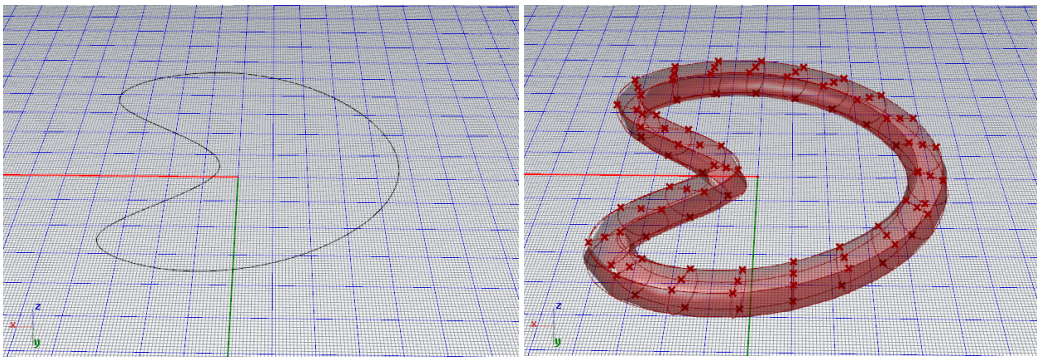
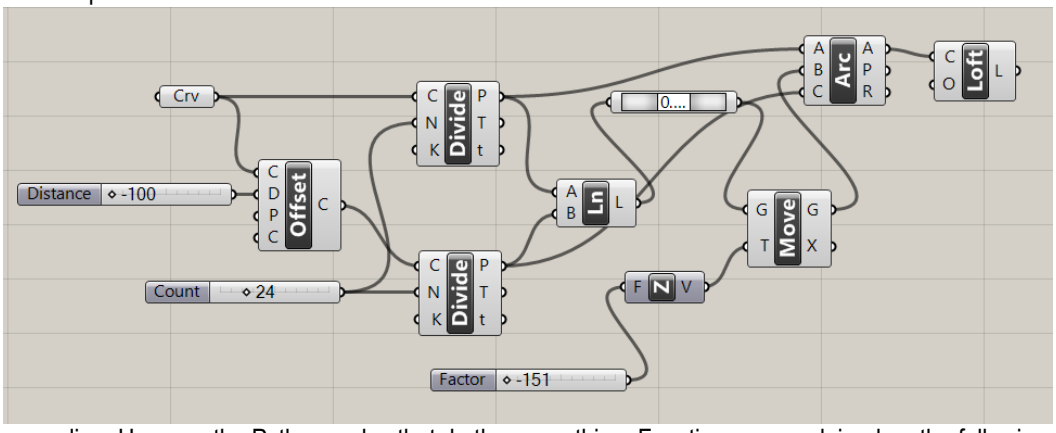
S206E057 -- Lecture 24, 5/27/2021, Interface between Python and Grasshopper

Final example for manipulating Rhino geometries through Python
The system unit is in "inch". Examples shown on pages 1,2 and 3 describe the methods. Final update of the method is given on pages 5-6.

Step 1: Draw an enclosed curve on the top view port as shown on the left. The image on the right is the final result of implementing the following GH components. Stepwise explanations are shown in the following.



Step 2: The GH components used to create a tube from the curve.



Step 3: Python coding: Here are the Python codes that do the same thing. Functions are explained on the following page.

```
import rhinoscriptsyntax as rs

crv01 = rs.GetObject("Select a curve", rs.filter.curve)
if rs.IsCurve(crv01):
   crv02 = rs.OffsetCurve(crv01,[0,0,0],-5.0) #Inward offset 5 units.

n = rs.GetInteger("No. of Division", 10,5,30)

p01 = [] #record of starting pts.
p02 = [] #record of ending pts.

pt01 = rs.DivideCurve(crv01,n)
pt02 = rs.DivideCurve(crv02,n)
for i in range(n):
   p01.append(rs.AddPoint(pt01[i]))
   p02.append(rs.AddPoint(pt02[i]))

ln = [] #record of lns.
for i in range(n): ln.append(rs.AddLine(pt01[i],pt02[i]))
```

```
pt03 = [] #record of midpts.
for i in range(n):
    t = rs.CurveDomain(ln[i])[1] / 2.0
    point = rs.EvaluateCurve(ln[i],t)
    pt03.append(rs.AddPoint(point))
    rs.MoveObject(pt03[i],[0,0,10])
crv = []
for i in range(n):
    crv.append(rs.AddArc3Pt(pt01[i],pt02[i],pt03[i]))

rs.AddLoftSrf(crv, closed=True)
```

Syntax functions (those functions shown in bold font above) applied from the Rhinoscriptsyntax library, and the required input (or arguments) in the Python program are explained in the following for reference. These pieces of information could be found from hitting the F1 function key to open the online manual help.

1. **Offsetcurve** function will offset a curve by a distance.

   Rhinoscriptsyntax.OffsetCurve (object_id, direction, distance, normal==None, style=1)
   - **object_id** is a required field, which is a string or guid of the object's identifier.
   - **direction** is also required, which is a list of 3 numbers. It usually is a point3d or vector3d that indicates the direction of the offset.
   - **distance** is also a required field and is a number indicating the offset distance.

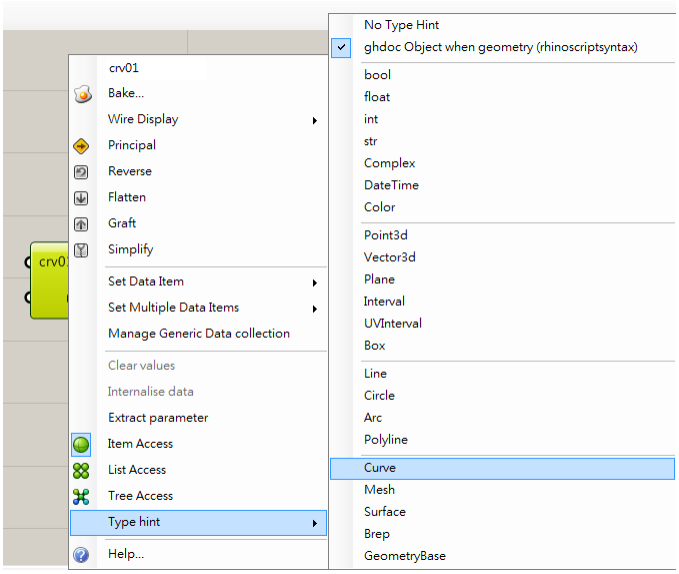
2. **IsCurve** is the function that verifies whether an object is a curve. If it is a curve, it will return **True**, otherwise **False.**
3. **DivideCurve** will divide a curve object into a specified number of segments. There are two required parameters of **curve_id** and the number of **segment**s.
4. **MoveObject** is to move a single object. It requires the identifier of the object to move, and a list of 3 numbers indicating the translation distance.

Step 4: Implementing Python through GH

Converting Python codes to Grasshopper components by copying Python codes to the **Python Script editor,** then save the contents. Python codes will be modified inside the **Python Script Editor**.

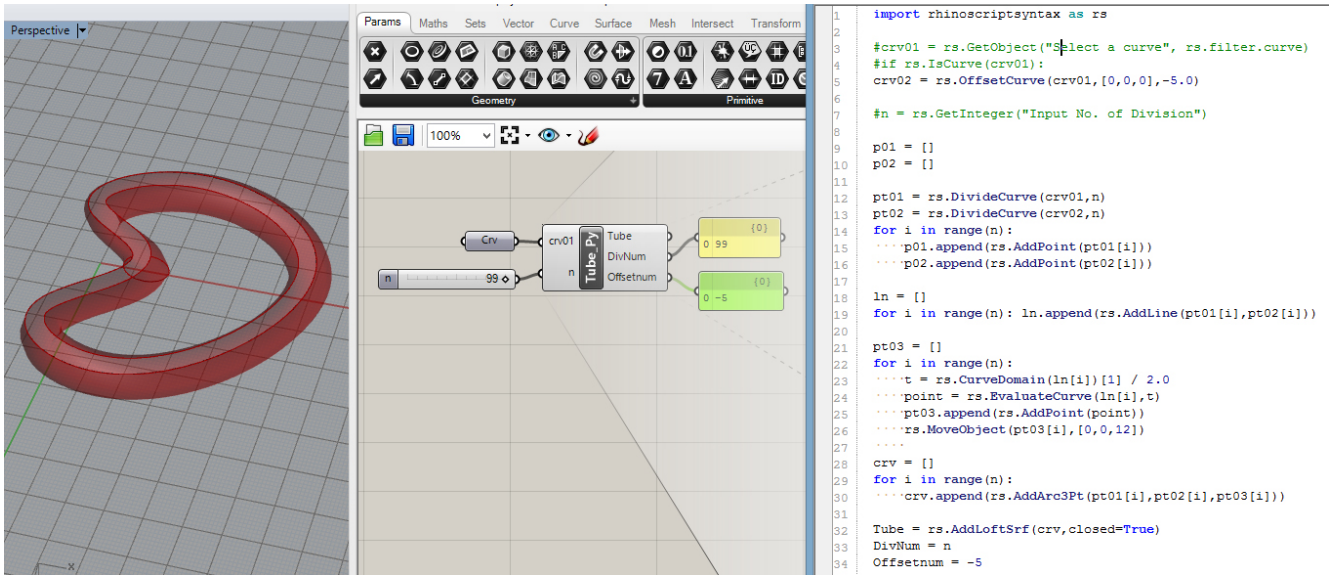
- **Modifying Python codes** on input part: the input of the curve and the definition of number of divided points will be executed by GH "**curve**" and "**number of slider**". Thus, turn lines #3, 4, and 7 in the Python coding file to comments.

- **Define input on GH** component: Two curve variables of crv01, crv02 represent the original and offset curves. The original curve is crv01, which is obtained by GH **curve** component input. Similarly, the second variable of the number of divisions should also be changed to n. Then, redefine the input variable names and their **variable type** by moving the mouse to the GH **crv01** input component and right click, **Type Hint** > select **Curve**, and **n** > **Type Hint** > as **Int** type.



- **Defining output** on GH: Add the following codes to the Python Script coding. Here, Tube is the variable name for the tube object. We must have the variable defined on the output area to make the geometry visible. DivNum is the number of division and Offsetnum is the offset distance, both will be displayed on GH panels.
  Tube = rs.**AddLoftSrf**(crv,closed=**True**)

See the results on the following images.

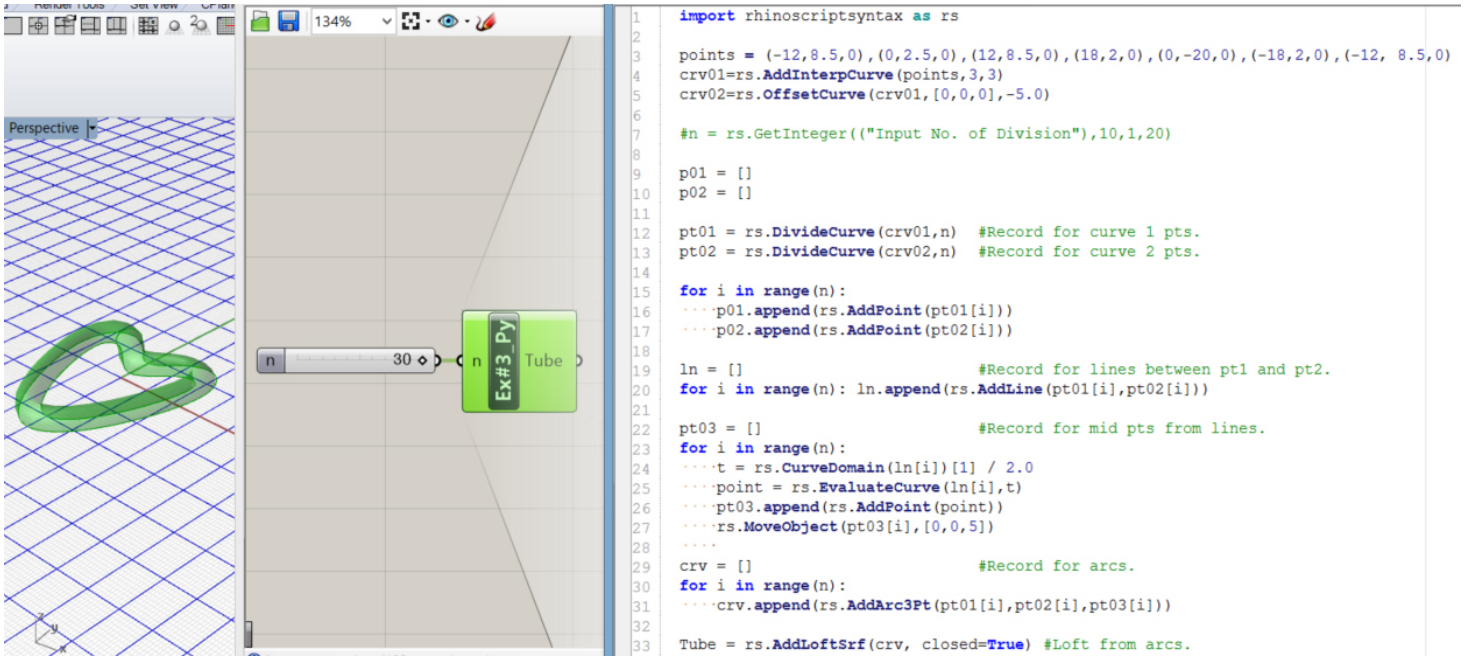


Example 2: Methods of generating a tube from a curve created by Python code.

This example will use the **AddInterpCurve** and **OffSetCurve** functions to generate two curves, then the number of divisions will be determined by the **GetInteger** function. A Rhino tube will be created by the Python coding inside the GH component and displayed graphically. When the Python coding was executed, it will run the AddLoftSrf directly and create a geometry in Rhino. But, when the codes are transferred to GH, the loft object must have a variable (here, it is called **Tube**) for output display. See the images below.

```
import rhinoscriptsyntax as rs

points = (-12,8.5,0),(0,2.5,0),(12,8.5,0),(18,2,0),(0,-20,0),(-18,2,0),(-12, 8.5,0)
crv01=rs.AddInterpCurve(points,3,3)
crv02=rs.OffsetCurve(crv01,[0,0,0],-5.0)

n = rs.GetInteger(("Input No. of Division"),10,1,20)

p01 = []
p02 = []

pt01 = rs.DivideCurve(crv01,n)  #Record for curve 1 pts.
pt02 = rs.DivideCurve(crv02,n)  #Record for curve 2 pts.

for i in range(n):
    p01.append(rs.AddPoint(pt01[i]))
    p02.append(rs.AddPoint(pt02[i]))

ln = []                    #Record for lines between pt1 and pt2.
for i in range(n): ln.append(rs.AddLine(pt01[i],pt02[i]))

pt03 = []                  #Record for mid pts from lines.
for i in range(n):
    t = rs.CurveDomain(ln[i])[1] / 2.0
    point = rs.EvaluateCurve(ln[i],t)
    pt03.append(rs.AddPoint(point))
    rs.MoveObject(pt03[i],[0,0,5])
```

```
crv = []                    #Record for arcs.
for i in range(n):
   crv.append(rs.AddArc3Pt(pt01[i],pt02[i],pt03[i]))

rs.AddLoftSrf(crv, closed=True) #Loft from arcs.
```




```
import rhinoscriptsyntax as rs

points = (-12,8.5,0),(0,2.5,0),(12,8.5,0),(18,2,0),(0,-20,0),(-18,2,0),(-12, 8.5,0)
crv01=rs.AddInterpCurve(points,3,3)
crv02=rs.OffsetCurve(crv01,[0,0,0],-5.0)

#n = rs.GetInteger(("Input No. of Division"),10,1,20)

p01 = []
p02 = []

pt01 = rs.DivideCurve(crv01,n)  #Record for curve 1 pts.
pt02 = rs.DivideCurve(crv02,n)  #Record for curve 2 pts.

for i in range(n):
    p01.append(rs.AddPoint(pt01[i]))
    p02.append(rs.AddPoint(pt02[i]))

ln = []                         #Record for lines between pt1 and pt2.
for i in range(n): ln.append(rs.AddLine(pt01[i],pt02[i]))

pt03 = []                       #Record for mid pts from lines.
for i in range(n):
    t = rs.CurveDomain(ln[i])[1] / 2.0
    point = rs.EvaluateCurve(ln[i],t)
    pt03.append(rs.AddPoint(point))
    rs.MoveObject(pt03[i],[0,0,5])
    
crv = []                        #Record for arcs.
for i in range(n):
    crv.append(rs.AddArc3Pt(pt01[i],pt02[i],pt03[i]))

Tube = rs.AddLoftSrf(crv, closed=True) #Loft from arcs.
```

Note: if the output has not been displayed on the screen, then right click the output node on the right of the name area and check if the preview on is clicked and whether it is defined as an object (it might be defined as text by the system) output format. Otherwise, erase the name and re-define.

Summaries: These are the methods and examples of turning Python codes into Grasshopper components. Pros and cons of using Python in GH are:

1. We could use Python to work out things to meet our goals and implement the codes in GH.
2. The data input in GH is more convenient than it is executed in Python, the same for output methods.
3. Personally, I prefer to use Python codes for executing tasks and apply GH (input and output) to see graphic results.

Applications:

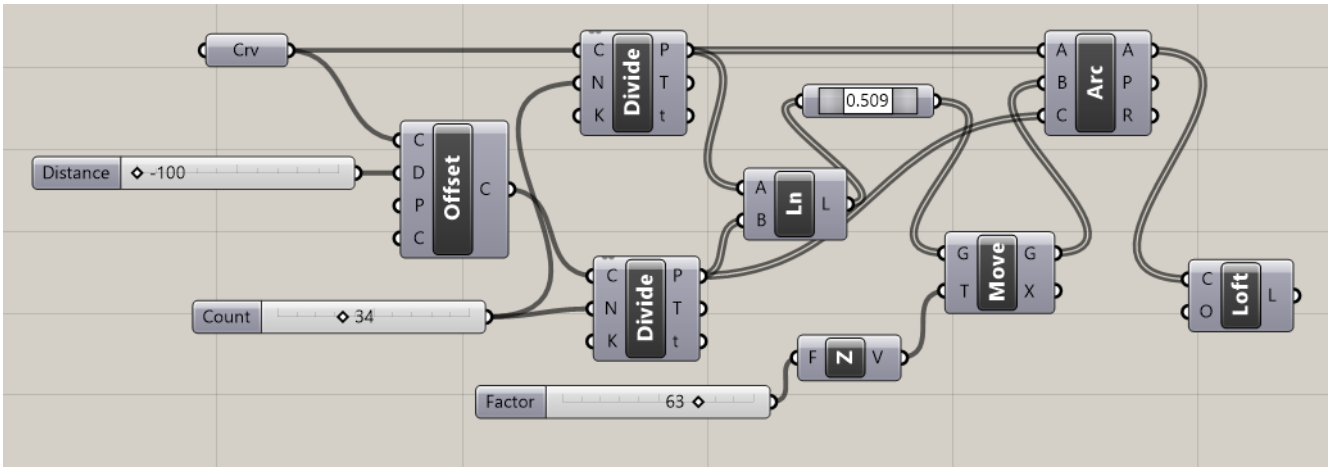
- It is critical to have your ideas generated on utilizing the Python codes for form generation.
- It also is an event on applying Python codes for form modification other than form creation.
- However, we could use both Python code and GH components to create major architectural form or details.

Please read the assignment handout carefully for final project submission on 6/6/2021.

Happy Grasshopper and Python coding….

Update version of curve to tube (2021-05-26)

GH codes:



Python codes:

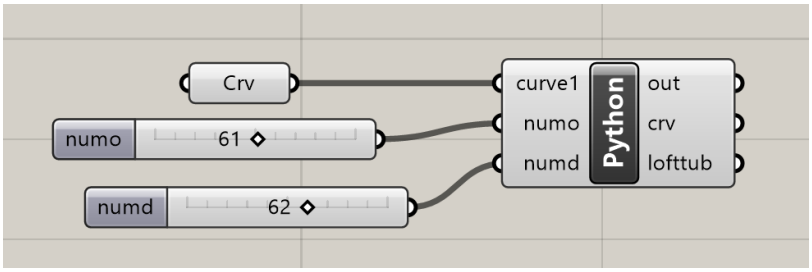
```
import rhinoscriptsyntax as rs

curve1 = rs.GetObject("Pick a Curve",rs.filter.curve)
numo =rs.GetInteger("Offset distance", 50,5,100)
numd = rs.GetInteger("Division number", 20,5,100)

curve2 = rs.OffsetCurve(curve1, [0,0,0], numo)

pt1 = rs.DivideCurve(curve1,numd)
pt2 = rs.DivideCurve(curve2,numd)

for i in range (numd):
    rs.AddPoint(pt1[i])
    rs.AddPoint(pt2[i])

ln=[]
for i in range (numd):
    ln.append(rs.AddLine(pt1[i],pt2[i]))

pt3=[]
for i in range (numd):
    t = rs.CurveDomain(ln[i])[1]/2
    pt = rs.EvaluateCurve(ln[i],t)
    pt3.append(rs.AddPoint(pt))
    rs.MoveObject(pt3[i],[0,0,10])

crv=[]
for i in range(numd):
    crv.append(rs.AddArc3Pt(pt1[i],pt2[i],pt3[i]))

rs.AddLoftSrf(crv, closed=True)
```

Python in GH codes:

```
import rhinoscriptsyntax as rs

curve2=rs.OffsetCurve(curve1,[0,0,0],numo)

pt1=rs.DivideCurve(curve1,numd)
pt2=rs.DivideCurve(curve2,numd)

ptlist1=[]
ptlist2=[]

for i in range (numd):
    ptlist1.append(rs.AddPoint(pt1[i]))
    ptlist2.append(rs.AddPoint(pt2[i]))

ln=[]
for i in range (numd):
    ln.append(rs.AddLine(pt1[i],pt2[i]))

pt3=[]
for i in range (numd):
    t = rs.CurveDomain(ln[i])[1]/1

    pt = rs.EvaluateCurve(ln[i],t)
    pt3.append(rs.AddPoint(pt))
    rs.MoveObject(pt3[i],[0,0,10])

crv=[]
for i in range(numd):
    crv.append(rs.AddArc3Pt(pt1[i],pt2[i],pt3[i]))

lofttub = rs.AddLoftSrf(crv, closed=True)
```

GHPython component:

Three inputs of curve1, numo, numd; and two outputs of crv, and lofttub are declared in the Python GH component.



Final implementation: