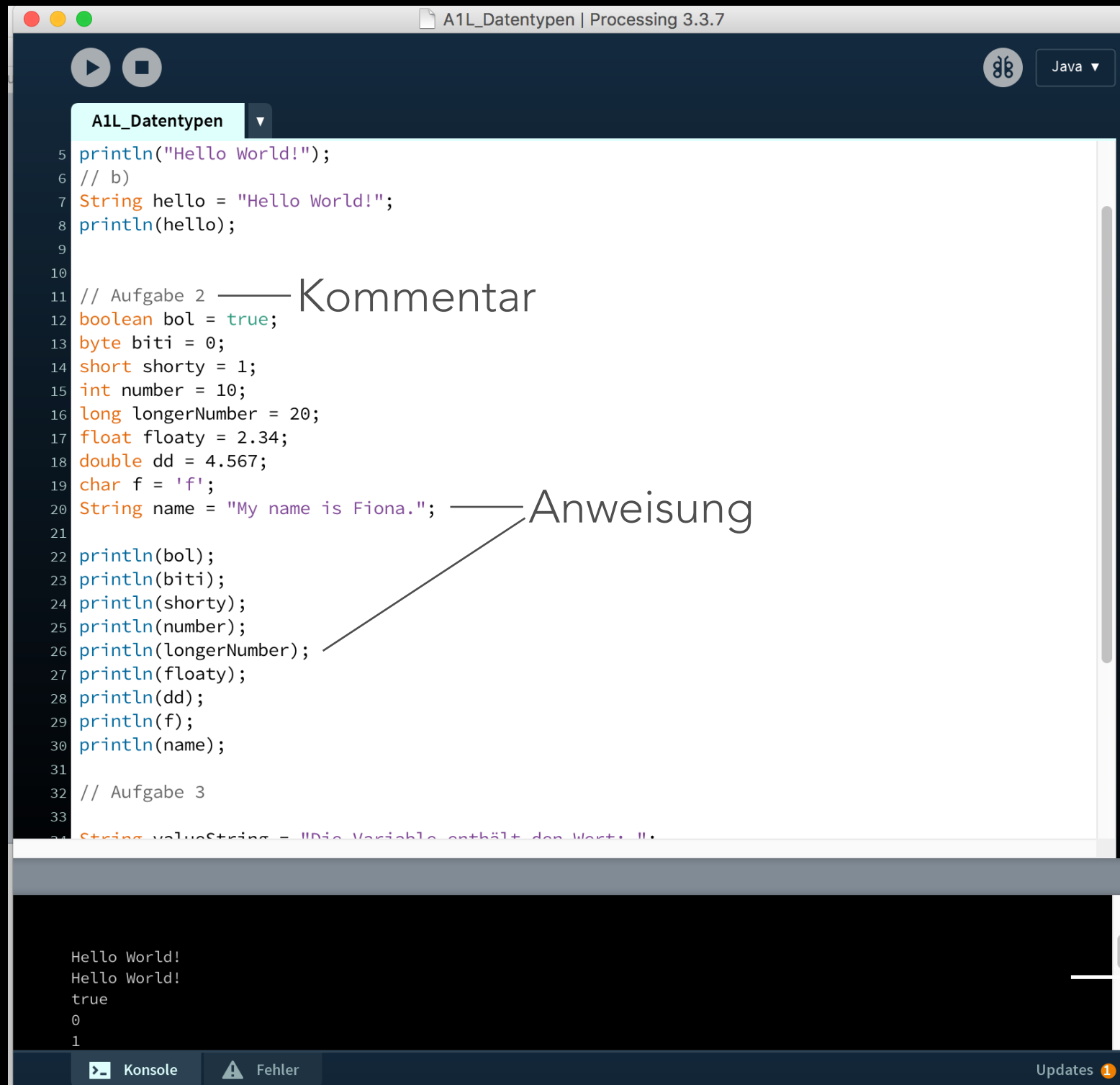


BRÜCKENKURS PROGRAMMIEREN - FIONA NÜESCH

# REKAPITULATION

# Processing IDE



The screenshot shows the Processing IDE interface. The title bar indicates the file is 'A1L\_Datentypen' and the version is 'Processing 3.3.7'. The main editor area contains the following code:

```
5 println("Hello World!");  
6 // b)  
7 String hello = "Hello World!";  
8 println(hello);  
9  
10  
11 // Aufgabe 2 ——— Kommentar  
12 boolean bol = true;  
13 byte biti = 0;  
14 short shorty = 1;  
15 int number = 10;  
16 long longerNumber = 20;  
17 float floaty = 2.34;  
18 double dd = 4.567;  
19 char f = 'f';  
20 String name = "My name is Fiona."; ——— Anweisung  
21  
22 println(bol);  
23 println(biti);  
24 println(shorty);  
25 println(number);  
26 println(longerNumber);  
27 println(floaty);  
28 println(dd);  
29 println(f);  
30 println(name);  
31  
32 // Aufgabe 3  
33 String valueString = "Die Variable enthält den Wert: ";
```

The console output at the bottom shows the results of the program execution:

```
Hello World!  
Hello World!  
true  
0  
1
```

Pro Anweisung eine Zeile

Eine Anweisung wird mit einem Semicolon abgeschlossen

Konsole

- Variable = Gefäß um Daten zu speichern
- kann in Java nur Daten von einem definiertem Wert aufnehmen

```
int variableName;           // Deklaration. Es wird deklariert welchen
                             // Typ und Namen unser Speichergefäß hat.
                             //...

variableName = 2;           // später im Code: es kann immer wieder
println(variableName);      // auf die Variable zugegriffen werden.
```

// primitive Datentypen		// Referenztyp
byte	<u>b</u> ;	String <u>str</u> ;
short	<u>s</u> ;	
int	<u>i</u> ;	
long	<u>l</u> ;	
	float <u>f</u> ;	
	double <u>d</u> ;	
	char <u>c</u> ;	
	boolean <u>bol</u> ;	

Mit **Operatoren** können Zuweisungen und Berechnungen vorgenommen und Bedingungen formuliert und geprüft werden.

```
int c = a + b;
```

```
int c = a - b;
```

```
int c = a * b;
```

```
int c = a / b;
```

```
int c = a % b;
```

```
a++;
```

```
b--;
```

```
a < b
```

```
a <= b
```

```
a > b
```

```
a >= b
```

```
a == b
```

```
a != b
```

```
!
```

```
&
```

```
^
```

```
|
```

```
&&
```

```
||
```

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

# Array = Sammlung von Elementen des gleichen Datentyps

(Kann sich als Sammlung von Variablen desselben Datentyps vorgestellt werden)

```
int[] arrayName;           // Array Deklaration. Typ[] und Name

arrayName = new int[10];    // Entweder: Erzeugung mit Angabe wie viele Elemente
                             // die Sammlung enthalten soll
arrayName = {1, 2, 6, 4, 8}; // Oder: Erzeugung direkt mit Angabe der enthaltenen Elementen
```

```
int index = 1;

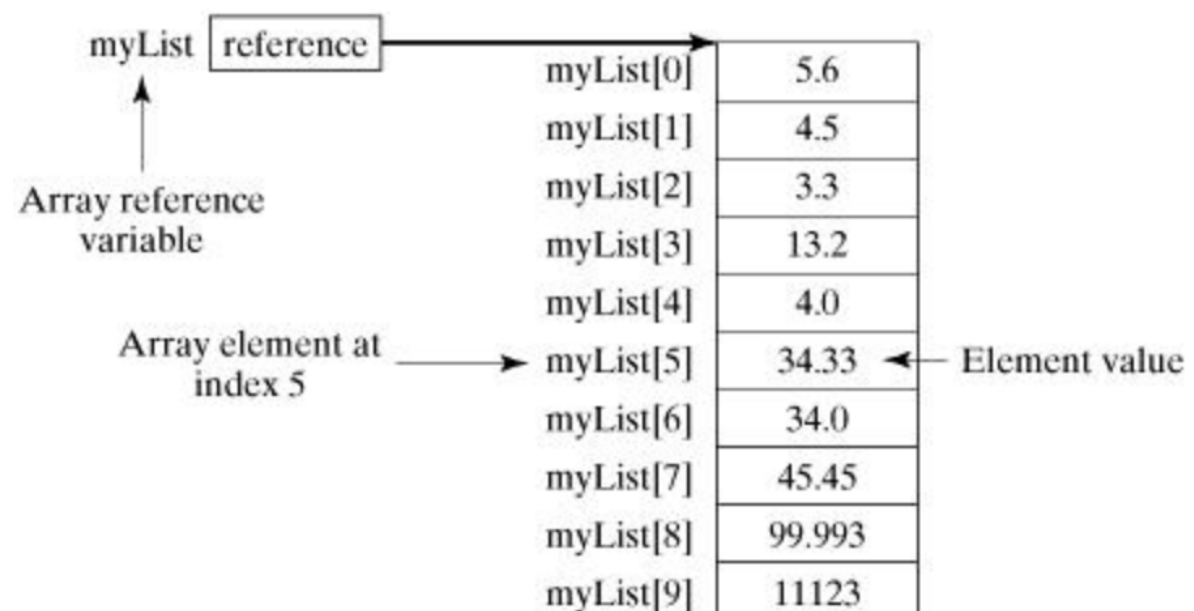
// Array Elementzugriff:

int element = arrayName[index];
arrayName[index] = 2;
```

**MERKE:** Das erste Element erhält man mit `index = 0`.

```
double[] myList = new double[10];
```

Following picture represents array `myList`. Here, `myList` holds ten double values and the indices are from 0 to 9.



# ARRAYS - LENGTH

- möchte man auf die Anzahl der Element in einem Array zugreifen macht man das mit:

`array.length`

```
int arraySize = arrayName.length; // Die Anzahl der Elemente im Array
```

# KONTROLLSTRUKTUREN

- BLOCK { }

- { } fasst mehrere Anweisungen zu einem Block zusammen
- Variablen innerhalb eines Blocks müssen einzigartig benannt sein
- Variablen sind nur innerhalb eines Blocks sichtbar

```
{  
    int i = 1;  
  
    {  
        println(i);  
    }  
}  
  
println(i);
```

The variable "i" does not exist

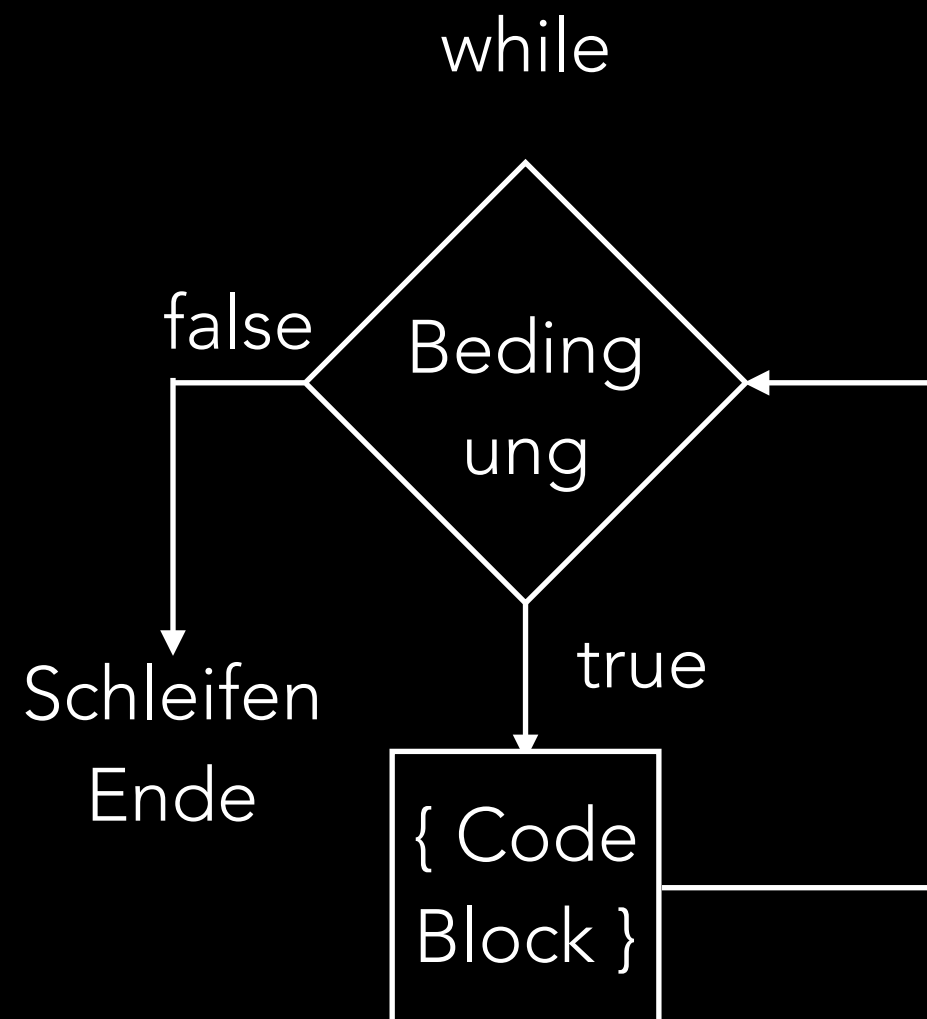
```
{  
    int variable = 1;  
    String variable = "hello";  
}
```

Duplicate local variable variable



# KONTROLLSTRUKTUREN - SCHLEIFEN

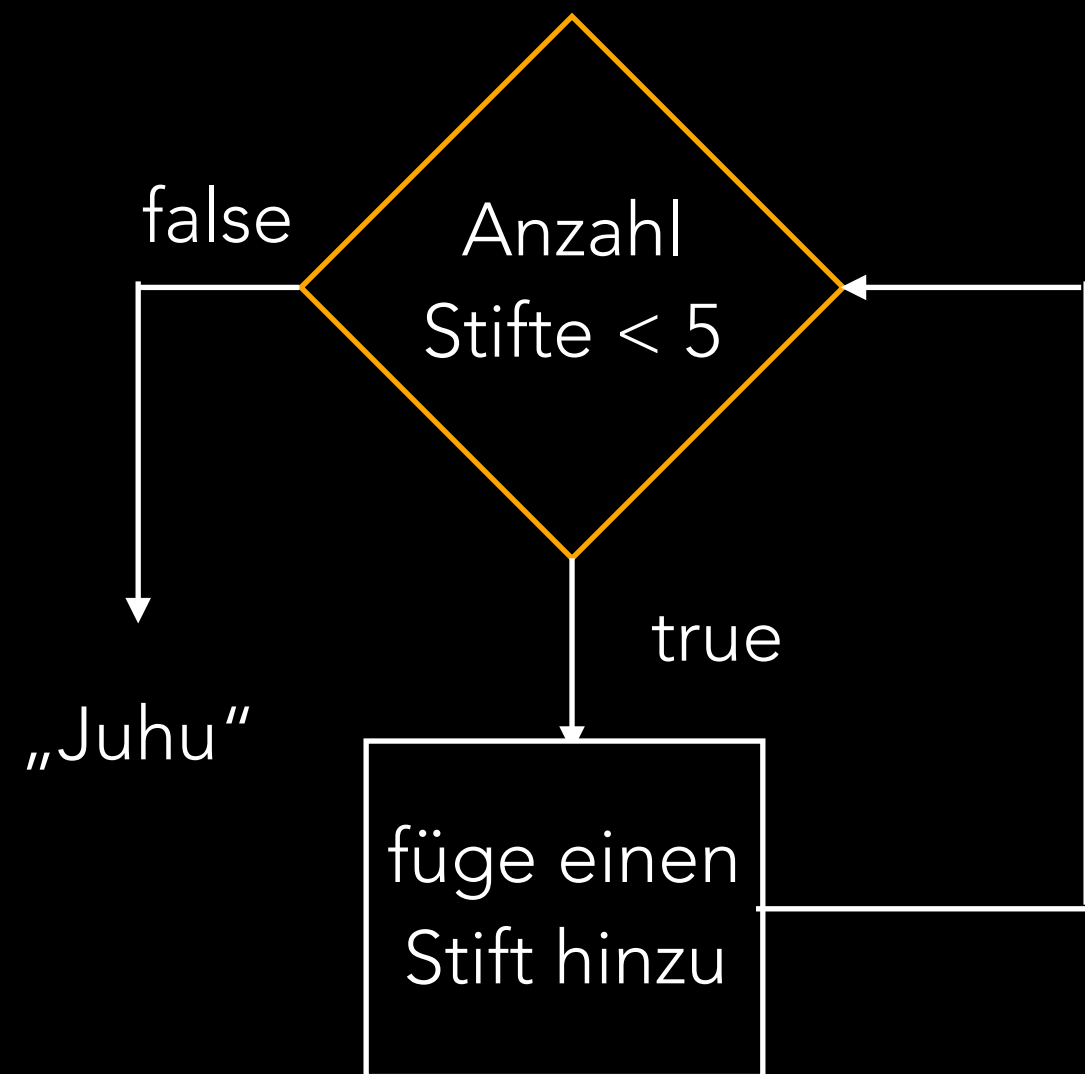
- Schleifen führen Anweisungen wiederholt aus, solange eine Bedingung erfüllt ist
- Es gibt **while**-Schleifen und **for**-Schleifen



# Die **while**-Schleife

Pseudocode:

```
while ( Anzahl Stifte < 5 ) {  
    füge einen Stift hinzu;  
}  
  
rufe „Juhu“;
```



# Die **while**-Schleife

```
boolean abbruchsbedingung;  
  
while ( abbruchsbedingung ){  
    // Anweisungs Block  
}  
  
do {  
    // Anweisungs Block  
} while ( abbruchsbedingung );
```

```
int max = 10;  
int sum = 10;  
  
while ( sum < max ){  
    sum = sum + 2;  
}  
println(sum);  
  
sum = 10;  
  
do{  
    sum = sum + 2;  
}while ( sum < max );  
  
println(sum);
```

# Die **while**-Schleife

```
boolean abbruchsbedingung;  
  
while ( abbruchsbedingung ){  
    // Anweisungs Block  
}  
  
do {  
    // Anweisungs Block  
} while ( abbruchsbedingung );
```

```
int max = 10;  
int sum = 10;  
  
while ( sum < max ){  
    sum = sum + 2;  
}  
println(sum);    10  
  
sum = 10;  
  
do{  
    sum = sum + 2;  
}while ( sum < max );  
  
println(sum);
```

# Die **while**-Schleife

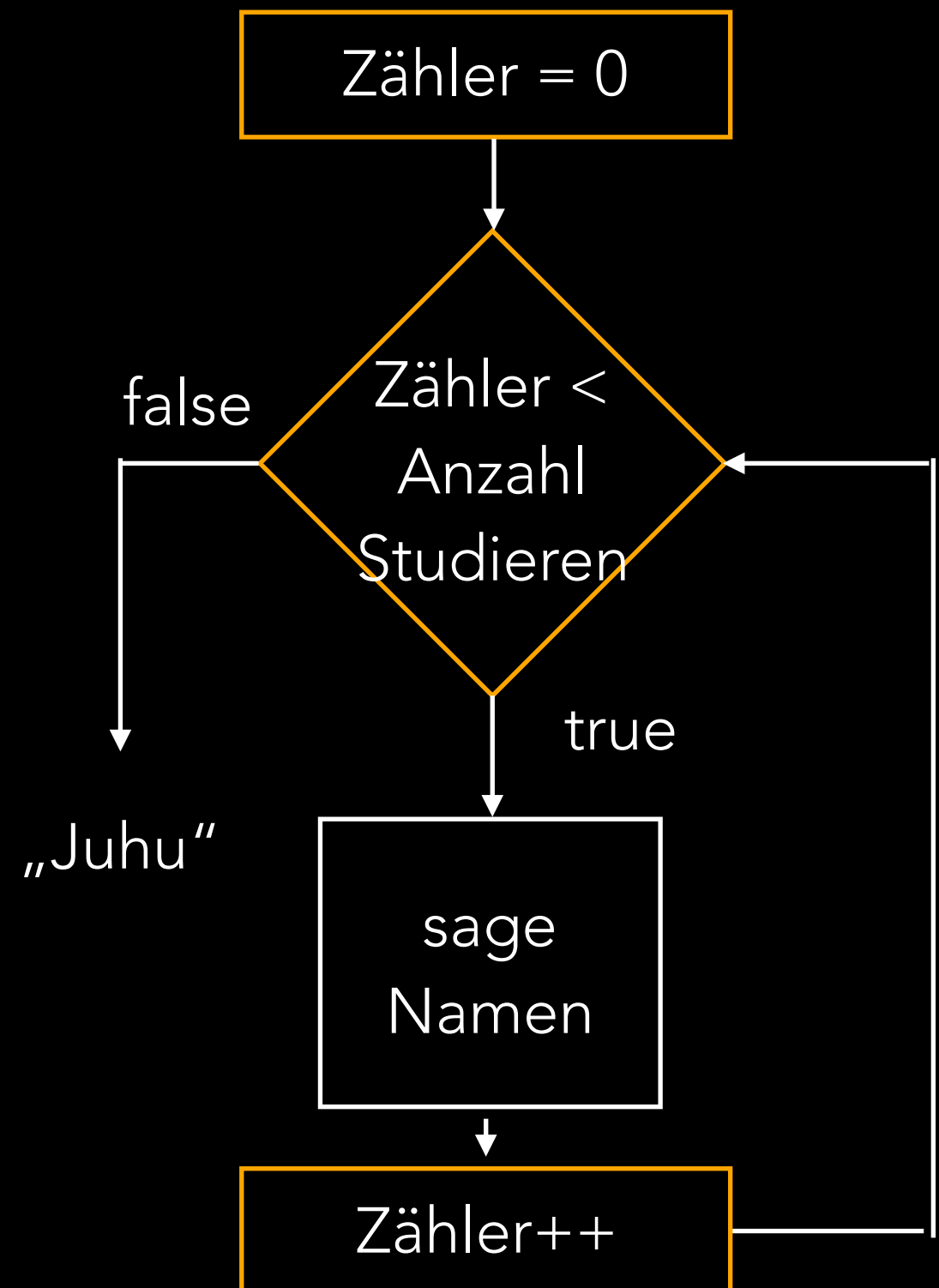
```
boolean abbruchsbedingung;  
  
while ( abbruchsbedingung ){  
    // Anweisungs Block  
}  
  
do {  
    // Anweisungs Block  
} while ( abbruchsbedingung );
```

```
int max = 10;  
int sum = 10;  
  
while ( sum < max ){  
    sum = sum + 2;  
}  
println(sum);    10  
  
sum = 10;  
  
do{  
    sum = sum + 2;  
}while ( sum < max );  
  
println(sum);    12
```

# Die **for**-Schleife

Pseudocode:

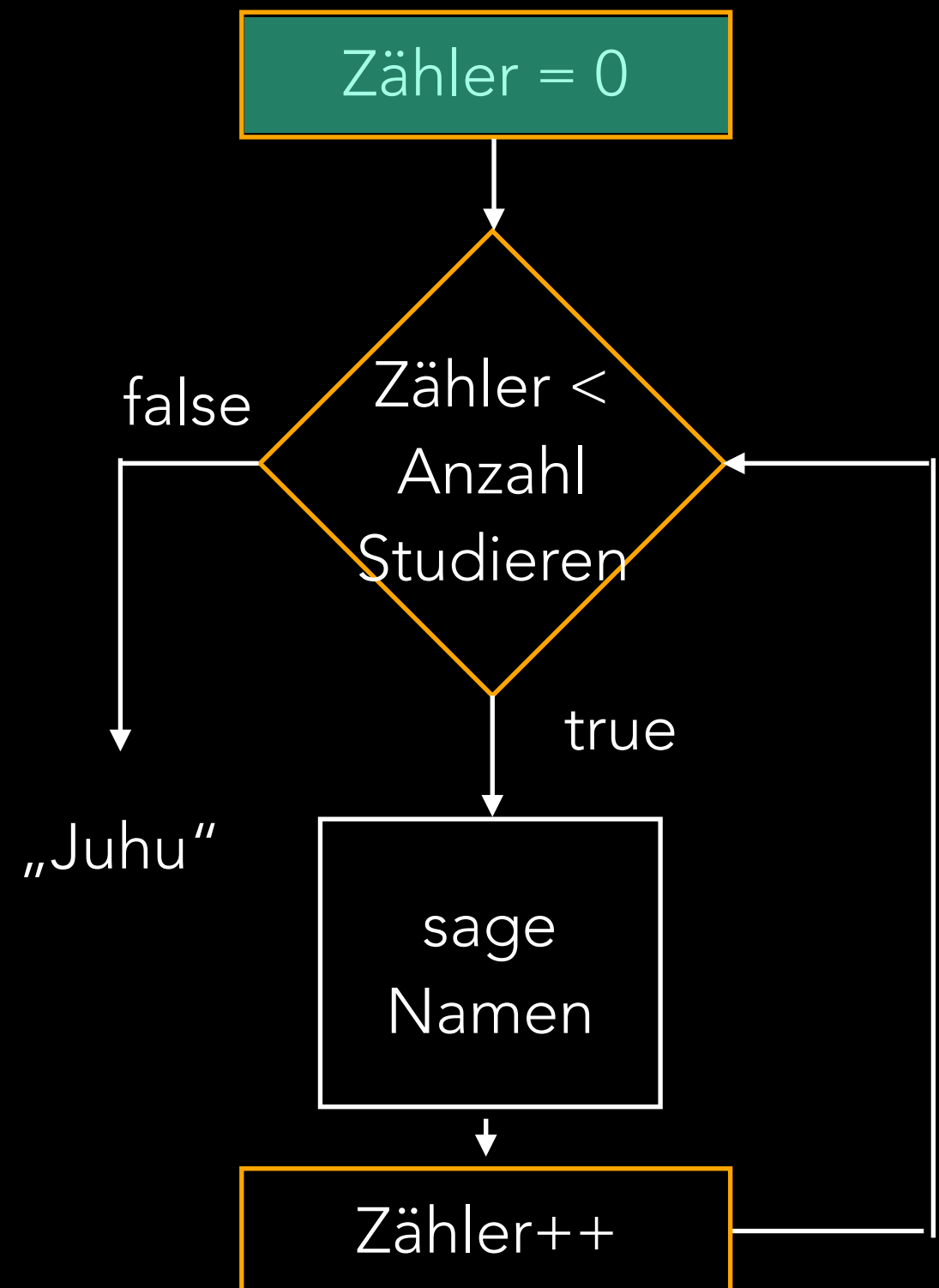
```
for ( Zähler = 0; Zähler < Anzahl  
Studierende; Zähler++ ) {  
  
sage den Namen vom  
Studierenden an Position = Zähler;  
  
}  
  
rufe „Juhu“;
```



# Die **for**-Schleife

Pseudocode:

```
for (Zähler = 0; Zähler < Anzahl  
Studierende; Zähler++) {  
  
sage den Namen vom  
Studierenden an Position = Zähler;  
  
}  
  
rufe „Juhu“;
```





# Die **for**-Schleife

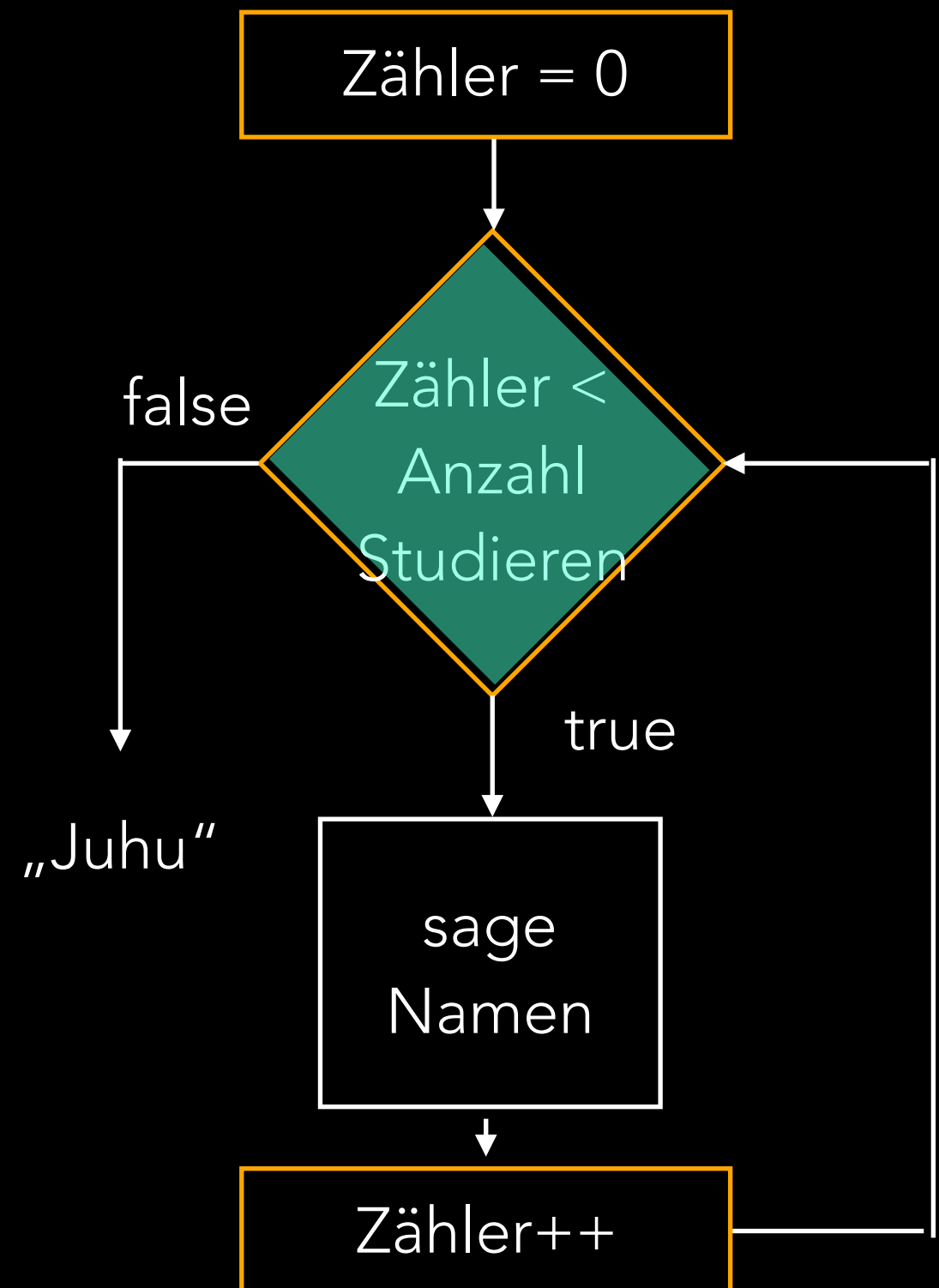
Pseudocode:

```
for ( Zähler = 0; Zähler < Anzahl  
Studierende; Zähler++ ) {
```

```
  sage den Namen vom  
  Studierenden an Position = Zähler;
```

```
}
```

```
rufe „Juhu“;
```



# Die **for**-Schleife

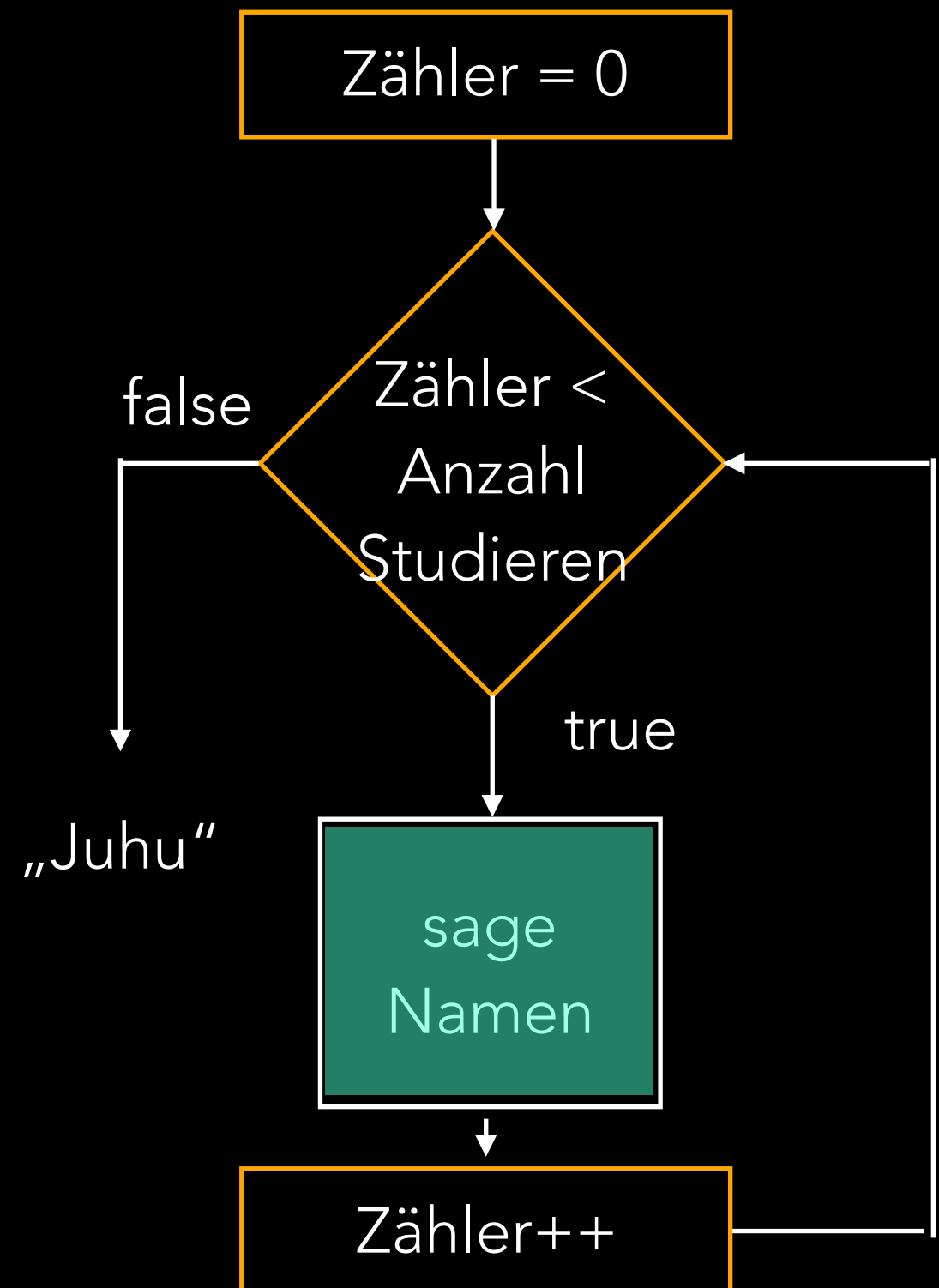
Pseudocode:

```
for ( Zähler = 0; Zähler < Anzahl  
Studierende; Zähler++ ) {
```

```
sage den Namen vom  
Studierenden an Position = Zähler;
```

```
}
```

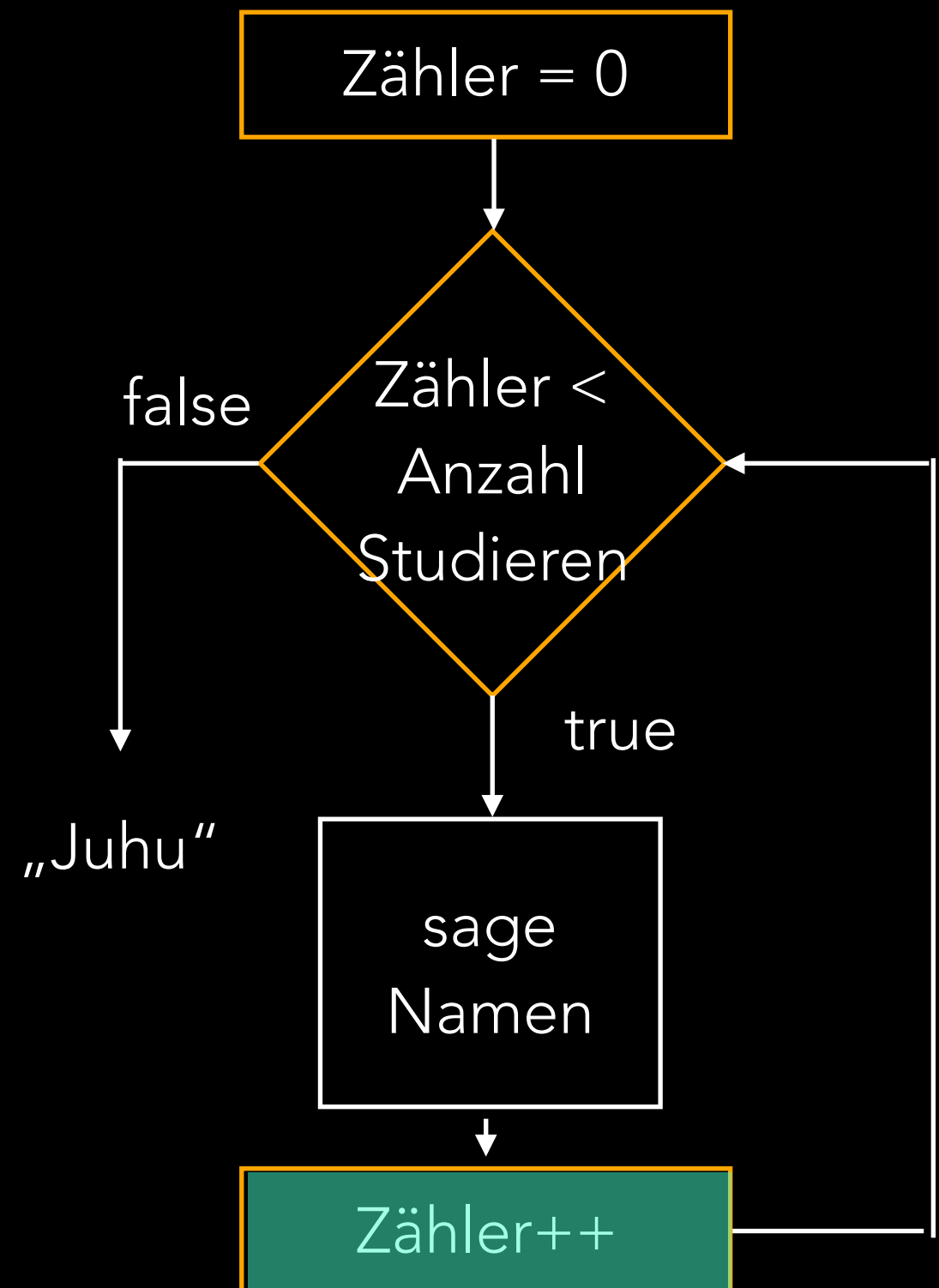
```
rufe „Juhu“;
```



# Die **for**-Schleife

Pseudocode:

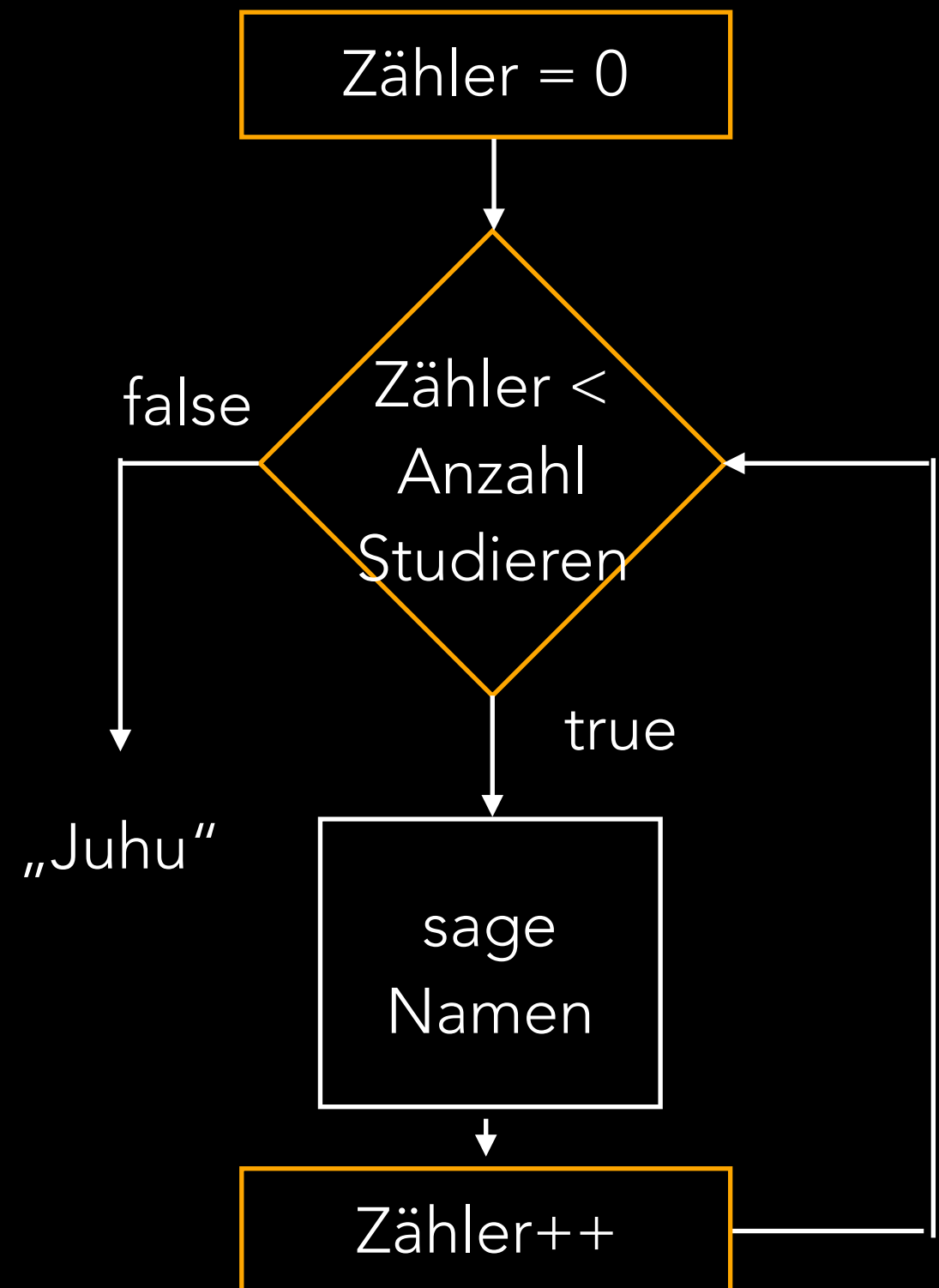
```
for ( Zähler = 0; Zähler < Anzahl  
Studierende; Zähler++ ) {  
  
sage den Namen vom  
Studierenden an Position = Zähler;  
  
}  
  
rufe „Juhu“;
```



# Die **for**-Schleife

Pseudocode:

```
for ( Zähler = 0; Zähler < Anzahl  
Studierende; Zähler++ ) {  
  
sage den Namen vom  
Studierenden an Position = Zähler;  
  
}  
  
rufe „Juhu“;
```



# Die **for**-Schleife

```
int max = 10;

for( int i = 0; i < max; i++){
    // Anweisungs Block
}
```

# Die **for**-Schleife

```
int max = 10;

for( int i = 0; i < max; i++){
    // Anweisungs Block
}
```

Initialisierung der Zähler Variable vom Typ int

# Die **for**-Schleife

```
int max = 10;  
  
for( int i = 0; i < max; i++ ){  
    // Anweisungs Block  
}
```

Abbruchsbedingung

Initialisierung der Zähler Variable vom Typ int

# Die **for**-Schleife

```
int max = 10;  
  
for( int i = 0; i < max; i++){  
    // Anweisungs Block  
}
```

Update der Zähler Variable

Abbruchsbedingung

Initialisierung der Zähler Variable vom Typ int



# while

vs.

# for

```
int count = 0;
while (count < 100) {
    //do stuff
    count++;
}
```

```
for (int x = 0; x < 100; x++) {
    //executed until x >= 100
}
```

findet Verwendung  
wenn die Anzahl  
Durchläufe nicht klar ist

wird verwendet wenn  
die Anzahl Durchläufe  
bekannt ist.

## while

findet Verwendung  
wenn die Anzahl  
Durchläufe nicht klar ist

```
int number = getNumberFromUserInput();  
  
while (number > 1) {  
    number = number / 2;  
}
```

```
while(gameIsRunning){  
    // do some Animation  
}
```

```
while(true){  
    // endlose Wiederholungen  
}
```

vs.

## for

wird verwendet wenn  
die Anzahl Durchläufe  
bekannt ist.

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

## Übung Schleifen

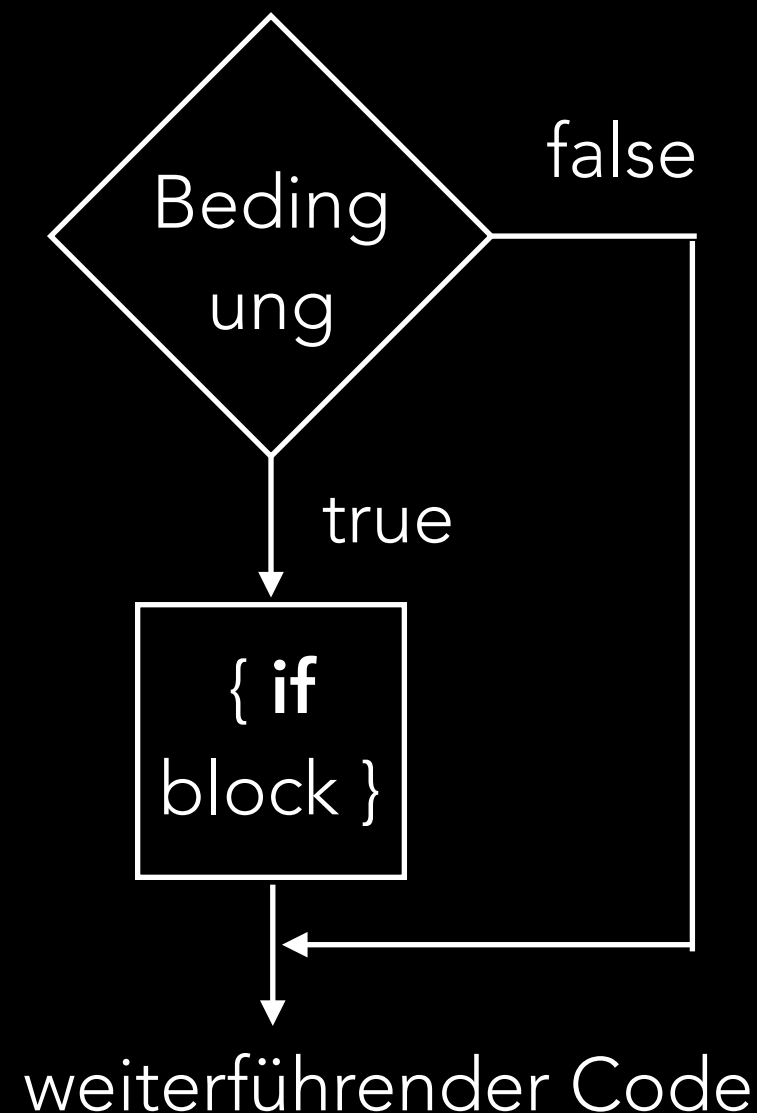
„ Die ersten 10'000 Aufnahmen sind die schlechtesten. “

– HELMUT NEWTON

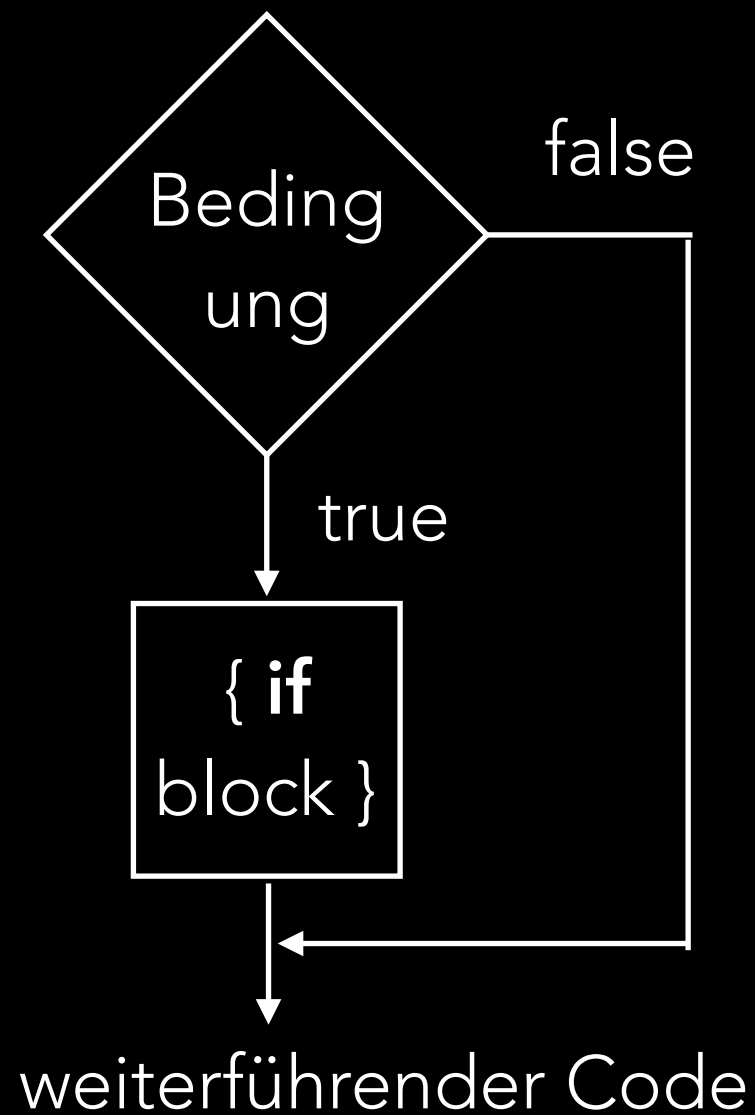
# KONTROLLSTRUKTUREN

- IF / ELSE (VERZWEIGUNG)

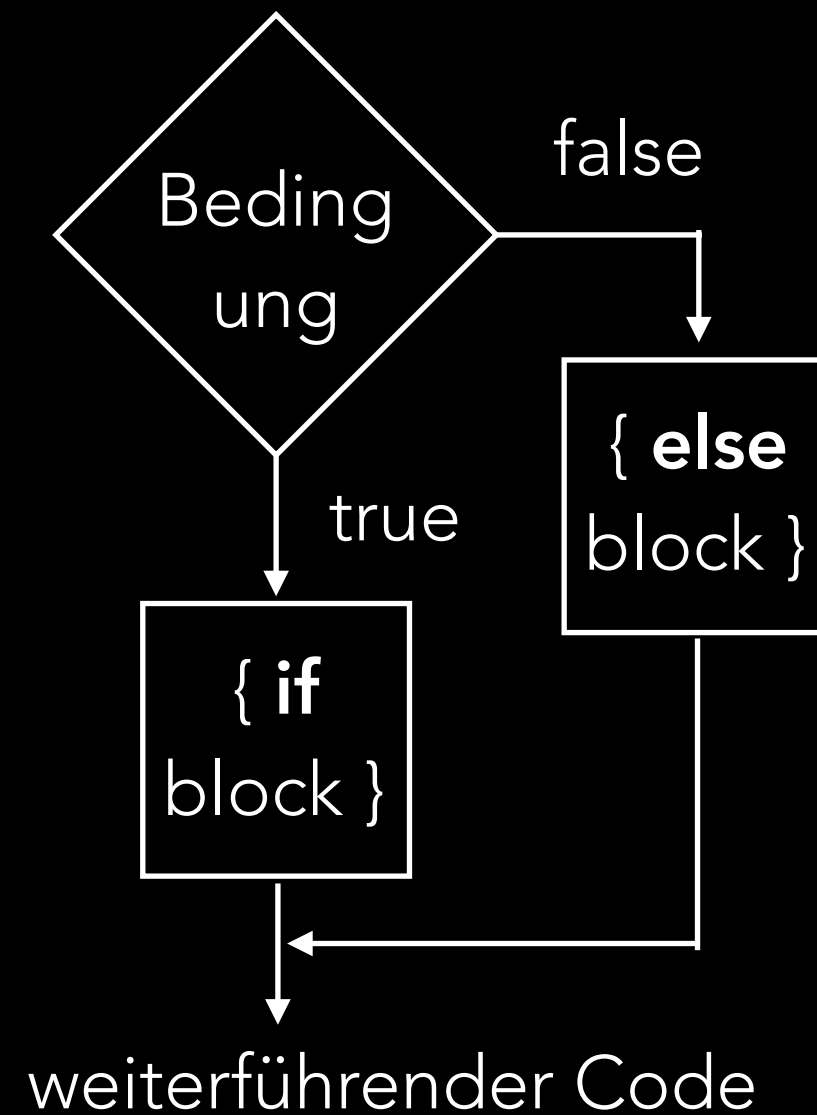
- Verzweigungen erlauben einen Code Block nur auszuführen wenn eine Bedingung erfüllt ist



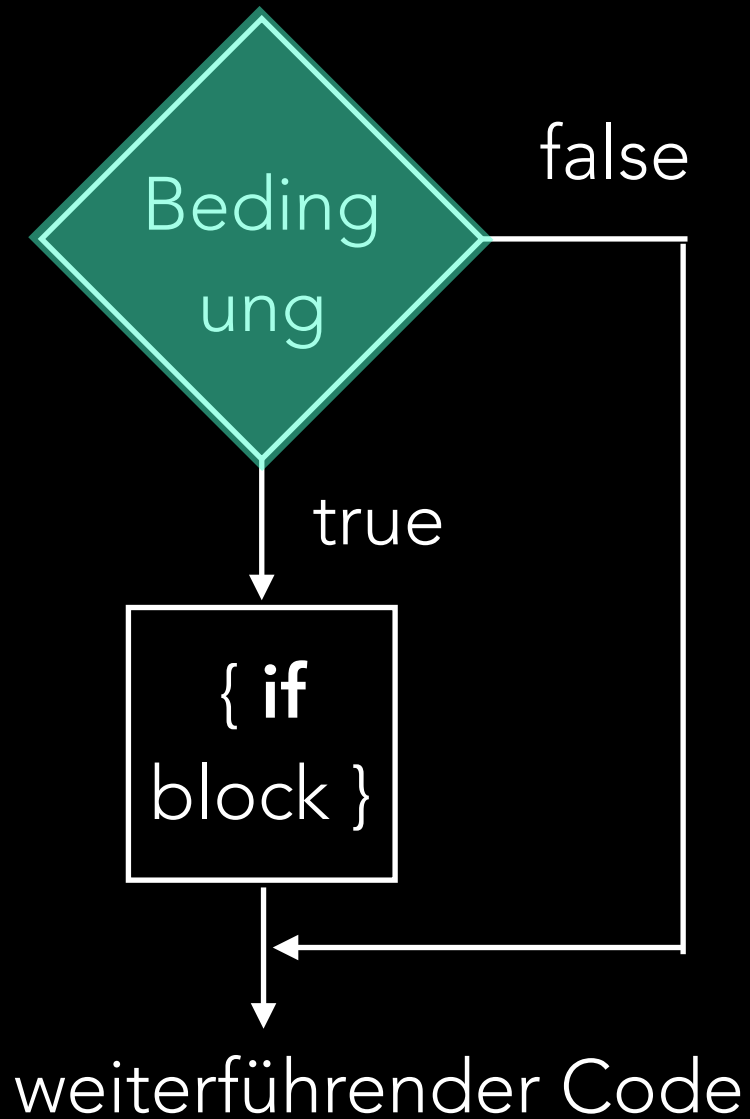
```
if( a < b ){
    // if code block
}
```



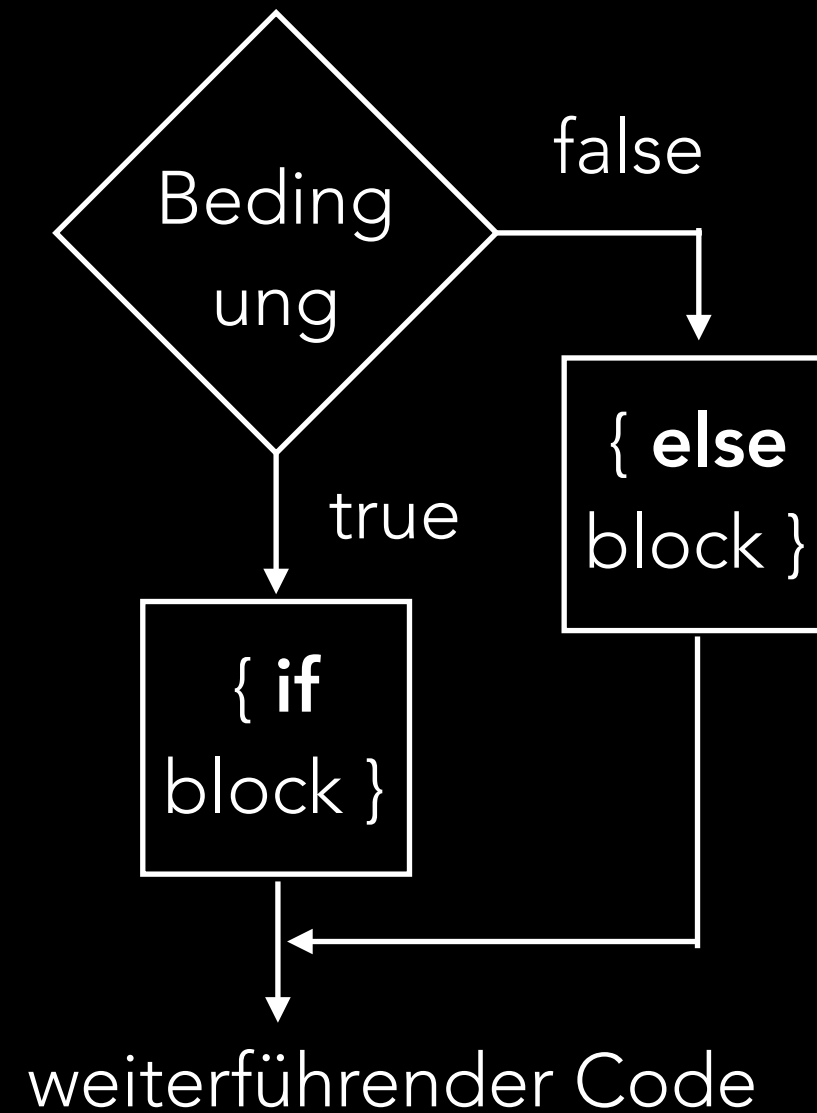
```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



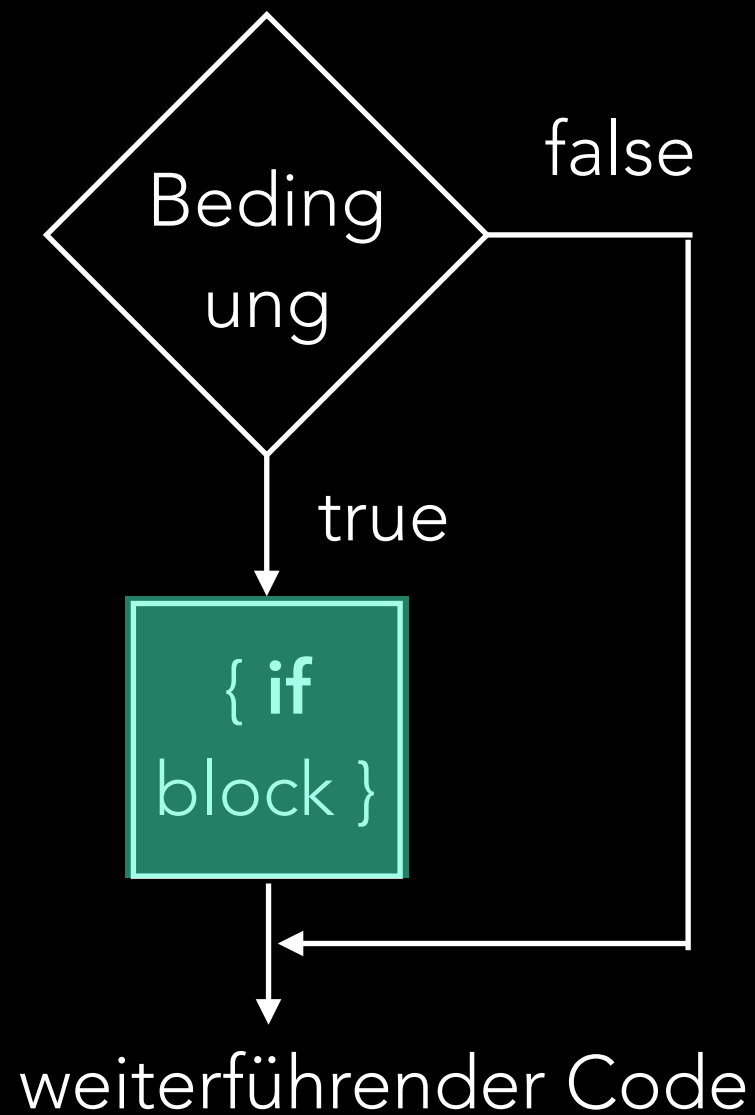
```
if( a < b ){
    // if code block
}
```



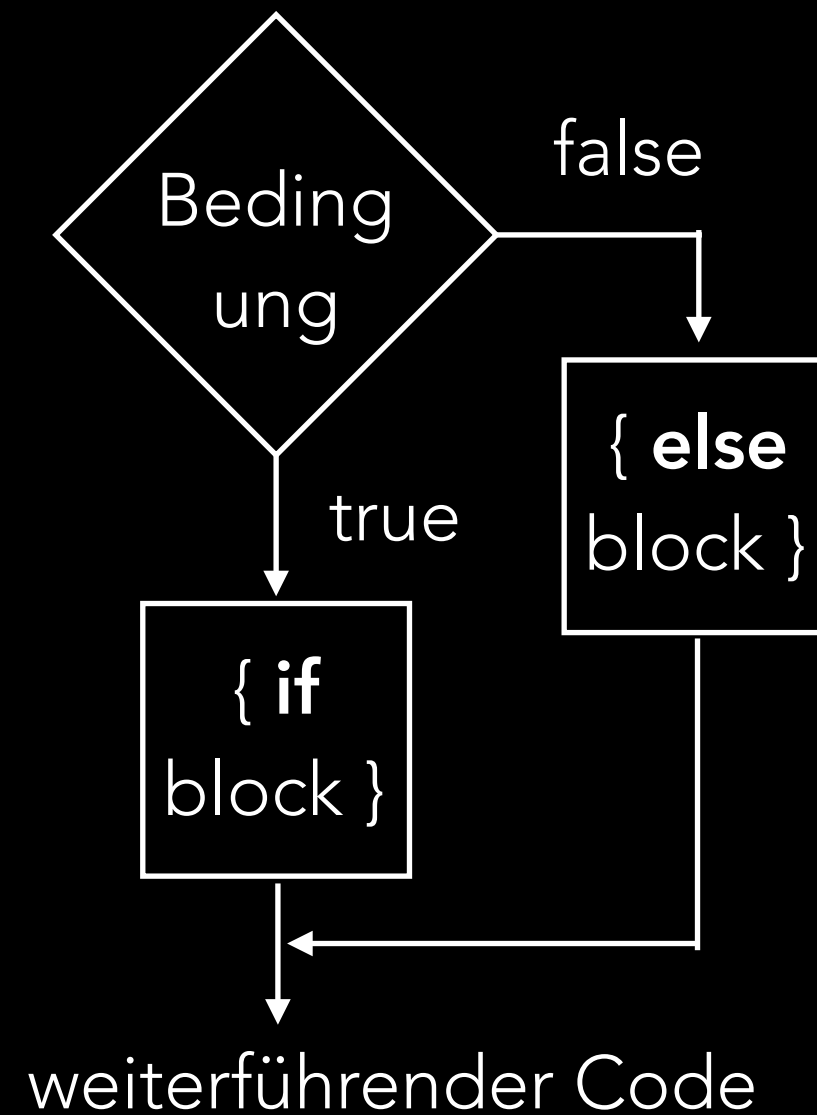
```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



```
if( a < b ){
    // if code block
}
```

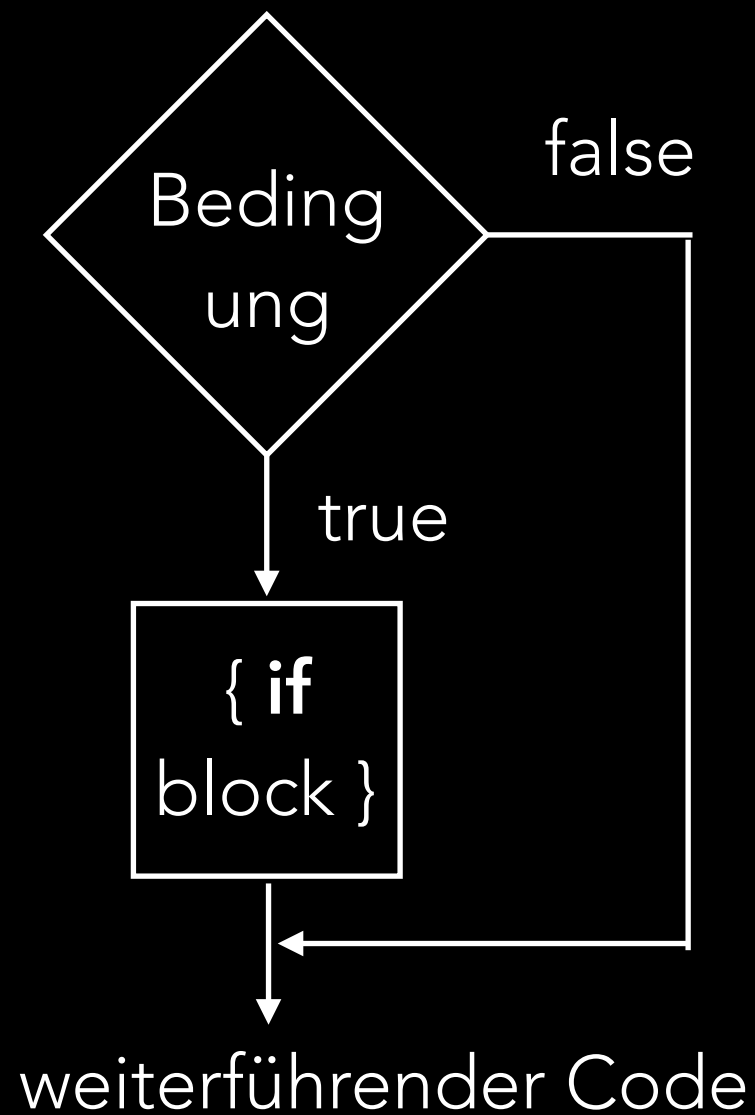


```
if( a < b ){
    // if code block
}else{
    // else code block
}
```

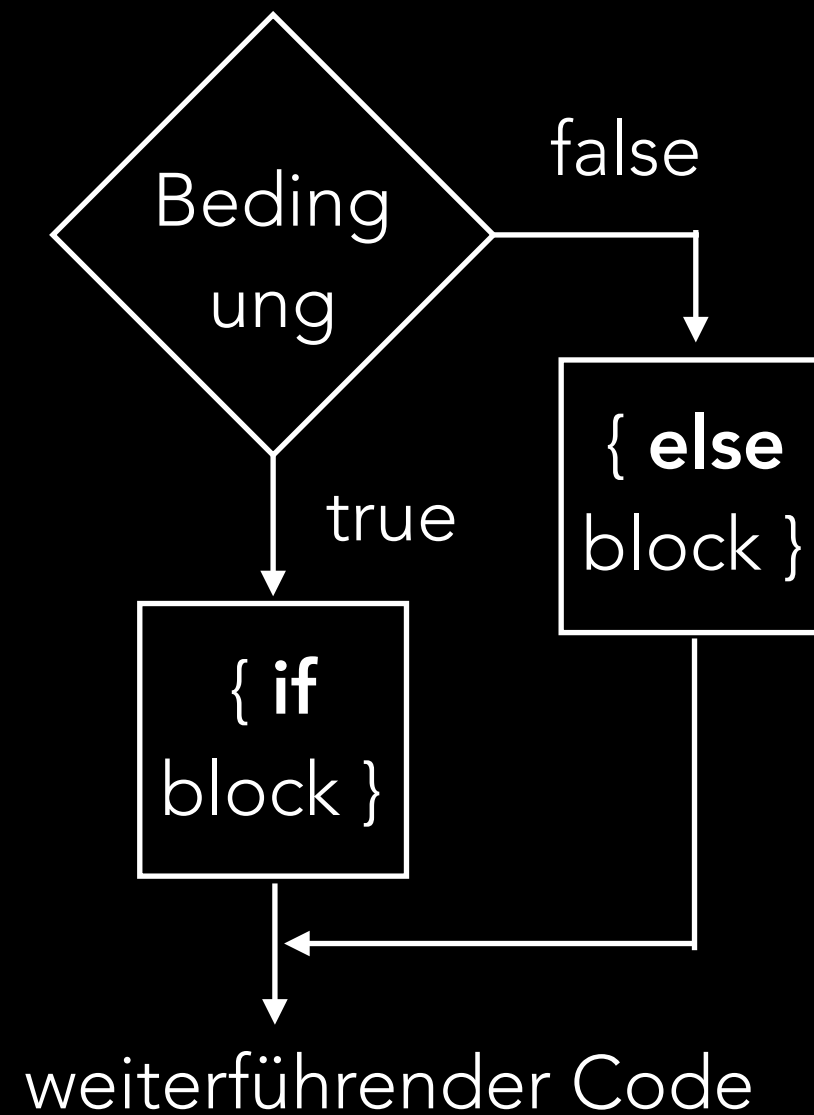




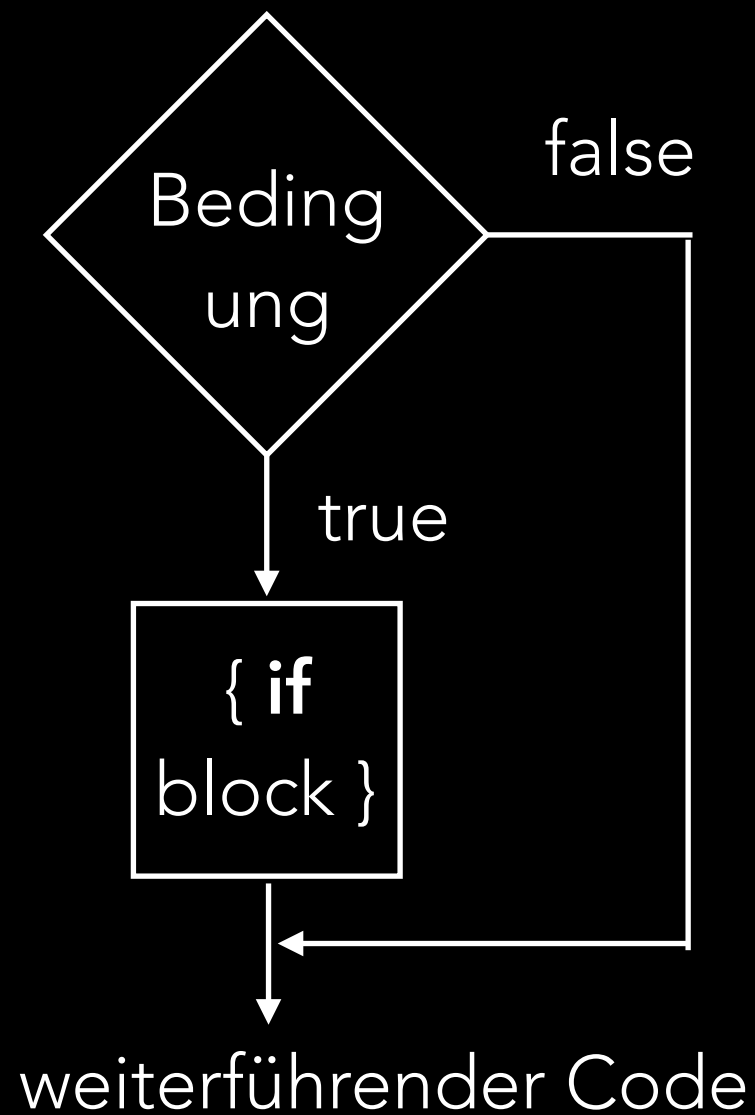
```
if( a < b ){
    // if code block
}
```



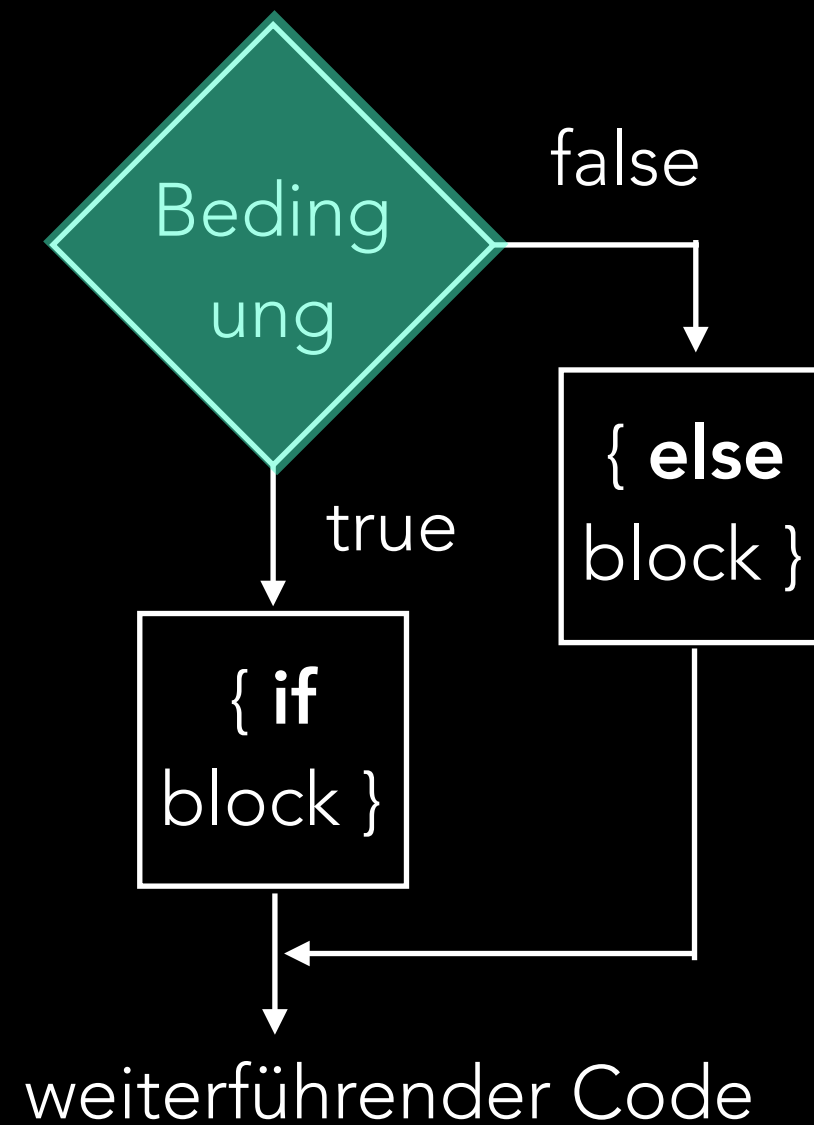
```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



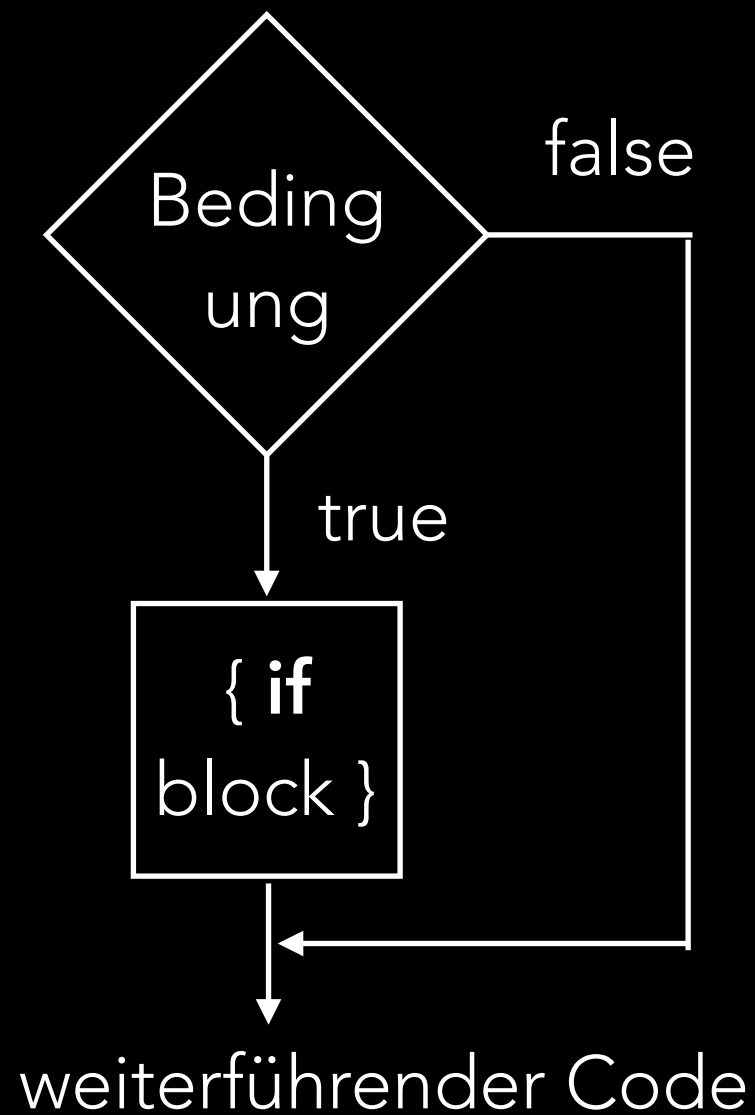
```
if( a < b ){
    // if code block
}
```



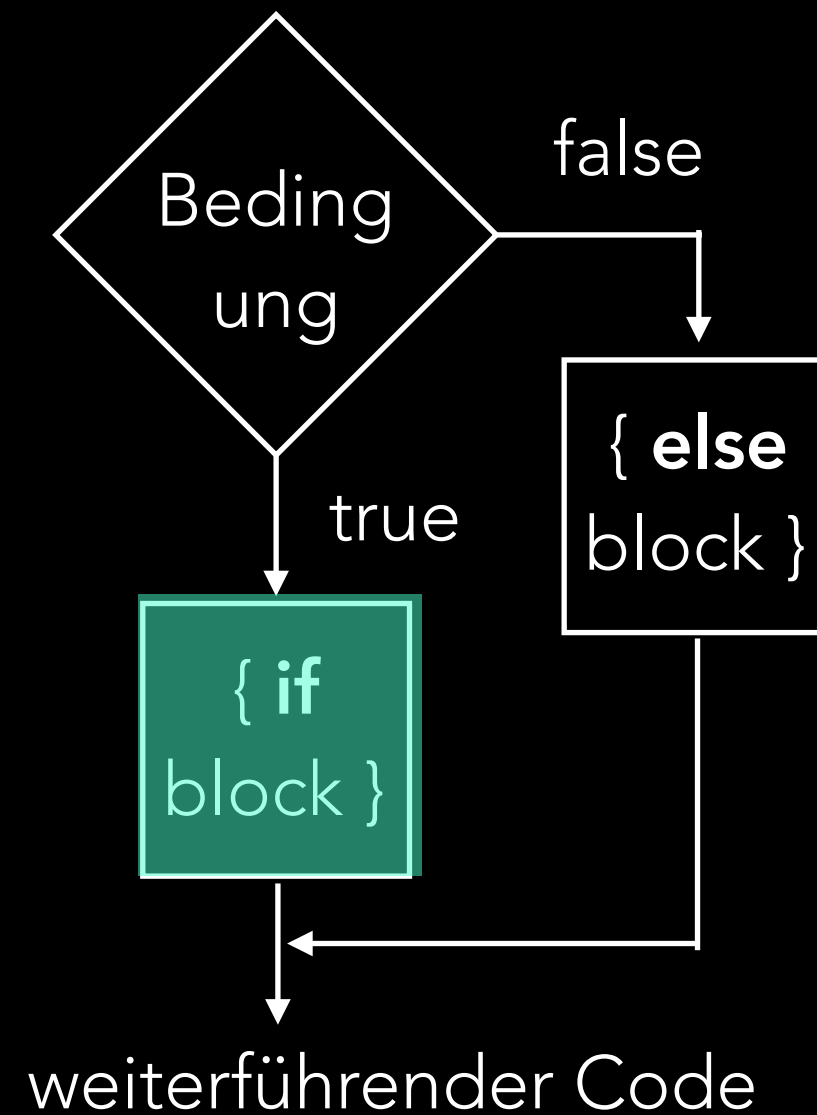
```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



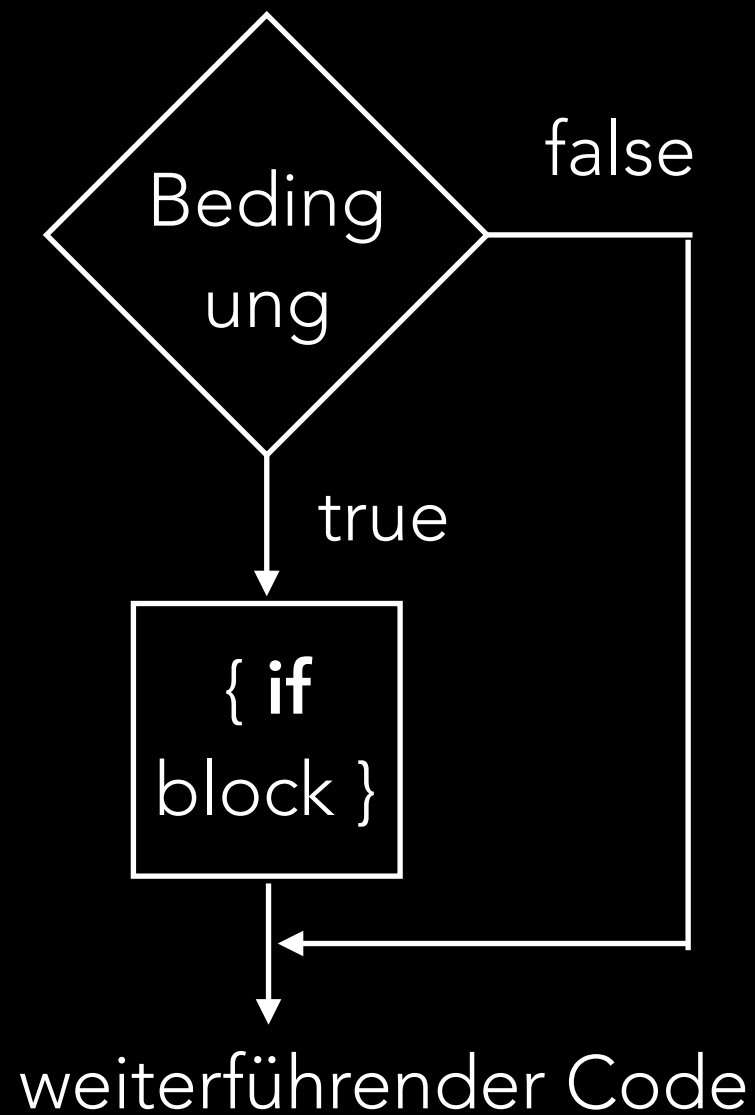
```
if( a < b ){
    // if code block
}
```



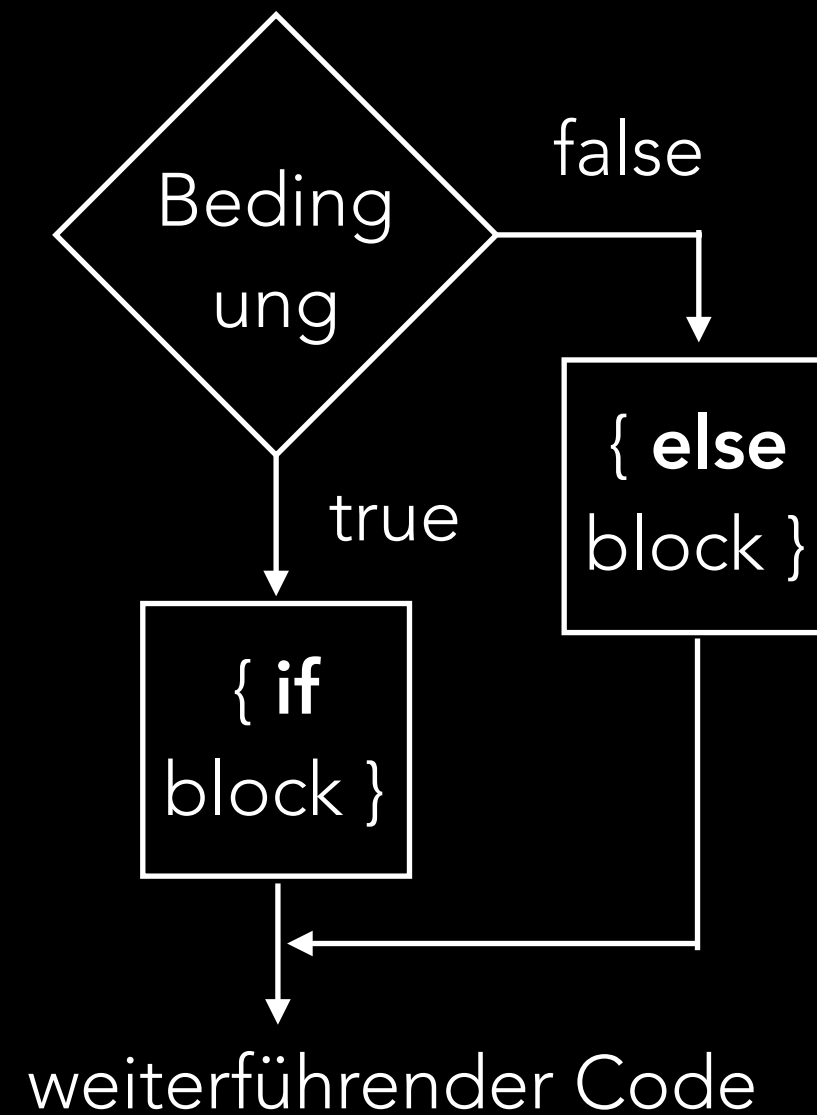
```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



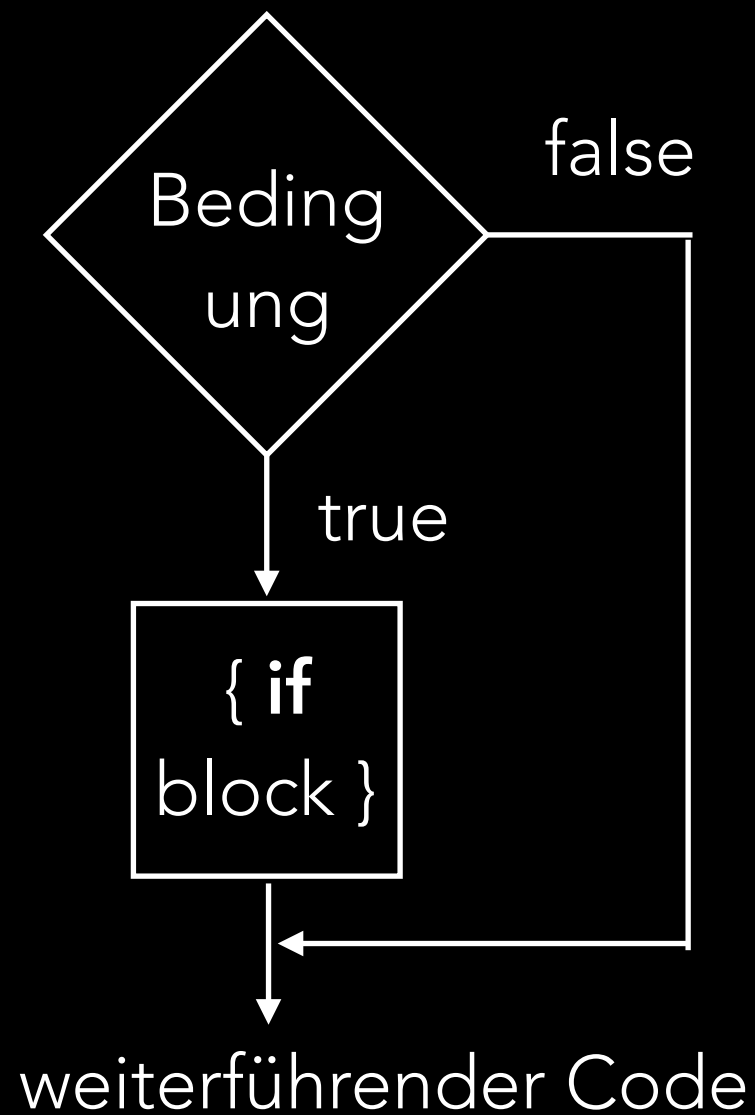
```
if( a < b ){
    // if code block
}
```



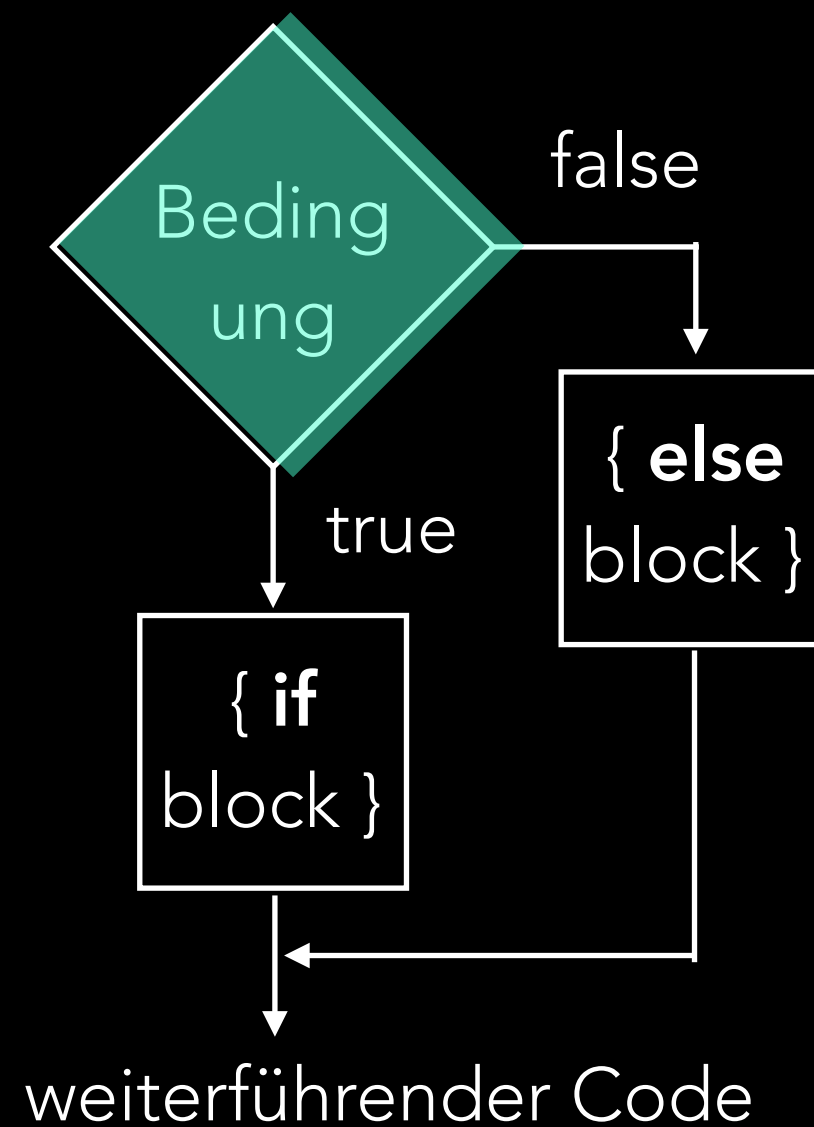
```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



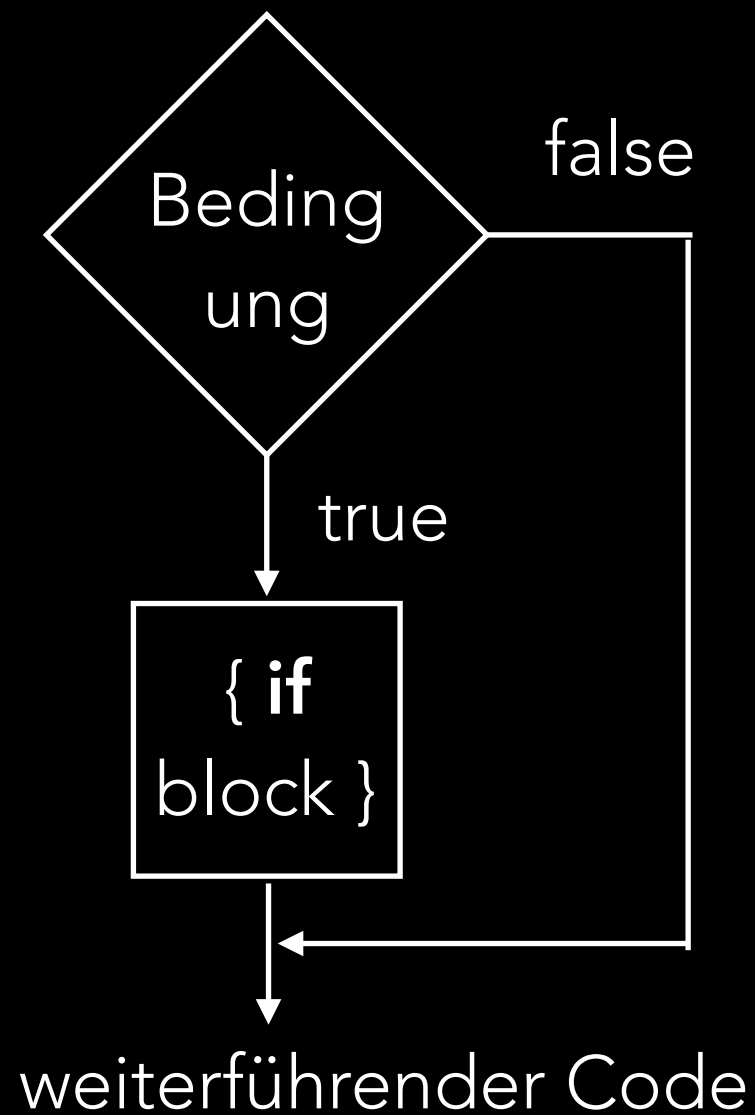
```
if( a < b ){
    // if code block
}
```



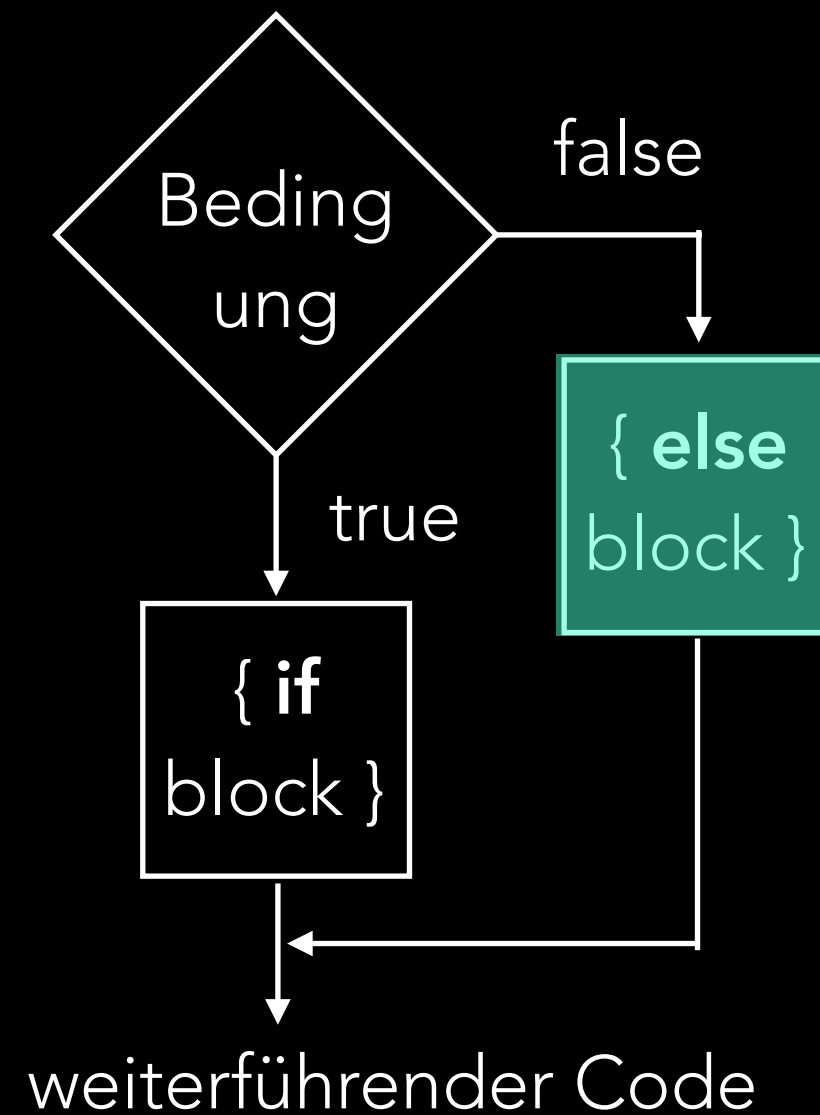
```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



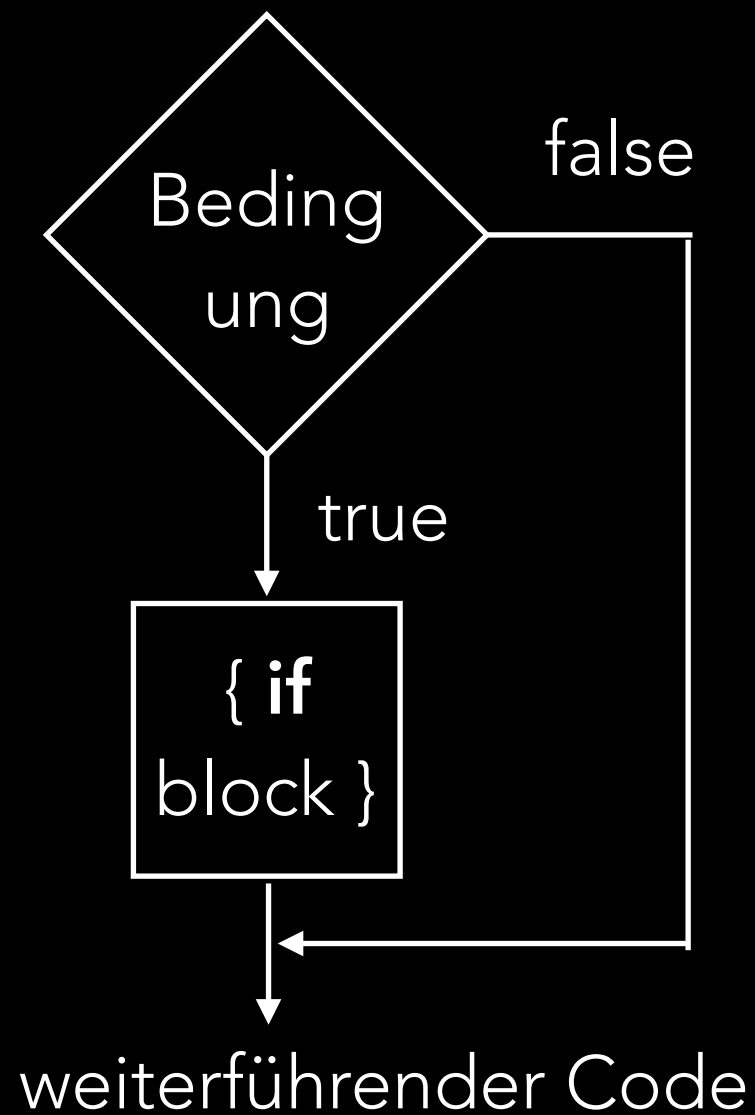
```
if( a < b ){
    // if code block
}
```



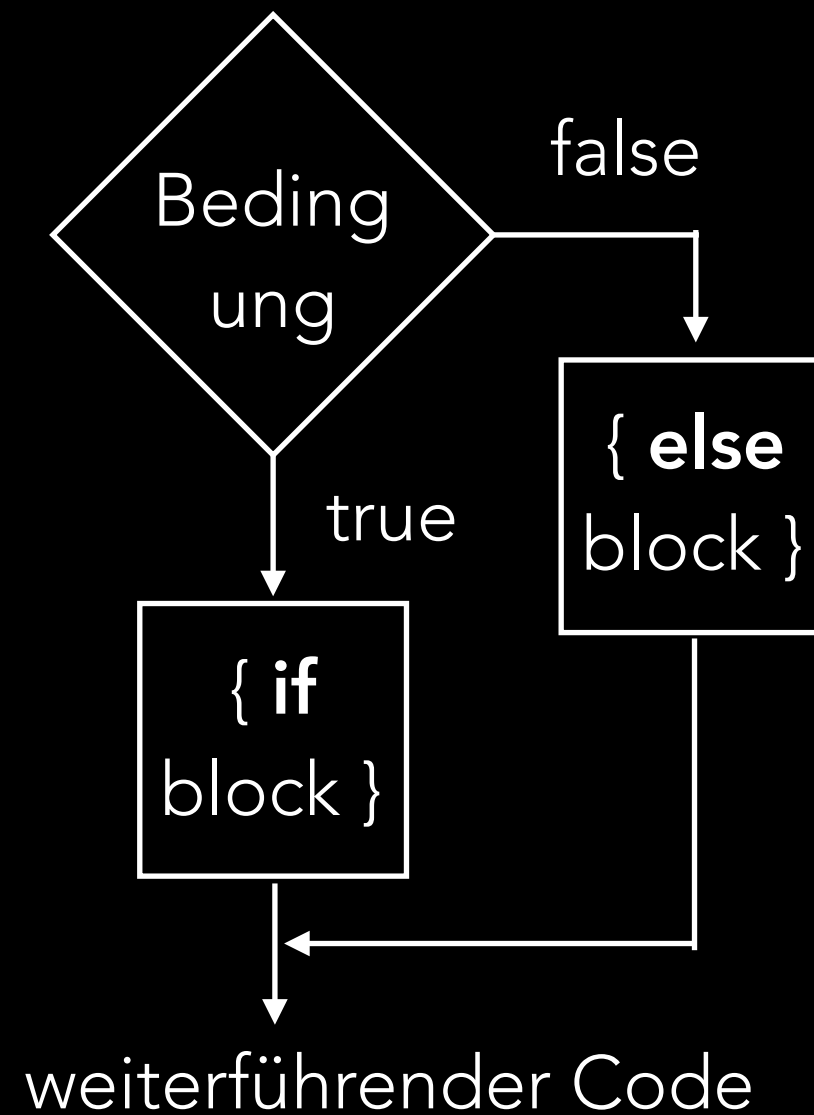
```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



```
if( a < b ){
    // if code block
}
```



```
if( a < b ){
    // if code block
}else{
    // else code block
}
```



Übung if / else

„Run, Forrest! Run!“

–(FORREST GUMP)



# METHODEN ( FUNKTIONEN )

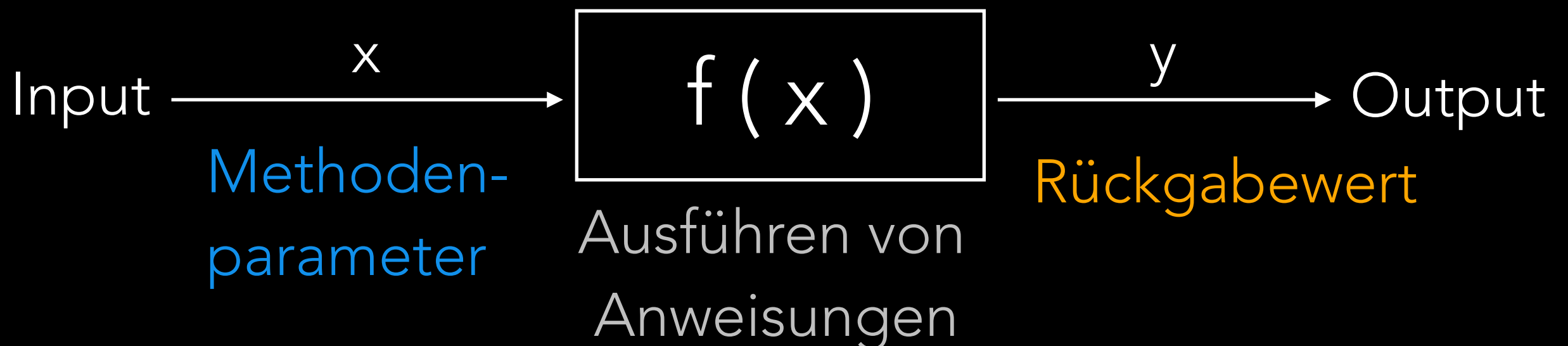
- Kann sich (in etwa) wie eine mathematische Funktion vorgestellt werden

Funktionsargument

$$y = f(x)$$

Resultat

Ausführen von Berechnung



$$y = 2 * x + 1$$

$$x = 2 \longrightarrow \boxed{2 * 2 + 1} \longrightarrow 5$$

$$y = \sin(x)$$

$$x = \text{PI} / 2 \longrightarrow \boxed{\sin(\text{PI} / 2)} \longrightarrow 1$$

# Methoden Definition

**Rückgabe Typ** (Datentyp des Outputs)      **Methodenname**      **Methodenparameter** (Input mit Typ und Name)

```
int line(int x){  
  
    int y = 2 * x + 1;  
  
    return y;  
}
```

Methoden  
Code  
Block

return Statement

- Definiert welche Variable zurückgegeben wird.
- Muss dem Typ des Rückgabewerts entsprechen.
- Beendet die Methode.

# Methoden Rückgabetypen

Rückgabotyp  
( Datentyp des Outputs )



```
int line(int x) { ...
```

- Datentypen die wir schon kennen
- neuer Datentyp: **void** für keinen Rückgabewert  
( keine return-Anweisung )

# Methoden Parameter

Methodenparameter  
( Input mit Typ und Name )

```
void printWords(String word1, String word2, String word3, String word4){  
    println( word1 + word2 + word3 +word4 );  
}
```

- Methodenparameter definieren welchen Input die Methode entgegennimmt
- Sind Variablen die im Methoden Block gültig sind
- Sind optional

# Methoden Aufruf

```
printWords("my", "name", "is", "fiona");
```

word1 = „my“, word2 = „name“  
word3 = „is“, word4 = „fiona“ → my name is fiona

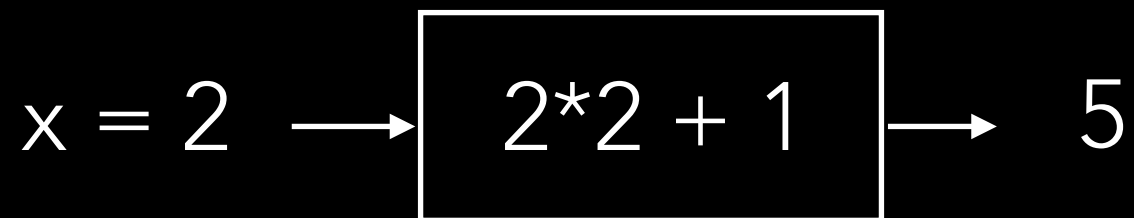
```
int result = line(2);
```

$x = 2 \rightarrow 2*2 + 1 \rightarrow \text{result} = 5$

# Methoden Aufruf

```
line(2);
```

der Rückgabewert wird in diesem Fall nicht gespeichert. Der Methodenaufruf entspricht aber dem Rückgabewert.



```
int result = line(2) + line(3) ;
```

12 = 5 + 7



# Startpunkt

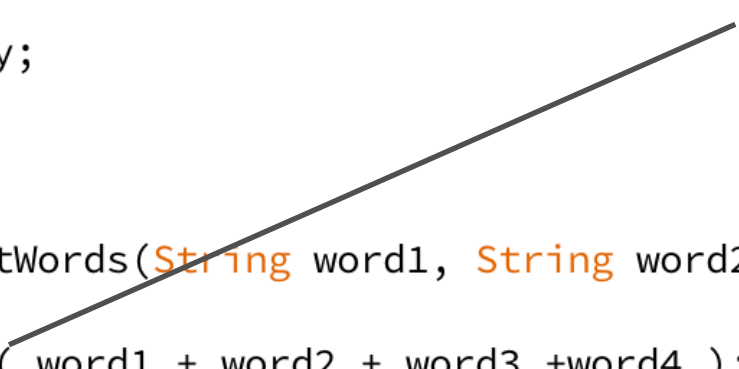
```
sketch_180611a
1
2 void setup(){
3
4   line(2);
5
6   int result = line(2);
7
8   printWords("my", "name", "is", "fiona");
9
10 }
11
12
13 int line(int x){
14
15   int y = 2 * x + 1;
16
17   return y;
18 }
19
20
21 void printWords(String word1, String word2, String word3, String word4){
22
23   println( word1 + word2 + word3 +word4 );
24
25 }
26
```

- Die Methode **setup()** bildet den Startpunkt in Processing
- jeglicher Code befindet sich in Methoden

# Startpunkt

```
sketch_180611a
1
2 void setup(){
3
4   line(2);
5
6   int result = line(2);
7
8   printWords("my", "name", "is", "fiona");
9
10 }
11
12
13 int line(int x){
14
15   int y = 2 * x + 1;
16
17   return y;
18 }
19
20
21 void printWords(String word1, String word2, String word3, String word4){
22
23   println( word1 + word2 + word3 +word4 );
24
25 }
26
```

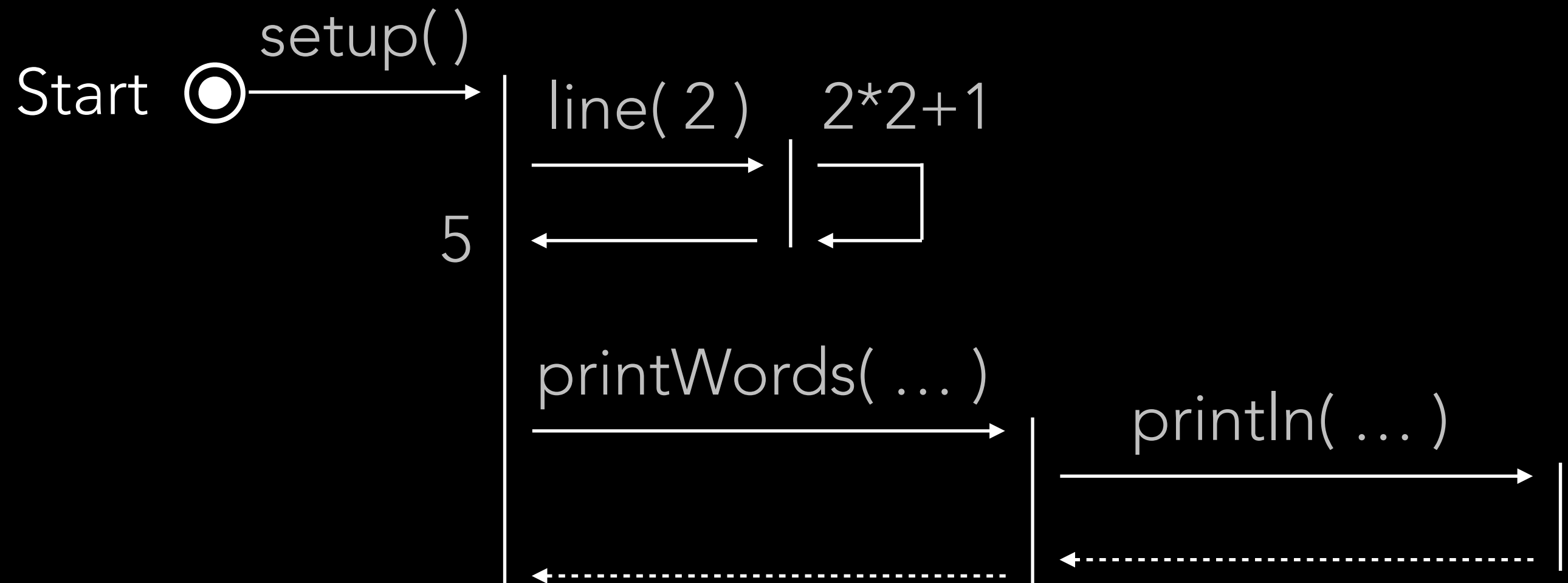
in Methoden können  
weitere Methoden  
aufgerufen werden



- Die Methode **setup()** bildet den Startpunkt in Processing
- jeglicher Code befindet sich in Methoden

```
1 void setup(){
2
3
4   int result = line(2);
5
6   printWords("my", "name", "is", "fiona");
7
8 }
```

```
12
13 int line(int x){
14
15   int y = 2 * x + 1;
16
17   return y;
18 }
19
20
21 void printWords(String word1, String word2, String word3, String word4){
22
23   println( word1 + word2 + word3 +word4 );
24
25 }
26
```



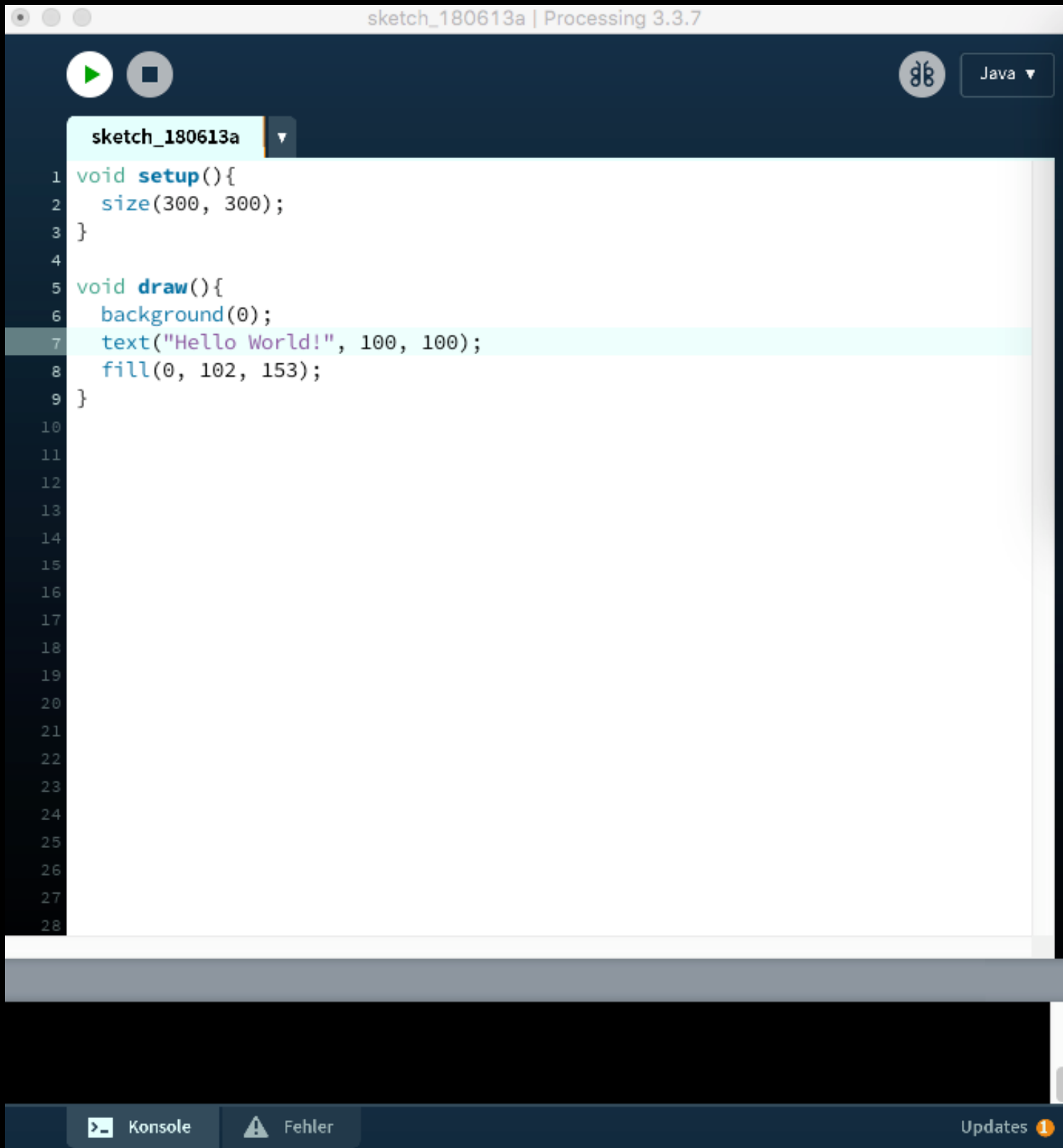
```
1 int line(int x){
2     int y = 2 * x + 1;
3     return y;
4 }
5
6 /*
7  * Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
8  * eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
9  * enim ad minim veniam, quis nostrud exercitation ullamco laboris
10 * nisi ut aliquid ex ea commodo consequat. Quis aute iure reprehenderit
11 * in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
12 */
13
14 void setup(){
15     int result = line(2);
16     printWords("my", "name", "is", "fiona");
17 }
18
19
20 void someDummyMethod(){
21     // nothing done here
22 }
23
24 void printWords(String word1, String word2, String word3, String word4){
25     println( word1 + word2 + word3 +word4 );
26 }
27
28
29 int nonsenseCalculation(int start, int max){
30     while(start < max){
31         start++;
32     }
33     return start;
34 }
```

## Übung Methoden

„ Genius is one percent inspiration and ninety-nine percent perspiration. “

– THOMAS EDISON

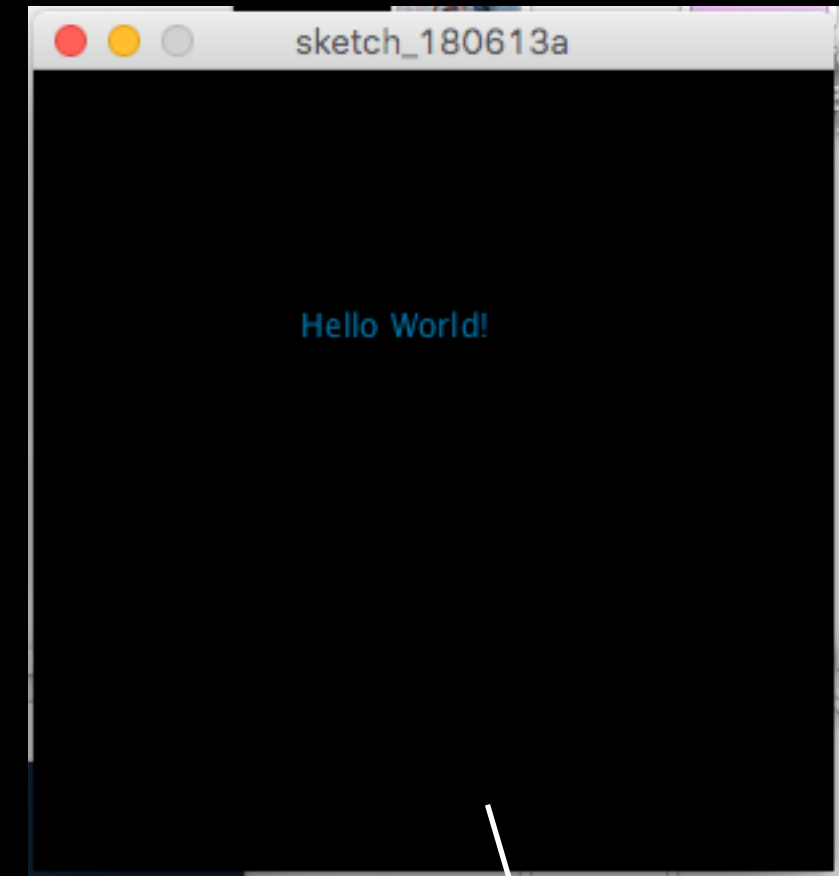
PROCESSING



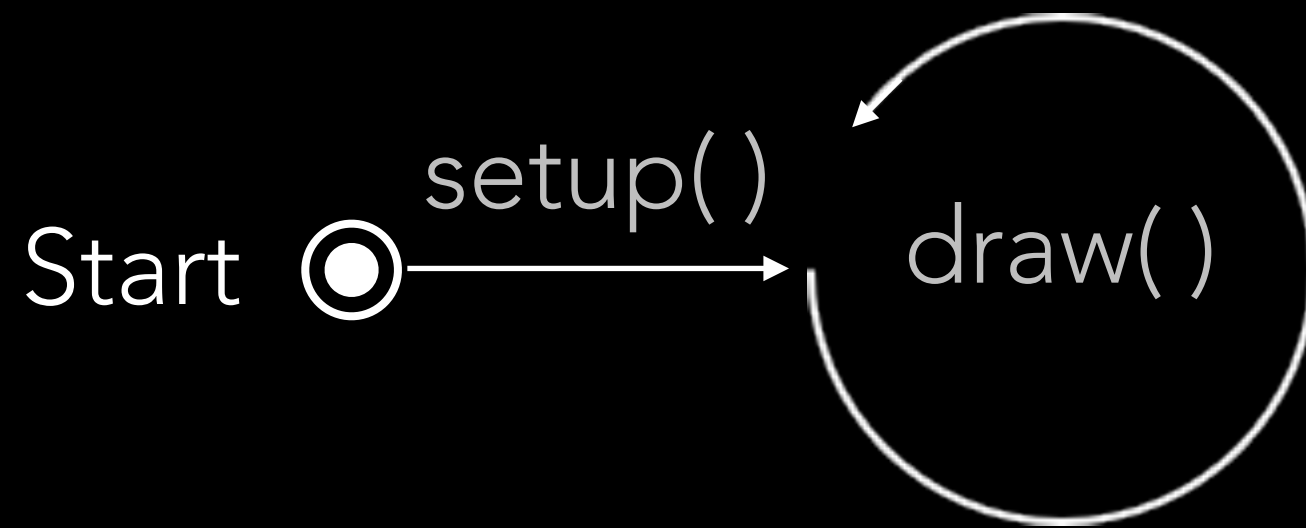
The screenshot shows the Processing IDE interface. The title bar reads 'sketch\_180613a | Processing 3.3.7'. The top toolbar contains a play button, a stop button, and a language dropdown set to 'Java'. The main code editor displays the following code:

```
1 void setup(){
2   size(300, 300);
3 }
4
5 void draw(){
6   background(0);
7   text("Hello World!", 100, 100);
8   fill(0, 102, 153);
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```

At the bottom, there is a panel with tabs for 'Konsole' and 'Fehler', and an 'Updates' button with a notification icon.



Display Window



sketch\_180613a | Processing 3.3.7

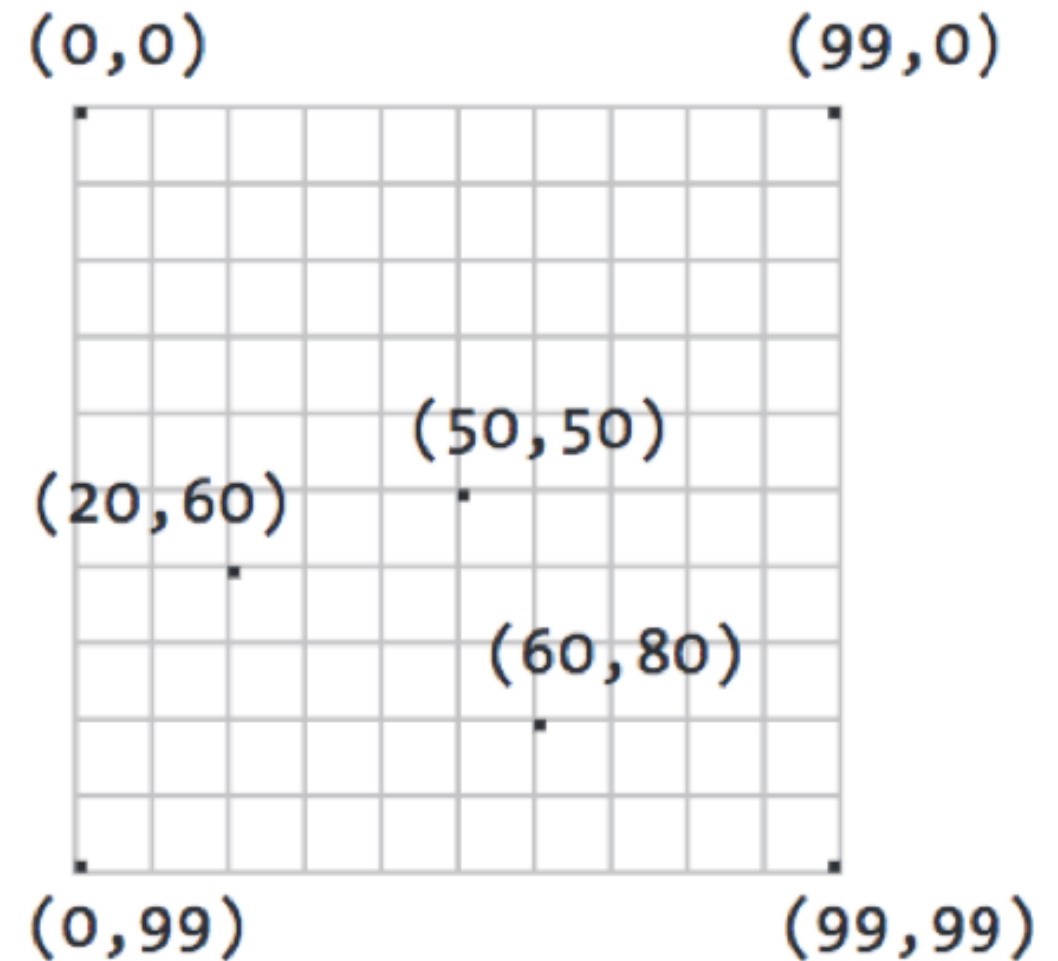
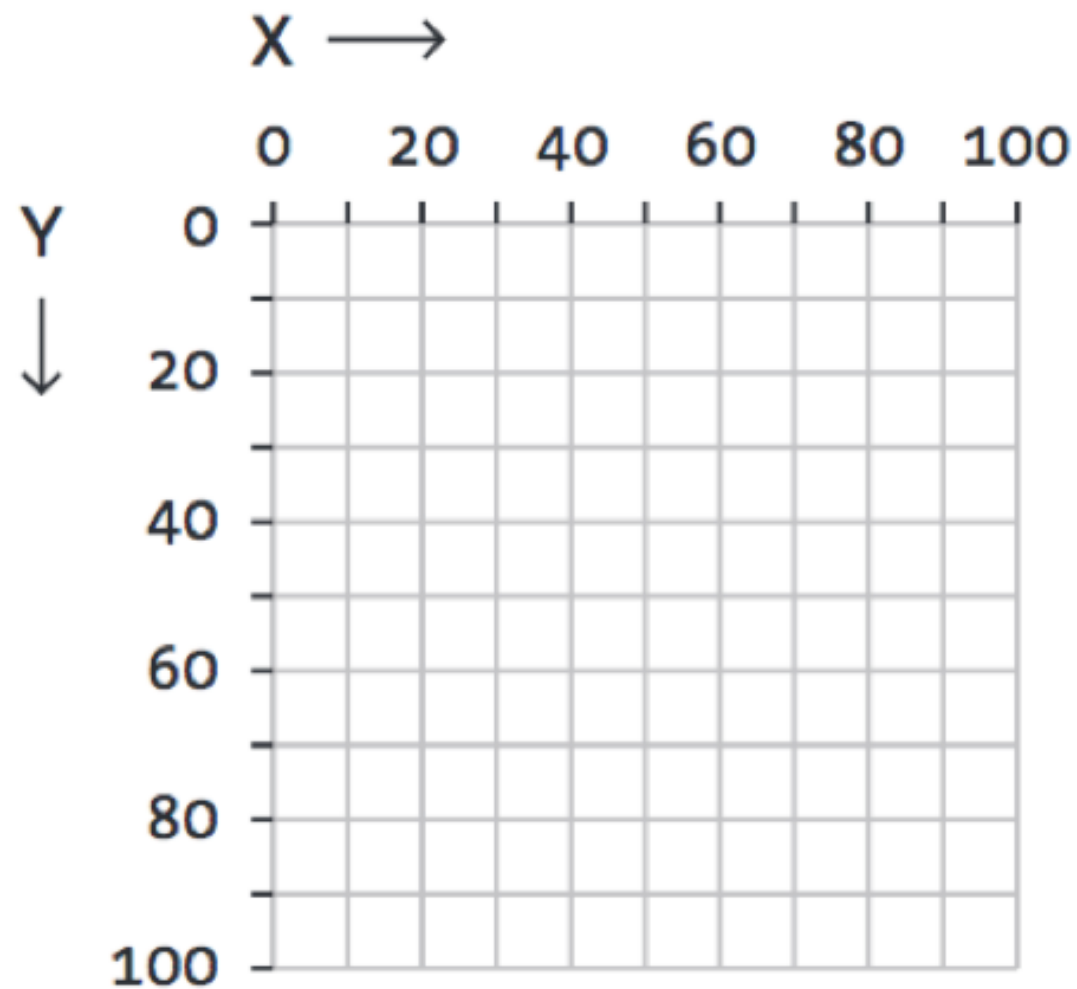
▶ ◻

sketch\_180613a ▼

```
1 void setup(){
2   size(300, 300);
3 }
4
5 void draw(){
6   background(0);
7   text("Hello World!", 100, 100);
8   fill(0, 102, 153);
9 }
10
11
```



# Das Koordinatensystem



# Erste Anweisungen

## **size( int: width, int: height )**

Definiert die Dimension des display window. Entspricht Pixeln ( bzw. der Breite und Höhe des Koordinatensystems ). Muss immer als erstes und in der setup() Methode definiert werden.

[https://processing.org/reference/size\\_.html](https://processing.org/reference/size_.html)

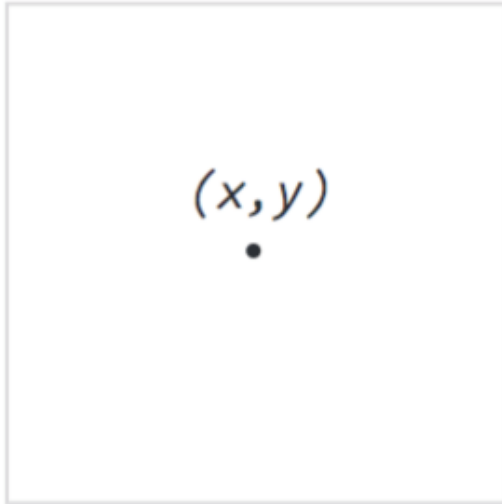
## **background( int: rgb )**

## **background( int: v1, int: v2, int: v3 )**

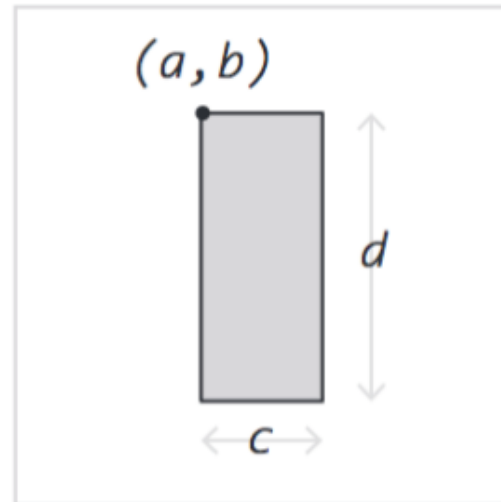
Setzt die Hintergrundfarbe im display Window.

[https://processing.org/reference/background\\_.html](https://processing.org/reference/background_.html)

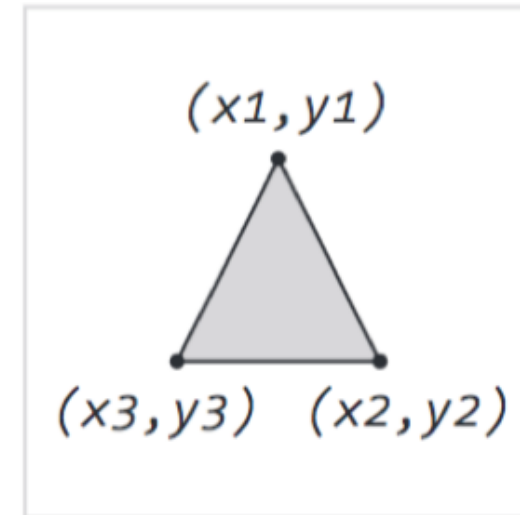
# Erste Anweisungen



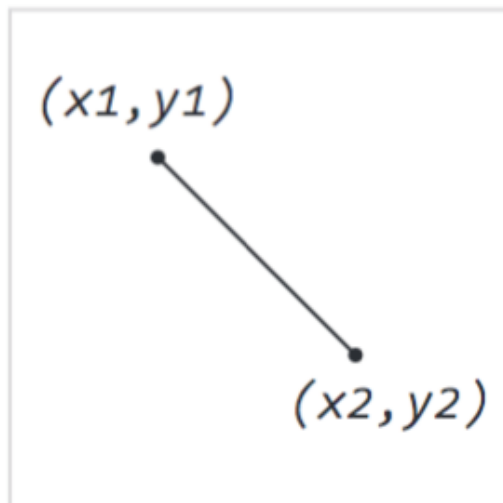
`point(x, y)`



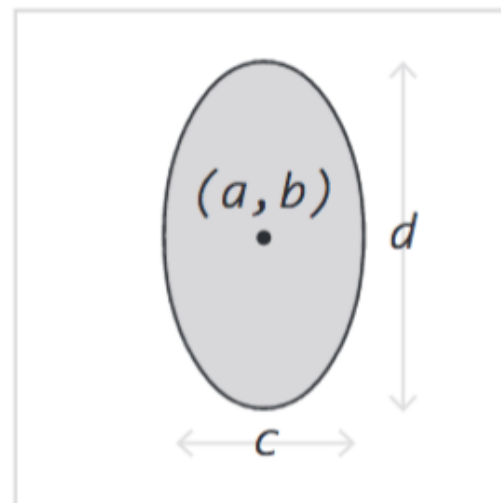
`rect(a, b, c, d)`



`triangle(x1, y1, x2, y2, x3, y3)`



`line(x1, y1, x2, y2)`



`ellipse(a, b, c, d)`

# Erste Anweisungen

**fill( int: rgb )**

**fill( int: v1, int: v2, int: v3 )**

Setzt die Farbe mit der die nachfolgenden Formen gefüllt werden sollen.

[https://processing.org/reference/fill\\_.html](https://processing.org/reference/fill_.html)

**stroke( int: rgb )**

**stroke( int: v1, int: v2, int: v3 )**

Setzt die Farbe für den Rahmen der nachfolgenden Formen.

[https://processing.org/reference/stroke\\_.html](https://processing.org/reference/stroke_.html)

**noStroke( )**

Deaktiviert den Rahmen für die nachfolgenden Formen.

[https://processing.org/reference/noStroke\\_.html](https://processing.org/reference/noStroke_.html)

Dokumentation: <https://processing.org/reference/>

[Processing](#)[p5.js](#)[Processing.py](#)[Processing for Android](#)[Processing for Pi](#)[Processing Foundation](#)

# Processing

[Cover](#)[Download](#)[Donate](#)[Exhibition](#)[Reference](#)[Libraries](#)[Tools](#)[Environment](#)[Tutorials](#)[Examples](#)[Books](#)[Handbook](#)[Overview](#)[People](#)[Shop](#)

Reference. Processing was designed to be a flexible software sketchbook.

## Structure

() (parentheses)  
, (comma)  
. (dot)  
/\* \*/ (multiline comment)  
/\*\* \*/ (doc comment)  
// (comment)  
; (semicolon)  
= (assign)  
[] (array access)  
{ } (curly braces)  
catch  
class  
draw()  
exit()  
extends

## Shape

createShape()  
loadShape()  
PShape  
  
2D Primitives  
arc()  
ellipse()  
line()  
point()  
quad()  
rect()  
triangle()

## Curves

## Color

### Setting

background()  
clear()  
colorMode()  
fill()  
noFill()  
noStroke()  
stroke()

### Creating & Reading

alpha()  
blue()  
brightness()  
color()

## Übung Processing - Erste Anweisungen

„Es wird vielleicht nicht einfacher, aber du wirst  
immer besser.“

–INTERNET