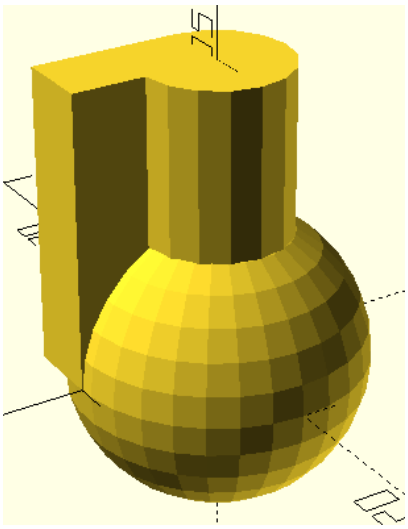
## Variable

OpenSCAD-Variablen werden durch eine Anweisung mit Name oder Bezeichner, Zuweisung über einen Ausdruck und ein Semikolon erstellt.

Die gültigen Bezeichner dürfen nur aus einfachen Zeichen und Unterstrichen, also „a-z“, „A-Z“, „0-9“ und „_“ bestehen und erlauben keine andere Zeichen.

Innerhalb eines Scriptes können mehrere Variable angegeben werden.
Achtung: OpenSCAD unterscheidet zwischen Groß- und Kleinschreibung!

```
cylinder(h=20, r=5);
cube([5, 10, 20]);
sphere(10);
```



stattdessen kann auch geschrieben werden:

```
f=5;
t=10;
s=20;
cylinder(h=s, r=f);
cube([f, t, s]);
sphere(t);
```

In beiden Fällen ergeben die Scripte das gleiche Ergebnis wie im Bild gezeigt.

Der Vorteil liegt darin, dass bei einer Änderung nur der Wert der Variable am Anfang des Scriptes getauscht werden muss. Alle Maße innerhalb des Scriptes werden auf einmal geändert wiedergegeben.

```
f=5;
t=10;
s=20;
cylinder(h=s, r=f);
cube([f, t, s]);
sphere(T);
```

ergibt die Fehlermeldung: WARNING: Ignoring unknown variable 'T' in file , line 6

Die Variable „t“ wurde angegeben, jedoch „T“ nicht. Ändern wir dies ab, verschwindet die Fehlermeldung und es wird korrekt angezeigt.

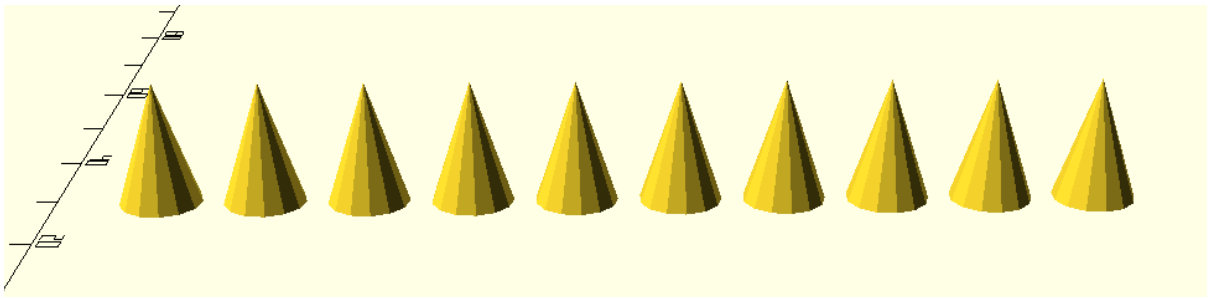
008-02

## for ( Variable = [ Anfangswert : Schrittweite : Endwert ] ] { ….. }

**Schleifen erzeugen**

Der „for“-Befehl erzeugt eine Schleife in der einer oder mehrere Befehle vorkommen können. Diese wird sooft durchlaufen und ausgeführt, bis sie das Ende erreicht hat.

Es sollen 10 Kegel in einer Reihe dargestellt werden:



Diese einfache Lösung birgt beim Kopieren und Einfügen, sowie beim Ändern der Werte, viele Fehlerquellen:

```
translate([10, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([20, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([30, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([40, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([50, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([60, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([70, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([80, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([90, 30, 0]) cylinder(h=12, r1=4, r2=0);
translate([100, 30, 0]) cylinder(h=12, r1=4, r2=0);
```

Die einzige Änderung von einem Kegel zum nächsten ist die Erhöhung der Position entlang der X-Achse : Die x-Position des ersten Kegels ist 10, die x-Position des zweiten Kegels ist 20 und so weiter, bis der letzte Kegel gezeichnet ist: eine x-Position von 100.

Anstatt 10 separate Code-Zeilen zu schreiben, kann eine einzelne for-Schleife verwenden werden um die ganze Sammlung von Kegeln zu erzeugen.

Der folgende Pseudocode zeigt die Syntax der for-Schleife:

```
for (variable = [Anfangswert : Schrittweite : Endwert]) {
// ein oder mehrere Befehle zum Wiederholen
}
```

Das Schlüsselwort „for“ gibt an, dass OpenSCAD Anweisungen in einer Schleife wiederholt werden sollen.
Dazu wird eine Variable erstellt, um den sich ändernden Wert jeder Wiederholung zu verfolgen. Die Variable hat einen Anfangswert, eine Schrittweite und einen Endwert. Alles wird in geschweifte Klammern {  } gestellt, um alle zu wiederholende Anweisungen einzuschließen.

Das folgende Beispiel verwendet eine einzelne for-Schleife, um 10 Kegel zu zeichnen, anstatt 10 separate Zeilen zu verwenden:

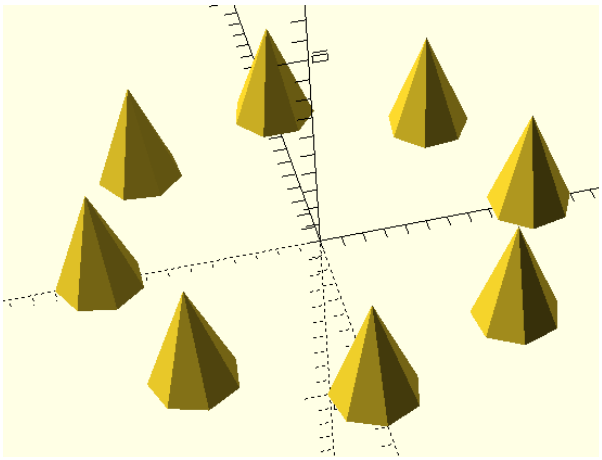
```
for (x_pos = [10:10:100]) {
translate ([x_pos, 30, 0]) cylinder(h=12, r1=4, r2=0);
}
```

Eine Variable namens „x_pos“  verfolgt die Position jedes Kegels. Diese Variable hat einen Anfangswert von 10;  jedesmal wenn die for-Schleife wiederholt wird, erhöht sich der Wert von x_pos um 10, so dass der nächste Kegel 10 Einheiten weiter entlang der X-Achse gezeichnet wird.
Sobald x_pos gleich 100 ist, wird der letzte Kegel gezeichnet und die Schleife wird beendet.

Das Ergebnis sieht genauso aus wie die Verwendung von 10 separaten Anweisungen, wie im oberen Bild gezeigt wird.

008-04

Das Bild zeigt einen Kegel, der sich in einem Rotationsmuster um die Z-Achse wiederholt:



Es werden zwischen den geschweiften Klammern bei jedem Durchgang der Schleife drei Befehle hintereinander ausgeführt:

008-Kegelreihe rund.scad

```
for (drehung=[0:45:315]) {
    rotate([0, 0, drehung])
    translate([10, 0, 0])
   cylinder(h=5, r1=2, r2=0);
}
```

Innerhalb der geschweiften Klammern erzeugt die Schleife einen Kegel, versetzt ihn um 10 Einheiten entlang der x-Achse und dreht ihn dann um Winkelgrade der Variable „drehung“.

Der erste Kegel wird gezeichnet, wenn der Wert der Winkelvariablen 0 ist - also nicht gedreht wurde, gleich als ersten Kegel.
Der Wert der Winkelvariable „drehung“ erhöht sich bei jedem Schleifen-durchgang um 45 Grad, wobei jeder Kegel dann entsprechend gedreht wird.

Der letzte Wert der Winkelvariablen ist 315, also wird der letzte von der Schleife gezeichnete Kegel um 315 Grad um die Z-Achse gedreht.

## module ModulName ( Parameter ) { ….. }

**mehrfach aufrufbare separierte Programmteile**

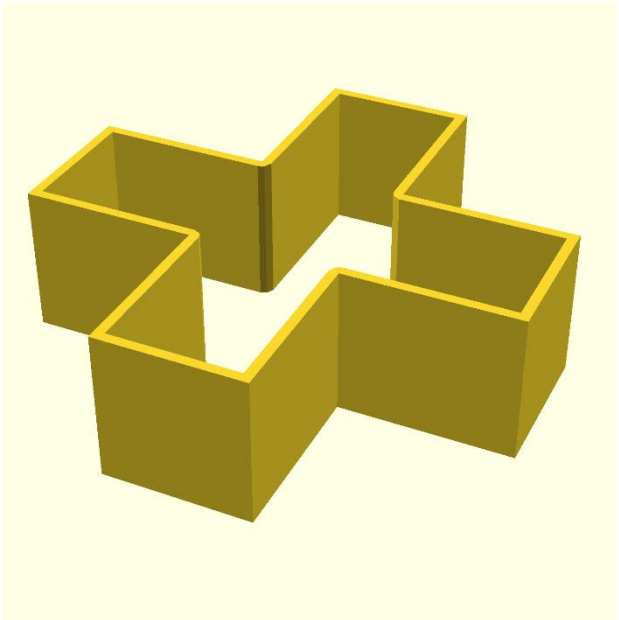
Module sind einfache Programmabschnitte, die mehrfach aufgerufen werden können. Sollte der Script lang und länger werden, können einzelne Abschnitte in Module separat abgespeichert werden. Dies hat den Vorteil, der Script wird um einiges übersichtlicher und besser lesbar.

Das Beste ist jedoch, wenn eine bestimmte Form doppelt oder mehrfach genutzt werden soll, wird dies als Modul abgelegt. Der Script wird also dadurch kürzer.

Ferner können beim Aufruf der Module auch Variable mit übergeben werden, die das Modul verarbeiten kann.

**Vereinfachen des Codes mit Modulen**

Um die Verwendung von Modulen besser verstehen zu können, wird nochmals gebacken: die Backform mit dem kreuzförmigen Ausstecher.



Hier nochmal den Script dazu.

Sind die Wiederholungen sichtbar?

```
linear_extrude(30) {
  difference() {
    union() {
      square([100, 30], center=true);
      square([30, 100], center=true);
    }
    offset(-2) {
      square([100, 30], center=true);
      square([30, 100], center=true);
    }
  }
}
```

Der Ausstecher wird über die Differenz zweier Kreuze hergestellt, dadurch werden die quadratischen Befehle zum Erstellen der Kreuzform zweimal wiederholt.

008-06

Doppelter Code verursacht fast immer Probleme, da bei Änderungen sich gern Fehler einschleichen.

Für diesen Zweck wird „module“ verwendet. Die Kreuzform wird einmal als Modul geschrieben und kann zweimal aufgerufen werden. Der Befehl für „module“ lautet:

```
module ModulName() {
// Script zur Formerstellung
}
```

Die Definition eines Modules beginnt mit „module“ gefolgt von einem aussagekräftigen Namen z.B. „kreuzform“. Dire Modulnamen besitzen die gleichen Einschränkungen wie die Variablennamen: es dürfen nur Klein- und Großbuchstaben, Unterstriche oder die Ziffern 0 bis 9 verwendet werden.

Daran werden zwei runde Klammern ( ) angefügt, gefolgt vom Scriptteil in in geschweiften Klammern. Der Code zwischen den geschweiften Klammern unterscheidet sich nicht von anderem OpenSCAD-Code.

Die Moduldefinition wird als separater Abschnitt als eigenständigen Datei im Entwurf stehen. Ist ein Modul definiert, kann es die neue Form nicht wirklich zeichnen. Es ist wie ein Rezept zu verstehen, das die Erstellung der Form beschreibt. Um die Form darzustellen, muss das Modul namentlich aufgerufen werden:

```
ModulName();
```

Ein Modul ist ein Beispiel für eine vom Programmierer definierte Form. Eigentlich sind alle bisher verwendete OpenSCAD-Befehle, wie Kugel, Zylinder, und linear_extrude, in die Sprache eingebaute Module!

Es wird ein neuer Script für den Ausstecher geschrieben, indem ein Kreuzmodul erstellt wird: (008-offset02.scad)

```
module kreuzform() {
square([100, 30], center=true);
square([30, 100], center=true);
} // Ende module kreuzform

linear_extrude(30) {
    difference() {
     kreuzform();
     offset(-2) kreuzform();
     } // Ende difference
} // Ende linear_extrude
```

Dieses Script ergibt den selben Ausstecher wie der weiter vorn.

Am Einfachsten wird der Modul-Teil an den Anfang des Scriptes gesetzt. Ihm wird zur Beschreibung der Form passend der Namen „kreuzform“ zuteil. In den geschweiften Klammern kommt der Script der die Form des Kreuzes definiert.

Durch den Aufruf des Modulnamens mit ( ) innerhalb des Scriptes wird OpenSCAD angewiesen das Kreuz zu zeichnen.
In der nächsten Zeile wird über „offset(-2) kreuzform();“ das kleinere Kreuz erstellt und mittels difference vom großen Kreuz abgezogen.

Fertig ist die Ausstecher-Form.

Übersicht eines Scriptes:

```
// Scriptanfang

module MeinName() {
// Inhalt des Modules
} // Ende module MeinName

// eigentlicher Script
….
….
MeinName();
….
….
MeinName();
….
….
MeinName();
….
```

Somit lassen sich über nur eine einzelne Zeile das Modul öfters aufrufen.
Es können selbstverständlich mehrere Module mit unterschiedlichen Namen eingesetzt werden.

008-08

## Der Einsatz von Parametern in Modulen

Integrierte OpenSCAD-Module wie sphere können einen Parameter wie sphere(r=30); annehmen, wobei der Parameter den Radius der Kugel angibt.

Solche Parameter können auch den Modulen zugefügt werden.

Der folgende Code zeigt die vollständige Syntax zur Angabe eines Moduls mit Parameter:

```
module ModulName(ParameterName = Wert, ...) {
// Zeilen zur Erstellung von Formen
}
```

Anstatt die Klammern nach der Moduldefinition leer zu lassen, werden hier die Parameternamen – Platzhalter für einen Wert – eingetragen.

Um mehrere Parameter zu erstellen werden mehrere Parameternamen angegeben, getrennt durch Kommata. Es ist sehr wichtig, dass jeder Parameter einen anderen Namen bekommt.

Parameter sehen fast aus wie Variable. In der Tat, innerhalb eines Moduls verhalten sich Parameter wie Variable. Es empfiehlt sich, Parametern passende Namen zu geben, die ihren Zweck beschreiben.
Wie bei Variablen und Modulnamen, dürfen für Parameternamen nur Buchstaben, Unterstriche, oder Zahlen verwendet werden.

Das Listing zeigt, wie dem kreuzform-Modul Parameter hinzugefügt werden:

```
module kreuzform(breite=30, laenge=100) {
square([laenge, breite], center=true);
square([breite, laenge], center=true);
}
```

Innerhalb der Klammern werden die Breiten- und Längenparameter hinzugefügt, die die Breite und Länge jedes Kreuzarms definieren.

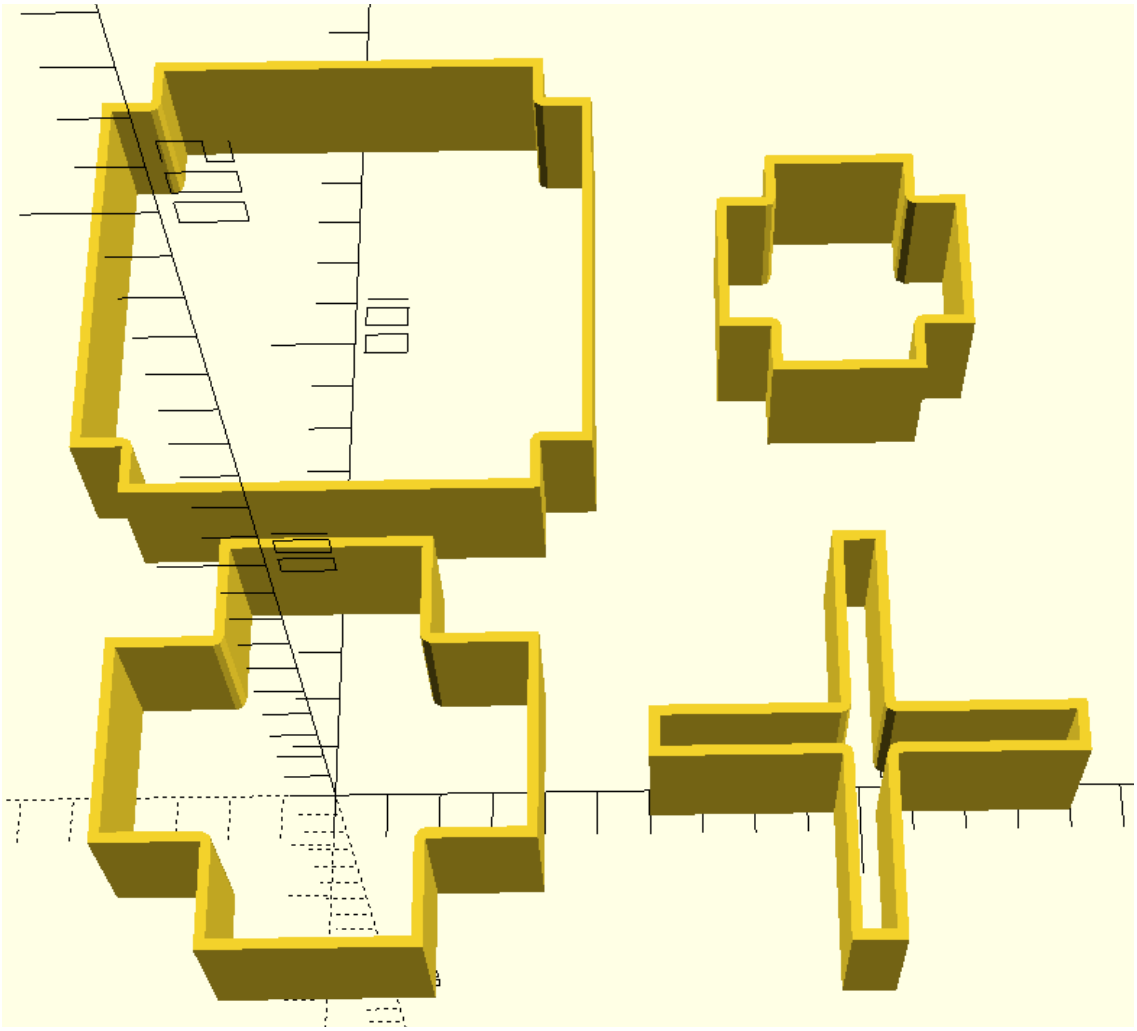
Um eine Kreuzform mit dem Kreuzmodul zu erstellen, werden bestimmte Werte für jeden Parameter jedes Mal angegeben.

```
linear_extrude(30) {
  difference() {
  kreuzform(20, 100);
  offset(-2) kreuzform(20, 100);
  }
}
```

Die Reihenfolge der Zahlen gibt die Breite und die Länge des Kreuzes an.
Da der Breitenparameter in der Definition des Moduls an erster Stelle steht, wird die erste Zahl in Klammern dem Breitenparameter und die zweite Zahl dem Längenparameter zugewiesen.

Werden beim Aufruf des Moduls **keine Parameter** mit angegeben (), so übernimmt OpenSCAD die vorgegebenen Parameter im Modul.

Werden im Aufruf mit Modulnamen jedoch Parameter **mitangegeben**, so **überschreiben** diese die Vorgaben im Modul!



Somit kann mit einem Modul verschiedene Kreuze erstellt werden.
Hier der Script mit variabler Ausführung zum vorhergehenden Bild:

```
module kreuzform(breite=30, laenge=100) {
   square([laenge, breite], center=true);
   square([breite, laenge], center=true);
}

linear_extrude(30) {
   difference() {
      kreuzform(40, 80);
      offset(-2) kreuzform(40, 80);
   }
}

translate([100,0,0])linear_extrude(30) {
   difference() {
      kreuzform(10, 80);
      offset(-2) kreuzform(10, 80);
   }
}

translate([100,100,0])linear_extrude(30) {
   difference() {
      kreuzform(30, 50);
      offset(-2) kreuzform(30, 50);
   }
}

translate([0,100,0])linear_extrude(30) {
   difference() {
      kreuzform(100, 80);
      offset(-2) kreuzform(100, 80);
   }
}
```

Bei diesem Beispiel sieht man sehr gut, dass die Werte beim Modul-Aufruf ändern. So werden auch die Formen der Ausstecher verändert.

Wie bereits gesagt, die Werte beim Auf des Modules überschreiben die vorgegebenen. Nur wenn keine Angaben gemacht werden, also die Klammern leer bleiben ( ) , werden die Vorgaben des Moduls verwendet.

Jederzeit können ebenso die Parameter-Namen den Werten vorangestellt werden:

```
…
kreuzform(breite=100, laenge=40);
offset(-2) kreuzform(breite=100, laenge=40);
…
```

Es bleibt alles gleich.

Bei der Angabe der Parameternamen beim Aufruf können diese auch vertauscht werden, solange die Namen mit den Werten übereinstimmen:

```
…
kreuzform(laenge=40, breite=100);
offset(-2) kreuzform(breite=100, laenge=40);
…
```

008-12

## use < ….. >

**Einbinden von Programmteilen**

Wenn ein neues Design erstellt wird, wird manchmal eine Komponente aus einem früheren Projekt benötigt.

Eine gute Möglichkeit dafür besteht darin, die Komponente zu einem Modul zu machen. Durch das Einfügen der Moduldefinition in eine separate Datei kann diese anschließend problemlos in beiden Scripten verwendet werden.

Das separate Speichern von Modulen hilft sie leichter wiederzufinden und ebenso in vielen Projekten zu verwenden.

Das Organisieren von Moduldefinitionen in separaten Dateien wird oft als Erstellen einer Bibliothek bezeichnet, insbesondere wenn mehrere zugehörige Module in einer neuen Datei definiert sind.

Um zu erfahren, wie ein Modul in einer separaten Datei gespeichert wird, wird das kreuzförmige Ausstecher-Design in zwei Dateien aufgeteilt.

Eine Datei wird zum Definieren der Kreuzform gespeichert.
Die zweite Datei verwendet zur Erstellung der Form dann dieses Modul.

Zunächst werden zwei leere OpenSCAD-Dateien erstellt:
„kreuzform-modul.scad“ und „ausstecher.scad“

Damit OpenSCAD beiden Dateien finden kann sollten beide im selben Ordner gespeichert sein. Außerdem wurden die Namen für ihren jeweiligen Zweck gewählt.

Speichern unter: **008-kreuzform_modul.scad**

| | |
|---|---|
| Nun wird in<br>**008-kreuzform_modul.scad**<br>die Moduldefinition kopiert: | module kreuzform() {<br>square([100, 30], center=true);<br>square([30, 100], center=true);<br>} // Ende module kreuzform |

Speichern unter: **008-ausstecher.scad**

| | |
|---|---|
| Nun wird am Anfang von<br>**008-ausstecher.scad**<br>noch eine Zeile eingefügt, die die Datei der Form aufruft. | use <008-kreuzform_modul.scad><br>linear_extrude(30) {<br>difference() {<br>kreuzform();<br>offset(-2) kreuzform();<br>}<br>} |

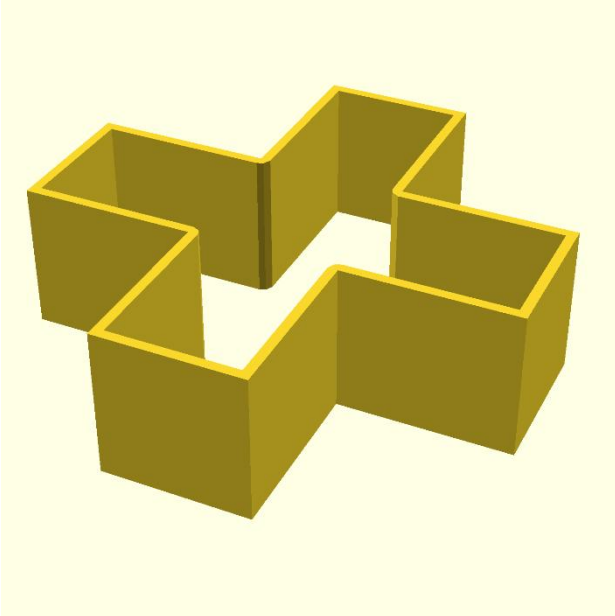
Anstatt die Moduldefinition in ausstecher.scad einzugeben, weist die erste Zeile OpenSCAD an, Code aus kreuzform_modul.scad zu verwenden.

Das Schlüsselwort „**use**“ weist OpenSCAD an, die Module von einer anderen Datei zu laden. Die Syntax für das Schlüsselwort use lautet wie folgt:

**use <verzeichniss/nocheinverzeichnis/dateiname.scad>**

Nach dem Schlüsselwort use folgen spitze Klammern < > zwischen denen der Namen der .scad-Datei steht.

Sollten sich die Dateien zum Nachladen nicht im gleichen Verzeichnis wie die Hauptdatei befinden, wird einfach der Pfad mit Verzeichnisnamen zur Datei angegeben.



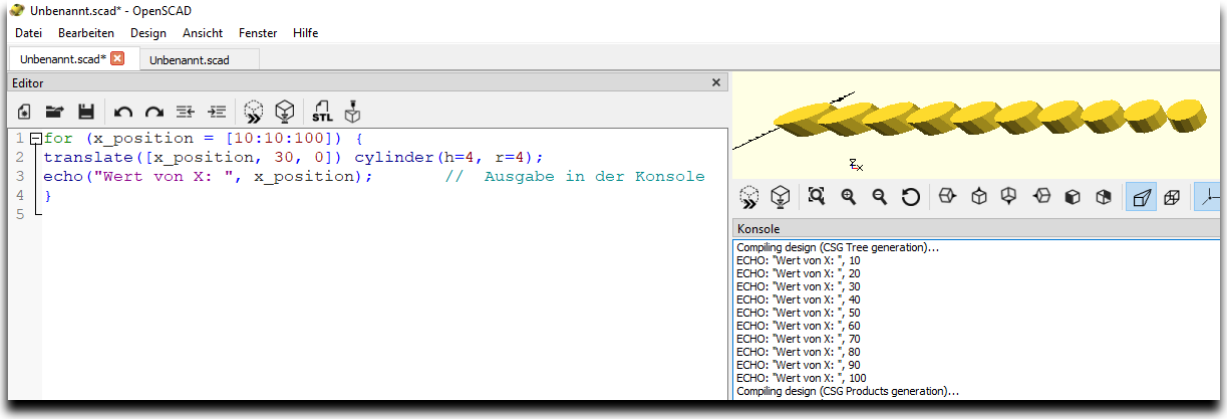
008-14

## echo ( „Mein Text“, Variable );

### Fehlersuche in Schleifen mit echo

Manchmal ist es nützlich, den Wert einer Variablen zu untersuchen, während sie sich in den Wiederholungen einer for-Schleife befindet und sich ständig verändert.
Die echo-Funktion zeigt die aufeinanderfolgende Werte einer Variablen im Konsolenfenster zur Überprüfung an:

```
for (x_position = [10:10:100]) {
translate([x_position, 30, 0]) cylinder(h=4, r=4);
echo("Wert von X: ", x_position);          //  Ausgabe in der Konsole
}
```




ECHO: "Wert von X: ", 10
ECHO: "Wert von X: ", 20
ECHO: "Wert von X: ", 30
ECHO: "Wert von X: ", 40
ECHO: "Wert von X: ", 50
ECHO: "Wert von X: ", 60
ECHO: "Wert von X: ", 70
ECHO: "Wert von X: ", 80
ECHO: "Wert von X: ", 90
ECHO: "Wert von X: ", 100

Zusätzlich zur Ausgabe der Zylinder im Vorschaufenster wird in der Konsole die Zeile ECHO bis zum Ende der Schleife ausgegeben. Die ECHO-Funktion ist bei der Fehlersuche im Script sehr hilfreich. In der Konsole können alle Werte der Variablen, die die Wiederholungen der Schleife beeinflussen, angezeigt werden. Außerdem können Texte zur Erklärung ( wie „Wert von X: „ ) ergänzt werden. Texte und Variable sollten durch Kommas (,) getrennt werden.