S206E057 -- Lecture 22, 5/27/2021, Python and Rhino interface

Rhino and Python interface: An example

How to run Rhino model by executing Python programs? And what kind of information is needed by Rhino to generate 3D forms? In this lecture, methods done by WHILE/FOR loops plus conditional cases are introduced first to serve as the beginning exercise.

First of all, use **import** command to allow Rhino script functions to be applied in Python. These Rhino script functions are the available libraries or a bunch of functions developed as classes. In the following example, import the library of **rhinoscriptsyntax** and named it as **rs**, which is in short of **rhinoscriptsyntax**. Information about the contents available in **rhinoscriptsyntax** could be found by hitting **F1** key (in PC machines) to bring up the programming reference manual.

Note: When we call syntax from the **rs** library, we always put "**rs.**" in front of the function name that follows. The dot operator after **rs** is used to link the functions developed in the class saved in the **rhinoscriptsyntax** library.

In the following coding example, we call syntaxes stored in "**rs**", use **AddPoint** to put a point at (0, 0, 0), put two spheres in Rhino through the Python coding. Results of execution would generate a point and two spheres in Rhino. Here, **AddPoint** requires coordination of 3 points. **AddSphere** needs a center point and the radius of the sphere. The required format of **AddPoint** are explained in the "**Rhino.Python Programmer's Reference**", which is available by hitting F1 function key. Here is the format.

rhinoscriptsyntax.AddPoint (point, y=None, z=None)

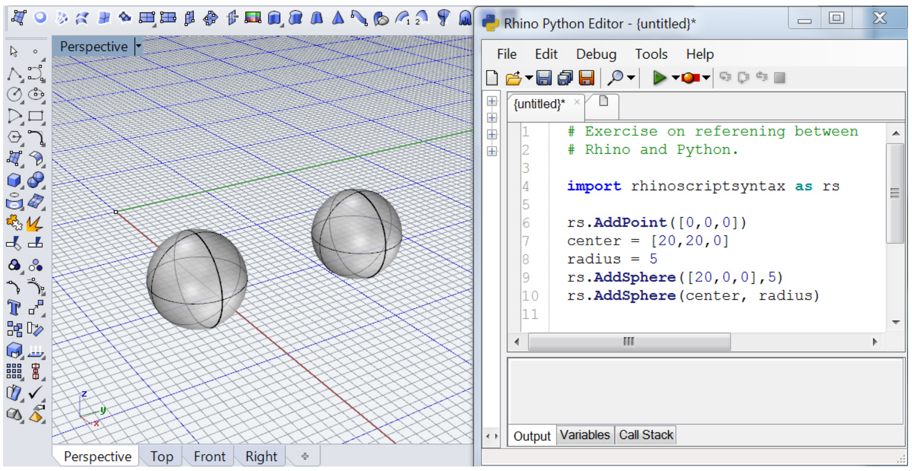
The first of the three parameters after the Addpoint is a required point, which is Point3d or a list of 3 values. The following x, y and z parameters represent the coordinates of x, y and z axis values. For instance, the point at x of 1, y of 2, and z of 3 will be called as:
rs.AddPoint((1,2,3)) or **rs.AddPoint([2,3,4]),** or **rs.AddPoint(20,10,10)** (The middle method is a safe play)

Format of AddShpere is:

```
rs.AddSphere(center_or_plane, radius)        #Center_or_Plane could be a list of 3 numbers, Vector3d, or Point3d.
```

Example #1: Exercise on referencing between Rhino & Python interface.

```
import rhinoscriptsyntax as rs

rs.AddPoint(0,0,0)
rs.AddPoint([10,10,0])
center = [10,10,0]
radius = 5
rs.AddSphere(center,radius)
rs.AddSphere([20,20,0],10)
rs.AddCircle(center,radius + 10)
```



```
# Exercise on referening between
# Rhino and Python.

import rhinoscriptsyntax as rs

rs.AddPoint([0,0,0])
center = [20,20,0]
radius = 5
rs.AddSphere([20,0,0],5)
rs.AddSphere(center, radius)
```

User input methods:

```
import rhinoscriptsyntax as rs

center = rs.GetPoint("Pick a point")
radius = rs.GetReal("Enter a number")
rs.AddSphere(center,radius)
rs.AddCircle(center,radius)
```

After codes were written, then click **run** to see if there is any error. If there is no error, then the point and geometries were created in Rhino, which could be edited by Rhino functions. In other words, Python codes in Rhino run through its own syntax window will generate 3D Rhino objects, which are different to the "**preview type**" of objects created by GH in Rhino.

Example #2: Applications of FOR and WHILE loops for object creations

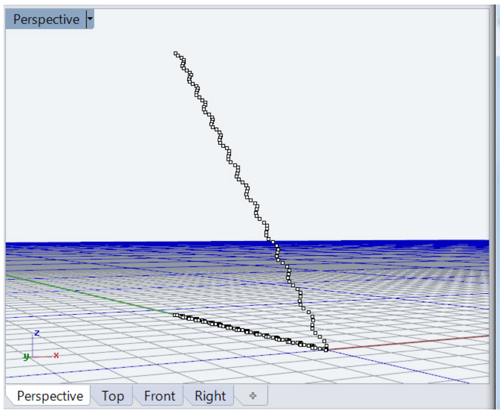
The second example is to use a WHILE and a FOR loop for creating two series of points and spheres. Again, in this example, the **rhinoscriptsyntax** is imported to the program and abbreviated as **rs**. Also, **math** is imported in the second group of coding below and named as **m** for the purposes of using mathematic operators of **sin** for generating a wave pattern.

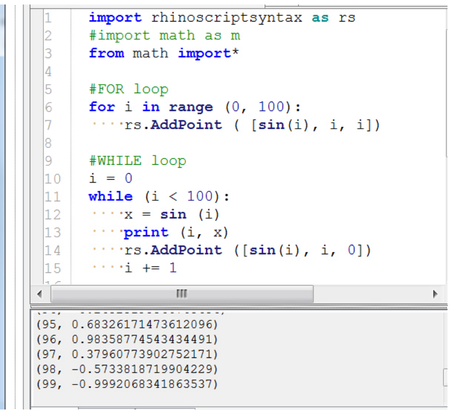
```
from rhinoscriptsyntax import*

center = rs.GetPoint("Pick a point")
radius = rs.GetReal("Enter a number")
rs.AddSphere(center,radius)
rs.AddCircle(center,radius)
```

Note: for importing functions from library, we could use either "**import** rhinoscriptsyntax **as** rs" or "**from** rhinoscriptsyntax **import*** to get all library contents in the library. If we use **from…import***, then we don't have to add the file name in front of the call-imported-function names. For instance, the two scripts on previous page and the one on left will get the same results. But, the left one will consume a large memory space which might freeze operations. It is fine to include math library by using the **from..import*** method in this example, but we shall use conventional way to import rhinoscriptsyntax for other scripts.

In the following example of **FOR** loop, the algorithm is to add points to Rhino. The coordinates of x were defined by sinθ, which is used to create the sine wave along x axis. The degree of θ ranged from 0 to 100, which is the angle represented by the variable **i** in this example. 100 is the stopping number, and it is where the calculation stops. These two examples explain the applications of math library in Python for Rhino 3D form generation. Here, we had not declared or defined any "**function**" or making any function calls yet. The program is executed one statement after the other. This style is similar to the style of scripting that executes codes line-by-line without making function-calls. Details of sin/cos/tan are at: https://www.mathsisfun.com/sine-cosine-tangent.html.

```
import rhinoscriptsyntax as rs
from math import*  #import math

# FOR loop
for i in range (0, 100):
    rs.AddPoint (sin(i), i, i)
# WHILE loop,
# Inside while, it prints the sine value.
x = 0
while (x < 100):
    print (sin(x))
    rs.AddPoint ([sin(x), x, 0])
    x += 1
```





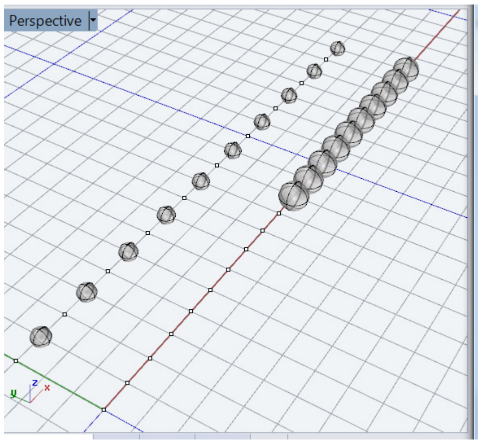
```
import rhinoscriptsyntax as rs
#import math as m
from math import*

#FOR loop
for i in range (0, 100):
    rs.AddPoint ( [sin(i), i, i])

#WHILE loop
i = 0
while (i < 100):
    x = sin (i)
    print (i, x)
    rs.AddPoint ([sin(i), i, 0])
    i += 1
```

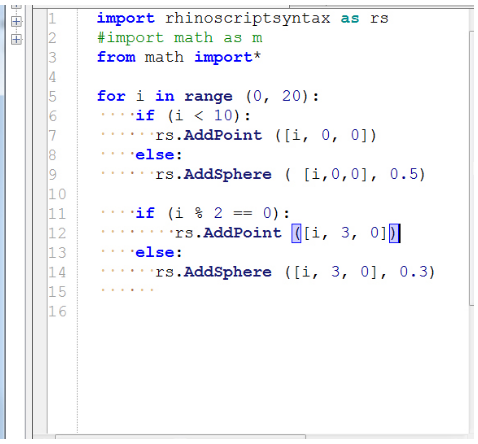
```
(95, 0.68326171473612096)
(96, 0.98358774543434491)
(97, 0.37960773902752171)
(98, -0.5733818719904229)
(99, -0.9992068341863537)
```

Example #3: IF-ELSE condition of Python in Rhino

In this third example, the FOR loop will be executed 20 times from 0 to 19 and stopped at 20. For each number in the FOR loop, it will test to see whether **i** is less than 10 or not and draw either a point or a sphere along the X axis. Then it will find even number to draw another point 3 units to the north and otherwise it draws a sphere with 0.3 radiuses. The way to find even number is by the % function (modulus), which divides two numbers and return only the remainder to indicate that it is an even number.

```
import rhinoscriptsyntax as rs
from math import*

for i in range (0, 20):
    if (i < 10):
        rs.AddPoint ([i, 0, 0])
    else:
        rs.AddSphere([i, 0, 0], 0.5)
    if (i % 2 == 0):
        rs.AddPoint ([i, 3, 0])
    else:
        rs.AddSphere([i, 3, 0], 0.3)
```

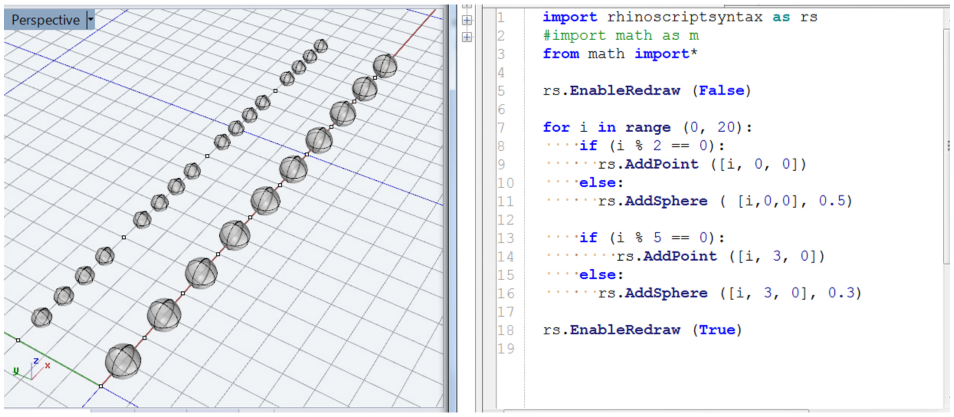




```
import rhinoscriptsyntax as rs
#import math as m
from math import*

for i in range (0, 20):
    if (i < 10):
      rs.AddPoint ([i, 0, 0])
    else:
      rs.AddSphere ( [i,0,0], 0.5)

    if (i % 2 == 0):
        rs.AddPoint ([i, 3, 0])
    else:
      rs.AddSphere ([i, 3, 0], 0.3)
```

Example #4: This example is to divide the number of **i** by five. When the remainder is 0, then draw a point, otherwise draw sphere. In this case, a variable of **rs.EnableRedraw** is used. This function lets Rhino draw final results at the end of executing codes, instead of drawing the screen one by one while executing each FOR loop to speed up drawing sequences.

```
import rhinoscriptsyntax as rs
from math import* #import math as m

rs.EnableRedraw (False)
for i in range (0, 20):
   if (i % 2 == 0):
     rs.AddPoint ([i, 0, 0])
   else:
     rs.AddSphere ( [i,0,0], 0.5)
   if (i % 5 == 0):
      rs.AddPoint ([i, 3, 0])
   else:
     rs.AddSphere ([i, 3, 0], 0.3)
rs.EnableRedraw (True)
```



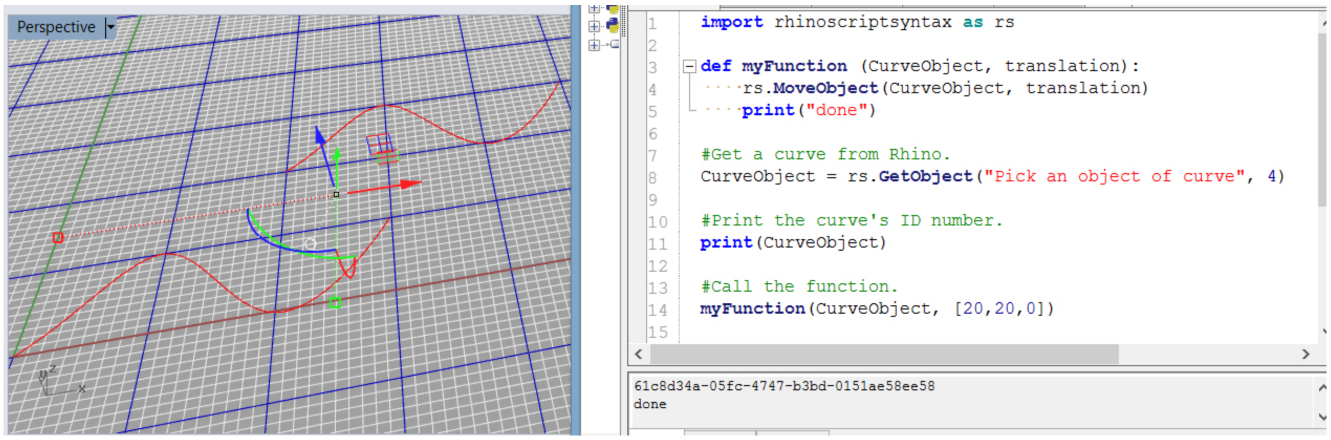
```
import rhinoscriptsyntax as rs
#import math as m
from math import*

rs.EnableRedraw (False)

for i in range (0, 20):
    if (i % 2 == 0):
      rs.AddPoint ([i, 0, 0])
    else:
      rs.AddSphere ( [i,0,0], 0.5)

    if (i % 5 == 0):
       rs.AddPoint ([i, 3, 0])
    else:
      rs.AddSphere ([i, 3, 0], 0.3)

rs.EnableRedraw (True)
```

Interaction between Python functions and Rhino model:

Python functions used in Rhino are defined the same way as they are defined in Python Interpreter. As long as the function is defined, it could be executed in Rhino for doing the things that could be done manually. The following is an example explaining the interaction between Rhino model and Python programming.

An interface function of **GetObject** is available to interact with Rhino models. It allows users to pick or select objects through the operation of filters, which determine the type of geometry to catch from the scene. For instance, **0** is set up for all objects, "**1**" is for points, "**4**" is for curves, and "**32**" is for meshes.

Thus, draw a curve or a circle on the original point in Rhino, run the program, and select the circle / curve, then print the Rhino's Object ID of the circle and move it along x and y of 20 units. This example has a function "**myFunction**" defined to call **myFunction(CurveObject, [20, 20, 0])**, the object of **CurbeObject** is the curve obtained through the **GetObject**.



```
import rhinoscriptsyntax as rs

def myFunction (CurveObject, translation):
    rs.MoveObject(CurveObject, translation)
    print("done")

#Get a curve from Rhino.
CurveObject = rs.GetObject("Pick an object of curve", 4)

#Print the curve's ID number.
print(CurveObject)

#Call the function.
myFunction(CurveObject, [20,20,0])
```

```
61c8d34a-05fc-4747-b3bd-0151ae58ee58
done
```

Here the MoveObject script function has the format of:

```
rs.MoveObject (object_id, translation)
```

The object ID is the system number created by Rhino right after the GetObject function is executed. The value of the amount of translation is the list value of **[20,20,0]**.

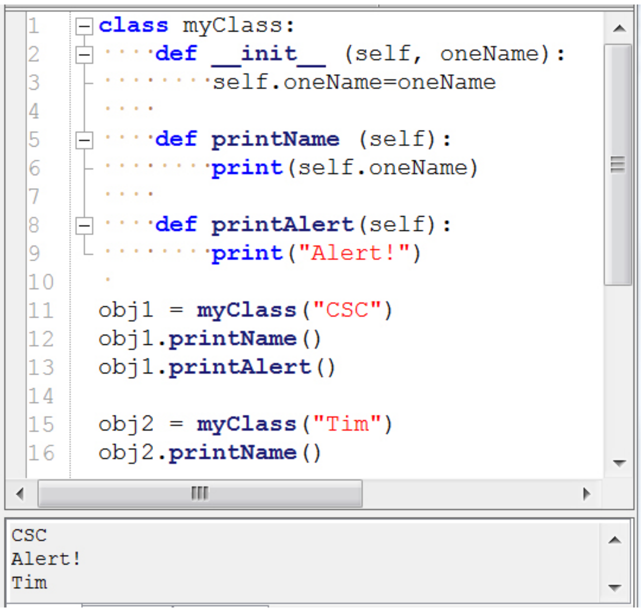
Example codes of the MoveObject provided in the reference manual will move objects to the exact location. The "end – start" is the calculation of the movement distance provided for your reference.

```
import rhinoscriptsyntax as rs

id = rs.GetObject("Select object to move")
if id:
  start = rs.GetPoint("Point to move from")
  if start:
    end = rs.GetPoint("Point to move to")
    if end:
      translation = end - start
      rs.MoveObject(id, translation)
      print(end - start)
```

Python Class in Rhino

This example is to understand how the Python concept of "**class**" could be implemented from in Rhino. It actually is the same as it runs in the Terminal mode outside the Rhino. The notion of "class", which is the nature of Object-Oriented Programming Language (OOPL), was further extended from notions of "module".

```
class myClass:
  def __init__ (self, oneName):
    self.oneName=oneName

  def printName (self):
    print(self.oneName)

  def printAlert(self):
    print("Alert!")

obj1 = myClass("CSC")
obj1.printName()
obj1.printAlert()

obj2 = myClass("Tim")
obj2.printName()
```

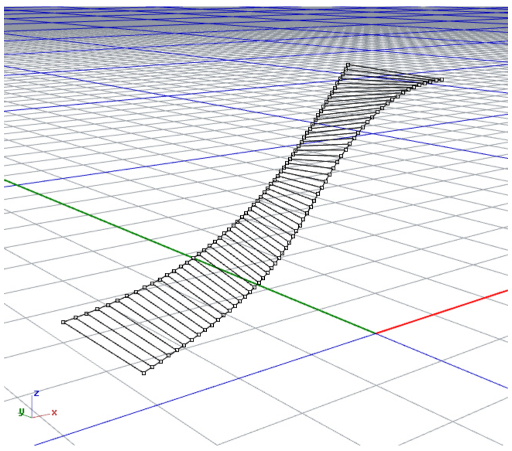


```
class myClass:
    def __init__ (self, oneName):
        self.oneName=oneName
    
    def printName (self):
        print(self.oneName)
    
    def printAlert(self):
        print("Alert!")
 
obj1 = myClass("CSC")
obj1.printName()
obj1.printAlert()

obj2 = myClass("Tim")
obj2.printName()
```

```
CSC
Alert!
Tim
```

A Python class code runs in Rhino

This example will define two objects (strCO1 and strCO2), and each object will have a line picked by user. The pick function is done by **GetObject** script function. Then the program will go through the class of "**LnBetCurves**" to divide the curves by a number of divisions into points and connect these points with straight lines. See the attached image. (Note: In this example, there is **DivideCurve** function applied, which takes four inputs.)

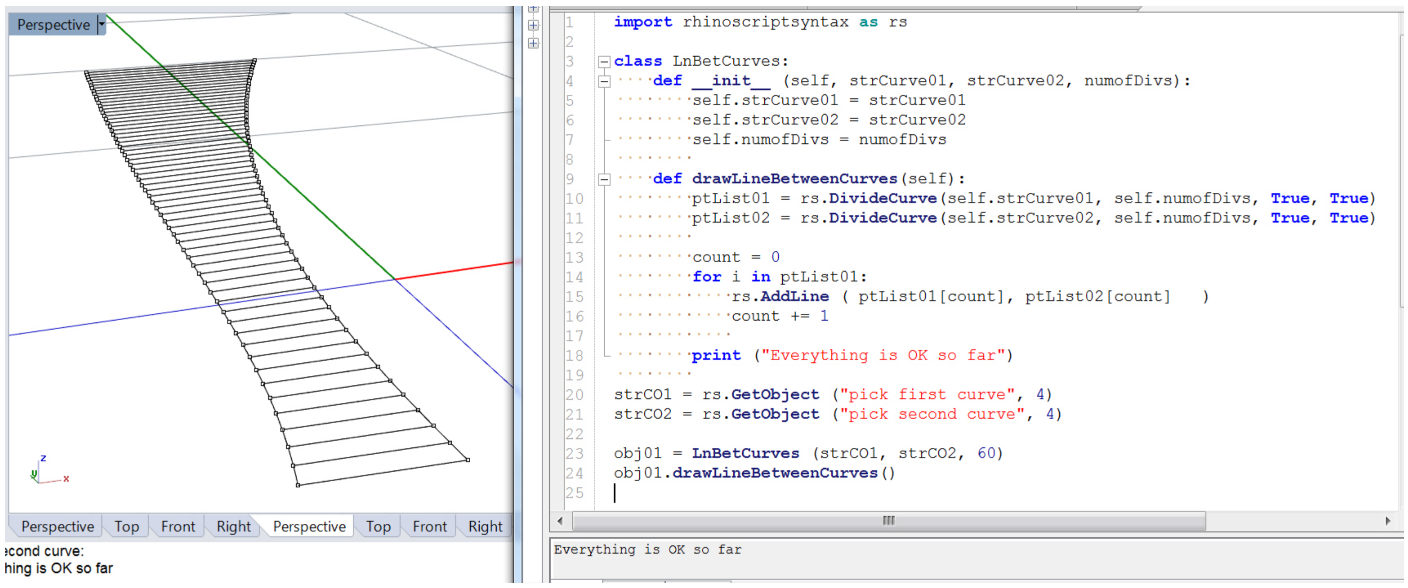


1. Curve-ID, which is the object's identifier.
2. The number of segments to be divided.
3. Create the division points. If omitted or False, a list of points are not created.
4. Return points. If omitted or True, an array containing division points are returned.

Note: In Python, there are input methods of: **raw_input** (get characters) and **input** (get numbers). In Rhinoscriptsyntax, there are around 35 functions for getting point, object, integers, or meshes from Rhino available – **GetPoint**, **GetObject**, **GetInteger**, etc. As long as we type **rs.**, available functions will show up for pick. Please check the reference manual (F1 function key).

Code example:

```
import rhinoscriptsyntax as rs

class LnBetCurves:
    def __init__ (self, strCurve01, strCurve02, numofDivs):
        self.strCurve01 = strCurve01
        self.strCurve02 = strCurve02
        self.numofDivs = numofDivs

    def drawLineBetweenCurves(self):
        ptList01 = rs.DivideCurve(self.strCurve01, self.numofDivs, True, True)
        ptList02 = rs.DivideCurve(self.strCurve02, self.numofDivs, True, True)
        count = 0
        for i in ptList01:
            rs.AddLine (ptList01[count], ptList02[count])
            count += 1
        print ("Everything is OK so far")

strCO1 = rs.GetObject ("pick first curve", 4)
strCO2 = rs.GetObject ("pick second curve", 4)

obj01 = LnBetCurves (strCO1, strCO2, 60)
obj01.drawLineBetweenCurves()
```



```
econd curve:
hing is OK so far
```

```
import rhinoscriptsyntax as rs

class LnBetCurves:
    def __init__ (self, strCurve01, strCurve02, numofDivs):
        self.strCurve01 = strCurve01
        self.strCurve02 = strCurve02
        self.numofDivs = numofDivs

    def drawLineBetweenCurves(self):
        ptList01 = rs.DivideCurve(self.strCurve01, self.numofDivs, True, True)
        ptList02 = rs.DivideCurve(self.strCurve02, self.numofDivs, True, True)

        count = 0
        for i in ptList01:
            rs.AddLine ( ptList01[count], ptList02[count]    )
            count += 1

        print ("Everything is OK so far")

strCO1 = rs.GetObject ("pick first curve", 4)
strCO2 = rs.GetObject ("pick second curve", 4)

obj01 = LnBetCurves (strCO1, strCO2, 60)
obj01.drawLineBetweenCurves()
```

```
Everything is OK so far
```

Call class outside of the class environment:

The Python class could be saved to a file and named as **LnBetCurvesClass.py**. The function of **drawLineBetweenCurves** inside the class file could be utilized by other files. Here is the class.

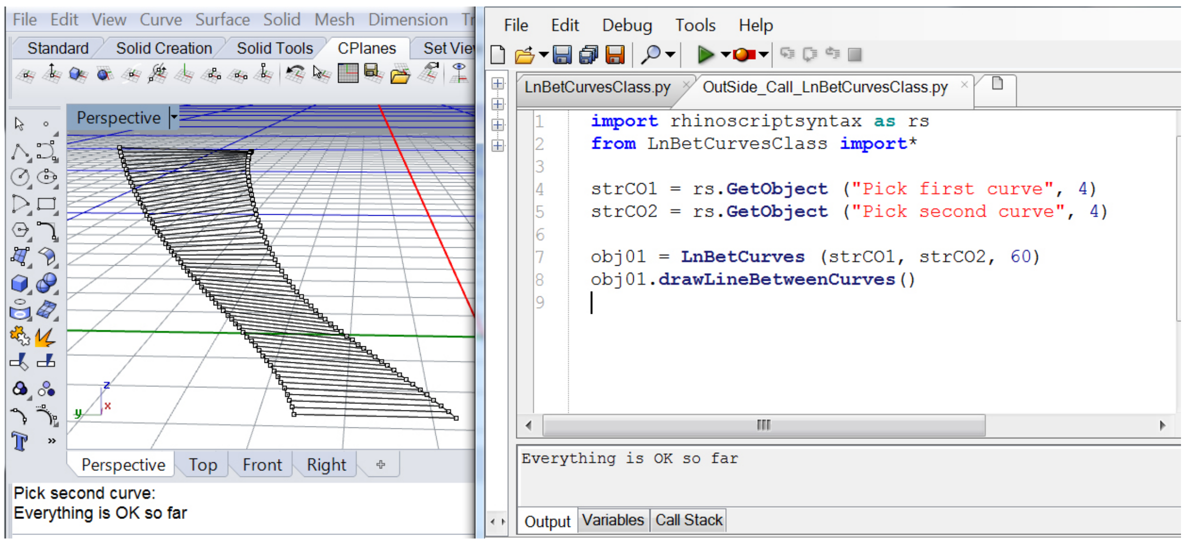
```
import rhinoscriptsyntax as rs

class LnBetCurves:
    def __init__ (self, strCurve01, strCurve02, numofDivs):
        self.strCurve01 = strCurve01
        self.strCurve02 = strCurve02
        self.numofDivs = numofDivs

    def drawLineBetweenCurves(self):
```

```
ptList01 = rs.DivideCurve(self.strCurve01, self.numofDivs, True, True)
ptList02 = rs.DivideCurve(self.strCurve02, self.numofDivs, True, True)
print(self.numofDivs)
count = 0
for i in ptList01:
    rs.AddLine (ptList01[count], ptList02[count])
    print(ptList01[count])
    count += 1
print ("Everything is OK, so far so good")
```

Here is the outside call example that imports the **LnBetCurvesClass.py** file and utilizes the **drawLineBetweenCurves** function.The **GetInteger** function has been used to get user input of the number of divisions, which has the following format:

rs.GetObject (A_Prompt_Message, Default_number, Minimum_number, Maximum_number)

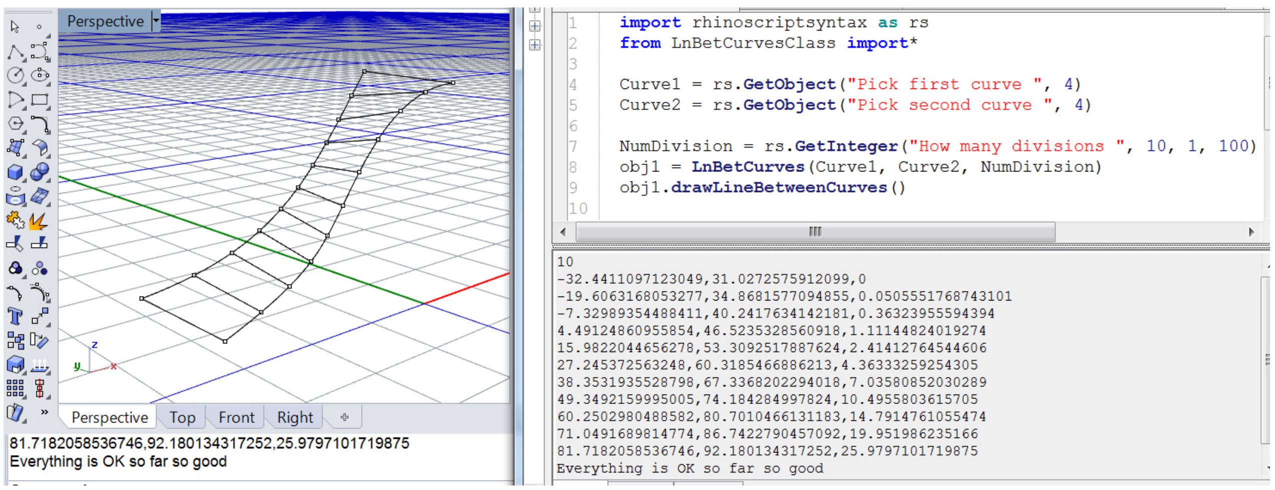
```
import rhinoscriptsyntax as rs
from LnBetCurvesClass import*

Curve1 = rs.GetObject("Pick first curve ", 4)                  #variable represents a curve.
Curve2 = rs.GetObject("Pick second curve ", 4)

NumDivision = rs.GetInteger("How many divisions ", 10, 1, 100)
obj1 = LnBetCurves(Curve1, Curve2, NumDivision)                #represents an object of executing the class
obj1.drawLineBetweenCurves()
```



```
import rhinoscriptsyntax as rs
from LnBetCurvesClass import*

strCO1 = rs.GetObject ("Pick first curve", 4)
strCO2 = rs.GetObject ("Pick second curve", 4)

obj01 = LnBetCurves (strCO1, strCO2, 60)
obj01.drawLineBetweenCurves()
```

```
Pick second curve:
Everything is OK so far
```

```
Everything is OK so far
```



```
import rhinoscriptsyntax as rs
from LnBetCurvesClass import*

Curve1 = rs.GetObject("Pick first curve ", 4)
Curve2 = rs.GetObject("Pick second curve ", 4)

NumDivision = rs.GetInteger("How many divisions ", 10, 1, 100)
obj1 = LnBetCurves(Curve1, Curve2, NumDivision)
obj1.drawLineBetweenCurves()
```

```
10
-32.4411097123049,31.0272575912099,0
-19.6063168053277,34.8681577094855,0.0505551768743101
-7.32989354488411,40.2417634142181,0.36323955594394
4.49124860955854,46.5235328560918,1.11144824019274
15.9822044656278,53.3092517887624,2.41412764544606
27.245372563248,60.3185466886213,4.3633259254305
38.3531935528798,67.3368202294018,7.03580852030289
49.3492159995005,74.184284997824,10.4955803615705
60.2502980488582,80.7010466131183,14.7914761055474
71.0491689814774,86.7422790457092,19.951986235166
81.7182058536746,92.180134317252,25.9797101719875
Everything is OK so far so good
```

```
81.7182058536746,92.180134317252,25.9797101719875
Everything is OK so far so good
```