStroke();
for (int i = 0; i < shades; i++) {
  fill(i);
  rect(i * factor, 0, factor, height);
}
```



Komplexeres Beispiel 1 – Abschluss

- Einfarbiger Balken über den Hintergrund
 - Mit Hilfe von width und height flexibel realisieren
 - Mittlere Graustufe
- 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

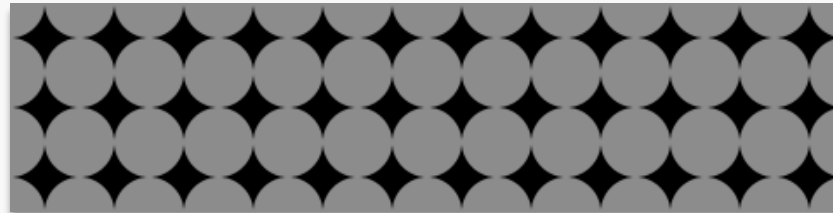
- Beliebige große Zeichenfläche
 - Von links nach rechts pro X-Koordinate eine vertikale Linie
 - Graustufe ergibt sich aus der X-Koordinate der Linie
 - width Punkte und 256 Graustufen
 - Aktuelle X-Koordinate auf den Bereich 0 – 255 abbilden (map)
- Ergebnis

```
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 verschachtelt werden
- 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);  
    }  
}
```



Verschachtelte Schleifen – Ablauf

- Beispiel

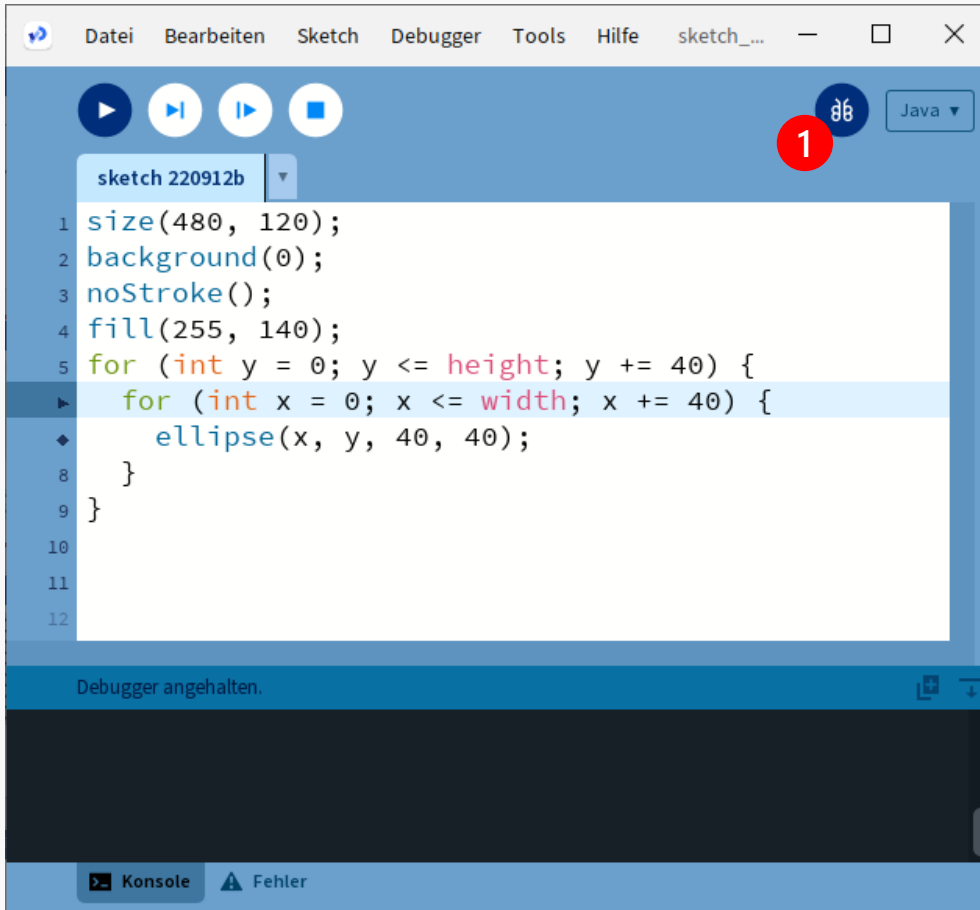
```
size(480, 120);  
background(0);  
noStroke();  
fill(255, 140);  
for (int y = 0; y <= height; y += 40) {  
    for (int x = 0; x <= width; x += 40) {  
        ellipse(x, y, 40, 40);  
    }  
}
```

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



The screenshot shows the 'Variables' window in the Processing IDE. It displays a list of variables and their current values. A red circle with the number '2' highlights the 'Variables' window.

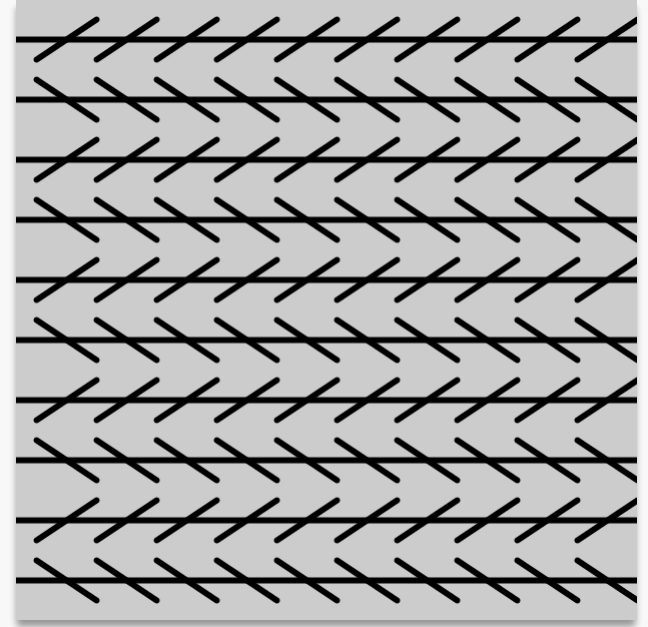
Name	Value
y	40
x	240
Processing	
width	480
height	120
mouseX	0
mouseY	0
pmouseX	0
pmouseY	0
key	
keyCode	0
keyPressed	false
focused	false
frameRate	60.0
frameCount	0

1 Debugger einschalten

2 Variablen inspizieren

Komplexeres Beispiel 2 – Beschreibung

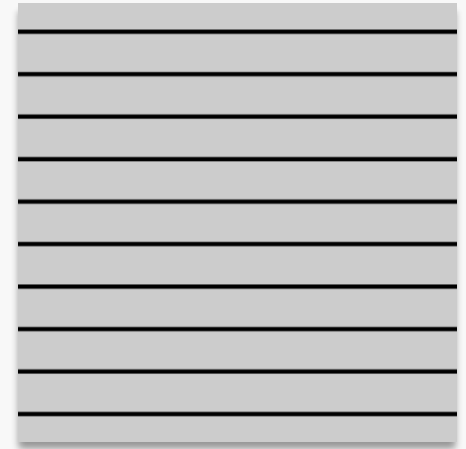
- Optische Täuschung
 - Beispiel siehe rechts
- Vorgehensweise
 - Schleife für horizontale Linien
 - Schleife für kürzere Linien
 - Abwechselnd nach links oder nach rechts geneigt



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

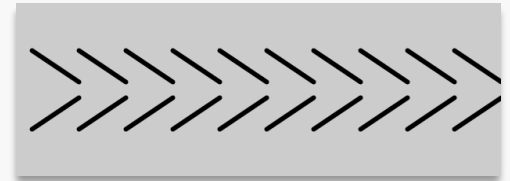
```
size(620, 620);  
strokeWeight(6);  
for (int y = 40; y < height; y += 60) {  
    line(0, y, width - 1, y);  
}
```



Komplexeres Beispiel 2 – Kürzere Linien

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

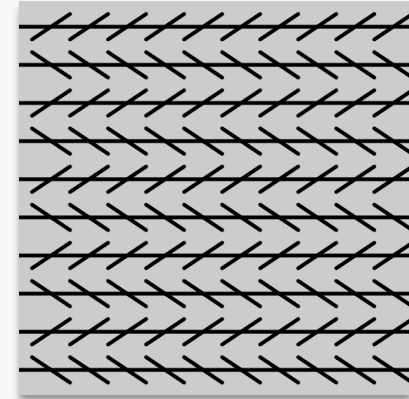
```
size(620, 220);
strokeWeight(6);
int increment = 60;
int step = 40;
for (int x = 20; x < width; x += increment) {
    line(x, 60, x + increment, 60 + step);
    line(x + increment, 120, x, 120 + step);
}
```



Komplexeres Beispiel 2 – Abschluss

- Schleifen verschachteln
 - Abwechselnd kürzere Linien nach links oder nach rechts zeichnen
- Ergebnis

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



Komplexeres Beispiel 2 – Alternative

- Adaptive Implementierung

```
size(620, 620);
strokeWeight(6);
int numberOfLines = 10;
int increment = height/numberOfLines;
int ystart = increment/2;
int xstart = increment/6;
int distance = increment/3;
for (int y = ystart; y < height; y += increment) {
    line(0, y, width - 1, y);
    for (int x = xstart; x < width; x += increment) {
        if (round(y/increment) % 2 == 1) {
            line(x, y - distance, x + increment, y + distance);
        } else {
            line(x + increment, y - distance, x, y + distance);
        }
    }
}
```