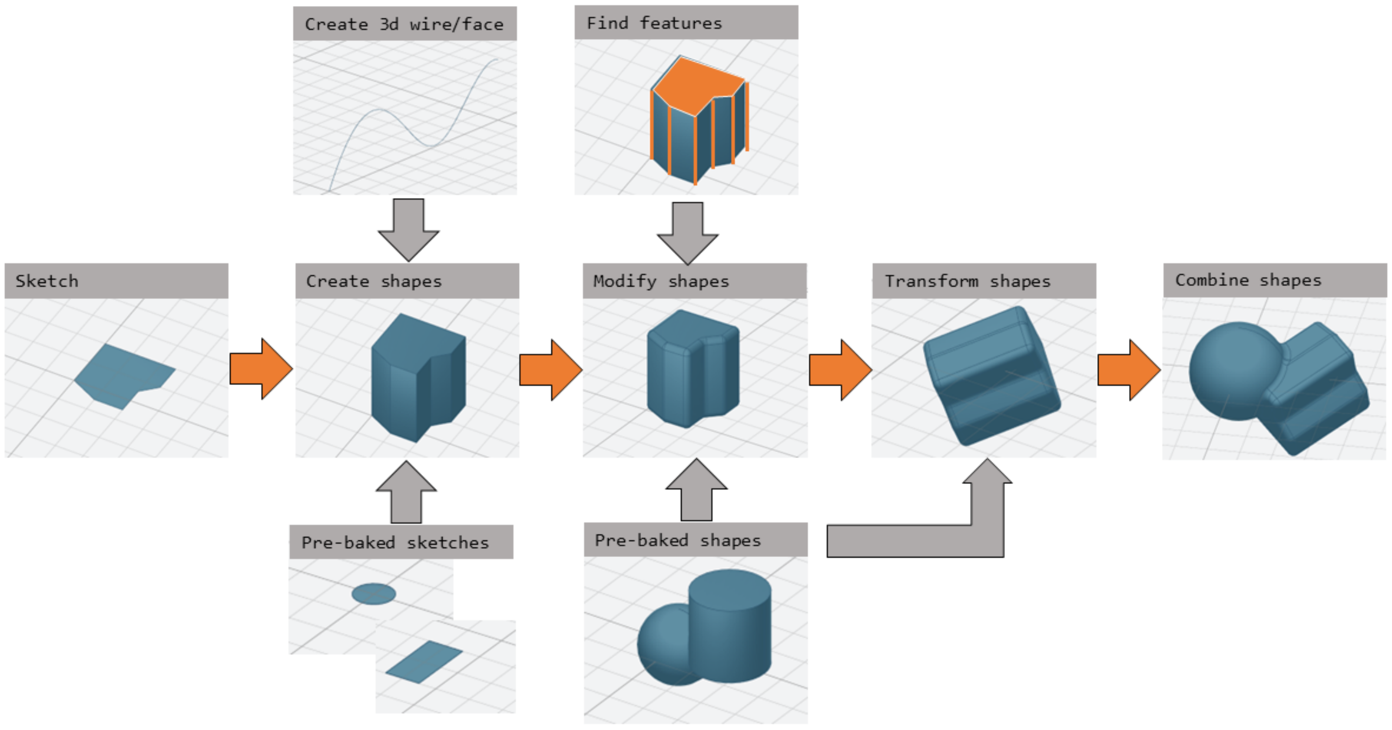
# Replicad Quick Reference

## 1. Process

The process to draw a shape in Replicad looks like this:



The "pre-baked" sketches can be used as a shortcut to speed-up the process.

## 2. File template

A model in Replicad is built using a javascript input file. The template for this file looks like this:

```
// next lines allow intellisense help in VS Code
/** @typedef { typeof import("replicad") } replicadLib */
/** @type {function(replicadLib, typeof defaultParams): any} */

function main(
{
    Sketcher,
    sketchRectangle,
    .. functions used in the code below ..
})
{
    // add code to describe the shape
return  shape   |
return  {shape: [shape], highlight: [foundFeature]}
}
```

Alternatively you can use the arrow notation for the javascript function

```
const main = (
  { Sketcher, sketchRectangle, ... },
  {}
) => {
    // add code to describe the shape
return  shape   |
return  {shape: [shape], highlight: [foundFeature]}
}
```

## 3. Sketch

To start a sketch, use the `new Sketcher` command

```
let sketch = new Sketcher("XZ",-5)
".sketchCommands"        (see below)
.close()                    // ends the sketch with line to starting point
.done()                     // ends the sketch without closing
.closeWithMirror()          // closes the sketch with mirror on axis from start to end
```

Use the following commands to describe the sketch:

```
".sketchCommands  = "
.movePointerTo([x,y])           // move pointer without drawing, can only be used at start
.lineTo([x,y])                  // line to absolute coordinates
.line(dx,dy)                    // line to relative coordinates
.vLineTo(y)                     // vertical line to absolute y
.vLine(dy)                      // vertical line to relative y
.hLineTo(x)                     // horizontal line to absolute x
.hLine(dx)                      // horizontal line to relative x
.polarLineTo([radius,theta])    // line to absolute polar coordinates
.polarLine(distance,angle)      // line to relative polar coordinates
.tangentLine(distance)          // tangent extension over distance
.threePointsArcTo(point_end,point_mid) // arc from current to end via mid, absolute
coordinates
.threePointsArc(dx,dy,dx_via,dy_via)   // arc from current to end via mid, relative
coordinates
.sagittaArcTo(point_end,sagitta)    // arc from current to end with sag , absolute
coordinates
.sagittaArc(dx,dy,sagitta)          // arc from current to end with sag, relative coordinates
.vSagittaArc(dy,sagitta)            // vertical line to endpoint with sag, relative y
.hSagittaArc(dx,sagitta)            // horizontal line to endpoint with sag, relative x
.tangentArcTo([x,y])                // arc tangent to current line to end, absolute
coordinates
.tangentArc(dx,dy)                  // arc tangent to current line to end, relative
coordinates
.ellipseTo([x,y],r_hor,r_vert)      // ellipse from current to end, absolute coordinates,
radii to hor and vert
.ellipse(dx,dy,r_hor,r_vert)        // ellipse from current to end, relative coordinates,
radii to hor and vert

// extra parameters ellipse: startangle, endangle, counterclockwise?
.halfEllipseTo([x,y],r_min)         // half ellipse with r_min as sag, absolute coordinates
.halfEllipse(dx,dy,r_min)// half ellipse with r_min as sag, relative coordinates
.bezierCurveTo([x,y],points[])      // Bezier curve to end along points[]
.quadraticBezierCurveTo([x,y],[x_ctrl,y_ctrl]) // Quadratic bezier curve to end with control
point
.cubicBezierCurveTo([x,y],p_ctrl_start,p_ctrl_end)
.smoothSplineTo([x,y],splineconfig) // smooth spline to end, absolute coordinates
.smoothSpline(dx,dy,splineconfig)// smooth spline to end, absolute coordinates
splineconfig = {startTangent:angle,endTangent:angle / "symmetric"}
```

## 4. Pre-baked sketches

```
sketchRectangle(length,width)
sketchRoundedRectangle(length,width,fillet,{plane:"XY",origin:dist|[point]})
sketchCircle(radius,{config})
// special case of creating a sketch/wire from a face
sketchFaceOffset(shape,thickness)
```

# 5. Create 3D face/wire

## 5.1. Create wires in 3D

In comparison to sketches which create wires or faces in 2D

```
makeLine([point],[point])
makeCircle(radius,[center],[normal])
makeEllipse(major,minor,[center],[normal])
makeHelix(pitch,height,radius,[center],[dir],lefthand?)
makeThreePointArc([point1],[point2],[point3])
makeEllipseArc(major,minor,anglestart,angleEnd,[center],[normal],[xDir?])
makeBSplineApproximation([points[]])
makeBezierCurve([points[]])
makeTangentArc([startPoint],[tangentPoint],[endPoint])
```

## 5.2. Create faces in 3D

```
makeFace(wire)
makeNewFaceWithinFace(face,wire)
makeNonPlanarFace(wire)
makePolygon(points[])
makeOffset(face,offset,tolerance)
MakePlaneFromFace()
```

## 6. Create shapes

```
shape = sketch."thicknessCommand"

"thicknessCommand ="
.face()             // create a face from the sketch

.extrude(distance,extrusionConfig?)

            extrusionConfig = {     extrusionDirection:[point],
                                    ExtrusionProfile:ExtrusionProfile,
                                    origin:[point],
                                    twistAngle:deg}

            extrusionProfile: {     profile:"linear" | "s-curve",
                                    endFactor: scale}

.loftWith([otherSketches],loftConfig,returnShell?)

            loftConfig =        {   endPoint:[point],
                                    ruled: boolean,
                                    startPoint:[point]}

.revolve(revolutionAxis:[point],config?)    // default is z-axis

            config      =       origin:[point]

.sweepSketch((plane, origin) => sketchFunction(plane,origin));

            function sketchFunction(plane,origin)
            {let section = new Sketcher(plane,origin)
                    (add sketch commands)
                    .close()
            return section}

            sketchRectangle(2, 30, { plane, origin })

makeSolid(faces[]|shell)
```

## 7. Pre-baked shapes

```
makeCylinder(radius,height,[location],[direction])
makeSphere(radius)
makeVertex([point])
```

## 8. Modify shapes

```
.chamfer(radiusConfig,filter?)
.fillet(radiusConfig,filter?)
.shell(thickness, (f) => f.inPlane("YZ",-20),{tolerance:number})

                    radiusConfig    = number or func
                    filter          = (e) => e.Edgefinder

makeOffset(shape,thickness)
addHolesInFace(face,holeWires[])
```

## 9. Find features

### 9.1. Faces

```
let foundFaces = new FaceFinder().inPlane("XZ",35)
```

```
inPlane("XZ",35)
ofSurfaceType("CYLINDRE")
        "PLANE"|"CYLINDRE"|"CONE"|"SPHERE"|"TORUS"|"BEZIER_SURFACE"|
        "BSPLINE_SURFACE"|"REVOLUTION_SURFACE"|"EXTRUSION_SURFACE"|
        "OFFSET_SURFACE"|"OTHER_SURFACE"
containsPoint([0,-15,80])
atAngleWith(direction,angle)    // atAngleWith("Z",20)
atDistance(distance,point)      //
inBox(corner1,corner2)
inList(elementList[])
inPlane(inputPlane,origin)      // inPlane("XY",30)
parallelTo(plane|face|standardplane)

and

either
            const houseSides = new FaceFinder().either([
            (f) => f.inPlane("YZ", 50),
            (f) => f.inPlane("YZ", -50),]);
not
            const frontWindow = new EdgeFinder()
            .ofCurveType("CIRCLE")
            .not((f) => f.inPlane("XZ"));

find(shape,options)         // returns all the elements that fit the filters
            options {unique: true}

            new FaceFinder().inPlane("XZ", 30).find(house)
```

### 9.2. Edges

Todo

## 10. Transform shapes

The transform functions require a shape or face. A sketch cannot be transformed, with the exception of creating an offset.

```
transformedShape = shape."transformCommand"

"transformCommand = "
.translate([dx,dy,dz])
.translateX(dx)
.translateY(dy)
.translateZ(dz)
.rotate(angleDeg,axisOrigin[x,y,x],axisEnd[x,y,x])
.scale(number)
.mirror("YZ",[-10,0])
.clone()
```

## 11. Combine shapes

```
.cut(tool,{optimisation:"none" | "commonFace" | "sameFace"})
.fuse(otherShape,.. )
.intersect(tool)

compoundShapes(shapeArray[])
makeCompound(shapeArray[])
```

todo