MPDA20

IP Introduction to Python

Workshop 15.01.2020

IP workshop overview

0. Python RECAP

- variables
- casting
- operators
- functions
- conditionals
- collections
- iteration

1. From Geometry to Computation

- Basic knowledge of the library
- Generation of base geometries
  - i. Points
  - ii. Lines
  - iii. Polylines
  - iv. Curves
  - v. Planes
  - vi. Vectors
- Transformations

why learn to code (if we are designers)?

1. **Parametric design**

   Teaches you about geometry

   Enables you to design otherwise impossible things

2. **Automatization**

   Make repetitive tasks for big projects

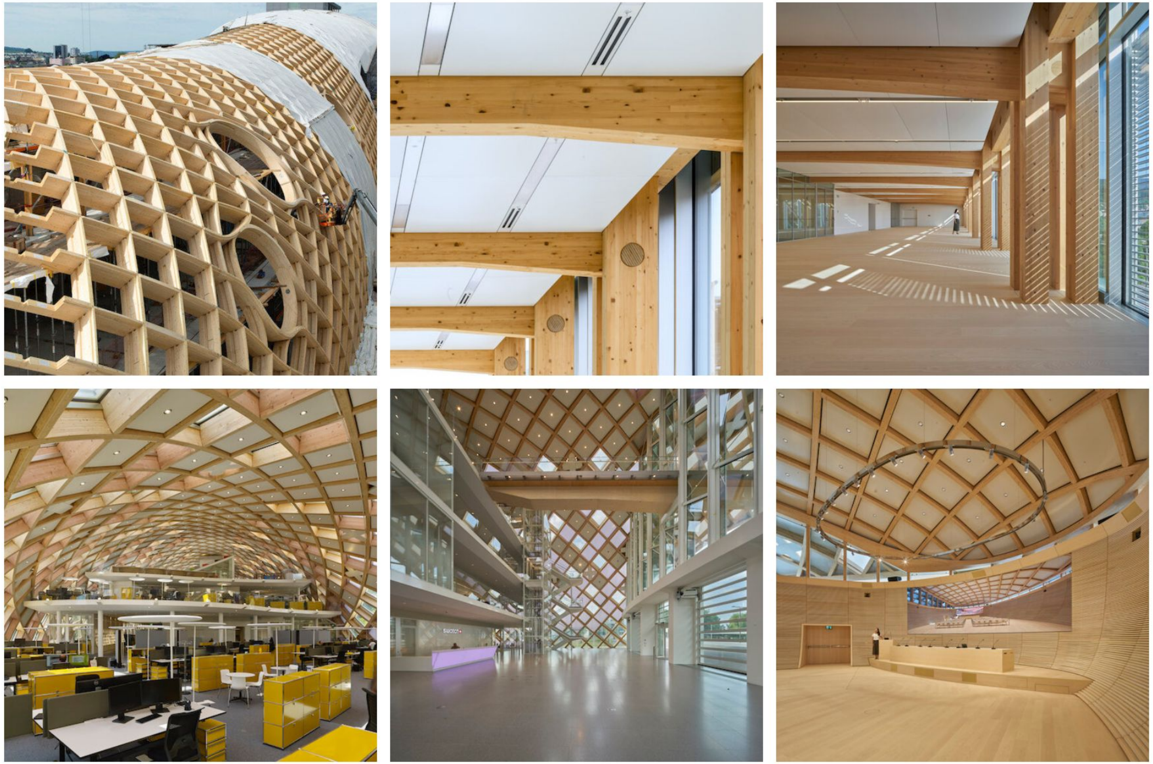
3. **Interoperability**

   Ride the BIM wave

   Revit, Blender, Unreal, etc...

These are very desirable skills in the AEC market nowadays.

why learn to code (if we are designers)?



Shigeru Ban / design to production

Swatch headquarters

Basel/Biel

why python?

Python is perhaps the most used programming language nowadays, and has an extended community and it is used by organizations such as **Google**, **NASA**, the **CIA**, and **Disney**.

what is python?

1. readability: closer to human language than machine language
2. portability: can be used in many machines.

Python is a high-level programming language, with applications in numerous areas, including web programming, scripting, scientific computing, and artificial intelligence.

flavours of python

Python is an interpreted language, so it has many implementations:

CPython : baseline implementation of python language

Python 2.7 : legacy

Python 3.X : current development

Jython : runs python for JVM (Java virtual machine)

IronPython : based on CPython 2.7

: runs python in Microsoft CLR (.NET framework) >>> **Rhino**

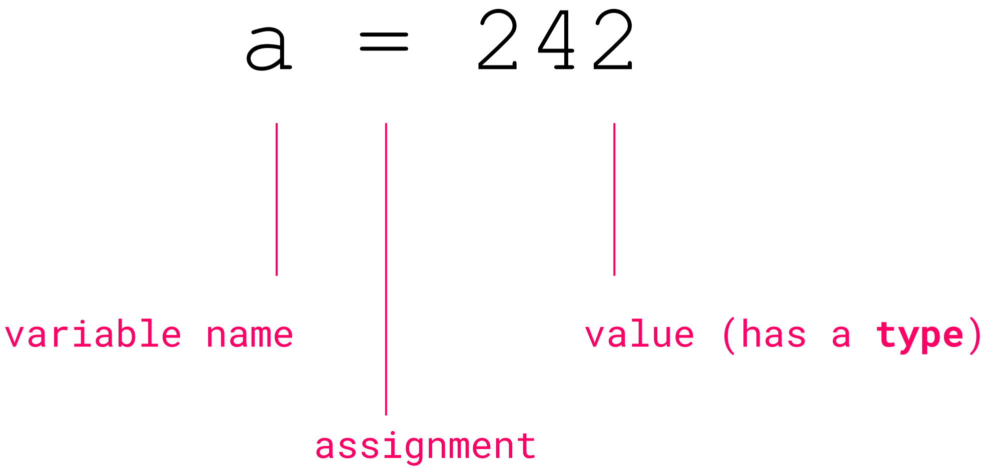
TrumpScript : make python great again!

other (cool) uses of python

1. Web Development: frontend and backend
2. Game Development: PyGame
3. Machine Learning and Artificial Intelligence: Tensorflow
4. Data Science and Visualization: NumPy & SciPy
5. Web Scraping: Scrappy & BeautifulSoup
6. Robotics: Linux/Raspi

python RECAP

variables



a = 242

variable name

assignment

value (has a **type**)

(forbidden) keywords

and
del
from
not
while
as
elif
global
or
with
assert
else
if
pass
yield
break
except
import
print
class
exec
in
raise
continue
finally
is
return
def
for
lambda
try

variable types

int	542	: an integer number
float	5.42	: a decimal number (so to speak)
string	“542”	: text (a list of characters)
bool	*True* or *False*	: to be or not to be
None	no value	: a placeholder

type casting

```
var = type(var)
```

```
a = 542             #assign 542 to a

print type(a)       >>> <type 'int'>

b = str(a)          #cast from int to str type

print type(b)       >>> <type 'str'>
```

operators

addition	+	20 + 45
subtraction	-	1 - 1
multiplication	*	hour*60+minute
division	/	minute/60
exponentiation	**	5**2
modulus	%	10%3
hierarchy	()	(5+9)*(15-7)

boolean operators

equality	==	x==y
inequality	!=	x!=y
greater than	>	x>y
less than	<	x<y
greater or equal	>=	x>=y
less or equal	<=	x<=y

logical operators

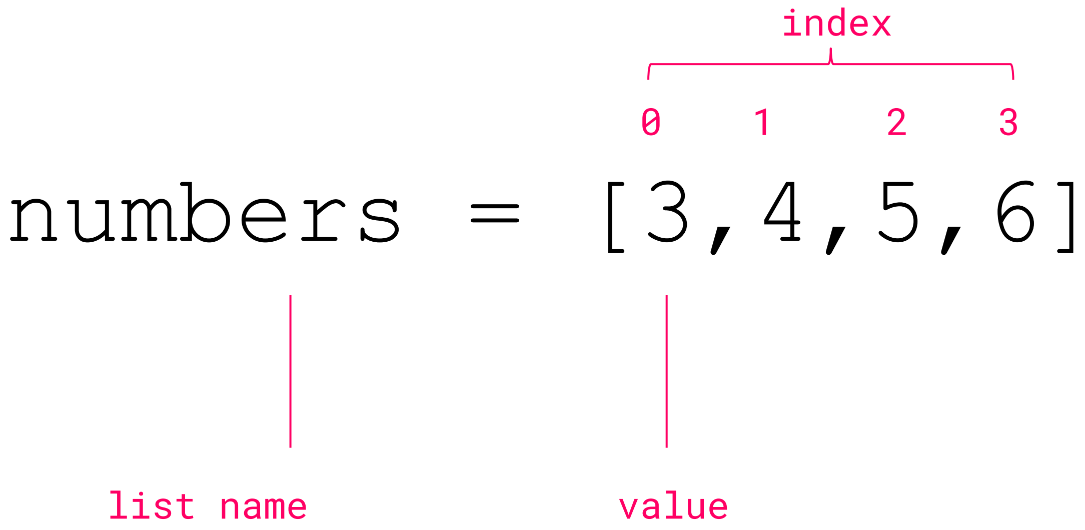
logical **AND**	`and`	`x and y`
logical **OR**	`or`	`x or y`
logical **NOT**	`not`	`not(x and y)`
identity	`is`	`x is y`
inclusion	`in`	`x in y`

conditionals

```
if(condition):
    do something
elif(condition):
    do another thing
else:
    do something else
```

```
if(x>0):
    print “x is a positive number”
elif(x<0):
    print “x is a negative number”
else:
    print “x is equal to zero”
```

lists



index

0 1 2 3

```
numbers = [3,4,5,6]
```

list name

value

list

NUMBERS

```
index 0 = 3
index 1 = 4
index 2 = 5
index 3 = 6
```

lists

```
list = []                     #initialize empty list

list = [5.2, 10, "dog"]       #initialize list with variables

list[2]                       #gets the item at index 2 >>> 'dog'

list[-2]                      #gets the item at index -2 >>> 10

list.append(5)                #add item 5 to the end list

list.insert(2, "fish")        #inserts "fish" at index 2

list.pop(0)                   #remove item 0 from the list

list.reverse()                #reverses the list
```

lists slices

```
list = [0, 1, 2, 3, 4]        #initialize list with 5 items
list2 = list[:]           #copy list into list2
print list[3:]                #gets items after index 3 >>>[3, 4]
print list[:1]                #gets items before index >>>[0, 1]
print list[2:4]           #gets items between i. 2 and 4 >>>[2, 3]
```

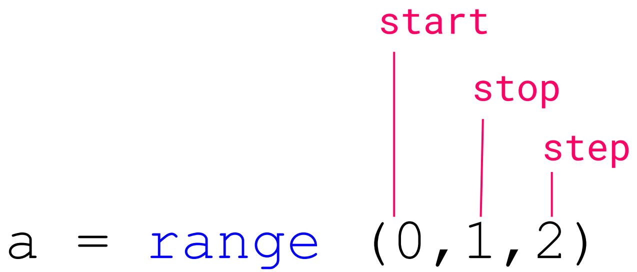
lists functions

cmp	Compares elements of both lists	**cmp**(list1, list2)
len	Returns the total length of the list	**len**(list)
max	Returns item from the list with max value	**max**(list)
min	Returns item from the list with min value	**min**(list)
seq	Converts a tuple into list	**list**(seq)

more list functions: https://www.tutorialspoint.com/python/list_list.htm

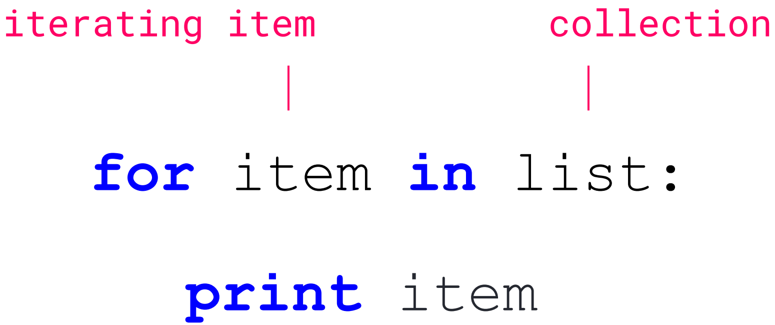
ranges

sequences. start from 0



range(5)	Creates a list of 10 consecutive elements	[0,1,2,3,4,]
range(2,7	Creates a list from 2 to 7 (not including 7)	[2,3,4,5,6]
range(0,10,2)	Creates a list of even numbers from 0 to 10	[0,2,3,4,8]
range(10,0,-2)	Creates a descending list of evens from 10-0	[10,8,6,4,2]

iteration: for loops



iterating item

collection

```
for item in list:
    print item
```

iteration: for loops

iterating variable

a list from 0 to 9:
[0,1,2,3,4,5,6,7,8,9]

```
for i in range(10):

    print i
```

prints:

```
0
1
2
3
4
5
6
7
8
9
```

iteration: for loops

```
aList = [0,1,2,3,4,5,6,7,8,9]

for i in aList:

    print i
```

prints:

```
0
1
2
3
4
5
6
7
8
9
```

iteration: for loops

```
aList = [a,b,c,d,e,f,g,h,i,j]

  for i in aList:

      print i
```

prints:

```
a
b
c
d
e
f
g
h
i
j
```

iteration: for loops

```
aList = [a,b,c,d,e,f,g,h,i,j]
   for i in range(len(aList)):
     print aList[i]
```

prints:

```
a
b
c
d
e
f
g
h
i
j
```

iteration: nested for loops

iterating variable

a list from 0 to 9:
[0,1,2,3,4,5,6,7,8,9]

```
for i in range(10):
    for j in range(10):
        print(i,j)
```

another list from 0 to 9:
[0,1,2,3,4,5,6,7,8,9]

iteration: nested for loops

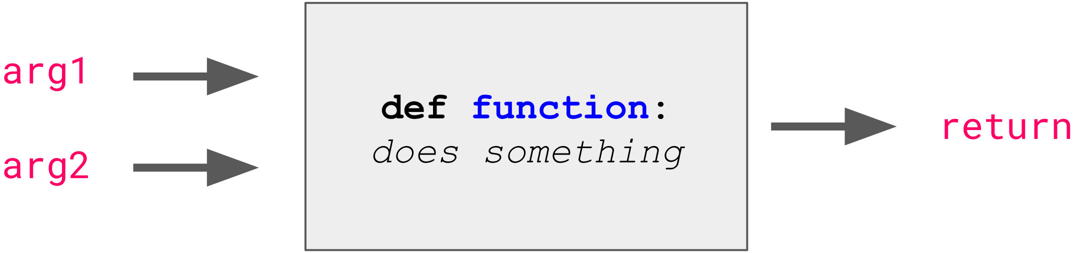
```
              i=0                     i=1                     i=2                     i=3                     i=4                     i=5                     i=6                     i=7                     i=8                     i=9
J= [ [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] , [0,1,2,3,4,5,6,7,8,9] ]
      0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9     0,1,2,3,4,5,6,7,8,9
```

iteration: nested for loops

i =	0	1	2	3	4	5	6	7	8	9
j =	0	0	0	0	0	0	0	0	0	0
j =	1	1	1	1	1	1	1	1	1	1
j =	2	2	2	2	2	2	2	2	2	2
j =	3	3	3	3	3	3	3	3	3	3
j =	4	4	4	4	4	4	4	4	4	4
j =	5	5	5	5	5	5	5	5	5	5
j =	6	6	6	6	6	6	6	6	6	6
j =	7	7	7	7	7	7	7	7	7	7
j =	8	8	8	8	8	8	8	8	8	8
j =	9	9	9	9	9	9	9	9	9	9

functions

a Grasshopper component, sort of...



functions

```
def function(argument1, argument2):          #non-fruitful

    print argument1, argument2


def function(argument1, argument2):          #fruitful: returns
something

    output = argument1 + argument2

    return output
```

modules

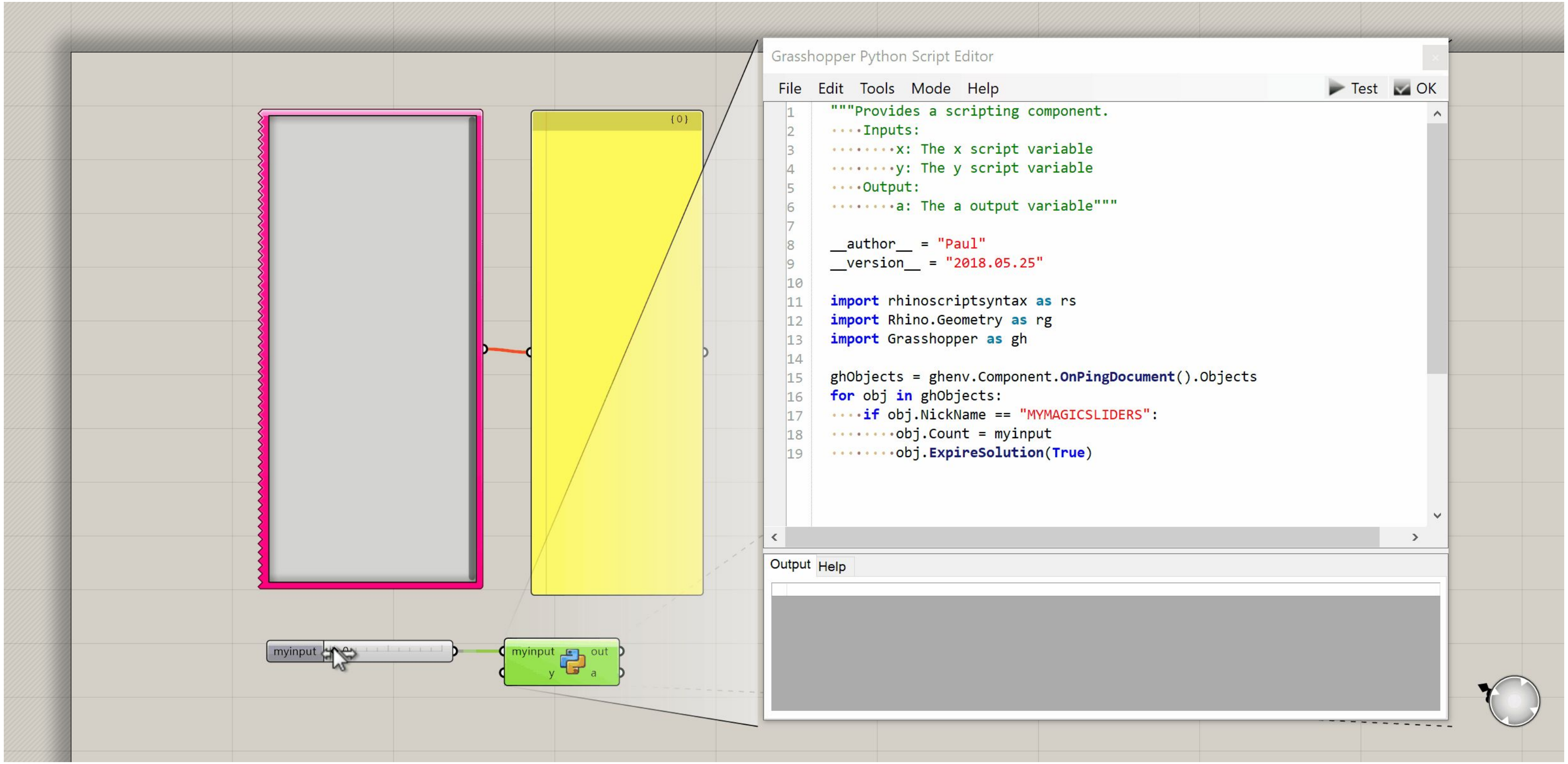
import	`import Rhino`
from...import	`from Rhino import Geometry`
from...import*	`from Rhino import *`
reload()	`reload(Rhino)`
globals()	`globals()`
locals()	`locals()`

basic modules: math and random

import math	math.sin()	:sine of an angle
	math.cos()	:cosine of an angle
	math.pi()	:constant of pi
import random	random.uniform()	:random float between 0 and 1
	random.randint(0,1)	:random int. between 0 and 10

ghpython

ghpython



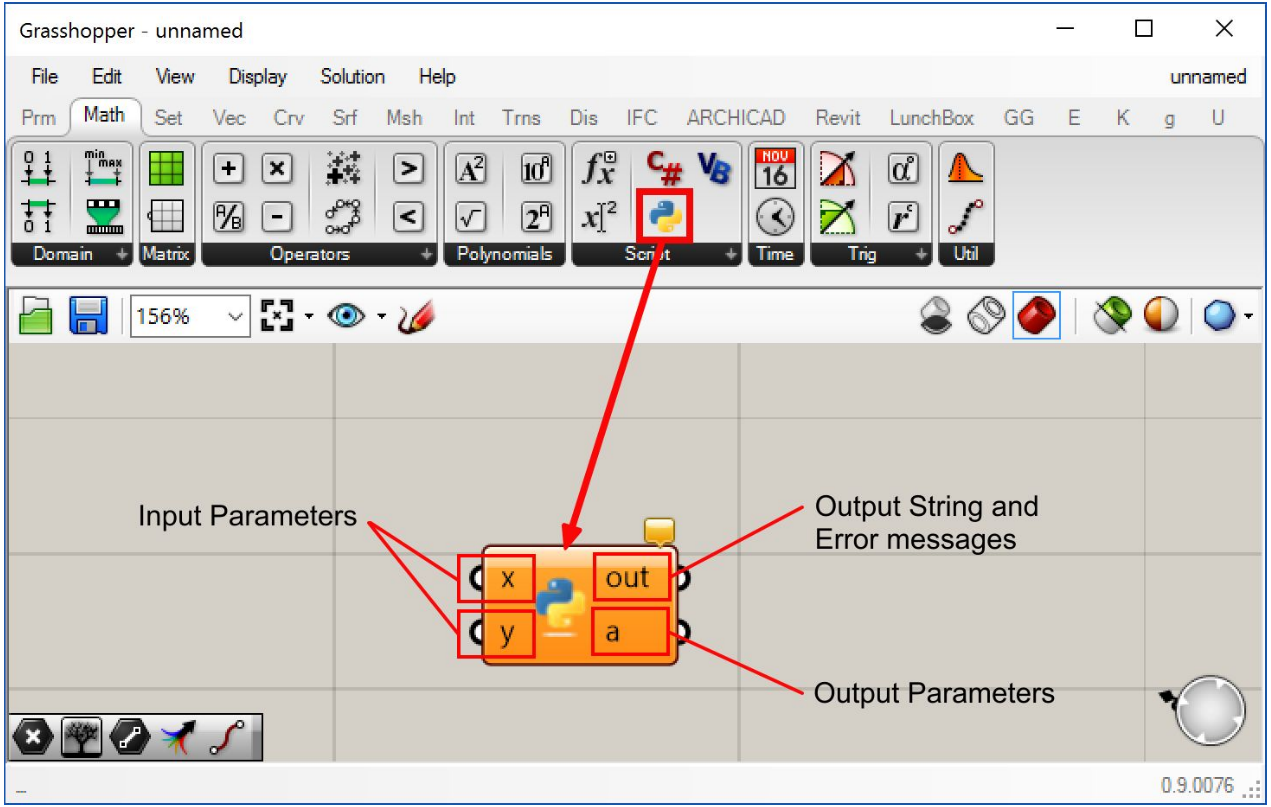
```
"""Provides a scripting component.
    Inputs:
        x: The x script variable
        y: The y script variable
    Output:
        a: The a output variable"""

__author__ = "Paul"
__version__ = "2018.05.25"

import rhinoscriptsyntax as rs
import Rhino.Geometry as rg
import Grasshopper as gh

ghObjects = ghenv.Component.OnPingDocument().Objects
for obj in ghObjects:
    if obj.NickName == "MYMAGICSLIDERS":
        obj.Count = myinput
        obj.ExpireSolution(True)
```

anatomy of python component



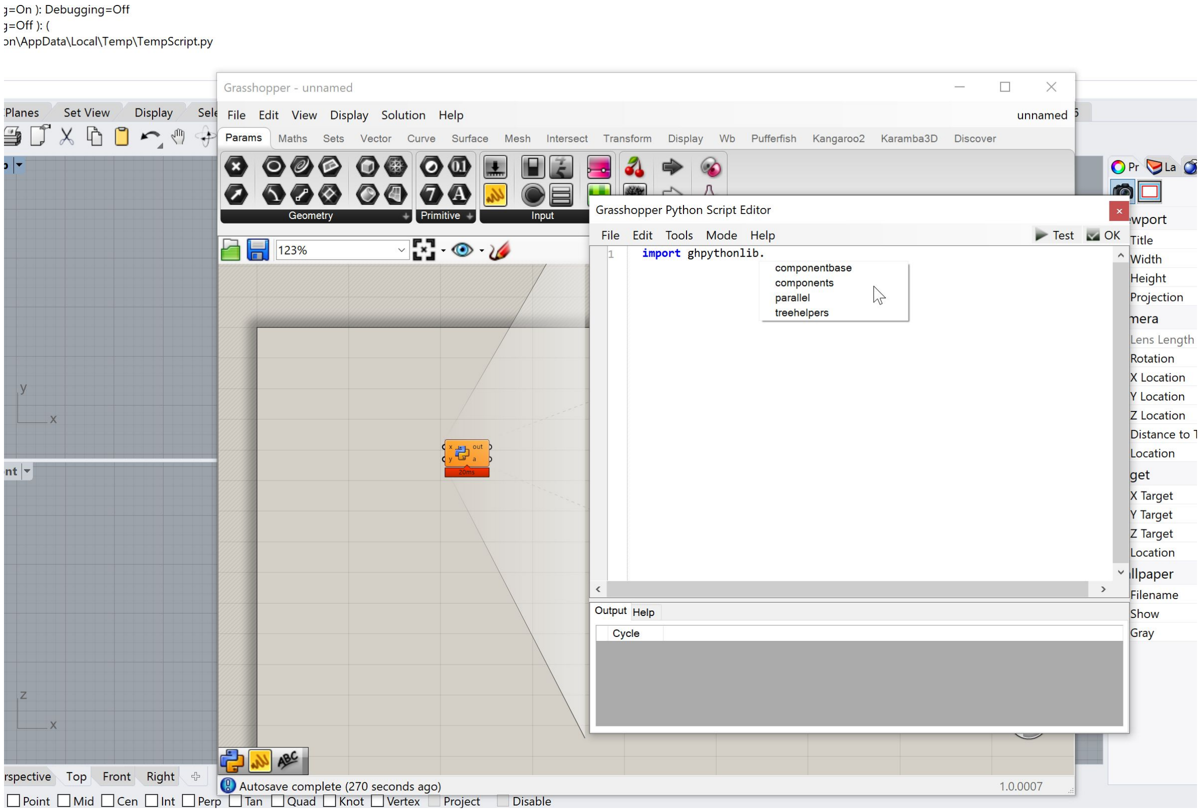
Input Parameters

Output String and Error messages

Output Parameters

ghpythonlib

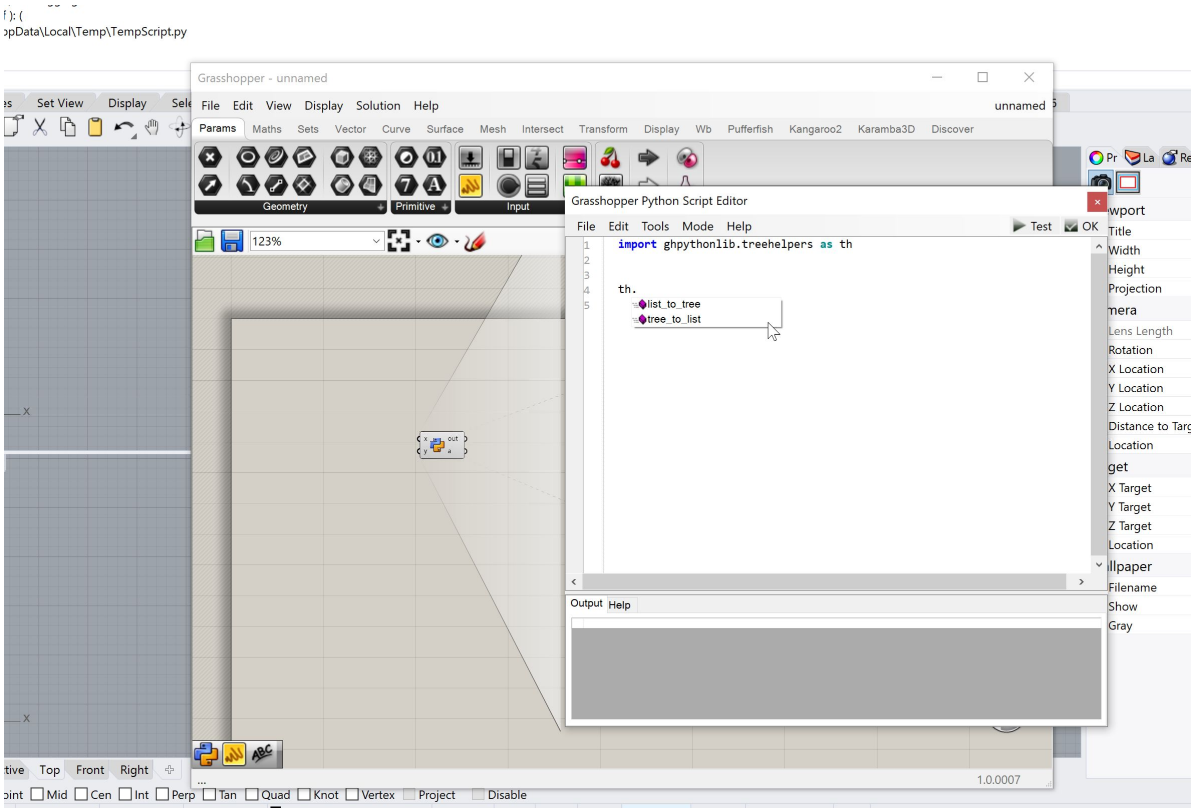
ghpythonlib.components



```
ı=On ): Debugging=Off
ı=Off ): (
ɔn\AppData\Local\Temp\TempScript.py
```

```
import ghpythonlib.
```

ghpythonlib.treehelpers

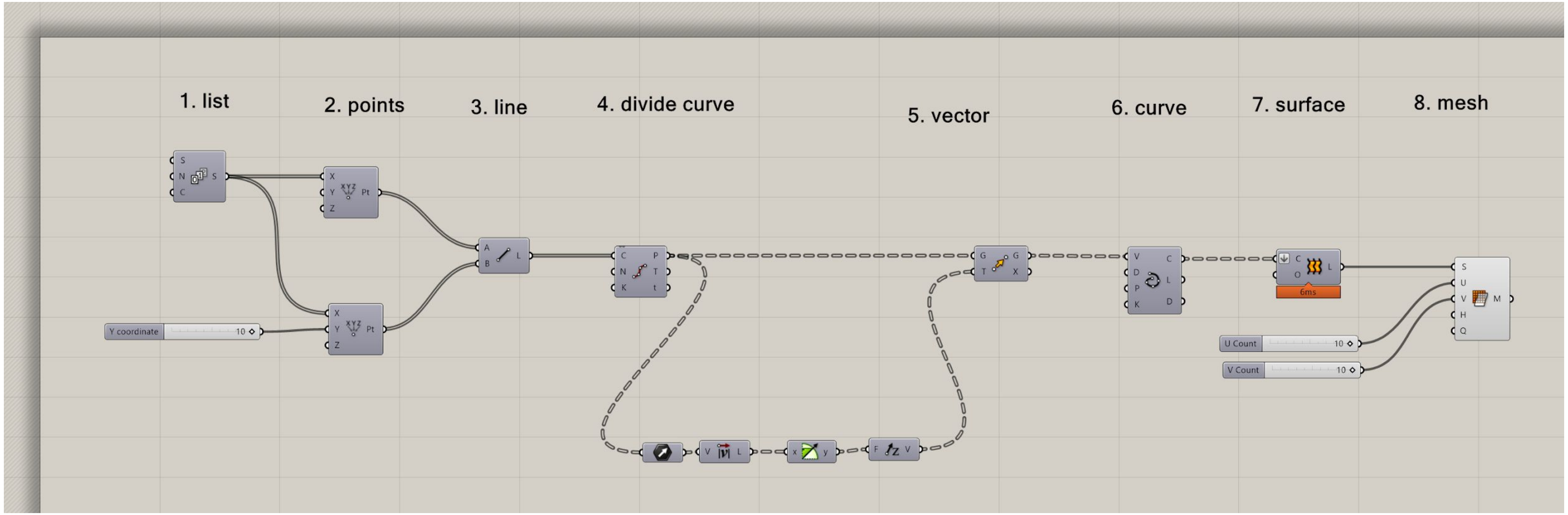


```
import ghpythonlib.treehelpers as th


th.
```

exercise 01:

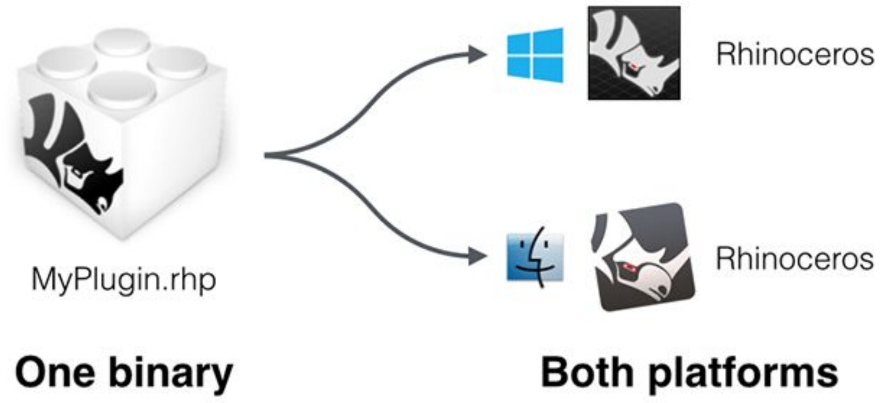
from grasshopper to python



https://github.com/dadandroid/MPDA20/

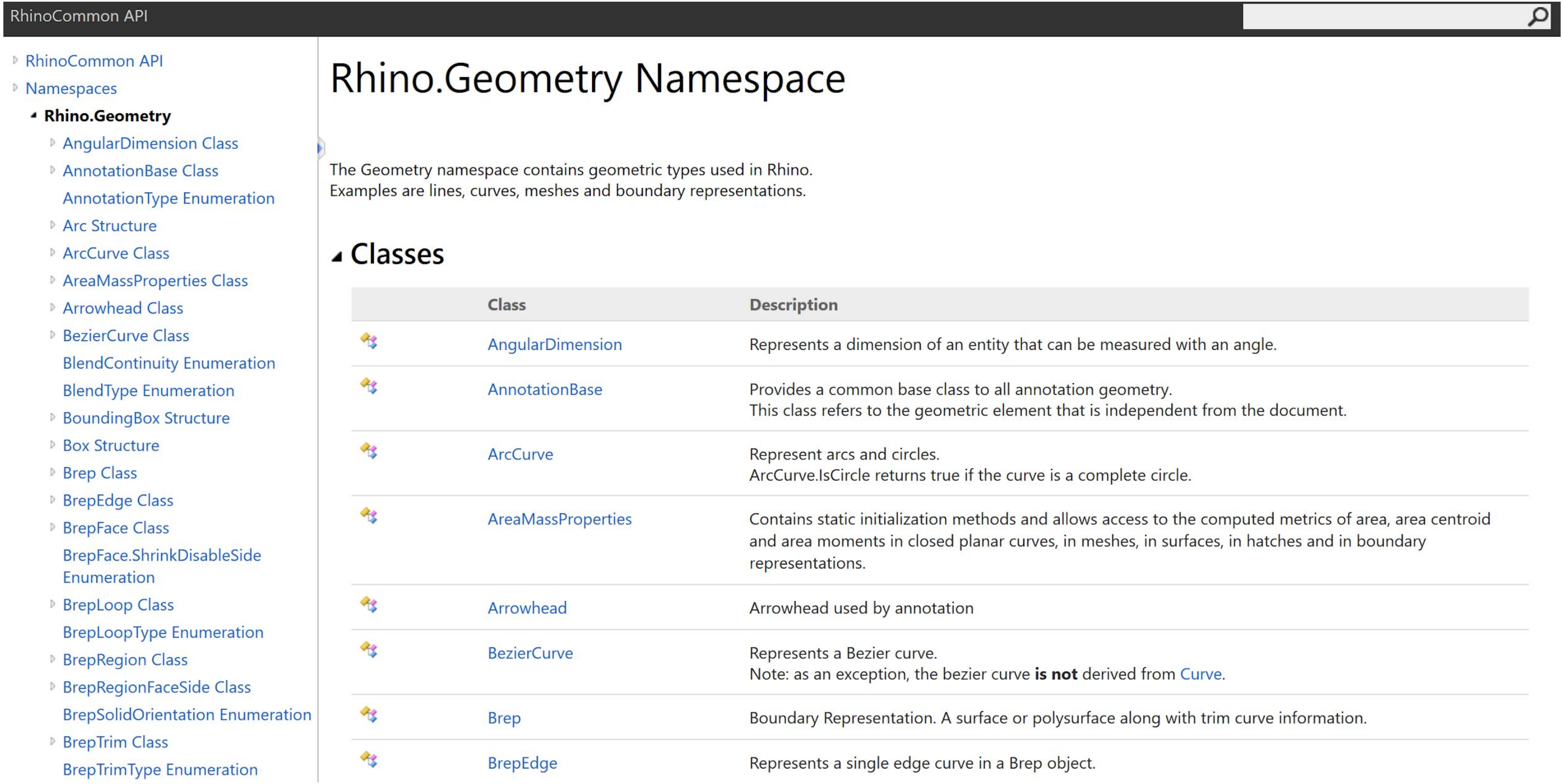
from Geometry to **Computation**

what is rhinocommon?



RhinoCommon is the SDK for Rhino,Grasshopper, python... built atop the portions of the .NET framework that are common on both Windows and macOS.

RhinoCommon allows developers to run .NET code on both Rhino for Windows and Rhino for Mac.



RhinoCommon API

- RhinoCommon API
- Namespaces
  - **Rhino.Geometry**
    - AngularDimension Class
    - AnnotationBase Class
    - AnnotationType Enumeration
    - Arc Structure
    - ArcCurve Class
    - AreaMassProperties Class
    - Arrowhead Class
    - BezierCurve Class
    - BlendContinuity Enumeration
    - BlendType Enumeration
    - BoundingBox Structure
    - Box Structure
    - Brep Class
    - BrepEdge Class
    - BrepFace Class
    - BrepFace.ShrinkDisableSide Enumeration
    - BrepLoop Class
    - BrepLoopType Enumeration
    - BrepRegion Class
    - BrepRegionFaceSide Class
    - BrepSolidOrientation Enumeration
    - BrepTrim Class
    - BrepTrimType Enumeration

Rhino.Geometry Namespace

The Geometry namespace contains geometric types used in Rhino.
Examples are lines, curves, meshes and boundary representations.

Classes

	Class	Description
	AngularDimension	Represents a dimension of an entity that can be measured with an angle.
	AnnotationBase	Provides a common base class to all annotation geometry. This class refers to the geometric element that is independent from the document.
	ArcCurve	Represent arcs and circles. ArcCurve.IsCircle returns true if the curve is a complete circle.
	AreaMassProperties	Contains static initialization methods and allows access to the computed metrics of area, area centroid and area moments in closed planar curves, in meshes, in surfaces, in hatches and in boundary representations.
	Arrowhead	Arrowhead used by annotation
	BezierCurve	Represents a Bezier curve. Note: as an exception, the bezier curve **is not** derived from Curve.
	Brep	Boundary Representation. A surface or polysurface along with trim curve information.
	BrepEdge	Represents a single edge curve in a Brep object.

rhinocommon vs rhinoscriptsyntax

- Rhinoscriptsyntax methods call into rhinocommon.
- For complex calculations, Rhinocommon can be faster than Rhinoscriptsyntax, and its functionality is more extended.
- Rhinoscriptsyntax requires less typing, better for interacting with Rhino

mix n´ match approach!

rhino.geometry

```
import Rhino.Geometry as rg
```

rg.Point3d

rg.Line

rg.Polyline

rg.Circle

rg.Plane

rg.Vector

rg.Curve

rg.Surface

rg.Brep

rg.Mesh

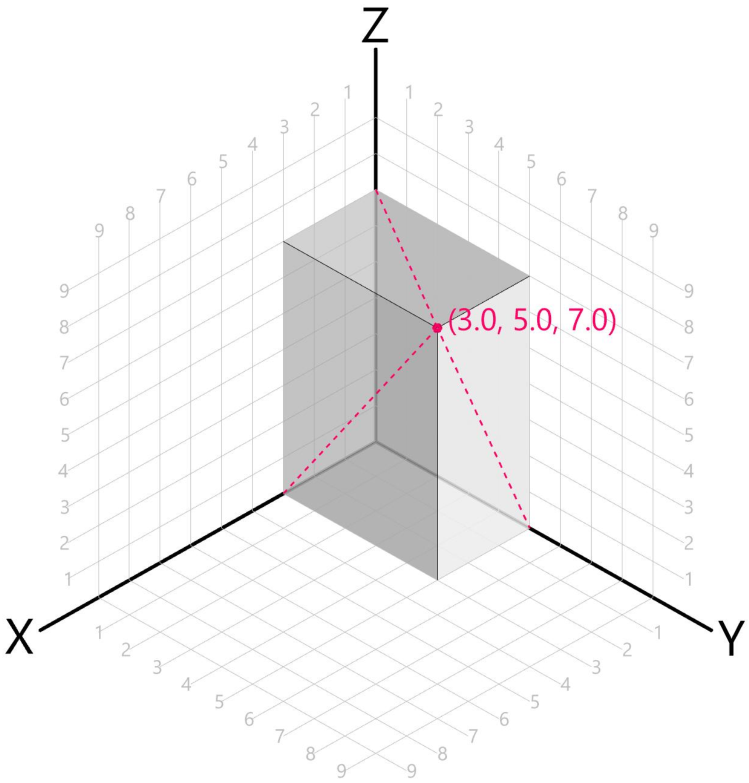
transformations

reference

http://developer.rhino3d.com/api/RhinoCommon/

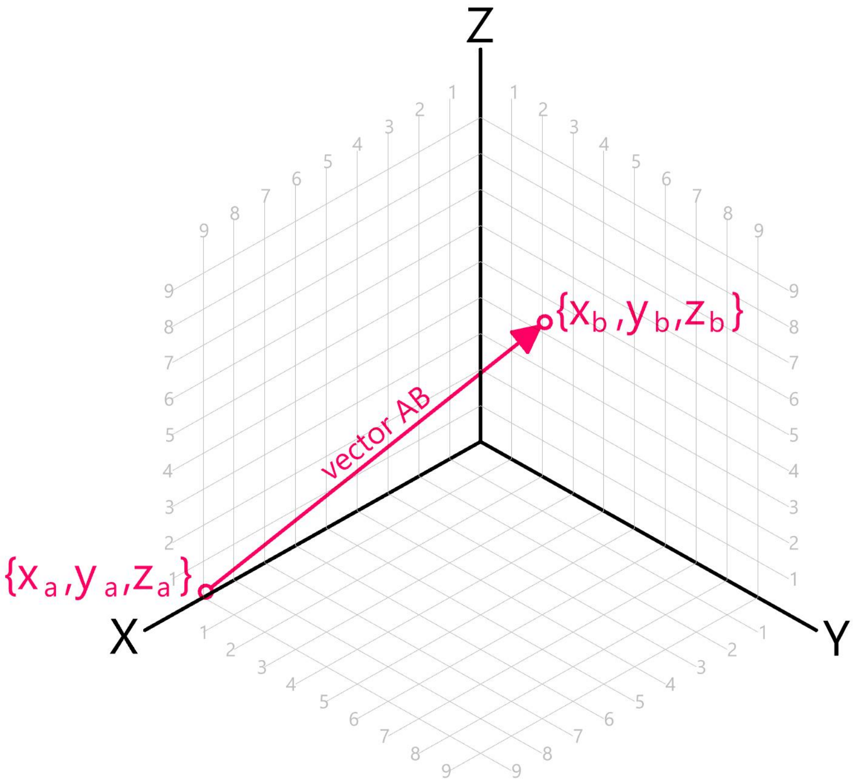
rg.Point3d

```
rg.Point3d(float X, float Y, float Z)
```



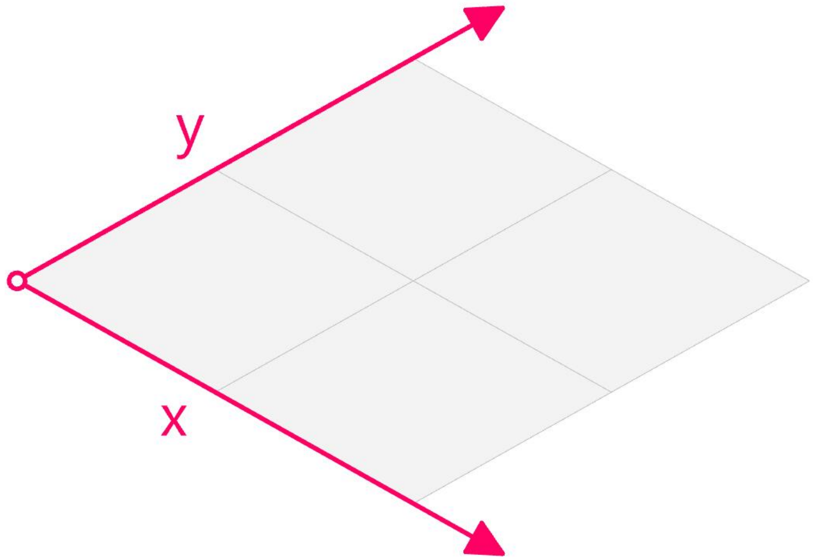
rg.Vector3d

`rg.Vector3d(float **X**, float **Y**, float **Z**)`

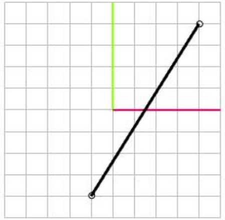


rg.Planes

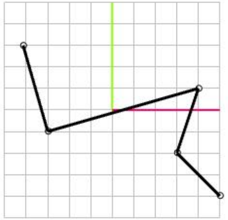
`rg.Plane(`**`Origin`**` Point3d, `**`xDirection`**` Vector3d, `**`yDirection`**` Vector3d)`



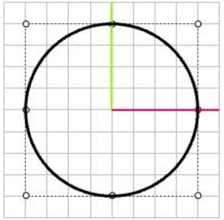
Curve types



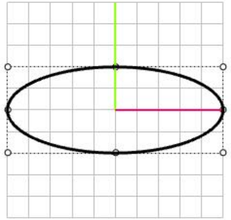
Line



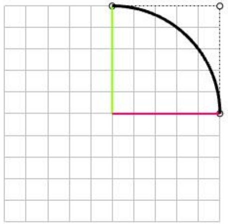
Polyline



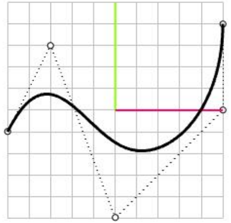
Circle



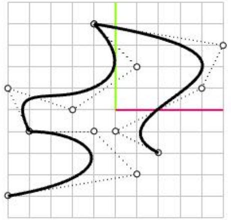
Ellipse



Arc



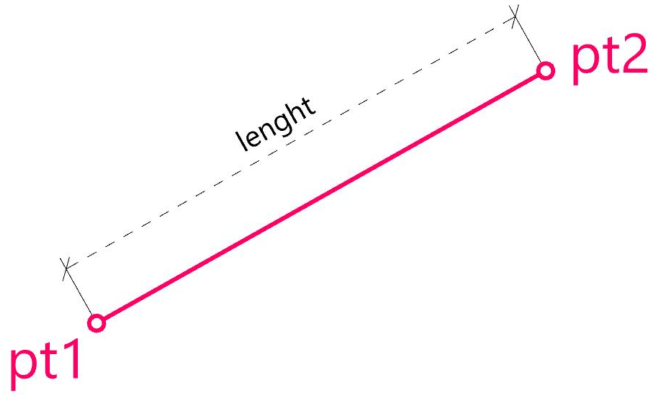
NURBS Curve



Polycurve

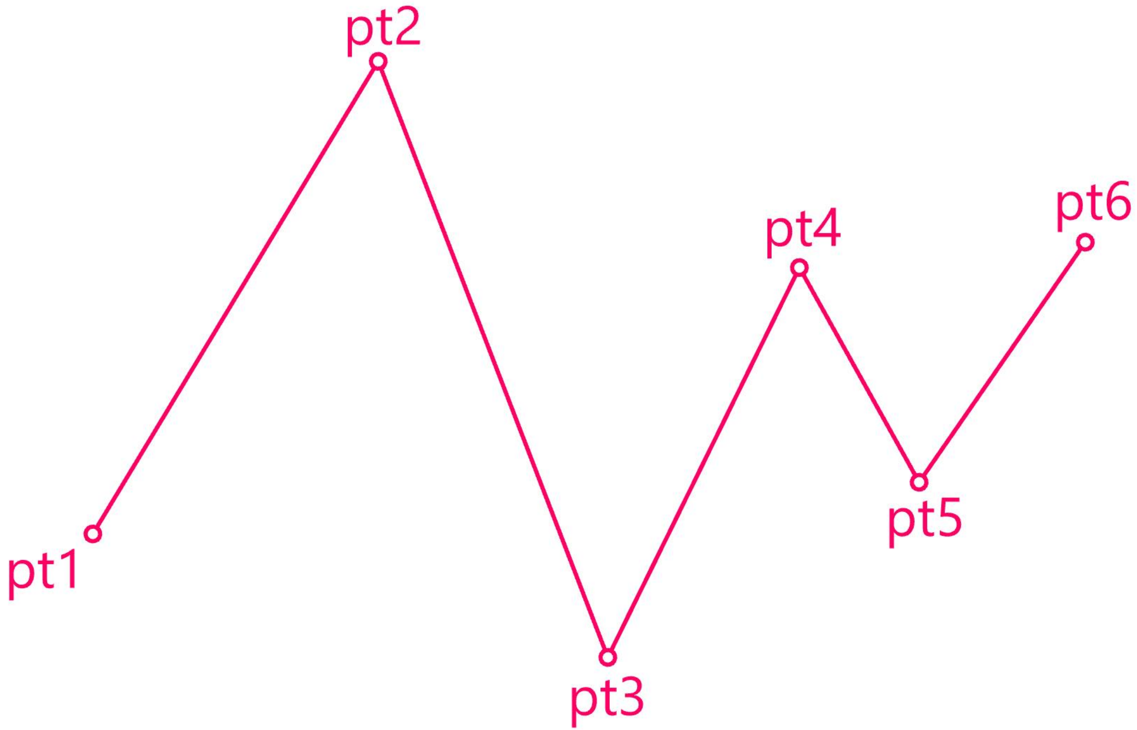
rg.Line

rg.Line(**start** Point3d, **end** Point3d)



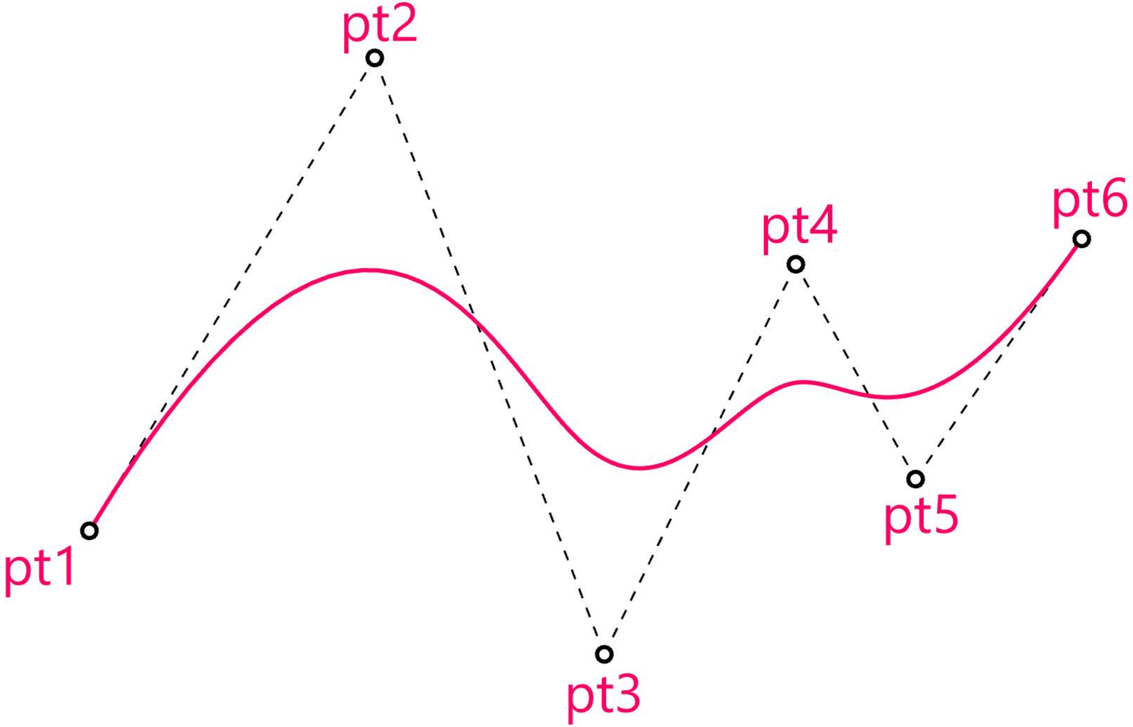
rg.Polyline

```
rg.Polyline([list of Point3d])
```



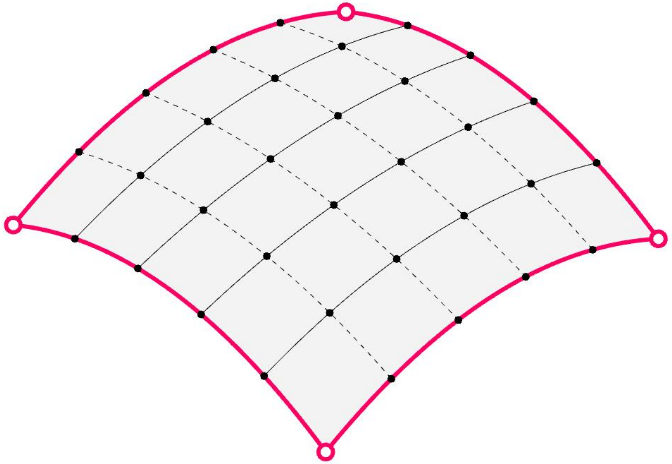
rg.Curve

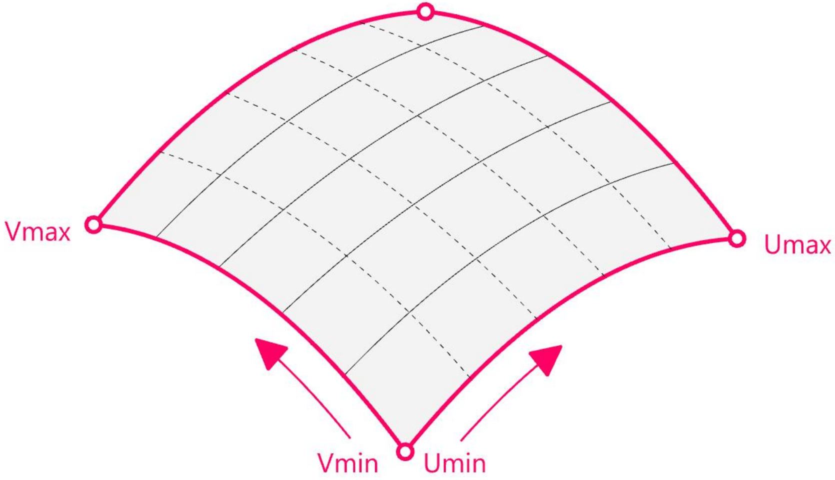
```
rg.NurbsCurve.Create(periodic bool, degree int, [list of Point3d])
```



rg.Surface

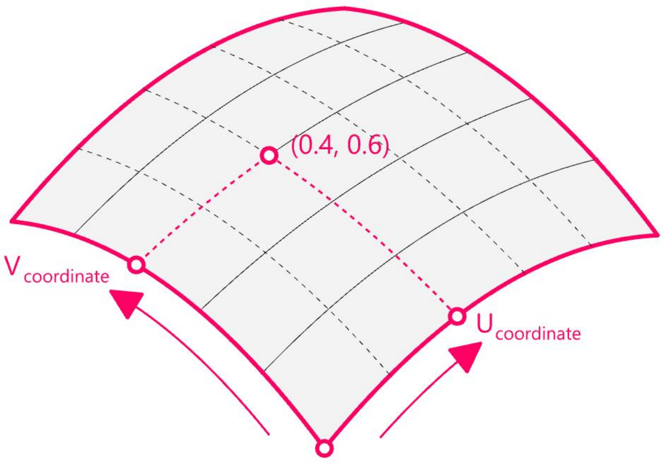
```
rg.NurbsSurface.CreateThroughPoints([list of Point3d], uCount int,
vCount int, uDegree int, vDegree int, uClosed bool, vClosed bool)
```

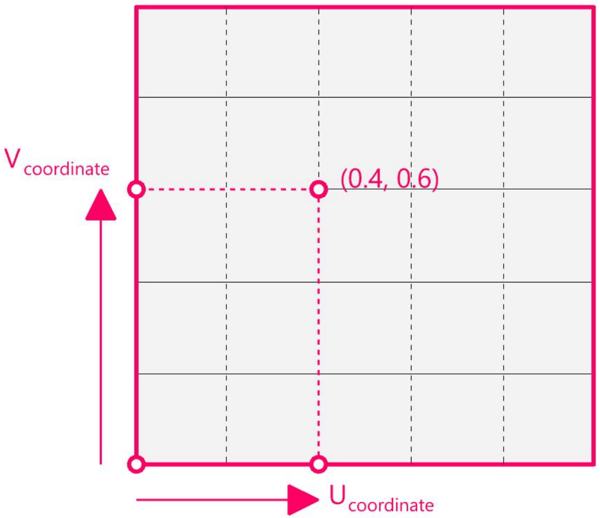




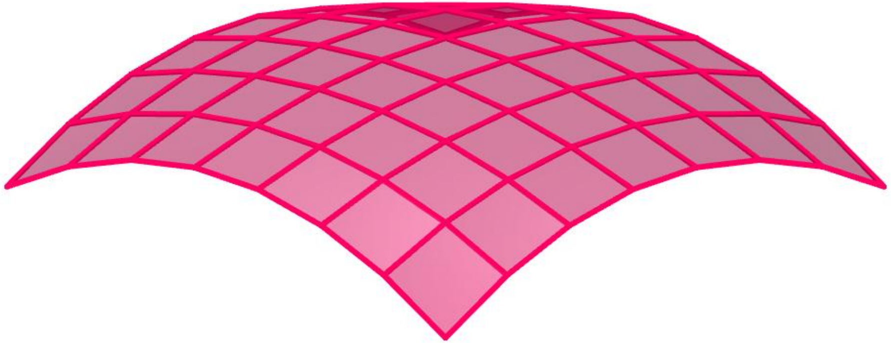
rg.Surface

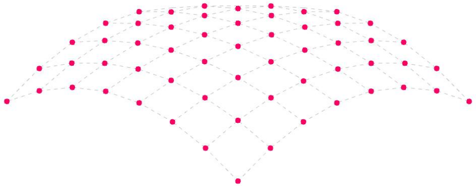
rg.NurbsSurface.**CreateThroughPoints**([**list of Point3d**], **uCount** int, **vCount** int, **uDegree** int, **vDegree** int, **uClosed** bool, **vClosed** bool)



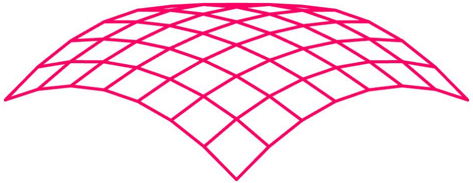


rg.Mesh

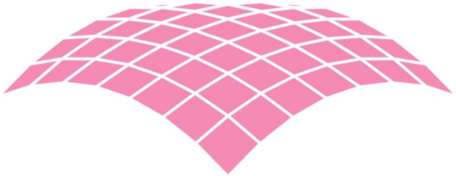




Mesh
Vertices

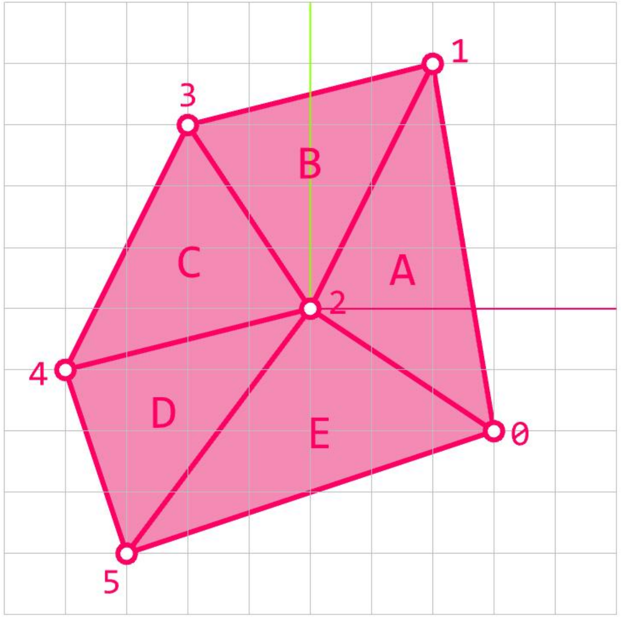


Mesh
Edges



Mesh
Faces

rg.Mesh



Vertex List

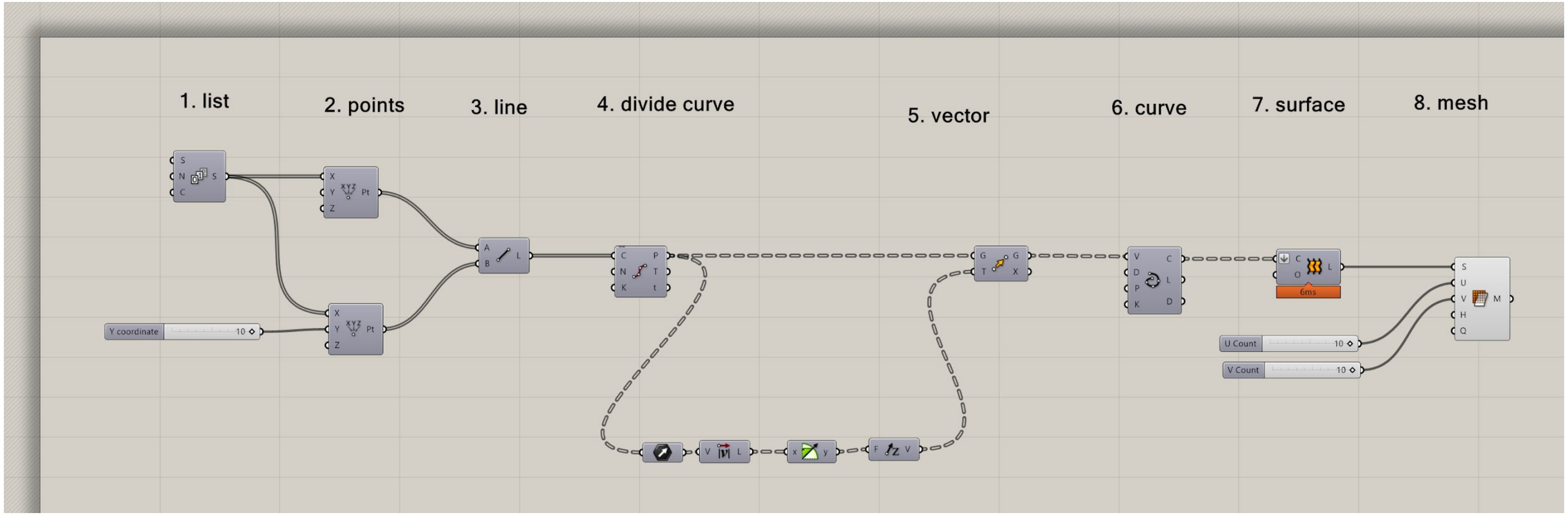
0	=	(3.0, -2.0, 0.0)
1	=	(2.0, 4.0, 0.0)
2	=	(0.0, 0.0, 0.0)
3	=	(-2.0, 3.0, 0.0)
4	=	(-4.0, -1.0, 0.0)
5	=	(-3.0, -4.0, 0.0)

Face List

A	=	{0, 2, 1}
B	=	{1, 2, 3}
C	=	{3, 2, 4}
D	=	{4, 2, 5}
E	=	{5, 2, 0}

exercise 02:

from gh to python using Rhinocommon



https://github.com/dadandroid/MPDA20/