# Untitled

October 29, 2024

[ ]:
```
#BOX
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

result = cq.Workplane("XY").rect(4,5).extrude(3)

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[ ]:
```
# braile
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *
from collections import namedtuple

# Braille text input (You can change this for actual Braille patterns)
# Unicode Braille patterns (e.g., '\u2801' represents Braille with dot 1)
text_lines = [u'\u2801', u'\u2803']  # Sample Braille text

# Braille cell geometry configuration
horizontal_interdot = 2.5
vertical_interdot = 2.5
horizontal_intercell = 6
vertical_interline = 10
dot_height = 0.5
dot_diameter = 1.3
base_thickness = 1.5

# Defining Braille cell geometry as a named tuple
BrailleCellGeometry = namedtuple(
    "BrailleCellGeometry",
    (
        "horizontal_interdot",
        "vertical_interdot",
        "intercell",
```

```
        "interline",
        "dot_height",
        "dot_diameter",
    ),
)

class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

    def __len__(self):
        return 2

    def __getitem__(self, index):
        return (self.x, self.y)[index]

    def __str__(self):
        return "({}, {})".format(self.x, self.y)

# Function to convert Braille characters into points on the plate
def brailleToPoints(text, cell_geometry):
    # Unicode bit pattern (cf. https://en.wikipedia.org/wiki/Braille_Patterns).
    masks = [0b00000001, 0b00000010, 0b00000100, 0b00001000, 0b00010000,␣
 ↪0b00100000, 0b01000000, 0b10000000]

    # Corresponding dot positions
    w = cell_geometry.horizontal_interdot
    h = cell_geometry.vertical_interdot
    positions = [
        Point(0, 2 * h), Point(0, h), Point(0, 0), Point(w, 2 * h),
        Point(w, h), Point(w, 0), Point(0, -h), Point(w, -h)
    ]

    # Braille blank pattern
    blank = u'\u2800'
    points = []

    # Position of dot1 along the x-axis (horizontal)
    character_origin = 0
    for c in text:
        delta_to_blank = ord(c) - ord(blank)
        for m, p in zip(masks, positions):
            if m & delta_to_blank:
```

```
                points.append(p + Point(character_origin, 0))
        character_origin += cell_geometry.intercell
    return points

# Get dimensions of the Braille plate
def get_plate_height(text_lines, cell_geometry):
    return (
        2 * cell_geometry.vertical_interdot
        + 2 * cell_geometry.vertical_interdot
        + (len(text_lines) - 1) * cell_geometry.interline
    )

def get_plate_width(text_lines, cell_geometry):
    max_len = max([len(t) for t in text_lines])
    return (
        2 * cell_geometry.horizontal_interdot
        + cell_geometry.horizontal_interdot
        + (max_len - 1) * cell_geometry.intercell
    )

# Get cylinder radius for the Braille dots
def get_cylinder_radius(cell_geometry):
    h = cell_geometry.dot_height
    r = cell_geometry.dot_diameter / 2
    return (r**2 + h**2) / 2 / h

# Base thickness calculation
def get_base_plate_thickness(plate_thickness, cell_geometry):
    return (
        plate_thickness + get_cylinder_radius(cell_geometry) - cell_geometry.
 ↪dot_height
    )

# Create the base of the plate
def make_base(text_lines, cell_geometry, plate_thickness):
    base_width = get_plate_width(text_lines, cell_geometry)
    base_height = get_plate_height(text_lines, cell_geometry)
    base_thickness = get_base_plate_thickness(plate_thickness, cell_geometry)
    base = cq.Workplane("XY").box(
        base_width, base_height, base_thickness, centered=False
    )
    return base

# Create the embossed Braille plate with dots as spherical caps
def make_embossed_plate(text_lines, cell_geometry):
    base = make_base(text_lines, cell_geometry, base_thickness)
```

```
    dot_pos = []
    base_width = get_plate_width(text_lines, cell_geometry)
    base_height = get_plate_height(text_lines, cell_geometry)
    y = base_height - 3 * cell_geometry.vertical_interdot
    line_start_pos = Point(cell_geometry.horizontal_interdot, y)

    for text in text_lines:
        dots = brailleToPoints(text, cell_geometry)
        dots = [p + line_start_pos for p in dots]
        dot_pos += dots
        line_start_pos += Point(0, -cell_geometry.interline)

    r = get_cylinder_radius(cell_geometry)
    base = (
        base.faces(">Z")
        .vertices("<XY")
        .workplane()
        .pushPoints(dot_pos)
        .circle(r)
        .extrude(r)
    )

    # Make a fillet almost the same radius to get a pseudo spherical cap.
    base = base.faces(">Z").edges().fillet(r - 0.001)

    # Create a "hiding box" to hide the lower parts of the cylinders
    hidding_box = cq.Workplane("XY").box(
        base_width, base_height, base_thickness, centered=False
    )
    result = hidding_box.union(base)

    return result

# Geometry setup for Braille cells
_cell_geometry = BrailleCellGeometry(
    horizontal_interdot,
    vertical_interdot,
    horizontal_intercell,
    vertical_interline,
    dot_height,
    dot_diameter,
)

# Ensure base thickness is sufficient
if base_thickness < get_cylinder_radius(_cell_geometry):
    raise ValueError("Base thickness should be at least {}".
 ↪format(get_cylinder_radius(_cell_geometry)))
```

```
# Generate the embossed plate
result = make_embossed_plate(text_lines, _cell_geometry)

# Display the result
show_object(result)
```

[ ]:
```
#classic OCP Bottle
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

(L, w, t) = (20.0, 6.0, 3.0)
s = cq.Workplane("XY")
# Draw half the profile of the bottle and extrude it
p = (
s.center(-L / 2.0, 0)
.vLine(w / 2.0)
.threePointArc((L / 2.0, w / 2.0 + t), (L, w / 2.0))
.vLine(-w / 2.0)
.mirrorX()
.extrude(30.0, True)
)
# Make the neck
p = p.faces(">Z").workplane(centerOption="CenterOfMass").circle(3.0).extrude(2.
 ↪0, True)
# Make a shell
result = p.faces(">Z").shell(0.3)

# show_object(p)
#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[ ]:
```
#cone with two circle
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

Cone = cq.Workplane("XY").circle(5).workplane(offset=5).circle(0.1).loft()

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(Cone)
```

[ ]:
```
#cone with height and radius
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

Cone = cq.Workplane("XY").lineTo(5,0).lineTo(0,15).close().
 ↪revolve(360,(0,0),(0,5))

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(Cone)
```

[ ]:
```
# cycloidal gear
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *
from math import sin, cos, pi, floor


# define the generating function
def hypocycloid(t, r1, r2):
    return (
        (r1 - r2) * cos(t) + r2 * cos(r1 / r2 * t - t),
        (r1 - r2) * sin(t) + r2 * sin(-(r1 / r2 * t - t)),
    )
def epicycloid(t, r1, r2):
    return (
        (r1 + r2) * cos(t) - r2 * cos(r1 / r2 * t + t),
        (r1 + r2) * sin(t) - r2 * sin(r1 / r2 * t + t),
    )
def gear(t, r1=4, r2=1):
    if (-1) ** (1 + floor(t / 2 / pi * (r1 / r2))) < 0:
        return epicycloid(t, r1, r2)
    else:
        return hypocycloid(t, r1, r2)

# create the gear profile and extrude it
result = (
    cq.Workplane("XY")
    .parametricCurve(lambda t: gear(t * 2 * pi, 6, 1))
    .twistExtrude(15, 90)
    .faces(">Z")
    .workplane()
    .circle(2)
    .cutThruAll()
)
```

```
#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[ ]:
```
#cylinder tube
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

cylinder_tube = cq.Workplane("XY").center(0,5).circle(3).circle(5).extrude(100)

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(cylinder_tube)
```

[ ]:
```
#cylinder
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

cylinder = cq.Workplane("XY").center(0,5).circle(3).extrude(10)

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(cylinder)
```

[ ]:
```
# lego brick
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

#####
# Inputs
######
lbumps = 6 # number of bumps long
wbumps = 2 # number of bumps wide
thin = True # True for thin, False for thick
#
# Lego Brick Constants-- these make a Lego brick a Lego :)
#
pitch = 8.0
clearance = 0.1
bumpDiam = 4.8
```

```
bumpHeight = 1.8
if thin:
    height = 3.2
else:
    height = 9.6

t = (pitch - (2 * clearance) - bumpDiam) / 2.0
postDiam = pitch - t # works out to 6.5
total_length = lbumps * pitch - 2.0 * clearance
total_width = wbumps * pitch - 2.0 * clearance

# make the base
s = cq.Workplane("XY").box(total_length, total_width, height)

# shell inwards not outwards
s = s.faces("<Z").shell(-1.0 * t)

# make the bumps on the top
s = (
    s.faces(">Z")
    .workplane()
    .rarray(pitch, pitch, lbumps, wbumps, True)
    .circle(bumpDiam / 2.0)
    .extrude(bumpHeight)
)
# add posts on the bottom. posts are different diameter depending on geometry
# solid studs for 1 bump, tubes for multiple, none for 1x1
tmp = s.faces("<Z").workplane(invert=True)
if lbumps > 1 and wbumps > 1:
    tmp = (
        tmp.rarray(pitch, pitch, lbumps - 1, wbumps - 1, center=True)
        .circle(postDiam / 2.0)
        .circle(bumpDiam / 2.0)
        .extrude(height - t)
)
elif lbumps > 1:
    tmp = (
        tmp.rarray(pitch, pitch, lbumps - 1, 1, center=True)
        .circle(t)
        .extrude(height - t)
)
elif wbumps > 1:
    tmp = (
        tmp.rarray(pitch, pitch, 1, wbumps - 1, center=True)
        .circle(t)
        .extrude(height - t)
)
```

```
else:
    tmp = s

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(tmp)
```

[ ]:
```
#loft geometry
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

result = (
cq.Workplane("front")
.box(4.0, 4.0, 0.25)
.faces(">Z")
.circle(1.5)
.workplane(offset=3.0)
.rect(0.75, 0.5)
.loft(combine=True)
)

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[1]:
```
# parametric box
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

# Parameter definitions
p_outerWidth = 100.0  # Outer width of the box enclosure
p_outerLength = 150.0  # Outer length of the box enclosure
p_outerHeight = 50.0  # Outer height of the box enclosure
p_thickness = 3.0  # Thickness of the box walls
p_sideRadius = 10.0  # Radius for the curves around the sides of the box
p_topAndBottomRadius = 2.0  # Radius for the curves on the top and bottom edges␣
 ↪of the box
p_screwpostInset = 12.0  # Distance from edges for screw posts
p_screwpostID = 4.0  # Inner diameter of the screw post holes
p_screwpostOD = 10.0  # Outer diameter of screw posts
p_boreDiameter = 8.0  # Diameter of the counterbore hole
p_boreDepth = 1.0  # Depth of the counterbore hole
p_countersinkDiameter = 0.0  # Outer diameter of countersink
```

```
p_countersinkAngle = 90.0  # Countersink angle
p_flipLid = True  # Whether to flip the lid upside down
p_lipHeight = 1.0  # Height of lip on the underside of the lid

# Step 1: Create the outer shell
oshell = (
    cq.Workplane("XY")
    .rect(p_outerWidth, p_outerLength)
    .extrude(p_outerHeight + p_lipHeight)  # Extrude the box height including lip
)

# Apply fillets in the correct order based on the radii comparison
if p_sideRadius > p_topAndBottomRadius:
    oshell = oshell.edges("|Z").fillet(p_sideRadius)
    oshell = oshell.edges("#Z").fillet(p_topAndBottomRadius)
else:
    oshell = oshell.edges("#Z").fillet(p_topAndBottomRadius)
    oshell = oshell.edges("|Z").fillet(p_sideRadius)

# Step 2: Create the inner shell and subtract it from the outer shell
ishell = (
    oshell.faces("<Z")
    .workplane(p_thickness, True)
    .rect(p_outerWidth - 2 * p_thickness, p_outerLength - 2 * p_thickness)
    .extrude(p_outerHeight - 2 * p_thickness, combine=False)
)
ishell = ishell.edges("|Z").fillet(p_sideRadius - p_thickness)

# Subtract the inner shell to form the box
box = oshell.cut(ishell)

# Step 3: Create the screw posts
POSTWIDTH = p_outerWidth - 2 * p_screwpostInset
POSTLENGTH = p_outerLength - 2 * p_screwpostInset
box = (
    box.faces(">Z")
    .workplane(-p_thickness)
    .rect(POSTWIDTH, POSTLENGTH, forConstruction=True)
    .vertices()
    .circle(p_screwpostOD / 2.0)
    .circle(p_screwpostID / 2.0)
    .extrude(-1.0 * (p_outerHeight + p_lipHeight - p_thickness), True)
)

# Step 4: Split the lid into top and bottom parts
(lid, bottom) = (
    box.faces(">Z")
```

```
    .workplane(-p_thickness - p_lipHeight)
    .split(keepTop=True, keepBottom=True)
    .all()
)

# Step 5: Translate the lid and cut to form the lid inset
lowerLid = lid.translate((0, 0, -p_lipHeight))
cutlip = lowerLid.cut(bottom).translate(
    (p_outerWidth + p_thickness, 0, p_thickness - p_outerHeight + p_lipHeight)
)

# Step 6: Create centers for screw holes in the lid
topOfLidCenters = (
    cutlip.faces(">Z")
    .workplane(centerOption="CenterOfMass")
    .rect(POSTWIDTH, POSTLENGTH, forConstruction=True)
    .vertices()
)

# Step 7: Add the holes (counterbore or countersink)
if p_boreDiameter > 0 and p_boreDepth > 0:
    topOfLid = topOfLidCenters.cboreHole(
        p_screwpostID, p_boreDiameter, p_boreDepth, 2 * p_thickness
    )
elif p_countersinkDiameter > 0 and p_countersinkAngle > 0:
    topOfLid = topOfLidCenters.cskHole(
        p_screwpostID, p_countersinkDiameter, p_countersinkAngle, 2 * p_thickness
    )
else:
    topOfLid = topOfLidCenters.hole(p_screwpostID, 2 * p_thickness)

# Step 8: Flip the lid upside down if required
if p_flipLid:
    topOfLid = topOfLid.rotateAboutCenter((1, 0, 0), 180)

# Step 9: Combine the lid and bottom to form the final box result
result = topOfLid.union(bottom)

# Display the result
show_object(result)
```

[ ]:
```
#rect tube
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

rect_tube = cq.Workplane("XY").rect(4,5).rect(3,4).extrude(50)
```

```
#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(rect_tube)
```

[ ]:
```
# sphere
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

Sphere = cq.Workplane("XY").sphere(5)

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(Sphere)
```

[ ]:
```
#spacer
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

c = cq.Workplane("XY").box(1, 1, 1).faces(">Z").workplane().circle(0.25).
 ↪cutThruAll()
# now cut it in half sideways
result = c.faces(">Y").workplane(-0.5).split(keepTop=True)

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(c)
show_object(result)
```

[ ]:
```
# donut
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *


# Parameters for the torus
major_radius = 10.0  # Major radius (distance from the center of the torus to␣
 ↪the center of the tube)
minor_radius = 1.0  # Minor radius (radius of the tube)

# Draw a 2D circle for the minor radius (the tube of the torus)
torus_profile = cq.Workplane("XZ").center(major_radius, 0).circle(major_radius)
```

```
# Revolve the profile 360 degrees around the Z-axis to create the torus
torus = torus_profile.revolve(angleDegrees=360, axisStart=(-50, 0, 0),␣
 ↪axisEnd=(-50, 1, 0))


#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(torus)
```

[ ]:
```
#cylindrical_rect spacer
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

# cylinder = cq.Workplane('XY').circle(1.5).extrude(3)
result = (
    cq.Workplane("front")
    .circle(2.0)
    .rect(0.5, 0.75)
    .extrude(0.5)
)

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[ ]:
```
#spur gear
import cadquery as cq
from ocp_vscode import *
from cadquery import exporters
from math import *

# Gear parameters
module = 2        # Module (mm)
teeth_number = 20    # Number of teeth
thickness = 12        # Gear thickness (mm)
bore_diameter = 8     # Center hole diameter (mm)
pressure_angle = 20   # Pressure angle (degrees)
clearance = 1.5      # Clearance (mm)
backlash = 0.1       # Backlash (mm)

# Calculations for different circles
P_circle = module * teeth_number
A_circle = P_circle + 2 * module
```

```
D_circle = P_circle - 2.5 * module
B_circle = D_circle + 2 * clearance

# Deep calculation for flank geometry
pitch = 1/module
theta = radians(180/teeth_number)
theta1 = radians(90) + theta
offset = teeth_number / pitch * sin(pi/2/teeth_number) / 2 - 7 * backlash / 4
flank_radius = teeth_number / pitch / 5

# Convert polar coordinates to Cartesian
cache_radius = sqrt(P_circle**2 - offset**2)
x1 = cache_radius * cos(theta1)
y1 = cache_radius * sin(theta1)

di_radius = sqrt((0 - x1)**2 + (P_circle - y1)**2)
theta2 = di_radius / theta * (di_radius - offset) + radians(90)
x2 = P_circle / 2 * cos(theta2)
y2 = P_circle / 2 * sin(theta2)

# Flank center
theta3 = (pi) * theta + radians(90)
x3 = B_circle / 2 * cos(theta3)
y3 = B_circle / 2 * sin(theta3)

# Function to calculate intersection points of two circles
def get_circle_intersections(circle1_center, circle1_radius, circle2_center,␣
 ↪circle2_radius):
    x1, y1 = circle1_center
    x2, y2 = circle2_center
    r1 = circle1_radius
    r2 = circle2_radius
    d = sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

    if d > r1 + r2 or d < abs(r1 - r2):
        return []
    if d == 0 and r1 == r2:
        return []

    a = (r1 ** 2 - r2 ** 2 + d ** 2) / (2 * d)
    h = sqrt(r1 ** 2 - a ** 2)

    x3 = x1 + a * (x2 - x1) / d
    y3 = y1 + a * (y2 - y1) / d

    intersection1 = (x3 + h * (y2 - y1) / d, y3 - h * (x2 - x1) / d)
    intersection2 = (x3 - h * (y2 - y1) / d, y3 + h * (x2 - x1) / d)
```

```
    return [intersection1, intersection2]

# Calculate intersection points
intersection1 = get_circle_intersections((0, 0), D_circle / 2, (x3, y3),␣
 ↪flank_radius)[0]
x4, y4 = intersection1
intersection2 = get_circle_intersections((0, 0), A_circle / 2, (x3, y3),␣
 ↪flank_radius)[0]
x5, y5 = intersection2

# Function to create the profile for a single tooth
def create_tooth_profile(A_circle, D_circle, flank_radius, x4, y4, x5, y5):
    tooth0 = (
        cq.Workplane("XY")
        .moveTo(0, D_circle / 2)
        .lineTo(0, A_circle / 2)
        .radiusArc((x5, y5), -A_circle / 2)
        .radiusArc((x4, y4), flank_radius)
        .radiusArc((0, D_circle / 2), D_circle / 2)
        .close()
        .extrude(thickness)
    )
    tooth1 = tooth0.mirror(mirrorPlane="YZ", basePointVector=(0, 0, 0))
    tooth = tooth0.union(tooth1)
    return tooth

# Create a single tooth profile
tooth_profile = create_tooth_profile(A_circle, D_circle, flank_radius, x4, y4,␣
 ↪x5, y5)

# Function to pattern the tooth profile around the gear
def pattern_teeth(tooth_profile, teeth_number, P_circle):
    """
    Pattern the teeth profile around the gear.
    :param tooth_profile: The single tooth profile to be patterned.
    :param teeth_number: Total number of teeth to pattern.
    :param P_circle: Pitch circle diameter.
    :return: Gear with patterned teeth.
    """
    # Create an empty workplane for the gear with patterned teeth
    gear = cq.Workplane("XY")

    # Calculate the rotation angle between each tooth
    rotation_angle = 360 / (teeth_number)

    for i in range(teeth_number):
```

```
        # First, position the tooth at the correct radial distance (pitch circle␣
↪radius)
        positioned_tooth = tooth_profile.translate((0, 0, 0))

        # Then, rotate the tooth around the Z-axis by the necessary angle
        rotated_tooth = positioned_tooth.rotate((0, 0, 1), (0, 0, 0),␣
↪rotation_angle * (i))

        # Union each rotated tooth to the gear
        gear = gear.union(rotated_tooth)

    return gear



# Main gear body without teeth
main_body = (
    cq.Workplane("XY")
    .circle(A_circle / 2)
    .circle(bore_diameter)
    .extrude(thickness)

)

# Generate the patterned gear with teeth
gear_with_teeth = pattern_teeth(tooth_profile, teeth_number, A_circle)

gear55 = main_body.cut(gear_with_teeth)

exporters.export(gear55,'./gear.step')

# Display the result
show_object(gear55)
# show_object(tooth_profile)
```

[3]:
```
# line_arc geometry
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

# cylinder = cq.Workplane('XY').circle(1.5).extrude(3)
result = (
    cq.Workplane("front")
    .lineTo(2.0, 0)
    .lineTo(2.0, 1.0)
    .threePointArc((1.0, 1.5), (0.0, 1.0))
    .close()
```

```
    .extrude(0.25)
)

#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[ ]:
```
#pillow block
import cadquery as cq
from ocp_vscode import *
from cadquery import exporters

# %%
(length, height, diam, thickness, padding) = (30.0, 40.0, 22.0, 10.0, 8.0)
result = (
    cq.Workplane("XY")
    .box(length, height, thickness)
    .faces(">Z")
    .workplane()
    .hole(diam)
    .faces(">Z")
    .workplane()
    .rect(length - padding, height - padding, forConstruction=True)
    .vertices()
    .cboreHole(2.4, 4.4, 2.1)
)
#exporters.export(result,'./pillowblock608.step')
#exporters.export(result.section(), "result.dxf")

# Display the result
show_object(result)
```

[ ]:
```
#polygon
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

# cylinder = cq.Workplane('XY').circle(1.5).extrude(3)
result = (
cq.Workplane("front")
.box(3.0, 4.0, 0.25)
.pushPoints([(0, 0.75), (0, -0.75)])
.polygon(6, 1.0)
.cutThruAll()
)
```

```
#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[ ]:
```
# motor coupler
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

# cylinder = cq.Workplane('XY').circle(1.5).extrude(3)
r = cq.Workplane("front").circle(2.0) # make base
r = r.pushPoints(
[(1.5, 0), (0, 1.5), (-1.5, 0), (0, -1.5)]
) # now four points are on the stack
r = r.circle(0.25) # circle will operate on all four points
result = r.extrude(0.125) # make prism
result = (
    result
    .faces(">Z")
    .workplane()
    .circle(0.5)
    .extrude(2.5)
)


#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[ ]:
```
#circular battery
import cadquery as cq
from ocp_vscode import show_object

# # Parameters for the battery
# battery_length = 88  # Length of the battery (mm)
# battery_diameter = 16  # Diameter of the battery (mm)
# cap_height = 5  # Height of the cap (mm)

# Create the main body of the battery
def battery_func(L,H,D):
    battery_body = (
        cq.Workplane("XY")
        .circle(D)  # Base circle for the battery body
        .extrude(L)  # Extrude to create the battery body
    )
```

```
    # Create the top cap of the battery
    battery_cap = (
        battery_body
        .faces(">Z")
        .circle(D/ 2)  # Base circle for the cap
        .extrude(H)  # Extrude to create the cap
    )



    # Combine the battery body, cap, and terminal
    result = battery_body.union(battery_cap)
    return(result)
battery = battery_func(88,5,16)

# Display the final battery model
show_object(battery)
```

[5]:
```
#lithium ion battery
import cadquery as cq
from ocp_vscode import show_object

# # Parameters for the battery
# battery_length = 25  # Length of the battery (mm)
# battery_width = 15   # Width of the battery (mm)
# battery_height = 50  # Height of the battery (mm)

# Create the main body of the battery
def battery(battery_length,battery_width,battery_height):
    battery_body = (
        cq.Workplane("XY")
        .rect(battery_length, battery_width)  # Create a rectangular base for␣
 ↪the battery
        .extrude(battery_height)  # Extrude to the height of the battery
        .edges("|Z").fillet(1)
    )
    battery_body = (
        battery_body
        .faces(">Z")
        .workplane()
        .moveTo(7,0)
        .circle(3.5)
        .circle(3)
        .moveTo(-7,0)
        .polygon(6,7,False,False)
        .polygon(6,6.5,False,False)
```

```
        .extrude(2)
    )
    return (battery_body)


battery_body= battery(25,15,50)
# battery_body = (
#     battery_body
#     .faces("<Z")
#     .workplane()
#     .rect(battery_length-2,battery_width-2)
#     .cutBlind(2)
# )

# Display the final battery model
show_object(battery_body)
```

[ ]:
```
#complex channel
import cadquery as cq
from cadquery import exporters
from ocp_vscode import *

result0 = (
cq.Workplane("XY")
.moveTo(10, 0)
.lineTo(5, 0)
.threePointArc((3.9393, 0.4393), (3.5, 1.5))
.threePointArc((3.0607, 2.5607), (2, 3))
.lineTo(1.5, 3)
.threePointArc((0.4393, 3.4393), (0, 4.5))
.lineTo(0, 13.5)
.threePointArc((0.4393, 14.5607), (1.5, 15))
.lineTo(28, 15)
.lineTo(28, 13.5)
.lineTo(24, 13.5)
.lineTo(24, 11.5)
.lineTo(27, 11.5)
.lineTo(27, 10)
.lineTo(22, 10)
.lineTo(22, 13.2)
.lineTo(14.5, 13.2)
.lineTo(14.5, 10)
.lineTo(12.5, 10)
.lineTo(12.5, 13.2)
.lineTo(5.5, 13.2)
.lineTo(5.5, 2)
.threePointArc((5.793, 1.293), (6.5, 1))
```

```
.lineTo(10, 1)
.close()
)
result = result0.extrude(100)

result = result.rotate((0, 0, 0), (1, 0, 0), 90)
result = result.translate(result.val().BoundingBox().center.multiply(-1))
mirXY_neg = result.mirror(mirrorPlane="XY", basePointVector=(0, 0, -30))
mirXY_pos = result.mirror(mirrorPlane="XY", basePointVector=(0, 0, 30))
mirZY_neg = result.mirror(mirrorPlane="ZY", basePointVector=(-30, 0, 0))
mirZY_pos = result.mirror(mirrorPlane="ZY", basePointVector=(30, 0, 0))
result = result.union(mirXY_neg).union(mirXY_pos).union(mirZY_neg).
 ↪union(mirZY_pos)




#exporters.export(cylinder,'./cylinder.stl')

# Display the result
show_object(result)
```

[ ]:
```
#box cylinder
import cadquery as cq
from ocp_vscode import show_object

#Function for creating a box
def create_box(width: float, height: float, depth: float):
    box = cq.Workplane("XY").box(width, height, depth)
    return box

#Function for creating a cylinder
def create_cylinder(radius: float, height: float):
    cylinder = cq.Workplane("XY").circle(radius).extrude(height)
    return cylinder

merge = create_box(10, 10, 10).union(create_cylinder(5, 5).translate((0, 0, 5)))



# Display the final battery model
show_object(merge)
```