if ( ) { ….. }

Auswahl-Anweisungen

Eine if-Anweisung verwendet einen booleschen Ausdruck (ein Ausdruck, der entweder auf wahr oder falsch auswertet), um zwei Werte zu vergleichen.

Ist der boolesche Ausdruck in einer if-Anweisung „true“ (wahr), werden die angegebenen Codeanweisungen ausgeführt.
Ansonsten werden die Anweisungen vollständig übersprungen.

Der boolesche Ausdruck beschreibt eine Bedingung, die erfüllt sein muss, damit die angegebenen Aussagen zum Design hinzugefügt werden können.

Das Folgende zeigt die if-Anweisungs-Syntax:

if (<boolean anweisung>) {
// Code der ausgeführt wird, wenn die Anweisung „true“ (wahr) ist
}

Mittels einer „if“-Anweisung kann nun entschieden werden, ob es in die eine Richtung oder in die andere weitergehen soll.

```
if (a == 0){                    wenn ( „a“ gleich 0 ) {
    tueren(b,a);                        Auswahl zutreffend;
    }                                    }
  else {                              sonst {
    buero(b,a);                         Auswahl nicht zutreffend;
    }                                    }
```

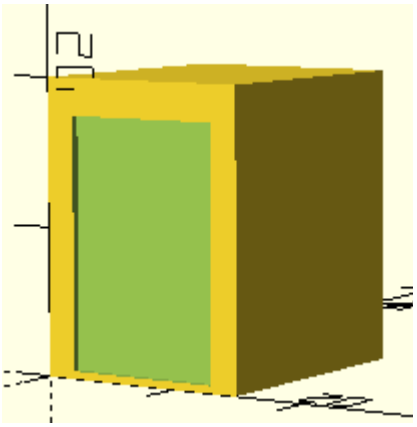
010-02

Auswahl von Fenster und Türen im Hochhaus erstellen

So ein Hochhaus ohne Eingang? Keine Türen? Machen wir es doch!

Als Modul „tueren“ wird ein Büroblock mit einer Tür erstellt:

```
module tueren (x=0, z=0) {
  difference(){
translate([x,0,z]) cube([15,18,20]); // + Büro
translate([x+2,-.5,z+.5]) cube([11,1,17]); // - Tür
   } // difference
} // module "tueren"
```



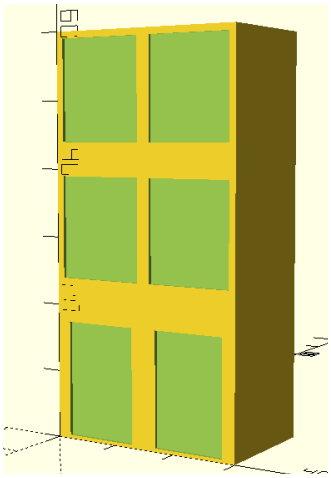
Im Prinzip ist eine Tür nichts anderes als ein Fenster, das fast bis zum Boden reicht.

Der Aufruf mit „tueren“ erfolgt über eine If / else Anweisung.

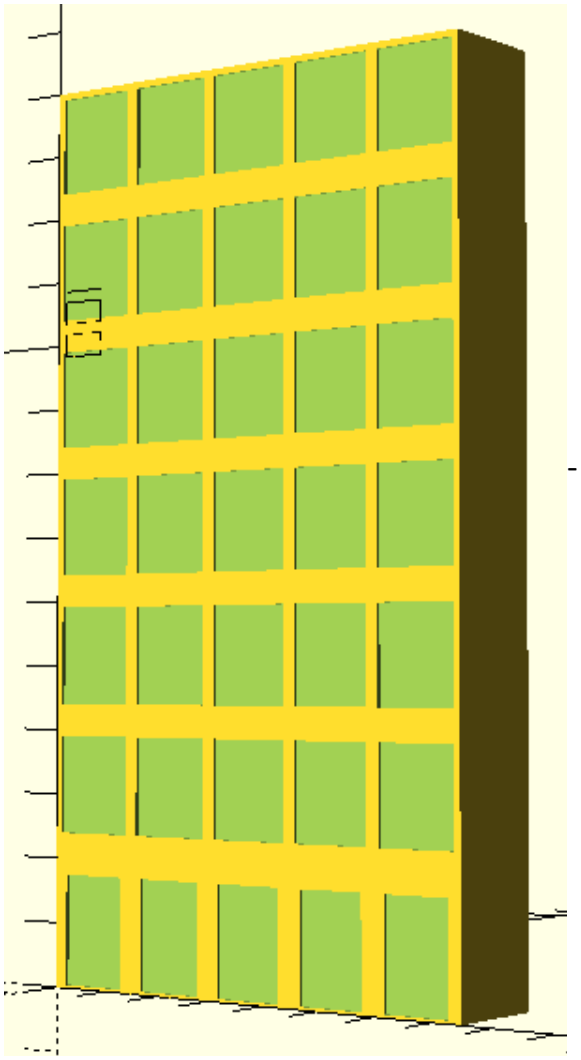
Dies bedeutet, wenn die „etage“ gleich 0 ist, wird eine Tür erzeugt.

Ansonsten (= else) wird ein Büroblock mit einem Fenster erstellt:

```
if (a == 0){
   tueren(b,a);
   }
else {
   buero(b,a);
   }
```



```
// hochhaus1_einfache_tueren.scad

reihe = 5;
etage=7;

module buero (x=0, z=0) {
  difference(){
translate([x,0,z])cube([15,18,20]); // + Büro
translate([x+1,-.5,z+4])cube([13,1,15]); // -
Fenster
   } // difference
} // module "buero"



module tueren (x=0, z=0) {
  difference(){
translate([x,0,z])cube([15,18,20]); // +
Büro
translate([x+2,-.5,z+.5])cube([11,1,17]); //
Tür
   } // difference
} // module "tueren"



for (b=[0:15:(reihe*15)-1])   {

   for (a=[0:20:(etage*20)-1])   {
   //   ([ 0 : Breite Büro : Etagen - 1])

      if (a == 0){
         tueren(b,a);
         }

       else {
         buero(b,a);
         }

   } // for (a...

} // for (b...
```



010-04

Boolesche Operatoren

Es kann eine if-Anweisung verwendet werden, um viele Arten von Bedingungen auszuwerten, indem eine Kombination aus sechs booleschen Operatoren und einem von zwei logischen Operatoren verwendet wird.

Ferner kann ein Standard-Szenario angeben werden, das dann ausgeführt wird, wenn die angegebene Bedingung falsch ist.
Dies geschieht indem man eine if-Anweisung mit einer else-Anweisung verbindet.

Boolesche Operatoren auswählen

OpenSCAD verwendet sechs boolesche Operatoren, um den Inhalt innerhalb eines booleschen Ausdrucks von Variablen auszuwerten. Jeder dieser Operatoren ergibt „true“, wenn der Vergleich gültig ist. Hingegen wird „false“
(falsch) ausgegeben sollte der Vergleich nicht gültig sein

< kleiner als

> größer als

<= kleiner als oder gleich mit

>= größer als oder gleich mit

== gleich mit

!= nicht gleich mit

Die verwendeten Symbole sind meist aus dem Mathe-Unterricht bekannt. OpenSCAD (wie bei den meisten anderen Programmiersprachen) ändern sich die Symbole ein wenig, damit sie leichter der Tastatur eingegeben werden können.
Es ist sehr wichtig, **==** nicht mit **=** zu verwechseln. Weil das einzelne Gleichheitszeichen bereits eine Verwendung hat (einer Variablen einen Wert zuzuweisen). Hingegen verwenden Boolesche Ausdrücke das doppelte Gleichheitszeichen (==), um zu testen, ob zwei Werte jeweils „gleich“ sind.

Im Wolkenkratzer-Beispiel-Listing testet der boolesche Operator zwei Werte mit dem Gleichheitszeichen (==) miteinander:

```
if (a == 1) {
    // hier folgt die Ausführung der Tür
}
```

Die booleschen Operatoren bieten somit viele Möglichkeiten zur Auswertung von Variablen. So kann einfach bestimmt werden, ob eine Bedingung wahr oder falsch ist.

Es kann damit eine Schleife erstellt werden, die je nach der Variablen eine andere Form ausgeben kann.

Ebenso kann damit erreicht werden, nur dann eine bestimmte Form zu erstellen, wenn eine vorgegebene Bedingung zutrifft.

Logische Operatoren mit booleschen Operatoren kombinieren

Außerdem können mehrere boolesche Ausdrücke mit einem der beiden logische Operatoren:

&& (steht für „**und**“)

|| (steht für „**oder**“).

Wird der Operator **&&** verwendet, müssen alle Bedingungen wahr sein, um die angegebenen Anweisungen auszuführen.

Wird der Operator **||** genutzt, muss mindestens eine von mehreren Bedingungen zutreffen (true / wahr sein).

Beispiel: „**&&**“ „**und**“

```
if (x > 10 && y <= 20) {
translate([x, y, 0]) cube([3, 4, 3]);
}
```

Dieser Codeschnipsel zeichnet nur dann einen Würfel, wenn x größer als ist 10 und y ist kleiner oder gleich 20.

Beispiel: „**||**“ „**oder**“ -Operator:

```
if (x > 10 || y <= 20) {
translate([x, y, 0]) cube([3, 4, 3]);
}
```

Ein Würfel wird gezeichnet, wenn entweder x größer als 10 oder y kleiner oder gleich 20 ist.

Nur einer der booleschen Ausdrücke muss als wahr ausgewertet werden, damit die Form gezeichnet wird.

010-06

Nach einer erweiterten Reihenfolge der Operationen können komplexe boolesche Ausdrücke erstellt werden, die viele arithmetische, boolesche und logische Operatoren umfassen.

OpenSCAD wertet Ausdrücke nach einer genau definierten Reihenfolge von Operationen aus:

1. **( )**	runde Klammer
2. **^**	Potenzieren
3. *****, **/**, **%**	Multiplikation, Division, Rest
4. **+**, **-**	Addition, Subtraktion
5. **<**, **>**, **<=**, **>=**	kleiner, größer, kleiner oder gleich, größer oder gleich
6. **==**, **!=**	gleich, nicht gleich
7. **&&**	logisches und
8. **||**	logisches oder

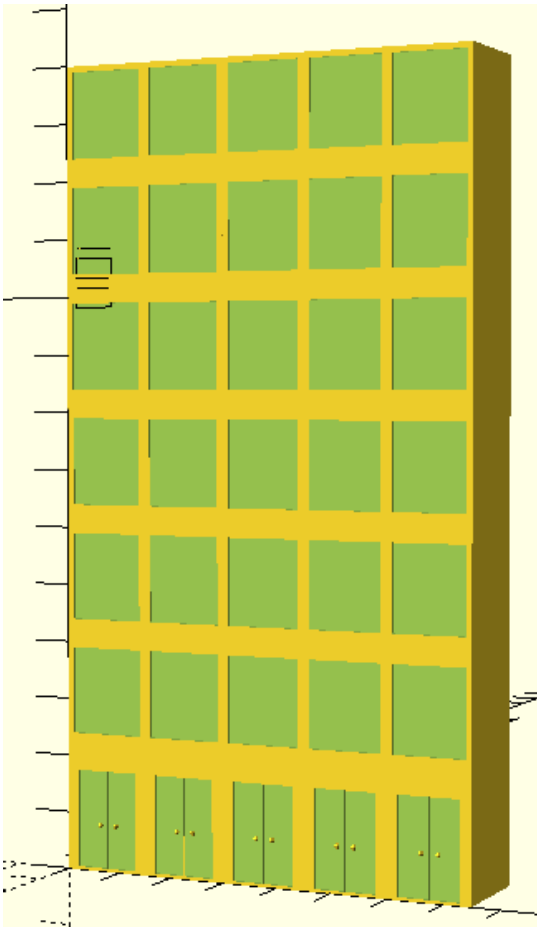
Operatoren auf der gleichen Ebene in der Reihenfolge der Operationen werden, entsprechend der Reihenfolge ihres Auftretens im gelesenen Ausdruck, von links nach rechts ausgeführt.
Andernfalls haben Operatoren an der Spitze dieser Liste einen höheren Rang vor den Operatoren am Ende der Liste, auch wenn das bedeutet, dass der Ausdruck von innen nach außen berechnet wird.

Das **^** - Potenzieren -Zeichen wird durch die Taste links der „1“ in der obersten Reihe erreicht – nichts passiert – Leertaste drücken und es erscheint!

Das Zeichen „**||**“ wird über die Taste „ >, < und |“ erreicht. Diese liegt über der linken „Strg“-Taste.
Das |-Zeichen (Piping) wird über die rechte „AltGr“-Taste **(1)** und gleichzeitigem Drücken der Taste „ >, < und |“ **(2)** erzielt.
Die rechte „AltGr“-Taste sitzt direkt rechts der langen Leertaste.

```
// hochhaus2a_mit_tueren.scad

reihe = 5;
etage=7;

module buero (x=0, z=0) {
  difference(){
translate([x,0,z])cube([15,18,20]); // + Büro
translate([x+1,-.5,z+4])cube([13,1,15]); // - Fenster
   } // difference
} // module "buero"

module tueren (x=0, z=0) {
  difference(){
translate([x,0,z])cube([15,18,20]); // + Büro
translate([x+2,-.5,z+.5])cube([5.4,1,17]); // - Tür links
translate([x+7.6,-.5,z+.5])cube([5.4,1,17]); // - Tür rechts
   } // difference
translate([x+6,+.5,z+8])sphere(.5); // + Griff links
translate([x+9,+.5,z+8])sphere(.5); // + Griff rechts
} // module "tueren"

for (b=[0:15:(reihe*15)-1])   {

   for (a=[0:20:(etage*20)-1])   {
   //   ([ 0 : Breite Büro : Etagen - 1])

      if (a == 0){
         tueren(b,a);
         }

       else {
         buero(b,a);
         }

   } // for a

} // for b
```

Mit etwas Verfeinerungen kann eine schönere Tür gestaltet werden …..



010-08

if ( )
{ ….. }
else
{ ….. }

Zwei-Wege-Entscheidungen mit if…else-Anweisungen treffen

Eine einfache if-Anweisung führt einen Codeabschnitt nur dann aus, wenn der boolesche Wert einen bestimmten Zustand einnimmt.
Um alternativen Code auszuführen, wenn die boolesche Bedingung falsch (false) sein sollte, wird einfach eine else-Anweisung noch an den Befehl angehängt.

Eine if...else-Anweisung erstellt somit eine Auswahl von zwei Möglichkeiten, mit der nun verschiedene Anweisungen { ….. } für jede Wahrheitsbedingung ausführen können.

Die Syntax für if...else:

```
if (<boolesche Ausdruck>) {
// Code der nur zutrifft, wenn die Aussage „true“ (wahr) ist
}
else {
// Code der zutrifft, wenn die Aussage „false“ (falsch) ist
}
```

Wenn der boolesche Ausdruck in der if-Anweisung wahr ist, wird die erste Gruppe von Anweisungen ausgeführt wird.

Ist der boolesche Ausdruck in der if-Anweisung false, werden die im else-Abschnitt enthaltenen Anweisungen ausgeführt.

Das Listing für den Wolkenkratzer kann leicht umgestaltet werden, indem eine else-Anweisung eingefügt wird. Der Wolkenkratzer soll genau eine Türreihe bekommen. Alle übrigen Etagen bekommen Fenster.

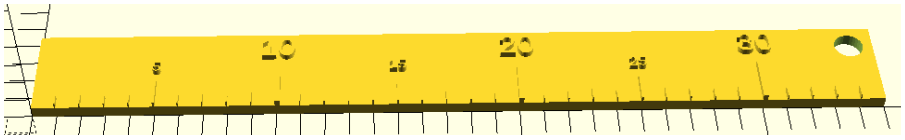
Die for-Schleife soll manchmal eine Reihe Türen und alle anderen Male eine Fensterreihe zeichnen, so wird die if-Anweisung wie folgt umgeschrieben:

// hochhaus2b_…..

```
…..
….
…
..
for (b=[0:15:(reihe*15)-1])   {
   for (a=[0:20:(etage*20)-1])   {
   //   ([ 0 : Breite Büro : Etagen - 1])
      if (a == 0){
      // wenn a gleich "0" dann eine Tür erstellen
         tueren(b,a);
         }
      else { // ansonsten Büro erstellen
         buero(b,a);
         }
   } // for a
} // for b
```

Ist der boolesche Ausdruck „a == 1“ wahr, wird eine Reihe Türen gezeichnet.
Ist dagegen der boolesche Ausdruck falsch, wird eine Reihe Fenster gezeichnet.

if ()
{.....}
else if ()
{.....}
else
{...}



Erweiterte if-Anweisungen

Eine erweiterte if-Anweisung verknüpft eine Bedingung an eine else-Anweisung, um eine Entscheidungen zu treffen.
OpenSCAD wertet die aus Boolesche Ausdrücke in einer erweiterten if-Anweisung in der aufgezählten Reihenfolge aus, bis einer der Ausdrücke als wahr ausgewertet werden kann.

Es kann optional eine else-Anweisung am Ende einer erweiterten if-Anweisung angefügt werden, um alle anderen möglichen false-Ausgaben abzufangen.

```
if (<boolesche Ausdruck>) {
// Code der nur zutrifft, wenn die Aussage „true“ (wahr) ist
}
else if (<boolesche Ausdruck>) {
// Code der zutrifft, wenn die ersteAussage „false“ (falsch) ist
// und dafür diese zweite Aussage zutrifft (true)
}
else {
// Code der zutrifft, wenn alle anderen Aussagen „false“ (falsch) sind
}
```

Wenn der boolesche Ausdruck in der if-Anweisung wahr ist, wird nur diese erste Anweisung ausgeführt.
Es können zusätzlich beliebig viele weitere if-Anweisungen hinzugefügt werden, um dadurch eine größere Auswahlmöglichkeit zu erhalten.

Nur der jeweilige zutreffende Codeabschnitt wird dann ausgeführt, während die restlichen Abschnitte übersprungen werden. Sollten gar keine booleschen Ausdrücke zutreffen, wird der im optionalen Else-Abschnitt (falls vorhanden) angegebene Code ausgeführt.

Weil dieser else-Abschnitt den Standard beschreibt, muss er am Ende einer erweiterten if-Anweisung eingefügt werden.

Das folgende Listing verwendet eine erweiterte if-Anweisung, um Teilstriche unterschiedlicher Größe mit Beschriftung auf einem Lineal darzustellen. Dieses Listing erstellt ein Lineal (mit mm-Angaben, Länge nach Wunsch) mit Markierungen in drei sich wiederholenden Intervallen: ein, fünf und zehn Millimeter.

Markierungen im 10mm-Abstand sind die längsten, gefolgt von den Markierungen im 5mm-Abstand, bis hin zu den ganz kurzen mm-Abstands-Markierungen.

```
// Lineal in mm

$fn=30;

// Programmbereich
masstab(45); // Gesamtlänge in mm eingeben

// Modulbereich
module masstab(mm) {
millimeter = 1; // 1 Einheit pro mm
gesamt_markierung = mm; // je mm
laenge = millimeter * gesamt_markierung;
tiefe = 4 * millimeter;
dicke = 0.5 * millimeter;
strich_breite = 0.1 * millimeter;
strich_hoehe = 1.5 * dicke;

// Haupt-Form des Massstabs
difference() {
cube([laenge, tiefe, dicke]);
translate([laenge-millimeter, tiefe-millimeter, -0.5])
cylinder(h=dicke+1, r=0.15*tiefe);
} // Ende von: difference()

// Strich-Markierung
 for(zaehler = [1:1:gesamt_markierung-1]) {
markierung_x = millimeter * zaehler - 0.5 * strich_breite;

  if (zaehler%10 == 0) { // Beschriftungs-Abstand
  // Schriftgestaltung
  translate([millimeter * zaehler, 0.65 * tiefe, 0])
  linear_extrude(strich_hoehe)
```

```
  text(str(zaehler), size=millimeter, halign="center");
  // Gestaltung des Striches
  translate([markierung_x, 0, 0])
  cube([strich_breite * 2, 0.5 * tiefe, strich_hoehe]);
  } // Ende von: if (zaehler%10 == 0)

  else if (zaehler%5 == 0) { // 5 er mm -Markierung
  // Schriftgestaltung
  translate([millimeter * zaehler, 0.45 * tiefe, 0])
  linear_extrude(strich_hoehe)
  text(str(zaehler), size=millimeter * 0.5, halign="center");
  // Gestaltung des Striches
  translate([markierung_x, 0, 0])
  cube([strich_breite *1.5, 0.325 * tiefe, strich_hoehe]);
  } // Ende von : else if (zaehler%5 == 0)

  else { // mm - Markierung
  // Gestaltung des Striches
  translate([markierung_x, 0, 0])
  cube([strich_breite, 0.125 * tiefe, strich_hoehe]);
  } // Ende von: else

} // Ende von: for(zaehler = [1:1:gesamt_markierung-1])
} // Ende von: module masstab(mm) {
```

Erklärung des Listings:

In diesem Script ist der Programmteil am Anfang und besteht lediglich aus einer Zeile. Danach folgt das Modul.

1. masstab(**45**); hier wird die Gesamtlänge des Lineals in mm an den Modulbereich übergeben.

2. Zuerst wird eine Sammlung von Variablen definiert, um bei der weiteren Programmierung zu helfen:

module masstab(**mm**) die Variable mm wird zu 35
millimeter = **1**; pro mm soll ein Strich als Markierung erscheinen
gesamt_markierung = mm; Wie weit soll markiert werden?
laenge = millimeter * gesamt_markierung; = 1 * 35 = Gesamtlänge
tiefe = 4 * millimeter; = Höehe / Y / des Lineals
dicke = 0.5 * millimeter; = Dicke des Lineals
strich_breite = 0.1 * millimeter; = Breite des Striches
strich_hoehe = 1.5 * hoehe; = Höhe des Striches

Mittels „masstab(35);“ wird das gleichnamige Modul aufgerufen und der Wert von 35 als Variable übergeben.

Im Abschnitt Haupt-Form des Moduls wird das Lineal leer erzeugt, ebenso das Loch zum Aufhängen.

Im Abschnitt „Strich-Markierung“ läuft eine for-Schleife von 1 bis zum Ende der Variable (-1) von 35.

Nun folgen die if-Anweisungen:
Der Clou des Ganzen ist das **%** Prozentzeichen!

„**if (zaehler%10 == 0)**“

würde bedeuten, dass die Variable „zaehler“ durch 10 geteilt wird, der eventuelle entstehende Rest wird auch angezeigt.
Wird jedoch „(zaehler%10 == 0)“ als Anweisung geschrieben, heißt es, dass der „zaehler“ durch 10 geteilt werden soll, jedoch nur, wenn der Rest == 0 ist!

„=“ bedeutet eine Wertzuweisung einer Variablen, dagegen „==“ vergleicht zwei Variable miteinander.

Somit wird jeder 10.te Strich erzeugt: 10, 20, 30 - alle nachfolgende Möglichkeiten werden übersprungen.

Trifft dies nicht zu, also zaehler%10 mit Rest gleich 0, dann if-Anweisung überspringen und zur nächsten gehen.

„**else if (zaehler%5==0)**“

tritt nun in Aktion. Somit darf diese Anweisung nur ausgeführt werden, wenn „zaehler“ durch 5 ohne Rest teilbar ist.
So wird jeder 5.te Strich erzeugt: 5, 15, 25 - alle nachfolgende Möglichkeiten werden übersprungen.

Trifft dies wieder nicht zu, also zaehler%5 mit Rest gleich 0, dann if-Anweisung überspringen und zur nächsten gehen.

else {
Jetzt folgt als Schluss nur noch die restliche Möglichkeit einen einfachen mm-Strich zu gestalten. Dies wird ausgeführt, wenn die oberen zwei Anweisungen **nicht** zutreffen, also bei 1, 2, 3, 4, 6, 7, 8, 9, 11, 12, ….

Ist nun der „zaehler“ beim Maximum der Gesamtlänge angekommen, wird die for-Schleife beendet und das Lineal ist damit fertig.

010-14

Die einzelnen Ergebnisse des „zaehlers“ wird wie folgt erstellt:

…..
if (zaehler%10 == 0)
echo("10er Zaehler: ",zaehler);
…..

ergibt als Ergebnis:

ECHO: "10er Zaehler: ", 10
ECHO: "10er Zaehler: ", 20
ECHO: "10er Zaehler: ", 30

Es werden demnach nur die Zählerstände angezeigt die durch 10 ohne Rest teilbar sind.

…..
else if (zaehler%5 == 0) {
echo("5er Zaehler: ",zaehler);
…..

Ergebnis:
ECHO: "5er Zaehler: ", 5
ECHO: "5er Zaehler: ", 15
ECHO: "5er Zaehler: ", 25

1. Die Ergebnisse beim normalen Zähler sind:
   ECHO: "1er Zaehler: ", 1
   ECHO: "1er Zaehler: ", 2
   ECHO: "1er Zaehler: ", 3
   ECHO: "1er Zaehler: ", 4
   ECHO: "5er Zaehler: ", 5
   ECHO: "1er Zaehler: ", 6
   ECHO: "1er Zaehler: ", 7
   ECHO: "1er Zaehler: ", 8
   ECHO: "1er Zaehler: ", 9
   ECHO: "1er Zaehler: ", 11
   ECHO: "1er Zaehler: ", 12
   …..

   Hier ist es egal, was der Zähler angibt, denn es soll an jedem Millimeter eine Strichbreite von 0,1 erstellt werden.

   Außerdem sobald ein 10er oder 5er Strich erzeugt wurde, wird aus der „if“ Schleife gesprungen und an den Anfang gesetzt. Und zwar solange, bis das Schleifenende den vorgegebenen Wert erreicht hat

% = Gib mir den Rest

Mittels „%“ kann der Rest einer Division ausgegeben werden.

Eine normale Division mit „/“ erzeugt z.B.:

```
for (zaehler =[1:1:20]){
   echo("Zaehler: ",zaehler);
   echo("Ergebnis ",zaehler," / 3: ", zaehler/3);
   }
```

Das Listing erzeugt eine Zahlenreihe von 1 bis 20 – in Schritten von 1. In der Konsole erscheint das Ergebnis wie im rechten Bild:

1 / 3 : 0,333333
2 / 3 : 0,666667
3 / 3 : 1
4 / 3: 1,33333
5 / 3 : 1,666667
6 / 3 : 2
usw.

```
ECHO: "Ergebnis ", 1, " / 3 : ", 0.333333
ECHO: "Ergebnis ", 2, " / 3 : ", 0.666667
ECHO: "Ergebnis ", 3, " / 3 : ", 1
ECHO: "Ergebnis ", 4, " / 3 : ", 1.33333
ECHO: "Ergebnis ", 5, " / 3 : ", 1.66667
ECHO: "Ergebnis ", 6, " / 3 : ", 2
ECHO: "Ergebnis ", 7, " / 3 : ", 2.33333
ECHO: "Ergebnis ", 8, " / 3 : ", 2.66667
ECHO: "Ergebnis ", 9, " / 3 : ", 3
ECHO: "Ergebnis ", 10, " / 3 : ", 3.33333
ECHO: "Ergebnis ", 11, " / 3 : ", 3.66667
ECHO: "Ergebnis ", 12, " / 3 : ", 4
ECHO: "Ergebnis ", 13, " / 3 : ", 4.33333
ECHO: "Ergebnis ", 14, " / 3 : ", 4.66667
ECHO: "Ergebnis ", 15, " / 3 : ", 5
ECHO: "Ergebnis ", 16, " / 3 : ", 5.33333
ECHO: "Ergebnis ", 17, " / 3 : ", 5.66667
ECHO: "Ergebnis ", 18, " / 3 : ", 6
ECHO: "Ergebnis ", 19, " / 3 : ", 6.33333
ECHO: "Ergebnis ", 20, " / 3 : ", 6.66667
```

Wie man sieht, erscheinen Ergebnisse mit Stellen hinter dem Komma.

Wird nun der laufende „zaehler“ durch 3 geteilt, kann ebenso noch der Rest der Division über eine Rechnung ausgegeben werden:

```
for (zaehler =[3:1:20]){
   echo("Ergebnis ",zaehler," / 3 : ", zaehler/3);
   echo("und der Rest ist: ", zaehler%3);
   }
```

zaehler/3 ergibt das Resultat der Division durch 3.
zaehler%3 zeigt nur den **Rest** der Division durch 3 an.

ECHO: „Ergebnis „, 8, „ / 3 : “, 2,66667	= 8 / 3 = 2,66667
ECHO: „und der Rest ist:“, 2	= ….. Rest: 2
ECHO: „Ergebnis „, 9, „ / 3 : “, 3	= 9 / 3 = 3
ECHO: „und der Rest ist:“, 0	= ….. Rest: 0

Bei der Division 8:3 = 3 + 3 = 6 dann noch 8 - 6 = 2 somit bleibt ein Rest von 2.
Bei der Division 9:3= 3 + 3+3 = 9 bleibt ein Rest von 0.

Über den Rest durch (**%**) kann also eine zusätzliche Auswahl getroffen werden.

Über die Zeile :
if (zaehler%10 == 0) wird also nur ausgewählt, wenn der zaehler durch 10 ohne Rest teilbar ist: 10, 20 und 30.

else if (zaehler%5 == 0) wird nur dann ausgewählt, wenn die vorhergehende Anweisung nicht erfüllt war (%10 == 0) und dafür nun das durch 5 geteilte Ergebnis kein Rest aufweist. Dies trifft bei 5, 15 und 25 zu.

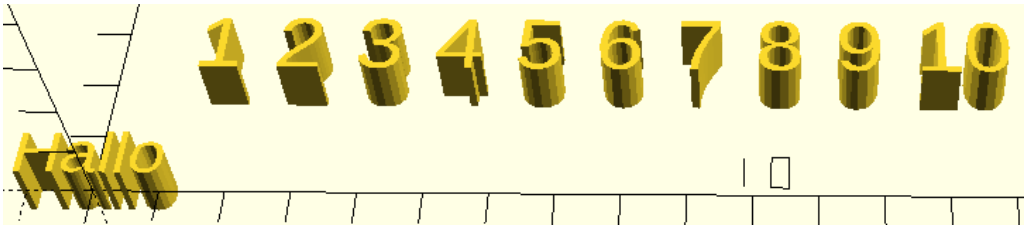
else { = alle anderen Möglichkeiten, wie:
1,2,3,4,-,6,7,8,9,-,11,12,13,14,-,16,17,18,19,-,21,22,23,24,-,26,27,28,29,-,…
demnach alle Zahlen außer 5er und 10er!

text(str(variable) , size=0.5, halign="center");

Eine Zahl (Variable) als String = Text ausgeben

Eine Erweiterung von „**text**“ kann der Befehls-Zusatz „**str**“ eine Zahl (Variable) als Text darstellen. Im Falle des Lineals wird sogar die Größe der Schrift und dessen Ausrichtung (horizontal - zentriert) angegeben.

```
// text/str = Zahlen als Text ausgeben

// Hier nur Text:
das_wort = "Hallo";
text(str(das_wort) , size=0.75, halign="center");

// Zahlenreihe erstellen
translate([0,2,0]){
for (die_zahl =[1:1:10]){
        translate([die_zahl*1.25,0,0])
        text(str(die_zahl) , size=1);
        } // Ende von: for (die_zahl =[1:1:10]){
 } // Ende von: translate([0,2,0]){
```



&& : besonderes UND

Das Verschachteln von if-Anweisungen (eine in eine andere einsetzen) ist nur eine Möglichkeit zwei Anweisungen auf beider Richtigkeit zu überprüfen.

Eine weitere Option ist: **&&** als UND-Überprüfung

Auf einer Ebene kann eine verschachtelte if-Anweisung das && ersetzen:

```
if (x < 8 && y == 10) {
// Wird nur ausgeführt, wenn beide Anweisungen zutreffen
}
```

So kann auch stattdessen eine verschachtelte if-Anweisung mit dem gleichen Ergebnis erfolgen:

```
if (x < 8) {
    if (y == 10) {
    // Wird nur ausgeführt, wenn beide Anweisungen zutreffen
    }
}
```

Am besten ist es, den Operator && für einfache Kombinationen von Booleschen Ausdrücken zu verwenden - die alle wahr sein müssen, damit eine bestimmte Bedingung erfüllt wird.

Die Verwendung von verschachtelten if-Anweisungen kann jedoch einfacher sein, wenn das Ergebnis mehrere boolesche Ausdrücke testen soll, die beides sein könnten:

richtig oder falsch:

```
x=9;
y=10;
echo("-------------------------");
echo(" X= ",x,"   Y= ",y);
echo("-------------------------");
if (x < 8) {

if (y == 10) {
echo(" Y==10"," = "," x < 8 und y == 10");
```

```
}

else if (y < 10) {
echo(" Y<10"," = "," x < 8 und y < 10");
}

else {
echo(" x < 8 und y > 10");
}

} else {

if (y == 10) {
echo(" x >= 8 und y ==10");
}

else {
echo(" x >= 8 and y !=10");
}

}
```

Es ist normalerweise möglich, komplexe Zustände mit einer Vielfalt von Kombinationen aus booleschen und logischen Operatoren, erweiterten if- und verschachtelten if-Anweisungen zu beschreiben.

Oft ist jedoch die beste Wahl die Kombination von Bedingungen, die für die Person am sinnvollsten erscheint, die den Entwurf erstellt.

Nützliche Anwendungen von if-Anweisungen

Eine if-Anweisung sollte eingefügt werden, wann immer eine bestimmte Bedingung variiert werden soll.
Die folgenden Situationen ist ein Beispiele dafür, dass eine if-Anweisungen in einem Projekten zwischen Ansichtsmodus und 3D-Druck-Modus entscheidet.

Betrachtet man das Projekt Türme von Hanoi, wird festgestellt das beim Entwerfen der Stapelscheiben es praktisch war, diese senkrecht auf einem Stift zu stapeln.

Diese Konfiguration ist jedoch nicht die beste für den 3D-Druck. Da alle Scheiben aufeinander liegen, würden sie als ein Stück gedruckt werden

Eine nützliche Technik besteht darin, zwei Versionen des Entwurfs zu erstellen: einmal zur Ansicht des Endergebnisses und eine für den 3DDruck.

Der Entwurf verwendet ein Modul um die unterschiedlichen Scheiben zu gestalten.

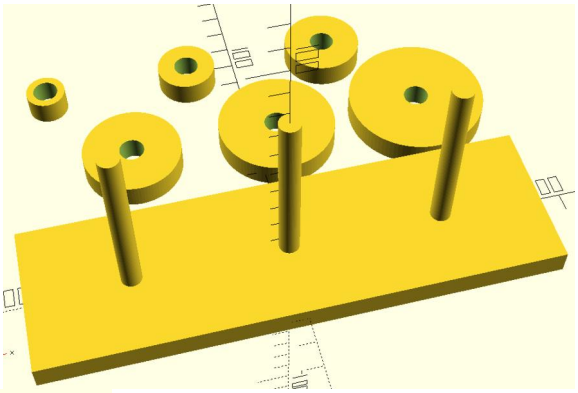
Das folgende Listing ist auf „**3ddruck“** voreingestellt:

```
auswahl = "3ddruck"; // oder "ansicht"
//auswahl = "ansicht"; // oder "3ddruck"
```

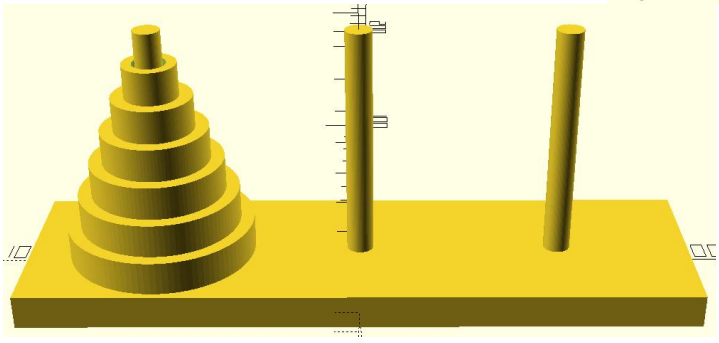
Werden vor diese Zeile // gesetzt wird diese zu einem Kommentar. Dafür sollte man aus der nächsten Zeile die zwei // entfernen, um den Ansichtsmodus „**ansicht**“ zu erstellen:

```
//auswahl = "3ddruck"; // oder "ansicht"
auswahl = "ansicht"; // oder "3ddruck"
```

Der 3DDruck-Modus rechts



Und unten der Ansichts-Modus



010-20

```
// Tuerme von Hanoi

$fn = 100;

// Auswahlbereich
//auswahl = "3ddruck"; // oder "ansicht"
auswahl = "ansicht"; // oder "3ddruck"

// Programmteil
cube([200, 60, 10], center=true);
  for (x = [-60:60:60]) {
  translate([x, 0, 5]) cylinder(h=70, r=4);
  } // Ende von: for (x = [-60:60:60]) {

  if (auswahl == "ansicht") {
    for (d = [2:1:7]) {
    translate([-60, 0, 10 + (7-d)*10]) scheibe(d*4, 5);
    } // Ende von: for (d = [2:1:7]) {
  } // Ende von: if (auswahl == "ansicht") {

else if (auswahl == "3ddruck") {
  for (d = [2:1:7]) {
    if (d > 4) {
    translate([60*d - 350, 60, 0]) scheibe(d*4, 5);
    }

    else {
    translate([60*d - 200, 100, 0]) scheibe(d*4, 5);
    } // Ende von: else {

  } // Ende von: for (d = [2:1:7]) {

} // Ende von: else if (auswahl == "3ddruck") {

// Modulbereich
module scheibe(scheiben_radius, loch_radius) {
  difference() {
  cylinder(h=10, r=scheiben_radius, center=true);
  cylinder(h=11, r=loch_radius, center=true);
  } // Ende von: difference() {
} // Ende von: module scheibe(scheiben_r ...
```