

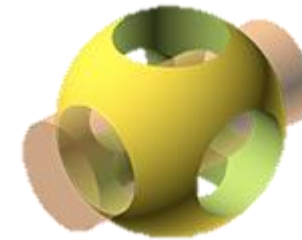
**Modelado 3D con OpenSCAD**

[Cristian González García](#)  
[gonzalezcristian@uniovi.es](mailto:gonzalezcristian@uniovi.es)

v 1.4.1 octubre 2021

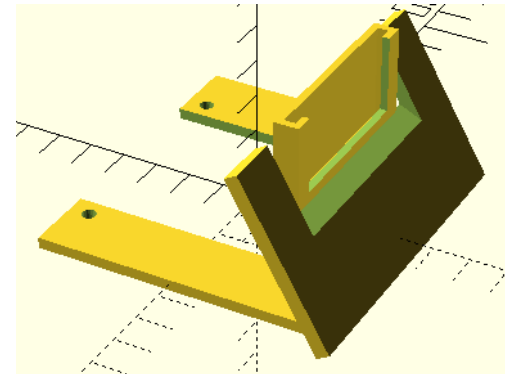
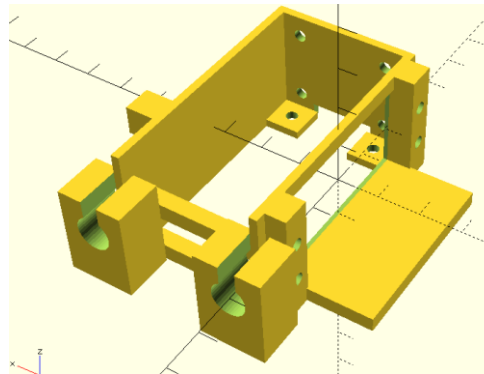
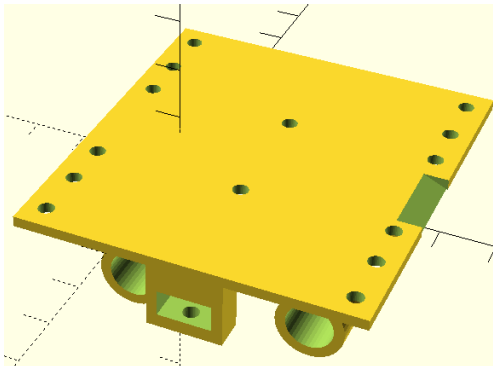
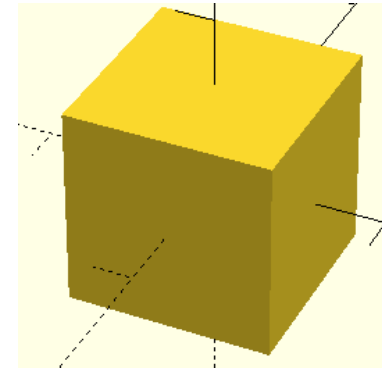
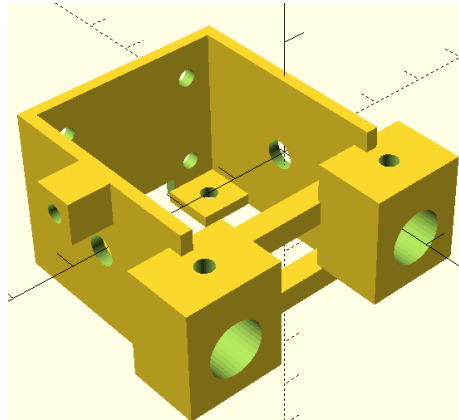
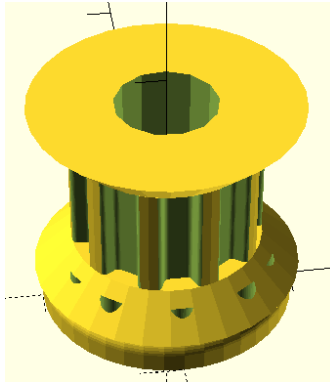
# Modelado 3D

- Existen **muchos programas populares** para crear modelos 3D
  - Blender, TopMod3, AutoCAD, SketchUp, TinkerCAD, FreeCAD, etc.
- OpenSCAD**
  - <http://www.openscad.org>
  - No es** un modelador interactivo (**gráfico**)
  - Utiliza un **lenguaje de script** para definir los modelos
  - Al compilar el lenguaje, se genera el modelo** (exportable a STL)
- Lenguaje OpenSCAD
  - Permite **definir** elementos en **2D y 3D**
  - Incluye** varias **características de lenguajes de programación**
    - Funciones, parámetros, condiciones, variables, etc.



# Ejemplos OpenSCAD

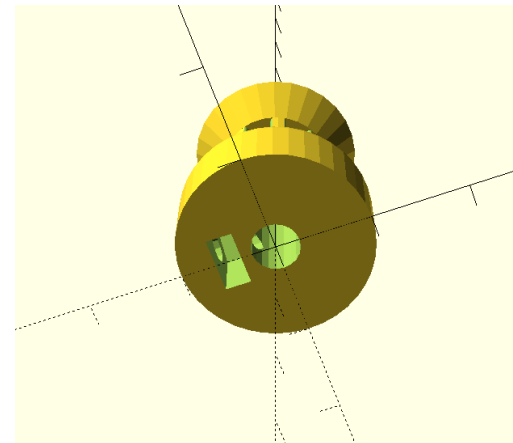
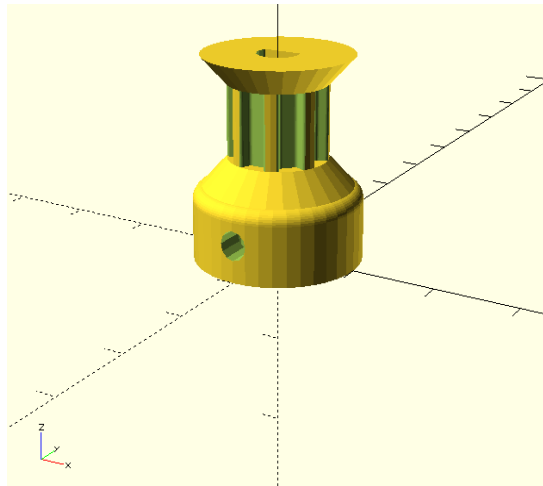
- <https://mega.nz/#!T1AxyA4l!UzszhAcUsGrqy3cUObgdwpY5luPUYrD57yZmXve2X40>



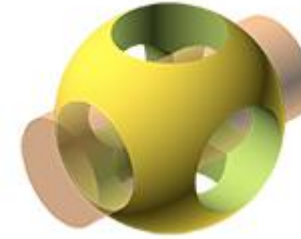
# Modelado 3D en OpenSCAD

## Lo utilizaremos para crear...

- Estructuras, carcasas, soportes, etc.
- Varios elementos mecánicos
  - Engranajes, poleas, cremalleras, etc.



## Puntos fuertes



- **Libre, gratuito y multiplataforma**
- **Gran comunidad** de desarrolladores (Soporte)
- Aplicación directa de conocimientos de programación
- Creación de **modelos parametrizables**
  - Dimensiones configurables, formas, etc.
- Favorece la **reutilización de código**
  - Librerías, módulos, fragmentos
  - Gran cantidad de recursos disponibles
- **Buena documentación**
- En cada versión incluyen nuevas características
  - Estable (ultima versión del 16 mayo 2019): 2019.05
  - Siguen trabajando en una nueva versión
  - <http://www.openscad.org/news.html>

## Puntos débiles

- **No** tiene sistema **de edición gráfico** o «drag and drop»
- **No** dispone de **reglas para medir** el tamaño
  - Hay que ir contabilizando todo
- Algunos **problemas gráficos al previsualizar** piezas

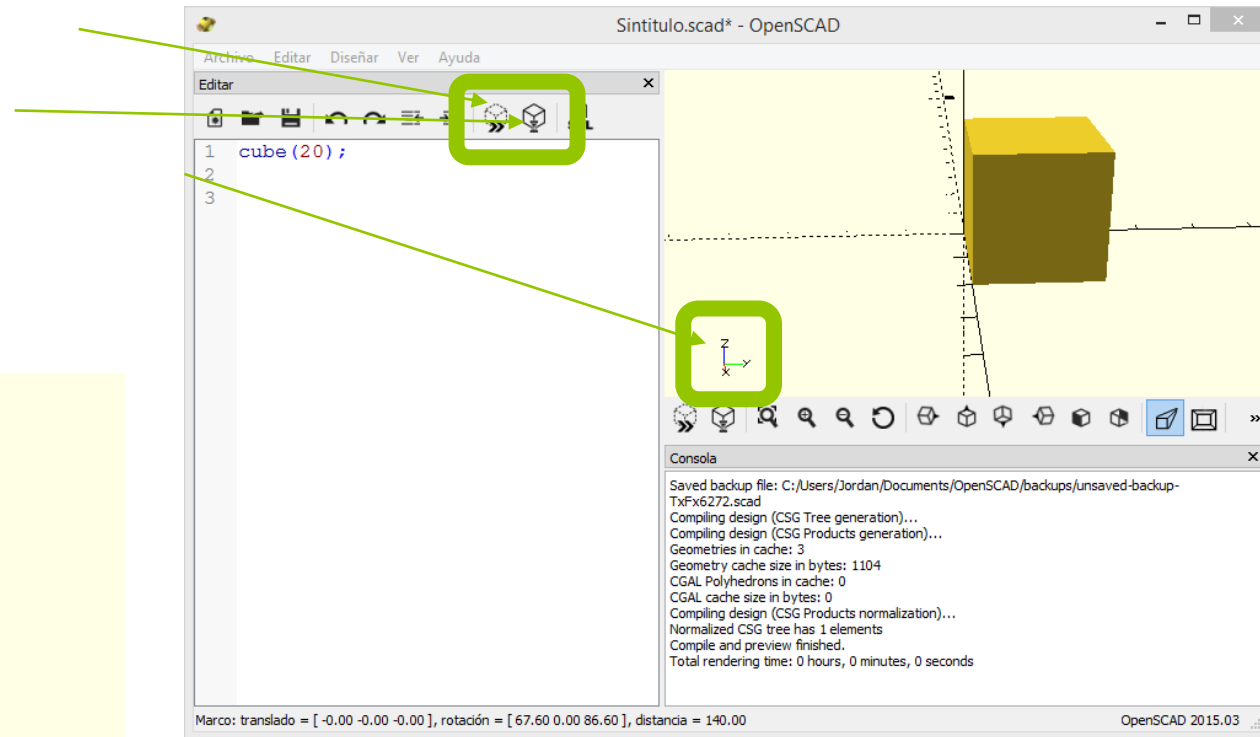
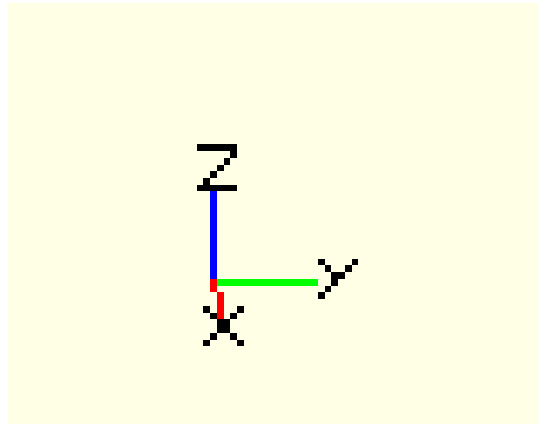
# Medidas

- OpenScad es «**unitless**»
- Las medidas no son en milímetros (mm), dependen del intérprete
  - Intérprete: slicers como el Cura u otros (editor de Windows) en dónde se importan las figuras
  - Por defecto/Convención, las impresoras 3D lo interpretan en mm, pero en el intérprete se puede cambiar o se pueden tener otras como pulgadas (25,4mm)



# OpenSCAD

- Antes de comenzar
  - Previsualizar F5
  - Compilar F6
  - Eje de coordenadas
  - Es **Case Sensitive**



## Preview (F5) vs Render (F6) I

- **Preview (F5)**

- **Forma rápida** de mostrar los resultados, muestra una aproximación (**puede contener fallos**)
  - Usa las librerías OpenCSG (librería sobre OpenGL) y OpenGL
- **Baja calidad**
- Puede producir «**artifacts**»
- Sirve para **depurar** y cuenta con diferentes funciones para ello
- Puede dar **problemas con algunas GPUs** integradas de Intel debido a drivers viejos o con problemas en el OpenGL
  - No realiza, o no realiza bien, las **diferencias** u otras operaciones
  - Dejar **finas capas** en las diferencias
    - Soluciones
      - Ejecutar el programa con GPU no integrada (NVIDIA, ...)
      - Editar -> Preferencias -> Avanzado -> Goldfeather (Es un algoritmo de OpenCSG)

- **Render (F6)**

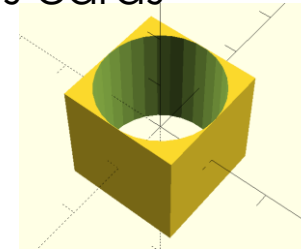
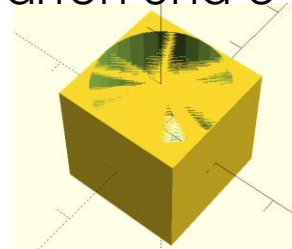
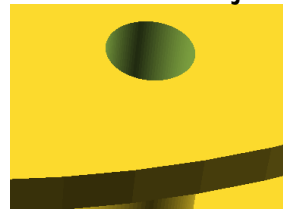
- **Alta calidad**
- Genera la geometría **exacta**
- Usa la librería CGAL (algoritmos de geometría para renderizar 3D)
- **Tarda** en mostrar los resultados
  - Puede tardar minutos e incluso horas

- [https://en.wikibooks.org/wiki/OpenSCAD\\_User\\_Manual/Building\\_OpenSCAD\\_from\\_Sources](https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Building_OpenSCAD_from_Sources)

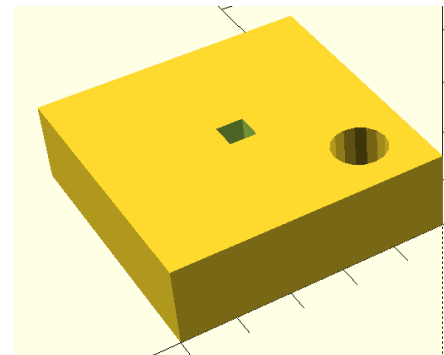
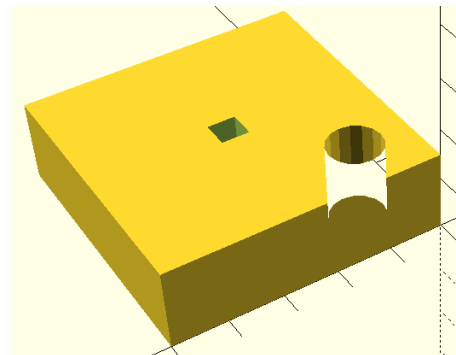
## Preview (F5) vs Render (F6) II

- OpenScad tiene algunos problemas en la previsualización
  - Esto no lo soluciona el algoritmo Goldfeather

Cuando varios objetos comparten una o más caras



Convexidad (Al importar piezas o poner convexidad baja)



# Formas geométricas básicas

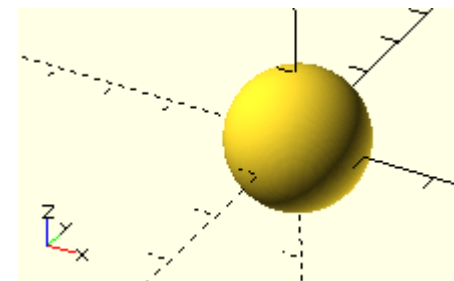
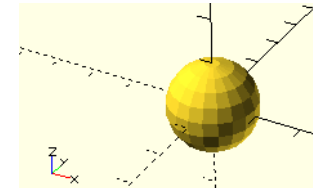
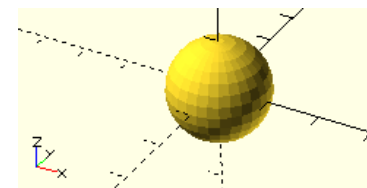
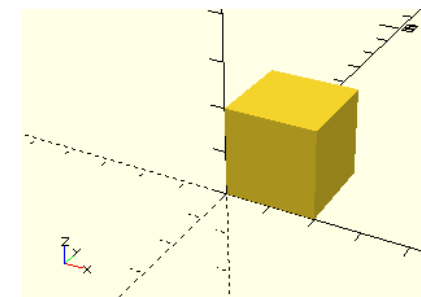
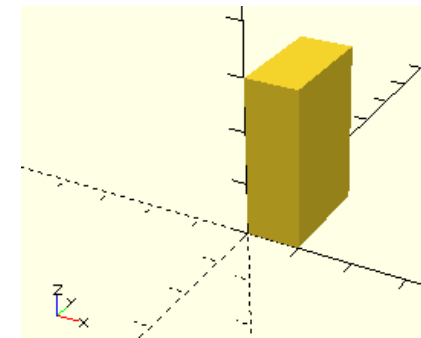
## Prismas rectangulares y esferas

- Formas geométricas básicas

```
cube([10,20,30]); // Sus 3  
medidas  
cube(20); // De 20*20*20
```

- d: diámetro
- \$fn: resolución («número de caras»)

```
sphere(d = 20);  
sphere(d = 20, $fn = 20);  
sphere(d = 20, $fn = 200);
```



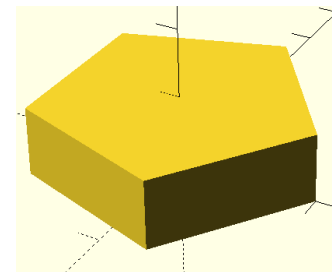
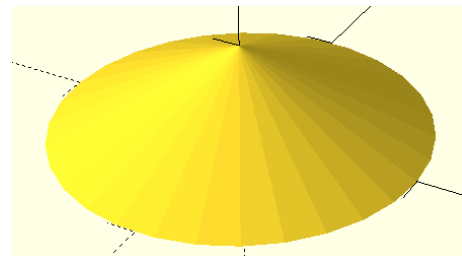
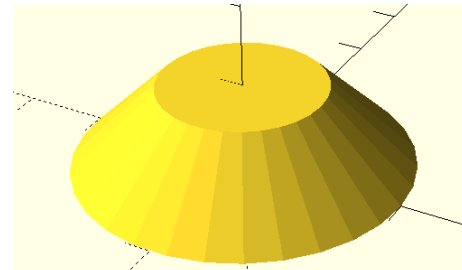
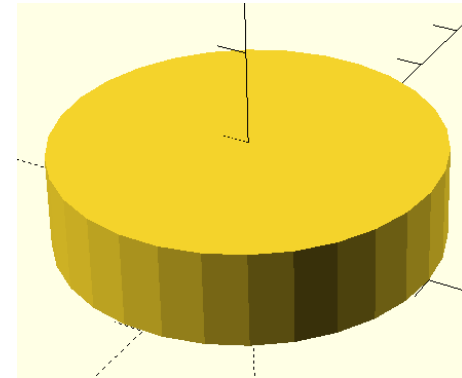
## Cilindros, conos y prismas

- h: altura
- r: radio, r1: radio de abajo, r2: radio de arriba

```
cylinder(h = 10, r=20);  
cylinder(h = 10, r1=20, r2=10);  
cylinder(h = 10, r1=20, r2=0);
```

- \$fn: número de caras

```
cylinder(h = 10, r=20, $fn=5);
```



# Propiedades

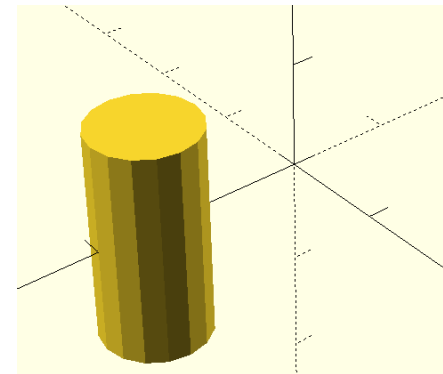
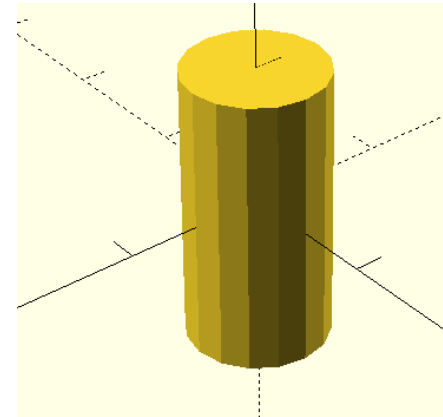
## Centrar y trasladar

- **Centrar** en el eje
  - Propiedad center

```
cylinder(h= 20, d = 10, center = true);
```

- Transformación **trasladar**
  - Se traslada sobre su propio eje XYZ
  - Afecta al elemento inferior

```
translate([15,0,0])  
cylinder(h= 20, d = 10, center = true);
```

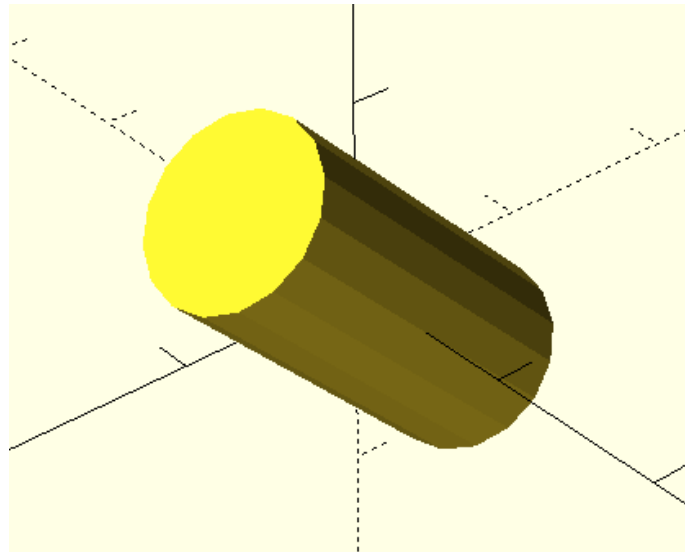




# Rotar

- Transformación **rotar**
  - Cambia el eje del elemento
  - Afecta al elemento inferior

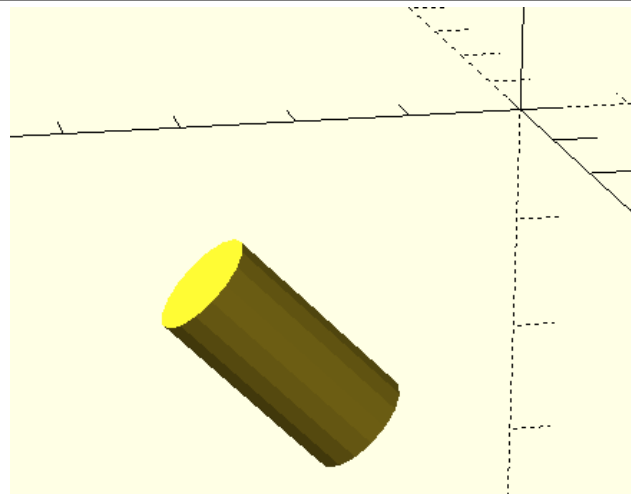
```
rotate([0,45,0])  
cylinder(h= 20, d = 10, center = true);
```



## Combinar transformaciones I

- Primero **rota** y luego se **traslada**
  - Como ya esta rotado, **se traslada sobre su propio eje** (diagonal)

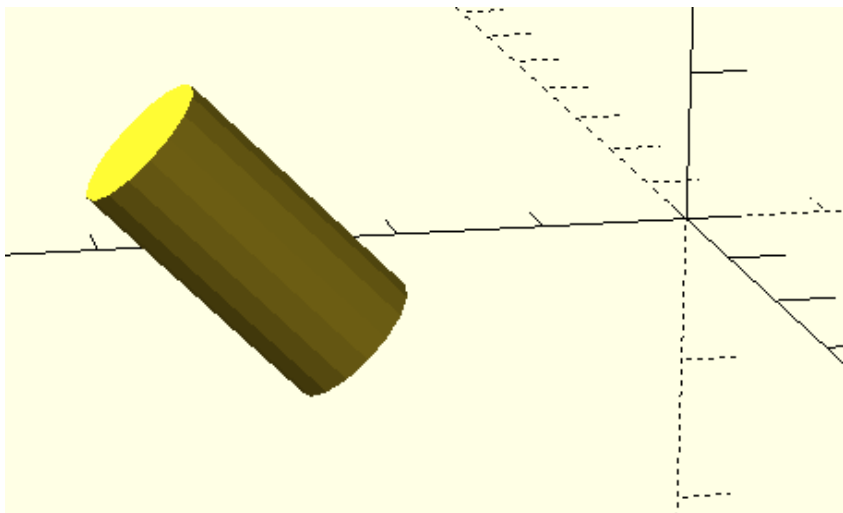
```
rotate([0,45,0])  
translate([30,0,0])  
cylinder(h= 20, d = 10, center = true);
```



## Combinar transformaciones II

- Primero se traslada y luego rota
  - Como primero se traslada, se mueve en horizontal y después rota

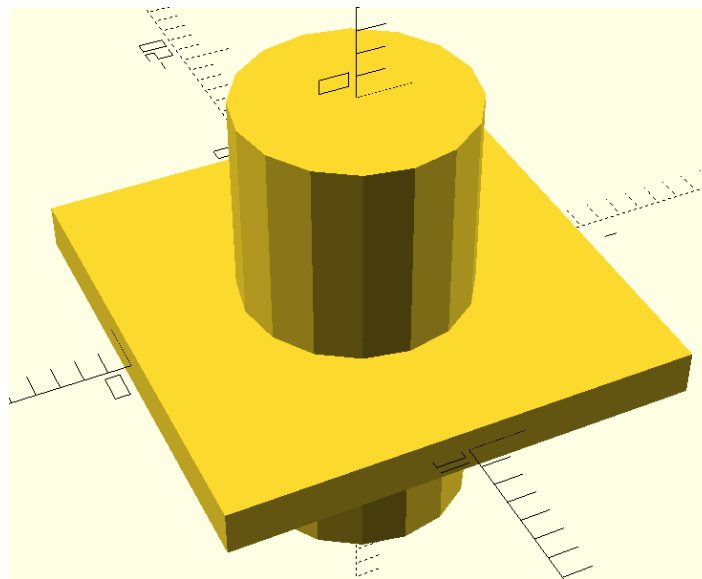
```
translate([30,0,0])  
rotate([0,45,0])  
cylinder(h= 20, d = 10, center = true);
```



## Solapamiento de objetos

- Se **solapan**

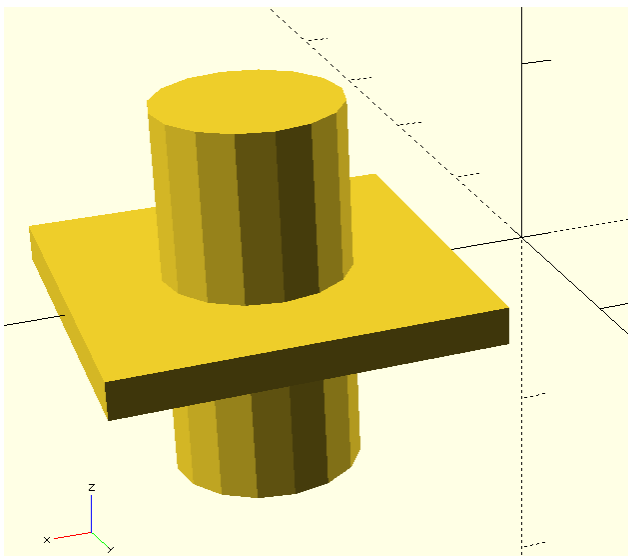
```
cylinder(h= 20, d = 10, center = true);  
cube([20,20,2], center = true);
```



## Agrupar transformaciones

- **Afecta a todos los elementos** { ... } en lugar de solo a uno

```
translate([15,0,0]){  
  cylinder(h= 20, d = 10, center = true);  
  cube([20,20,2], center = true);  
}
```

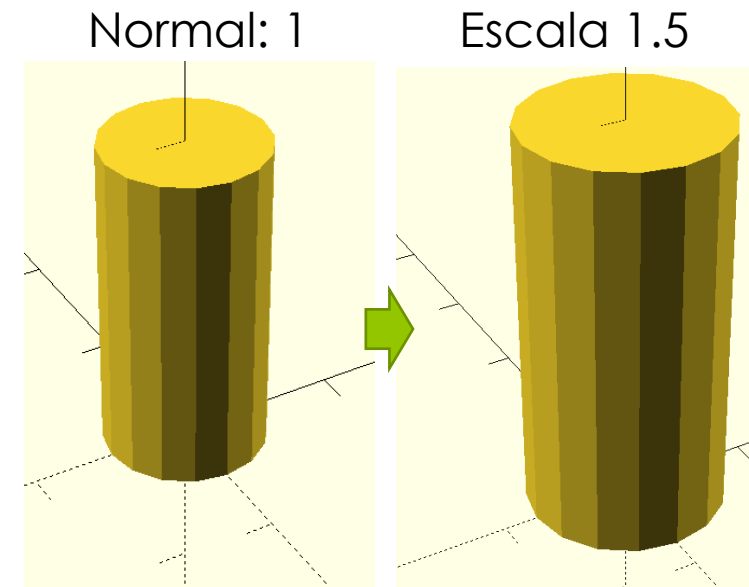


## Escalar

- Homogéneo

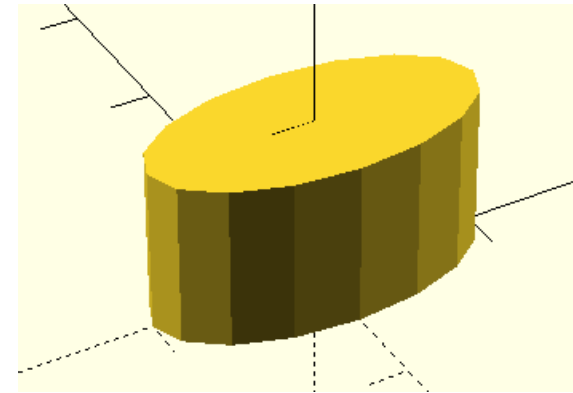
- Escala toda la pieza

```
scale(1.5)  
cylinder(h= 20, d = 10);
```



- Con factores** diferentes para cada eje (X, Y, Z)

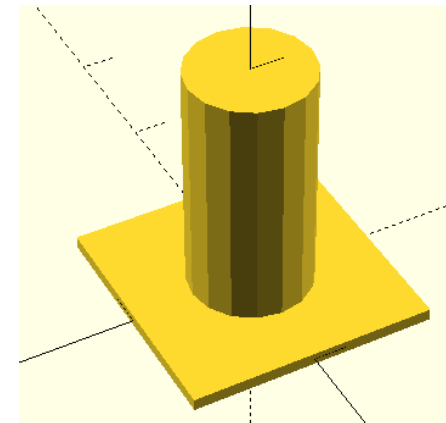
```
scale([0.5,1,2])  
cylinder(h= 20, d = 10);
```



# Uniones

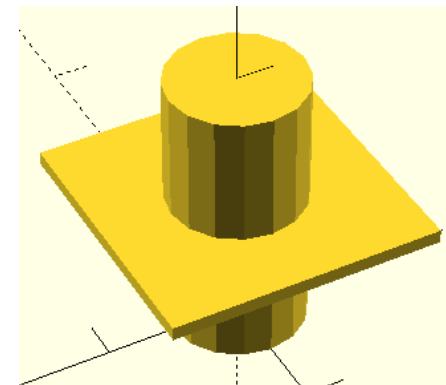
- El grupo de elementos { ... } **pasan a comportarse como uno**

```
union() {  
  cube([20,20,1], center = true);  
  translate([0,0,10])  
    cylinder(h= 20, d = 10, center = true);  
}
```



- La translación afecta depende de su colocación

```
translate([0,0,10])  
union() {  
  cube([20,20,1], center = true);  
  cylinder(h= 20, d = 10, center = true);  
}
```

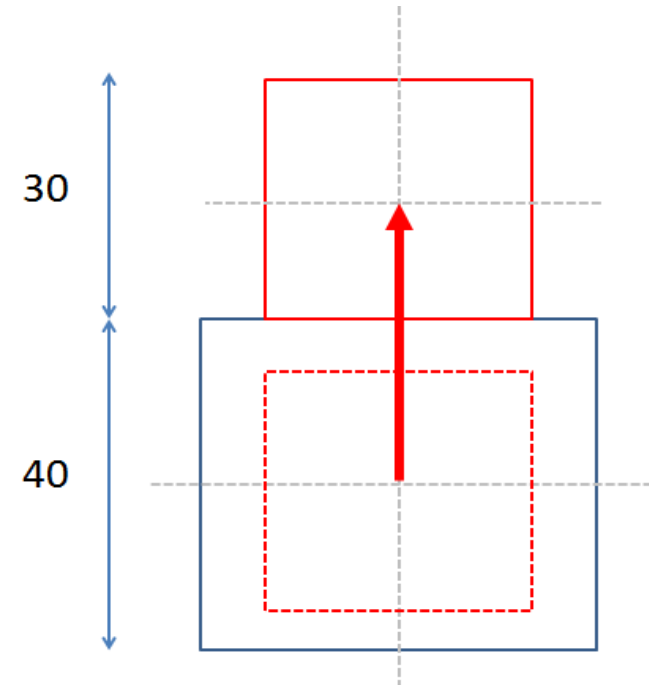
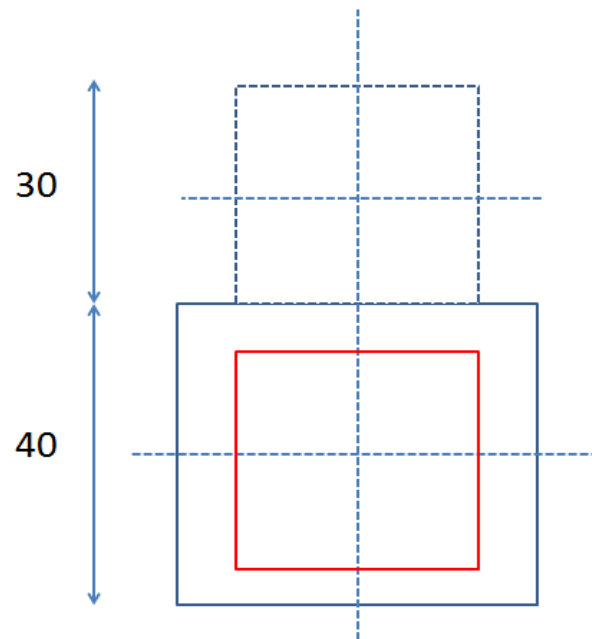


# Operaciones



# Cálculo de posiciones – Torre de cubos I

- Cubos de 30 (rojo) y 40 (azul) centrados

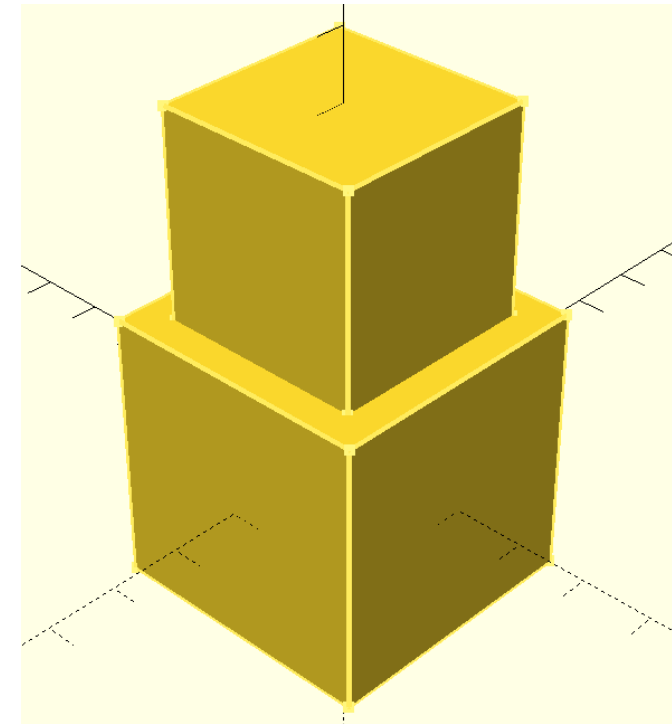


## Unión – Torre de cubos II

- Trasladando el segundo  $40/2 + 30/2$

```
union(){  
    cube([40,40,40], center =  
true);  
  
    translate([0,0, $40/2 + 30/2$ ],  
    cube([30,30,30], center =  
true);  
}
```

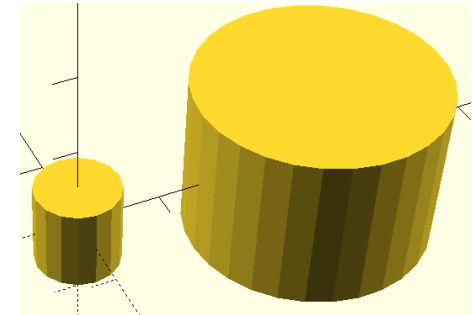
Válido



## Unión tangencial (hull)

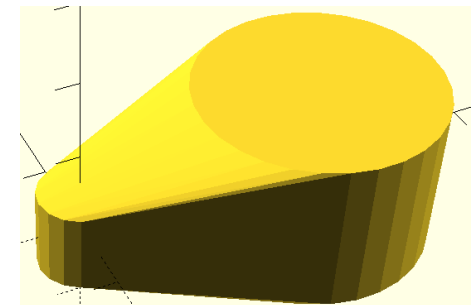
- Dos cilindros

```
cylinder (h = 10, d = 10, center = true);  
  
  translate([30,0,0])  
    cylinder (h = 20, d = 30, center = true);
```



- Completa la forma con uniones tangenciales

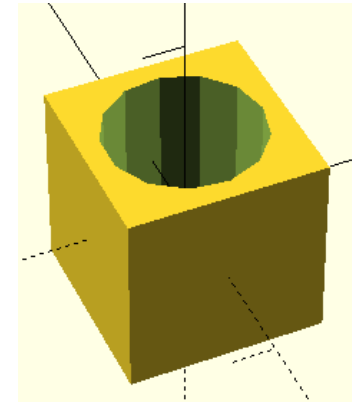
```
hull(){  
  cylinder (h = 10, d = 10, center = true);  
  
  translate([30,0,0])  
    cylinder (h = 20, d = 30, center = true);  
}
```



## Diferencia I

- **Sustraе al primer modelo todos los que se declaran a continuación**

```
difference() {
    cube (10, center = true);
    cylinder (h = 15, r= 4, center = true);
}
```



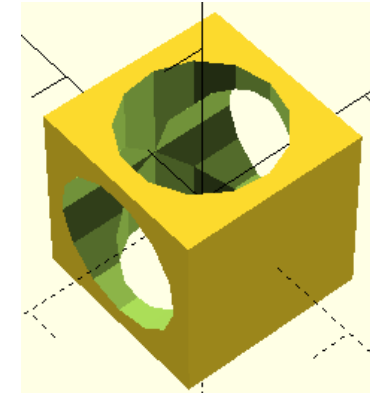
- **Puede no mostrarse en el modo previsualizar**
- **Recomendado** que las **diferencias no vayan justas** de tamaño para evitar problemas que sean difíciles de detectar (altura), siempre que se pueda

```
difference() {
    cube (10, center = true);
    cylinder (h = 10, r= 4, center = true);
}
```

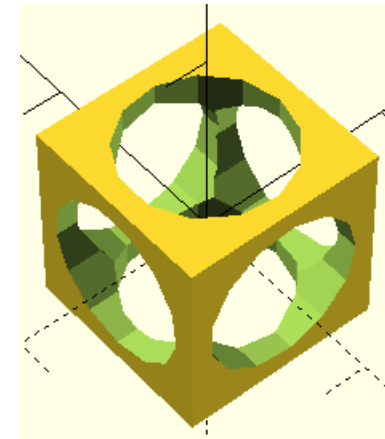
## Diferencia II

- Sustraer al primer modelo todos los que se declaran a continuación

```
difference() {  
  cube (10, center = true);  
  cylinder (h = 11, r = 4, center = true);  
  rotate([0,90,0])  
    cylinder (h = 11, r = 4, center = true);  
}
```

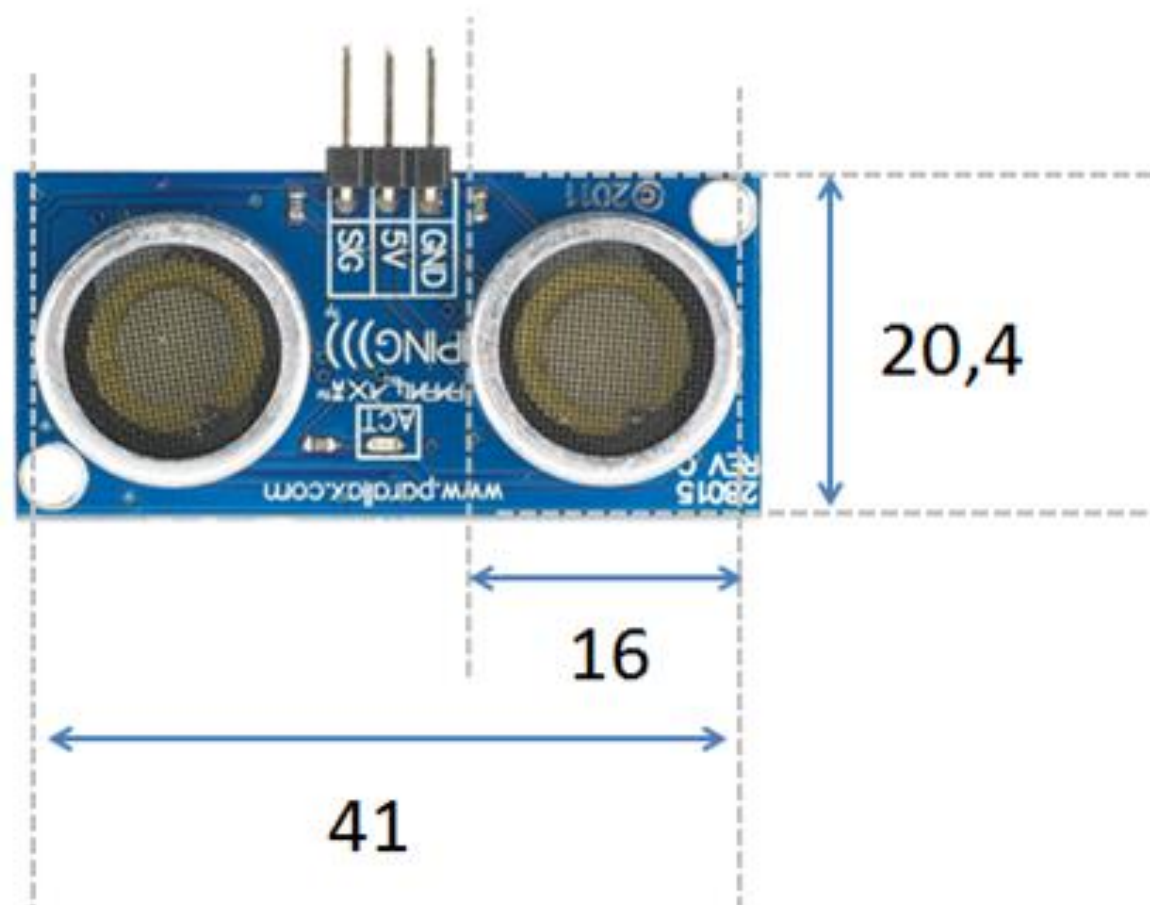


```
difference() {  
  cube (10, center = true);  
  cylinder (h = 11, r = 4, center = true);  
  rotate([0,90,0])  
    cylinder (h = 11, r = 4, center = true);  
  rotate([90,0,0])  
    cylinder (h = 11, r = 4, center = true);  
}
```



## Combinando uniones y diferencias

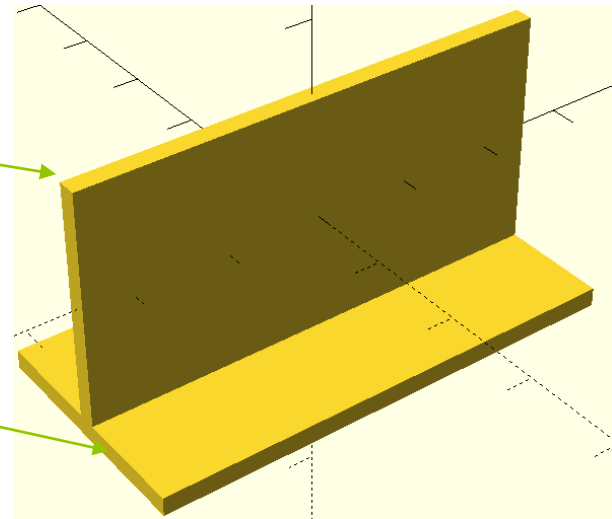
## Soporte para sensor de distancia



## Crear la estructura de la pieza

- Algo más grande que el sensor ej.: 50x25
- Las «paredes» pueden tener 2mm de profundidad

```
union(){  
  cube ([50,2,25], center = true);  
  
  translate([0,0,-25/2 - 2/2])  
    rotate([90,0,0])  
    cube ([50,2,25], center = true);  
}
```

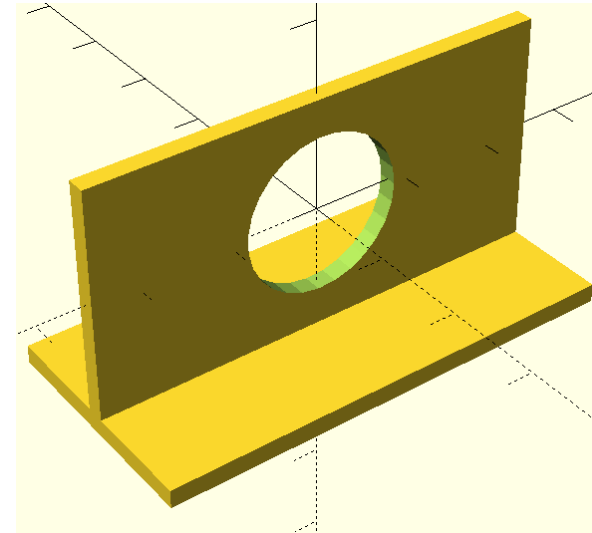




## Hacer los huecos

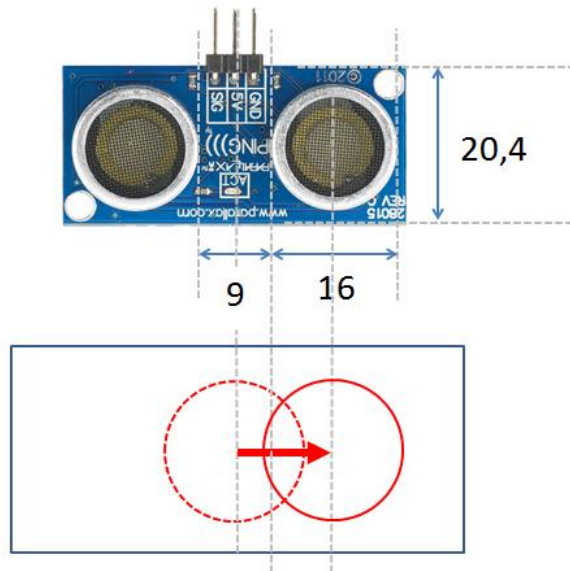
- **Dos cilindros** para los círculos
  - Primero los colocamos en el centro
- Son 16mm, pero hacemos 17mm
  - Al imprimir los cilindros **suelen ser algo más pequeños de lo indicado**

```
difference(){  
  union(){  
    .....  
  }  
  rotate([90,0,0])  
    cylinder (h = 65, d = 17, center = true);  
  rotate([90,0,0])  
    cylinder (h = 65, d = 17, center = true);  
}
```



## Trasladamos los huecos

- Debemos **trasladarlos hacia derecha e izquierda**



```
difference(){
  union(){
    .....
  }
}
```

```
translate([9/2 + 16/2,0,0])
```

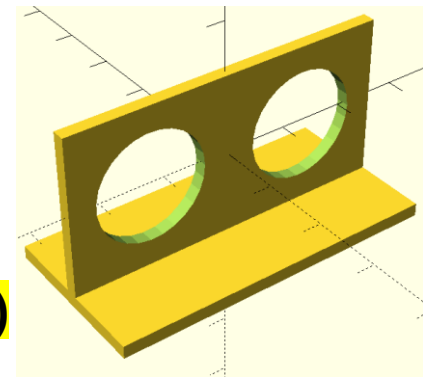
```
rotate([90,0,0])
```

```
cylinder (h = 65, d = 17, center =
true);
```

```
translate([- 9/2 - 16/2,0,0])
```

```
rotate([90,0,0])
```

```
cylinder (h = 65, d = 17, center = true);
}
```



## Fondo y depuración

## Modificadores

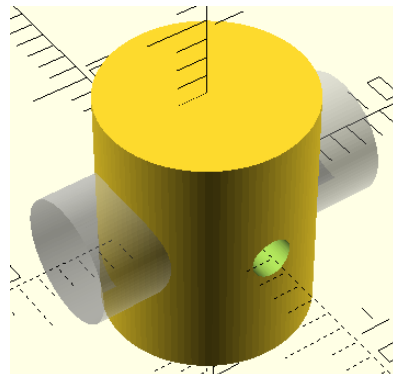
- **Se utilizan con previsualizar** en su mayoría, no con «renden»
- Sirven para **mostrar un «debug» de las piezas**
- **Afectan a todo el árbol seleccionado**
  - Suelen estar acotados por {...} o por funciones
- **Hay 4**
  - %, #, !, \*
- Se puede **imprimir por consola**
  - echo

## Modificador % (Background)

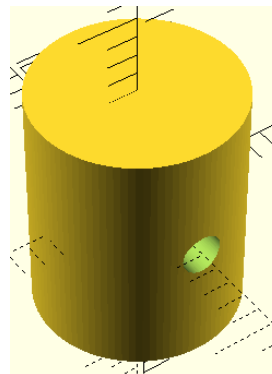
- Muestra el árbol transparente (blanco), solo en previsualizar
- Ignora el árbol en el renderizado

```
difference() {  
  cylinder (h = 12, r=5, center = true, $fn=100);  
  rotate ([90,0,0])  
  cylinder (h = 15, r=1, center = true, $fn=100);  
  %rotate ([0,90,0])  
  cylinder (h = 15, r=3, center = true, $fn=100);  
}
```

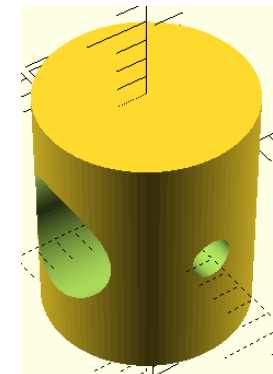
**Las diferencias  
pueden no  
mostrarse al  
previsualizar**



Previsualizar con %



Con % render

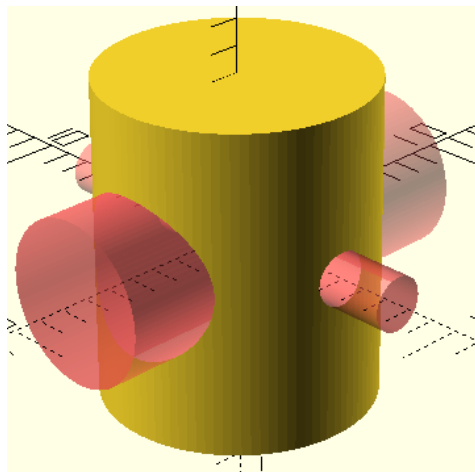


Sin % render

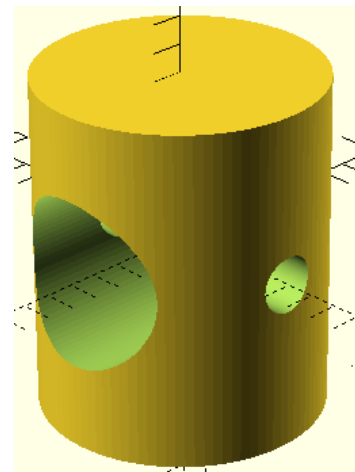
## Modificador # (Debug)

- Muestra el árbol marcado en magenta en previsualizar

```
difference() {  
  cylinder (h = 12, r=5, center = true, $fn=100);  
  #rotate ([90,0,0]) cylinder (h = 15, r=1, center = true, $fn=100);  
  #rotate ([0,90,0]) cylinder (h = 15, r=3, center = true, $fn=100);  
}
```



Previsualizar

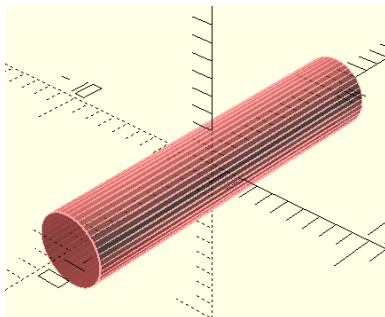


Renderizado

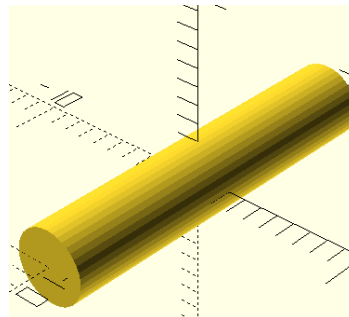
## Modificador ! (Root)

- Solo **mantiene el árbol marcado en magenta**, en **ambos modos**
  - Es decir, **ignora el resto del diseño**

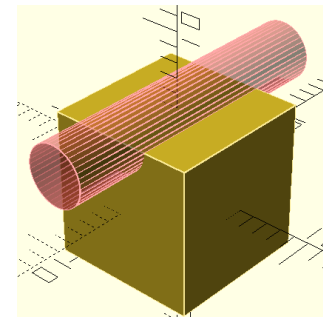
```
difference() {  
  cube(10, center = true);  
  translate([0, 0, 5]) {  
    !rotate([90, 0, 0]) {  
      #cylinder(r = 2, h = 20, center = true, $fn = 40);  
    }  
  }  
}
```



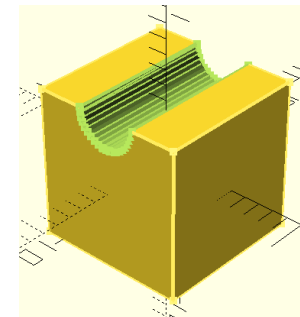
Previsualizar con ! y #



Render con !



Previsualizar # y sin !



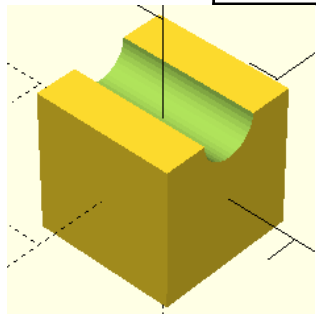
Render sin !

# Modificador \* (Disable)

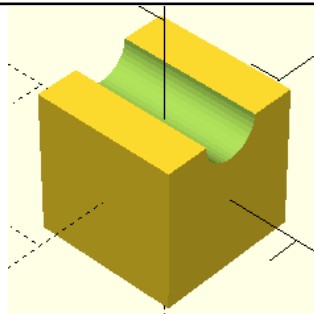
- Ignora todo el árbol seleccionado

```
difference() {
  cube(10, center = true);
  translate([0, 0, 5]) {
    rotate([0, 90, 0]) {
      cylinder(r = 2, h = 20, center = true, $fn = 40);
    }
    *rotate([90, 0, 0]) {
      #cylinder(r = 2, h = 20, center = true, $fn = 40);
    }
  }
}
```

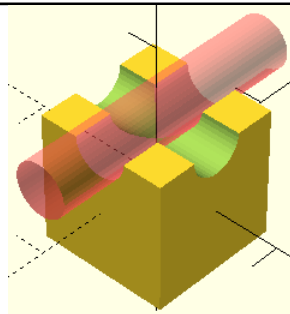
Previsualizar con \*  
y # en  
GPU Intel sin  
Goldfeather



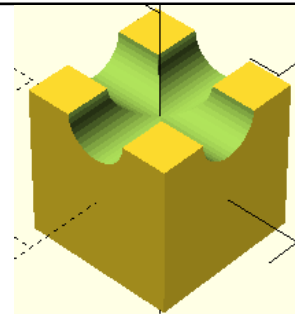
Previsualizar con \* y #



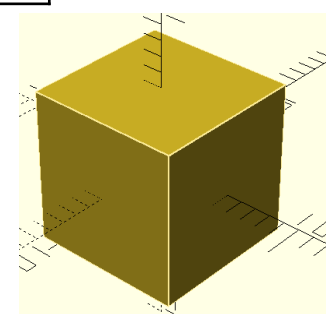
Render con \*



Previsualizar # y sin \*



Render sin \*





## Imprimir por consola (echo)

- Imprime por consola
- Tiene negrita «<b>» y cursiva «<i>»

```
my_h=50;  
my_r=100;  
echo("Cilindro de h=", my_h, " and r=", my_r);  
echo(my_h=my_h,my_r=my_r); // shortcut  
cylinder(h=my_h, r=my_r);  
//  
echo("<b>Hola Mundo Cruel</b><i>!</i>");
```

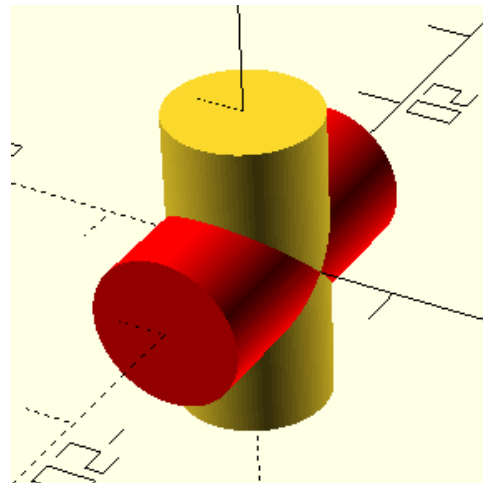


```
Compiling design (CSG Tree generation)...  
ECHO: "Cilindro de h=", 50, " and r=", 100  
ECHO: my_h = 50, my_r = 100  
ECHO: "Hola Mundo Cruel!"  
Rendering Polygon Mesh using CGAL...  
Geometries in cache: 65  
Geometry cache size in bytes: 242144  
CGAL Polyhedrons in cache: 72  
CGAL cache size in bytes: 12691824  
Total rendering time: 0 hours, 0 minutes, 0 seconds
```

## Cambiar color de la pieza

- Usar diferentes colores también puede servir para ver como se compone una pieza

```
union() {  
    cylinder (h = 20, d=10, center = true, $fn=100);  
    color("red")  
    rotate ([90,0,0])  
    cylinder (h = 20, d=10, center = true, $fn=100);  
}
```



# Variables y tipos de datos

# Variables y parámetros

- Permiten **crear modelos paramétricos** (Ejemplo rueda)

```
difference(){  
  cylinder(h= 10, r = 10, center = true); // rueda  
  #cylinder(h= 15, r = 5, center = true); // eje  
}
```

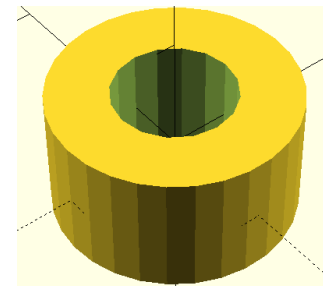
Sin  
parámetros

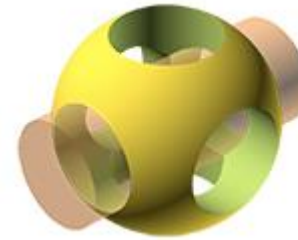


```
radioRueda = 10;  
ancho = 10;  
radioEje = 5;
```

```
difference(){  
  cylinder(h= ancho, r = radioRueda, center = true); // rueda  
  #cylinder(h= ancho*2, r = radioEje, center = true); // eje  
}
```

Con  
parámetros





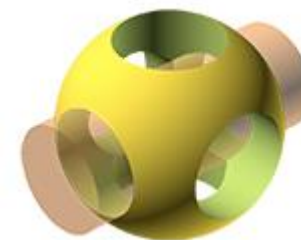
## Variables – Scope I

- Las **variables se calculan en tiempo de compilación**
- Realiza una **nueva copia del contenido del bucle para cada iteración**
  - Cada iteración: ¡tiene su propio Scope!
    - $y = y + 1$ ; // Sumar un número constante: 1, 2, 50, etc.
    - Es la única excepción que no funcionará

```
b = 0;  
c = 0;  
for(i = [1 : 5]){  
    b = b+i; // b se va incrementando con i  
    echo(b);  
    c = c+1; // c es constante = 0  
    echo(c);  
}
```



```
ECHO: 1  
ECHO: 1  
ECHO: 2  
ECHO: 1  
ECHO: 3  
ECHO: 1  
ECHO: 4  
ECHO: 1  
ECHO: 5  
ECHO: 1
```



## Variables – Scope II

- Las **variables se calculan en tiempo de compilación**
  - Reaprovechar variables puede dar problemas

```
a=1;  
echo(a);  
a=5;  
echo(a);  
  
for(i = [1 : 5]){  
    a = a+i;  
    echo(a);  
}  
  
a = 20;
```

¿Pero no vale 1?

ECHO: 20  
ECHO: 20  
ECHO: 21  
ECHO: 22  
ECHO: 23  
ECHO: 24  
ECHO: 25

Sobrescribe los valores anteriores de «a», «como en CSS»

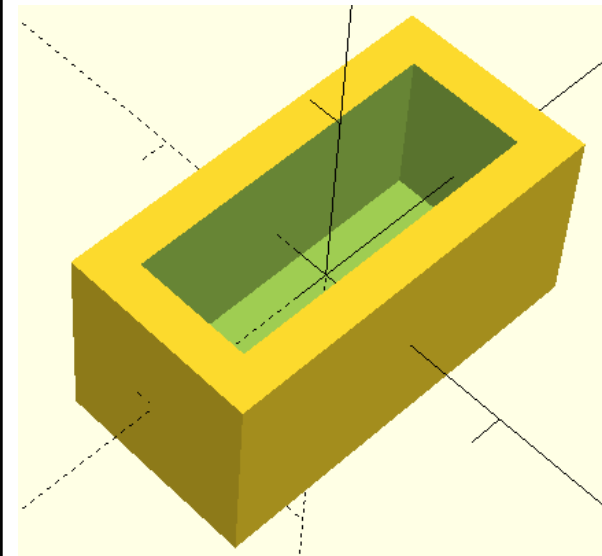
# Vectores

Creación y asignación

```
dim_caja = [10, 20, 10]; // ancho,  
profundidad, alto  
borde = 2;
```

```
difference(){  
  cube (dim_caja, center = true);  
  
  translate([0, 0, borde])  
  cube ([dim_caja[0]-borde*2,  
dim_caja[1]-borde*2, dim_caja[2]], center  
= true);  
}
```

Pasado como parámetro



Accedemos uno a uno y  
operamos sobre cada uno  
antes de pasarlos

## Bucles – for I

- Muy útil para **especificar tareas repetitivas**
  - Ej: soporte con agujeros cada 8mm
- **Definir piezas de forma iterativa**
  - Ej: unas escaleras, una pirámide de cubos, muros, etc.
- El **incremento** por defecto de 1 en 1
- Hay 4 tipos



# Bucles – for II

- Repeticiones de 1 a 5

```
for (i = [ 1 : 5 ]){
    echo(i);
}
```



ECHO: 1  
ECHO: 2  
ECHO: 3  
ECHO: 4  
ECHO: 5

- Definir el incremento en cada iteration

- for (i = [1 : <incremento> : 50])

```
for (i = [1 : 2 : 50]){
    echo(i);
}
```



ECHO: 1  
ECHO: 3  
ECHO: 5  
ECHO: 7  
ECHO: 9  
ECHO: 11  
ECHO: 13

- Iterar un vector

```
for (i = [1, 3, 5, 7, 10]){
    echo(i);
}
```



ECHO: 1  
ECHO: 3  
ECHO: 5  
ECHO: 7  
ECHO: 10

- for anidado

```
for (i=[0:10:50], j=[0:10:50]){
    echo(i, j);
}
```

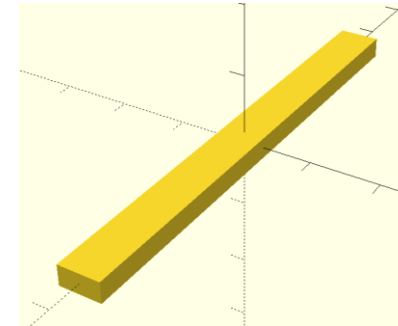


ECHO: 0, 0  
ECHO: 0, 10  
ECHO: 0, 20  
ECHO: 0, 30  
ECHO: 0, 40  
ECHO: 0, 50  
ECHO: 10, 0  
ECHO: 10, 10  
ECHO: 10, 20

## Bucles – for III

- Pieza sobre la que se realizarán los agujeros:

```
an_sop = 6; // ancho  
pr_sop = 70; // largo  
al_sop = 3; // alto  
cube([an_sop, pr_sop, al_sop], center =  
true);
```



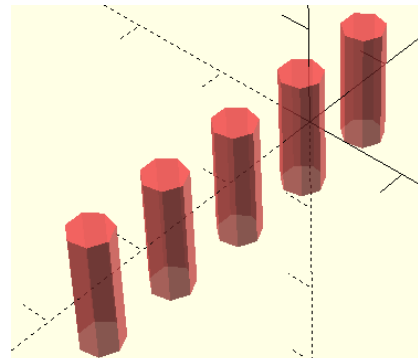
- Parámetros del agujero

```
d_agujero = 4;  
distancia = 8;
```

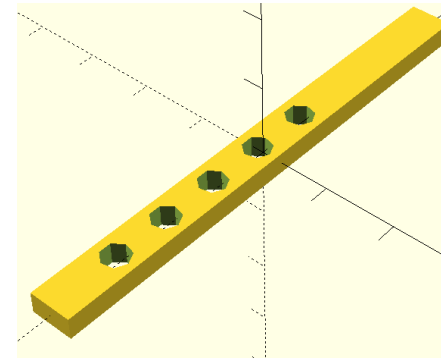
## Bucles – for IV

- 5 agujeros: empezando por el extremo izquierdo Y:  $-an\_sop/2$

```
... // Código de la diapositiva anterior  
difference()  
  cube([an_sop, pr_sop, al_sop], center = true);  
  for (i = [1 : 5]) {  
    #translate([0,- pr_sop/2 + d_agujero/2 + distancia*i,0])  
    cylinder(h= an_sop*2, d = d_agujero, center = true);  
  }  
}
```



Previsualizar con # y sin cubo

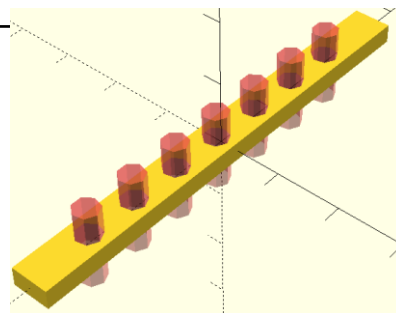


Renderizado

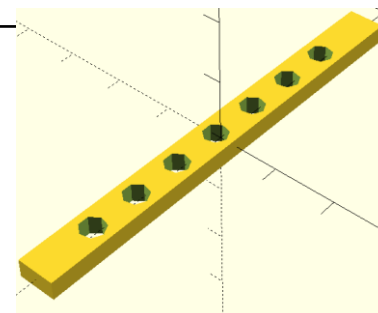
## Bucles – for V

- **Calcular** el número de **agujeros dinámicamente** (n)

```
... // Código de hace 2 diapositivas  
n = pr_sop / (d_agujero/2 + distancia);  
echo("Hay que hacer ", n);  
difference(){  
  cube([an_sop,pr_sop,al_sop], center = true);  
  for (i = [1 : n]) {  
    #translate([0,- pr_sop/2 + d_agujero/2 + distancia*i,0])  
    cylinder(h= an_sop*2, d = d_agujero, center = true);  
  }  
}
```



Previsualizar con #



Renderizado

# Condicionales – if

- Permiten **condicionar la ejecución** de parte **del script**
- Muy **útiles para piezas parametrizadas**
- Sentencia **tipo if - else if - else**

```
grande = true;

an_cubo = 0;
pr_cubo = 20;

if (an_cubo <= 10 && pr_cubo < 10){
    echo("Dentro");
}else if (grande){
    echo("Grande");
}else{
    echo("Fuera");
}
```

## Módulos e importación

# Módulos

- **Encapsulan** parte de un script
- **Reutilizables** mediante llamadas
  - Son similares a **funciones/métodos**, pero **sin retorno**
- **Admiten parámetros** de entrada
- **Admiten valores por defecto** en los parámetros
- **Declaración de un módulo**
  - `module <nombre>([parámetros])`

```
module rueda(radiusRueda, ancho, radiusEje){  
    difference(){  
        cylinder(h= ancho, r = radiusRueda, center = true); // rueda  
        #cylinder(h= ancho*2, r = radiusEje, center = true); // eje  
    }  
}
```

# Invocación de módulos y parámetros

- **Uso** de un módulo

```
rueda(10, 10, 3);  
translate([0, 20, 0])  
  rueda(10, 10, 3);
```

- **Valores por defecto**

```
module rueda(radiusRueda=10, ancho=10, radioEje=5){...}
```

- **Invocación con valores por defecto**

- Los argumentos **se envían** a los parámetros siempre **en orden**

```
rueda(); // Todos los valores por defecto  
rueda(9); // Cambia primer parámetro, resto por defecto  
rueda(radiusEje=9); // Cambia solo el radioEje y el resto por defecto
```



## Guardar y usar un fichero .scad

- Los módulos **se pueden guardar en ficheros independientes**
  - Esto suele ser **lo más recomendable** para mantener una buena organización y así usarse en otros scripts.
  - Archivo -> Salvar Como -> rueda.scad
- Ejemplo **importación y uso**
  - Módulo rueda en un fichero independiente y reutilizable

```
use <rueda.scad>
```

```
rueda();
```

```
...
```

- O si está en el mismo script no se usa use y sería como una función

## Módulo vs función

- Módulo
  - Realiza una acción
  - No devuelve resultado
- Función
  - Devuelve un resultado

**function** nombre (parámetros) = valor ;

**function** func0() = 5;

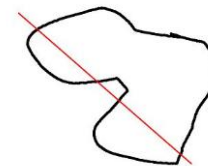
**function** func1(x=3) = 2\*x+1;

## Importar un modelo STL

- El STL importado se comporta como un bloque
- path del **modelo** en **formato STL, OFF o DXF**
- **OpenScad** guarda en formato **SCAD**. STL hay que exportarlo

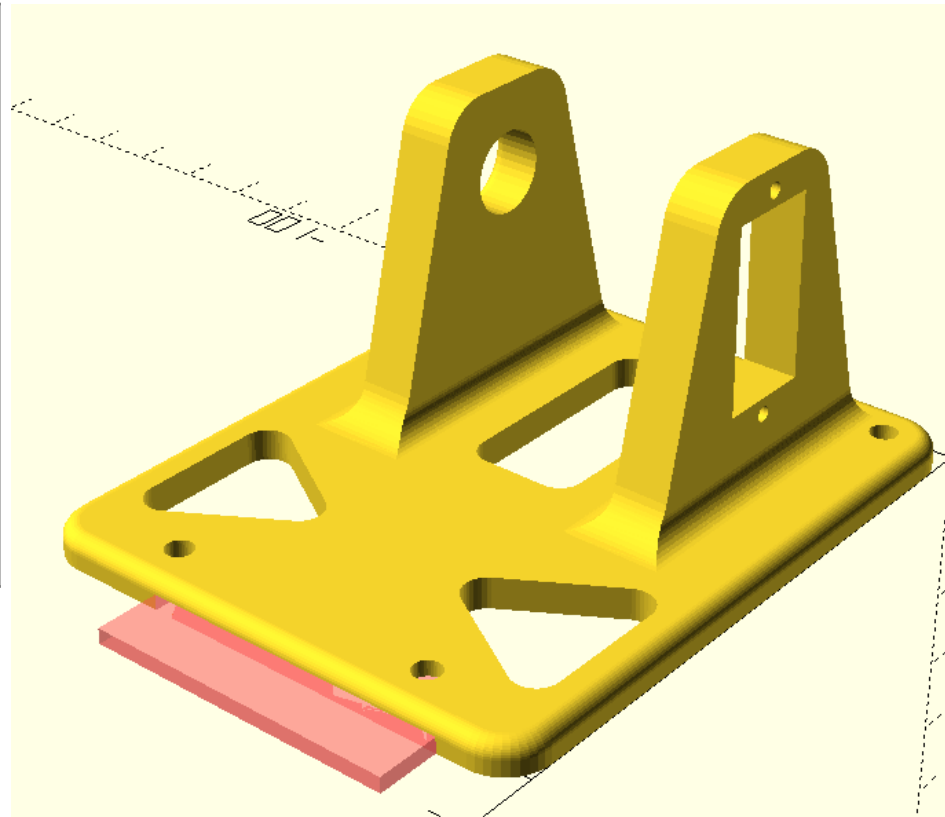
```
import("Base.stl");  
import("Base.stl", convexity=10);  
import("../Base.stl", convexity=10); // Windows  
import("C:\\y\\Base.stl", convexity=10); // Windows
```

- **Convexity** (opcional) indica el número máximo **de lados frontales que se permitirán ser atravesados por un rayo de luz**
  - En el ejemplo tiene una convexidad de 4, que es el número de veces que puede ser atravesado
  - **Solo útil cuando se renderiza** con OpenCSG en modo previo y no tiene efecto en la renderización del poliedro
  - **Recomendado 10**



## Importar un modelo STL con diferencia

```
difference(){  
  rotate([90,0,0])  
  import("Base.stl",  
    convexity=10);  
  
  translate([71,-50,0])  
  #cube([20,40,2]);  
}
```



## Extrusión lineal



## Extrusión lineal

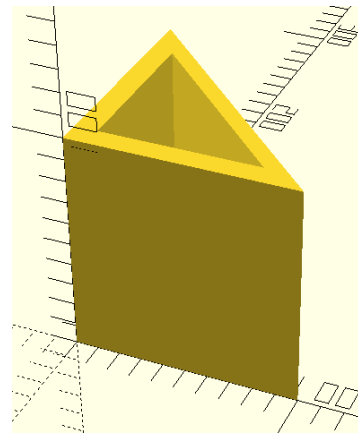
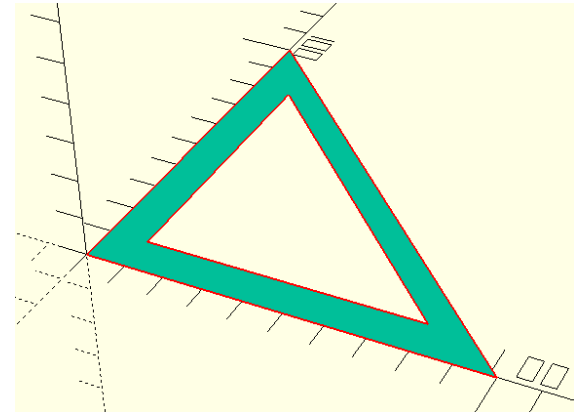
- Consiste en «estirar una forma», comúnmente plana, para convertirla en 3D

- Polígono 2D

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

- Extrusión lineal 100 aplicada al polígono

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



## Extrusión lineal con torsión

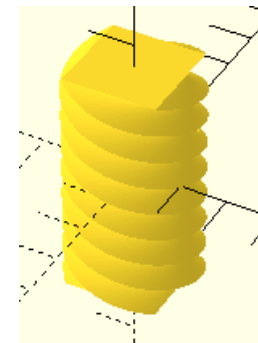
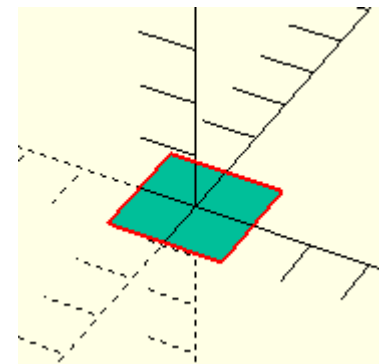
- «Estira y gira una forma plana»

- Twist: número de grados que se gira (desde el principio al final)
- \$fn: define el número de caras (efectos raros)
- Center: centra la figura en x e y

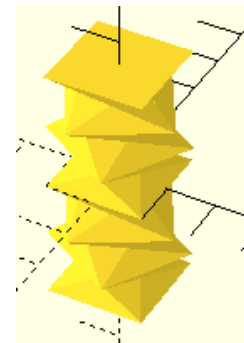
```
square([20,20], center = true);
```



```
linear_extrude(height = 50, twist =  
360*2, $fn=100, center = true)  
square([20,20], center = true);
```



fn = 100



fn = 1

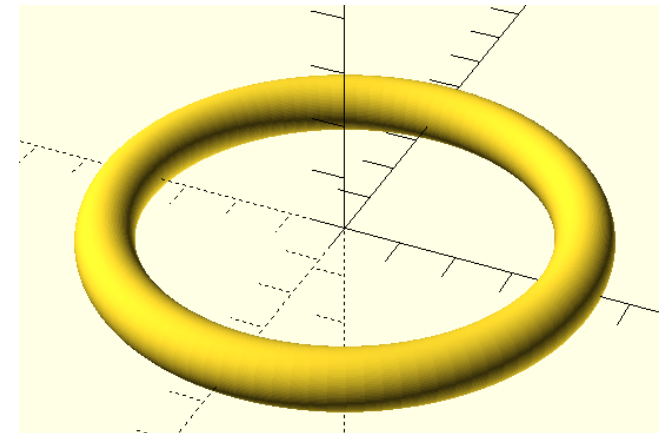
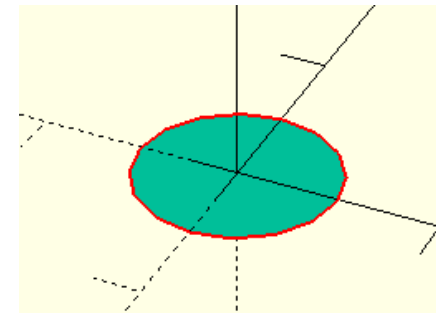
## Extrusión rotacional

- «**Estira y rota una forma plana**»
- \$fn= número de lados de la extrusión
  - Cuanto menor sea, más «cuadrado» queda

```
circle(d = 10);
```



```
rotate_extrude($fn= 100)  
translate([40,0,0])  
circle(d = 10);
```





## Cuando programemos

- **Usar variables y parámetros** para definir los modelos
  - No usar **nunca** variables «**hardcodeadas**»
- **Separar** el código **en módulos / funciones** (si es lógico)
- **Comentar el código**
  - Muy útil para trabajos en equipo y para la posteridad
- **Utilizar** los operadores de **depuración** %, #, \*, !, echo
- Respetar las tabulaciones (**hacer código legible**)
- **Utilizar librerías para simplificar** el trabajo
  - No reinventar la rueda
- **Revisar** los modelos varias veces
  - Asegurarse que todo es correcto
- **Tener en cuenta los consejos de diseño para impresión 3D**
  - Los veremos más adelante

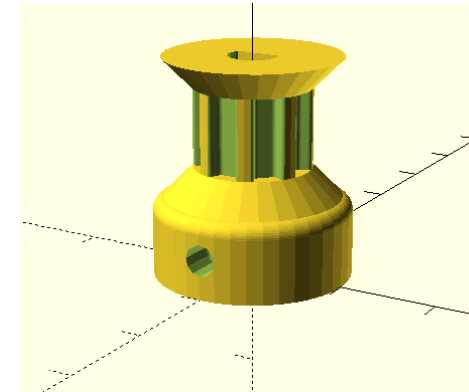
# Consejos

- ◉ **Librerías** OpenScad
  - ◉ Gran cantidad de módulos reutilizables
    - ◉ Gracias a la parametrización son muy potentes
    - ◉ Principales piezas: tornillos, engranajes, poleas, etc.
- ◉ **Modelos con código abierto**
  - ◉ Analizar como están hechos
  - ◉ Modificarlos / reutilizarlos total o parcialmente

# Librerías OpenScad

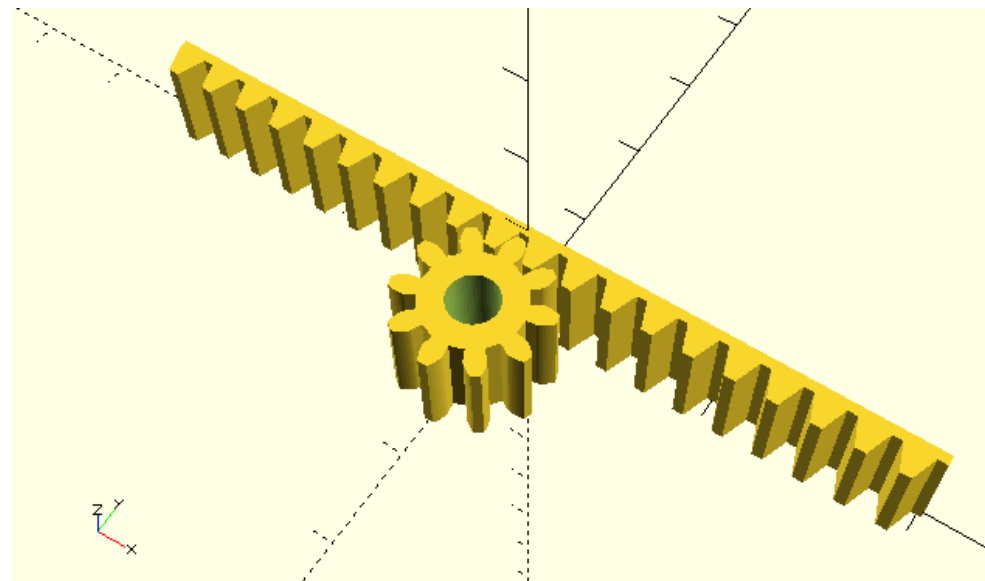
# Poleas

- <https://github.com/oc3d/Pulleys>
- teeth: nº de dientes
- belt: tipo de diente en la correa
  - T5, paso métrico 5mm una de las más comunes,
  - Otras T2, T5, T10... El ancho puede ser variable
- nut: tipo de tornillo para encaje inferior (M3 Largo)
- gear\_height: ancho del engranaje
  - Solo la parte del engranaje
  - Tiene que ser mayor que la correa
- Otras
  - <https://www.thingiverse.com/thing:2105020>



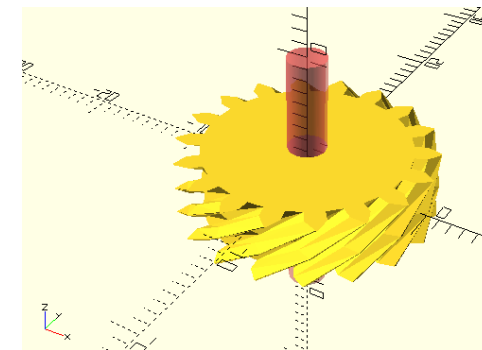
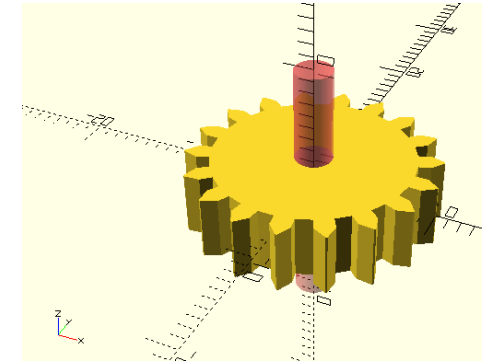
# Piñón Cremallera

- <https://github.com/Ardustorm/Rack-Pinion-openscadLib>
- `rack(4,20,10,1);`
  - Diente (mm)
  - Ancho rack(mm)
  - Alto rack (mm)
  - Alto base (mm)
- `pinion(4,10,10,5);`
  - Diente (mm)
  - Diámetro (mm)
  - Alto (mm)
  - Diámetro eje (mm)



# Engranajes

- <https://www.thingiverse.com/akaziuna/collections/openscad-gear-scripts>
- height : alto
- type: tipo de engranaje
  - Spur - normal,
  - Cog - 2D,
  - Double/Single Helix
- teeth: número de dientes
  - Calcula el diámetro en función a los dientes
  - Esta librería siempre usa el mismo tamaño de dientes
  - Todos los elementos son interoperables
- hole: incluir eje (Yes / No)
- holeSize = radio del eje
  - No confundir con el diámetro



## Repositorios de modelos 3D

## Repositorios de modelos

- <http://www.thingiverse.com/>
- <http://letsmakerobots.com/>
- <https://grabcad.com/>
- <http://pinshape.com/>
- <http://www.yeggi.com>
- <https://www.youmagine.com>
- <http://3dprint.nih.gov>
- <https://www.myminifactory.com>
- <http://www.123dapp.com>
- <http://libre3d.com/>
- <http://www.cgtrader.com/>
- <https://cubehero.com>
- <https://cults3d.com/>
- Muchos otros...

**MakerBot Thingiverse**



AUTODESK® 123D®





**vevox**  
Audience Engagement

Join at [vevox.app](https://vevox.app)

Or search **Vevox** in the app store

**ID: 170-519-236**

Join: **vevox**

**-713-930**



## El preview...




Vote for up to 2 choices

- ✓ 1. permite visualizar la pieza  
 0%
- 2. permite ver la pieza final de forma perfecta y bien renderizada  
 0%
- 3. tarda bastante en renderizar la pieza, incluso minutos y horas  
 0%
- ✓ 4. puede contener fallos visuales y crear mal la pieza, pero es un borrador  
 0%

(% = Percentage of Voters)

## El renderizado...

Vote for up to 2 choices

- ✓ 1. permite visualizar la pieza  
 78.26%
- ✓ 2. genera una visualización perfecta o casi perfecta de la pieza  
 100%
- 3. tarda muy poco en visualizar la pieza  
 4.35%

(% = Percentage of Voters)

## La función para hacer un prisma es:

1. `prism([20,30,40])`



2. `prism(20)`



✓ 3. `cube([20,30,40])`



4. `cube(20)`

0%

Al igual que con la multiplicación, el orden de trasladar y rotar o trasladar y rotar no altera el resultado.

1. Verdadero







2. Falso



## Que operaciones son «combinables»:

Vote for up to 4 choices

- ✓ 1. Aritméticas como parámetro a una «función»  
 72%
- ✓ 2. Traslaciones y rotaciones dentro de diferencias  
 76%
- ✓ 3. Traslaciones y rotaciones dentro de uniones  
 76%
- ✓ 4. Uniones, diferencias, traslaciones y rotaciones  
 76%

(% = Percentage of Voters)

## Los 4 modificadores sirven para...

1. Cambiar el color de la pieza



✓ 2. Ayudar a hacer debug



3. Hacer diferencias



4. Crear nuevas piezas cambiando el color a invisibles/blanquecino



## Leaderboard

Position	Participants	Score
=1	UO264046, UO264690, UO263946	6/6
=4	UO245110, UO263544, UO264184, UO264427	5/6
=8	UO250825, UO257745, UO258060, UO261072, UO264558, UO265111, UO264074	4/6
=15	UO251683, UO257795, UO264637, UO264761, UO265349, UO266575, UO253628 UO253628	3/6
=22	Ignacio Arias, UO264850	2/6
=24	UO271439, UO257410	1/6

Total Participants: 25

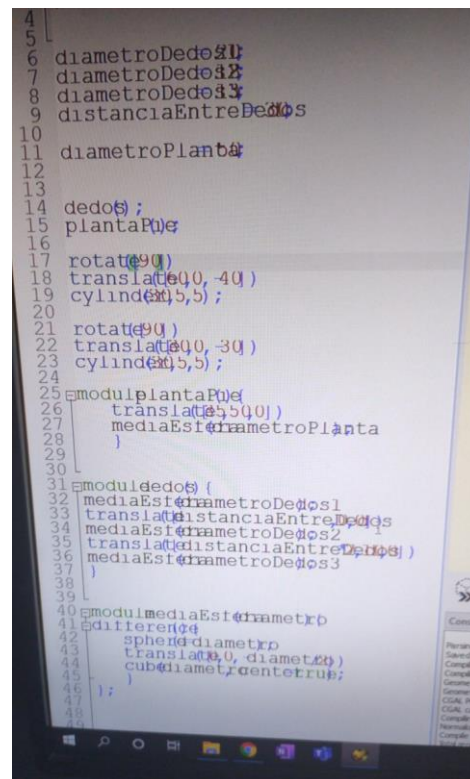
Average Score:3,7

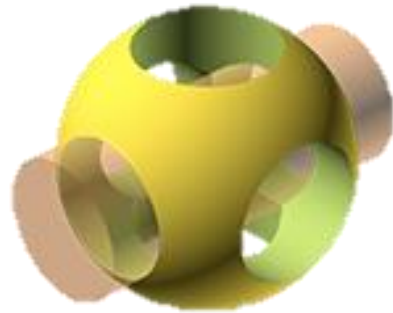


Posibles errores

## Posible error I

- Al mover una línea de código en OpenScad se me desconfiguró todo el fichero
  - Pasar texto al Notepad y volver a pegarlo en OpenScad lo arregla





**Modelado 3D con OpenSCAD**

Cristian González García  
[gonzalezcristian@uniovi.es](mailto:gonzalezcristian@uniovi.es)

v 1.4.1 octubre 2021