Python Examples

Python is a popular graphics scripting language. In these examples the Python plugin to Grasshopper is illustrated. There is also a Python scripting tool built directly into Rhino and which can be used without Grasshopper.

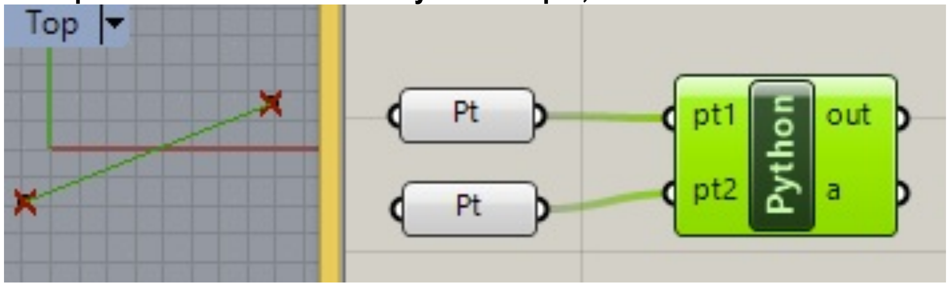
Python is a so-called “high level” language, where syntax has been somewhat simplified so that an expert in some applied area can readily create programs without having to address “lower level” considerations at a very detailed level, such as with respect to computer memory management.

For a quick on-line introduction and installation instructions see:
https://developer.rhino3d.com/guides/rhinopython/your-first-python-script-in-grasshopper/

Note that Grasshopper Plugin to Rhino including Python is automatically included in Rhino 6.0.

These examples range from a simple one line program to more complex examples.

Example 1: Trivial Case of Two Python Scripts, one with an internal function definition



1.a. Python code draws a line without an internal function:

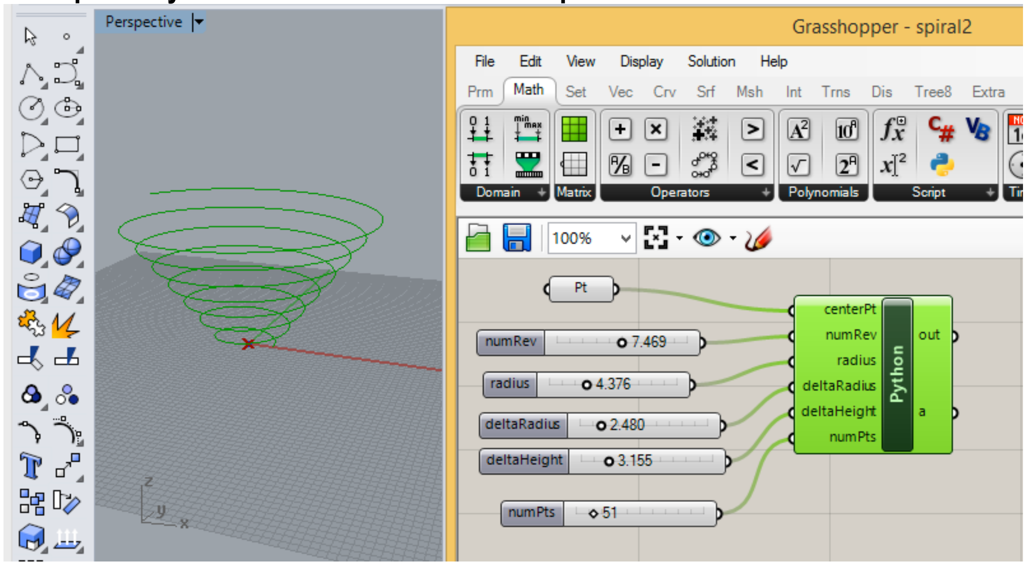
This is a kind of “hello world” example (typically the first program you would learn to write in any scripting language). Comments in the program are preceded by a # symbol.

```
#Draws a line and assigns it to output variable a
import rhinoscriptsyntax as rs

a = rs.AddLine(pt1, pt2)
```

1.b. Alternative Python code draws a line with an internal function “drawLine”:

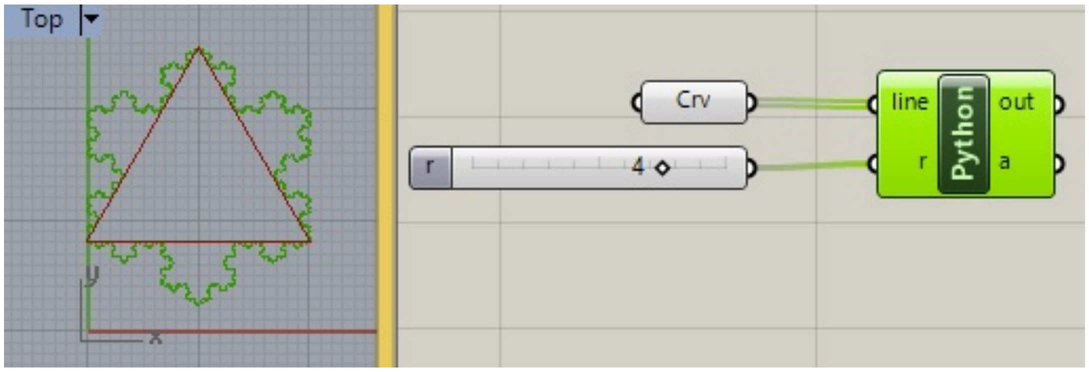
```
#Draws a line and assigns it to output variable a
import rhinoscriptsyntax as rs

def drawLine(pt1, pt2):
    line = rs.AddLine(pt1, pt2)
    return line

#main
a = drawLine(pt1, pt2)
```

Example 2: Python code with an iterative loop



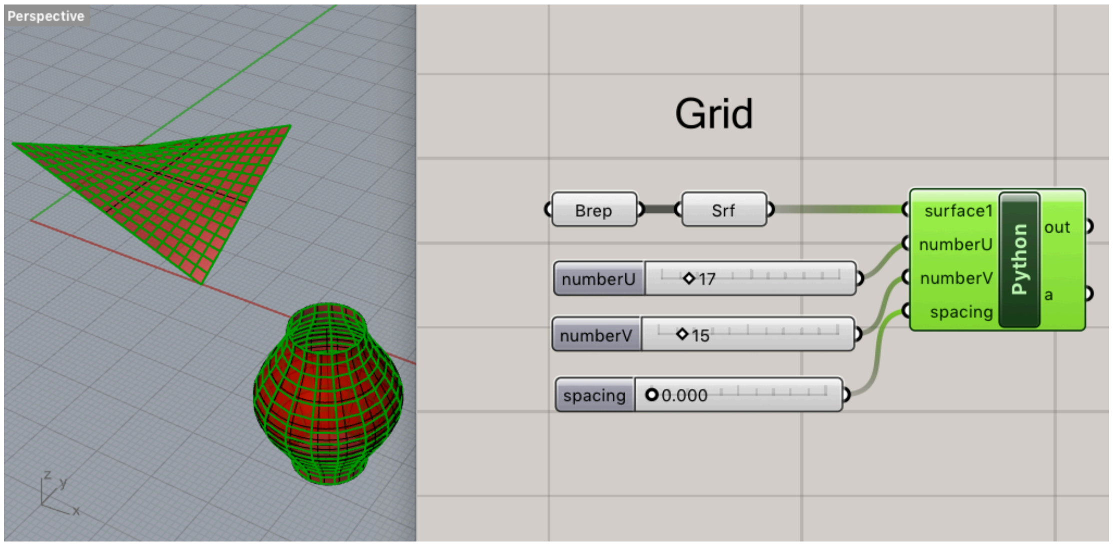
```
#draw a spiral from a centerPt using rhino library and trig functions
import rhinoscriptsyntax as rs
import math

#input arguments: numrev = revolutions in spiral; radius = start radius
#deltaRadius = radial increment per revolution; deltaHeight = height
#increment per revolution, numPts = control points per revolution
#output variable:
# a = Archimedes spiral (if flat) or Archimedes Curve

#determine incremental angle, radius and height between control points
deltaAngle = 360 / numPts
radiusIncrement = deltaRadius / numPts
heightIncrement = deltaHeight / numPts

#determine start angle and height
angle = 0
height = centerPt.Z

#setup list container for vertices of the spiral
ptList = []

#continue to add vertices to spiral while less than desired revolutions
while (angle < numRev * 360):
    y = centerPt.Y + math.sin(math.radians(angle)) * radius
    x = centerPt.X + math.cos(math.radians(angle)) * radius
    z = height
    curPt = rs.AddPoint(x, y, z)
    ptList.append(curPt)
    radius += radiusIncrement
    angle += deltaAngle
    height += heightIncrement

#create a polygon line from the control points
a = rs.AddPolyline(ptList)
```

Example3: Koch curve recursive example



```
# Creates Koch curve fractal lines by recursive function (see also Sierpinsky Triangle
(https://en.wikipedia.org/wiki/Sierpinski_triangle)
# modifed after http://atlv.org/education/ghpython/
# input: Line(s), r = recursion depth;  output: Koch curve

import rhinoscriptsyntax as rs
import Rhino.Geometry as rg
import math

def recursiveLine(line, depth, resultList):
   #select points from line at ends & 1/3rd and 2/3rd along its distance
   pt1 = line.PointAt(0)
   pt2 = line.PointAt(1.0/3.0)
   pt3 = line.PointAt(2.0/3.0)
   pt4 = line.PointAt(1.0)

   #determine vector from middle points
   dir = rg.Vector3d(pt3.X - pt2.X, pt3.Y - pt2.Y, pt3.Z - pt2.Z)

   #rotate vector around z axis
   dir.Rotate(math.radians(60), rg.Vector3d.ZAxis)

   #add new point based upon rotated vector
   pt5 = pt2 + dir

   #create lines from new points
   line1 = rg.Line(pt1, pt2)
   line2 = rg.Line(pt2, pt5)
   line3 = rg.Line(pt5, pt3)
   line4 = rg.Line(pt3, pt4)

   if(depth>0): # recursion call if still have further to go
      recursiveLine(line1, depth-1, resultList)
      recursiveLine(line2, depth-1, resultList)
      recursiveLine(line3, depth-1, resultList)
      recursiveLine(line4, depth-1, resultList)
   else: # add lines to list when reaching final recursive depth depth
      resultList.append(line1)
      resultList.append(line2)
      resultList.append(line3)
      resultList.append(line4)

# main
a = [ ]
recursiveLine(line, r, a)
```

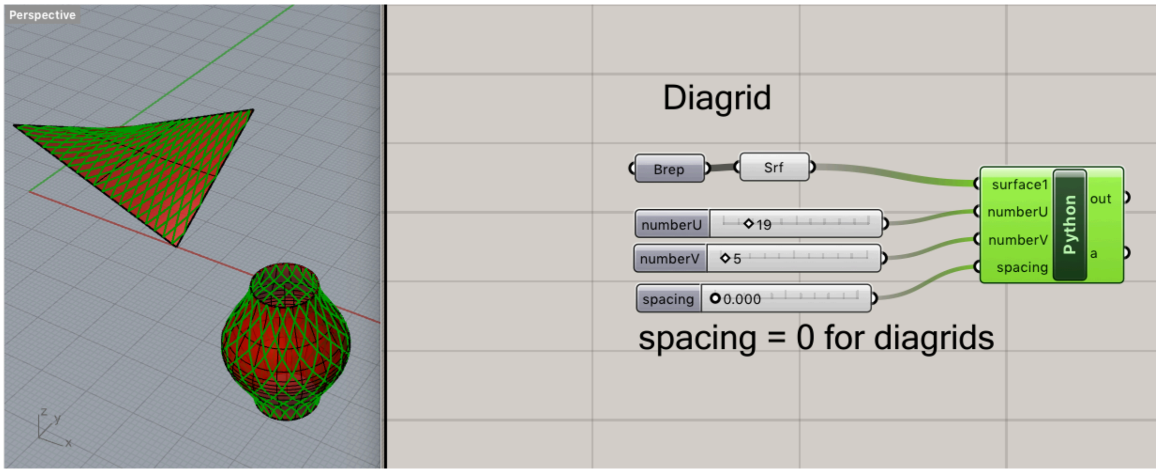
Example 4: Draw grid array on surface using parametric (0 – 1) values



```
#Place rectangle grid on surface input based upon reparameterization from 0 to 1
# See also www.grasshopper3d.com/profiles/blogs/python-snippet-reparameterizing-a-surface-using-python
#Earl Mark, November 4, 2020
import rhinoscriptsyntax as rs
import math
import Rhino as rc


def getParamPt(surface1, x, y):
#get parametric point on surface
   surfPt = rc.Geometry.Surface.PointAt(surface1, x, y)
   return surfPt

def drawRect(surface1, x, y, sizeX, sizeY):
#draw rectangle on surface at parametric location (0 to 1) x, y of sizes sizeX, sizeY
   ptList = [ ]
   pt1 = getParamPt(surface1,x, y);
   ptList.append(pt1)
   pt2 = getParamPt(surface1,x + sizeX, y);
   ptList.append(pt2)
   pt3 = getParamPt(surface1,x + sizeX, y + sizeY);
   ptList.append(pt3)
   pt4 = getParamPt(surface1,x, y + sizeY);
   ptList.append(pt4)
   ptList.append(pt1)
   pline = rs.AddPolyline(ptList)
   return pline

def drawRectGrid(surface1, sizeX, sizeY, spacingX, spacingY, numberHoriz, numberVert, polyLines):
   #draw rows and columns of a rectangular grid
   sizeX = sizeX - spacingX
   sizeY = sizeY - spacingY
   for i in range(0, int(numberVert), 1): #rows
      for ii in range(0, int(numberHoriz), 1): #columns
         x = ii * (sizeX + spacingX) + 0.5 * spacingX
         y = i * (sizeY + spacingY) + 0.5 * spacingY
         pline = drawRect(surface1, x, y, sizeX, sizeY)
```

```
        polyLines.append(pline)
    return polyLines

#main
polyLines = [ ]
#get size of surface in U and V direrctions
sizeU = surface1.Domain(0)
sizeV = surface1.Domain(1)

#parameterize surface from 0 to 1
surface1.SetDomain(0, rc.Geometry.Interval(0, 1))
surface1.SetDomain(1, rc.Geometry.Interval(0, 1))

#determine preliminary size of rectangles needed to fill surface
sizeX = 1/numberU
sizeY = 1/numberV

#initiate spacing parameter as a fraction of sizeX and sizeY at maximum value of 95%
spacingX = sizeX * spacing * 0.95
spacingY = sizeY * spacing * 0.95

#draw rectangles on surface
polyLines = drawRectGrid(surface1, sizeX, sizeY, spacingX, spacingY, numberU, numberV, polyLines)
a = polyLines
```

Example 5: Draw diagrid array on surface using parametric (0 – 1) values



```
#Place rectangle grid on surface input based upon reparameterization from 0 to 1
#Earl Mark, November 4, 2020
#input: surface
#output: diagrid

import rhinoscriptsyntax as rs
import math
import Rhino as rc


#helpful example
#http://www.grasshopper3d.com/profiles/blogs/python-snippet-reparameterizing-a-surface-
using-python

def getParamPt(surface1, x, y):
```

```
#get parametric point on surface
    surfPt = rc.Geometry.Surface.PointAt(surface1, x, y)
    return surfPt

def drawDiamond(surface1, x, y, sizeX, sizeY):
#draw rectangle on surface at parametric location (0 to 1) x, y of sizes sizeX, sizeY
    ptList = []
    pt1 = getParamPt(surface1,x, y - 0.5 * sizeY);
    ptList.append(pt1)
    pt2 = getParamPt(surface1,x + 0.5 * sizeX, y);
    ptList.append(pt2)
    pt3 = getParamPt(surface1,x, y + 0.5 * sizeY);
    ptList.append(pt3)
    pt4 = getParamPt(surface1,x - 0.5 * sizeX, y);
    ptList.append(pt4)
    ptList.append(pt1)
    pline = rs.AddPolyline(ptList)
    return pline

def drawDiaGrid(surface1, sizeX, sizeY, spacingX, spacingY, numberHoriz, numberVert,
polyLines):
    #draw rows and columns of a rectangular grid
    sizeX = sizeX - spacingX
    sizeY = sizeY - spacingY
    for i in range(0, int(numberVert), 1): #rows
        for ii in range(0, int(numberHoriz), 1): #columns
            x = ii * (sizeX + spacingX) + 0.5 * (sizeX + spacingX)
            y = i * (sizeY + spacingY) + 0.5 * (sizeY + spacingY)
            #x = ii * sizeX + 0.5 * sizeX + 0.5 * spacingX
            #y = i * sizeY + 0.5 * sizeY + 0.5 * spacingY
            pline = drawDiamond(surface1, x, y, sizeX, sizeY)
            polyLines.append(pline)
    return polyLines

#main

polyLines = []
#get size of surface in U and V direcrtions
sizeU = surface1.Domain(0)
sizeV = surface1.Domain(1)

#parameterize surface from 0 to 1
surface1.SetDomain(0, rc.Geometry.Interval(0, 1)) # reparameterize u at 0 to 1
surface1.SetDomain(1, rc.Geometry.Interval(0, 1)) # reparameterize v at 0 to 1

#determine preliminary size of rectangles needed to fill surface
sizeX = 1/numberU
sizeY = 1/numberV

#initiate spacing parameter as a fraction of sizeX and sizeY at maximum value of 90%
spacingX = sizeX * spacing * 0.90
spacingY = sizeY * spacing * 0.90

#draw rectangles on surface
polyLines = drawDiaGrid(surface1, sizeX, sizeY, spacingX, spacingY, numberU, numberV, polyLines)

a = polyLines
```