`build123d` Cheat Sheet

```
from build123d import *
```

Geometry Primitives

- **Vector** — `Vector(1,2,3)` · `.X .Y .Z` · `.length` · `.normalized()` · `.dot()` · `.cross()`
- **Axis** — `Axis.X / .Y / .Z` · `Axis((0,0,5), (0,1,0))`
- **Pos/Rot** — `Pos(x,y,z)` · `Pos(x=5)` · `Rot(Z=90)` · `Rot(rx,ry,rz)`
- **Location** — `Location((x,y,z),(rx,ry,rz))` · `loc1 * loc2` · `loc.inverse()`
- **Plane** — `Plane.XY / .XZ / .YZ` · `Plane.XY.offset(10)` · `Plane(face)`
- **Units** — `MM CM M IN FT` — e.g. `2 * IN`

Shape Constructors

3D → `Part`

- `Box(l, w, h, align=CENTER×3)`
- `Cylinder(r, h, arc_size=360)`
- `Cone(r1, r2, h)` — use `r2=0` for a true cone
- `Sphere(r)`
- `Torus(major_r, minor_r)`
- `Wedge(xsize, ysize, zsize, xmin, zmin, xmax, zmax)`

`align` values: `Align.MIN / CENTER / MAX` — one per axis,
e.g. `(Align.CENTER, Align.CENTER, Align.MIN)` puts the bottom at Z=0.

2D → `Sketch` {#2d–sketch}

```
Rectangle(w, h)          RectangleRounded(w, h, r)    Circle(r)
Ellipse(rx, ry)          RegularPolygon(r, n)          Polygon(*pts)
Text(s, font_size)       Trapezoid(w, h, angle)
Triangle(a,b,c,A,B,C) SlotOverall(w, h)        SlotCenterToCenter(sep, h)
SlotArc(arc, h)
```

1D → `Curve` {#1d–curve}

```
Line(p1, p2)             PolarLine(start, len, angle)  Polyline(*pts)
CenterArc(c, r, a0, da)  ThreePointArc(p1, p2, p3)    RadiusArc(p1, p2,
r) SagittaArc(p1, p2, sag)  TangentArc(p1, p2, tangent) Spline(*pts,
tangents, periodic)                          Bezier(*pts) Helix(pitch, h,
r, lefthand=False)
```

Close and fill a curve:

```
wire = Polyline((0,0),(10,0),(10,5),(0,5),(0,0))
face = make_face(wire)             # Sketch
```

Operators

- `a + b` — union (same-dimension objects)
- `a - b` — cut (not commutative)
- `a & b` — intersect
- `loc * shape` — position shape, e.g. `Pos(x,y,z) * Box(...)`
- `plane * shape` — place in plane's frame, e.g. `Plane.XZ * Cylinder(3,10)`
- `edge @ t` — point at parameter `t` in `[0,1]` ⟶ `Vector`
- `edge % t` — tangent direction at `t` ⟶ `Vector`
- `edge ^ t` — `Location` at `t` (pos + frame); use to align sweep sections

```
# Fuse a list:  Part() is the neutral element
combined = Part() + [Pos(i*15, 0) * Box(10,10,5) for i in range(4)]
```

Sketch → Solid

```
extrude(sketch, amount=5)                        # along face normal
extrude(sketch, amount=10, taper=5)             # with draft angle
extrude(sketch, amount=10, both=True)           # both directions
extrude(circle, until=Until.LAST, target=solid)  # up-to target
```

```
revolve(face, axis=Axis.Z, revolution_arc=360)    # rotate profile

loft([sketch1, sketch2, ...], ruled=False)        # blend sections

path = Spline((0,0,0), (20,0,20), (40,0,0))
section = (path ^ 0) * Circle(3)                  # align to path start
sweep(section, path=path)                          # pipe along path
```

Topology Selectors (`ShapeList`)

```
part.vertices()   part.edges()   part.wires()   part.faces()   part.solids()
```

- `.filter_by(Axis.Z)` — edges/faces parallel to axis
- `.filter_by(GeomType.CIRCLE)` — by geometry type (`PLANE`, `CIRCLE`, ...)
- `.filter_by_position(Axis.Z, lo, hi)` — by centroid coordinate range
- `.sort_by(Axis.X)` — sort by position along axis
- `.sort_by(SortBy.AREA)` — sort by area / `LENGTH` / `RADIUS`
- `.group_by(SortBy.LENGTH)[-1]` — group; last group = longest
- `.first` / `.last` — first/last after sort

Modifications

```
# Fillet / Chamfer
fillet(part.edges(), radius=2)
fillet(part.edges().filter_by(Axis.Z), radius=2)    # only vertical edges
chamfer(part.edges(), length=1)
chamfer(part.edges(), length=2, length2=1)           # asymmetric

# Offset / Shell
offset(box, amount=2)                                 # grow solid
offset(box, amount=-2, openings=box.faces().sort_by(Axis.Z).last)  # hollow

# Mirror
full = half + mirror(half, about=Plane.YZ)

# Scale
scale(shape, by=2)              # uniform
scale(shape, by=(2, 1, 3))    # non-uniform

# Split
hemisphere = split(Sphere(10), bisect_by=Plane.XY, keep=Keep.TOP)
# keep: Keep.TOP / BOTTOM / BOTH

# Cross-section
sketch = section(part, section_by=Plane.XZ, height=5)
```

Holes

```
top = Plane(part.faces().sort_by(Axis.Z).last)   # plane on top face

part -= top * Hole(radius=3)                      # through-hole
part -= top * Pos(10, 10) * CounterBoreHole(radius=2, counter_bore_radius=4,
                                            counter_bore_depth=3, depth=8)
part -= top * Pos(5, 5)  * CounterSinkHole(radius=1.5, counter_sink_radius=3,
                                           depth=8, counter_sink_angle=82)
# depth=None → through all
```

Location Generators

```
GridLocations(x_spacing, y_spacing, x_count, y_count)
PolarLocations(radius, count, start_angle=0, stop_angle=360, rotate=True)
HexLocations(apothem, x_count, y_count)
Locations(*pts)                          # explicit (x,y,z) tuples

# Usage:
locs = PolarLocations(radius=20, count=6)
part -= [top * loc * Hole(radius=2, depth=5) for loc in locs]
```

Placing on Faces

```
base = Box(20, 20, 10)
top_plane = Plane(base.faces().sort_by(Axis.Z).last)
result = base + top_plane * Cylinder(3, 5)       # boss on top face
```

Assemblies & Color

```
assy = Compound([axle, wheel_l, wheel_r])
assy.move(Pos(0, 0, 50))

part.color = Color("steelblue")
part.color = Color(0.2, 0.6, 1.0)          # RGB 0..1
part.color = Color("red", alpha=0.5)
```

Import / Export

```
export_step(part, "out.step");   part = import_step("in.step")
export_stl(part, "out.stl", linear_deflection=0.1)
```

```
export_brep(part, "out.brep");   export_gltf(part, "out.gltf")
export_svg(sketch, "out.svg");   curve = import_svg("logo.svg")
```

Visualize (ocp-vscode):

```
from ocp_vscode import show
show(part)
show(a, b, c, names=["a","b","c"])
```

Common Patterns

```
# Rounded box
box = fillet(Box(20, 20, 10).edges(), radius=2)

# Hollow box (shell)
box = offset(Box(20, 20, 10), amount=-2, openings=Box(20,20,10).faces().sort_by(Axis.Z).last)

# Pipe along a path
path = Spline((0,0,0), (30,0,20), (60,0,0))
pipe = sweep((path ^ 0) * Circle(3), path=path)

# Polar hole pattern
plate = Box(60, 60, 5)
top = Plane(plate.faces().sort_by(Axis.Z).last)
plate -= [top * loc * Hole(radius=2, depth=5) for loc in PolarLocations(20, 6)]

# Revolve profile → vase
profile = Plane.XZ * make_face(Spline(...) + Line(...))
vase = revolve(profile, axis=Axis.Z)
vase = offset(vase, amount=-2, openings=vase.faces().filter_by(GeomType.PLANE))
```