2D – FORMEN

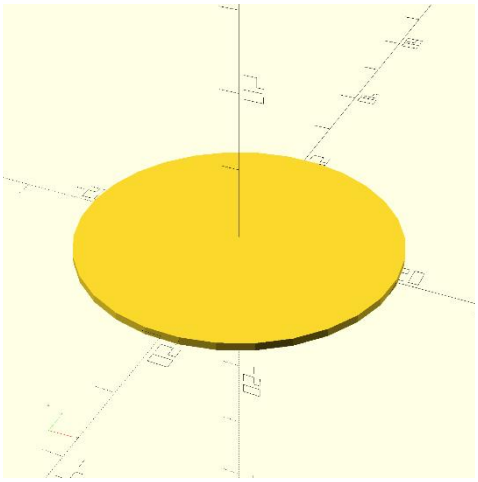
Wie bei 3D-Formen kann auch mit 2D komplexe Formen erstellt werden. Einfache 2D-Formen sind: Kreis, Quadrat und Polygon.

circle (radius); oder circle (d = Durchmesser);

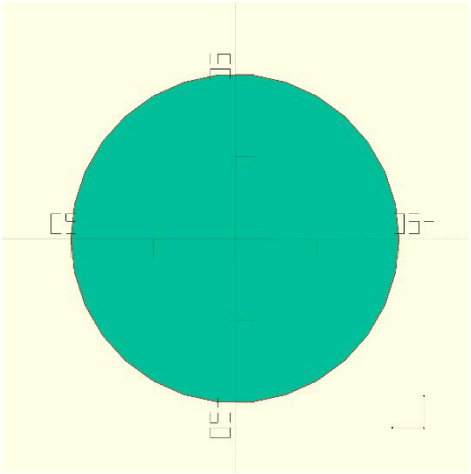
Kreise zeichnen

Mit dem Befehl „circle“ können 2D-Kreis gezeichnet werden, indem man seinen Radius angibt, ähnlich dem Befehl „sphere“ (Kugel).
Das folgende Beispiel zeichnet einen Kreis mit einem Radius von 20

```
circle(20);
```



Dieser 2D-Kreis wurde über F5 (Vorschau) dargestellt



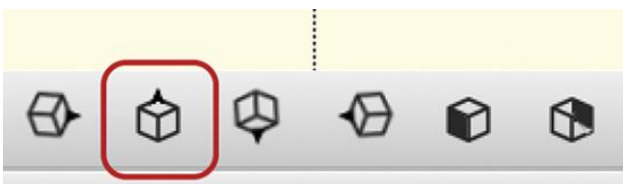
Dieser 2D-Kreis wurde über F6 bereits gerendert

Da 2D-Formen keine Höhe haben, existieren sie nur in der XY-Ebene.

Um 2D-Formen in ihrer wahren Form ohne deren Höhe anzuzeigen, verwenden Sie die Schaltfläche Rendern. (Beachten Sie, dass es nicht möglich ist, 2D- und 3D-Formen im Render-Modus zu mischen.)

Da 2D-Formen keine Tiefe haben, ist es oft am einfachsten, 2D-Designs damit zu erstellen.

Verwendet wird dafür das Symbol „Draufsicht“ in der Symbolleiste
Bild 3.3



006-02

square (X-Wert, Y-Wert, center=true);

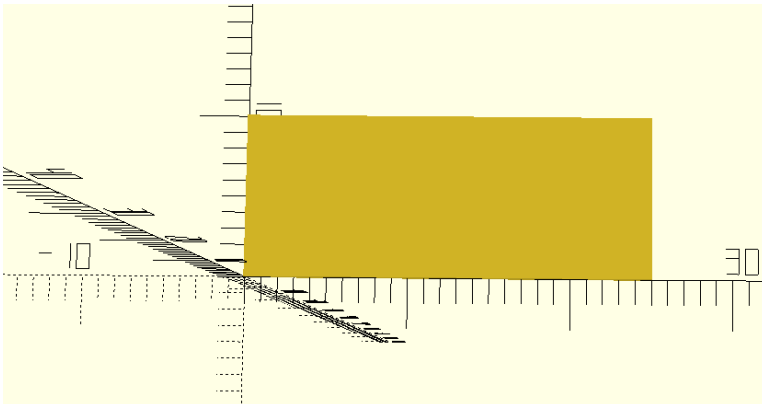
Rechtecke zeichnen

Der Befehl „square“ zum Rechtecke zeichnen, gibt die x- und y-Abmessungen als einzelne Werte an.

Die folgende Anweisung zeichnet ein Rechteck, das 25 entlang der x-Achse und 10 entlang der y-Achse sich erstreckt:

```
square([25, 10]);
```

Ein Rechteck 25 x 10



Mit dem Befehl „square“ gefolgt von ([…]) und ; wird ein Rechteck gezeichnet. Innerhalb der eckigen Klammern werden die Abmessungen, getrennt durch ein Komma, angegeben. Diese 2D-Form erfordert nur die X- und Y-Werte.

Die erste Zahl der Werte repräsentiert die Breite des Quadrats entlang der X-Achse, die zweite Zahl dagegen die Länge des Quadrats entlang der X-Achse.

Mit dem Angang center=true wird das Rechteck auch im Nullpunkt zentriert.

Transformationen und Boolsche Ausführungen an 2D-Formen

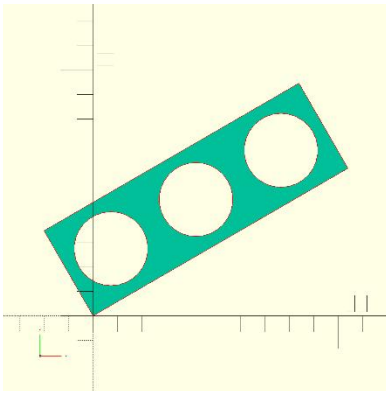
Die bisher gelernten Transformationen und boolschen Ausführungen von 3D-Formen können ebenso auf die 2D-Formen übertragen werden.

Der einzige und kleine Unterschied ist, dass von den drei Achsen nur zwei benötigt werden: X- und Y-Koordinaten. Der Rotationsbefehl erfordert hingegen nur einen einzigen Drehwinkel (den für die Z-Achse).

006-03

```
rotate(30) {
difference() {
square([120, 40]);
translate([20, 20]) circle(15);
translate([60, 20]) circle(15);
translate([100, 20]) circle(15);
}
}
```

Der oben gezeigte Script erzeugt das folgende Design, Es wird translate, difference und rotate verwendet um ein gedrehtes Rechteck mit drei ausgeschnittenen Kreisen zu zeichnen
Bild:



Genau wie bei den 3D-Formen wirkt sich die Reihenfolge, in der die Befehle auf eine 2D-Form angewendet wird, auf die Anordnung und die Platzierung der resultierenden Form aus.

Subtrahieren eines Kreises von einem Quadrat im Vergleich zum Subtrahieren eines Quadrats von einem Kreis.

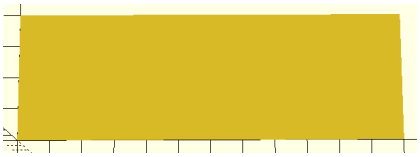
Selbstverständlich funktionieren **difference** / **translate** / **intersection** und **union** genauso mit 2D-Objekten, wie in 3D.

006-04

A.

square([120, 40]);

es wird ein Rechteck mit den angegebenen Maßen gezeichnet



B.

difference() {

square([120, 40]);

translate([20, 20]) circle(15);

}

Davon wird mittels difference ein Kreis abgezogen

C.

difference() {

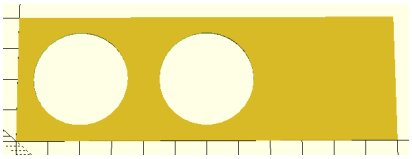
square([120, 40]);

translate([20, 20]) circle(15);

translate([60, 20]) circle(15);

}

Ein weiterer Kreis wird zusätzlich abgezogen



D.

difference() {

square([120, 40]);

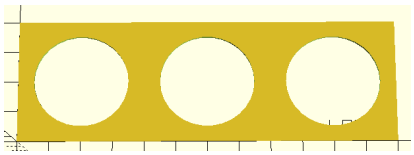
translate([20, 20]) circle(15);

translate([60, 20]) circle(15);

translate([100, 20]) circle(15);

}

Noch ein weiterer Kreis wird abgezogen



E.

rotate(30) {

difference() {

square([120, 40]);

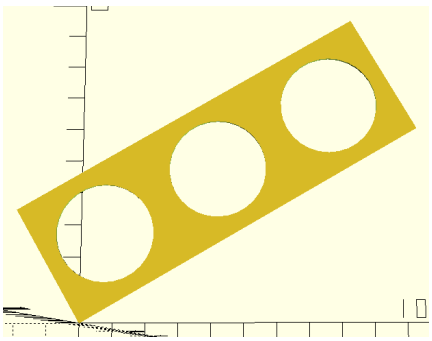
translate([20, 20]) circle(15);

translate([60, 20]) circle(15);

translate([100, 20]) circle(15);

}

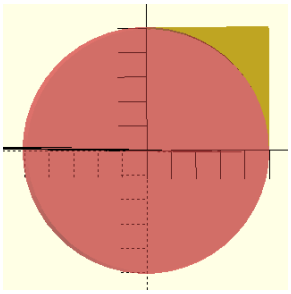
}

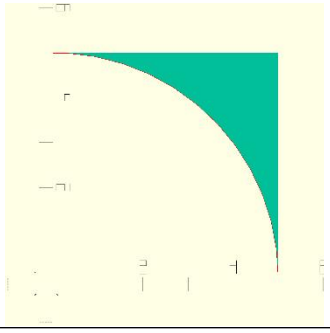


Zum Schluß wird das Ganze noch um 30 Grad gedreht.

Die folgende Differenzoperation subtrahiert einen Kreis von einem Quadrat:

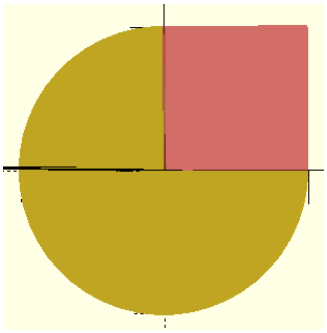
```
difference() {
square([5, 5]);
circle(5, $fn=50);
}
```

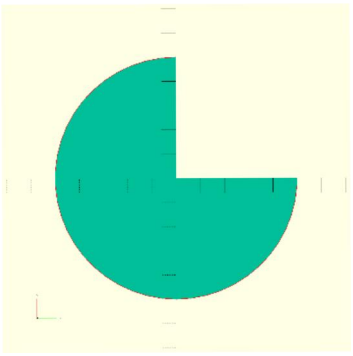




Und dieser **difference**-Befehl subtrahiert ein Quadrat von einem Kreis:

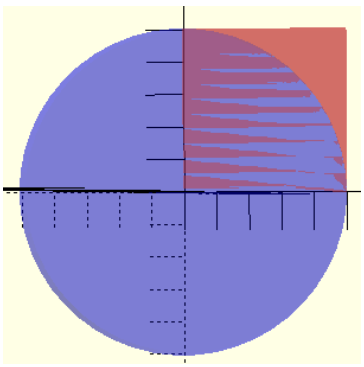
```
difference() {
circle(5, $fn=50);
square([5, 5]);
}
```

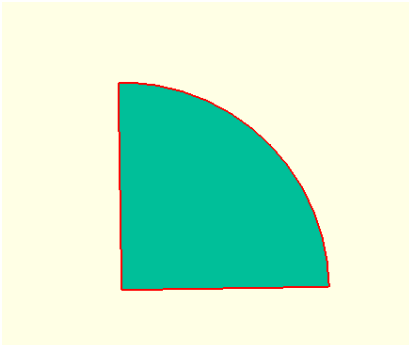




Hier wird die Schnittmenge (**intersection**) dargestellt:

```
intersection() {
circle(5, $fn=50);
square([5, 5]);
}
```





Auf der linken Seite werden beide Objekte dargestellt. Rechts dagegen sieht man nur noch das Ergebnis.

006-06

polygon ([x1, y1], [x2, y2], ….. [xn, yn]);

Vieleck/Polygon zeichnen

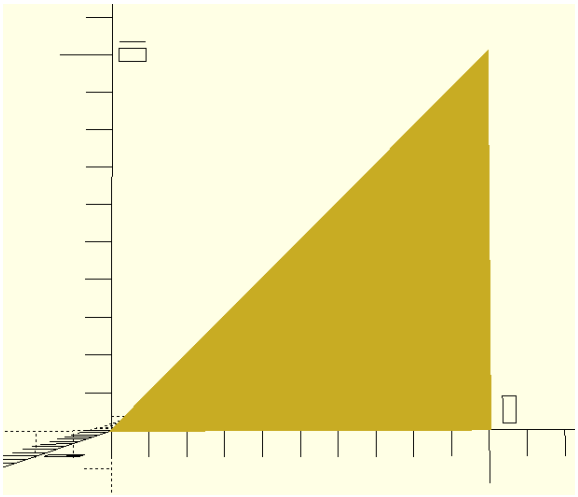
Um eine 2D-Form zu erstellen, die nicht im Umfang von OpenSCAD vorhanden ist, steht der Befehl „polygon“ bereit!

Der folgende Befehl verwendet polygon, um ein Dreieck mit den Scheitelpunkten bei [0, 0], [10, 0] und [10, 10] zu zeichnen

```
polygon([ [0, 0], [10, 0], [10, 10] ]);
```

Ein Dreieck,
erstellt durch den Befehl „polygon“

006-01



Ein Polygon wird durch eine Liste der Eckpunkte der Form definiert. Jeder Eckpunkt in dieser Liste ist ein Vektor, der die Koordinaten eines Eckpunktes des Vielecks enthält.

Einfach jeden Eckpunkt als Vektor (= 2 Werte) in eckigen Klammern gesetzt und zusätzlich das Ganze in ([und]) gepackt und mit ; abgeschlossen, ergibt bereits den kompletten Befehl.

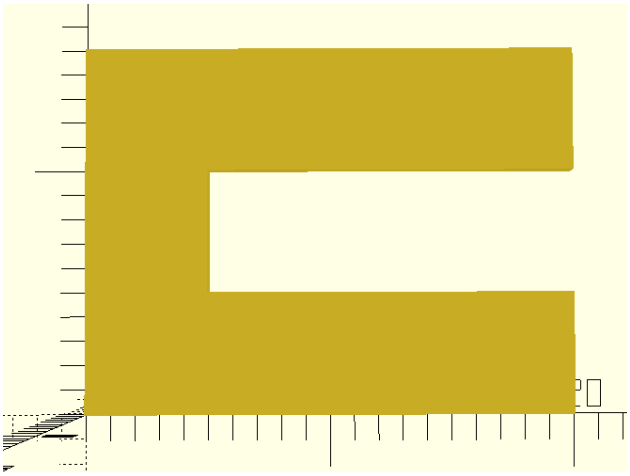
Die Reihenfolge der Eckpunkte ist wichtig. Daher so auflisten, als würde man um die Form herumgehen. Der Startpunkt ist immer gleichzeitig der Endpunkt, daher muss dieser Punkt nicht nochmals angegeben werden. Dies macht OpenSCAD von selbst!

Da Polygone eine beliebige Anzahl von Eckpunkten haben können, können immer komplexere Formen erstellt werden, wie z.B. diese mit acht Eckpunkten:

006-06-2

```
polygon([
[ 0, 0], [20, 0],
[20, 5], [ 5, 5],
[ 5, 10], [20, 10],
[20, 15], [ 0, 15]
]);
```

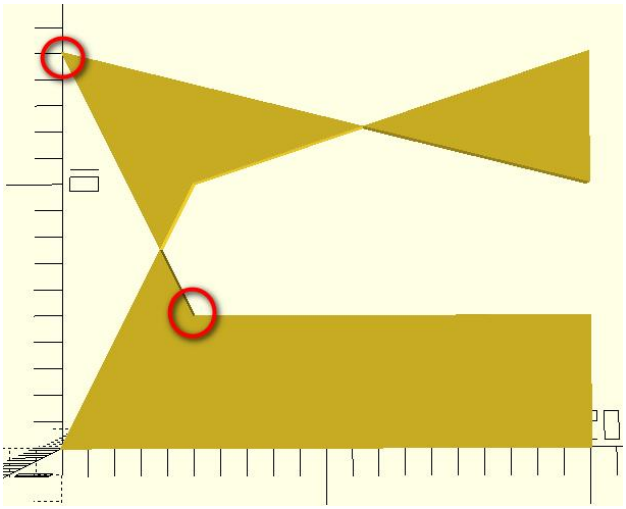
Eine etwas komplexere Form mittels „polygon“ erstellt:



Ein fataler Fehler:
Aus Versehen wurden zwei Werte vertauscht: Wert [5, 10] mit [0, 15]

```
// Fataler Fehler:
polygon([
[ 0, 0], [20, 0],
[20, 5], [ 5, 5],
[ 0, 15], [20, 10],
[20, 15], [ 5, 10]
]);
```

So lässt es sich am ehesten beweisen, dass man die Reihenfolge unbedingt einhalten sollte.

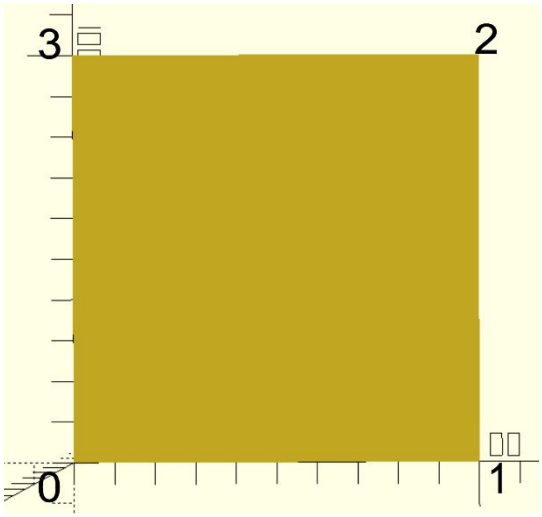


006-08

polygon (points= [[x1, y1], [x2, y2] … [xn, yn]], paths= [[0, 1, 2, ….n]], convexity=10);

Polygon mit Durchbrüchen

Der Befehl lautet:

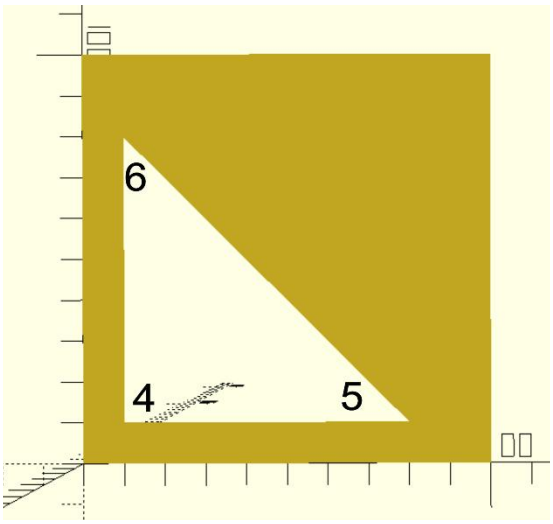


polygon(points=[
hier wird ein großes Quadrat mit den Punkten (points) : [0,0],[100,0],[100,100], [0,100] erzeugt.
],
paths=[
diese vier Punkte werden durch Pfade (paths) verbunden und automatisch mit den Zahlen 0 bis 3 nummeriert: [0,1,2,3] und als geschlossene Fläche dargestellt.
],convexity=10);

So entsteht das Quadrat aus den roten Zahlen. 006-08

```
polygon
 (points=[
 [0,0],[100,0],[100,100],[0,100]
 ],
 paths=[
 [0,1,2,3]
 ],convexity=10);
```

Nun soll ein Dreieck ausgeschnitten werden. Es werden einfach noch drei Punkte und ein Komma mehr angegeben: **, [10,10],[80,10], [10,80].**
Und bei den Pfaden werden weitere drei Pfade mit einem Komma angefügt: **,[4,5,6]**. Da diese drei separat in eckigen Klammern [ ] stehen, werden die Pfade wie bei „difference“ behandelt: vom Ersten abgezogen!



So wird nun das Dreieck mit den blauen Zahlen ausgeschnitten:

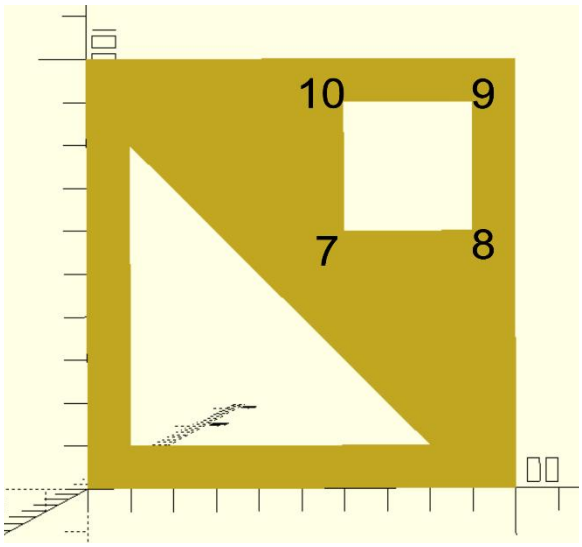
006-09

```
polygon
(points=[
 [0,0],[100,0],[100,100],[0,100],
 [10,10],[80,10],[10,80]
 ],
paths=[
 [0,1,2,3] ,
 [4,5,6]
 ],convexity=10);
```

Als letztes wird noch ein kleines Quadrat ausgeschnitten:

Wieder vier neue Punkte mit einem Komma, diesmal in grün:
, [60,60],[90,60],[90,90],[60,90]

Diese werden ebenso als Pfade mit Nummern versehen:
,[7,8,9,10].
Somit wäre das kleine Quadrat auch festgelegt. Wie bei „difference“ wird auch dieses ausgeschnitten.



So sieht nun der komplette Script aus:

006-polygon12.scad

```
polygon(points=[
 [0,0],[100,0],[100,100],[0,100],
 [10,10],[80,10],[10,80] ,
 [60,60],[90,60],[90,90],[60,90]
 ],
paths=[
 [0,1,2,3],
 [4,5,6] ,
 [7,8,9,10]
 ],convexity=10);
```

006-10

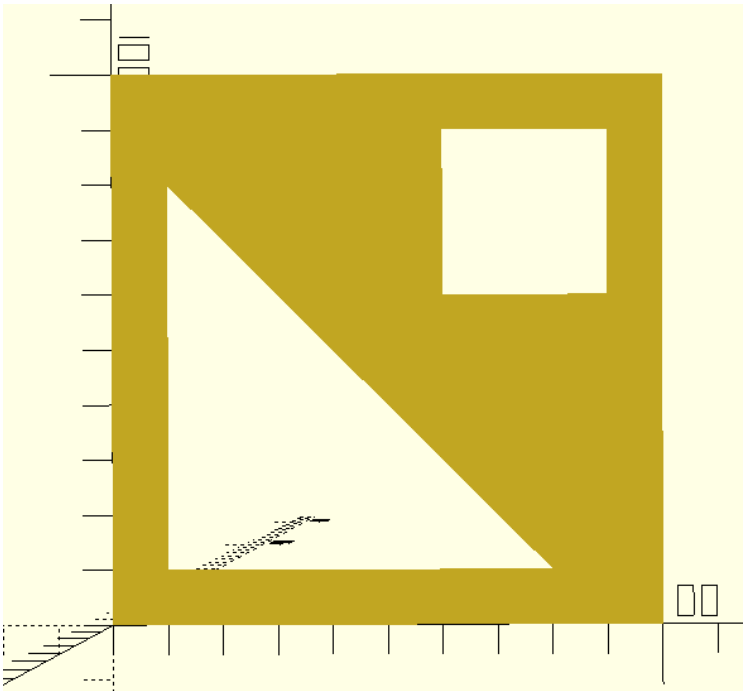
Unter points werden nur die Punkte angegeben, jeweils in eckigen Klammern durch Kommata getrennt. Hier ist nur die Zählung wichtig.

```
points=[
 [0,0],[100,0],[100,100],[0,100],    =  Punkt 0,1,2,3
 [10,10],[80,10],[10,80] ,           = Punkt 4,5,6
 [60,60],[90,60],[90,90],[60,90]     = Punkt 7,8,9,10
 ],
```

Genauso wichtig sind dafür bei den Pfaden die eckige Klammerangaben, die die jeweiligen Pfade einschließen:

```
paths=[
 [0,1,2,3],
 [4,5,6],
 [7,8,9,10]
 ],convexity=10);
```

Die Pfade [0,1,2,3] sind in eckigen Klammern eingeschlossen und bezeichnen damit das erste Element.
Das zweite Element, das Dreieck, [4,5,6] wird davon subtrahiert.
Das gleiche gilt für das dritte Element, das kleine Quadrat: [7,8,9,10]. Auch dieses wird vom ersten subtrahiert.



text („ “); **Texte zeichnen**

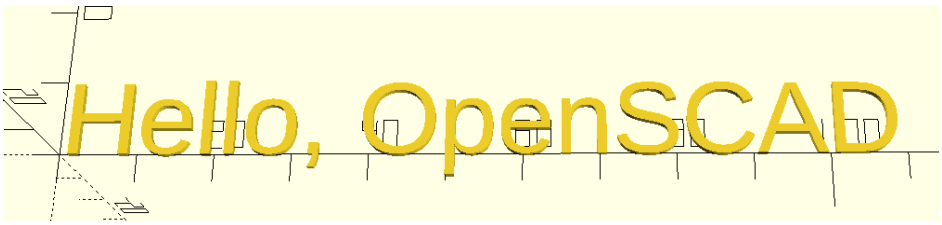
Eine weitere Art 2D-Formen zu erstellen, besteht darin, symbolische Muster zu erstellen, wie z.B. Wörter oder nur Zeichen.

Es können auch Emoji-Schriftarten verwendet werden, um auf vorgezeichnete Symbole zuzugreifen, oder einfach eine Versions- oder Seriennummer in das Design stempeln.

Ein Text in OpenSCAD (wie auch in anderen Programmiersprachen) wird als Zeichenfolge betrachtet. Da eine Zeichenkette beliebig lang sein kann, wird das Zitat (" ") als Markierung genutzt. Textzeichenfolgen können Buchstaben, Satzzeichen, Zahlen und (falls die Schriftart verwendete Unicode unterstützt) Emoji-Zeichen.

Diese Anweisung erstellt die Zeichenfolge „Hello, OpenSCAD“

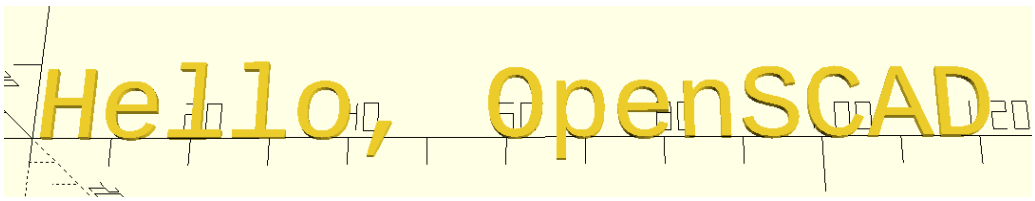
```
text("Hello, OpenSCAD", size=10);
```



Der Größenparameter „size“ ist für Textformen optional. Wird die Größe weglassen, ist die Standardtextgröße 10. Ein weiterer optionaler Parameter zum Zeichnen von Textformen ist die Schriftart. Es können zum Zeichnen ebenso beliebige Schriftarten verwendet werden die auf dem Computer installiert sind.

Die folgende Aussage zeichnet eine Textfolge in der Schriftart Courier

```
text("Hello, OpenSCAD", font="Courier");
```



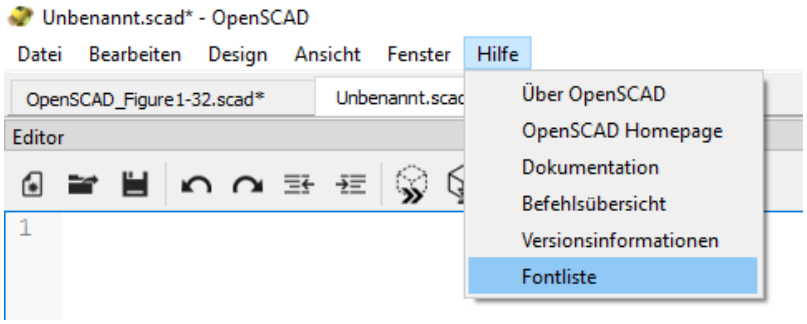
Hier wird nun die Schriftart Courier verwendet.

006-12

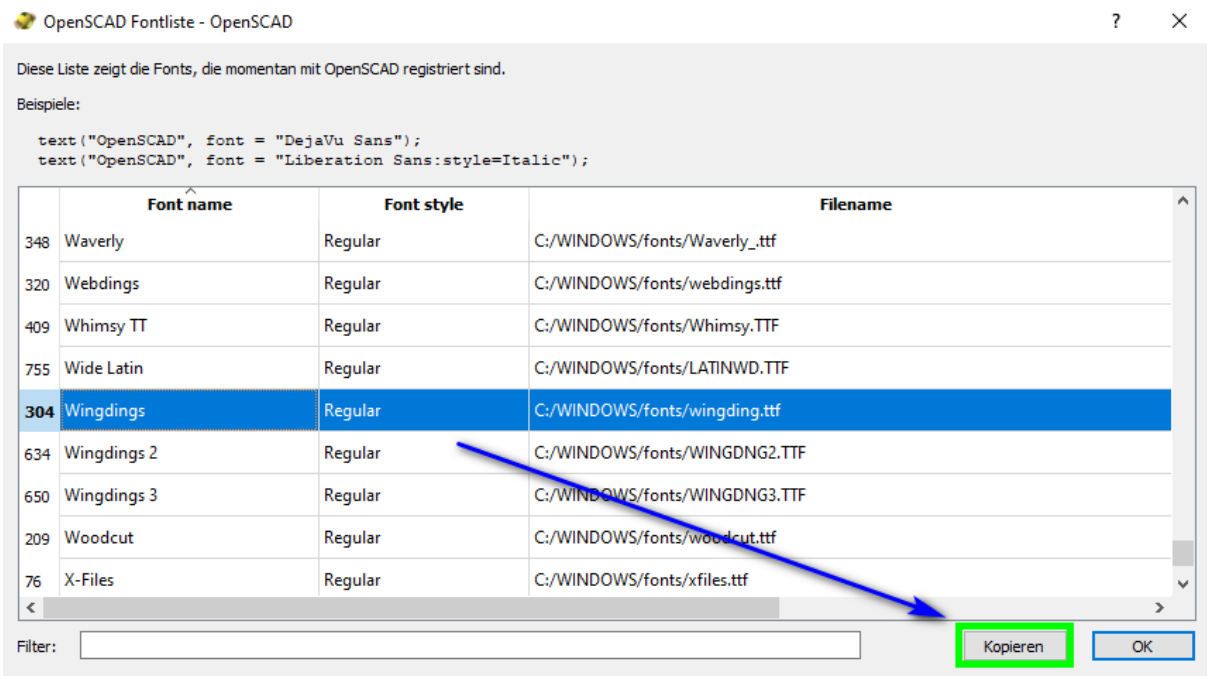
Zur Beachtung!

Meistens weiß man gar nicht welche Schriften auf dem PC installiert sind, jedoch kann OpenSCAD diese alle anzeigen!

In der oberen Menuzeile auf Hilfe, dann auf Fontliste klicken….



Dann erscheint das folgende Fenster:



Einfach eine Schrift auswählen und auf Kopieren klicken, dann auf OK.

Im Texteditor mittels „Paste“ die Schriftartbezeichnung, nebst Stil, einfügen:

```
"Wingdings:style=Regular"
```

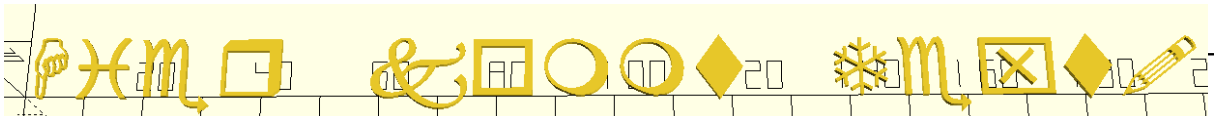
Davor wird gestellt : text("Hier kommt Text!", font=

```
text("Hier kommt Text!", font=
"Wingdings:style=Regular"
```

Vollendet wird das Ganze noch mit:);

```
text("Hier kommt Text!", font=
"Wingdings:style=Regular"
);
```

Nun kann mit F5 die Vorschau gestartet werden:

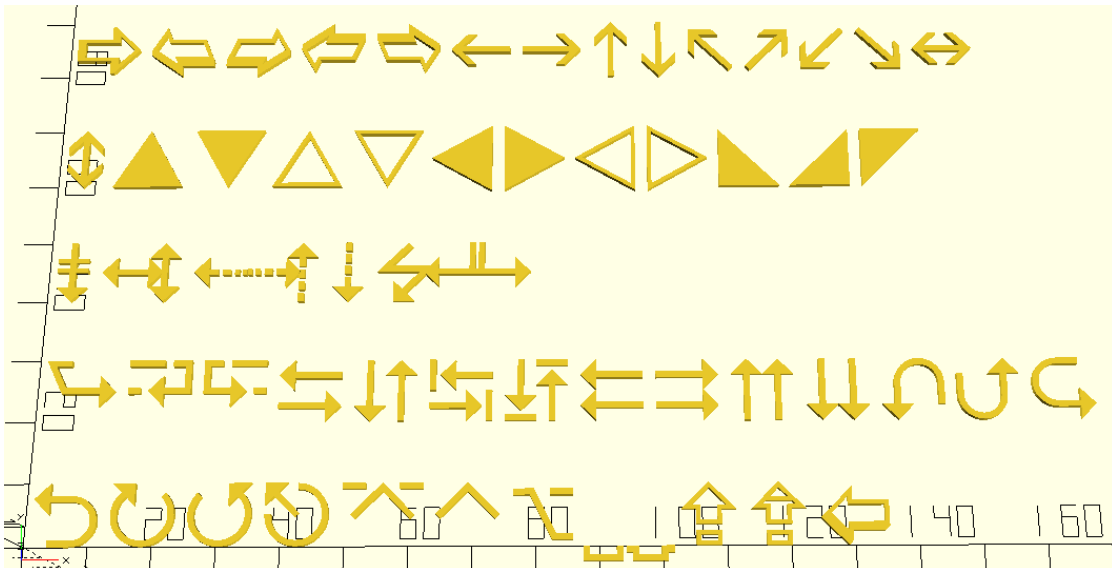


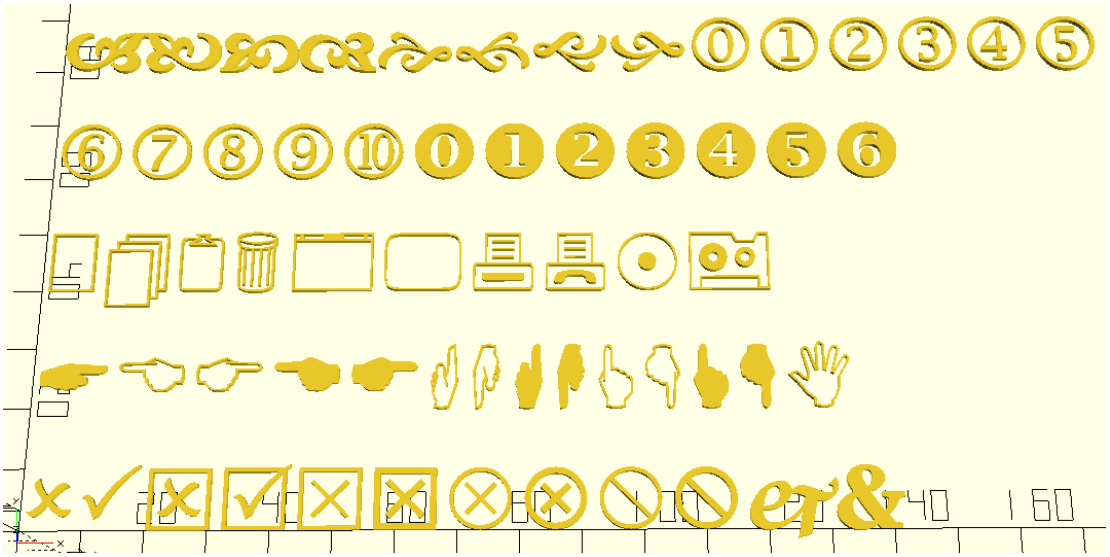
```
// Wingdings .....
translate([0,80,0])
text("abcdefghijklmn", font="Wingdings:style=Regular");
translate([0,60,0])
text("opqrstuvwxyz", font="Wingdings:style=Regular");
translate([0,40,0])
text("0123456789", font="Wingdings:style=Regular");
translate([0,20,0])
text("ABCDEFGHIJKLMN", font="Wingdings:style=Regular");
text("OPQRSTUVWXYZ", font="Wingdings:style=Regular");
```

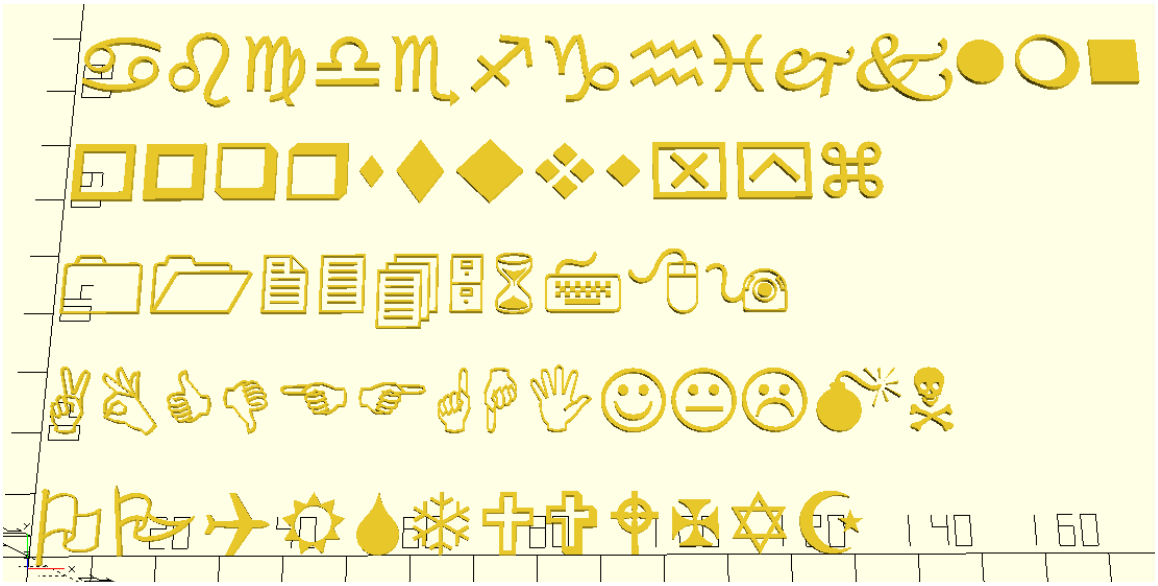
```
.....
text("mein Text ", font="Wingdings 2:style=Regular");
.....
```

```
.....
text("mein Text ", font="Wingdings 3:style=Regular");
.....
```

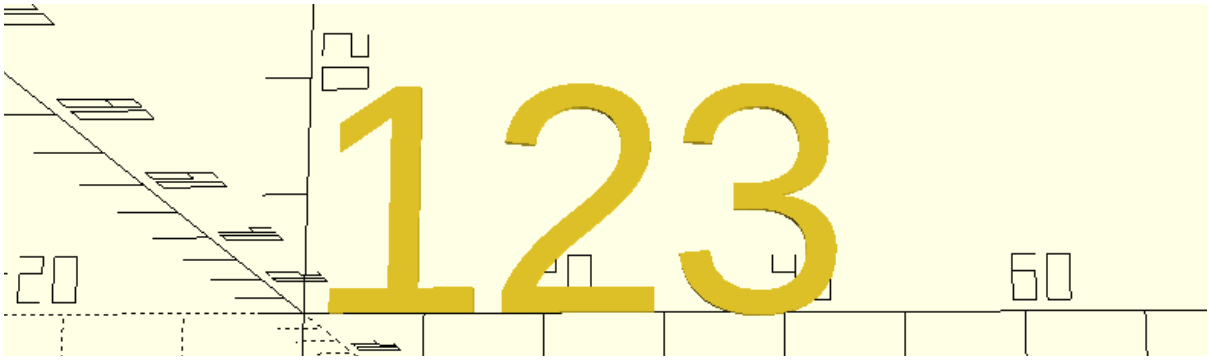
006-14







Es ist ferner möglich, numerische Werte mit dem Textbefehl zu zeichnen.
Soll eine Form mit einem numerischen Wert erstellt werden,



muss der Wert mit der str-Funktion in einen String umgewandelt werden:

```
text(str(123), size=20);
```

Anstatt die Zahl zwischen Anführungszeichen zu setzen, kann auch die str-Funktion angewendet werden, um die Zahl in einen String umzuwandeln.
Dies ist besonders hilfreich, wenn der numerische Wert in einer Variablen gespeichert wird - wie es später zu sehen sein wird.

Der Befehl „text“ beinhaltet viele Parameter die nicht immer eingesetzt werden müssen, jedoch der Erklärung bedürfen:

text Der zu generierende Text.

size Größenangabe in Dezimal. Der generierte Text hat eine Steigung (Höhe über der Grundlinie) von ungefähr dem angegebenen Wert. Der Standardwert ist 10. Verschiedene Schriftarten können etwas variieren und füllen möglicherweise nicht genau die angegebene Größe aus, normalerweise werden sie etwas kleiner dargestellt.

Schriftart Der Name der Schriftart, die verwendet werden soll. Dies ist nicht der Name der Schriftdatei, sondern der logische Schriftname (wird intern von der Bibliothek fontconfig verwaltet). Dies kann auch einen Stilparameter enthalten, siehe vordere Anleitung. Eine Liste der installierten Schriftarten und Stile erhalten Sie über den Schriftartenlistendialog (Hilfe -> Schriftartenliste).

halign Die horizontale Ausrichtung für den Text. Mögliche Werte sind „left“ (links), „center“ (mittig) und „right“ (rechts). Standard ist "left".

valign Die vertikale Ausrichtung für den Text. Mögliche Werte sind „top“ (oben), „center“ (mittig), „baseline“ (Schriftlinie) und „bottom“ (unten). Standard ist "baseline".

spacing Faktor zum Erhöhen/Verringern des Zeichenabstands. Der Standardwert von 1 ergibt den normalen Abstand für die Schriftart, ein Wert größer als 1 bewirkt, dass die Buchstaben weiter voneinander entfernt sind.

direction Richtung des Textflusses. Mögliche Werte sind „ltr“ (von links nach rechts), „rtl“ (von rechts nach links), „ttb“ (von oben nach unten) und „btt“ (von unten nach oben). Standard ist "ltr".

language Die Sprache des Textes. Standard ist "de".

script Das Skript des Textes. Standard ist "lateinisch".

$fn wird verwendet, um die von Freetype bereitgestellten gekrümmten Pfadsegmente zu unterteilen