SimplyRhino, London
February 12-14,2020

Python Scripting for Rhino/Grasshopper

Day 3 – Part 2

Long Nguyen

Camera

Example: setting camera position and direction

Please open file **Camera.gh**

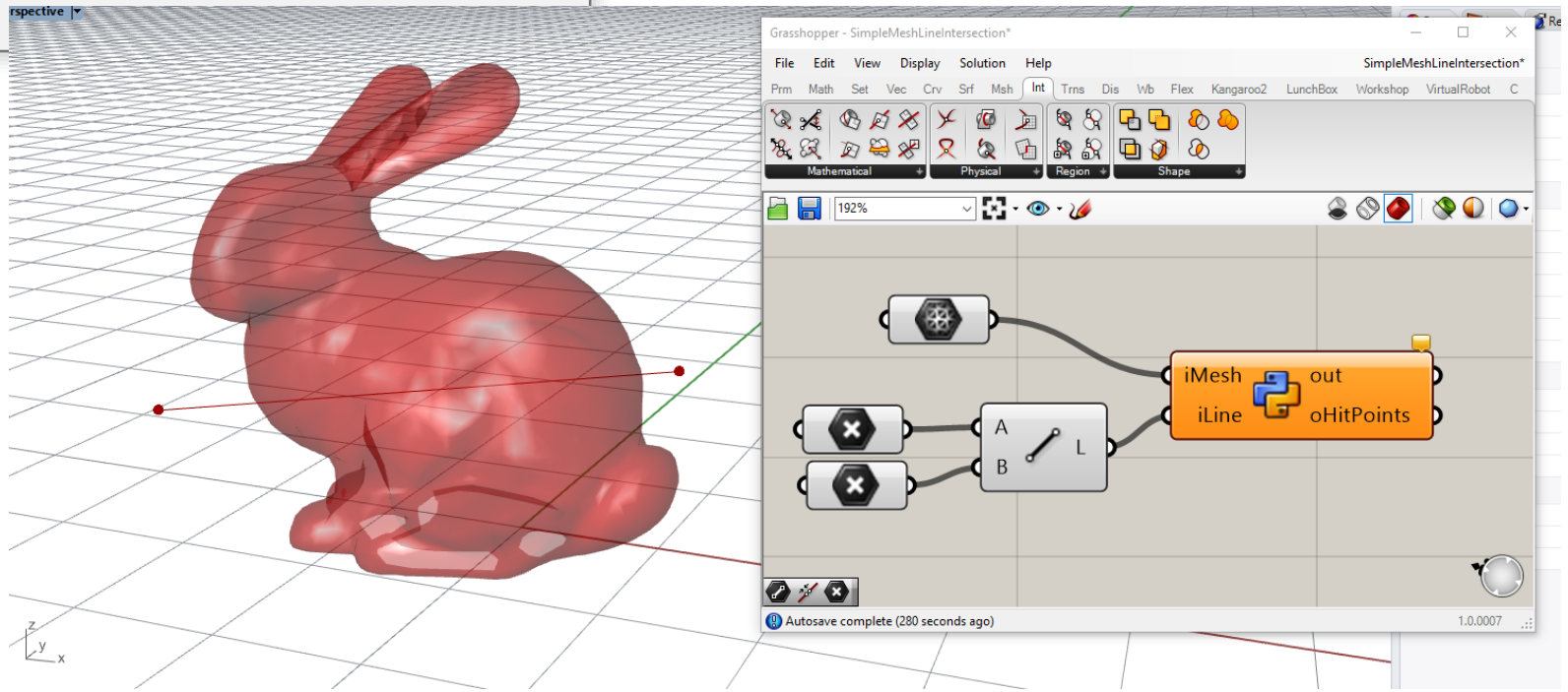
```
import Rhino

activeViewport = Rhino.RhinoDoc.ActiveDoc.Views.ActiveView.ActiveViewport
activeViewport.SetCameraLocations(iCameraTarget, iCameraLocation)
```

Geometric Intersections

Example: intersections between a line and a mesh

Please open file **SimpleMeshLineIntersections.gh**

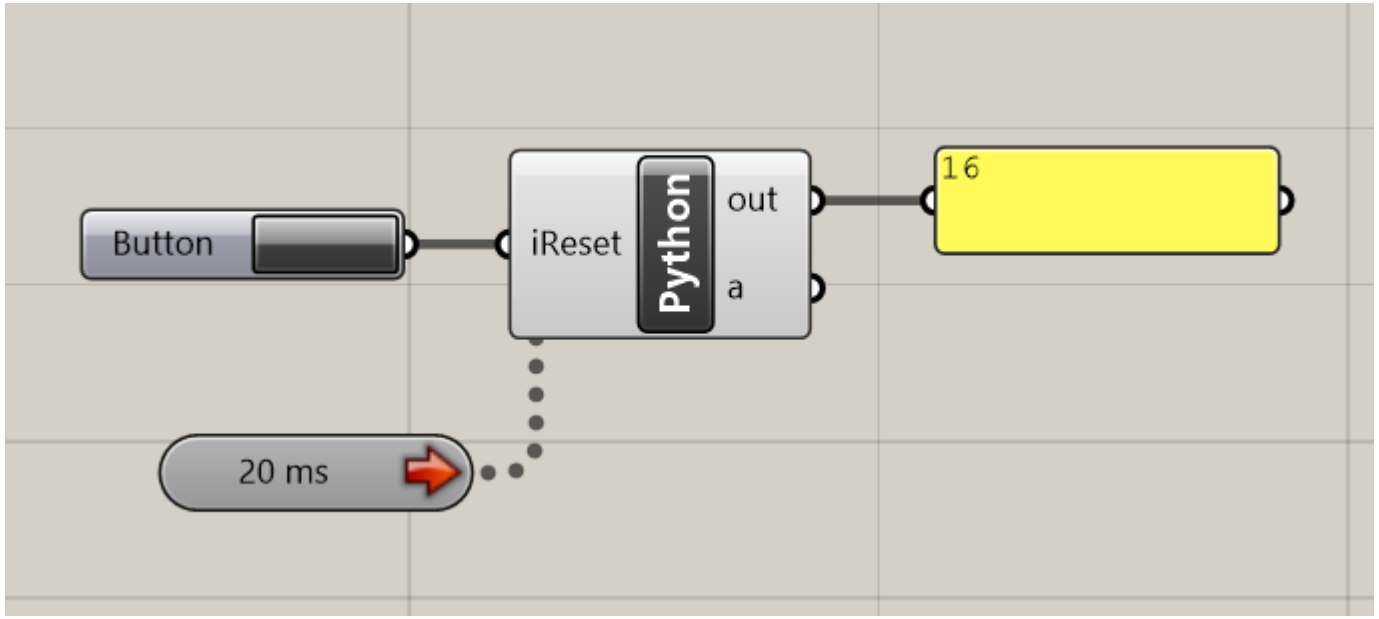
```
import Rhino.Geometry as rg
from Rhino.Geometry.Intersect import Intersection

result = Intersection.MeshLine(iMesh, iLine)
oHitPoints = result[0]
```



Persistent data

Persistent data: A simple example

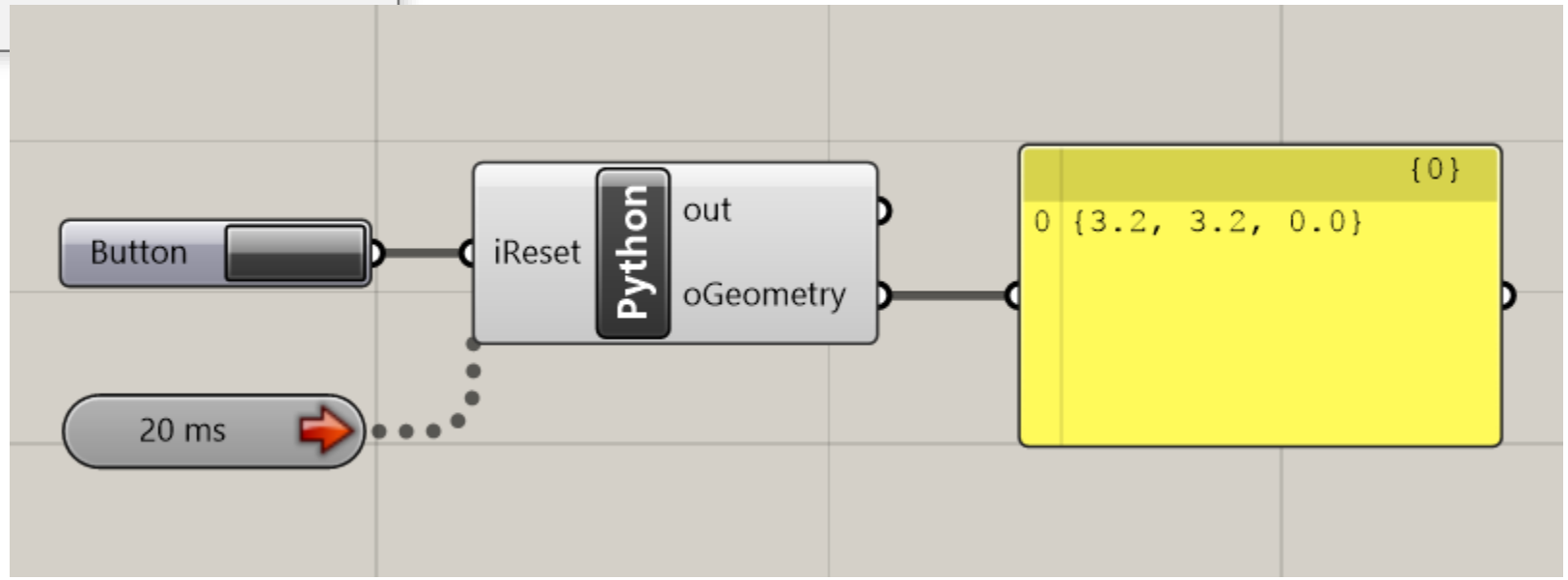

```
if iReset:
    m = 0
else:
    m += 1

print m
```



Notice that the value of the variable m is retained after each run of the Python component!

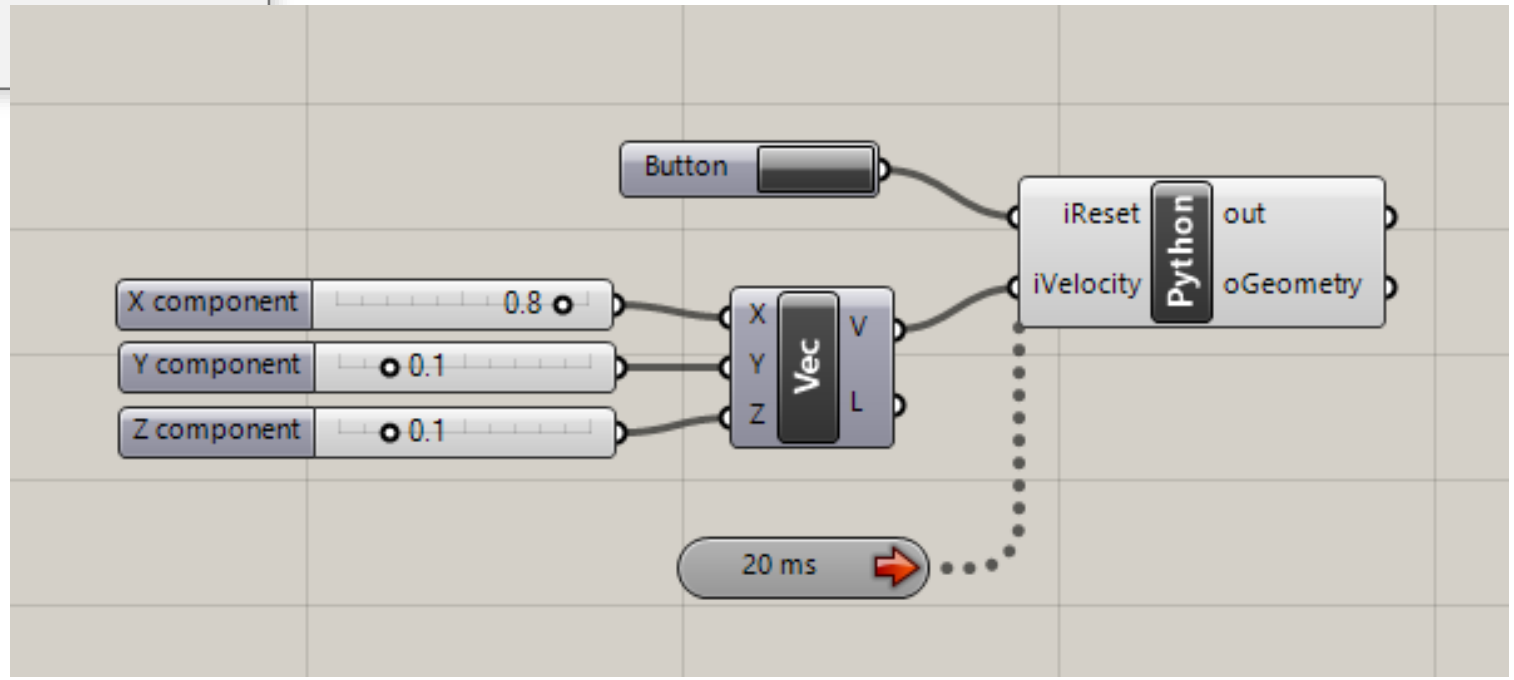
Persistent data: Animating geometries !


```python
import Rhino.Geometry as rg

if iReset:
    movingPoint = rg.Point3d(0, 0, 0)
else:
    movingPoint += rg.Vector3d(0.1, 0.1, 0.0)

oGeometry = movingPoint
```



Even better: Interactive animation !!!

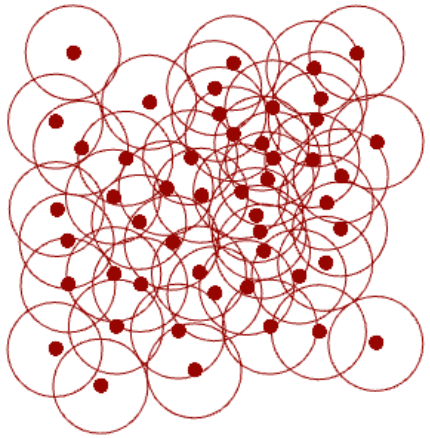

```
import Rhino.Geometry as rg

if iReset:
    movingPoint = rg.Point3d(0, 0, 0)
else:
    movingPoint += iVelocity

oGeometry = movingPoint
```

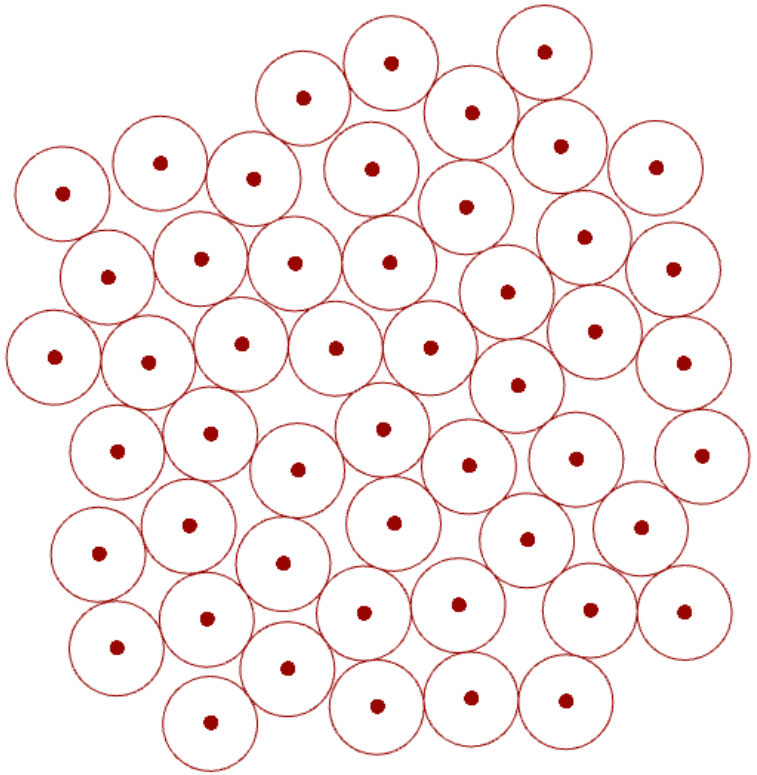


Live Example: Interactive mesh painting (using the mouse)

Live Example: "Particle" System

Circle Relaxation



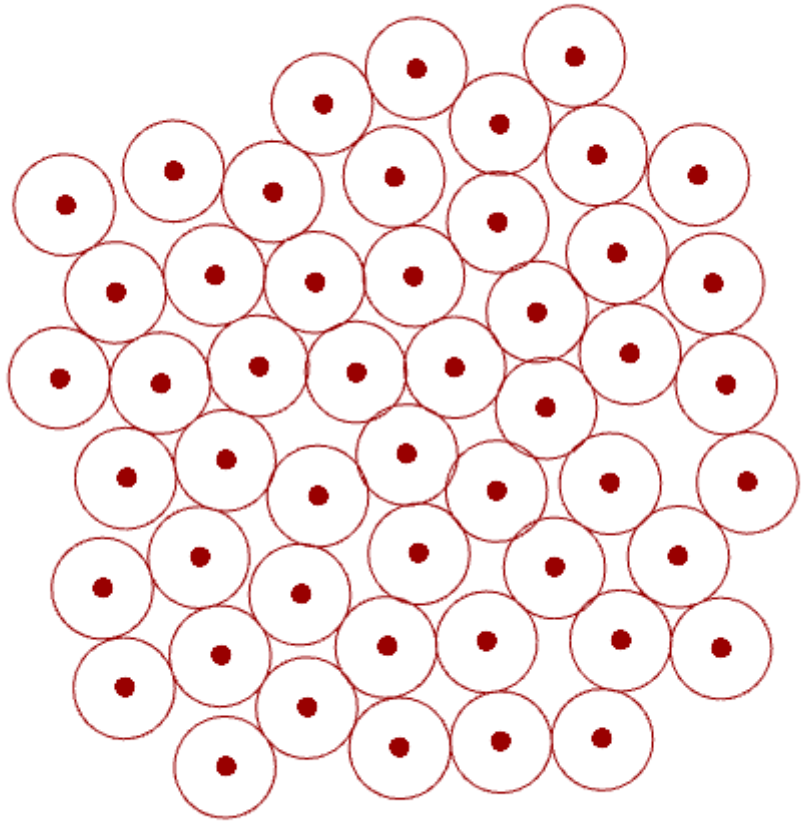


Circles overlapping

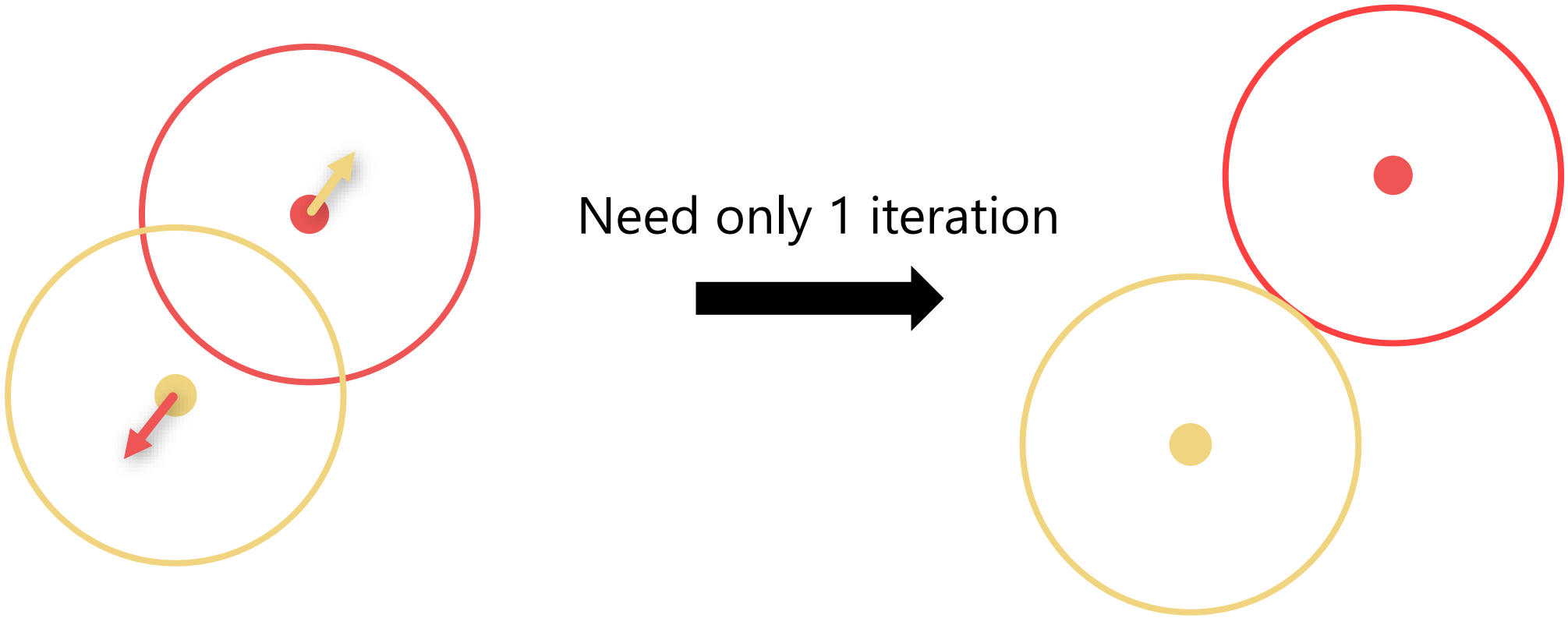
No more overlapping

Relaxation:
Iteratively push the overlapping circles away from each other

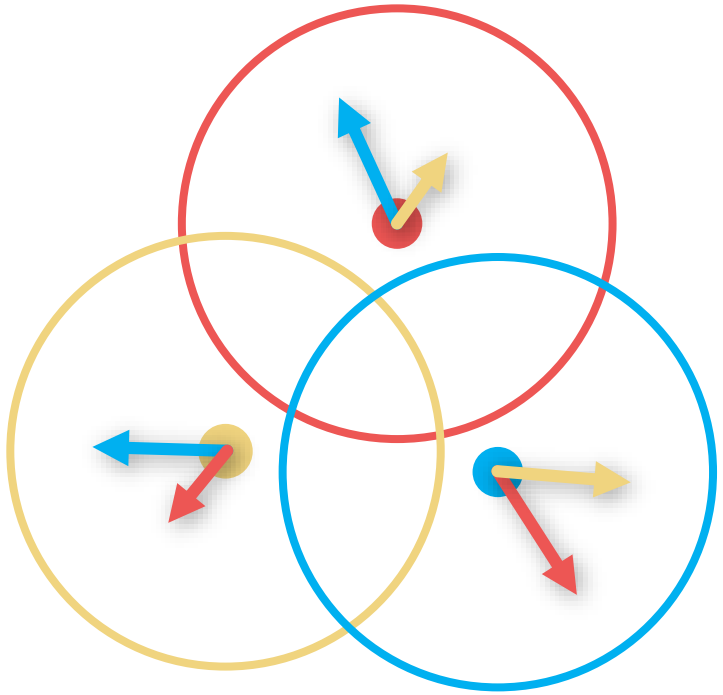
Iteration 0
1
2
3
4
5
6
7
…
17
…
100



Super simple if there only 2 circles

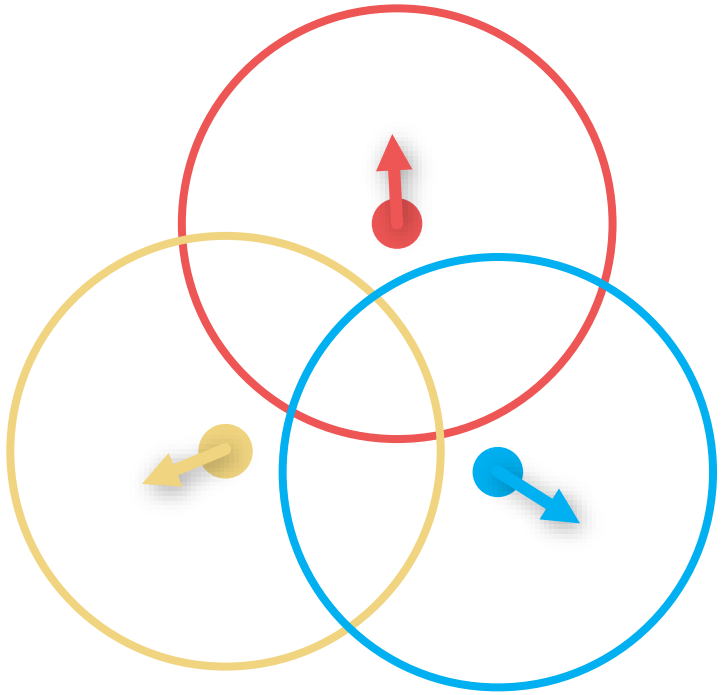


If there are more than 2 circles:



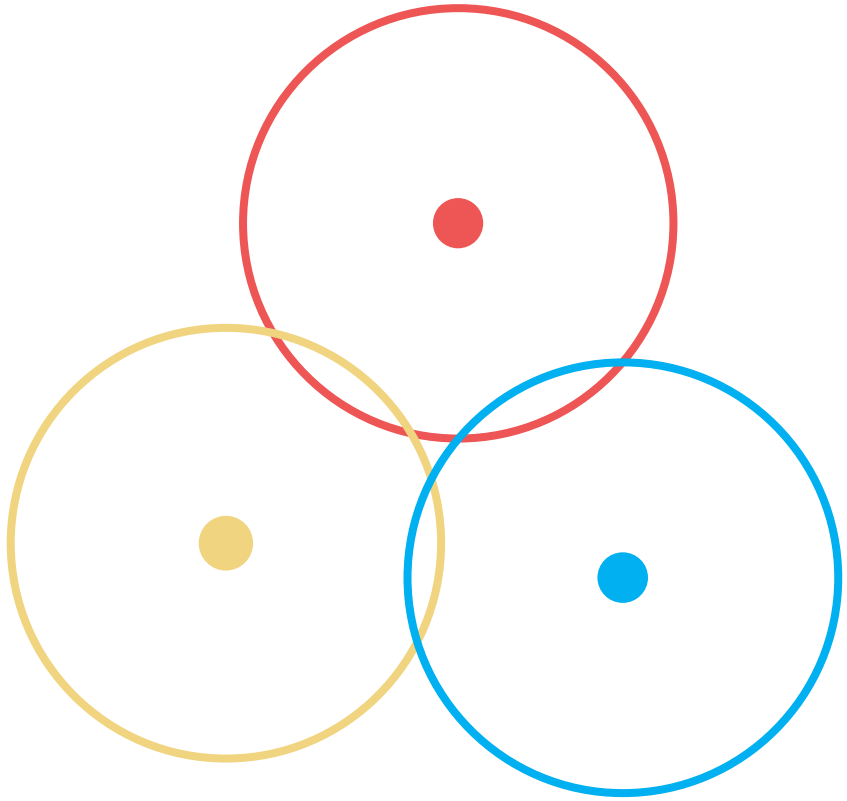
Step 1: Consider each pair of circles and calculate the move vectors to push them apart.

If there are more than 2 circles:



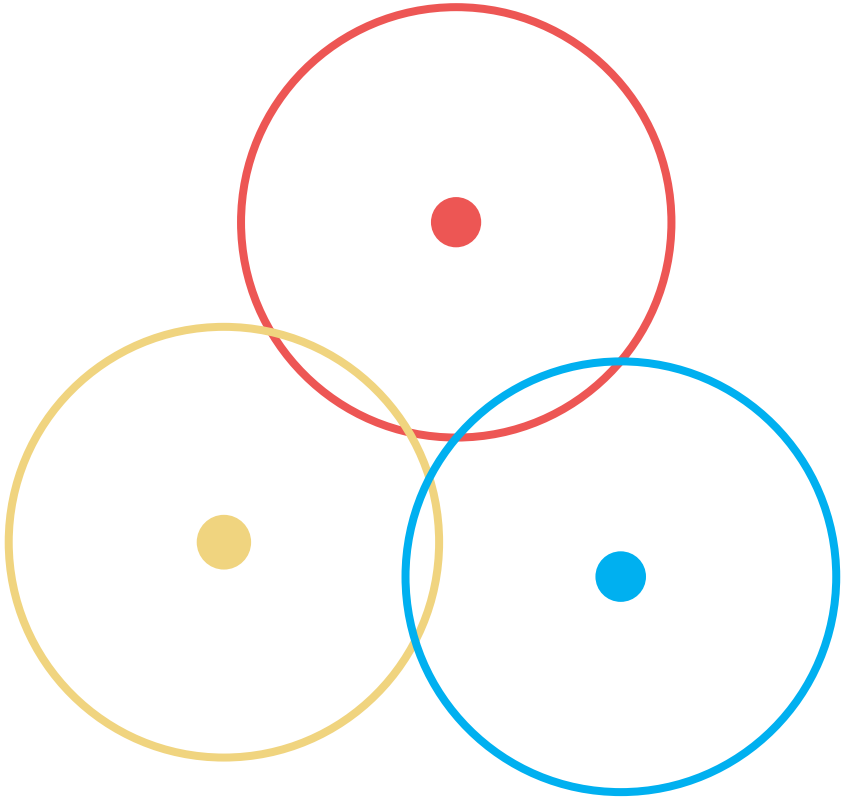
Step 2: Compute the AVERAGE move vector for each circle

If there are more than 2 circles:



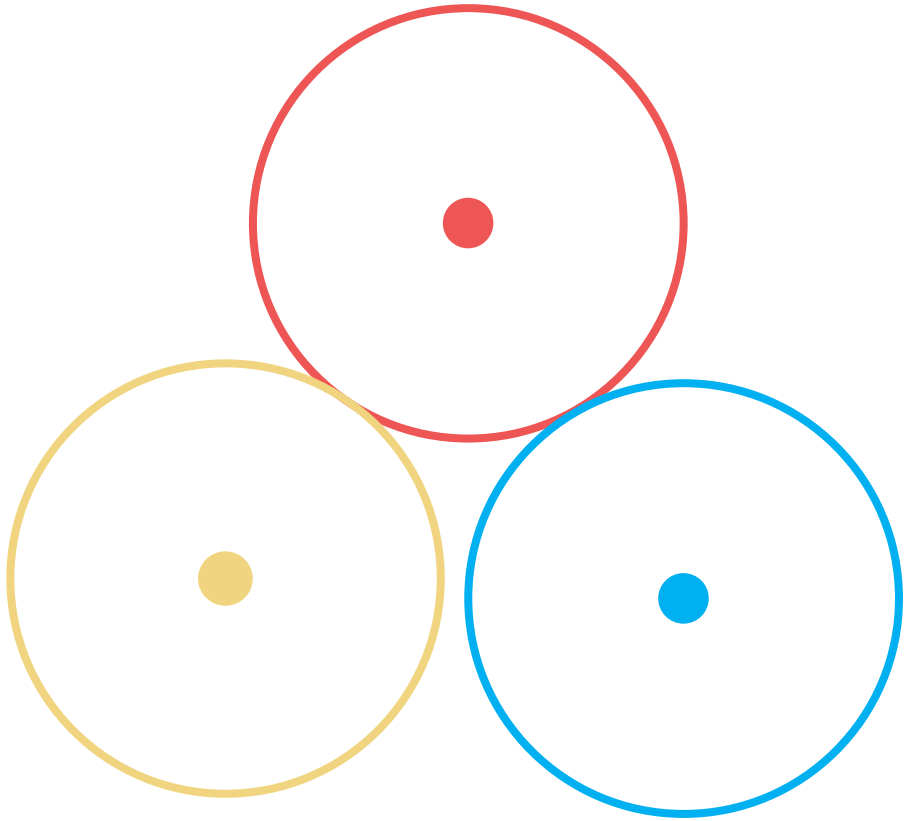
Step 3: Move each circle according to its AVERAGE move vector.

If there are more than 2 circles:



Notice how the overlapping has been slightly reduced!

If there are more than 2 circles:



Repeat the entire process many times
until there is near 0 overlapping

Object-Oriented Programming

Data types

- Built-in Python types: int, float, str, bool, list
- Externally-defined types: Point3d, Vector3d, Curve, Mesh

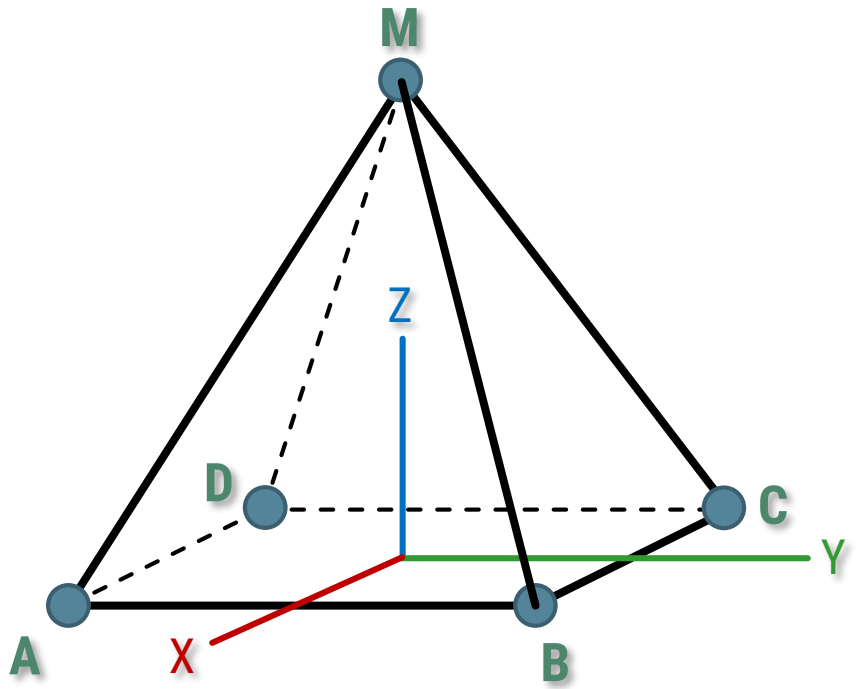
```
import Rhino.Geometry as rg

myMesh = rg.Mesh()
myPoint = rg.Point3d(0.2, 3, 4.2)
```

How about:

Our own custom data types?

Example: The “Pyramid” data type

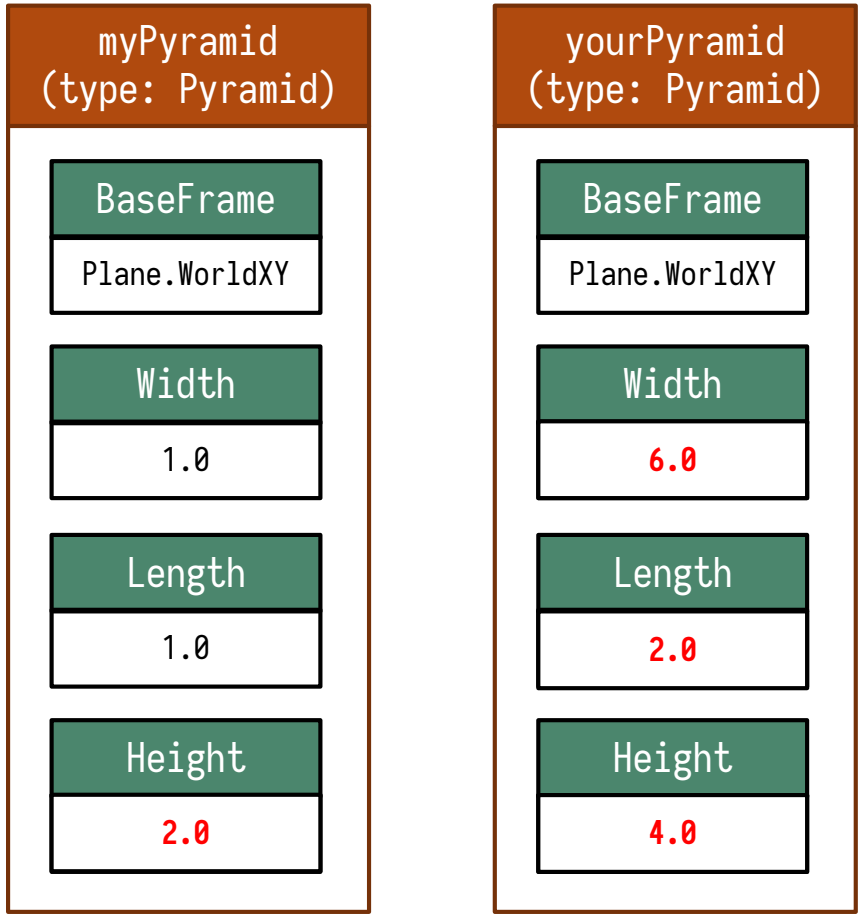

```
import Rhino.Geometry as rg

class Pyramid:
    def __init__(self):
        self.BaseFrame = rg.Plane.WorldXY
        self.Width = 1.0
        self.Length = 1.0
        self.Height = 1.0
```



Example: The “Pyramid” data type


```
import Rhino.Geometry as rg

class Pyramid:
    def __init__(self):
        self.BaseFrame = rg.Plane.WorldXY
        self.Width = 1.0
        self.Length = 1.0
        self.Height = 1.0

# Main Script:

myPyramid = Pyramid()
print myPyramid.Width
myPyramid.Height = 2
print myPyramid.Height

yourPyramid = Pyramid()
yourPyramid.Length = 2
yourPyramid.Width = 6
yourPyramid.Height = 4
```

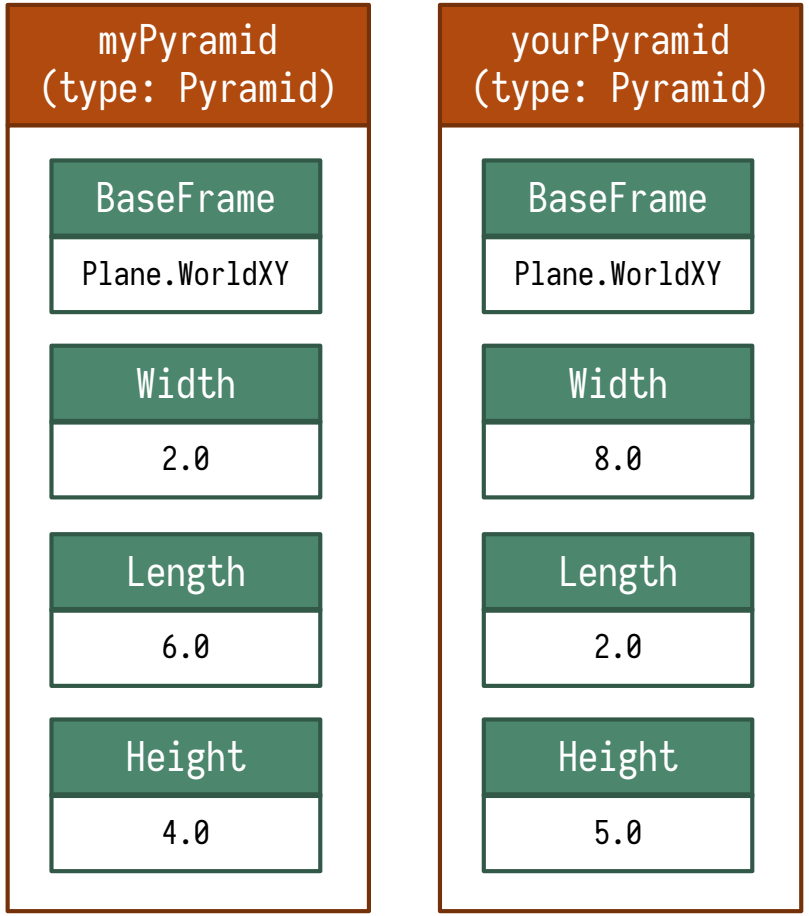
```
# Similar to:
myMesh = rg.Mesh()
```



Constructor with parameters


```
# Similar to:
myMesh = rg.Point3d(3.1, 2.0, 3.0)
```

```
import Rhino.Geometry as rg

class Pyramid:
    def __init__(self, baseFrame, width, length, height):
        self.BaseFrame = baseFrame
        self.Width = width
        self.Length = length
        self.Height = height

# Main Script:

myPyramid   = Pyramid(rg.Plane.WorldXY, 2, 6, 4)
yourPyramid = Pyramid(rg.Plane.WorldYZ, 8, 2, 5)
print myPyramid.Length
print yourPyramid.Length
```




```
2
8
```

A data type has a set of pre-defined methods:

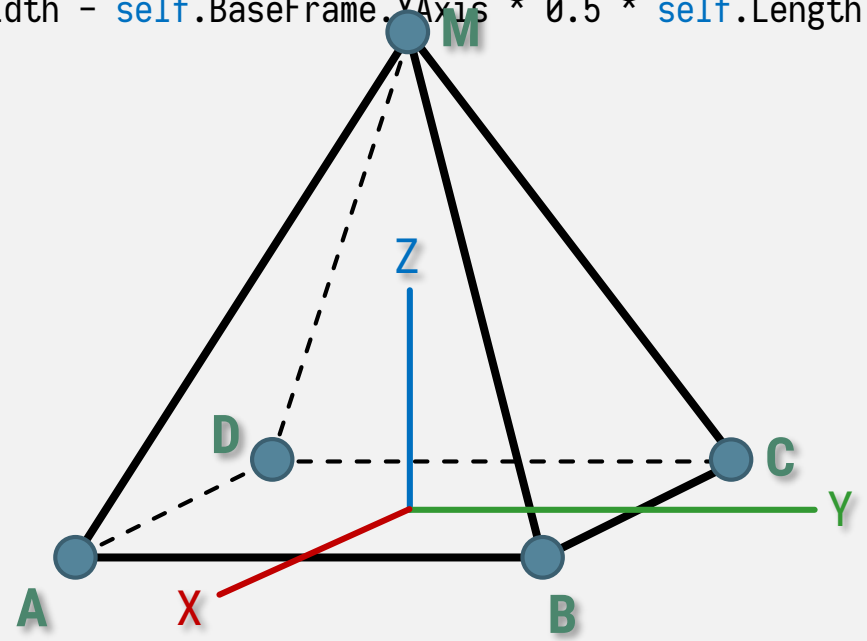
```
import Rhino.Geometry as rg

myLine = rg.Line(...)
myLine.PointAt(0.3)
```

Defining the method GetVolume for our Pyramid class


```
import Rhino.Geometry as rg

class Pyramid:
    def __init__(self, baseFrame, width, length, height):
        self.BaseFrame = baseFrame
        self.Width = width
        self.Length = length
        self.Height = height

    def GetVolume(self):
        return self.Width * self.Height * self.Length / 3

# Main Script:

myPyramid   = Pyramid(rg.Plane.WorldXY, 2, 6, 4)
yourPyramid = Pyramid(rg.Plane.WorldYZ, 8, 2, 5)
print myPyramid.GetVolume()
print yourPyramid.GetVolume()
```

Computing the edge lines of the pyramid


```
import Rhino.Geometry as rg

class Pyramid:
    def __init__(self, baseFrame, width, length, height):
        self.BaseFrame = baseFrame
        self.Width = width
        self.Length = length
        self.Height = height

    ...

    def GetEdges(self):
        A = self.BaseFrame.Origin - self.BaseFrame.XAxis * 0.5 * self.Width + self.BaseFrame.YAxis * 0.5 * self.Length
        B = self.BaseFrame.Origin + self.BaseFrame.XAxis * 0.5 * self.Width + self.BaseFrame.YAxis * 0.5 * self.Length
        C = self.BaseFrame.Origin + self.BaseFrame.XAxis * 0.5 * self.Width - self.BaseFrame.YAxis * 0.5 * self.Length
        D = self.BaseFrame.Origin - self.BaseFrame.XAxis * 0.5 * self.Width - self.BaseFrame.YAxis * 0.5 * self.Length
        M = self.BaseFrame.Origin + self.BaseFrame.ZAxis * self.Height
        edges = []
        edges.append(rg.Line(A, B))
        edges.append(rg.Line(B, C))
        edges.append(rg.Line(C, D))
        edges.append(rg.Line(D, A))
        edges.append(rg.Line(A, M))
        edges.append(rg.Line(B, M))
        edges.append(rg.Line(C, M))
        edges.append(rg.Line(D, M))
        return edges

# Main Script:

myPyramid = Pyramid(rg.Plane.WorldXY, 2, 6, 4)
oEdges = myPyramid.GetEdges()
```



A method can modify the values stored in the properties


```
import Rhino.Geometry as rg

myVector = rg.Vector3d(2, 2, 2)
myVector.Unitize()
print myVector.X
```

A method can modify the values stored in the properties


```
import Rhino.Geometry as rg

class Pyramid:
    def __init__(self, baseFrame, width, length, height):
        self.BaseFrame = baseFrame
        self.Width = width
        self.Length = length
        self.Height = height

    ...

    def Scale(self, scaleFactor):
        self.Length *= scaleFactor
        self.Width *= scaleFactor
        self.Height *= scaleFactor

# Main Script:

myPyramid = Pyramid(rg.Plane.WorldXY, 2, 6, 4)
myPyramid.Scale(2.5)
print myPyramid.Length
```

A method can create a brand new object of the same class


```
yourBrep = myBrep.Duplicate()
```

A method can create a brand new object of the same class


```
import Rhino.Geometry as rg

class Pyramid:
    def __init__(self, baseFrame, width, length, height):
        self.BaseFrame = baseFrame
        self.Width = width
        self.Length = length
        self.Height = height

    ...

    def Duplicate(self):
        newPyramid = Pyramid(self.BaseFrame, self.Width, self.Length, self.Height)
        return newPyramid

# Main Script:

myPyramid = Pyramid(rg.Plane.WorldXY, 2, 6, 4)
yourPyramid = myPyramid.Duplicate()
```

Within a class, a method can call another method


```
import Rhino.Geometry as rg

class Pyramid:
    def __init__(self, baseFrame, width, length, height):
        self.BaseFrame = baseFrame
        self.Width = width
        self.Length = length
        self.Height = height

    ...

    def GetVolume(self):
        return self.Width * self.Height * self.Length / 3

    def ComputeDensity(self, mass):
        volume = self.GetVolume()
        return mass / volume;

# Main Script:

myPyramid = Pyramid(rg.Plane.WorldXY, 2, 6, 4)
print myPyramid.ComputeDensity(4.5)
```

We can send a Pyramid object into a function/method


```
import Rhino.Geometry as rg

class Pyramid:
    ...


def CreateLineBetweenTwoPyramid(pyramidA, pyramidB):
    topA = pyramidA.BaseFrame.Origin + pyramidA.BaseFrame.ZAxis * pyramidA.Height
    topB = pyramidB.BaseFrame.Origin + pyramidB.BaseFrame.ZAxis * pyramidB.Height
    return rg.Line(topA, topB)

# Main Script:

myPyramid = Pyramid(rg.Plane.WorldXY, 2, 6, 4)
yourPyramid = Pyramid(rg.Plane(rg.Point3d(20, 10, 0)), 2, 6, 4)
connectingLine = CreateLineBetweenTwoPyramid(myPyramid, yourPyramid)
```

Live Example: Wandering Particles

A simple class that describes moving particles


```
import Rhino.Geometry as rg

class Particle:

    def __init__(self):
        self.Position = rg.Point3d.Origin
        self.Velocity = rg.Vector3d(0.1, 0.1, 0.0)

    def Update(self):
        self.Position += self.Velocity

# Main Script:

if iReset:
    myParticle = Particle()
else:
    myParticle.Update()

oGeometry = myParticle.Position
```

Applying random change to velocity


```
import Rhino.Geometry as rg
import random

class Particle:

    def __init__(self):
        self.Position = rg.Point3d.Origin
        self.Velocity = rg.Vector3d(0.1, 0.1, 0.0)

    def Update(self):
        self.Velocity.Rotate(random.uniform(-0.2, 0.2), rg.Vector3d.ZAxis)
        self.Position += self.Velocity

# Main Script:

if iReset:
    myParticle = Particle()
else:
    myParticle.Update()

oGeometry = myParticle.Position
```

Record the position history


```
import Rhino.Geometry as rg
import random

class Particle:

    def __init__(self):
        self.Position = rg.Point3d.Origin
        self.Velocity = rg.Vector3d(0.1, 0.1, 0.0)
        self.History = [self.Position]

    def Update(self):
        self.Velocity.Rotate(random.uniform(-0.2, 0.2), rg.Vector3d.ZAxis)
        self.Position += self.Velocity
        self.History.append(self.Position)

# Main Script:

if iReset:
    myParticle = Particle()
else:
    myParticle.Update()

oGeometry = myParticle.Position
oGeometry = myParticle.History
```

Draw the travelling path


```
import Rhino.Geometry as rg
import random

class Particle:

    def __init__(self):
        self.Position = rg.Point3d.Origin
        self.Velocity = rg.Vector3d(0.1, 0.1, 0.0)
        self.History = [self.Position]

    def Update(self):
        self.Velocity.Rotate(random.uniform(-0.2, 0.2), rg.Vector3d.ZAxis)
        self.Position += self.Velocity
        self.History.append(self.Position)

# Main Script:

if iReset:
    myParticle = Particle()
else:
    myParticle.Update()

oGeometry = myParticle.History
oGeometry = rg.PolylineCurve(myParticle.History)
```

Let's have two particles instead of one!


```
import Rhino.Geometry as rg
import random


class Particle:
    ...

# Main Script:

if iReset:
    myParticle = Particle()
    yourParticle = Particle()
else:
    myParticle.Update()
    yourParticle.Update()

oGeometry = rg.Polyline(List[rg.Point3d](myParticle.History))
paths = []
paths.append(rg.PolylineCurve(myParticle.History))
paths.append(rg.PolylineCurve(yourParticle.History))
oGeometry = paths
```

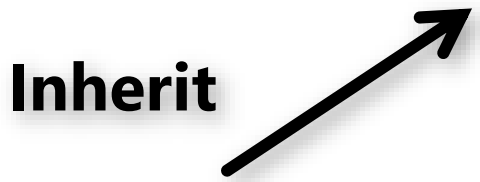
10-minute extension exercise

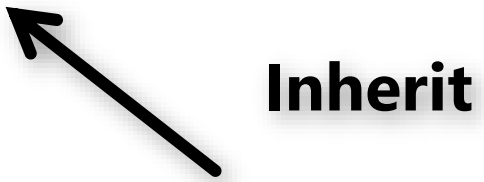
Let’s have plenty (30) of particles

Advanced concepts in Object-Oriented Programming

Inheritance

```python
class LandAnimal:
    def __init__(self, name, weight, legCount):
        self.Name = name
        self.Weight = weight
        self.LegCount = legCount
    def Eat(self):
        ...
```

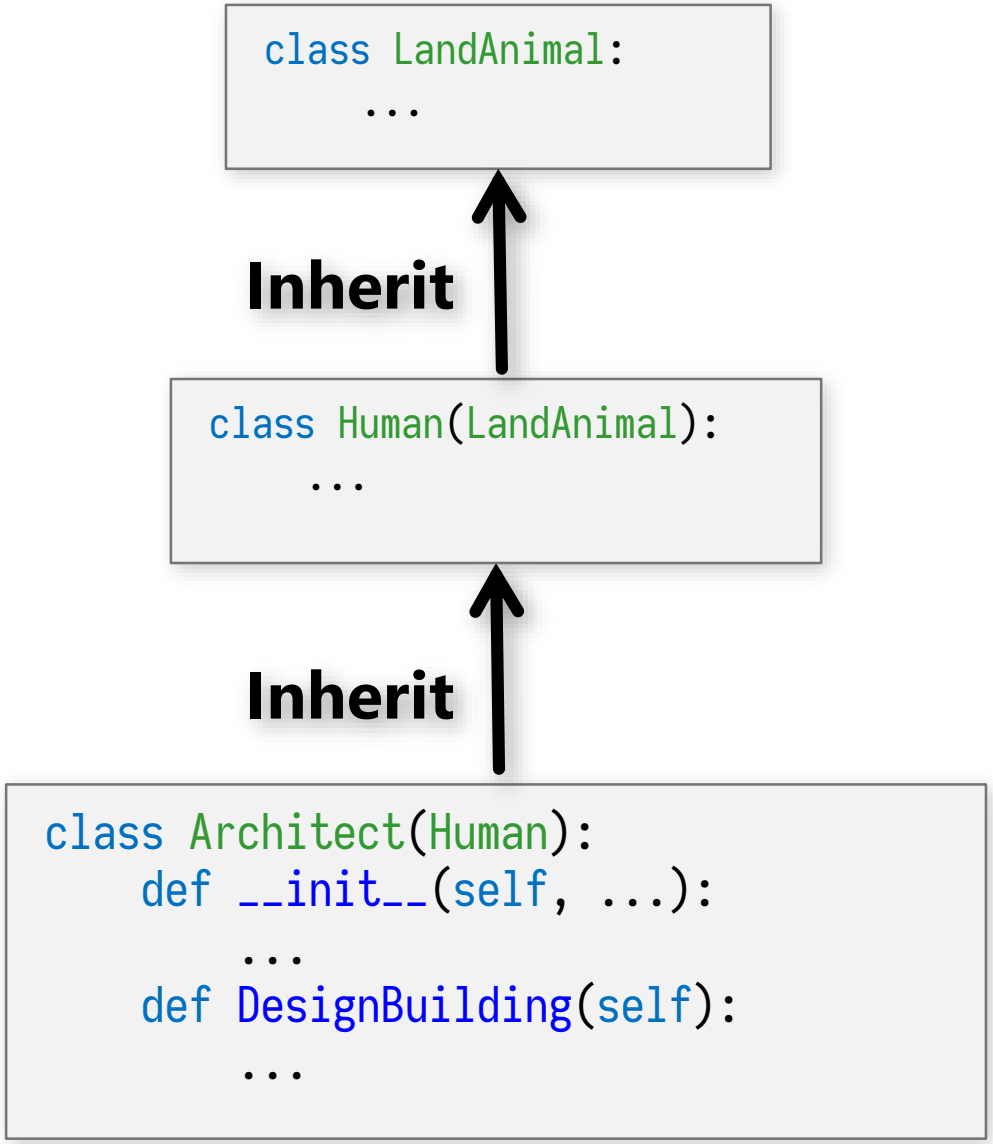





```python
class Dog(LandAnimal):
    def __init__(self, name, weight, breed):
        self.Name = name
        self.Weight = weight
        self.Breed = breed
        self.LegCount = 4
    def Bark(self):
        ...
```

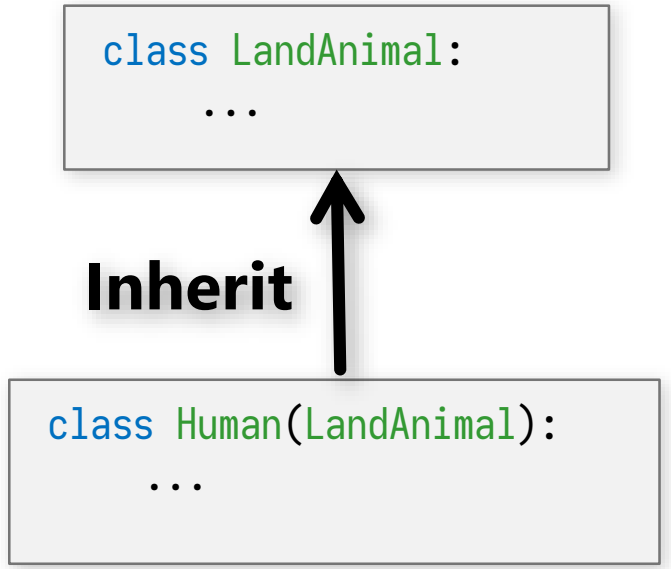
```python
class Human(LandAnimal):
    def __init__(self, name, weight, nationality):
        self.Name = name
        self.Weight = weight
        self.Nationality = nationality
        self.LegCount = 2
    def ChangeNationality(self):
        ...
```

Inheritance can be indirect and multi-level

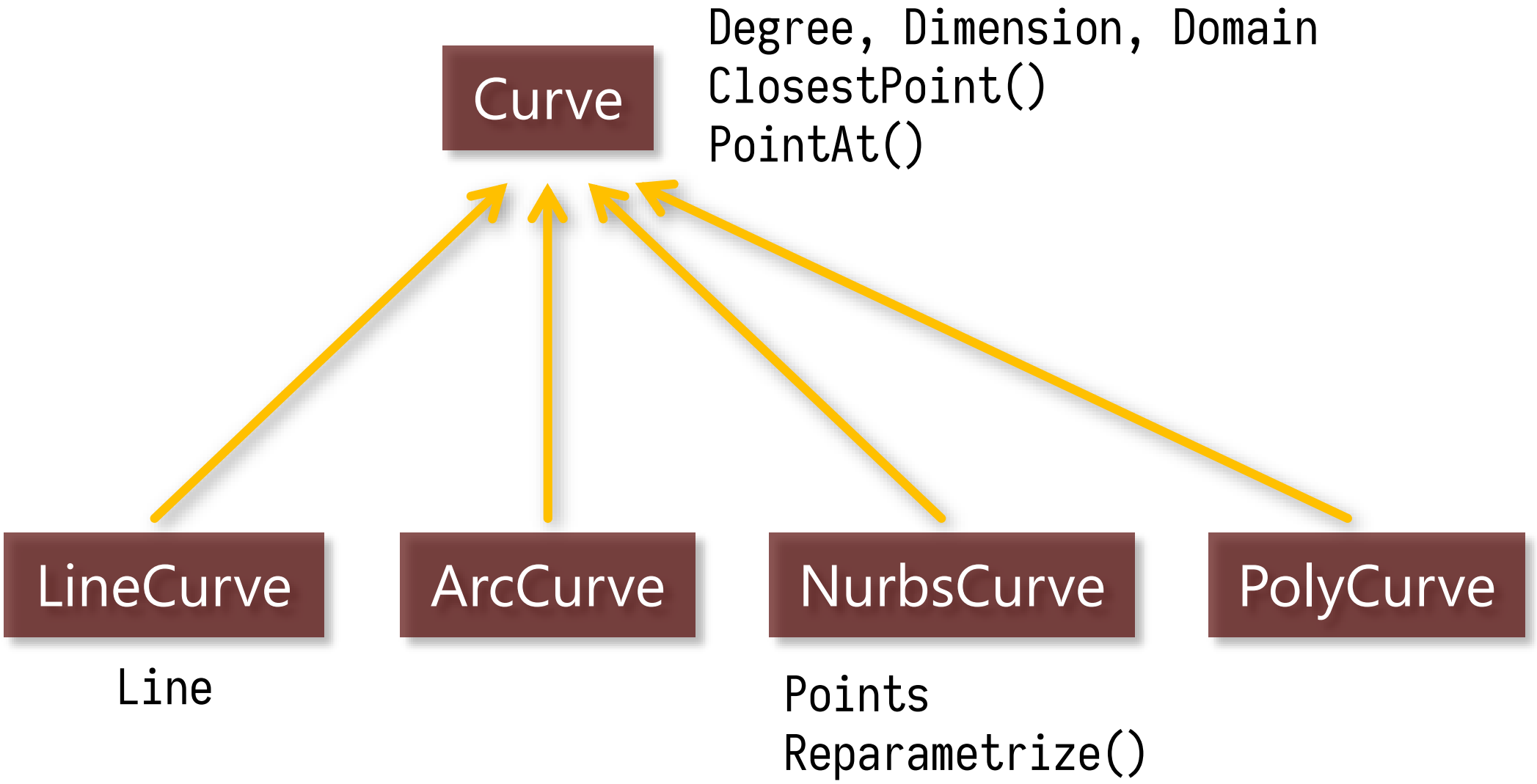


Some terminologies

- *Dog* **inherits** *LandAnimal*
- *Dog* is **derived** from *LandAnimal*
- *LandAnimal* is the **parent class** of *Dog*
  base class
- *Dog* is a **child class** of *LandAnimal*
  derived class
  subclass



Example of class inheritance in RhinoCommon



Example of class inheritance in RhinoCommon