Dynamo Primer

Paolo Emilio Serra

Implementation Consultant | @PaoloESerra; puntorevit.blogspot.com

AUTODESK.

Agenda

- Dynamo Overview
- User Interface
- Graphs Management
- Autodesk Standards
- Visual Programming Principles
- Filtering, Grouping & Sorting
- Dynamo-Excel Link
- Design Script
- Geometry Library
- Automation Applications
- Dynamo for Revit
- Dynamo & Python
- Object Oriented Programming
- Revit API Introduction
- Next Steps

Dynamo Overview

What is Dynamo?

- Open-source software platform
- Visual interface to construct logic routines
- Geometry creation
- Workflow automation
- Interface for multiple software

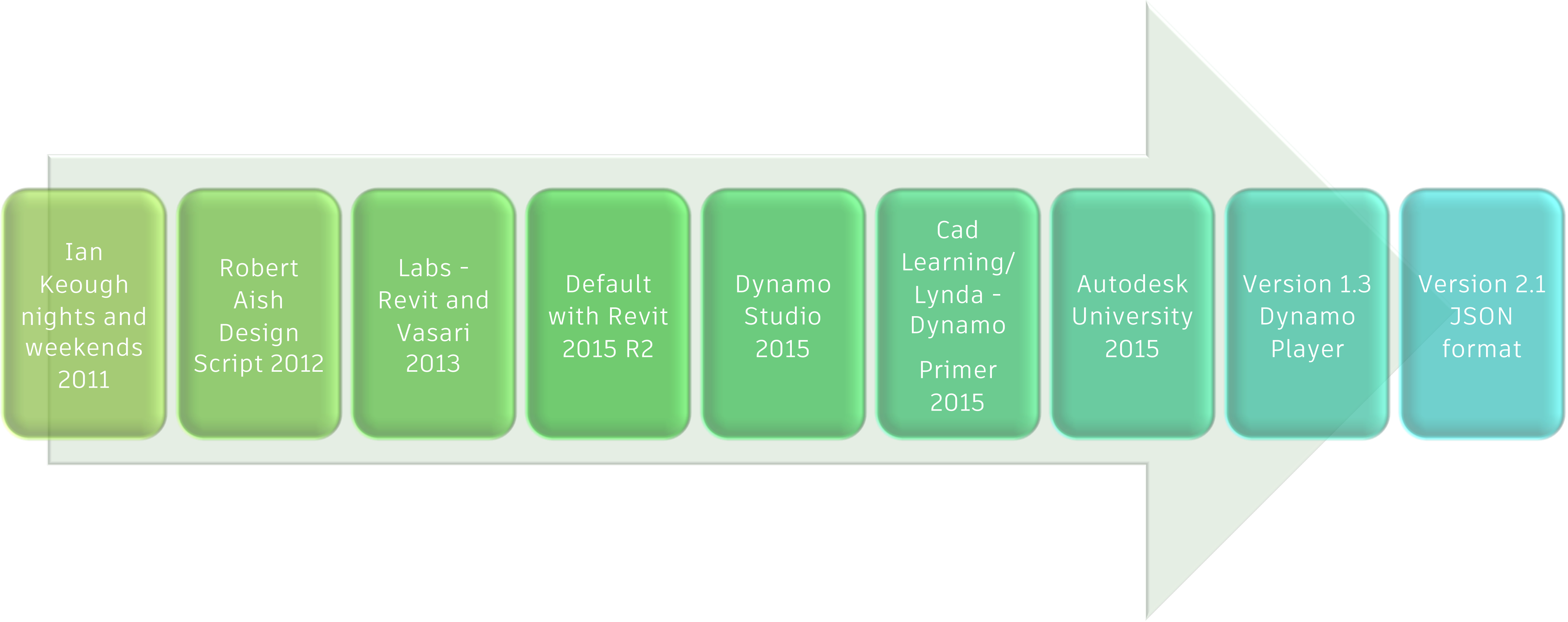
Automation Business Values

- Reducing man hours
- Ensure Data completeness
- Enable Interoperability between different platforms
- Improve the efficiency of existing workflows and create new ones
- Improve the collaboration
- Enhance technology adoption

Learning Resources

- DynamoBIM.org
- DynamoPrimer.com
- DynamoNodes.com
- GitHub/DynamoDS
- http://dynamods.github.io/DynamoAPI/
- Blogs, YouTube videos
- AU lessons and handouts
- http://www.revitapidocs.com/code/
- Lynda.com / CadLearning.com

Dynamo Timeline



Ian Keough nights and weekends 2011

Robert Aish Design Script 2012

Labs - Revit and Vasari 2013

Default with Revit 2015 R2

Dynamo Studio 2015

Cad Learning/ Lynda - Dynamo

Primer 2015

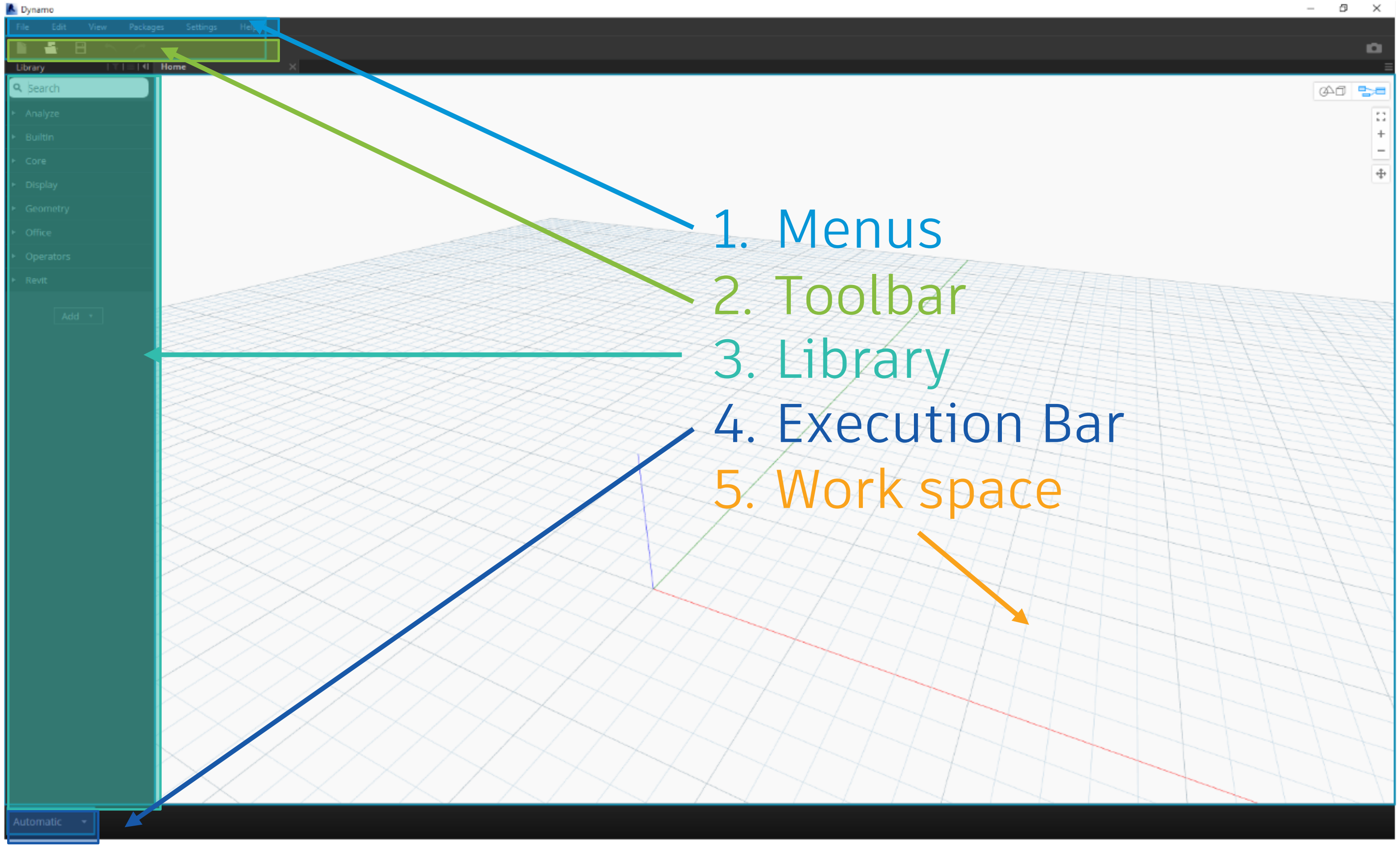
Autodesk University 2015

Version 1.3 Dynamo Player

Version 2.1 JSON format

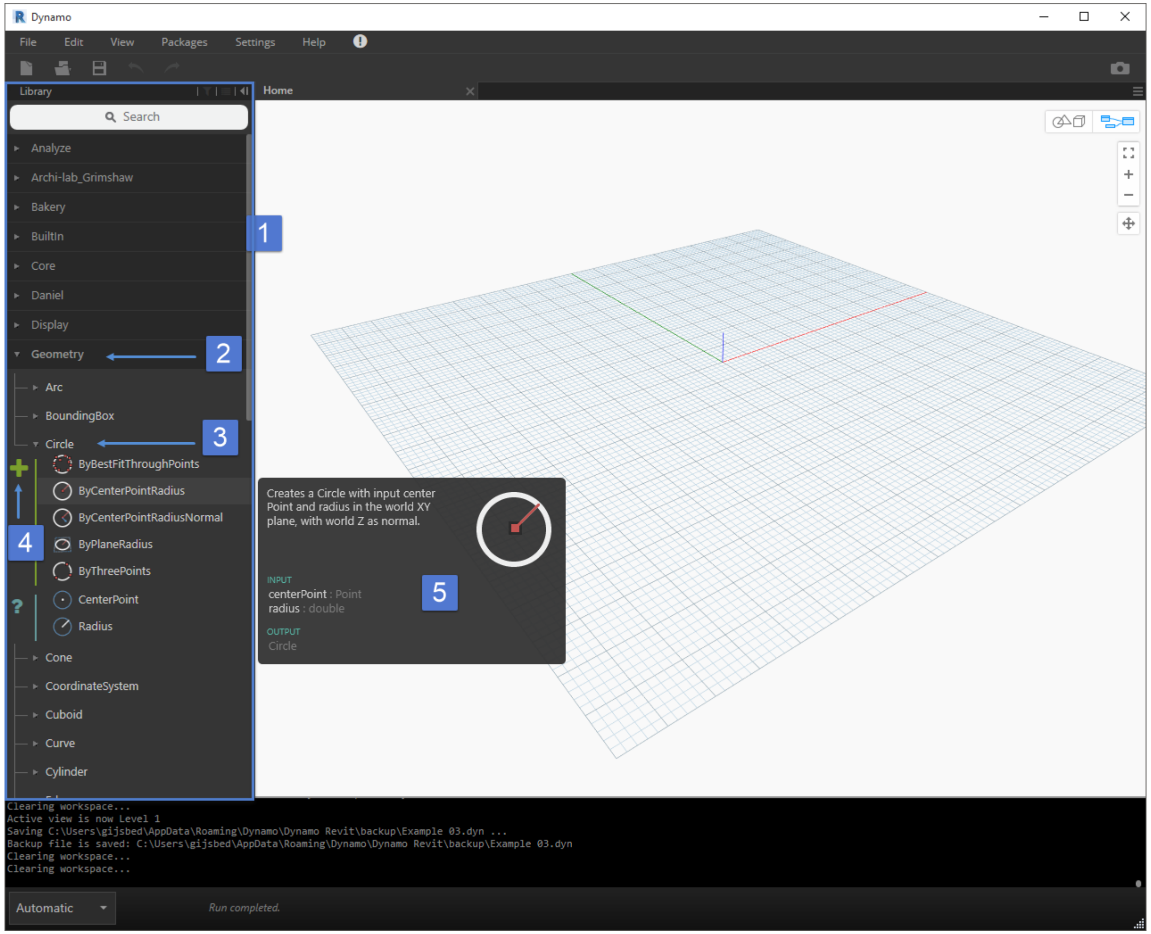
User Interface

Visual Programming



1. Menus
2. Toolbar
3. Library
4. Execution Bar
5. Work space

Visual Programming



Creates a Circle with input center Point and radius in the world XY plane, with world Z as normal.

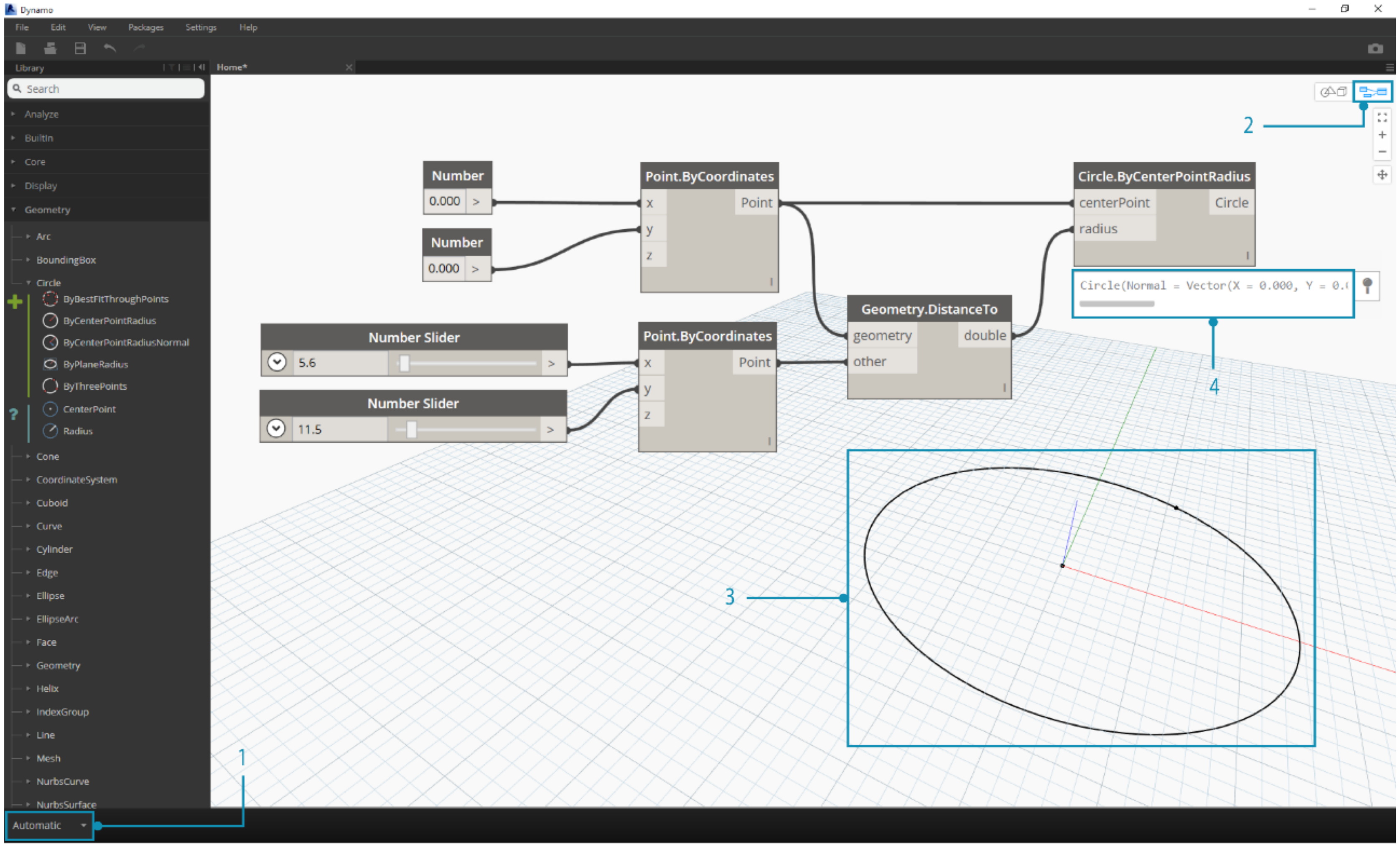
INPUT
centerPoint : Point
radius : double

OUTPUT
Circle

```
Clearing workspace...
Active view is now Level 1
Saving C:\Users\gijsbed\AppData\Roaming\Dynamo\Dynamo Revit\backup\Example 03.dyn ...
Backup file is saved: C:\Users\gijsbed\AppData\Roaming\Dynamo\Dynamo Revit\backup\Example 03.dyn
Clearing workspace...
Clearing workspace...
```

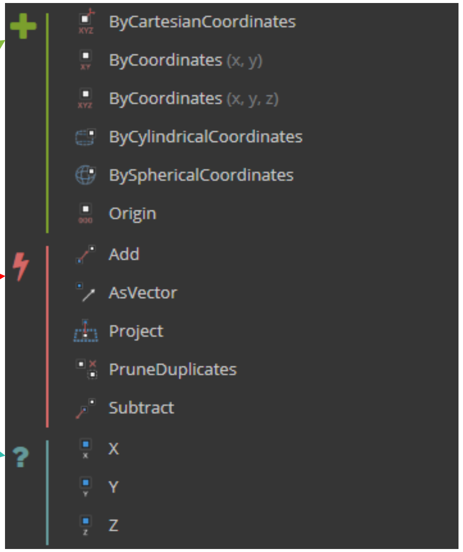
Run completed.

Visual Programming

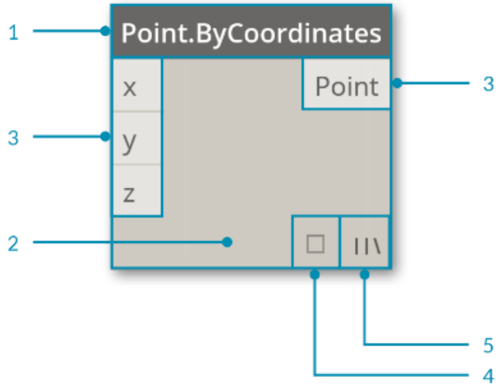


Nodes

- Each node is a **function** with Input and Output ports that take and create lists
- 3 behaviors:
  - Create (Constructor)
  - Action (Method)
  - Query (Property)
- Scripting integration
  - Design Script (Code block, .ds)
  - Python
  - …

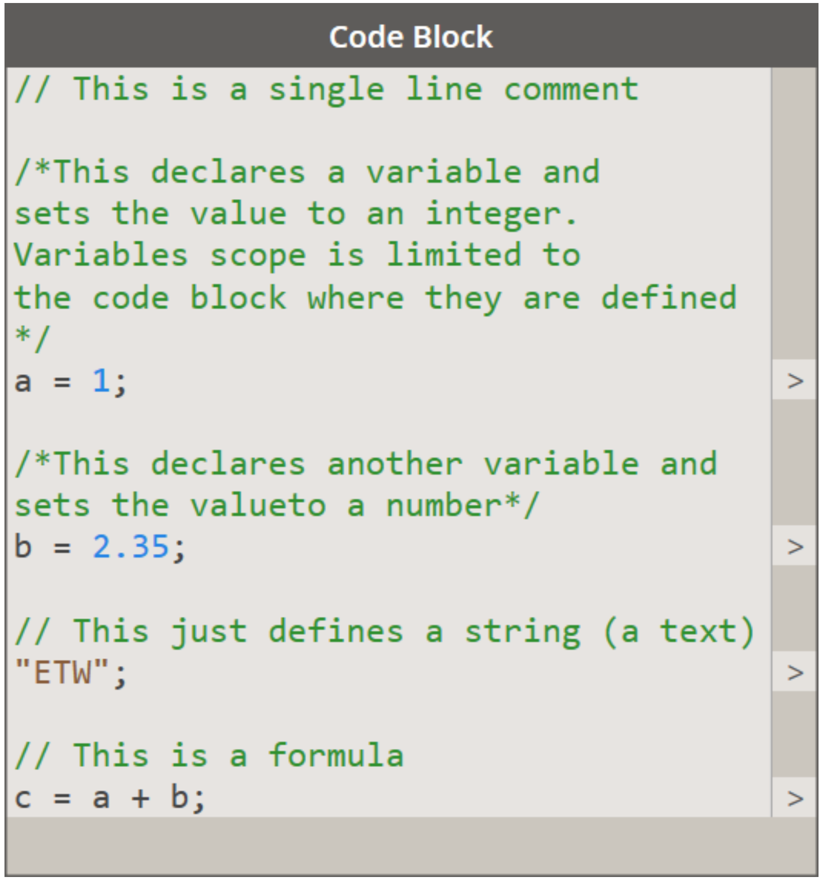


ByCartesianCoordinates
ByCoordinates (x, y)
ByCoordinates (x, y, z)
ByCylindricalCoordinates
BySphericalCoordinates
Origin
Add
AsVector
Project
PruneDuplicates
Subtract
X
Y
Z



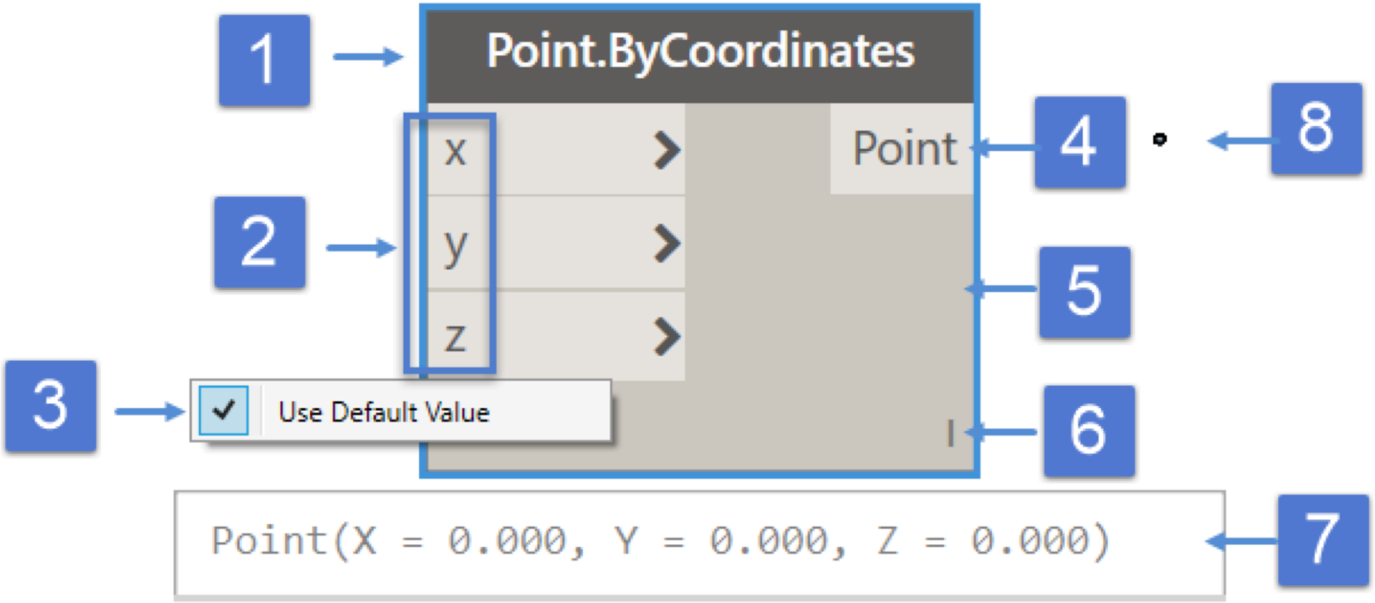
Code Blocks | Generic Purpose Nodes

- Double click on the canvas to create
- Name = Value;
- Case sensitive
- End with a semi-colon ";"



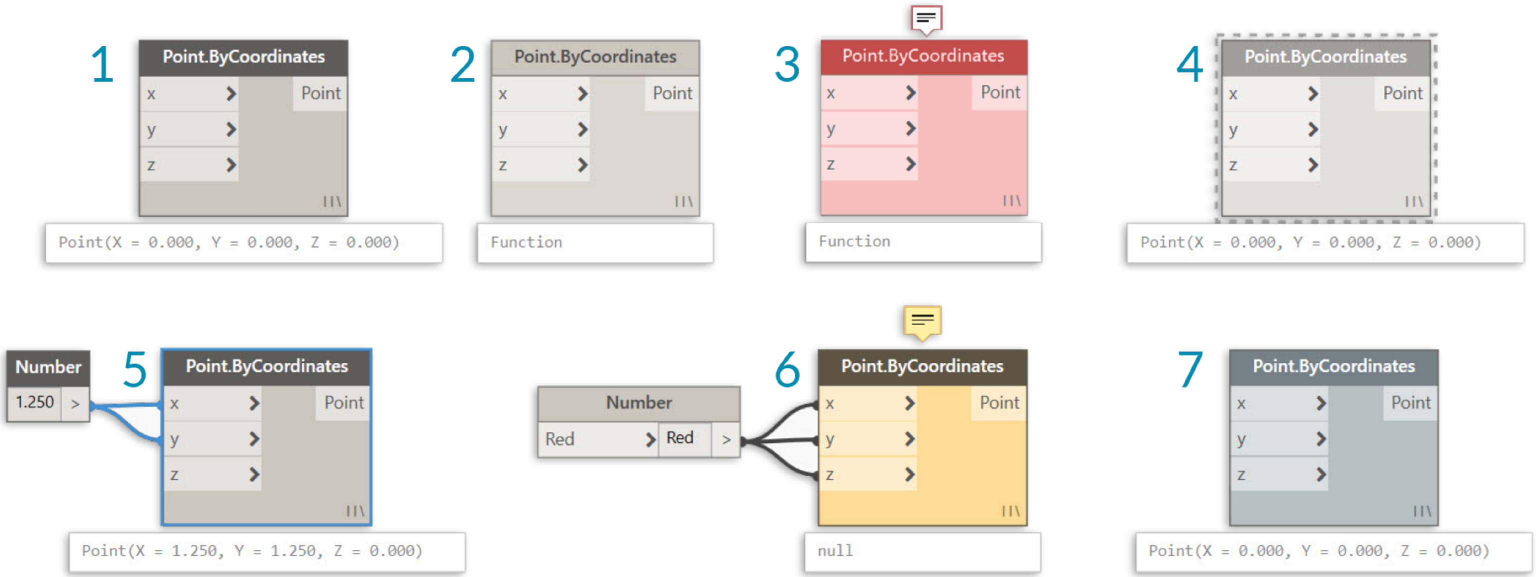
```
// This is a single line comment

/*This declares a variable and
sets the value to an integer.
Variables scope is limited to
the code block where they are defined
*/
a = 1;

/*This declares another variable and
sets the valueto a number*/
b = 2.35;

// This just defines a string (a text)
"ETW";

// This is a formula
c = a + b;
```

Anatomy of a Node



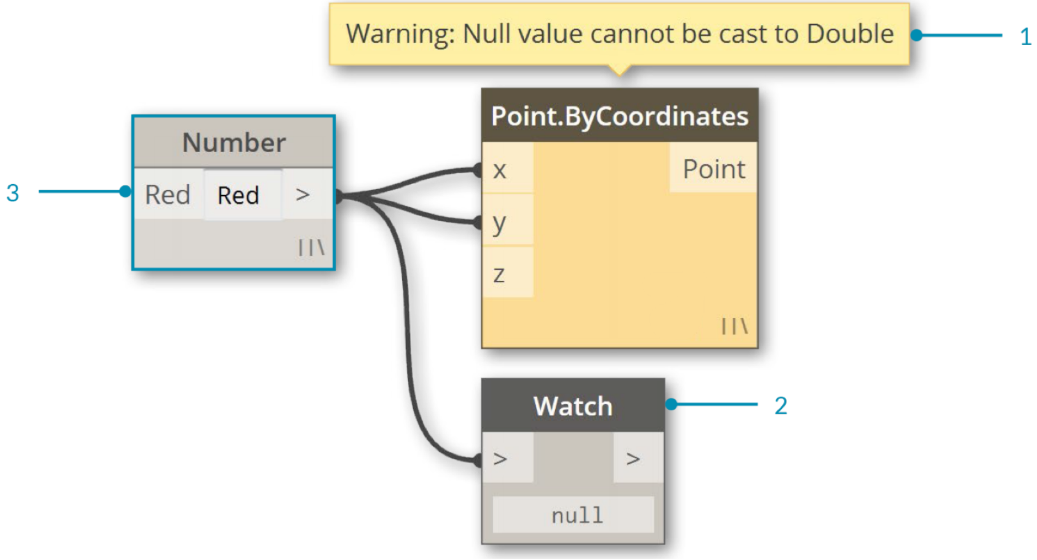
Point(X = 0.000, Y = 0.000, Z = 0.000)

Node States



```
Point(X = 0.000, Y = 0.000, Z = 0.000)
Function
Function
Point(X = 0.000, Y = 0.000, Z = 0.000)
Point(X = 1.250, Y = 1.250, Z = 0.000)
null
Point(X = 0.000, Y = 0.000, Z = 0.000)
```

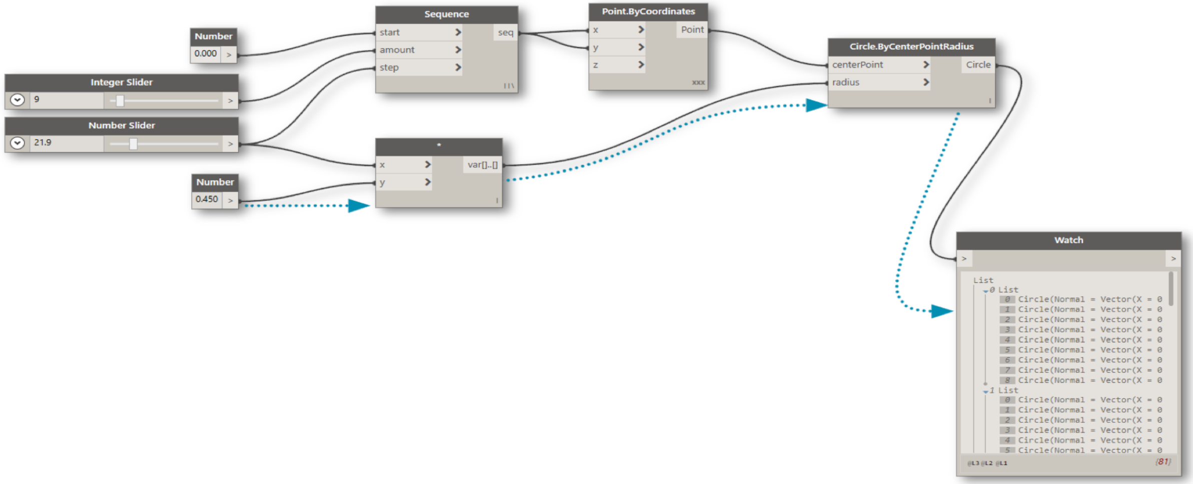
Warnings



Warning: Null value cannot be cast to Double

Connectors

- They define the routine flow of execution
- Always link an output port to an input port
- An output port can feed multiple input ports
- Selecting a node selects input and output connectors



```
List
  0 List
    0 Circle(Normal = Vector(X = 0
    1 Circle(Normal = Vector(X = 0
    2 Circle(Normal = Vector(X = 0
    3 Circle(Normal = Vector(X = 0
    4 Circle(Normal = Vector(X = 0
    5 Circle(Normal = Vector(X = 0
    6 Circle(Normal = Vector(X = 0
    7 Circle(Normal = Vector(X = 0
    8 Circle(Normal = Vector(X = 0
  1 List
    0 Circle(Normal = Vector(X = 0
    1 Circle(Normal = Vector(X = 0
    2 Circle(Normal = Vector(X = 0
    3 Circle(Normal = Vector(X = 0
    4 Circle(Normal = Vector(X = 0
    5 Circle(Normal = Vector(X = 0
@L3 @L2 @L1                      {81}
```

Graphs Management

Graph Management

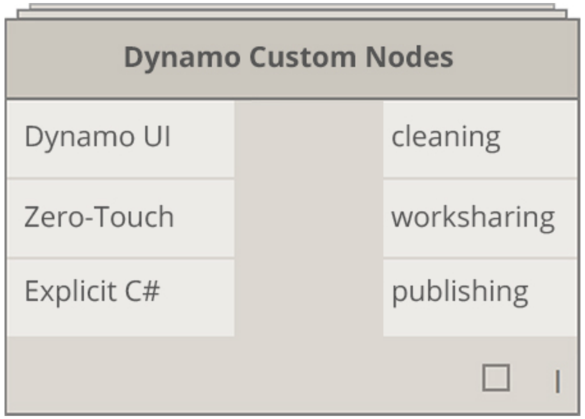
- Node Alignment
- Notes (Commenting)
- Grouping
- Color coding
  - Input
  - Function
  - Get
  - Set
  - Output
  - Debugging

Dynamo Graph Files

- DYN extension
- XML (up to 1.3.3) / JSON (since 2.0) structure
- Everyone can access the source code with a text editor
- There is still no way to compile and protect a Dynamo graph
- Future versions may be not backward compatible

Custom Nodes

- Different types:
  - Create from UI (DYF extension)
  - Zero Touch Essentials (C#, Standard Node Interface)
  - Design Script (.ds)
  - C# (Custom Interface)
- Benefits
  - Clean up definitions
  - Work sharing
  - Recursion
  - Quick adjustments
  - Expand the capabilities



Custom Nodes from UI

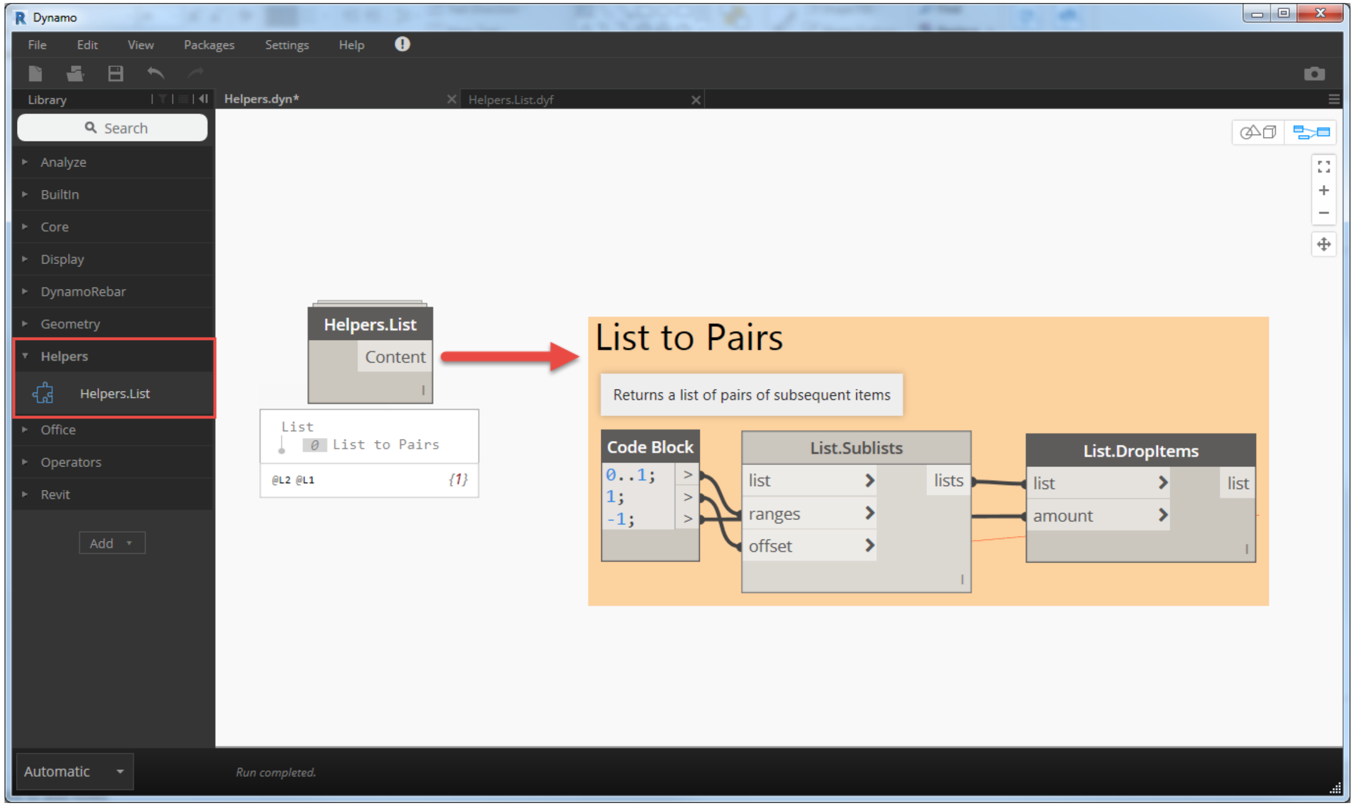
- The custom node is created locally on the machine
- Definition is not embedded in the graph
- Must distribute with the graph
- Easy to do, no coding experience required
- Make the graph more robust and easier to read
- Nodes can be grouped in custom shelves in the library

Package Manager

- Everyone can create a set of nodes and publish it
- There is a server that can be searched from within Dynamo or in alternative a website
- The packages are self-installing
- These are an important part of the quick success
- The packages are available for free and so there is no official support or maintenance for them

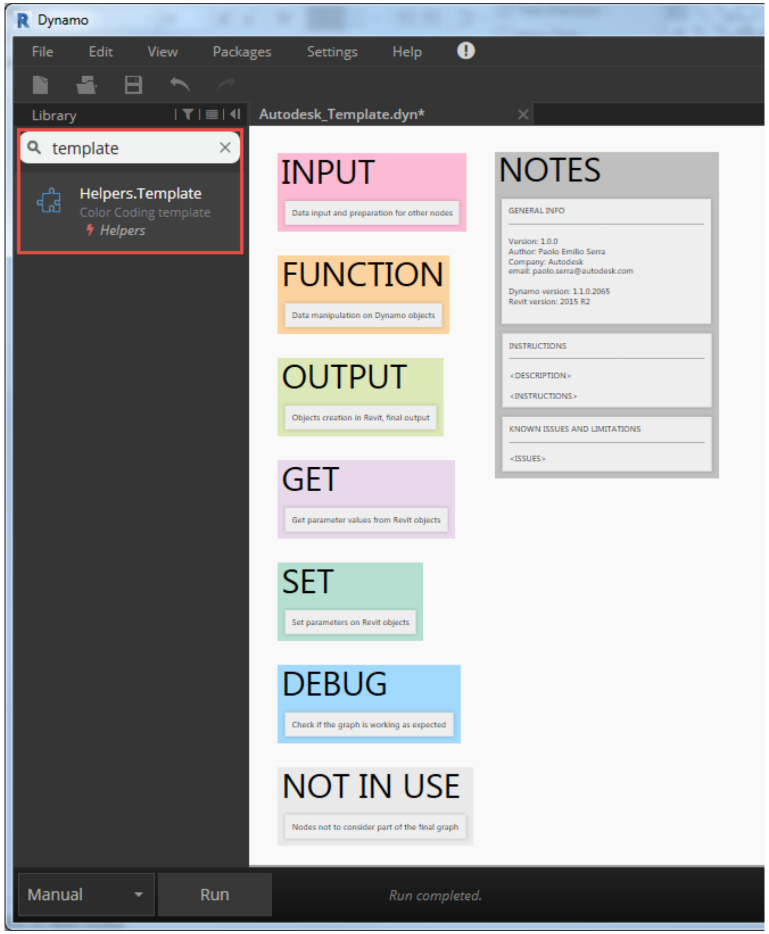
Custom Node Helpers

- Recall frequently used node structures
- Organize the nodes by custom criteria
- Copy from the Custom Node and Paste in the main graph
- Easier to maintain and search



Custom Node Helpers

- Enable searches in the Library
- Enforce Standards
- Reduce time to compose graphs
- Easier to maintain and update



INPUT

Data input and preparation for other nodes

FUNCTION

Data manipulation on Dynamo objects

OUTPUT

Objects creation in Revit, final output

GET

Get parameter values from Revit objects

SET

Set parameters on Revit objects

DEBUG

Check if the graph is working as expected

NOT IN USE

Nodes not to consider part of the final graph

NOTES

GENERAL INFO

Version: 1.0.0
Author: Paolo Emilio Serra
Company: Autodesk
email: paolo.serra@autodesk.com

Dynamo version: 1.1.0.2065
Revit version: 2015 R2

INSTRUCTIONS

<DESCRIPTION>

<INSTRUCTIONS>

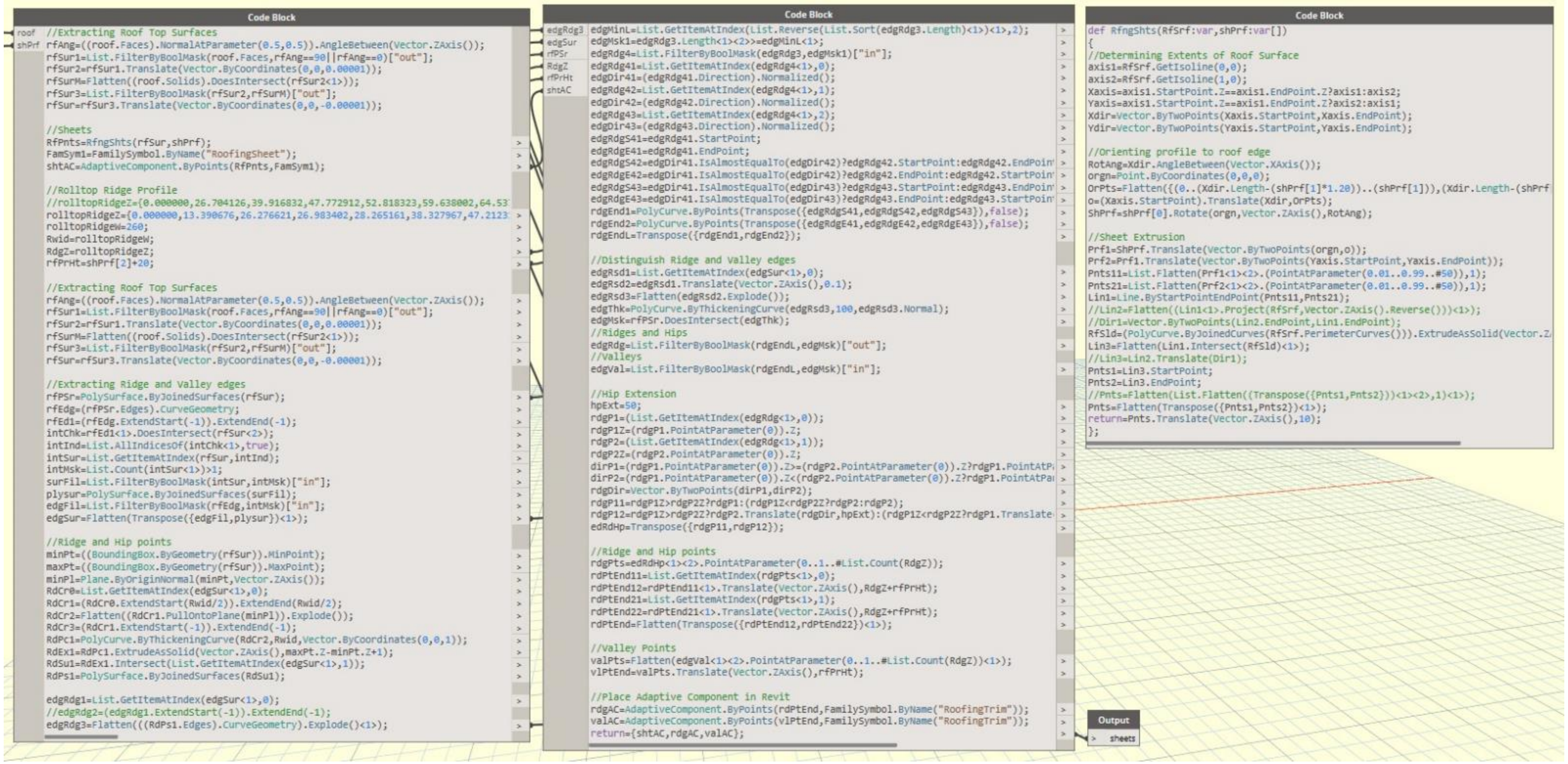
KNOWN ISSUES AND LIMITATIONS

<ISSUES>

Node to Code

- A feature that converts UI nodes into a single Code Block using Design Script
- Great way to learn Design Script syntax
- Clean up the graph
- Closer to standard coding
- New users tend to stay away from Scripts
- There is no “Code to Node” yet
- Performance wise there are no differences

Dynamo Script Example



```
//Extracting Roof Top Surfaces
rfAng=((roof.Faces).NormalAtParameter(0.5,0.5)).AngleBetween(Vector.ZAxis());
rfSur1=List.FilterByBoolMask(roof.Faces,rfAng==90||rfAng==0)["out"];
rfSur2=rfSur1.Translate(Vector.ByCoordinates(0,0,0.00001));
rfSurM=Flatten((roof.Solids).DoesIntersect(rfSur2<1>));
rfSur3=List.FilterByBoolMask(rfSur2,rfSurM)["out"];
rfSur=rfSur3.Translate(Vector.ByCoordinates(0,0,-0.00001));

//Sheets
RfPnts=RfngShts(rfSur,shPrf);
FamSym1=FamilySymbol.ByName("RoofingSheet");
shtAC=AdaptiveComponent.ByPoints(RfPnts,FamSym1);

//Rolltop Ridge Profile
//rolltopRidgeZ={0.000000,26.704126,39.916832,47.772912,52.818323,59.638002,64.53
rolltopRidgeZ={0.000000,13.390676,26.276621,26.983402,28.265161,38.327967,47.2123
rolltopRidgeW=260;
Rwid=rolltopRidgeW;
RdgZ=rolltopRidgeZ;
rfPrHt=shPrf[2]+20;

//Extracting Roof Top Surfaces
rfAng=((roof.Faces).NormalAtParameter(0.5,0.5)).AngleBetween(Vector.ZAxis());
rfSur1=List.FilterByBoolMask(roof.Faces,rfAng==90||rfAng==0)["out"];
rfSur2=rfSur1.Translate(Vector.ByCoordinates(0,0,0.00001));
rfSurM=Flatten((roof.Solids).DoesIntersect(rfSur2<1>));
rfSur3=List.FilterByBoolMask(rfSur2,rfSurM)["out"];
rfSur=rfSur3.Translate(Vector.ByCoordinates(0,0,-0.00001));

//Extracting Ridge and Valley edges
rfPSr=PolySurface.ByJoinedSurfaces(rfSur);
rfEdg=(rfPSr.Edges).CurveGeometry;
rfEd1=(rfEdg.ExtendStart(-1)).ExtendEnd(-1);
intChk=rfEd1<1>.DoesIntersect(rfSur<2>);
intInd=List.AllIndicesOf(intChk<1>,true);
intSur=List.GetItemAtIndex(rfSur,intInd);
intMsk=List.Count(intSur<1>)>1;
surFil=List.FilterByBoolMask(intSur,intMsk)["in"];
plysur=PolySurface.ByJoinedSurfaces(surFil);
edgFil=List.FilterByBoolMask(rfEdg,intMsk)["in"];
edgSur=Flatten(Transpose({edgFil,plysur})<1>);

//Ridge and Hip points
minPt=((BoundingBox.ByGeometry(rfSur)).MinPoint);
maxPt=((BoundingBox.ByGeometry(rfSur)).MaxPoint);
minPl=Plane.ByOriginNormal(minPt,Vector.ZAxis());
RdCr0=List.GetItemAtIndex(edgSur<1>,0);
RdCr1=(RdCr0.ExtendStart(Rwid/2)).ExtendEnd(Rwid/2);
RdCr2=Flatten((RdCr1.PullOntoPlane(minPl)).Explode());
RdCr3=(RdCr1.ExtendStart(-1)).ExtendEnd(-1);
RdPc1=PolyCurve.ByThickeningCurve(RdCr2,Rwid,Vector.ByCoordinates(0,0,1));
RdEx1=RdPc1.ExtrudeAsSolid(Vector.ZAxis(),maxPt.Z-minPt.Z+1);
RdSu1=RdEx1.Intersect(List.GetItemAtIndex(edgSur<1>,1));
RdPs1=PolySurface.ByJoinedSurfaces(RdSu1);

edgRdg1=List.GetItemAtIndex(edgSur<1>,0);
//edgRdg2=(edgRdg1.ExtendStart(-1)).ExtendEnd(-1);
edgRdg3=Flatten(((RdPs1.Edges).CurveGeometry).Explode()<1>);
```

```
edgMinL=List.GetItemAtIndex(List.Reverse(List.Sort(edgRdg3.Length)<1>)<1>,2);
edgMsk1=edgRdg3.Length<1><2>>=edgMinL<1>;
edgRdg4=List.FilterByBoolMask(edgRdg3,edgMsk1)["in"];
edgRdg41=List.GetItemAtIndex(edgRdg4<1>,0);
edgDir41=(edgRdg41.Direction).Normalized();
edgRdg42=List.GetItemAtIndex(edgRdg4<1>,1);
edgDir42=(edgRdg42.Direction).Normalized();
edgRdg43=List.GetItemAtIndex(edgRdg4<1>,2);
edgDir43=(edgRdg43.Direction).Normalized();
edgRdgS41=edgRdg41.StartPoint;
edgRdgE41=edgRdg41.EndPoint;
edgRdgS42=edgDir41.IsAlmostEqualTo(edgDir42)?edgRdg42.StartPoint:edgRdg42.EndPoin
edgRdgE42=edgDir41.IsAlmostEqualTo(edgDir42)?edgRdg42.EndPoint:edgRdg42.StartPoin
edgRdgS43=edgDir41.IsAlmostEqualTo(edgDir43)?edgRdg43.StartPoint:edgRdg43.EndPoin
edgRdgE43=edgDir41.IsAlmostEqualTo(edgDir43)?edgRdg43.EndPoint:edgRdg43.StartPoin
rdgEnd1=PolyCurve.ByPoints(Transpose({edgRdgS41,edgRdgS42,edgRdgS43}),false);
rdgEnd2=PolyCurve.ByPoints(Transpose({edgRdgE41,edgRdgE42,edgRdgE43}),false);
rdgEndL=Transpose({rdgEnd1,rdgEnd2});

//Distinguish Ridge and Valley edges
edgRsd1=List.GetItemAtIndex(edgSur<1>,0);
edgRsd2=edgRsd1.Translate(Vector.ZAxis(),0.1);
edgRsd3=Flatten(edgRsd2.Explode());
edgThk=PolyCurve.ByThickeningCurve(edgRsd3,100,edgRsd3.Normal);
edgMsk=rfPSr.DoesIntersect(edgThk);
//Ridges and Hips
edgRdg=List.FilterByBoolMask(rdgEndL,edgMsk)["out"];
//Valleys
edgVal=List.FilterByBoolMask(rdgEndL,edgMsk)["in"];

//Hip Extension
hpExt=50;
rdgP1=(List.GetItemAtIndex(edgRdg<1>,0));
rdgP1Z=(rdgP1.PointAtParameter(0)).Z;
rdgP2=(List.GetItemAtIndex(edgRdg<1>,1));
rdgP2Z=(rdgP2.PointAtParameter(0)).Z;
dirP1=(rdgP1.PointAtParameter(0)).Z>=(rdgP2.PointAtParameter(0)).Z?rdgP1.PointAtP
dirP2=(rdgP1.PointAtParameter(0)).Z<(rdgP2.PointAtParameter(0)).Z?rdgP1.PointAtPa
rdgDir=Vector.ByTwoPoints(dirP1,dirP2);
rdgP11=rdgP1Z>rdgP2Z?rdgP1:(rdgP1Z<rdgP2Z?rdgP2:rdgP2);
rdgP12=rdgP1Z>rdgP2Z?rdgP2.Translate(rdgDir,hpExt):(rdgP1Z<rdgP2Z?rdgP1.Translate
edRdHp=Transpose({rdgP11,rdgP12});

//Ridge and Hip points
rdgPts=edRdHp<1><2>.PointAtParameter(0..1..#List.Count(RdgZ));
rdPtEnd11=List.GetItemAtIndex(rdgPts<1>,0);
rdPtEnd12=rdPtEnd11<1>.Translate(Vector.ZAxis(),RdgZ+rfPrHt);
rdPtEnd21=List.GetItemAtIndex(rdgPts<1>,1);
rdPtEnd22=rdPtEnd21<1>.Translate(Vector.ZAxis(),RdgZ+rfPrHt);
rdPtEnd=Flatten(Transpose({rdPtEnd12,rdPtEnd22})<1>);

//Valley Points
valPts=Flatten(edgVal<1><2>.PointAtParameter(0..1..#List.Count(RdgZ))<1>);
vlPtEnd=valPts.Translate(Vector.ZAxis(),rfPrHt);

//Place Adaptive Component in Revit
rdgAC=AdaptiveComponent.ByPoints(rdPtEnd,FamilySymbol.ByName("RoofingTrim"));
valAC=AdaptiveComponent.ByPoints(vlPtEnd,FamilySymbol.ByName("RoofingTrim"));
return={shtAC,rdgAC,valAC};
```

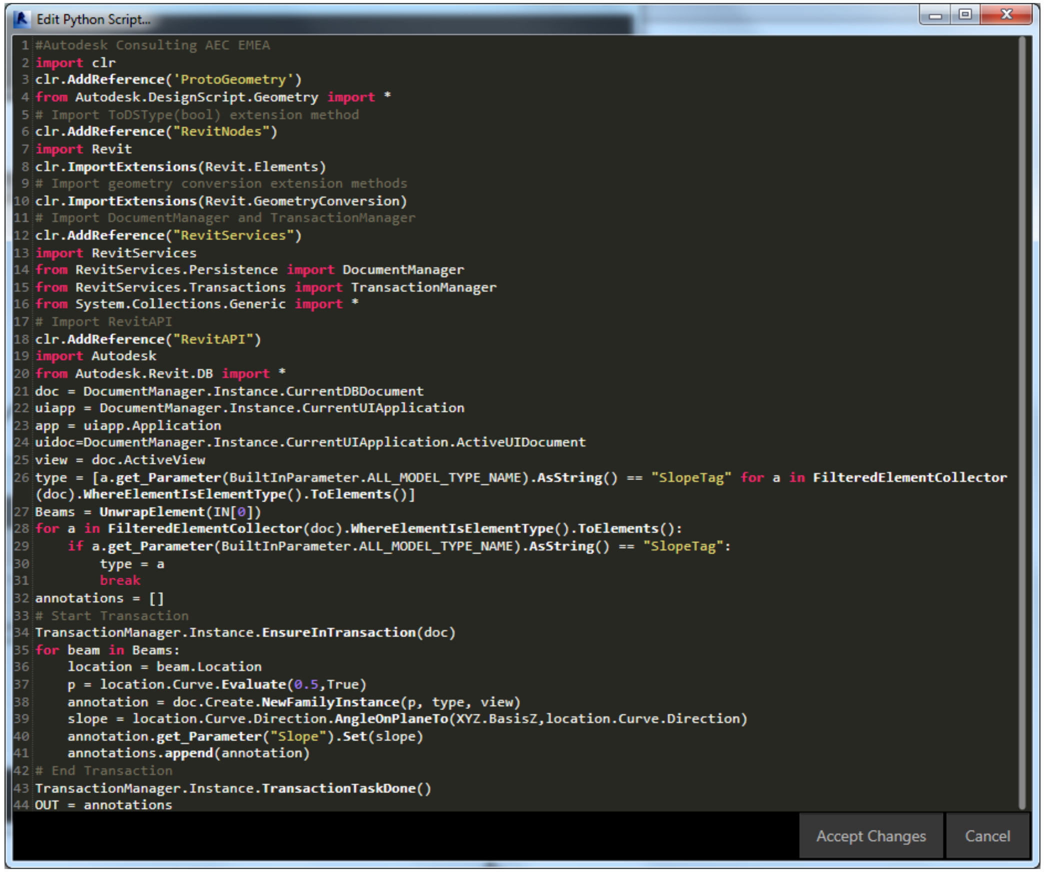
```
def RfngShts(RfSrf:var,shPrf:var[])
{
//Determining Extents of Roof Surface
axis1=RfSrf.GetIsoline(0,0);
axis2=RfSrf.GetIsoline(1,0);
Xaxis=axis1.StartPoint.Z==axis1.EndPoint.Z?axis1:axis2;
Yaxis=axis1.StartPoint.Z==axis1.EndPoint.Z?axis2:axis1;
Xdir=Vector.ByTwoPoints(Xaxis.StartPoint,Xaxis.EndPoint);
Ydir=Vector.ByTwoPoints(Yaxis.StartPoint,Yaxis.EndPoint);

//Orienting profile to roof edge
RotAng=Xdir.AngleBetween(Vector.XAxis());
orgn=Point.ByCoordinates(0,0,0);
OrPts=Flatten({(0..(Xdir.Length-(shPrf[1]*1.20))..(shPrf[1])),(Xdir.Length-(shPrf
o=(Xaxis.StartPoint).Translate(Xdir,OrPts);
ShPrf=shPrf[0].Rotate(orgn,Vector.ZAxis(),RotAng);

//Sheet Extrusion
Prf1=ShPrf.Translate(Vector.ByTwoPoints(orgn,o));
Prf2=Prf1.Translate(Vector.ByTwoPoints(Yaxis.StartPoint,Yaxis.EndPoint));
Pnts11=List.Flatten(Prf1<1><2>.(PointAtParameter(0.01..0.99..#50)),1);
Pnts21=List.Flatten(Prf2<1><2>.(PointAtParameter(0.01..0.99..#50)),1);
Lin1=Line.ByStartPointEndPoint(Pnts11,Pnts21);
//Lin2=Flatten((Lin1<1>.Project(RfSrf,Vector.ZAxis().Reverse()))<1>);
//Dir1=Vector.ByTwoPoints(Lin2.EndPoint,Lin1.EndPoint);
RfSld=(PolyCurve.ByJoinedCurves(RfSrf.PerimeterCurves())).ExtrudeAsSolid(Vector.Z
Lin3=Flatten(Lin1.Intersect(RfSld)<1>);
//Lin3=Lin2.Translate(Dir1);
Pnts1=Lin3.StartPoint;
Pnts2=Lin3.EndPoint;
//Pnts=Flatten(List.Flatten((Transpose({Pnts1,Pnts2}))<1><2>,1)<1>);
Pnts=Flatten(Transpose({Pnts1,Pnts2})<1>);
return=Pnts.Translate(Vector.ZAxis(),10);
};
```

Dynamo Script

- Associative / Imperative language blocks
- Replication guides (extends the lacing concept)
- Conditional statements
- Loops (while / for)
- Creation of geometry entities (nurbs curves, nurbs surfaces, loft, revolves, etc.)
- Manipulation of geometry entities (Boolean operations, intersection, trims)
- Creation of Custom Nodes

Iron Python 2.7

- Embedded IDE
- Very simple / no support
- .NET compatible
- References for Revit API
    - Revit.Services
    - Revit.Persistance
    - RevitNodes
    - Geometry Conversion



```python
#Autodesk Consulting AEC EMEA
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
# Import ToDSType(bool) extension method
clr.AddReference("RevitNodes")
import Revit
clr.ImportExtensions(Revit.Elements)
# Import geometry conversion extension methods
clr.ImportExtensions(Revit.GeometryConversion)
# Import DocumentManager and TransactionManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager
from System.Collections.Generic import *
# Import RevitAPI
clr.AddReference("RevitAPI")
import Autodesk
from Autodesk.Revit.DB import *
doc = DocumentManager.Instance.CurrentDBDocument
uiapp = DocumentManager.Instance.CurrentUIApplication
app = uiapp.Application
uidoc=DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument
view = doc.ActiveView
type = [a.get_Parameter(BuiltInParameter.ALL_MODEL_TYPE_NAME).AsString() == "SlopeTag" for a in FilteredElementCollector(doc).WhereElementIsElementType().ToElements()]
Beams = UnwrapElement(IN[0])
for a in FilteredElementCollector(doc).WhereElementIsElementType().ToElements():
    if a.get_Parameter(BuiltInParameter.ALL_MODEL_TYPE_NAME).AsString() == "SlopeTag":
        type = a
        break
annotations = []
# Start Transaction
TransactionManager.Instance.EnsureInTransaction(doc)
for beam in Beams:
    location = beam.Location
    p = location.Curve.Evaluate(0.5,True)
    annotation = doc.Create.NewFamilyInstance(p, type, view)
    slope = location.Curve.Direction.AngleOnPlaneTo(XYZ.BasisZ,location.Curve.Direction)
    annotation.get_Parameter("Slope").Set(slope)
    annotations.append(annotation)
# End Transaction
TransactionManager.Instance.TransactionTaskDone()
OUT = annotations
```

Backup

- Introduced in 0.8.2
- Creates a Backups folder in which stores the previous versions of the Dynamo graphs at the same path
- Accessible from the Home view
- It is possible to set the interval between two backups

Autodesk Standards

Coding Standards | Naming Convention

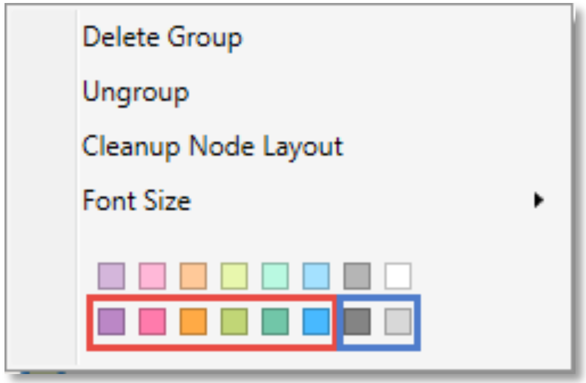
- Reflect Repository Structure
- Explicit binding with Revit models
- Enable Searches
- Drive sorting for Dynamo Player sequences

- Use Notes to add detail to the graph
- Rename key nodes in the graph appending a description

<Name> | <Description>

Coding Standards | Color Coding

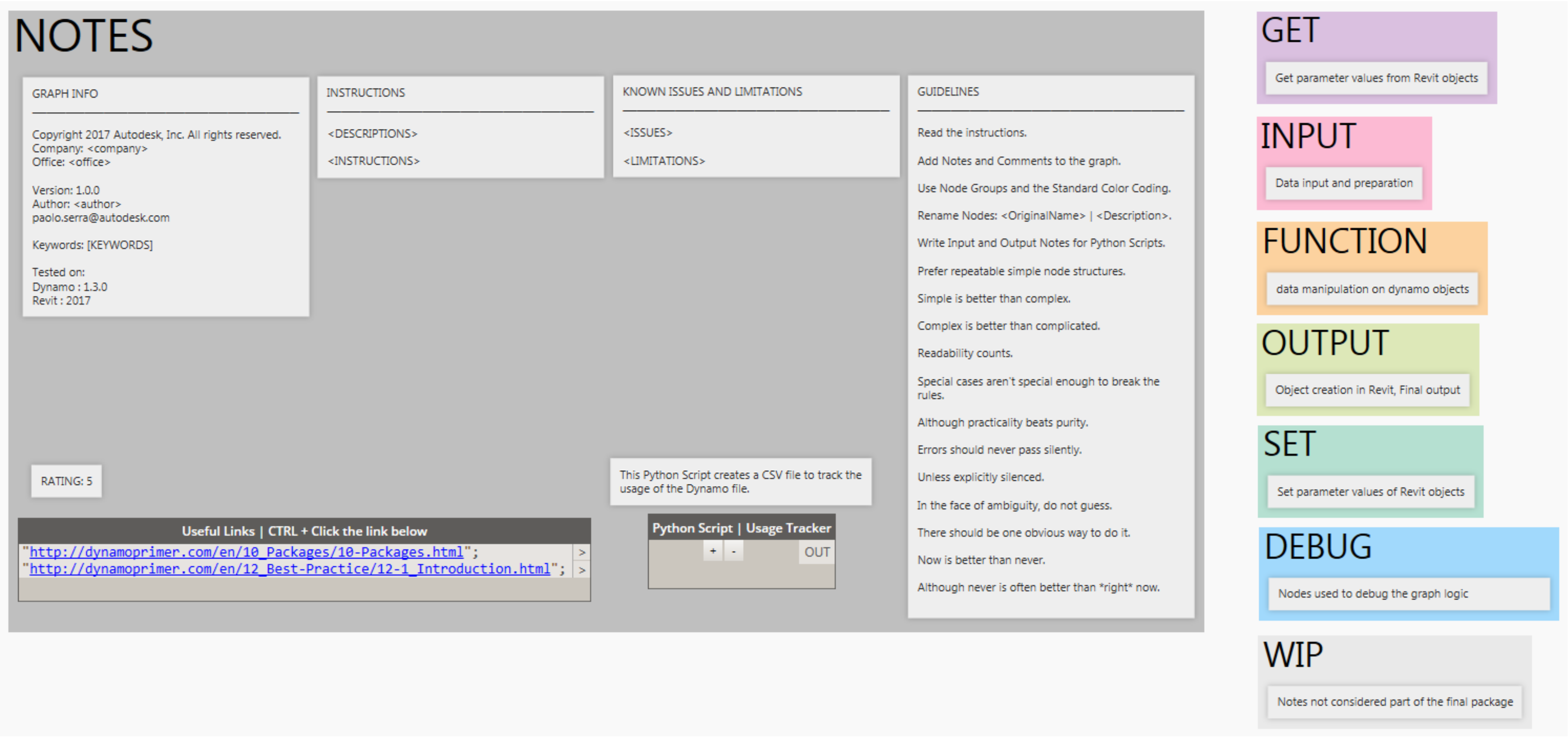
- Adopt existing standards (i.e. Autodesk)
- Provide guidance to create graphs and improve readability
- Development and maintenance can be picked up by someone else following the same rules



Coding Standards | Templates

- Graph General Info and search keywords
- Instructions
- Known Issues and Limitations
- Notes for Input and Output data structure and variable type for scripts
- Python and DesignScript templates (i.e. Notepad++)
- Clean Node Layout

Coding Standards | Autodesk Template



NOTES

GRAPH INFO

Copyright 2017 Autodesk, Inc. All rights reserved.
Company: <company>
Office: <office>

Version: 1.0.0
Author: <author>
paolo.serra@autodesk.com

Keywords: [KEYWORDS]

Tested on:
Dynamo : 1.3.0
Revit : 2017

INSTRUCTIONS

<DESCRIPTIONS>

<INSTRUCTIONS>

KNOWN ISSUES AND LIMITATIONS

<ISSUES>

<LIMITATIONS>

GUIDELINES

Read the instructions.

Add Notes and Comments to the graph.

Use Node Groups and the Standard Color Coding.

Rename Nodes: <OriginalName> | <Description>.

Write Input and Output Notes for Python Scripts.

Prefer repeatable simple node structures.

Simple is better than complex.

Complex is better than complicated.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, do not guess.

There should be one obvious way to do it.

Now is better than never.

Although never is often better than *right* now.

RATING: 5

This Python Script creates a CSV file to track the usage of the Dynamo file.

Useful Links | CTRL + Click the link below

```
"http://dynamoprimer.com/en/10_Packages/10-Packages.html";
"http://dynamoprimer.com/en/12_Best-Practice/12-1_Introduction.html";
```

Python Script | Usage Tracker

GET

Get parameter values from Revit objects

INPUT

Data input and preparation

FUNCTION

data manipulation on dynamo objects

OUTPUT

Object creation in Revit, Final output

SET

Set parameter values of Revit objects

DEBUG

Nodes used to debug the graph logic

WIP

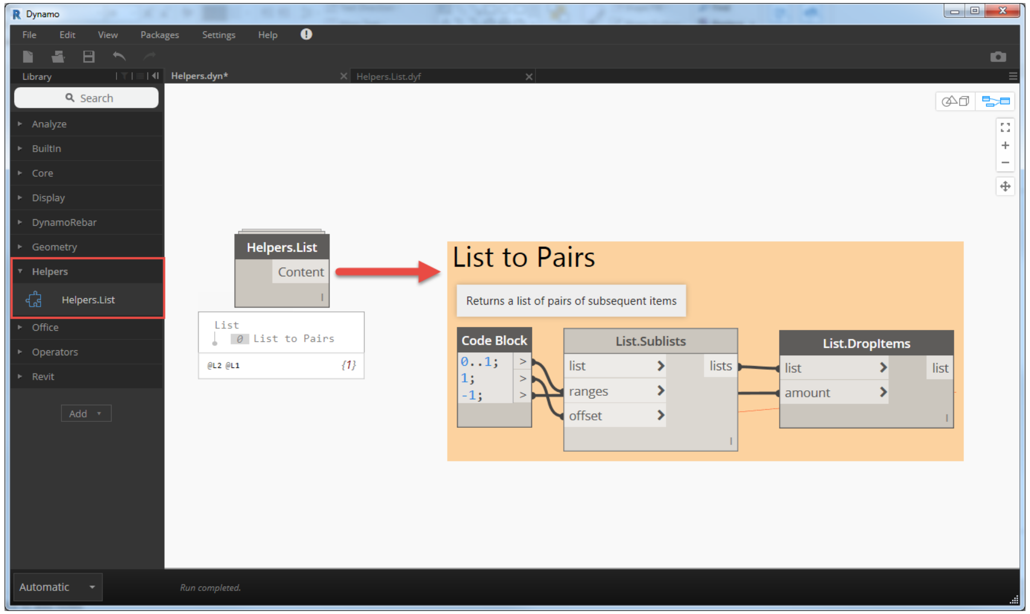
Notes not considered part of the final package

Coding Standards | Custom Nodes

- Avoid custom nodes in the graphs to be shared

- Use Helpers to collect and recall frequently used node structures

- Organize the nodes in categories

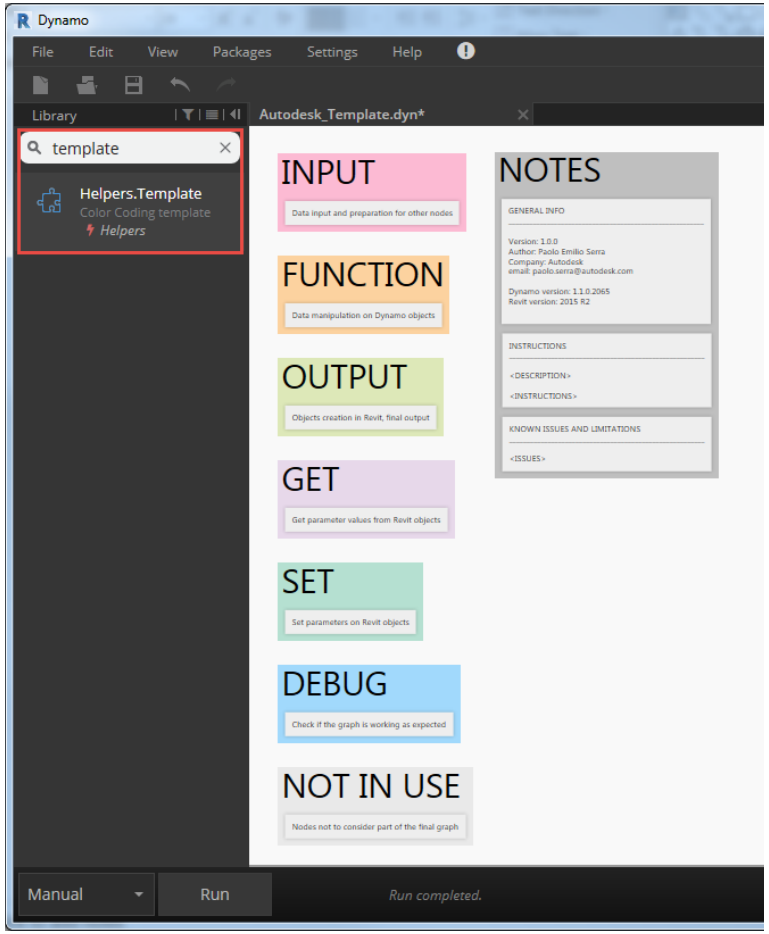


Returns a list of pairs of subsequent items

```
0..1;
1;
-1;
```

Coding Standards | Helpers

- Enable searches in the Library
- Enforce Standards
- Reduce time to compose graphs
- Easier to maintain and update



INPUT

Data input and preparation for other nodes

FUNCTION

Data manipulation on Dynamo objects

OUTPUT

Objects creation in Revit, final output

GET

Get parameter values from Revit objects

SET

Set parameters on Revit objects

DEBUG

Check if the graph is working as expected

NOT IN USE

Nodes not to consider part of the final graph

NOTES

GENERAL INFO

Version: 1.0.0
Author: Paolo Emilio Serra
Company: Autodesk
email: paolo.serra@autodesk.com

Dynamo version: 1.1.0.2065
Revit version: 2015 R2

INSTRUCTIONS

<DESCRIPTION>

<INSTRUCTIONS>

KNOWN ISSUES AND LIMITATIONS

<ISSUES>

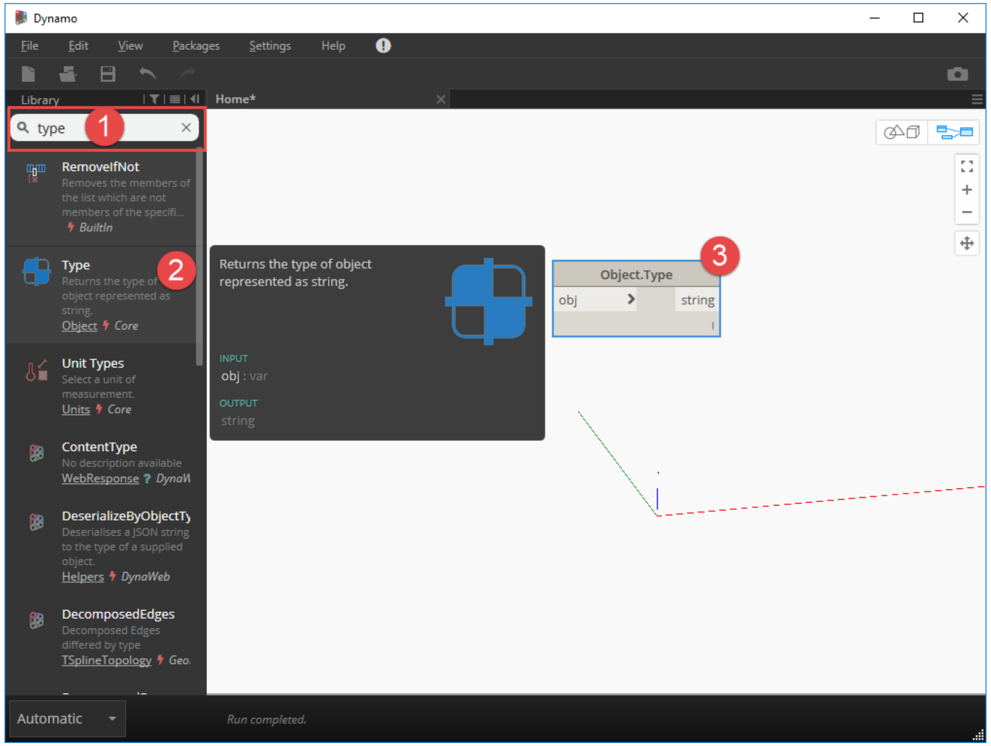
Coding Standards | Packages Policy

- Avoid relying on external packages
- Create package to control source and distribute approved functionalities
- Use Node To Code feature before deployment to reduce the risk of connectors being moved
- Restrict writing rights to Dynamo User Group only
- Enforce mirroring on local machines overnight
- Zero Touch is another option to preserve IP (advanced)

Visual Programming

Variable Types

- Integers
- Doubles (with a decimal part)
- Strings (texts in between speech marks)
- Boolean (true or false)
- Lists (indexed zero-based collections of items)
- Dictionaries (in Dynamo 2.0)
- Objects



RemoveIfNot: Removes the members of the list which are not members of the specifi...

Type: Returns the type of object represented as string.

Unit Types: Select a unit of measurement.

ContentType: No description available

DeserializeByObjectTy: Deserialises a JSON string to the type of a supplied object.

DecomposedEdges: Decomposed Edges differed by type

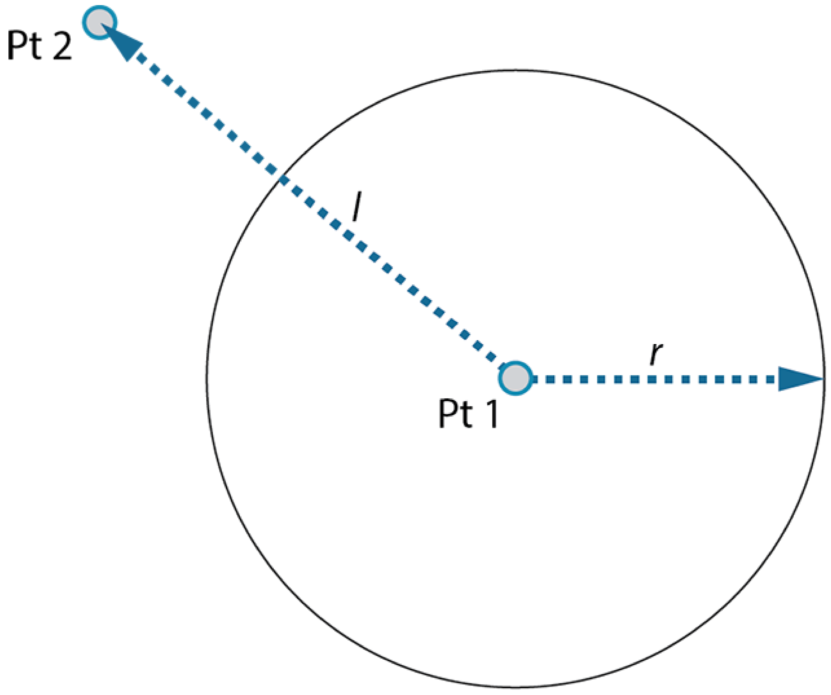
Returns the type of object represented as string.

INPUT
obj : var

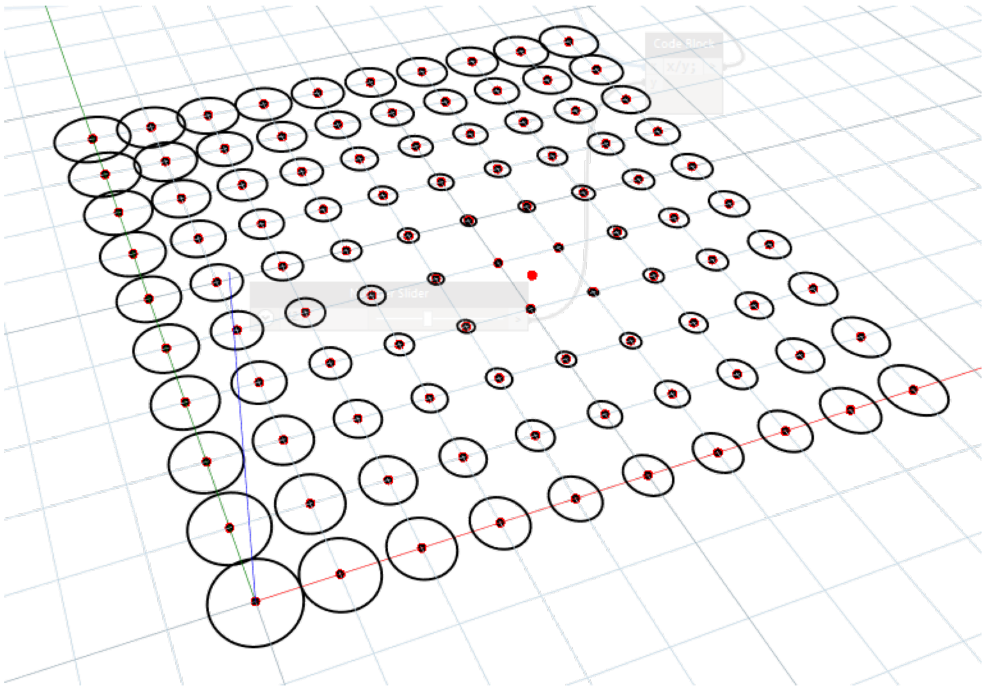
OUTPUT
string

Run completed.

Defining Objectives & Relationships

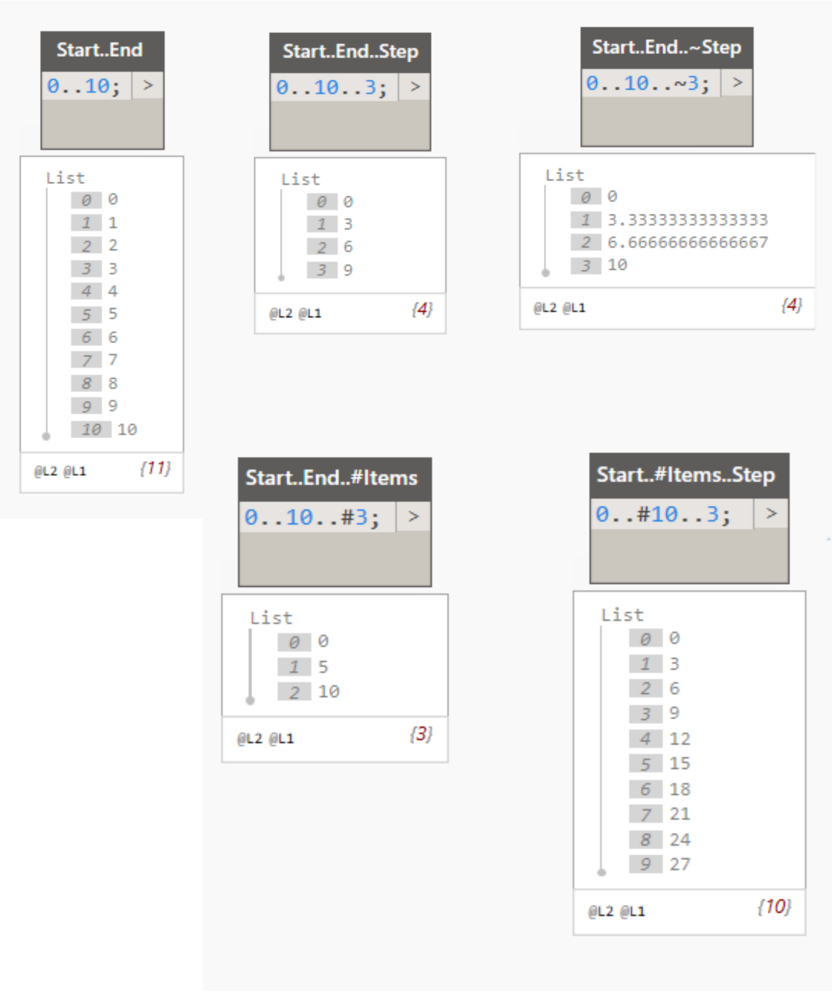


Length (*l*) ∝ Radius (*r*)



Number Ranges

- **Start..End;** a bounded range of numbers (ending value is included)
- **Start..End..Step;** a bounded range of number with step (the ending value may be not included in the output)
- **Start..End..~Step;** a bounded range with an approximate step (values are evenly distributed, ending value is included)
- **Start..End..#Items;** a bounded range of a given nr. Of items
- **Start..#Items..Step;** a sequence with step and nr. Of Items



Start..End

```
0..10;
```

```
List
  0 0
  1 1
  2 2
  3 3
  4 4
  5 5
  6 6
  7 7
  8 8
  9 9
  10 10
@L2 @L1          {11}
```

Start..End..Step

```
0..10..3;
```

```
List
  0 0
  1 3
  2 6
  3 9
@L2 @L1          {4}
```

Start..End..~Step

```
0..10..~3;
```

```
List
  0 0
  1 3.33333333333333
  2 6.66666666666667
  3 10
@L2 @L1          {4}
```

Start..End..#Items

```
0..10..#3;
```

```
List
  0 0
  1 5
  2 10
@L2 @L1          {3}
```

Start..#Items..Step

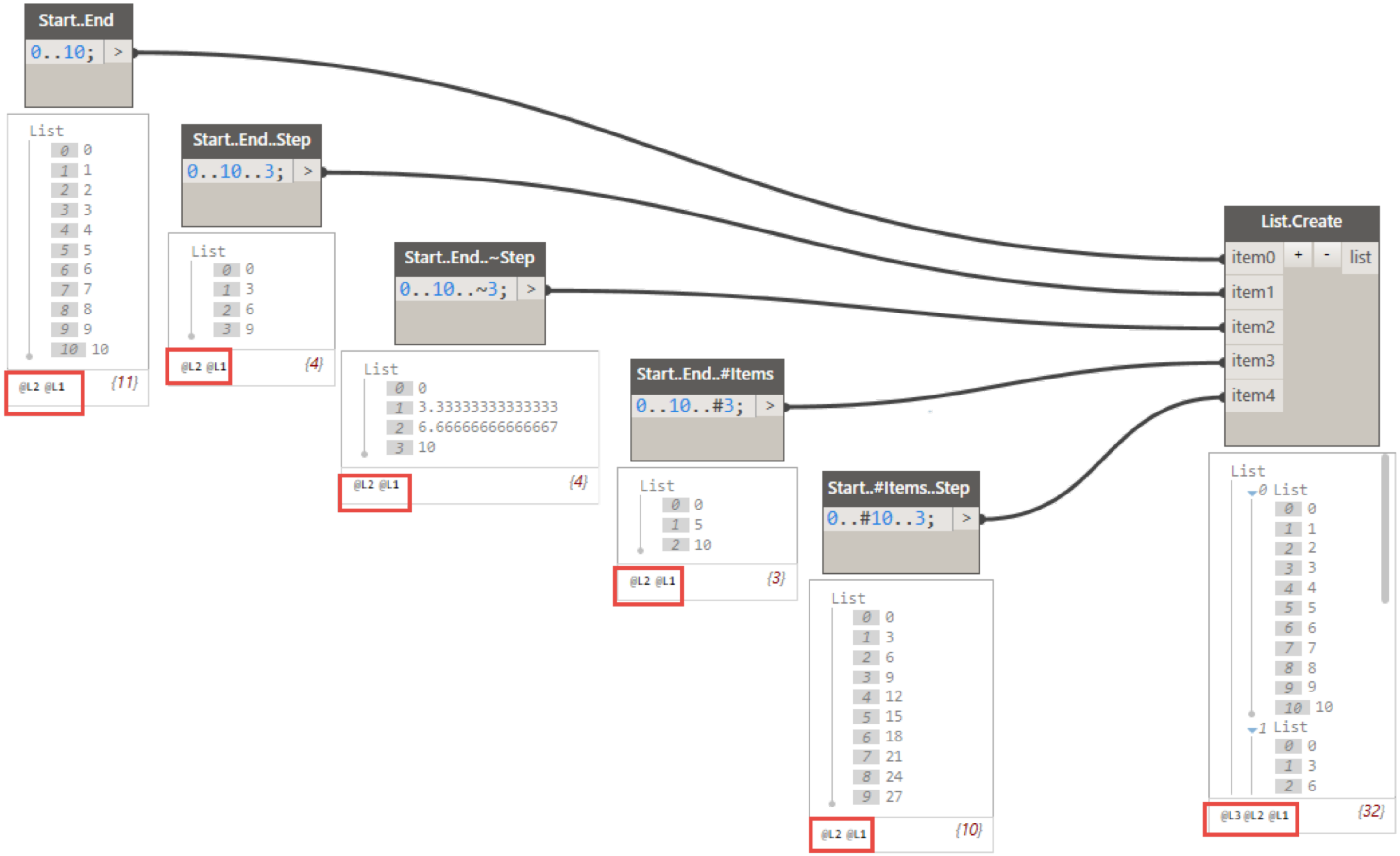
```
0..#10..3;
```

```
List
  0 0
  1 3
  2 6
  3 9
  4 12
  5 15
  6 18
  7 21
  8 24
  9 27
@L2 @L1          {10}
```

Lists

- **{ a, b, c};** declares a list in Code Blocks, items separated by commas “,”
- **List[n];** returns the element in the list at the index n, if n is negative the count starts from the end of the list
- **List[n][m];** returns the element in a sub-list at the index m of list n, if n is negative the count starts from the end of the list
- **List[{i, j, k}];** returns the elements in the list respectively at position i, j and k
- **List[n..m];** returns the elements in the list between the indices n to m
- **Flatten(List);** returns a list with the same amount of elements but without sub-lists
- Create Sub-lists, Chop lists, etc.

List of Lists



```
0..10;
```

```
List
 0  0
 1  1
 2  2
 3  3
 4  4
 5  5
 6  6
 7  7
 8  8
 9  9
 10 10
@L2 @L1   {11}
```

```
0..10..3;
```

```
List
 0  0
 1  3
 2  6
 3  9
@L2 @L1   {4}
```

```
0..10..~3;
```

```
List
 0  0
 1  3.33333333333333
 2  6.66666666666667
 3  10
@L2 @L1   {4}
```

```
0..10..#3;
```

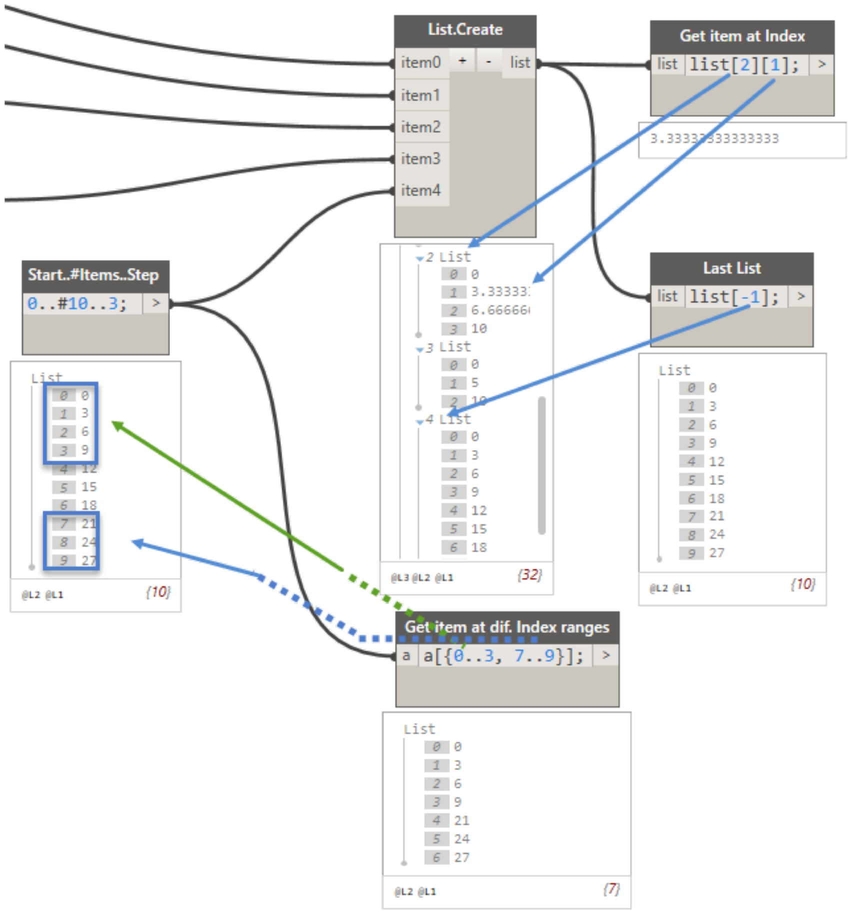
```
List
 0  0
 1  5
 2  10
@L2 @L1   {3}
```

```
0..#10..3;
```

```
List
 0  0
 1  3
 2  6
 3  9
 4  12
 5  15
 6  18
 7  21
 8  24
 9  27
@L2 @L1   {10}
```

```
List
 0 List
   0  0
   1  1
   2  2
   3  3
   4  4
   5  5
   6  6
   7  7
   8  8
   9  9
   10 10
 1 List
   0  0
   1  3
   2  6
@L3 @L2 @L1   {32}
```

Get Items at Index



```
list[2][1];
```

```
3.33333333333333
```

```
list[-1];
```

```
0..#10..3;
```

```
a[{0..3, 7..9}];
```

```
2 List
  0 0
  1 3.333333
  2 6.666666
  3 10
3 List
  0 0
  1 5
  2 10
4 List
  0 0
  1 3
  2 6
  3 9
  4 12
  5 15
  6 18
```

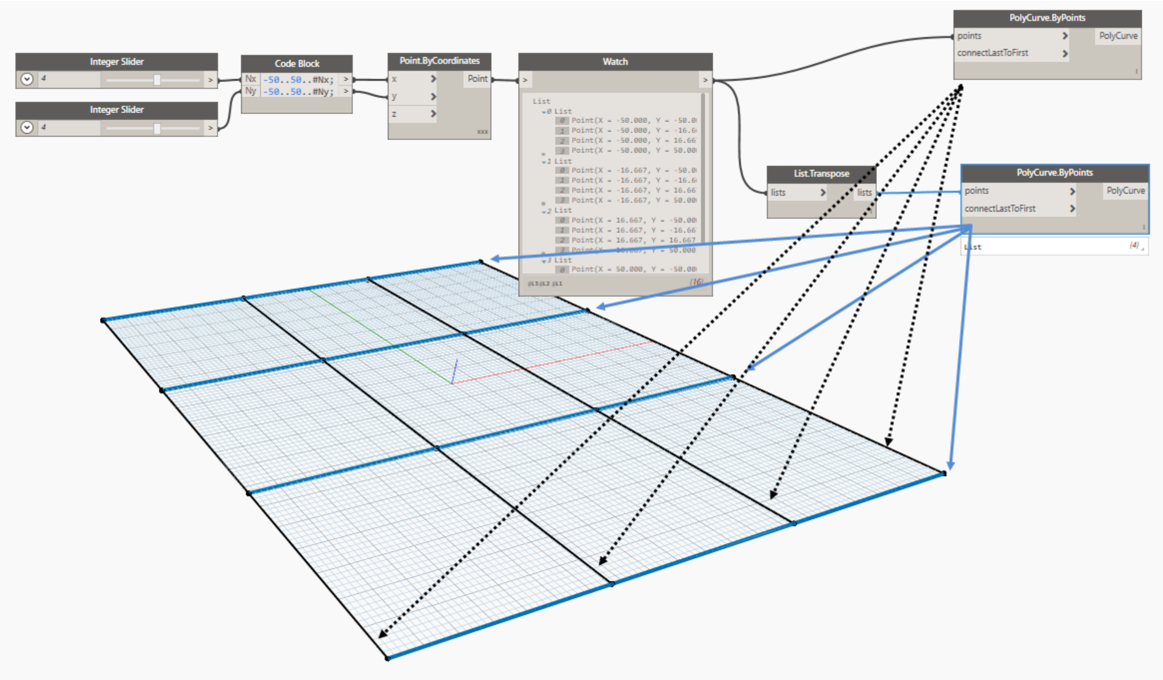
```
List
  0 0
  1 3
  2 6
  3 9
  4 12
  5 15
  6 18
  7 21
  8 24
  9 27
```

```
List
  0 0
  1 3
  2 6
  3 9
  4 12
  5 15
  6 18
  7 21
  8 24
  9 27
```

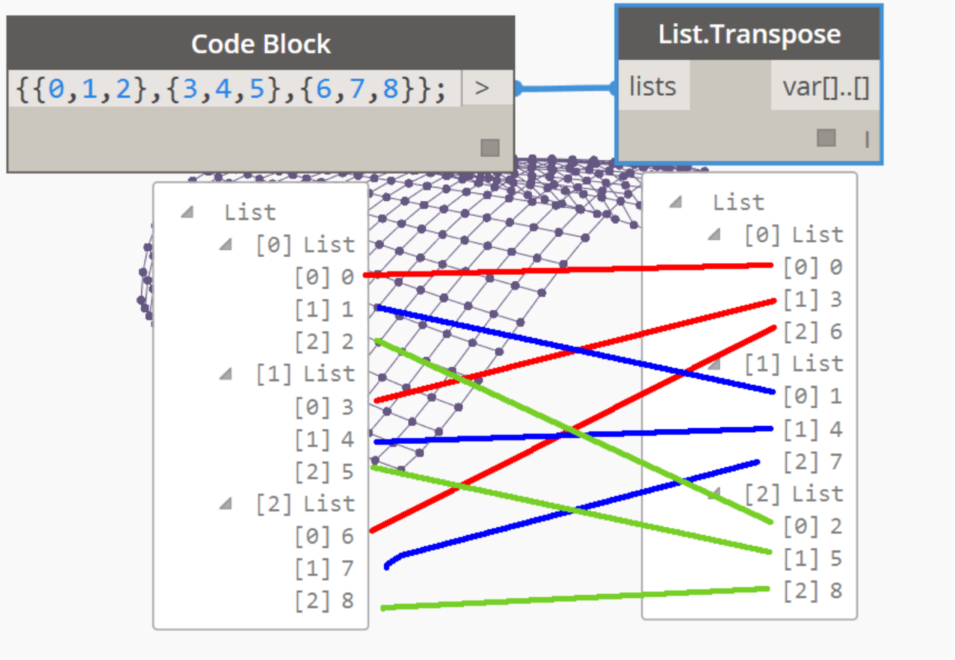
```
List
  0 0
  1 3
  2 6
  3 9
  4 21
  5 24
  6 27
```

Transpose List of Lists

- When transposing a list of lists the data structure changes so that all elements at the same index in each sub-lists are grouped together in the output



```
-50..50..#Nx;
-50..50..#Ny;
```



```
{{0,1,2},{3,4,5},{6,7,8}};
```

```
List
  [0] List
    [0] 0
    [1] 1
    [2] 2
  [1] List
    [0] 3
    [1] 4
    [2] 5
  [2] List
    [0] 6
    [1] 7
    [2] 8
```

```
List
  [0] List
    [0] 0
    [1] 3
    [2] 6
  [1] List
    [0] 1
    [1] 4
    [2] 7
  [2] List
    [0] 2
    [1] 5
    [2] 8
```

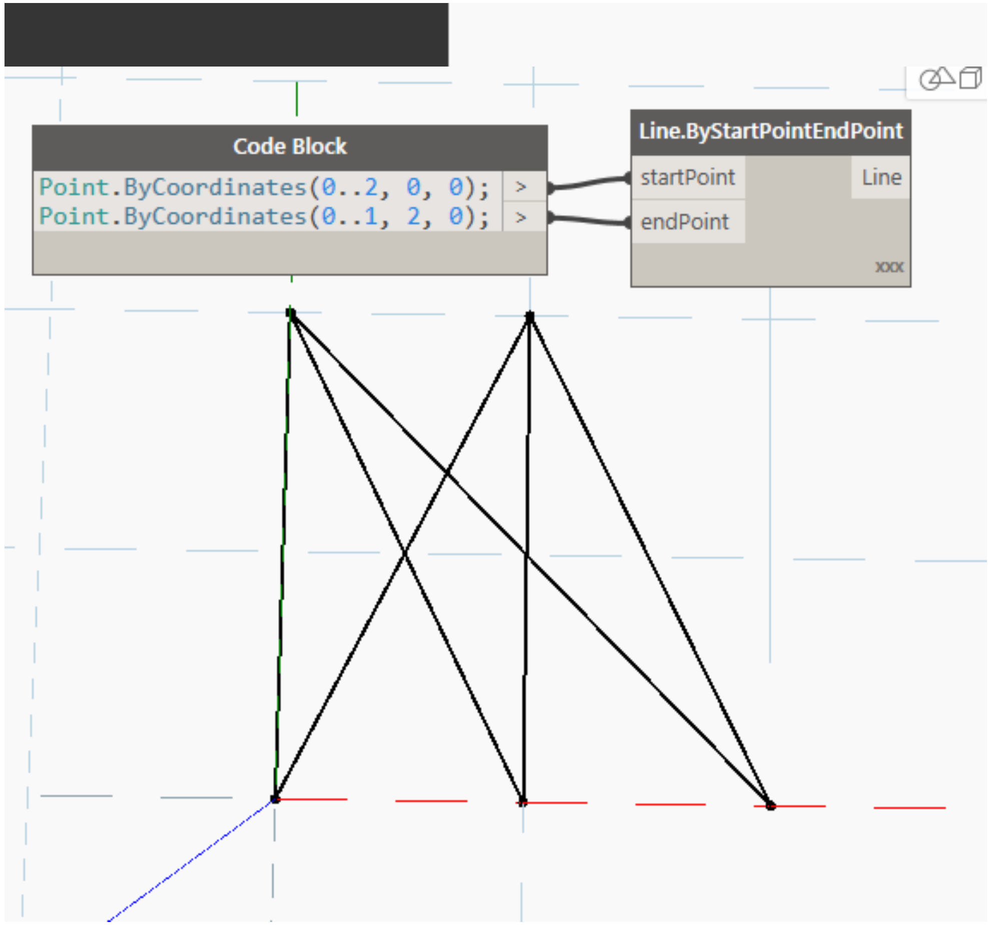
Lacing Strategy

- Each node is a **function** and can iterate over a list of inputs, this is how Visual Programming handles loops

- If the provided inputs have different lengths, the lacing strategy on the node is used to combine the inputs as function arguments

- The lacing affects the structure of the output

- Lacing Strategies
  - Shortest
  - Longest
  - Cross Product



[redacted]

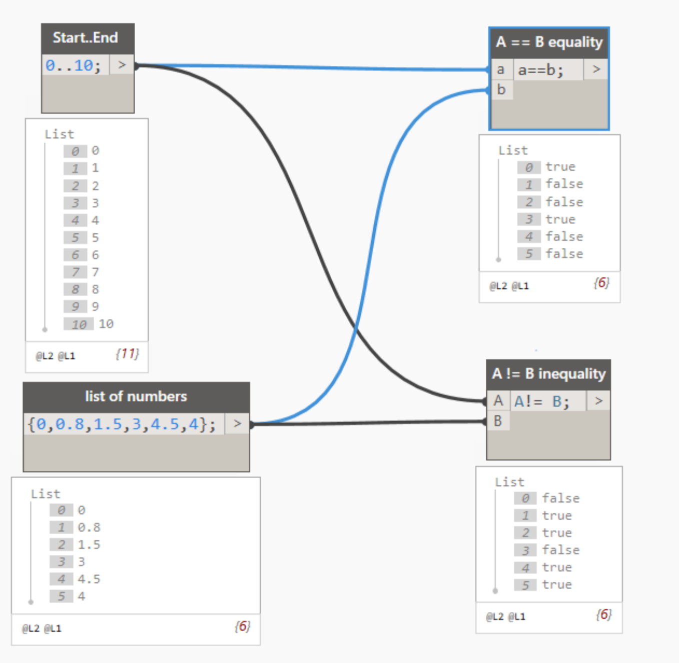
Code Block

```
Point.ByCoordinates(0..2, 0, 0);
Point.ByCoordinates(0..1, 2, 0);
```

Filtering, Grouping and Sorting

Boolean Expressions in Code Blocks

- A **==** B equality
- A **!=** B inequality
- A **<** B less than
- A **<=** B less than or equal to
- A **>** B greater than
- A **>=** B greater than or equal to
- **&&** logical AND (true only if all the arguments are true)
- **||** logical OR (true if at least one argument is true)



```
0..10;
```

```
List
  0 0
  1 1
  2 2
  3 3
  4 4
  5 5
  6 6
  7 7
  8 8
  9 9
  10 10
@L2 @L1        {11}
```

```
{0,0.8,1.5,3,4.5,4};
```

```
List
  0 0
  1 0.8
  2 1.5
  3 3
  4 4.5
  5 4
@L2 @L1        {6}
```

```
a==b;
```

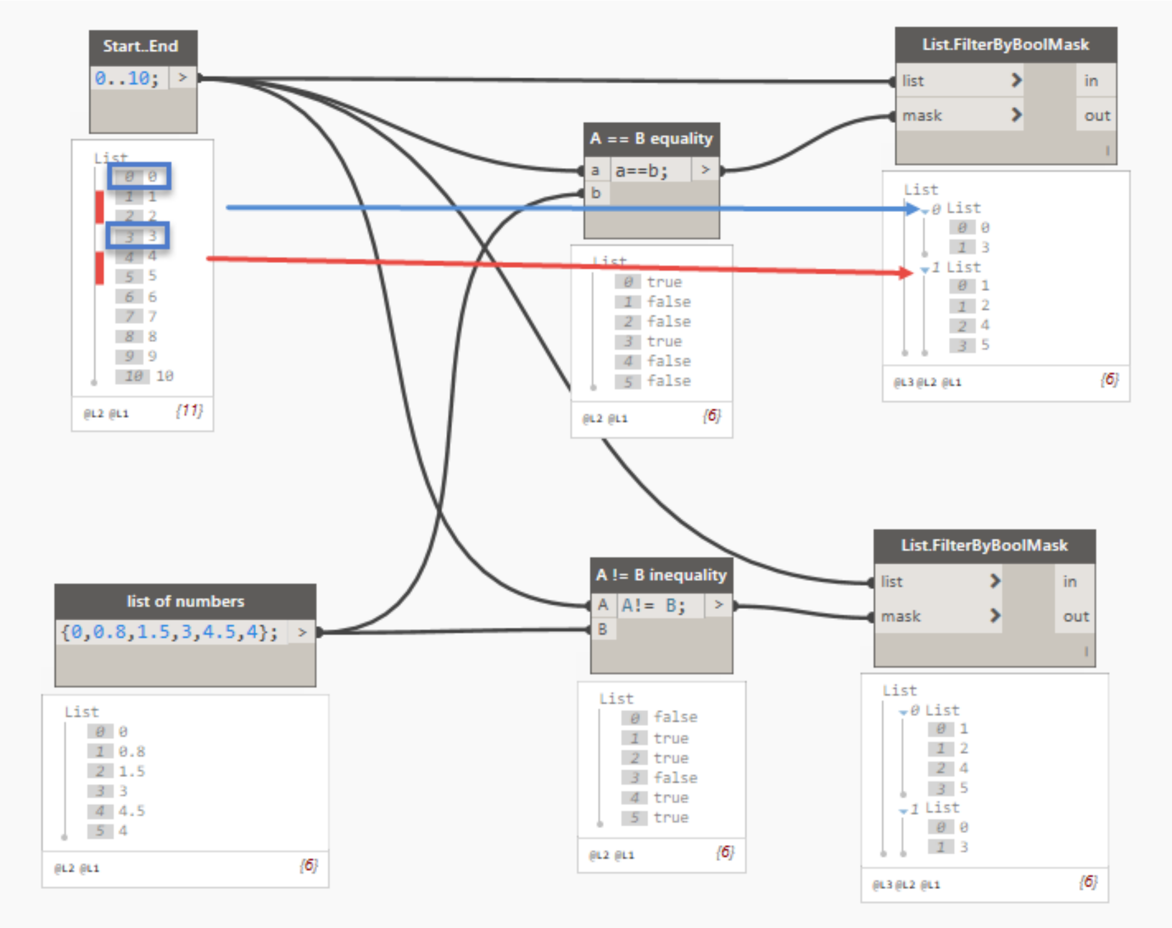
```
List
  0 true
  1 false
  2 false
  3 true
  4 false
  5 false
@L2 @L1        {6}
```

```
A!= B;
```

```
List
  0 false
  1 true
  2 true
  3 false
  4 true
  5 true
@L2 @L1        {6}
```

Filter By Boolean Mask

true : in = false : out



```
0..10;
```

```
List
  0 0
  1 1
  2 2
  3 3
  4 4
  5 5
  6 6
  7 7
  8 8
  9 9
  10 10
@L2 @L1                {11}
```

```
a==b;
```

```
List
  0 true
  1 false
  2 false
  3 true
  4 false
  5 false
@L2 @L1                {6}
```

```
List
  0 List
    0 0
    1 3
  1 List
    0 1
    1 2
    2 4
    3 5
@L3 @L2 @L1            {6}
```

```
{0,0.8,1.5,3,4.5,4};
```

```
List
  0 0
  1 0.8
  2 1.5
  3 3
  4 4.5
  5 4
@L2 @L1                {6}
```

```
A!= B;
```

```
List
  0 false
  1 true
  2 true
  3 false
  4 true
  5 true
@L2 @L1                {6}
```

```
List
  0 List
    0 1
    1 2
    2 4
    3 5
  1 List
    0 0
    1 3
@L3 @L2 @L1            {6}
```

Grouping

- Grouping by Key
  - Creates sub-lists grouping the items based on a sequence of keys

- Grouping by Function
  - Creates sub-lists grouping the items based on the results of a function applied to the items

Sorting

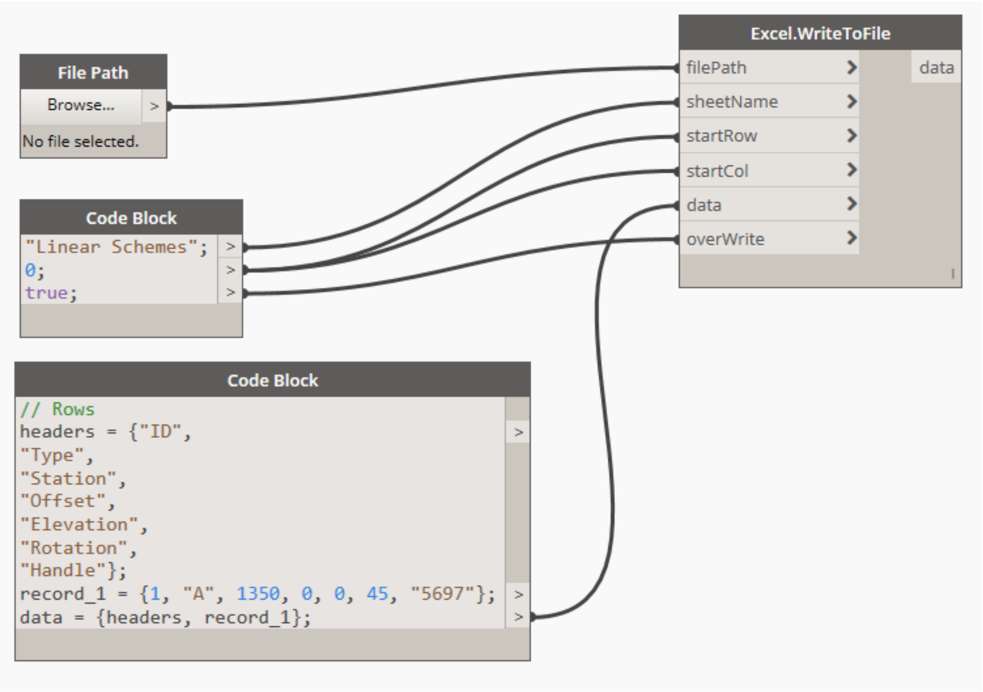
- Sorting by Key
  - Reordering the items in a list based on a sequence of keys

- Sorting by Function
  - Reordering the items in a list based on the results of a function

Dynamo-Excel Link

Write to Excel

- Specify the file path including the extension
- Specify the target tab name
- Specify the starting cell with row and column index
- Create the data by rows
- Specify behavior (create new or update)
- Excel will start automatically
- It is possible to write to multiple tabs at a time



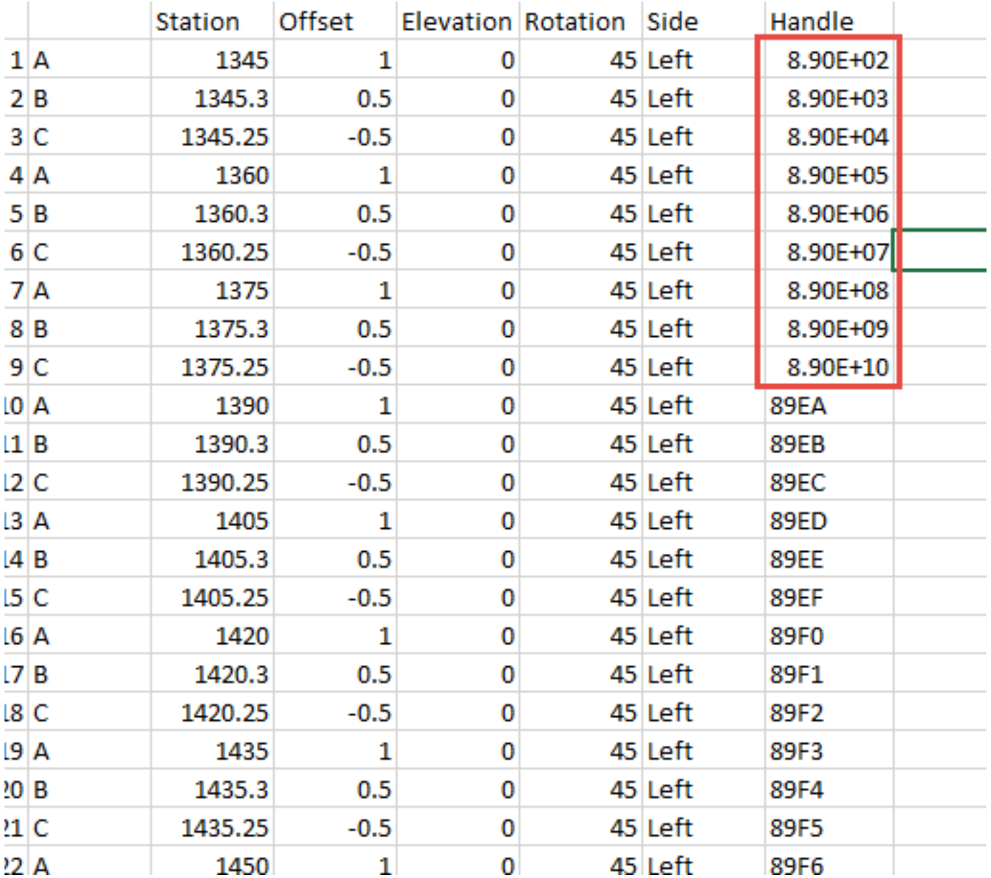
```
"Linear Schemes";
0;
true;
```

```
// Rows
headers = {"ID",
"Type",
"Station",
"Offset",
"Elevation",
"Rotation",
"Handle"};
record_1 = {1, "A", 1350, 0, 0, 45, "5697"};
data = {headers, record_1};
```

Write to Excel | NOTE

- Pay attention to Hexadecimal values (i.e. those used for AutoCAD Handles) as Excel gets easily confused with the 10^ notation

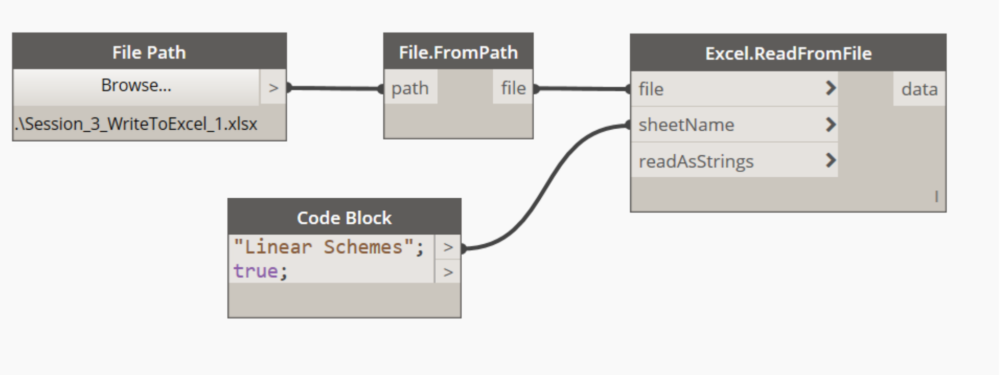
- Use String.Insert and use the quotes to make sure the values are passed as strings in Excel



		Station	Offset	Elevation	Rotation	Side	Handle
1	A	1345	1	0	45	Left	8.90E+02
2	B	1345.3	0.5	0	45	Left	8.90E+03
3	C	1345.25	-0.5	0	45	Left	8.90E+04
4	A	1360	1	0	45	Left	8.90E+05
5	B	1360.3	0.5	0	45	Left	8.90E+06
6	C	1360.25	-0.5	0	45	Left	8.90E+07
7	A	1375	1	0	45	Left	8.90E+08
8	B	1375.3	0.5	0	45	Left	8.90E+09
9	C	1375.25	-0.5	0	45	Left	8.90E+10
10	A	1390	1	0	45	Left	89EA
11	B	1390.3	0.5	0	45	Left	89EB
12	C	1390.25	-0.5	0	45	Left	89EC
13	A	1405	1	0	45	Left	89ED
14	B	1405.3	0.5	0	45	Left	89EE
15	C	1405.25	-0.5	0	45	Left	89EF
16	A	1420	1	0	45	Left	89F0
17	B	1420.3	0.5	0	45	Left	89F1
18	C	1420.25	-0.5	0	45	Left	89F2
19	A	1435	1	0	45	Left	89F3
20	B	1435.3	0.5	0	45	Left	89F4
21	C	1435.25	-0.5	0	45	Left	89F5
22	A	1450	1	0	45	Left	89F6

Read from Excel

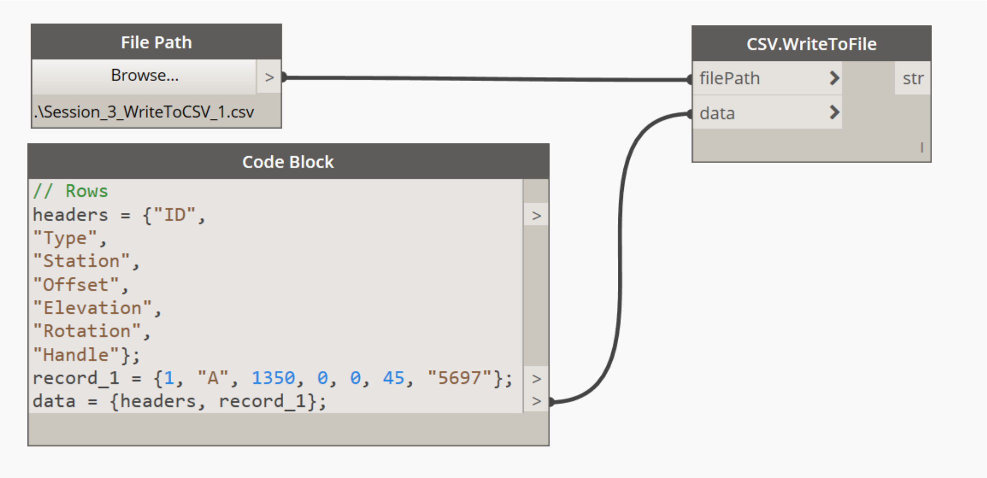
- Specify the file path (string)
- Create a data stream in the memory (File object)
- Specify tab to read the data from
- Read As Strings will convert all the data to text
- Excel will start automatically
- Use Deconstruct to split the headers
- Use Drop Items for a custom # rows to skip



```
"Linear Schemes";
true;
```

Write to CSV

- Similar to Excel but there are no tabs, rows or columns
- There is no output of the node
- Excel does not start

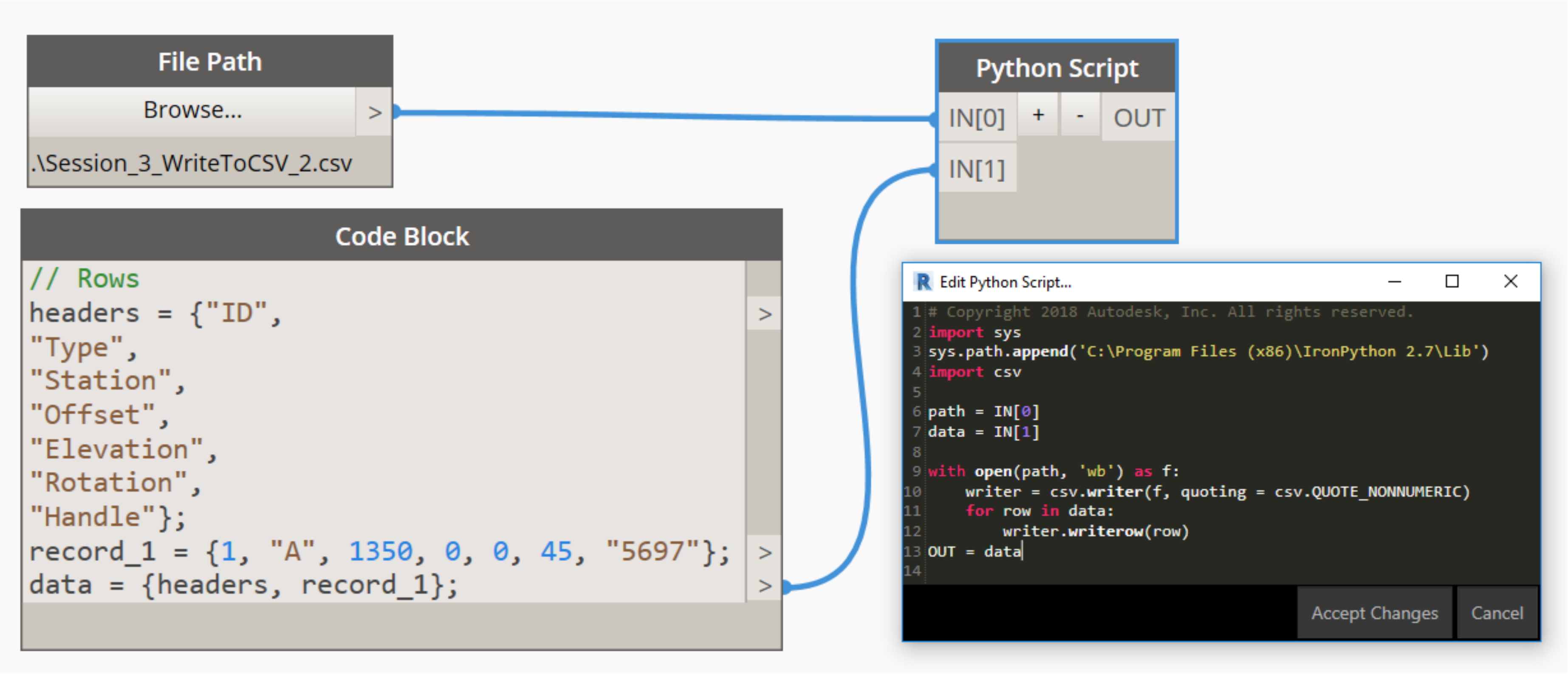


.\Session_3_WriteToCSV_1.csv

```
// Rows
headers = {"ID",
"Type",
"Station",
"Offset",
"Elevation",
"Rotation",
"Handle"};
record_1 = {1, "A", 1350, 0, 0, 45, "5697"};
data = {headers, record_1};
```

Write to CSV | Python

- Fidelity of data types

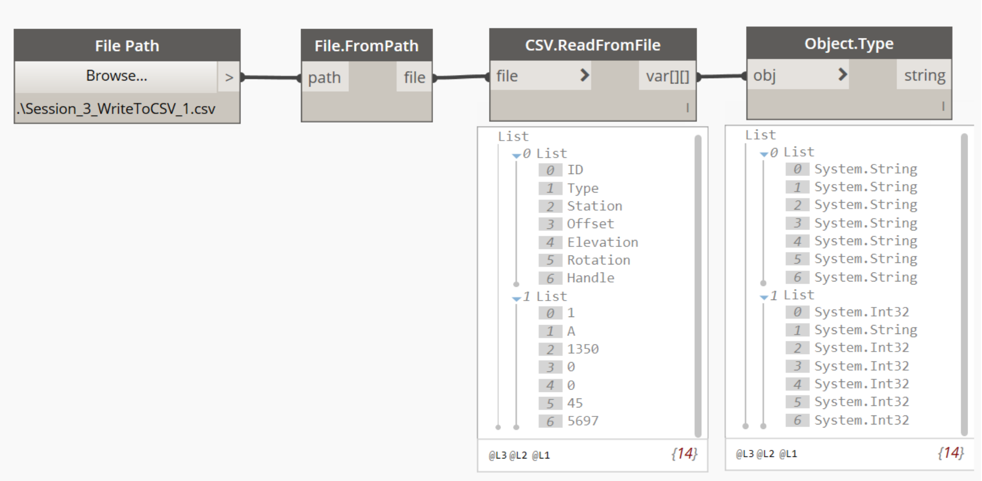


```
// Rows
headers = {"ID",
"Type",
"Station",
"Offset",
"Elevation",
"Rotation",
"Handle"};
record_1 = {1, "A", 1350, 0, 0, 45, "5697"};
data = {headers, record_1};
```

```
# Copyright 2018 Autodesk, Inc. All rights reserved.
import sys
sys.path.append('C:\Program Files (x86)\IronPython 2.7\Lib')
import csv

path = IN[0]
data = IN[1]

with open(path, 'wb') as f:
    writer = csv.writer(f, quoting = csv.QUOTE_NONNUMERIC)
    for row in data:
        writer.writerow(row)
OUT = data

```

Read from CSV

- Specify the path (string)
- Create the data stream in the memory (File)
- Read content by rows

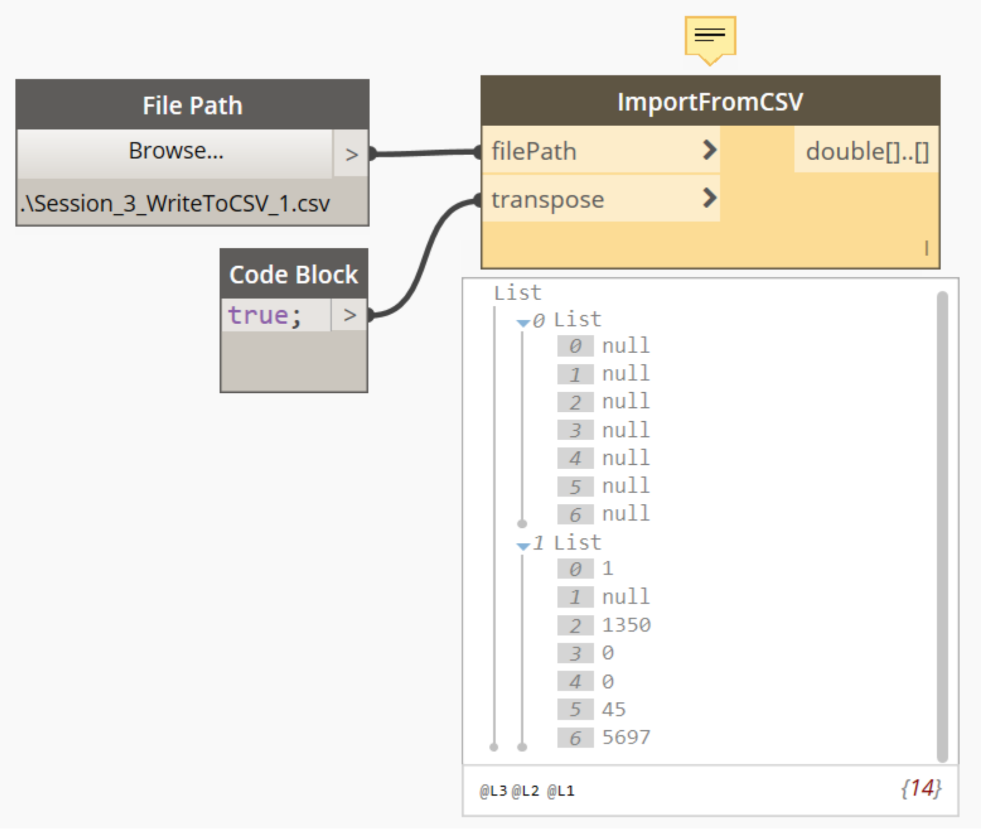


```
List
 0 List
   0 ID
   1 Type
   2 Station
   3 Offset
   4 Elevation
   5 Rotation
   6 Handle
 1 List
   0 1
   1 A
   2 1350
   3 0
   4 0
   5 45
   6 5697
@L3 @L2 @L1                {14}
```

```
List
 0 List
   0 System.String
   1 System.String
   2 System.String
   3 System.String
   4 System.String
   5 System.String
   6 System.String
 1 List
   0 System.Int32
   1 System.String
   2 System.Int32
   3 System.Int32
   4 System.Int32
   5 System.Int32
   6 System.Int32
@L3 @L2 @L1                {14}
```

Import from CSV

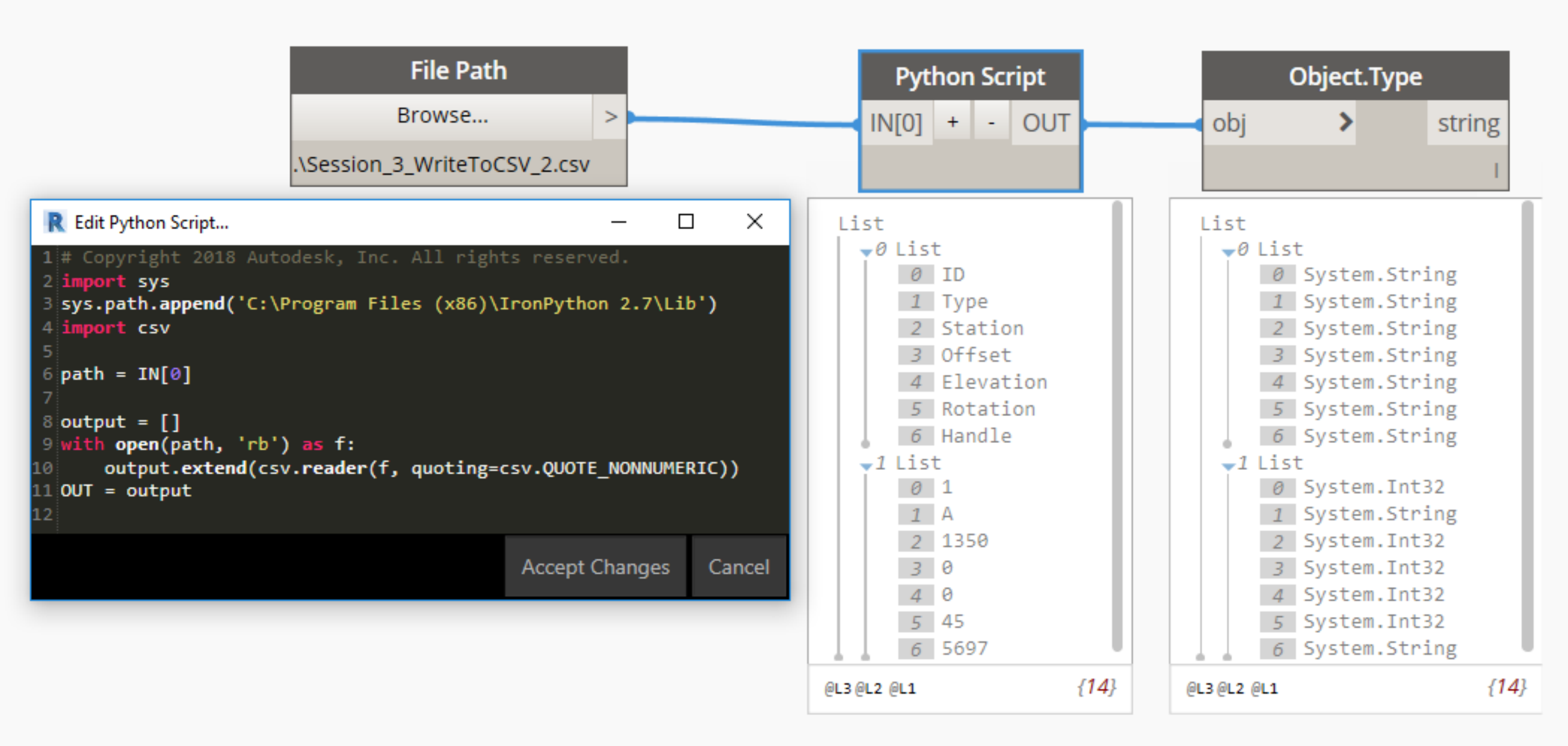
- Specify the path (string)
- True reads content by rows, false by columns
- Does not convert strings



```
List
 0 List
   0 null
   1 null
   2 null
   3 null
   4 null
   5 null
   6 null
 1 List
   0 1
   1 null
   2 1350
   3 0
   4 0
   5 45
   6 5697
@L3 @L2 @L1                {14}
```

Read from CSV | Python

- Specify path, fidelity of data type



```
# Copyright 2018 Autodesk, Inc. All rights reserved.
import sys
sys.path.append('C:\Program Files (x86)\IronPython 2.7\Lib')
import csv

path = IN[0]

output = []
with open(path, 'rb') as f:
    output.extend(csv.reader(f, quoting=csv.QUOTE_NONNUMERIC))
OUT = output
```

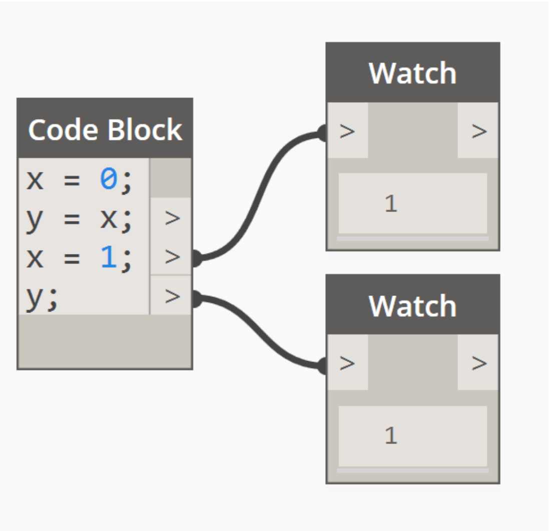
```
List
 0 List
   0 ID
   1 Type
   2 Station
   3 Offset
   4 Elevation
   5 Rotation
   6 Handle
 1 List
   0 1
   1 A
   2 1350
   3 0
   4 0
   5 45
   6 5697
@L3 @L2 @L1                {14}
```

```
List
 0 List
   0 System.String
   1 System.String
   2 System.String
   3 System.String
   4 System.String
   5 System.String
   6 System.String
 1 List
   0 System.Int32
   1 System.String
   2 System.Int32
   3 System.Int32
   4 System.Int32
   5 System.Int32
   6 System.String
@L3 @L2 @L1                {14}
```

Design Script

Language Blocks

- Associative

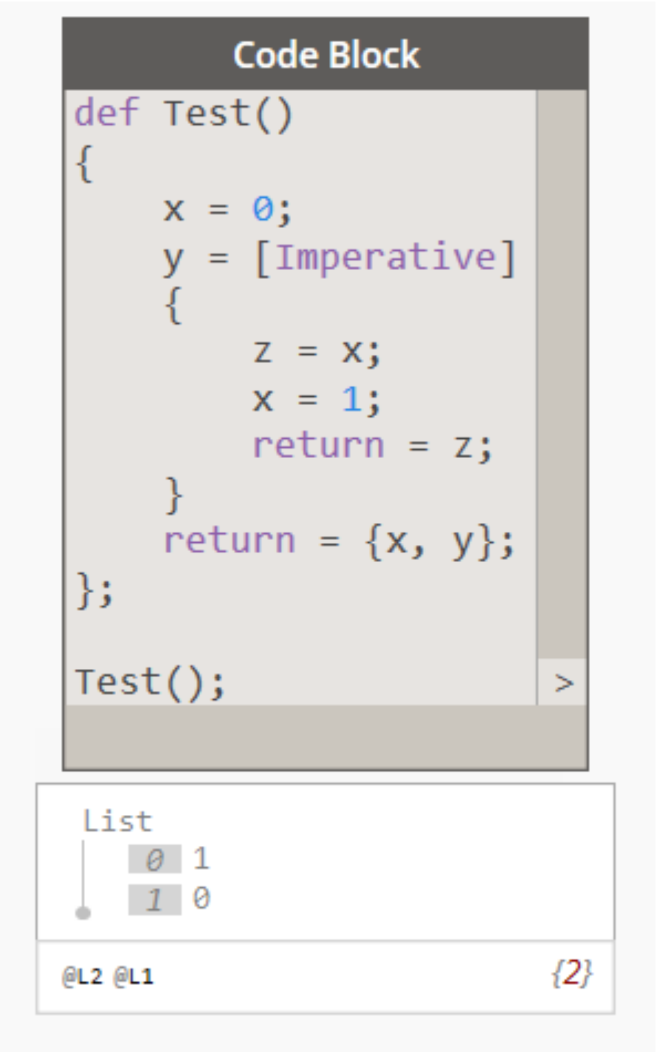


```
x = 0;
y = x;
x = 1;
y;
```

```
1
```

```
1
```

- Imperative (traditional scripting)

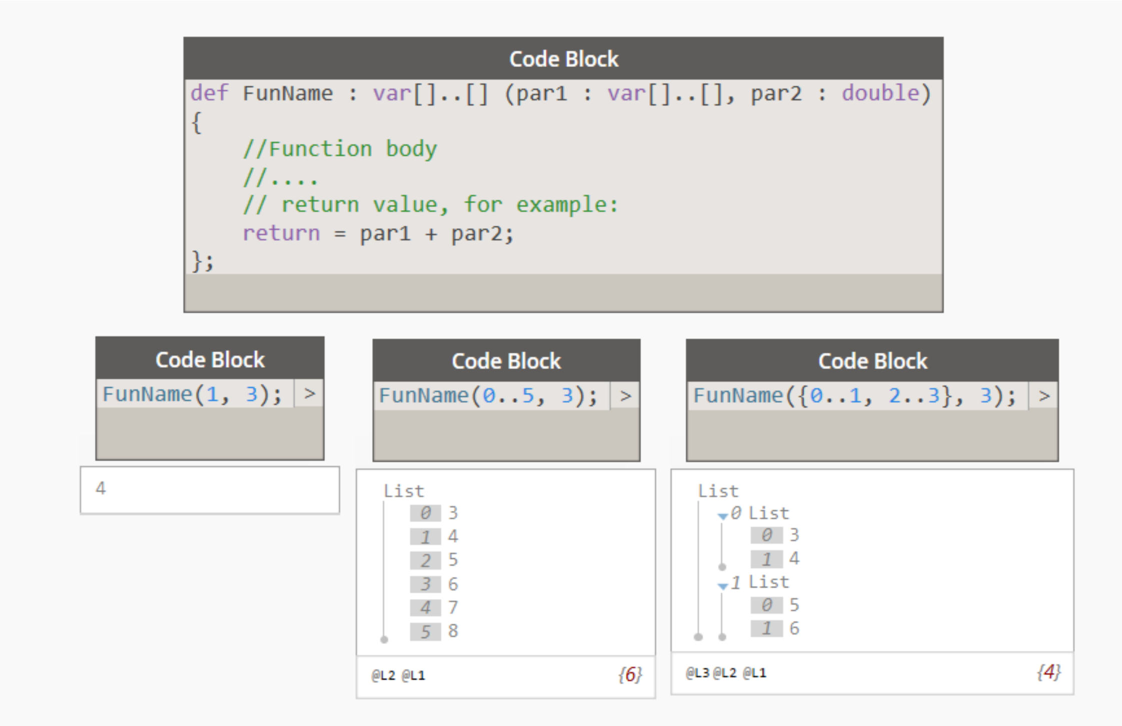


```
def Test()
{
    x = 0;
    y = [Imperative]
    {
        z = x;
        x = 1;
        return = z;
    }
    return = {x, y};
};

Test();
```

```
List
  0 1
  1 0
@L2 @L1                {2}
```

Custom Functions

- def keyword, Name with Pascal case, output and parameters rank and types, can be called in other CBs



```
def FunName : var[]..[] (par1 : var[]..[], par2 : double)
{
    //Function body
    //....
    // return value, for example:
    return = par1 + par2;
};
```

```
FunName(1, 3);
```

```
4
```

```
FunName(0..5, 3);
```

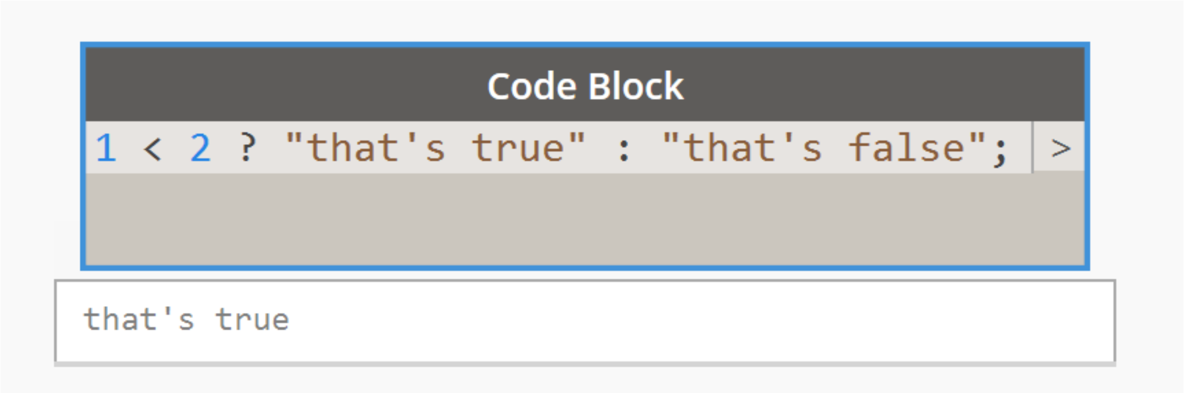
```
List
  0 3
  1 4
  2 5
  3 6
  4 7
  5 8
@L2 @L1                {6}
```

```
FunName({0..1, 2..3}, 3);
```

```
List
  0 List
      0 3
      1 4
  1 List
      0 5
      1 6
@L3 @L2 @L1            {4}
```

Conditional Statements

- Ternary operator, in line conditional statement
- Boolean test ? Return value if true : Return value if false

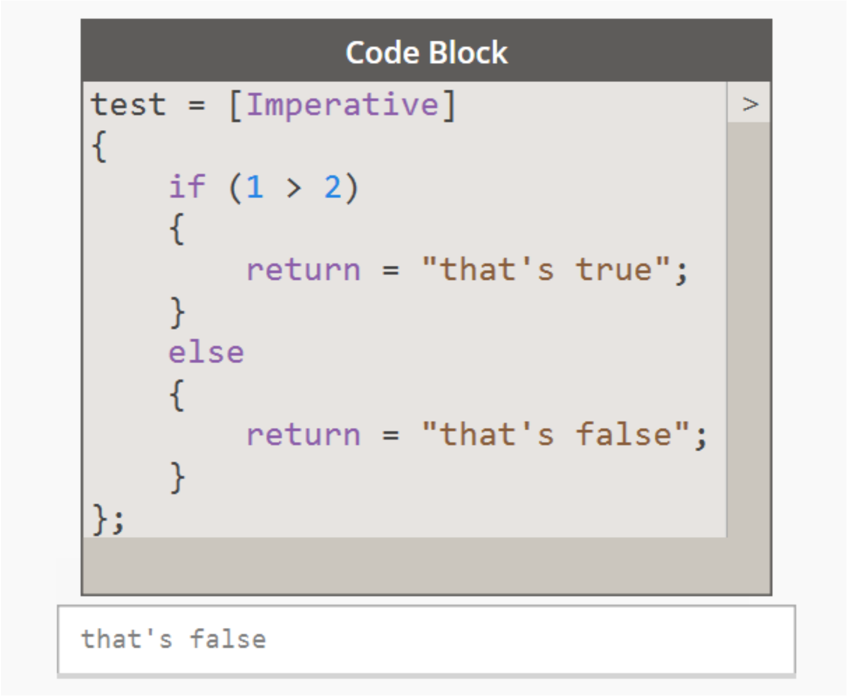


```
1 < 2 ? "that's true" : "that's false";
```

```
that's true
```

Conditional Statements

- If / Else only available in [Imperative] language block

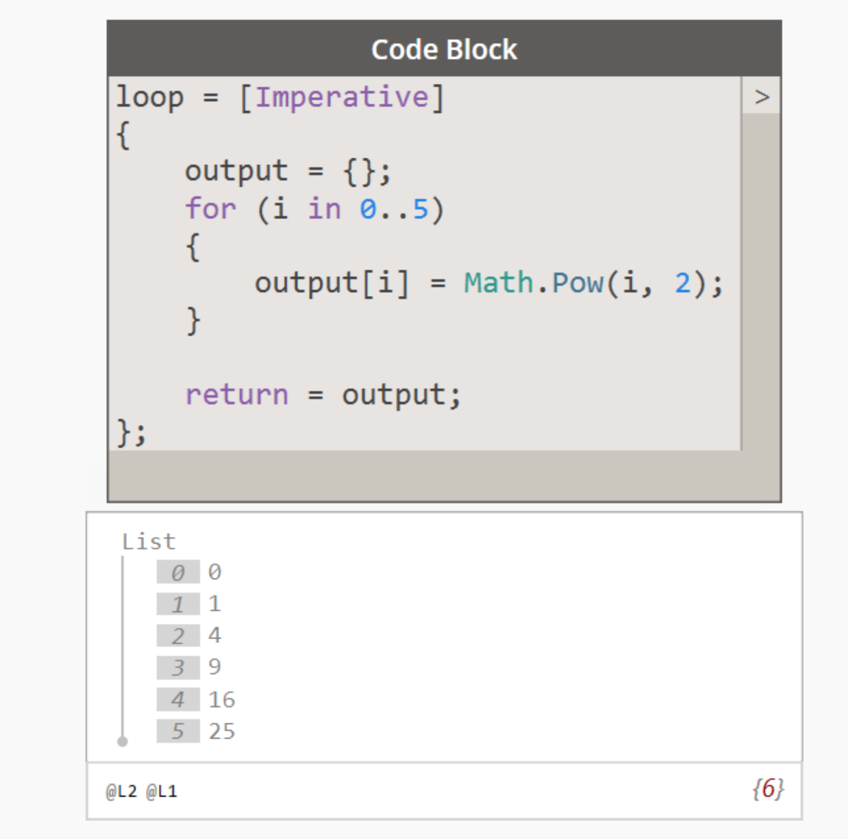


```
test = [Imperative]
{
    if (1 > 2)
    {
        return = "that's true";
    }
    else
    {
        return = "that's false";
    }
};
```

```
that's false
```

Loops

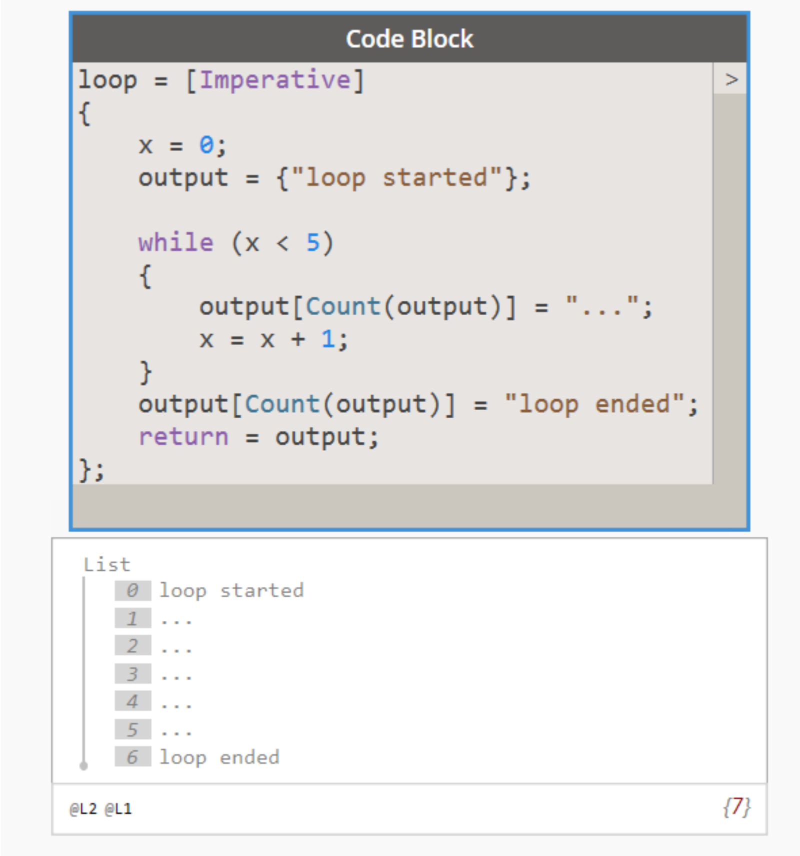
- For / While only available in [Imperative] language block



Code Block

```
loop = [Imperative]
{
    output = {};
    for (i in 0..5)
    {
        output[i] = Math.Pow(i, 2);
    }

    return = output;
};
```

```
List
  0 0
  1 1
  2 4
  3 9
  4 16
  5 25
@L2 @L1                                {6}
```



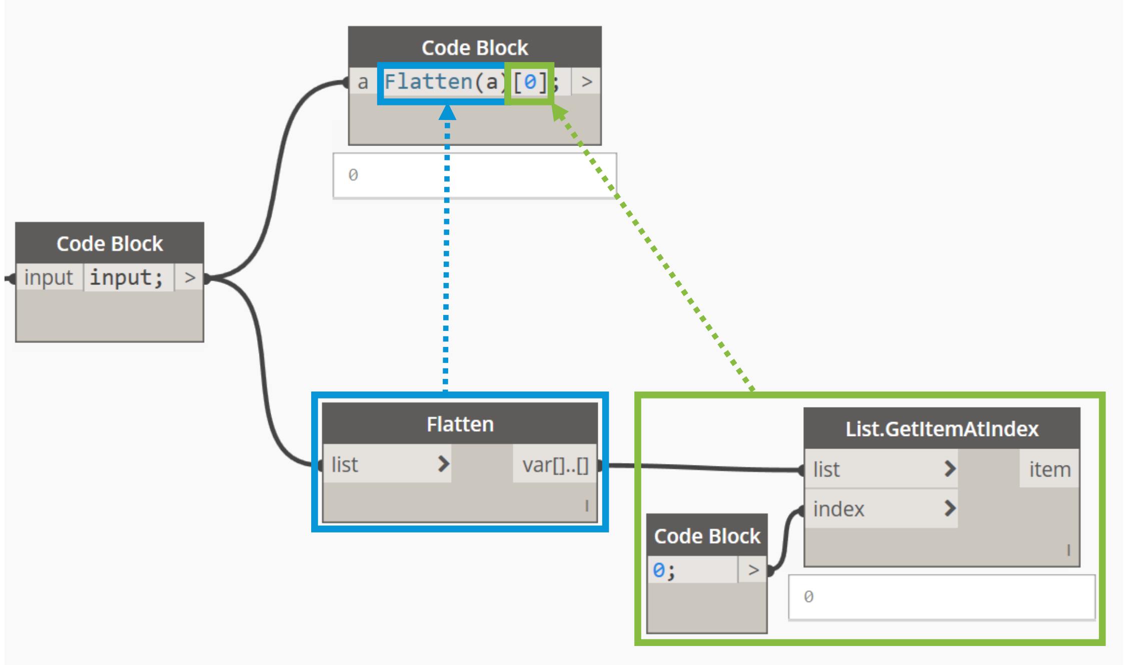
Code Block

```
loop = [Imperative]
{
    x = 0;
    output = {"loop started"};

    while (x < 5)
    {
        output[Count(output)] = "...";
        x = x + 1;
    }
    output[Count(output)] = "loop ended";
    return = output;
};
```

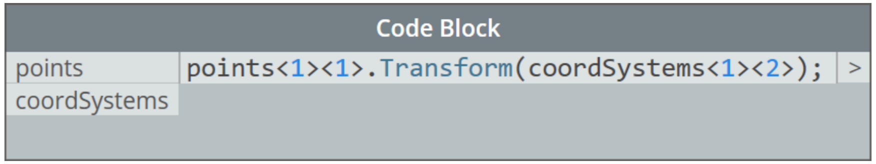
```
List
  0 loop started
  1 ...
  2 ...
  3 ...
  4 ...
  5 ...
  6 loop ended
@L2 @L1                                {7}
```

Design Script Syntax

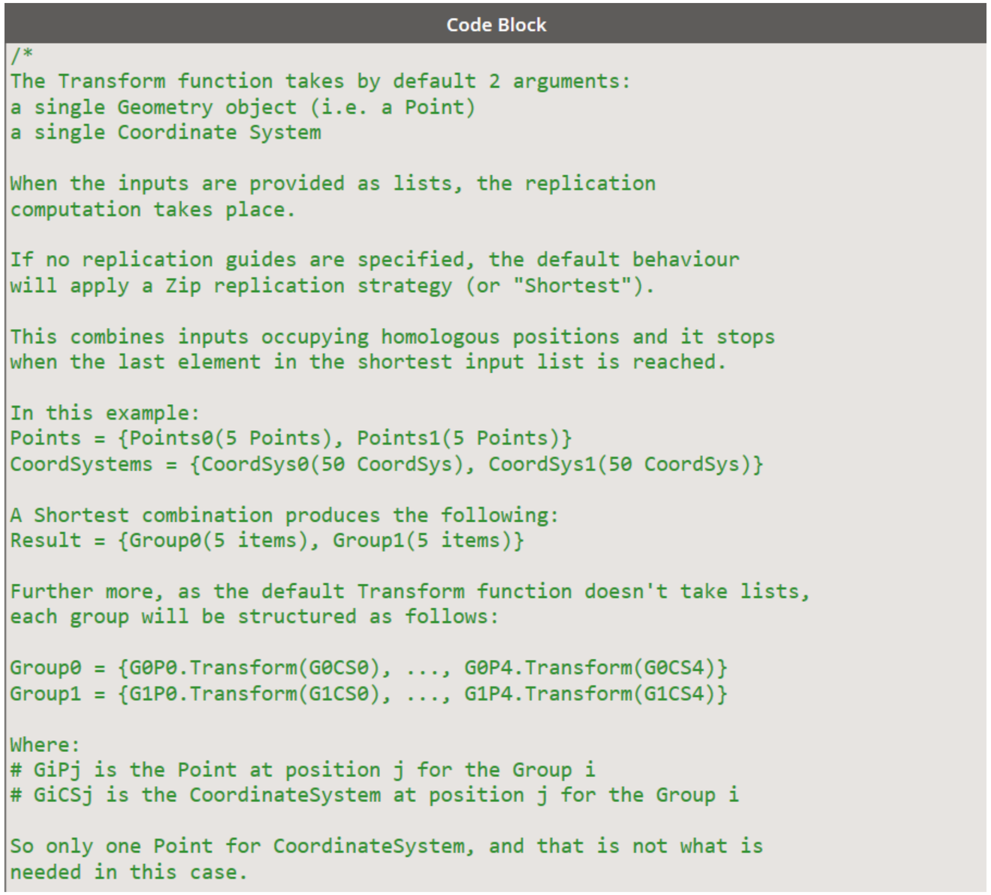
- Dynamo nodes are functions, they can called in a Code Block by name
- The input ports on nodes represent the arguments of the functions
- Retrieve an element from a list at position “n”:
  - x = list[n];



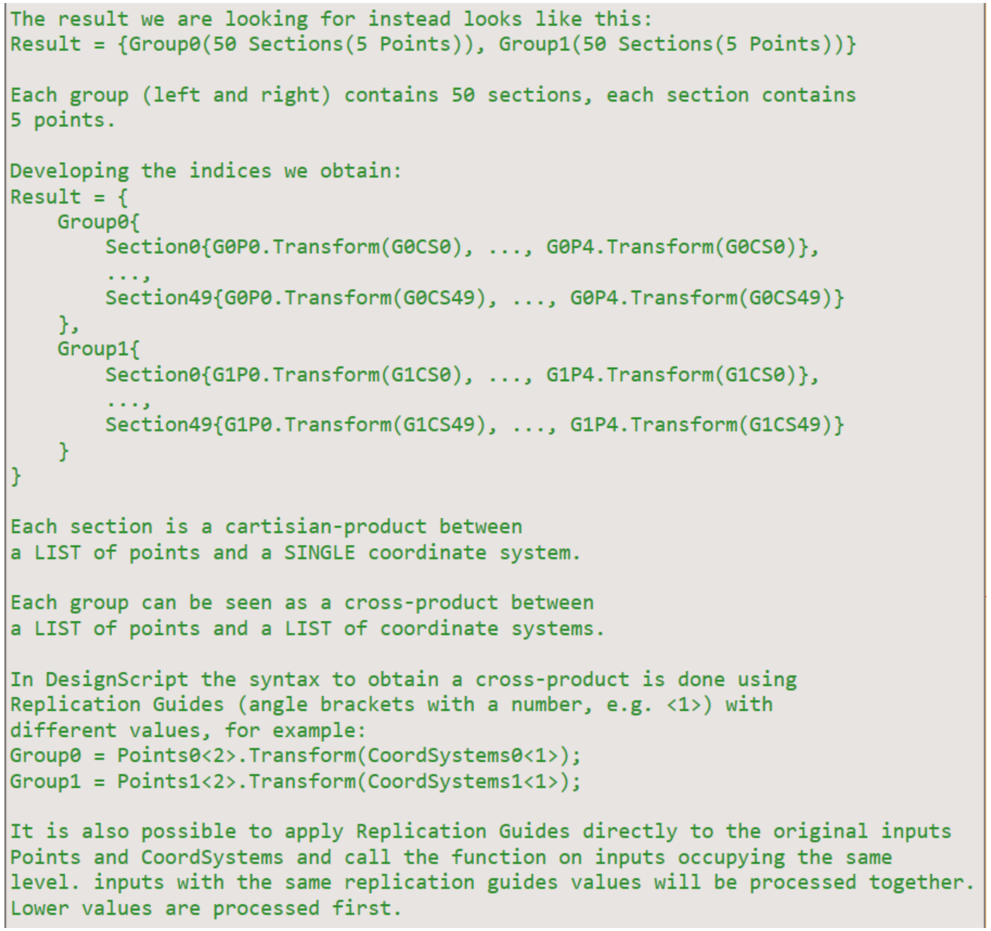
Design Script Syntax



```
points<1><1>.Transform(coordSystems<1><2>);
```

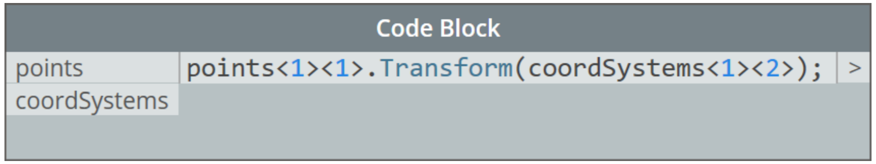


```
/*
The Transform function takes by default 2 arguments:
a single Geometry object (i.e. a Point)
a single Coordinate System

When the inputs are provided as lists, the replication
computation takes place.

If no replication guides are specified, the default behaviour
will apply a Zip replication strategy (or "Shortest").

This combines inputs occupying homologous positions and it stops
when the last element in the shortest input list is reached.

In this example:
Points = {Points0(5 Points), Points1(5 Points)}
CoordSystems = {CoordSys0(50 CoordSys), CoordSys1(50 CoordSys)}

A Shortest combination produces the following:
Result = {Group0(5 items), Group1(5 items)}

Further more, as the default Transform function doesn't take lists,
each group will be structured as follows:

Group0 = {G0P0.Transform(G0CS0), ..., G0P4.Transform(G0CS4)}
Group1 = {G1P0.Transform(G1CS0), ..., G1P4.Transform(G1CS4)}

Where:
# GiPj is the Point at position j for the Group i
# GiCSj is the CoordinateSystem at position j for the Group i

So only one Point for CoordinateSystem, and that is not what is
needed in this case.
```



```
The result we are looking for instead looks like this:
Result = {Group0(50 Sections(5 Points)), Group1(50 Sections(5 Points))}

Each group (left and right) contains 50 sections, each section contains
5 points.

Developing the indices we obtain:
Result = {
    Group0{
        Section0{G0P0.Transform(G0CS0), ..., G0P4.Transform(G0CS0)},
        ...,
        Section49{G0P0.Transform(G0CS49), ..., G0P4.Transform(G0CS49)}
    },
    Group1{
        Section0{G1P0.Transform(G1CS0), ..., G1P4.Transform(G1CS0)},
        ...,
        Section49{G1P0.Transform(G1CS49), ..., G1P4.Transform(G1CS49)}
    }
}

Each section is a cartisian-product between
a LIST of points and a SINGLE coordinate system.

Each group can be seen as a cross-product between
a LIST of points and a LIST of coordinate systems.

In DesignScript the syntax to obtain a cross-product is done using
Replication Guides (angle brackets with a number, e.g. <1>) with
different values, for example:
Group0 = Points0<2>.Transform(CoordSystems0<1>);
Group1 = Points1<2>.Transform(CoordSystems1<1>);

It is also possible to apply Replication Guides directly to the original inputs
Points and CoordSystems and call the function on inputs occupying the same
level. inputs with the same replication guides values will be processed together.
Lower values are processed first.
```

Design Script Syntax



```
points<1><1>.Transform(coordSystems<1><2>);
```

```
This formula returns the result as specified above:
Result = Points<1><2>.Transform(CoordSystems<1><1>);

For the application in this excercise though, we need PolyCurves connecting
homologous points. This can be achieved applying a transpose to the result above.

Result = {
    Group0{
        PolyCurve0{G0P0.Transform(G0CS0), ..., G0P0.Transform(G0CS49)},
        ...,
        PolyCurve4{G0P4.Transform(G0CS0), ..., G0P4.Transform(G0CS49)}
    },
    Group1{
        PolyCurve0{G1P0.Transform(G1CS0), ..., G1P0.Transform(G1CS49)},
        ...,
        PolyCurve4{G1P4.Transform(G1CS0), ..., G1P4.Transform(G1CS49)}
    }
}

This can be achieved directly swapping the Replication Guides
just like the formula that was used in the assignment:
Result = Points<1><1>.Transform(CoordSystems<1><2>);
```

```
In Python syntax this would look like this, assuming:
profile_points as the Points in the example above,
featureline_cs as the CoordSystems in the example above.


result = []
for i in range(len(profile_points)):  # this gives the group index, left or right
    group = []
    points = profile_points[i]
    coord_systems = feautureline_cs[i]
    for j in range(len(points)):
        for k in range(len(coord_systems)):
            p = points[j]
            cs = coord_systems[k]
            group.append(p.Transform(cs))
    result.append(group)
```

Replication

- Visual programming technique to represent iteration

- Nodes can accept a list in place of a single value

- Doing some operation for all elements in a set for a certain number of times where each operation runs independently

- Replications can be nested and create an extra level in the output list

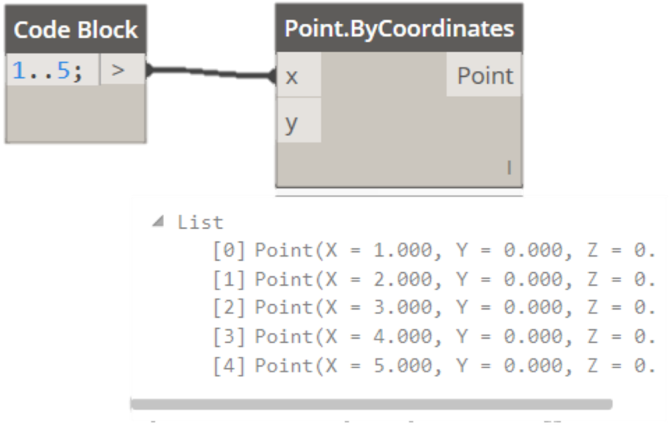
Rank

- Integer number that represents the dimensions of a list

Point	rank 0
Point[]	rank 1
Point[][]	rank 2
Point[][][]	rank 3
Point[]..[]	arbitrary rank

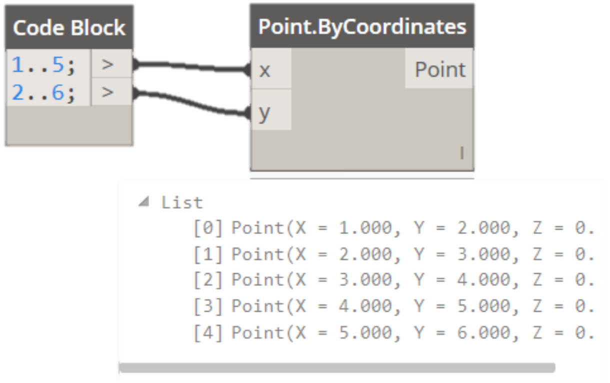
Replication Computation

- **Cartesian**, iterates through all the elements in an input



```
List
    [0] Point(X = 1.000, Y = 0.000, Z = 0.
    [1] Point(X = 2.000, Y = 0.000, Z = 0.
    [2] Point(X = 3.000, Y = 0.000, Z = 0.
    [3] Point(X = 4.000, Y = 0.000, Z = 0.
    [4] Point(X = 5.000, Y = 0.000, Z = 0.
```

- **Zip**, iterates through two or more inputs simultaneously and executes the node with these elements together with other inputs



```
List
    [0] Point(X = 1.000, Y = 2.000, Z = 0.
    [1] Point(X = 2.000, Y = 3.000, Z = 0.
    [2] Point(X = 3.000, Y = 4.000, Z = 0.
    [3] Point(X = 4.000, Y = 5.000, Z = 0.
    [4] Point(X = 5.000, Y = 6.000, Z = 0.
```

Replication Computation

- Depends on the difference between the ranks of the argument (input) and the parameter the node expects (internal function)

- Dk = Ak – Pk (skip arbitrary rank)

- Loop until Dk = 0

- If there are 2 or more Dk > 0 do Zip and decrease Dk by 1

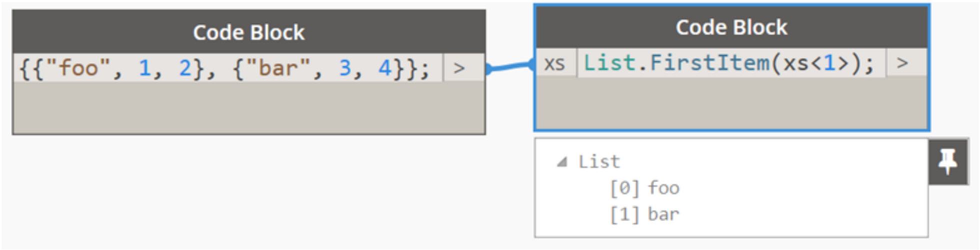
- If there is 1 Dk > 0 do Cartesian and decrease Dk by 1

Replication Guides

Applications

- When the lengths of the arguments are not the same
- Control the Cartesian Replication order
- Append one or more <n> or <nL> to the arguments
- Different replication levels
- Levels enforce iteration
- Processed before replication happens

- At a given level sort the replication guide values
- Apply Zip for equal values and Cartesian for the rest
- If all values are different is a Cross product



```
{{"foo", 1, 2}, {"bar", 3, 4}};
```

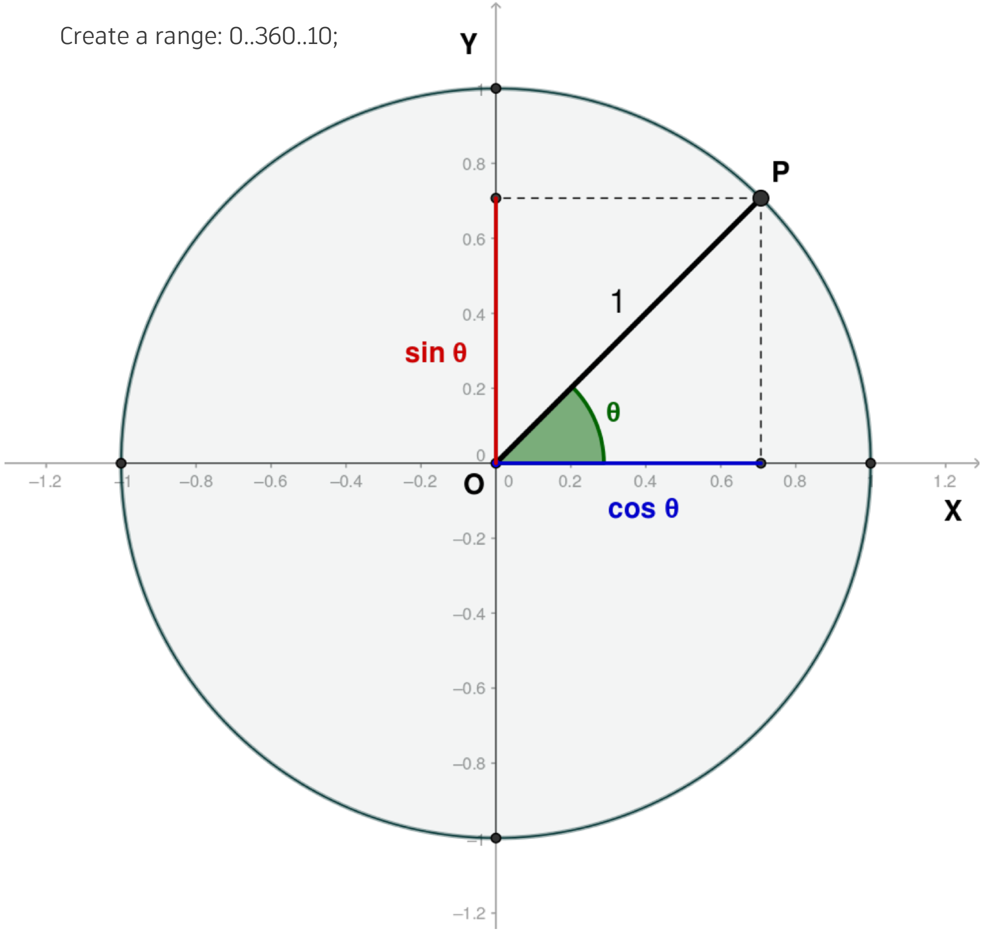
```
List.FirstItem(xs<1>);
```

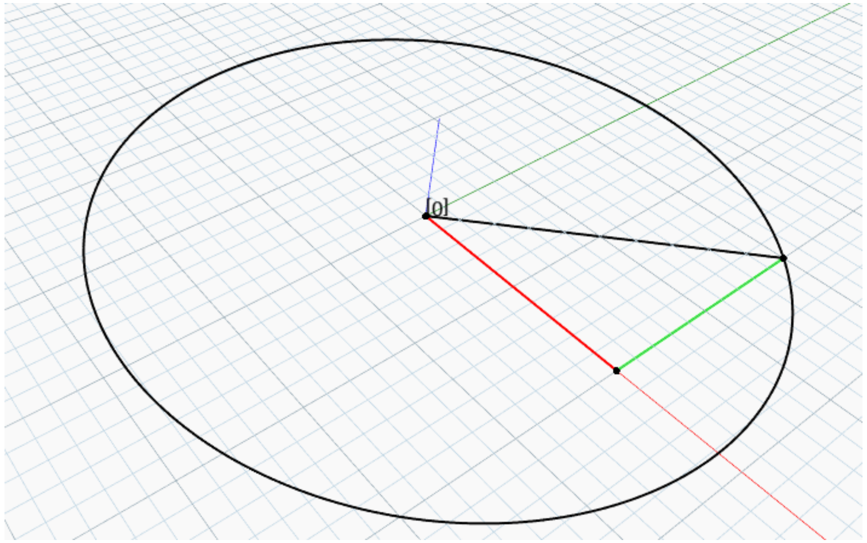
```
List
    [0] foo
    [1] bar
```

Geometry Library

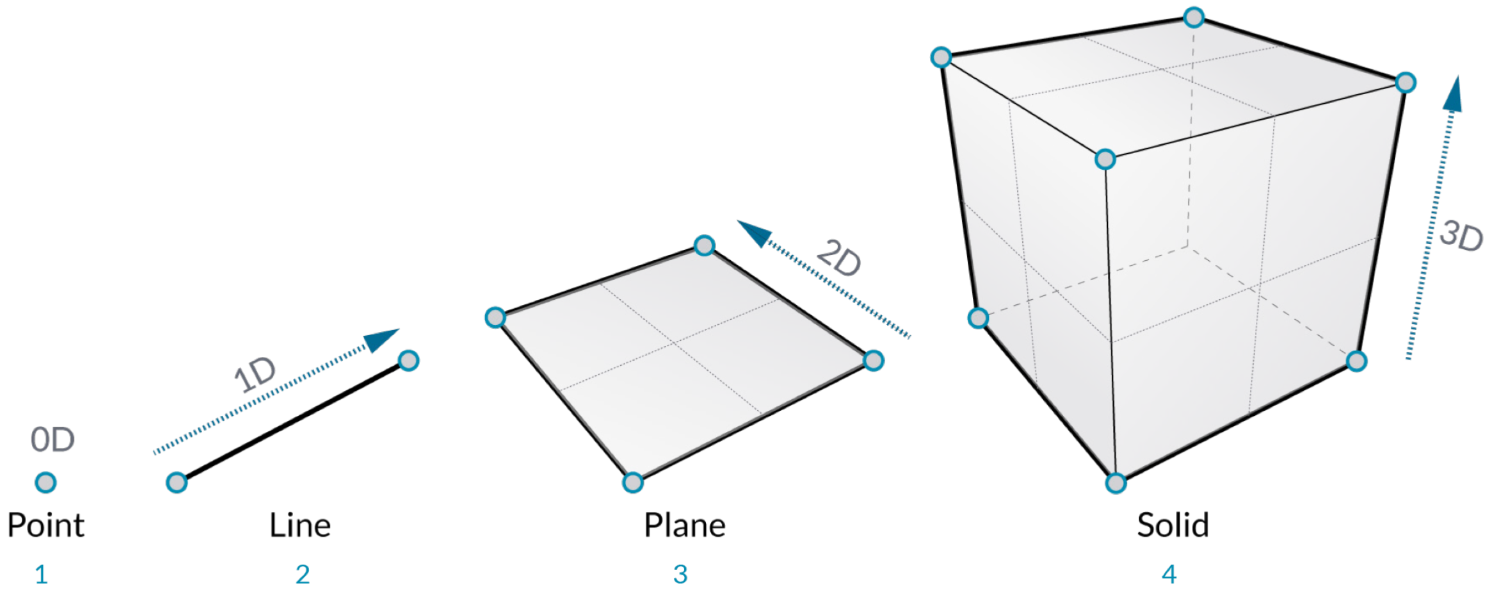
Trigonometry

Create a range: 0..360..10;

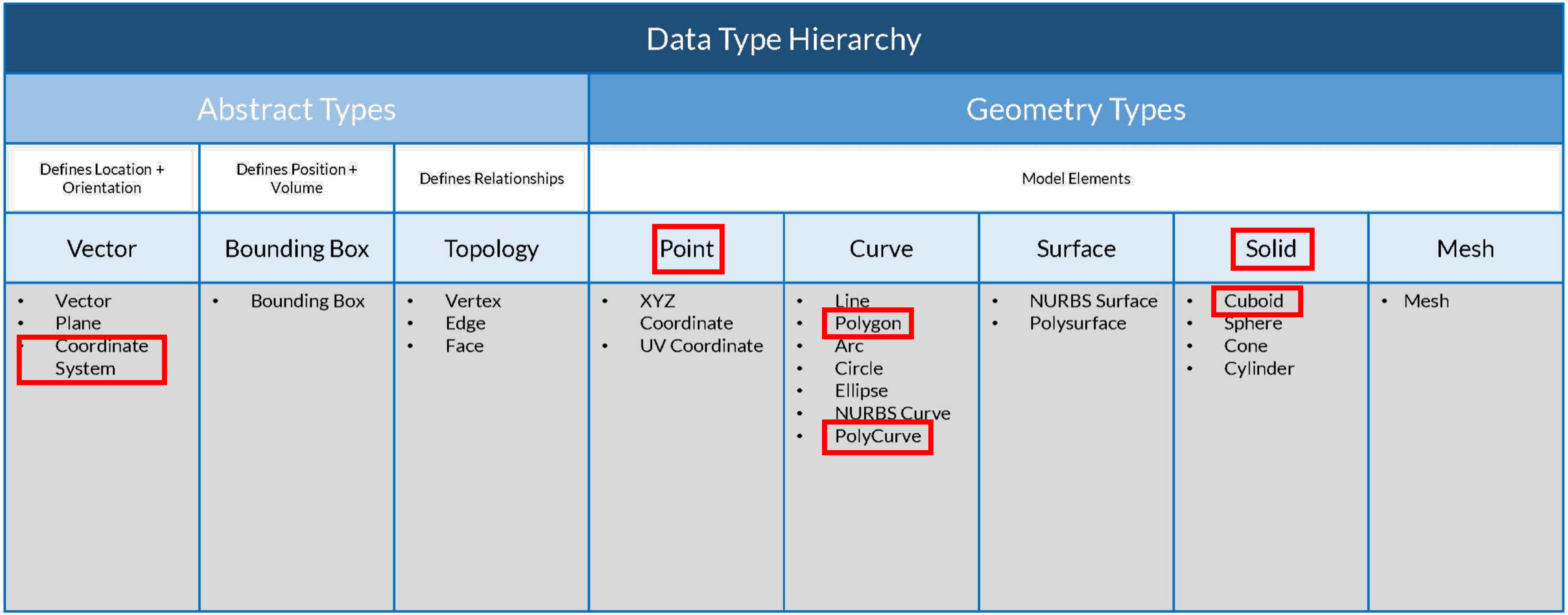




Geometry Objects

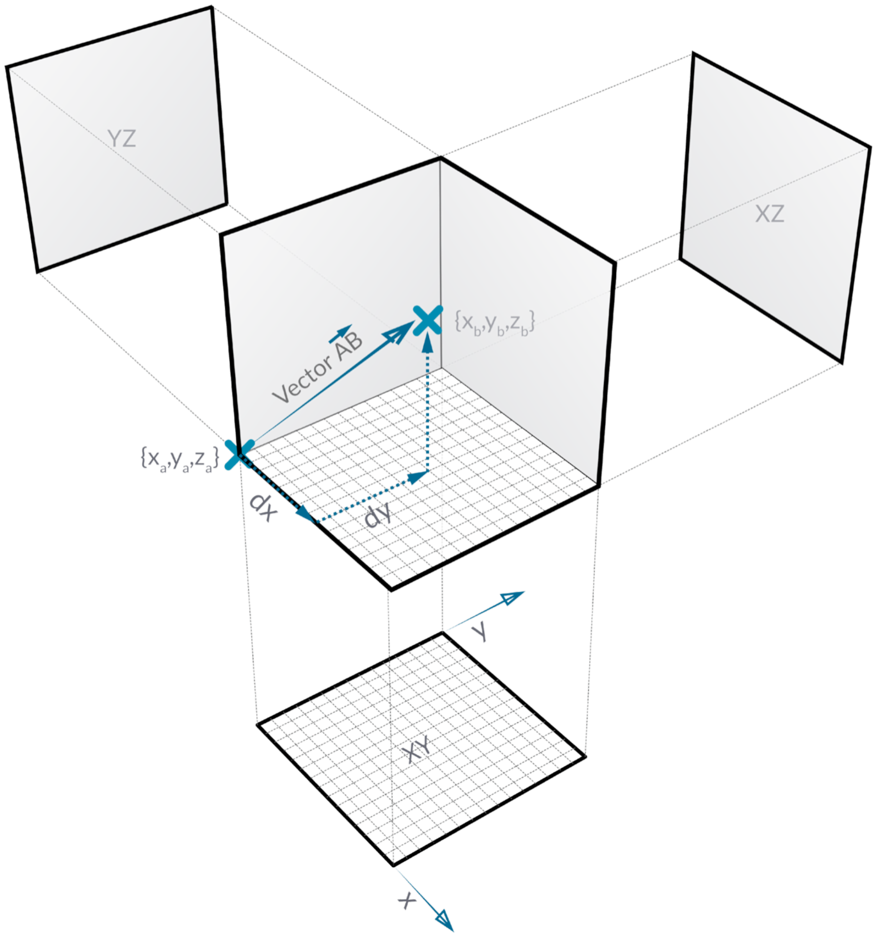


Dynamo Geometry Types

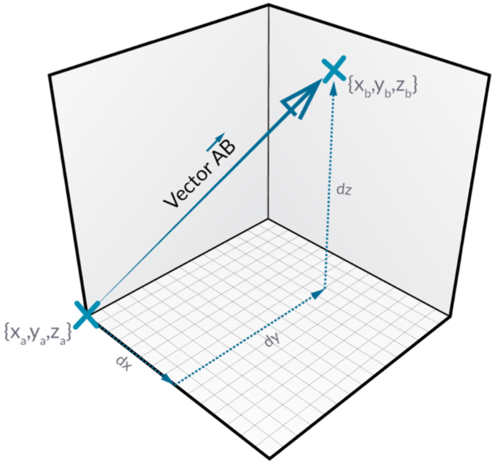


Data Type Hierarchy							
Abstract Types			Geometry Types				
Defines Location + Orientation	Defines Position + Volume	Defines Relationships	Model Elements				
Vector	Bounding Box	Topology	Point	Curve	Surface	Solid	Mesh
• Vector • Plane • Coordinate System	• Bounding Box	• Vertex • Edge • Face	• XYZ Coordinate • UV Coordinate	• Line • Polygon • Arc • Circle • Ellipse • NURBS Curve • PolyCurve	• NURBS Surface • Polysurface	• Cuboid • Sphere • Cone • Cylinder	• Mesh

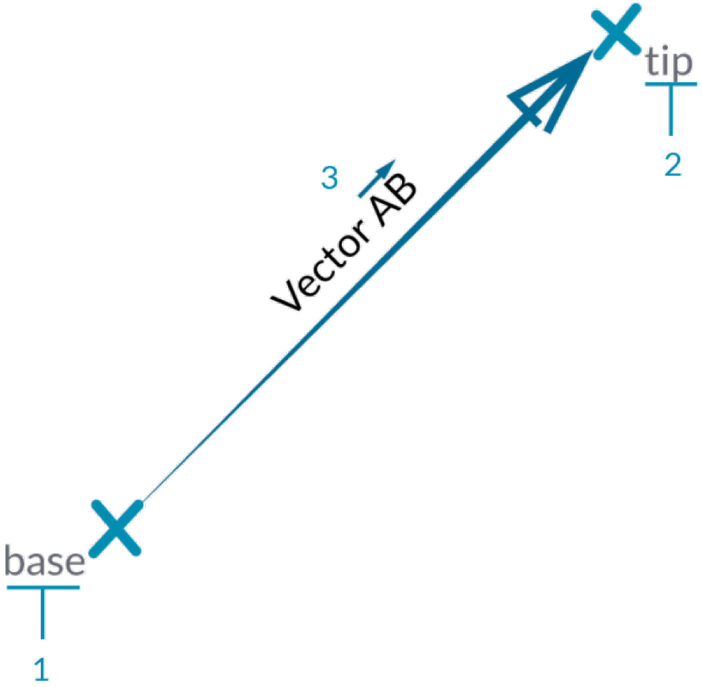
Vectors, Planes, and Coordinate Systems



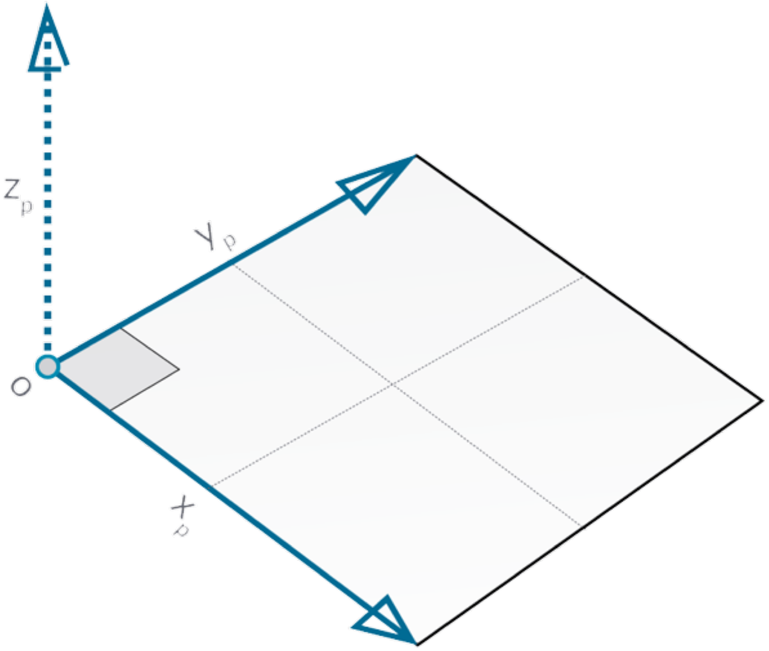
Vector



Vector $\overrightarrow{AB}$ =
{d_x,d_y,d_z} =
{x_b-x_a,y_b-y_a,z_b-z_a}

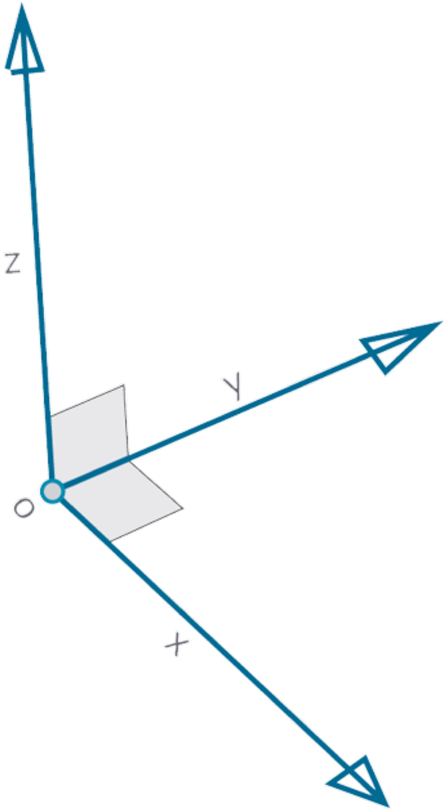


Plane

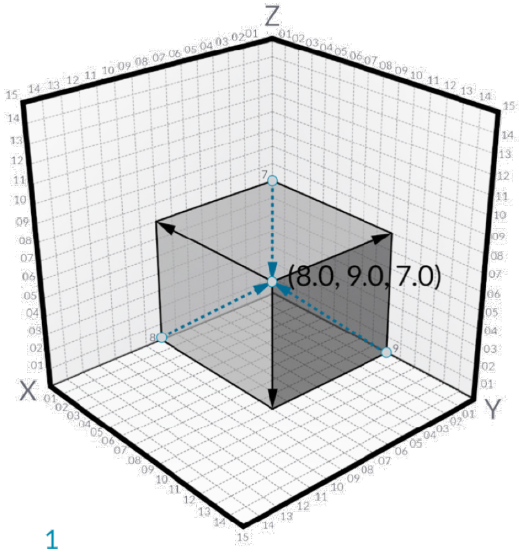


Coordinate System

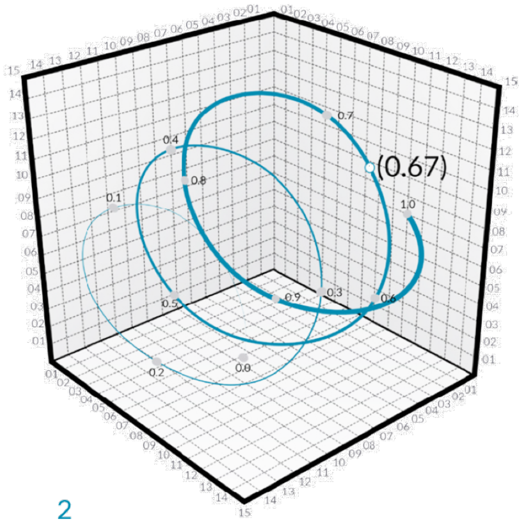
XYZ = RGB



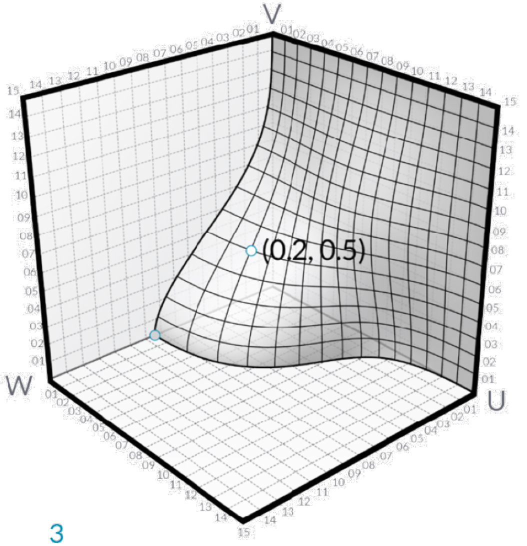
Points



1

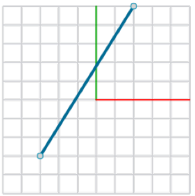


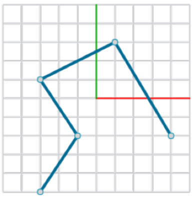
2

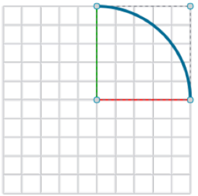


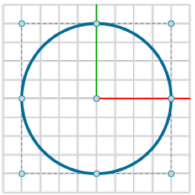
3

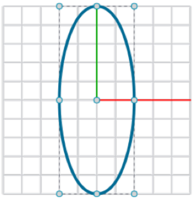
Curves

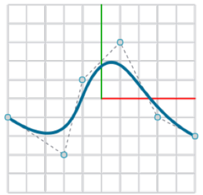

1

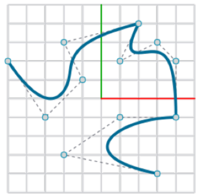

2


3

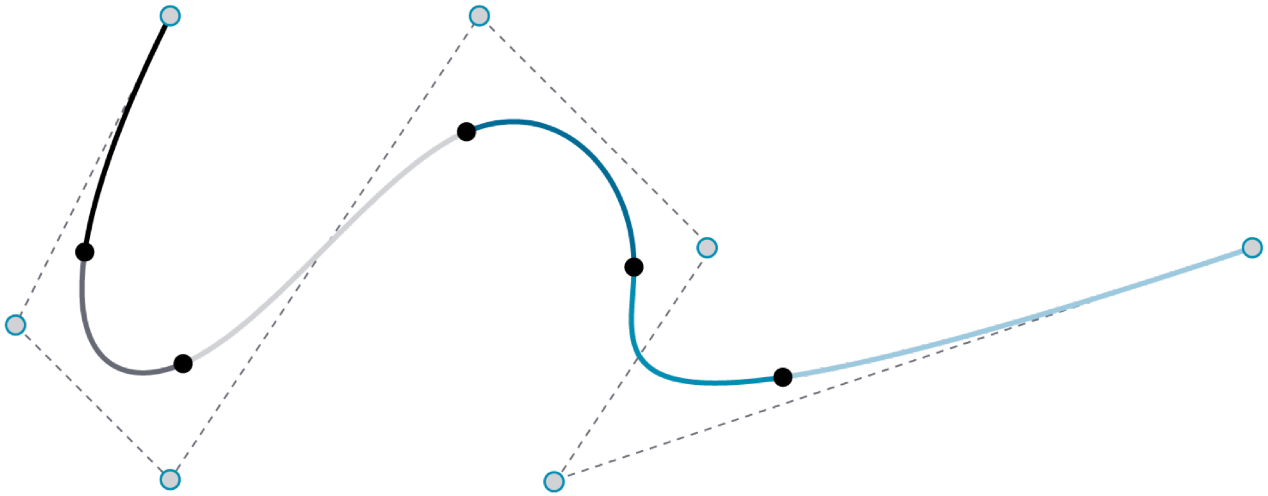

4


5

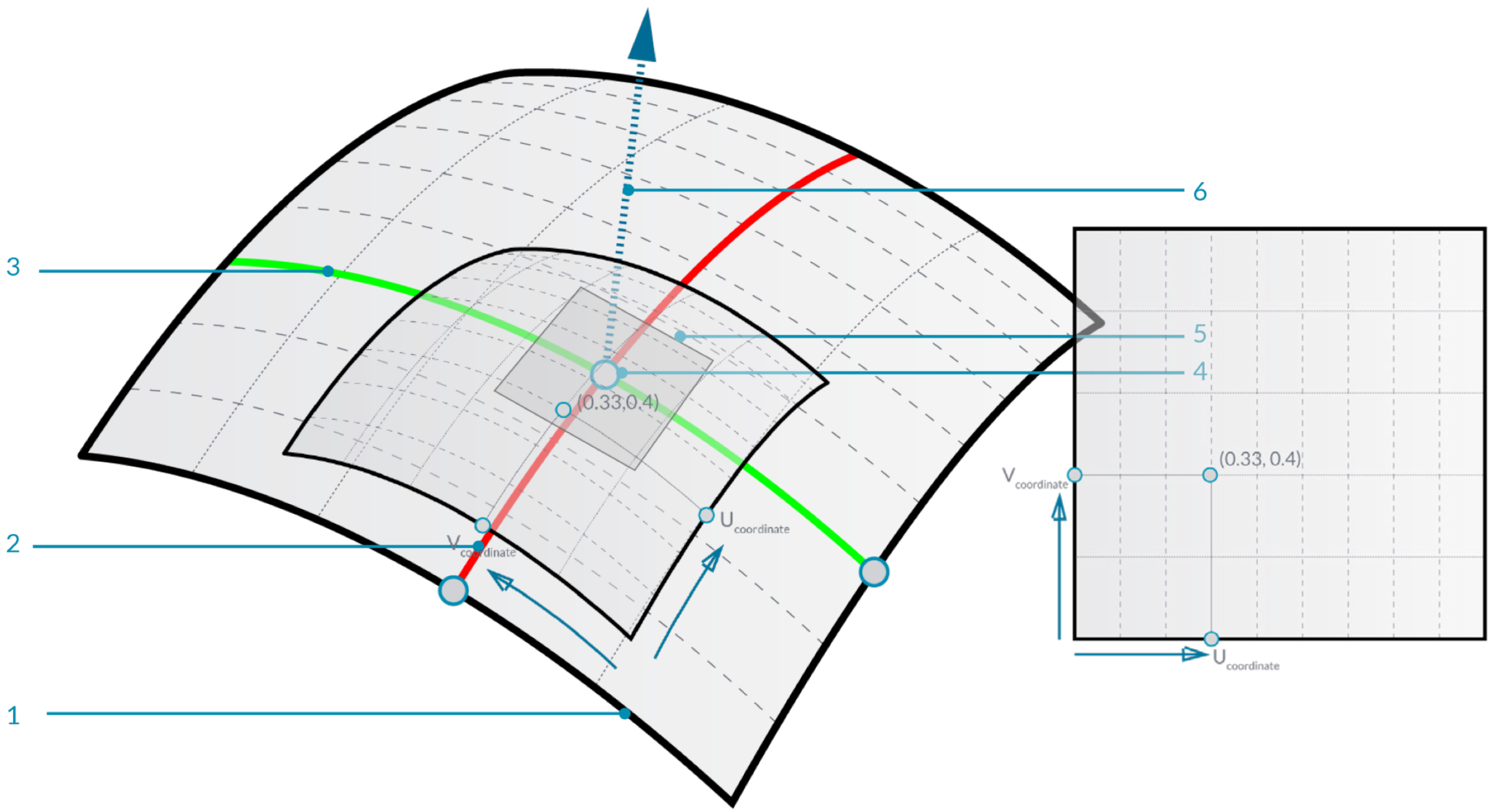

6


7

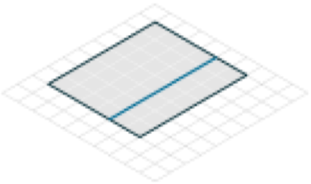
NURBS + Polycurves



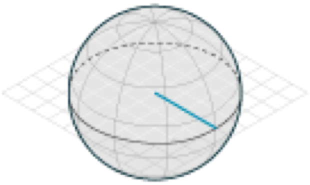
Surface


6
3
5
4
(0.33,0.4)
(0.33, 0.4)
V coordinate
U coordinate
V coordinate
U coordinate
2
1

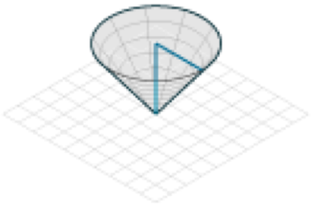
Solids



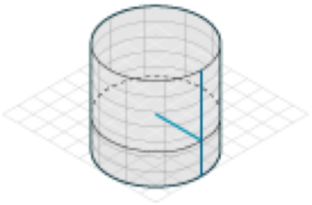
1.



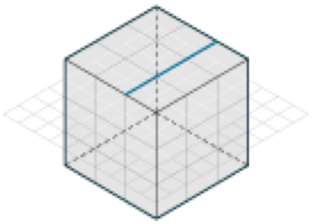
2.



3.

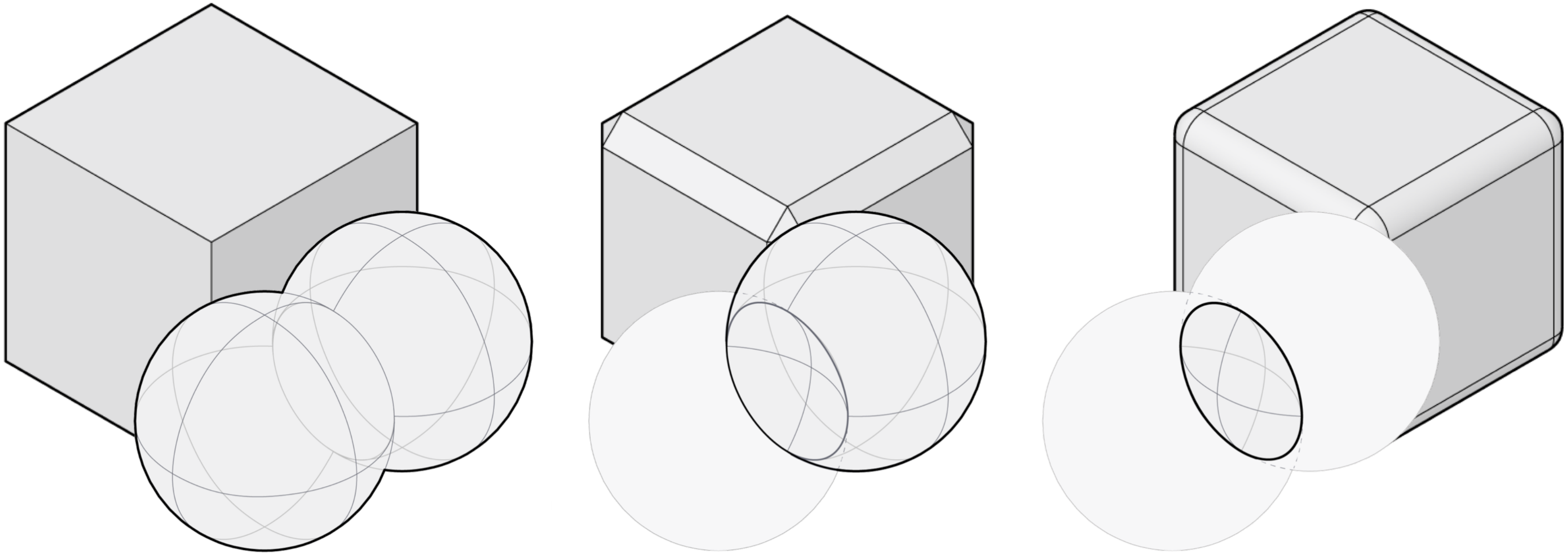


4.



5.

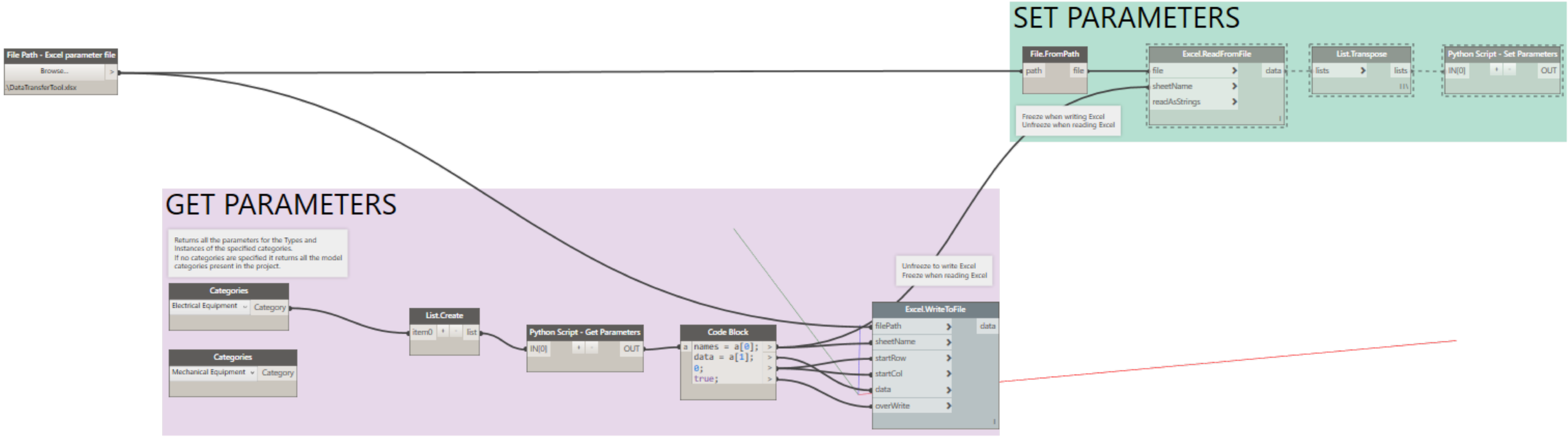
Boolean Operations



Automation Applications

Excel Interoperability

Data Transfer Tool



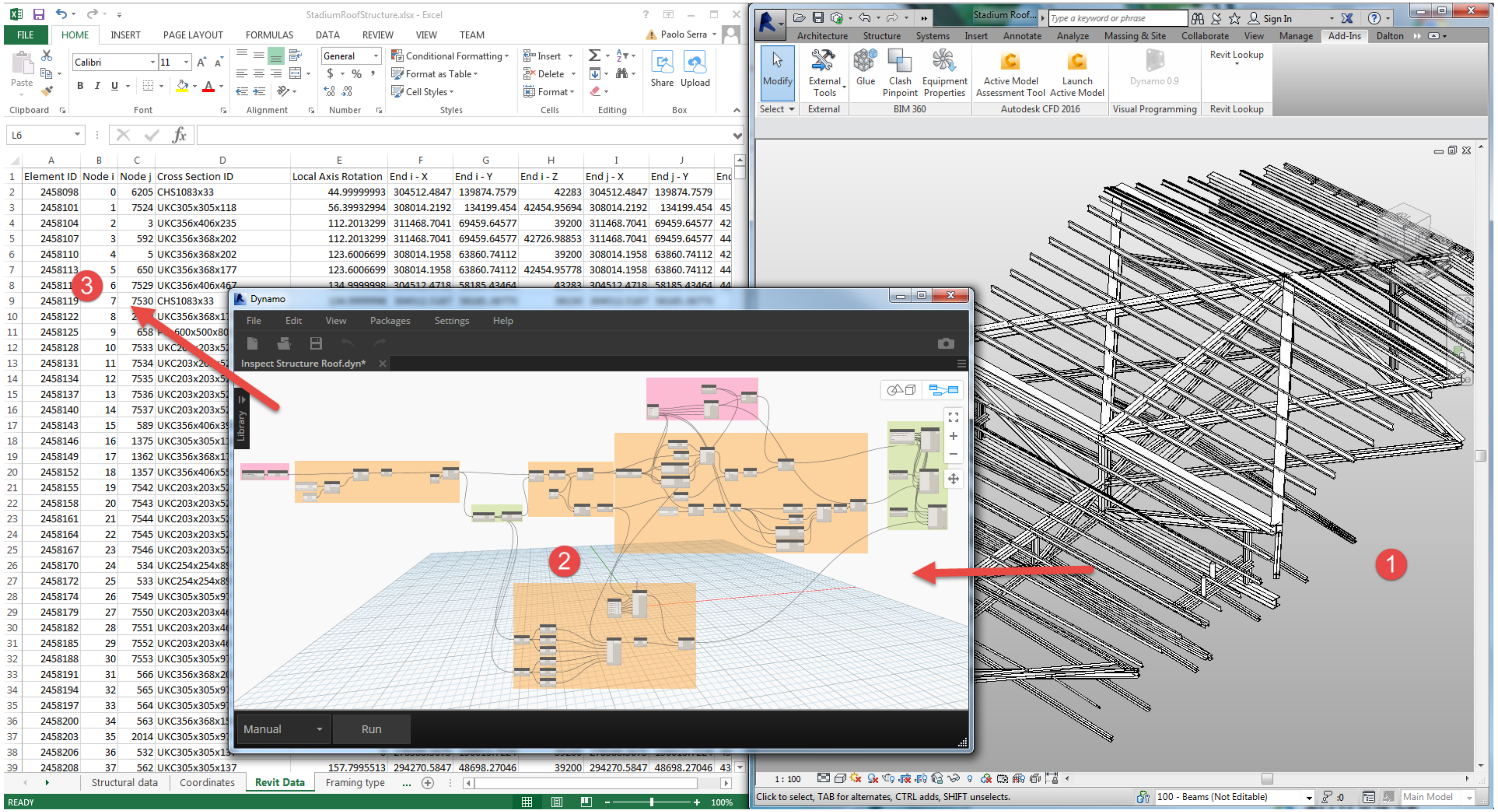
Freeze when writing Excel
Unfreeze when reading Excel

Returns all the parameters for the Types and Instances of the specified categories.
If no categories are specified it returns all the model categories present in the project.

```
names = a[0];
data = a[1];
0;
true;
```

Unfreeze to write Excel
Freeze when reading Excel

Revit – Data Mining



StadiumRoofStructure.xlsx - Excel

L6

	A	B	C	D	E	F	G	H	I	J	
1	Element ID	Node i	Node j	Cross Section ID	Local Axis Rotation	End i - X	End i - Y	End i - Z	End j - X	End j - Y	Enc
2	2458098	0	6205	CHS1083x33	44.99999993	304512.4847	139874.7579	42283	304512.4847	139874.7579	
3	2458101	1	7524	UKC305x305x118	56.39932994	308014.2192	134199.454	42454.95694	308014.2192	134199.454	45
4	2458104	2	3	UKC356x406x235	112.2013299	311468.7041	69459.64577	39200	311468.7041	69459.64577	42
5	2458107	3	592	UKC356x368x202	112.2013299	311468.7041	69459.64577	42726.98853	311468.7041	69459.64577	44
6	2458110	4	5	UKC356x368x202	123.6006699	308014.1958	63860.74112	39200	308014.1958	63860.74112	42
7	2458113	5	650	UKC356x368x177	123.6006699	308014.1958	63860.74112	42454.95778	308014.1958	63860.74112	44
8	245811	6	7529	UKC356x406x467	134.9999998	304512.4718	58185.43464	43283	304512.4718	58185.43464	44
9	2458119	7	7530	CHS1083x33							
10	2458122	8	2	UKC356x368x1							
11	2458125	9	658	P 600x500x80							
12	2458128	10	7533	UKC2 203x5							
13	2458131	11	7534	UKC203x2 5							
14	2458134	12	7535	UKC203x203x5							
15	2458137	13	7536	UKC203x203x5							
16	2458140	14	7537	UKC203x203x5							
17	2458143	15	589	UKC356x406x3							
18	2458146	16	1375	UKC305x305x1							
19	2458149	17	1362	UKC356x368x1							
20	2458152	18	1357	UKC356x406x5							
21	2458155	19	7542	UKC203x203x5							
22	2458158	20	7543	UKC203x203x5							
23	2458161	21	7544	UKC203x203x5							
24	2458164	22	7545	UKC203x203x5							
25	2458167	23	7546	UKC203x203x5							
26	2458170	24	534	UKC254x254x8							
27	2458172	25	533	UKC254x254x8							
28	2458174	26	7549	UKC305x305x9							
29	2458179	27	7550	UKC203x203x4							
30	2458182	28	7551	UKC203x203x4							
31	2458185	29	7552	UKC203x203x4							
32	2458188	30	7553	UKC305x305x9							
33	2458191	31	566	UKC356x368x2							
34	2458194	32	565	UKC305x305x9							
35	2458197	33	564	UKC305x305x9							
36	2458200	34	563	UKC356x368x1							
37	2458203	35	2014	UKC305x305x9							
38	2458206	36	532	UKC305x305x13							
39	2458208	37	562	UKC305x305x137	157.7995513	294270.5847	48698.27046	39200	294270.5847	48698.27046	43

Structural data | Coordinates | **Revit Data** | Framing type

READY

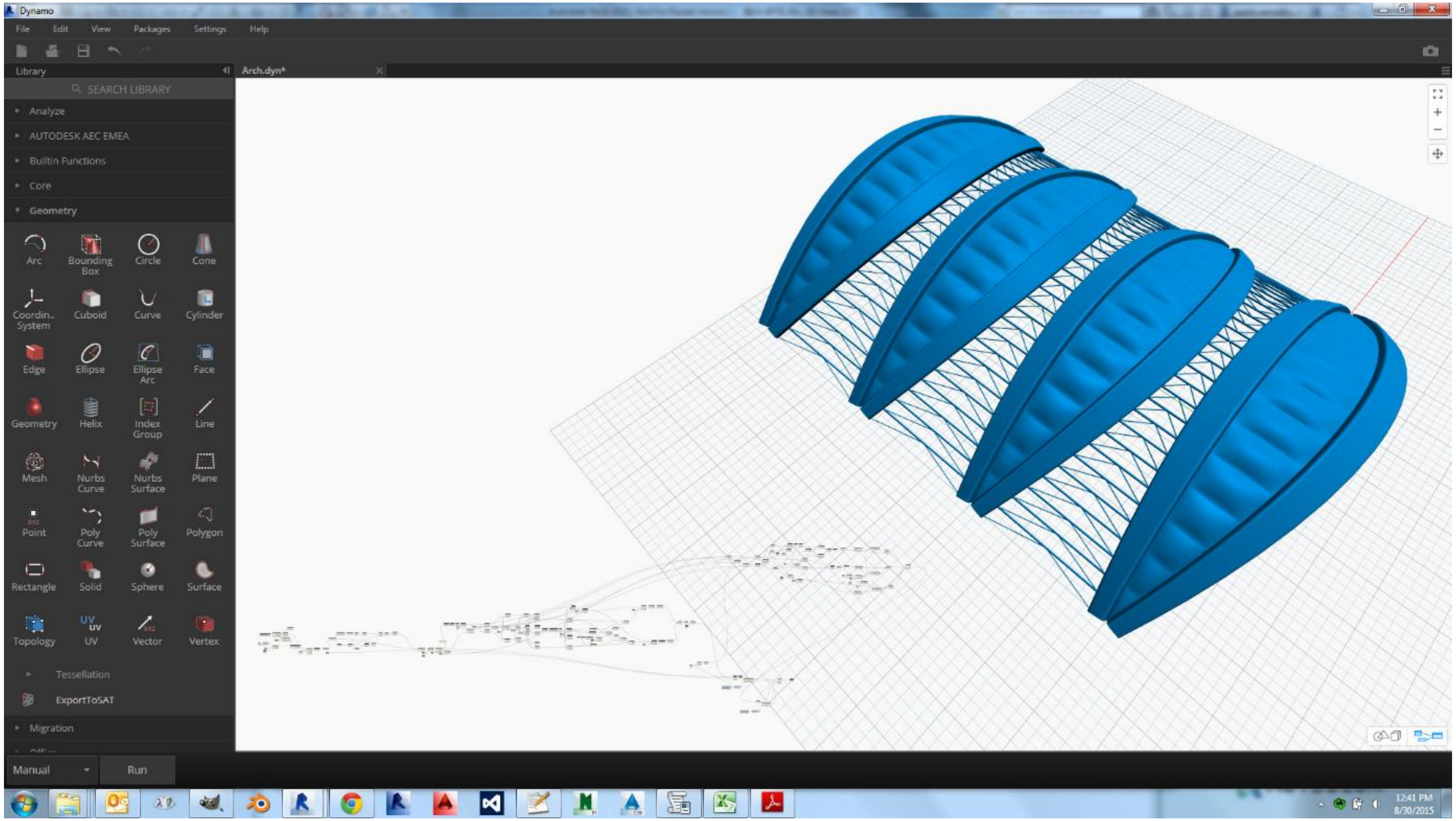
Dynamo

Inspect Structure Roof.dyn*

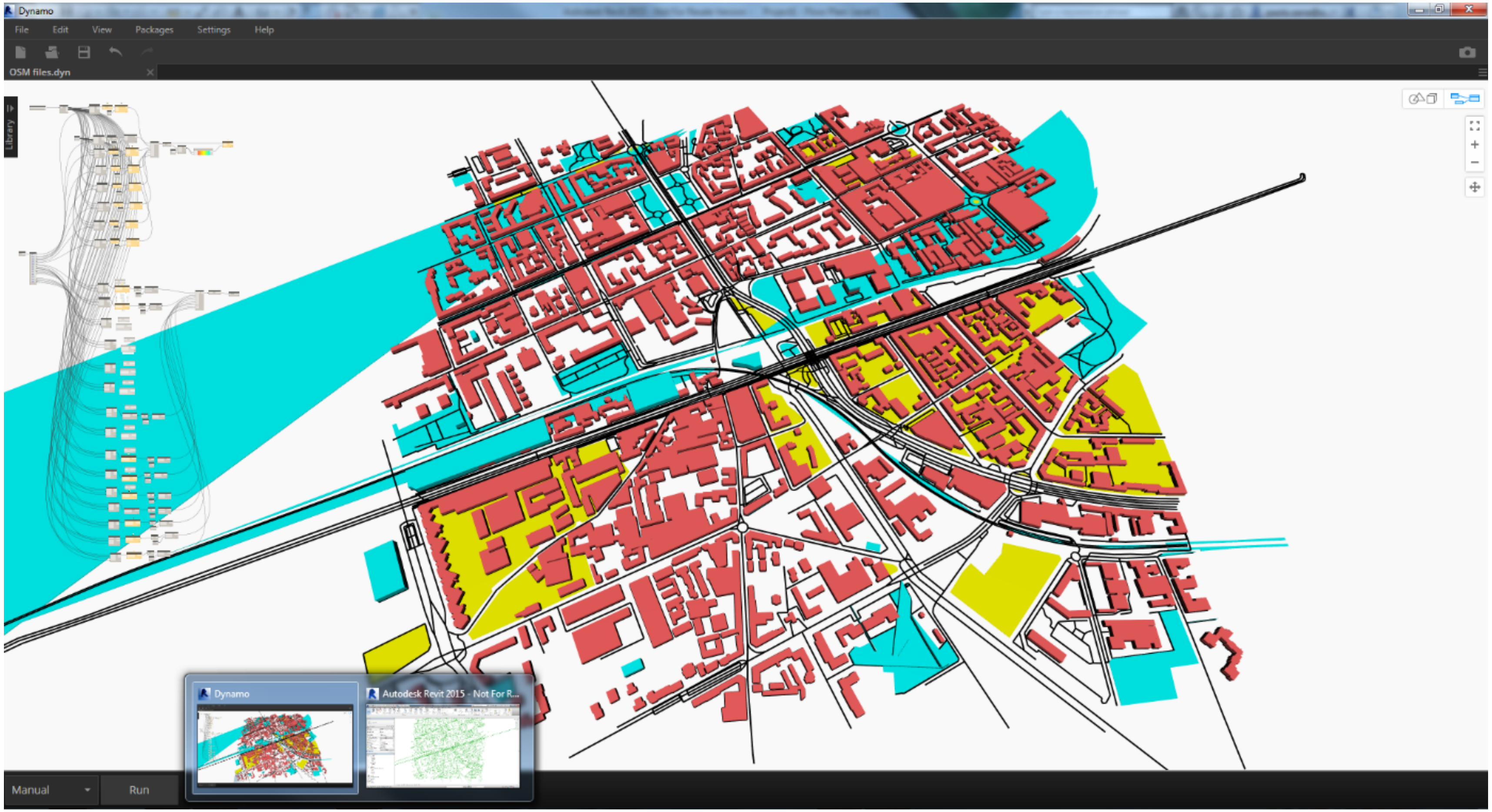
Click to select, TAB for alternates, CTRL adds, SHIFT unselects.

FILE HOME INSERT PAGE LAYOUT FORMULAS DATA REVIEW VIEW TEAM
Paolo Serra
Clipboard Font Alignment Number Styles Cells Editing Box
Conditional Formatting
Format as Table
Cell Styles
Insert Delete Format
Share Upload
File Edit View Packages Settings Help
Library
Manual
Run
Stadium Roof...
Type a keyword or phrase
Sign In
Architecture Structure Systems Insert Annotate Analyze Massing & Site Collaborate View Manage Add-Ins Dalton
Modify
External Tools
Glue
Clash Pinpoint
Equipment Properties
Active Model Assessment Tool
Launch Active Model
Dynamo 0.9
Revit Lookup
Select External BIM 360 Autodesk CFD 2016 Visual Programming Revit Lookup
1 : 100
100 - Beams (Not Editable)
Main Model
1
2
3

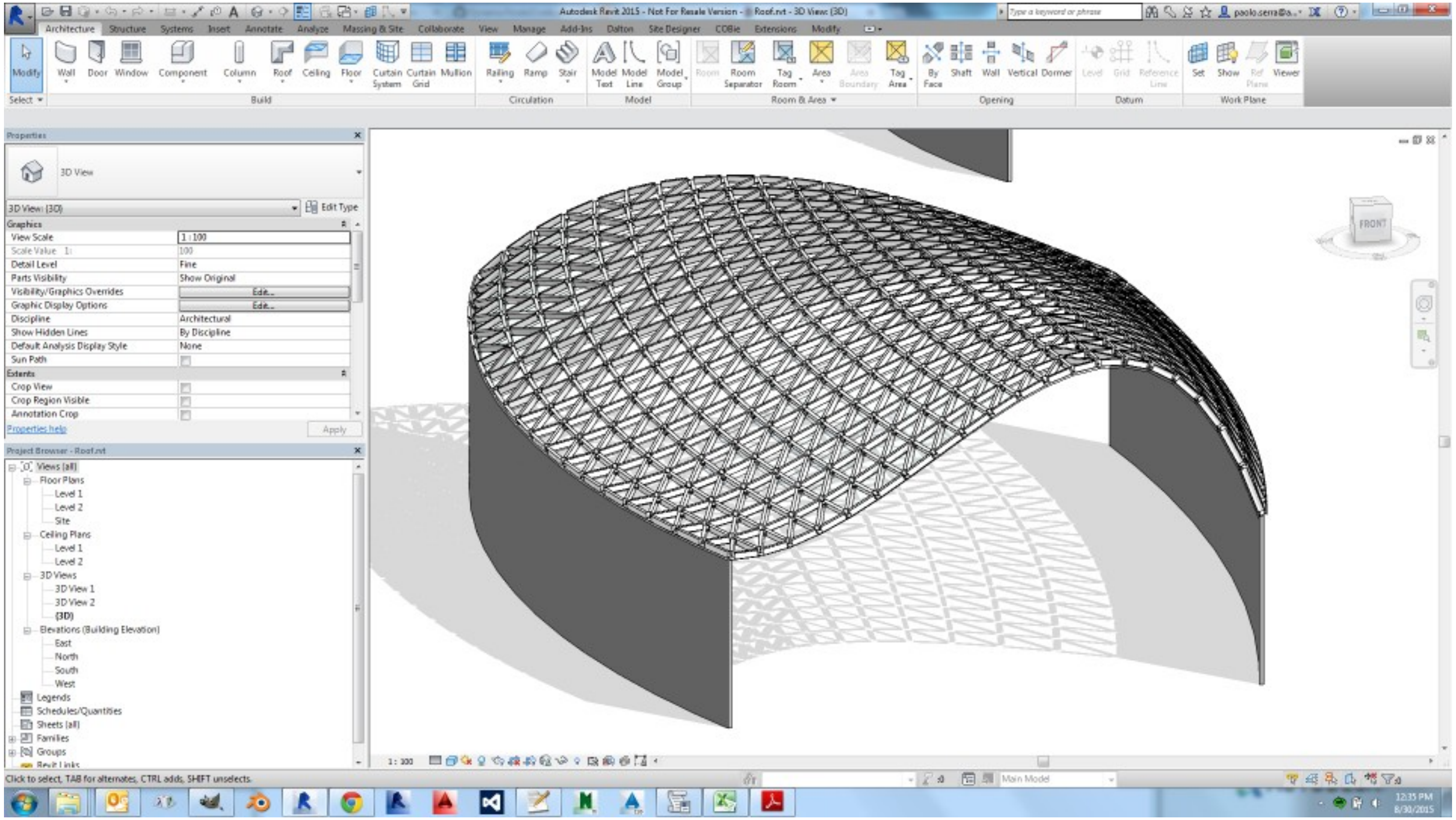
Computational Design



Access Open Street Map Data



Revit – Model Authoring



Autodesk Revit 2015 - Not For Resale Version - Roof.rvt - 3D View: {3D}

Properties

3D View

3D View: {3D} Edit Type

Graphics	
View Scale	1 : 100
Scale Value 1:	100
Detail Level	Fine
Parts Visibility	Show Original
Visibility/Graphics Overrides	Edit...
Graphic Display Options	Edit...
Discipline	Architectural
Show Hidden Lines	By Discipline
Default Analysis Display Style	None
Sun Path	□
Extents	
Crop View	□
Crop Region Visible	□
Annotation Crop	□

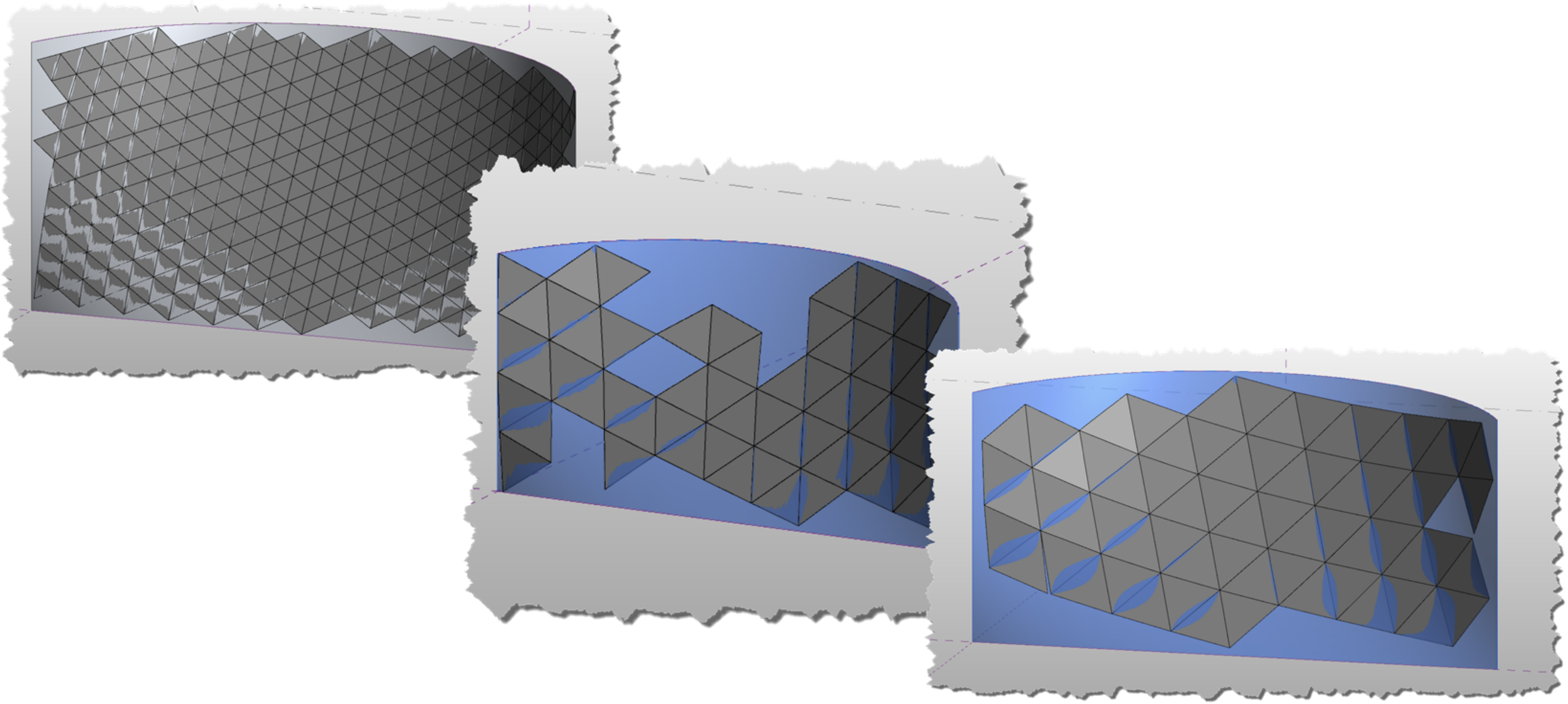
Properties help Apply

Project Browser - Roof.rvt

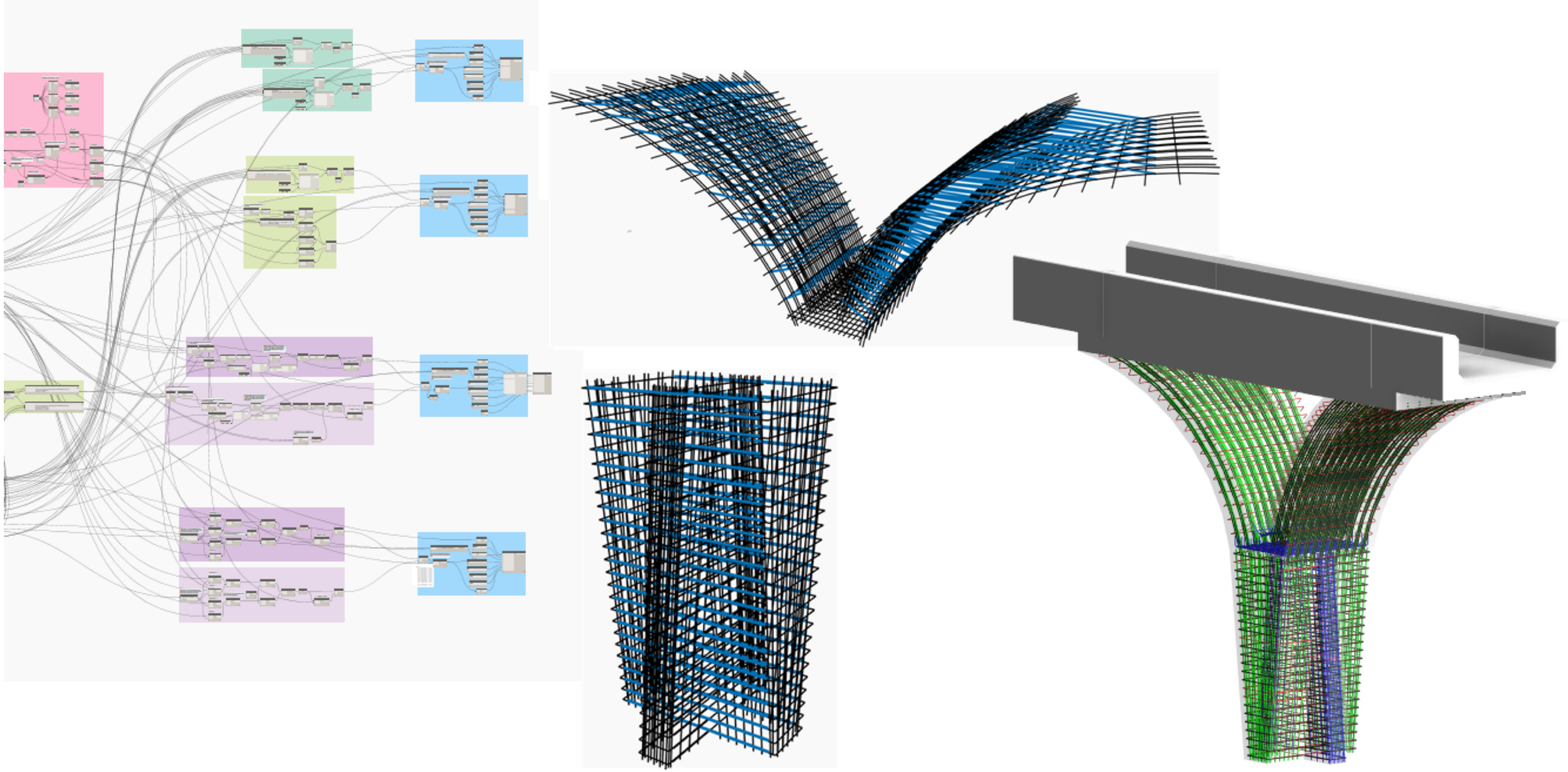
- Views (all)
  - Floor Plans
    - Level 1
    - Level 2
    - Site
  - Ceiling Plans
    - Level 1
    - Level 2
  - 3D Views
    - 3D View 1
    - 3D View 2
    - {3D}
  - Elevations (Building Elevation)
    - East
    - North
    - South
    - West
- Legends
- Schedules/Quantities
- Sheets (all)
- Families
- Groups
- Revit Links

Click to select, TAB for alternates, CTRL adds, SHIFT unselects.

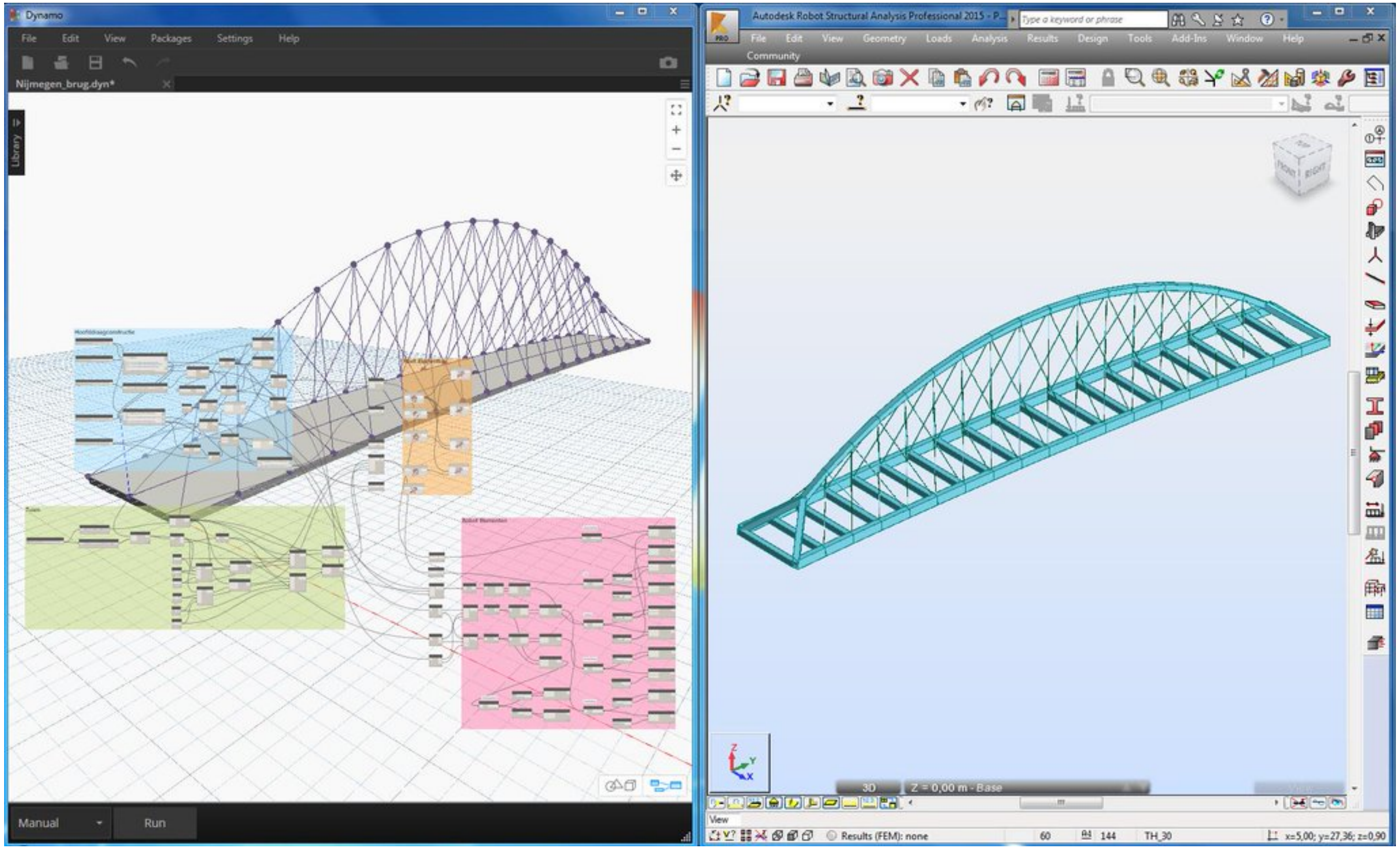
Revit – Fixed Dimensions Panels on Surface



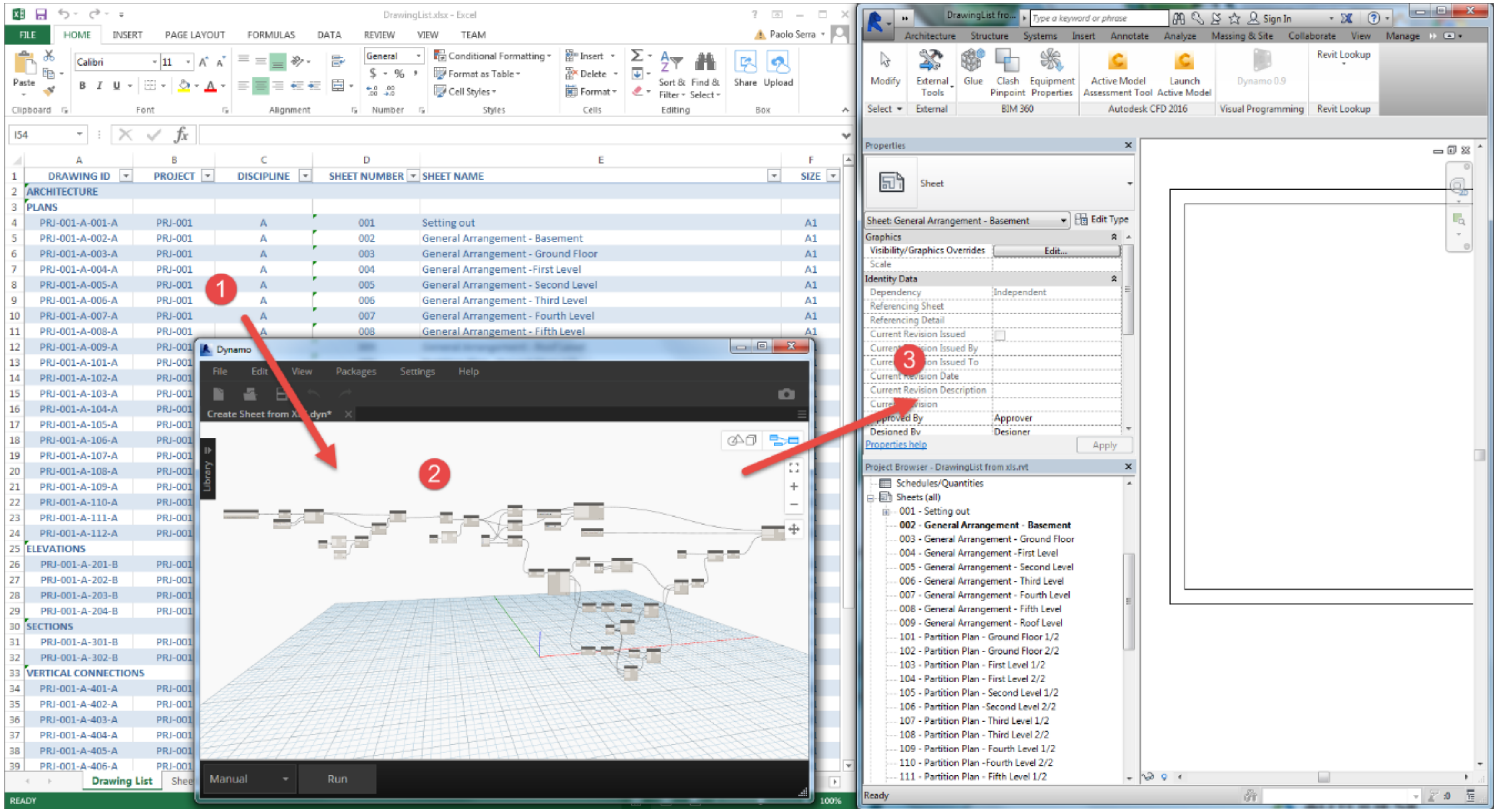
Revit – Rebar (2016+)



Robot Structural Analysis – React Struct



Revit – Drawing Production



	A	B	C	D	E	F
1	DRAWING ID	PROJECT	DISCIPLINE	SHEET NUMBER	SHEET NAME	SIZE
2	ARCHITECTURE					
3	PLANS					
4	PRJ-001-A-001-A	PRJ-001	A	001	Setting out	A1
5	PRJ-001-A-002-A	PRJ-001	A	002	General Arrangement - Basement	A1
6	PRJ-001-A-003-A	PRJ-001	A	003	General Arrangement - Ground Floor	A1
7	PRJ-001-A-004-A	PRJ-001	A	004	General Arrangement -First Level	A1
8	PRJ-001-A-005-A	PRJ-001	A	005	General Arrangement - Second Level	A1
9	PRJ-001-A-006-A	PRJ-001	A	006	General Arrangement - Third Level	A1
10	PRJ-001-A-007-A	PRJ-001	A	007	General Arrangement - Fourth Level	A1
11	PRJ-001-A-008-A	PRJ-001	A	008	General Arrangement - Fifth Level	A1
12	PRJ-001-A-009-A	PRJ-001				
13	PRJ-001-A-101-A	PRJ-001				
14	PRJ-001-A-102-A	PRJ-001				
15	PRJ-001-A-103-A	PRJ-001				
16	PRJ-001-A-104-A	PRJ-001				
17	PRJ-001-A-105-A	PRJ-001				
18	PRJ-001-A-106-A	PRJ-001				
19	PRJ-001-A-107-A	PRJ-001				
20	PRJ-001-A-108-A	PRJ-001				
21	PRJ-001-A-109-A	PRJ-001				
22	PRJ-001-A-110-A	PRJ-001				
23	PRJ-001-A-111-A	PRJ-001				
24	PRJ-001-A-112-A	PRJ-001				
25	ELEVATIONS					
26	PRJ-001-A-201-B	PRJ-001				
27	PRJ-001-A-202-B	PRJ-001				
28	PRJ-001-A-203-B	PRJ-001				
29	PRJ-001-A-204-B	PRJ-001				
30	SECTIONS					
31	PRJ-001-A-301-B	PRJ-001				
32	PRJ-001-A-302-B	PRJ-001				
33	VERTICAL CONNECTIONS					
34	PRJ-001-A-401-A	PRJ-001				
35	PRJ-001-A-402-A	PRJ-001				
36	PRJ-001-A-403-A	PRJ-001				
37	PRJ-001-A-404-A	PRJ-001				
38	PRJ-001-A-405-A	PRJ-001				
39	PRJ-001-A-406-A	PRJ-001				

Drawing List | Shee

Properties

Sheet

Sheet: General Arrangement - Basement | Edit Type

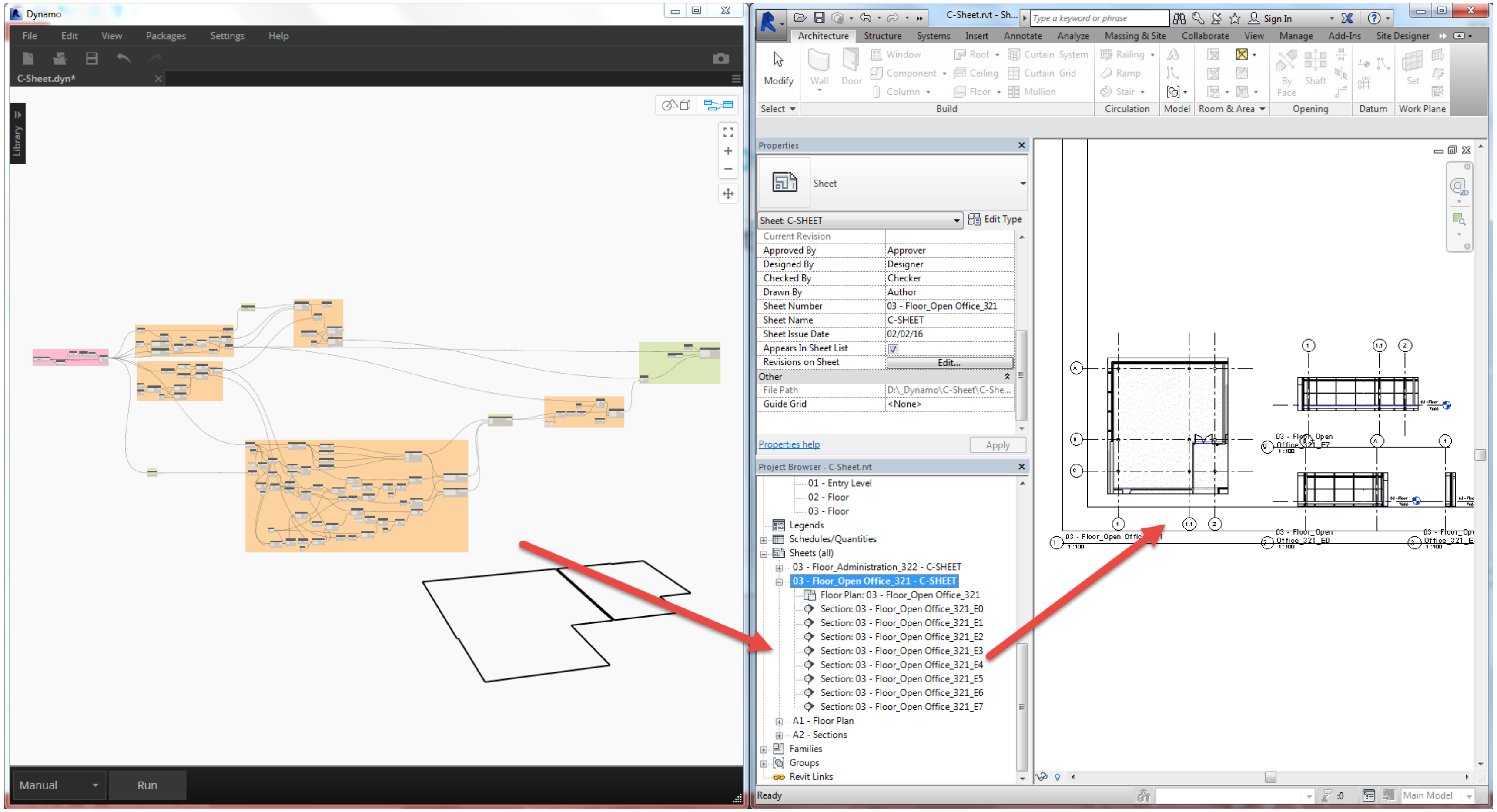
Graphics	
Visibility/Graphics Overrides	Edit...
Scale	
Identity Data	
Dependency	Independent
Referencing Sheet	
Referencing Detail	
Current Revision Issued	□
Curren ion Issued By	
Curre ion Issued To	
Current evision Date	
Current Revision Description	
Curre ision	
pproved By	Approver
Designed By	Designer

Properties help | Apply

Project Browser - DrawingList from xls.rvt

- Schedules/Quantities
- Sheets (all)
  - 001 - Setting out
  - **002 - General Arrangement - Basement**
  - 003 - General Arrangement - Ground Floor
  - 004 - General Arrangement -First Level
  - 005 - General Arrangement - Second Level
  - 006 - General Arrangement - Third Level
  - 007 - General Arrangement - Fourth Level
  - 008 - General Arrangement - Fifth Level
  - 009 - General Arrangement - Roof Level
  - 101 - Partition Plan - Ground Floor 1/2
  - 102 - Partition Plan - Ground Floor 2/2
  - 103 - Partition Plan - First Level 1/2
  - 104 - Partition Plan - First Level 2/2
  - 105 - Partition Plan - Second Level 1/2
  - 106 - Partition Plan -Second Level 2/2
  - 107 - Partition Plan - Third Level 1/2
  - 108 - Partition Plan - Third Level 2/2
  - 109 - Partition Plan - Fourth Level 1/2
  - 110 - Partition Plan -Fourth Level 2/2
  - 111 - Partition Plan - Fifth Level 1/2

Revit – Drawing Production



Properties

Sheet

Sheet: C-SHEET | Edit Type

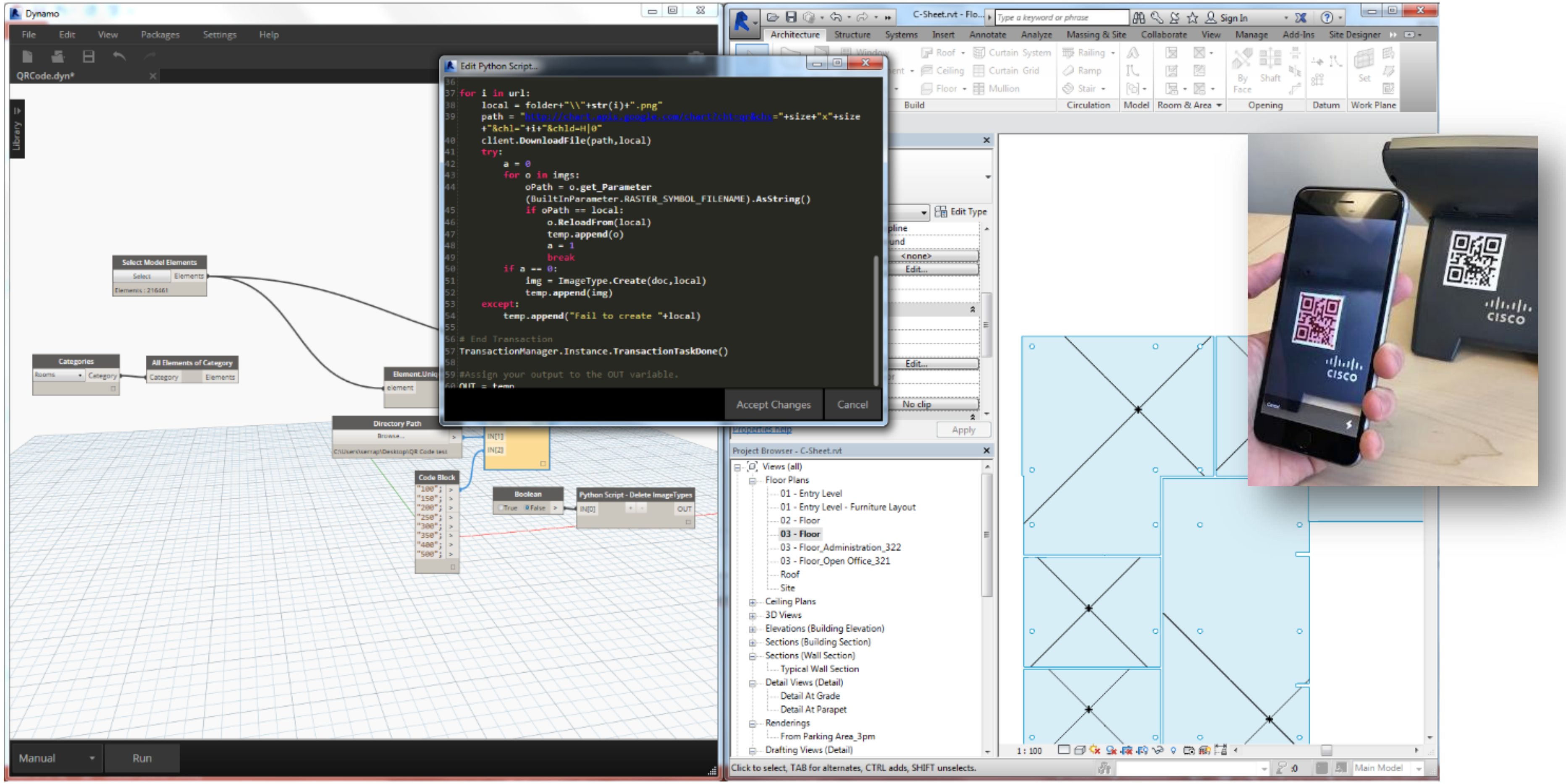
Current Revision	
Approved By	Approver
Designed By	Designer
Checked By	Checker
Drawn By	Author
Sheet Number	03 - Floor_Open Office_321
Sheet Name	C-SHEET
Sheet Issue Date	02/02/16
Appears In Sheet List	☑
Revisions on Sheet	Edit...
Other	
File Path	D:_Dynamo\C-Sheet\C-She...
Guide Grid	<None>

Properties help | Apply

Project Browser - C-Sheet.rvt

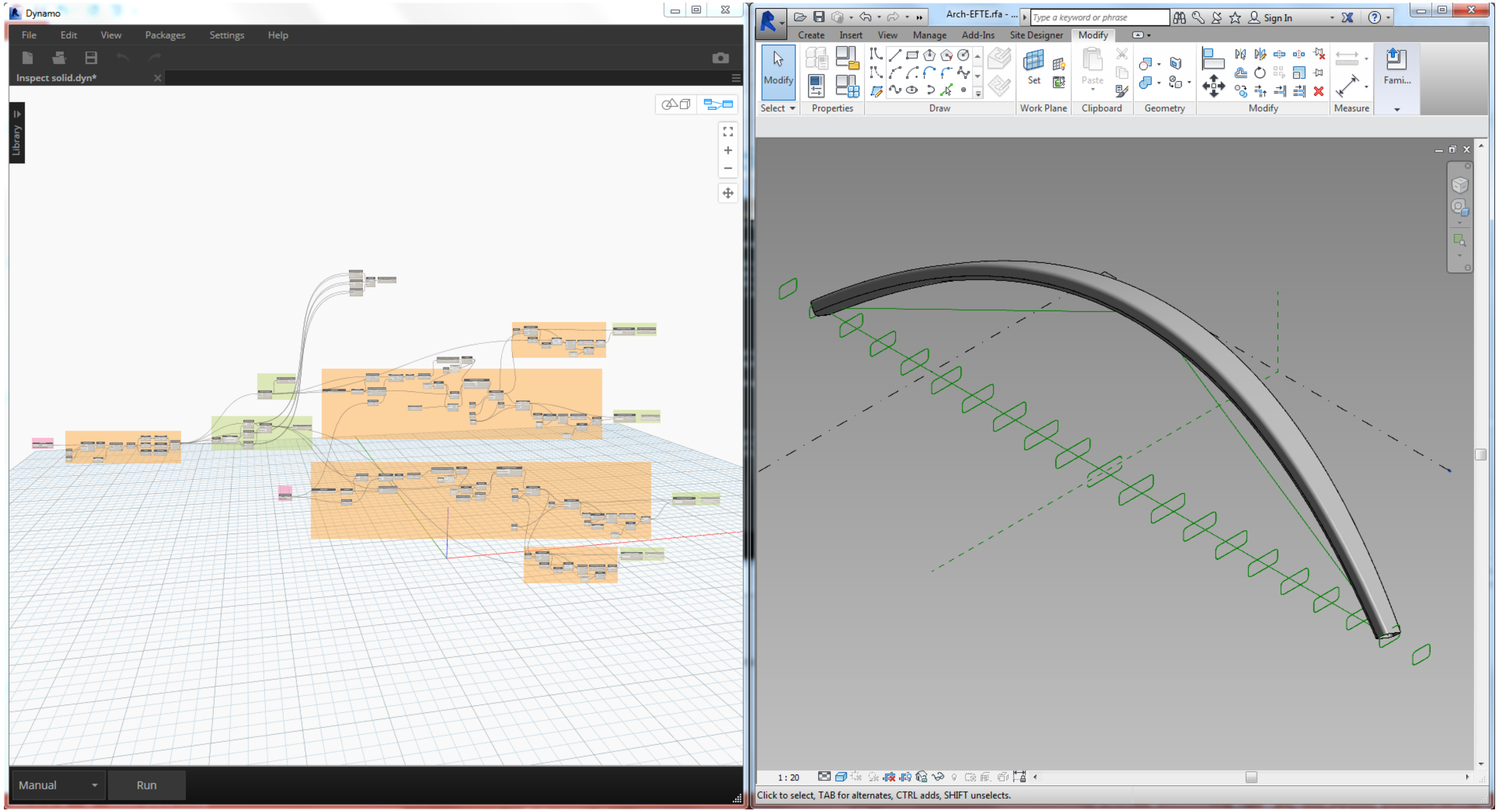
- 01 - Entry Level
- 02 - Floor
- 03 - Floor
- Legends
- Schedules/Quantities
- Sheets (all)
  - 03 - Floor_Administration_322 - C-SHEET
  - 03 - Floor_Open Office_321 - C-SHEET
    - Floor Plan: 03 - Floor_Open Office_321
    - Section: 03 - Floor_Open Office_321_E0
    - Section: 03 - Floor_Open Office_321_E1
    - Section: 03 - Floor_Open Office_321_E2
    - Section: 03 - Floor_Open Office_321_E3
    - Section: 03 - Floor_Open Office_321_E4
    - Section: 03 - Floor_Open Office_321_E5
    - Section: 03 - Floor_Open Office_321_E6
    - Section: 03 - Floor_Open Office_321_E7
  - A1 - Floor Plan
  - A2 - Sections
- Families
- Groups
- Revit Links

Revit – QR Codes for Asset Management



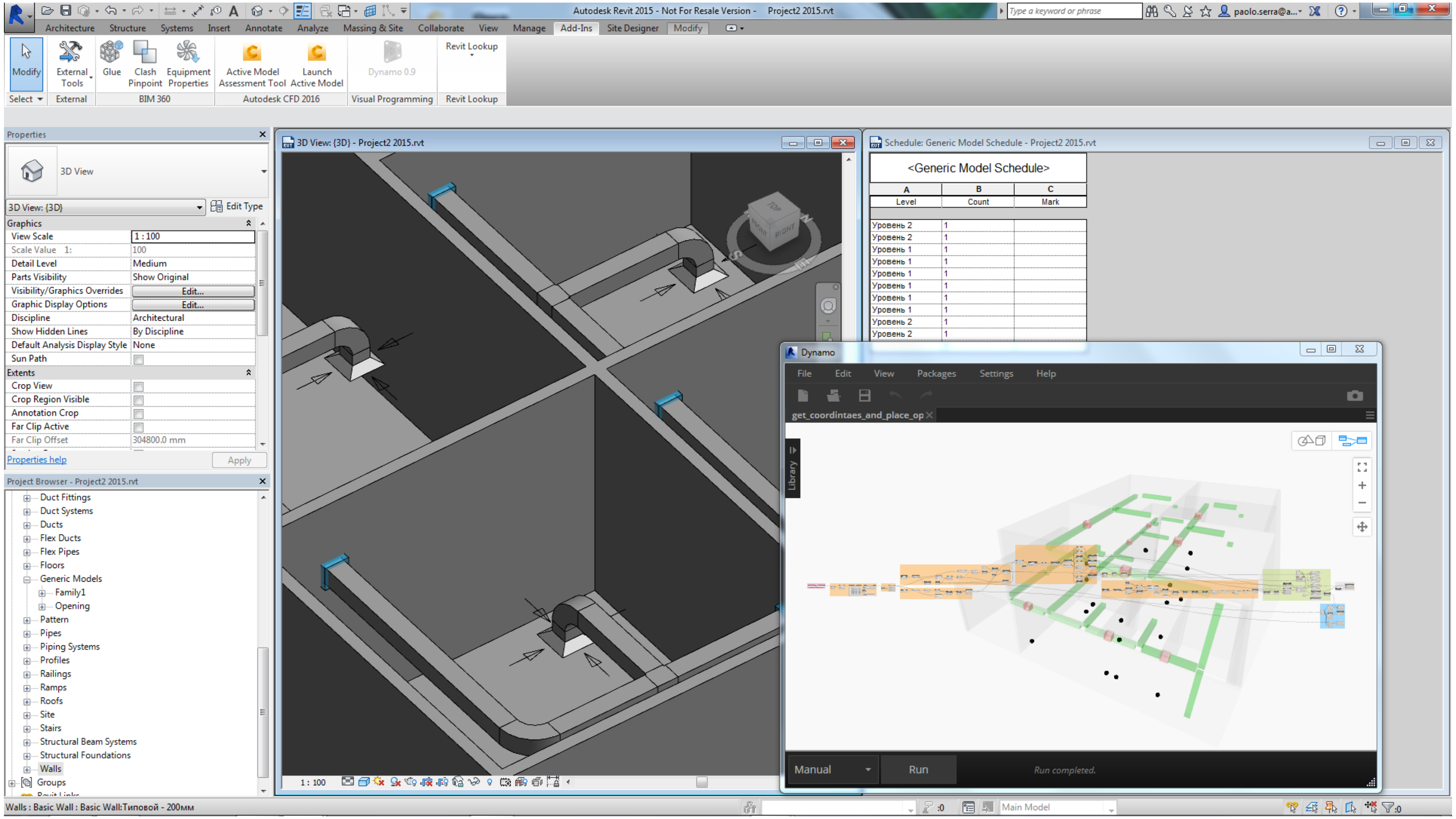
```
for i in url:
    local = folder+"\\"+str(i)+".png"
    path = "http://chart.apis.google.com/chart?cht=qr&chs="+size+"x"+size
    +"&chl="+i+"&chld=H|0"
    client.DownloadFile(path,local)
    try:
        a = 0
        for o in imgs:
            oPath = o.get_Parameter
            (BuiltInParameter.RASTER_SYMBOL_FILENAME).AsString()
            if oPath == local:
                o.ReloadFrom(local)
                temp.append(o)
                a = 1
                break
        if a == 0:
            img = ImageType.Create(doc,local)
            temp.append(img)
    except:
        temp.append("Fail to create "+local)

# End Transaction
TransactionManager.Instance.TransactionTaskDone()

#Assign your output to the OUT variable.
OUT = temp
```

Shape Analysis



Click to select, TAB for alternates, CTRL adds, SHIFT unselects.

3D Coordination



3D View: {3D}

Graphics	
View Scale	1 : 100
Scale Value 1:	100
Detail Level	Medium
Parts Visibility	Show Original
Visibility/Graphics Overrides	Edit...
Graphic Display Options	Edit...
Discipline	Architectural
Show Hidden Lines	By Discipline
Default Analysis Display Style	None
Sun Path	□
Extents	
Crop View	□
Crop Region Visible	□
Annotation Crop	□
Far Clip Active	□
Far Clip Offset	304800.0 mm

<Generic Model Schedule>

A	B	C
Level	Count	Mark
Уровень 2	1	
Уровень 2	1	
Уровень 1	1	
Уровень 1	1	
Уровень 1	1	
Уровень 1	1	
Уровень 1	1	
Уровень 1	1	
Уровень 2	1	
Уровень 2	1	

- Duct Fittings
- Duct Systems
- Ducts
- Flex Ducts
- Flex Pipes
- Floors
- Generic Models
  - Family1
  - Opening
- Pattern
- Pipes
- Piping Systems
- Profiles
- Railings
- Ramps
- Roofs
- Site
- Stairs
- Structural Beam Systems
- Structural Foundations
- Walls
- Groups

get_coordintaes_and_place_op

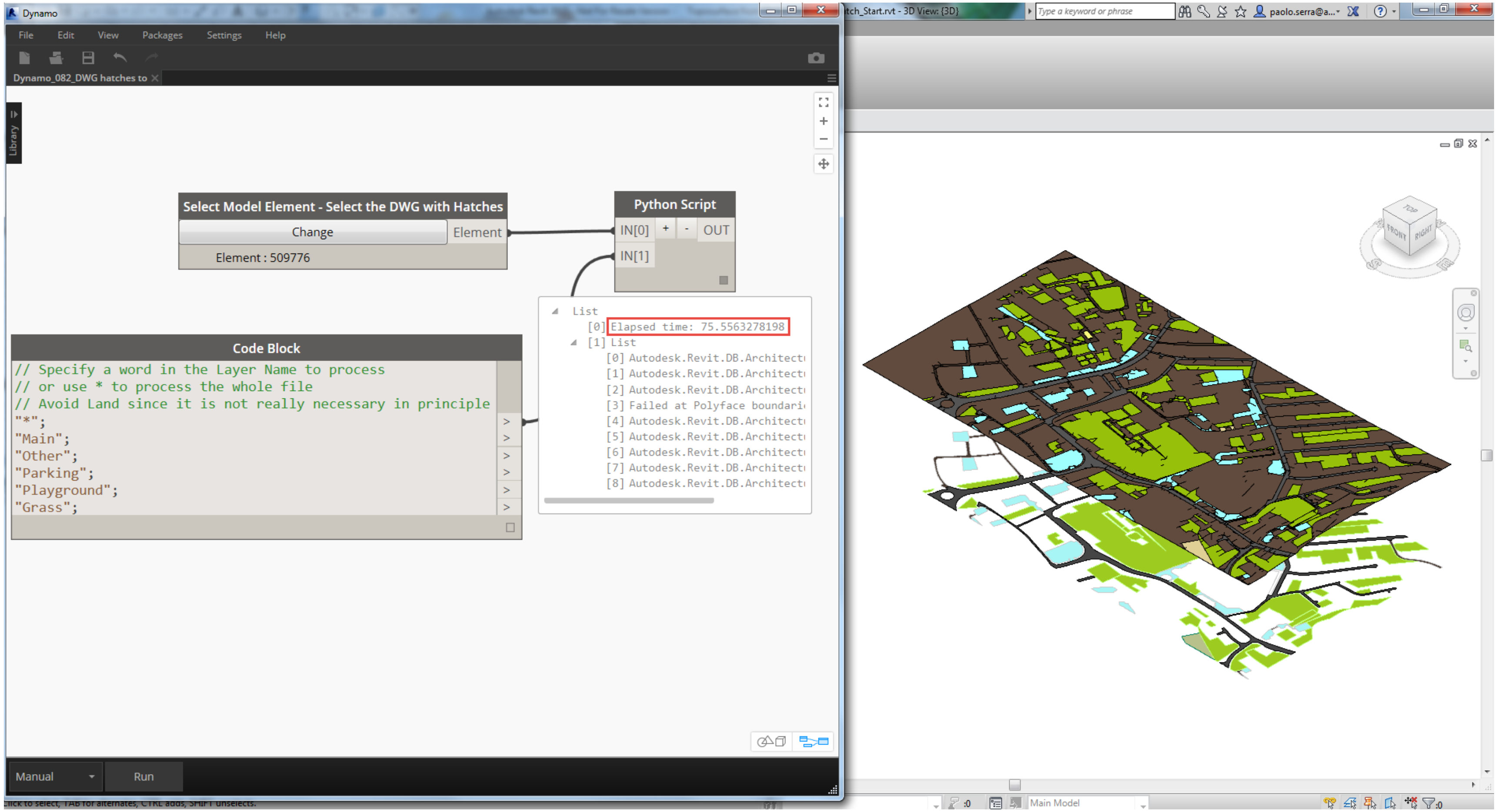
Manual

Run

Run completed.

Walls : Basic Wall : Basic Wall:Типовой - 200мм

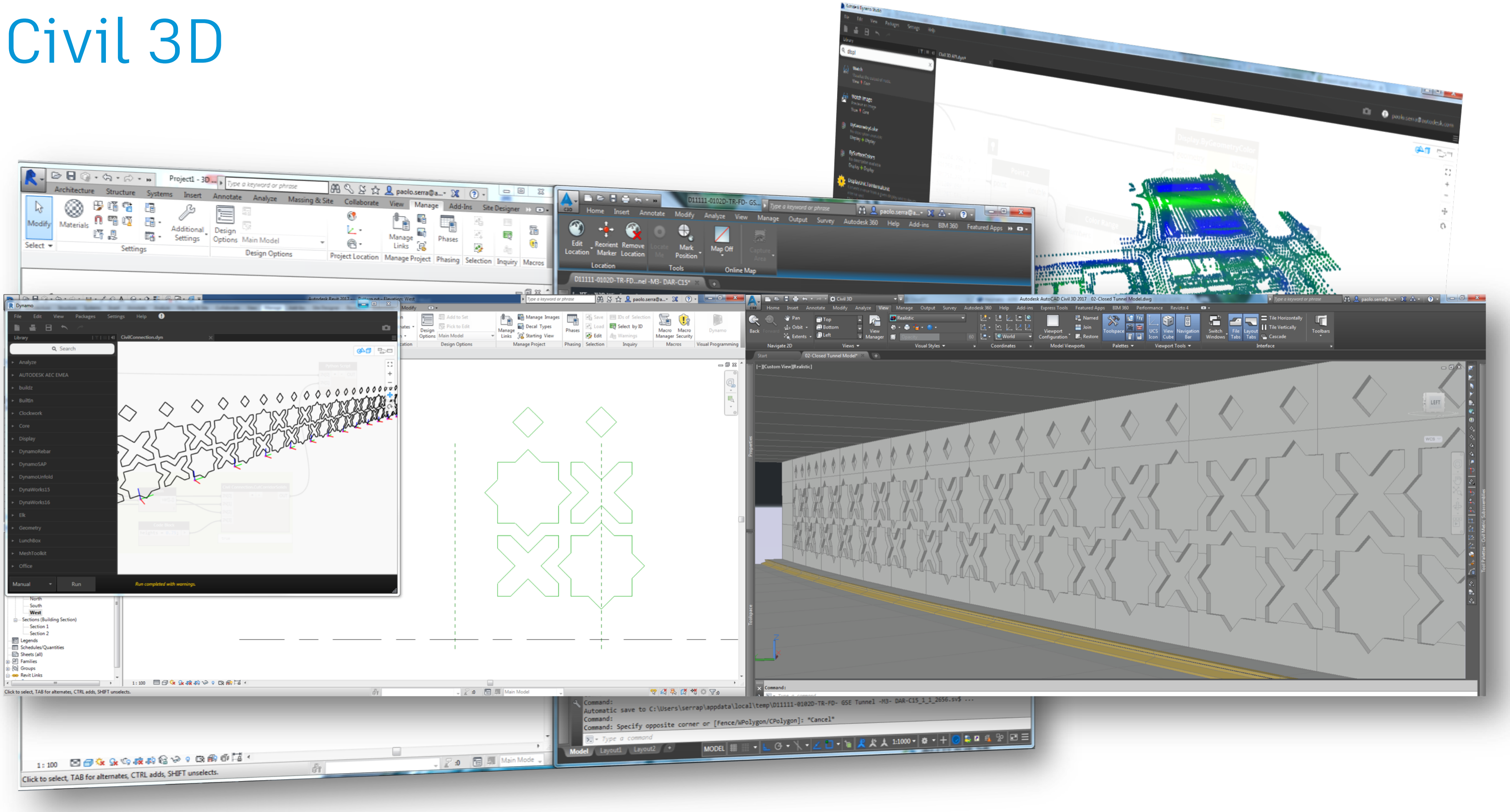
AutoCAD-Revit



```
// Specify a word in the Layer Name to process
// or use * to process the whole file
// Avoid Land since it is not really necessary in principle
"*";
"Main";
"Other";
"Parking";
"Playground";
"Grass";
```

```
List
  [0] Elapsed time: 75.5563278198
  [1] List
    [0] Autodesk.Revit.DB.Architect
    [1] Autodesk.Revit.DB.Architect
    [2] Autodesk.Revit.DB.Architect
    [3] Failed at Polyface boundari
    [4] Autodesk.Revit.DB.Architect
    [5] Autodesk.Revit.DB.Architect
    [6] Autodesk.Revit.DB.Architect
    [7] Autodesk.Revit.DB.Architect
    [8] Autodesk.Revit.DB.Architect
```

Civil 3D



Run completed with warnings.

Click to select, TAB for alternates, CTRL adds, SHIFT unselects.

```
Command:
Automatic save to C:\Users\serrap\appdata\local\temp\D11111-0102D-TR-FD- GSE Tunnel -M3- DAR-C15_1_1_2656.sv$ ...
Command:
Command: Specify opposite corner or [Fence/WPolygon/CPolygon]: *Cancel*
```

Dynamo for Revit

Select Revit Elements

- Pick an object (instance, face, edge)

- Window select many instances at once

- Select All Elements By a common characteristic (element type, category, family type, level)

Get Parameter Value By Name

- Revit objects (families, views, family types, etc.) are all ELEMENTS and they all have a specific set of PARAMETERS

- A parameter is a container with a name and a value

- The value the user is presented may be different from the one stored internally in Revit (use Revit Lookup add-in for more detail)

Set Parameter By Name

- Each parameter has a specific Storage Type and a specific range of values that can actually be assigned

- Use the Revit Lookup and the Revit SDK for more details

- When using Dynamo you don’t need to worry about the unit conversion for the parameter value

Create Revit Elements

- The geometry entities in Dynamo can define the location or the geometrical definition of Revit objects (points for family instances, curves for walls and structural framings, closed loops for slabs, etc.)
- Once created the elements are persistent for the Revit session (even closing and reopening Dynamo) but closing Revit will definitely break the tie
- First create the element, then change its parameters

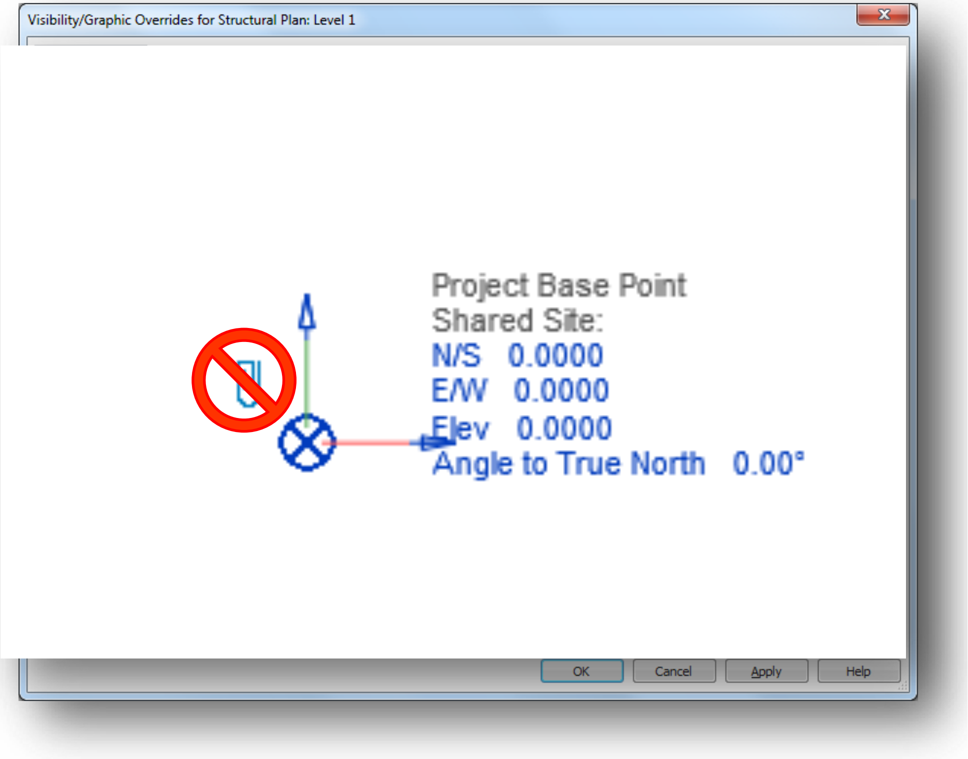
Revit Coordinate Systems

Revit Coordinate Systems

- Implicit coordinate systems
    - Internal (not visible)
    - Local Coordinate System -> Project Base Point
    - World Coordinate System -> Survey Point

Project Base Point

- Visible under Site category
- It is always parallel to the screen sides
- Contains an angle value
- <u>**NEVER**</u> unclip



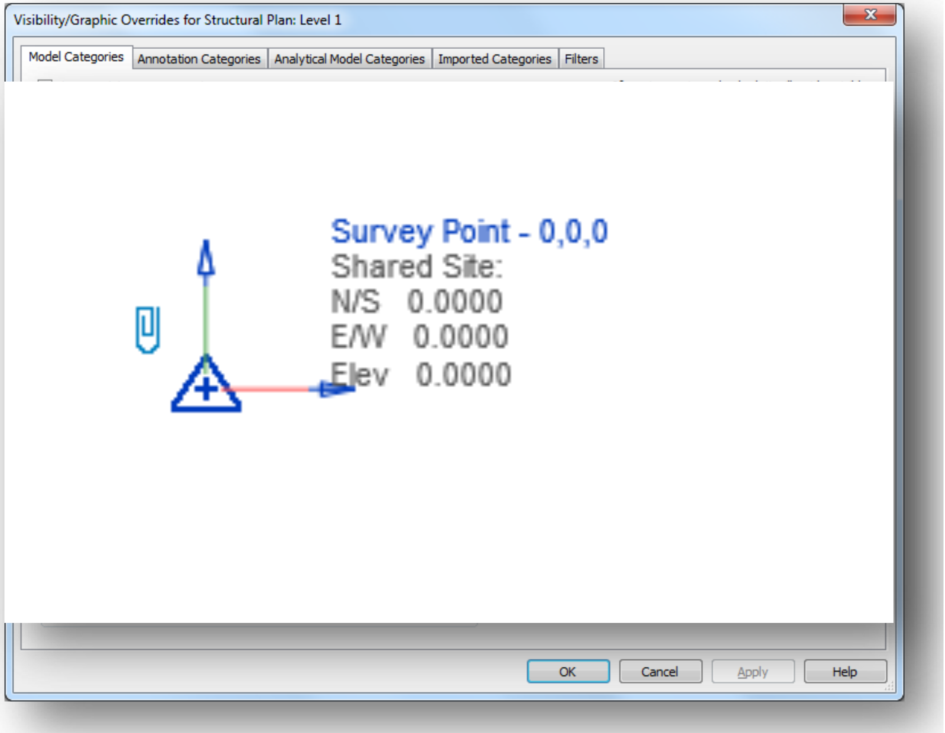
Project Base Point
Shared Site:
N/S 0.0000
E/W 0.0000
Elev 0.0000
Angle to True North 0.00°

Survey Point

- Visible under Site category

- It is always parallel to the World Coordinate System

- If it remains clipped identify the Origin of the WCS


Visibility/Graphic Overrides for Structural Plan: Level 1
Model Categories
Annotation Categories
Analytical Model Categories
Imported Categories
Filters
OK
Cancel
Apply
Help

Survey Point - 0,0,0
Shared Site:
N/S 0.0000
E/W 0.0000
Elev 0.0000

Internal Coordinate System

- All the objects created through the API or Dynamo are referring to the Internal Coordinate System

- By Default the Project Base Point is sitting on top of the Internal origin, that is why the PBP should never be unclipped

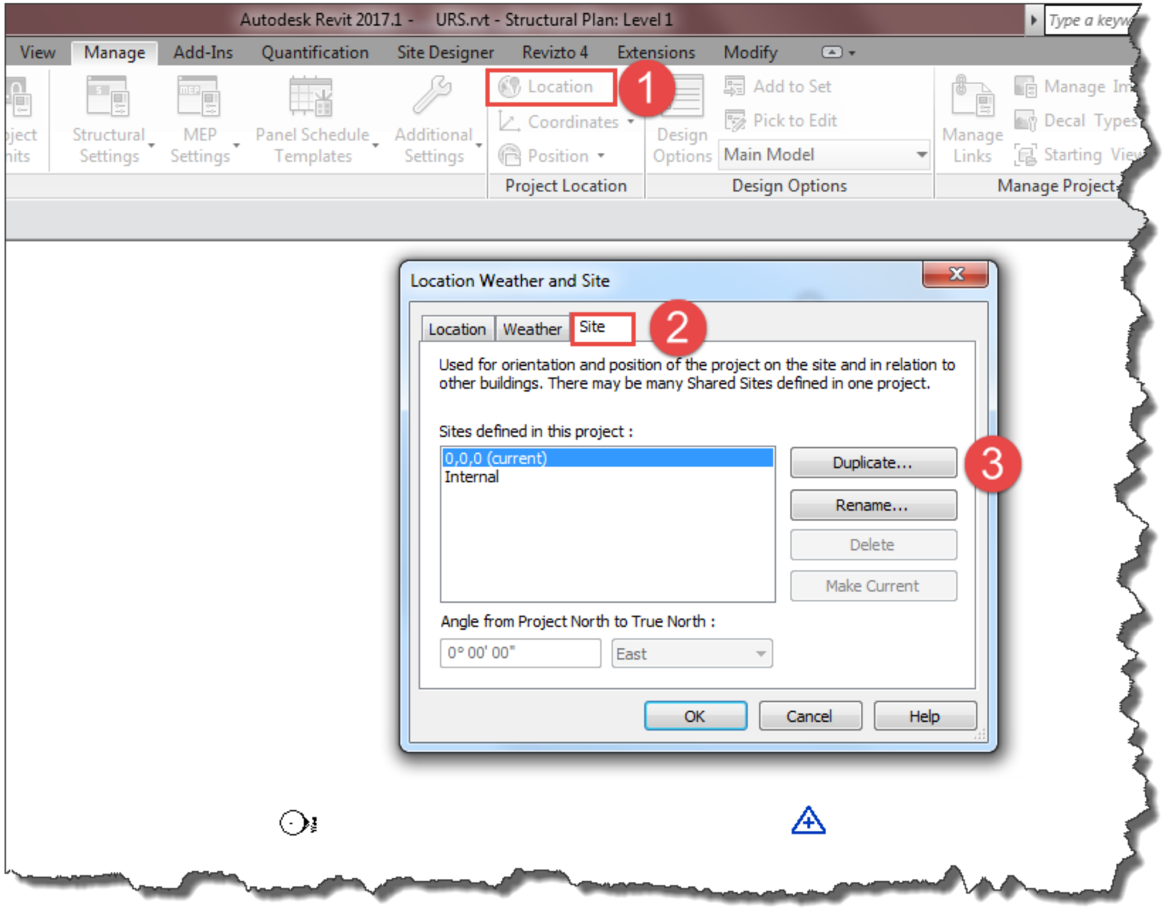
Shared Sites

- It is possible to define multiple site locations and orientations for the same project document

- These are called shared sites or named locations

- These allows to coordinate multiple files to each other

Manage multiple shared sites



Used for orientation and position of the project on the site and in relation to other buildings. There may be many Shared Sites defined in one project.

Sites defined in this project :

0,0,0 (current)
Internal

Angle from Project North to True North :

0° 00' 00" East

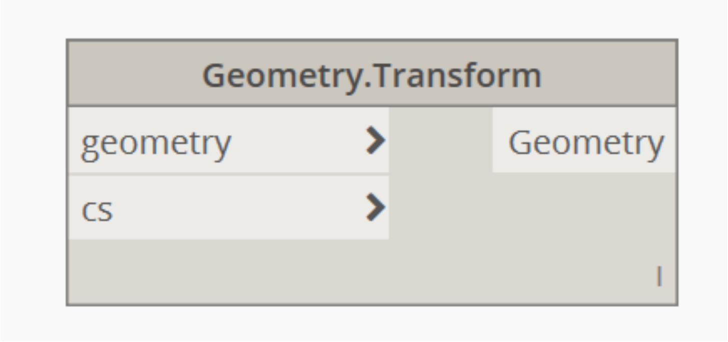
Linking RVT with multiple shared sites

- Every instance of the RVT link can be set to a particular shared site
  - Select the link
  - Enable the shared coordinates
  - Select the proper shared site

Total Transform & Coordinate System

- In Dynamo terms the Revit Shared Sites are Coordinate Systems objects that transform the coordinates from the local origin to the WCS and vice versa

- Extracting point coordinates from Revit or reading point coordinates from Excel require transformation



Dynamo & Python

Python

- PROs
  - Easy to understand and maintain, simple and yet powerful
  - Ideal for prototyping and learning
  - Productive and flexible
  - Large users community
  - Many custom modules
  - Present in Dynamo

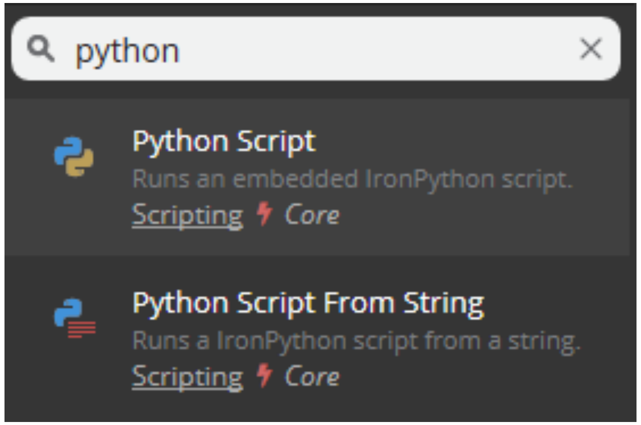
- CONs
  - Not intended for compiling (interpreted language)
  - Speed can be an issue > PyPy
  - Debugging can be cumbersome (errors show up at runtime only)
  - A lot of white space due to indentation
  - Not officially supported for API

Python Learning Resources

- Iron Python Documentation installed on your machine
- Courses online learning platforms
  - i.e. Lynda.com
- Many blogs
  - i.e. http://planetpython.org/
- Many videos on YouTube
- AU lessons and handouts
- http://www.revitapidocs.com/code/

Python Script Nodes

- Two nodes to use Python scripting
- Load .NET namespaces
- Access Revit API in process
- More on DynamoPrimer.com



Python Script
Runs an embedded IronPython script.
Scripting Core

Python Script From String
Runs a IronPython script from a string.
Scripting Core

- Tip
  - Use an external editor and then copy and paste the code in the node
  - Create your own modules to expand the functionalities

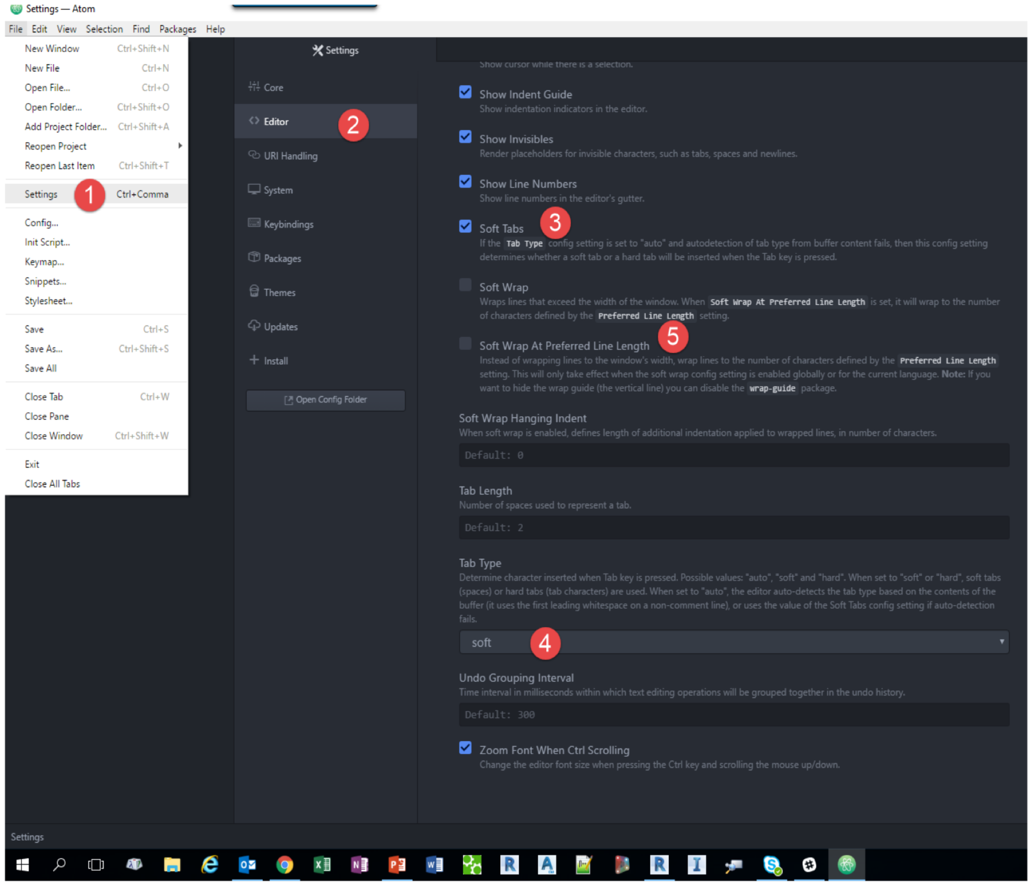
Python Editors

- PyCharm
- Sublime Text
- Visual Studio / Visual Studio Code
- Atom
- Spyder
- Ninja Python
- …

- PEP 8 / PEP 257 for coding style

Python Syntax

- Being consistent with the indentation is very important
- “Soft” Tabs: when the user presses the TAB key the editor places a number of spaces instead (usually 2 or 4)
- Breaking lines is allowed in Python but it can lead to mistakes



Show cursor while there is a selection.

☑ Show Indent Guide
Show indentation indicators in the editor.

☑ Show Invisibles
Render placeholders for invisible characters, such as tabs, spaces and newlines.

☑ Show Line Numbers
Show line numbers in the editor's gutter.

☑ Soft Tabs
If the `Tab Type` config setting is set to "auto" and autodetection of tab type from buffer content fails, then this config setting determines whether a soft tab or a hard tab will be inserted when the Tab key is pressed.

□ Soft Wrap
Wraps lines that exceed the width of the window. When `Soft Wrap At Preferred Line Length` is set, it will wrap to the number of characters defined by the `Preferred Line Length` setting.

□ Soft Wrap At Preferred Line Length
Instead of wrapping lines to the window's width, wrap lines to the number of characters defined by the `Preferred Line Length` setting. This will only take effect when the soft wrap config setting is enabled globally or for the current language. **Note:** If you want to hide the wrap guide (the vertical line) you can disable the `wrap-guide` package.

Soft Wrap Hanging Indent
When soft wrap is enabled, defines length of additional indentation applied to wrapped lines, in number of characters.
`Default: 0`

Tab Length
Number of spaces used to represent a tab.
`Default: 2`

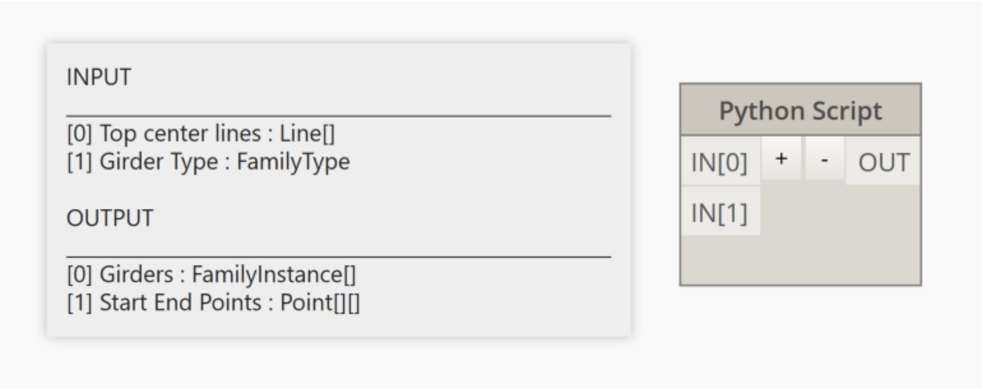
Tab Type
Determine character inserted when Tab key is pressed. Possible values: "auto", "soft" and "hard". When set to "soft" or "hard", soft tabs (spaces) or hard tabs (tab characters) are used. When set to "auto", the editor auto-detects the tab type based on the contents of the buffer (it uses the first leading whitespace on a non-comment line), or uses the value of the Soft Tabs config setting if auto-detection fails.
soft ▾

Undo Grouping Interval
Time interval in milliseconds within which text editing operations will be grouped together in the undo history.
`Default: 300`

☑ Zoom Font When Ctrl Scrolling
Change the editor font size when pressing the Ctrl key and scrolling the mouse up/down.

Python Script Nodes

- The inputs have the key "IN[#]" that reflects the node interface
- They cannot be changed to report a different name on the UI
- The output has the key "OUT" that reflects the node interface
- It cannot be changed to report a different name on the UI
- Use notes to describe the input and output structures and data types



INPUT

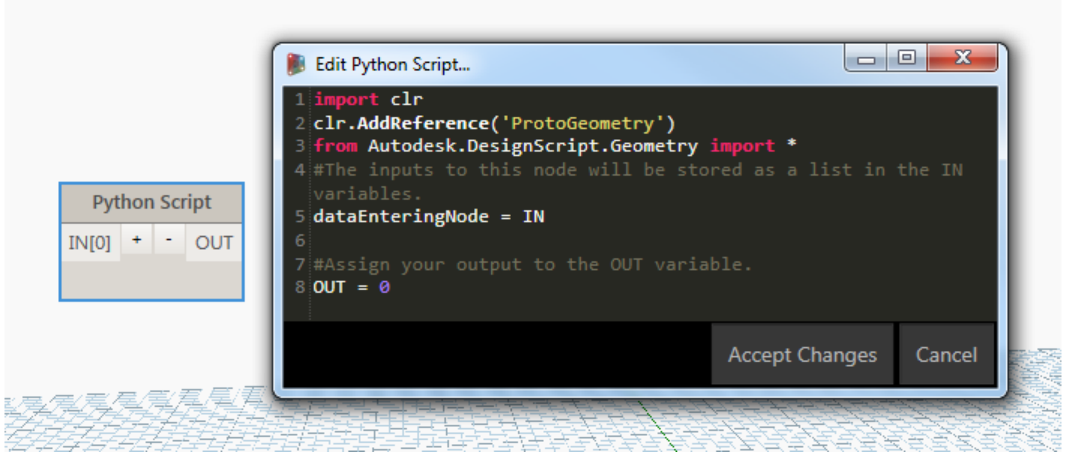
[0] Top center lines : Line[]
[1] Girder Type : FamilyType

OUTPUT

[0] Girders : FamilyInstance[]
[1] Start End Points : Point[][]

Iron Python | .NET Compatible

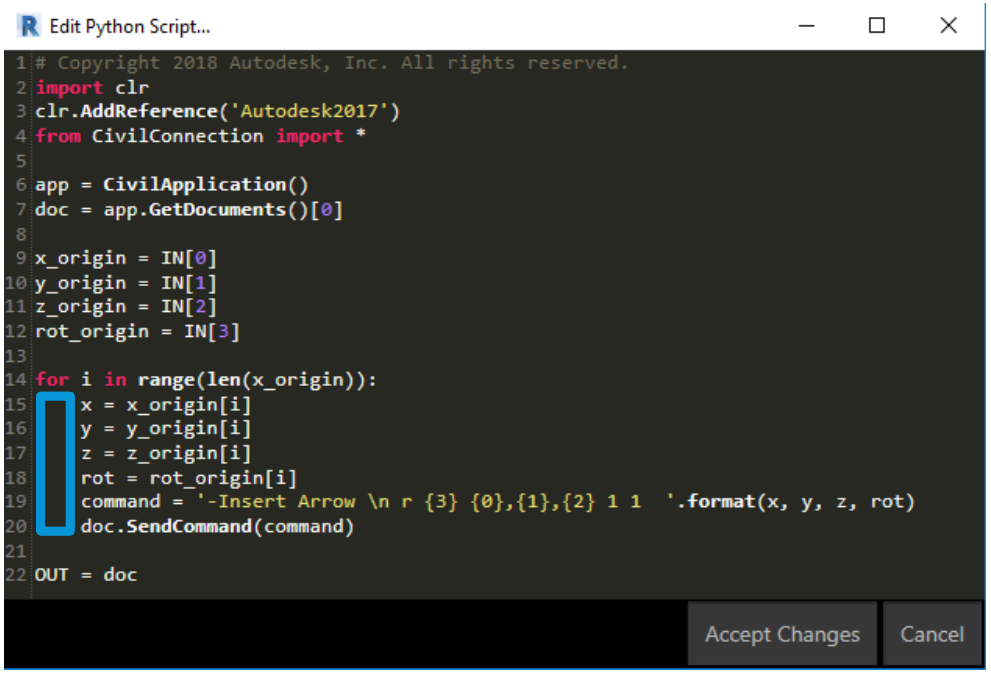
- Interpreted Programming Language (no need to compile)
- Statement grouping via indentation
- IronPython 2.7.3 installed with Dynamo
- .NET capabilities (i.e. Revit API)



```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

#Assign your output to the OUT variable.
OUT = 0
```

Syntax

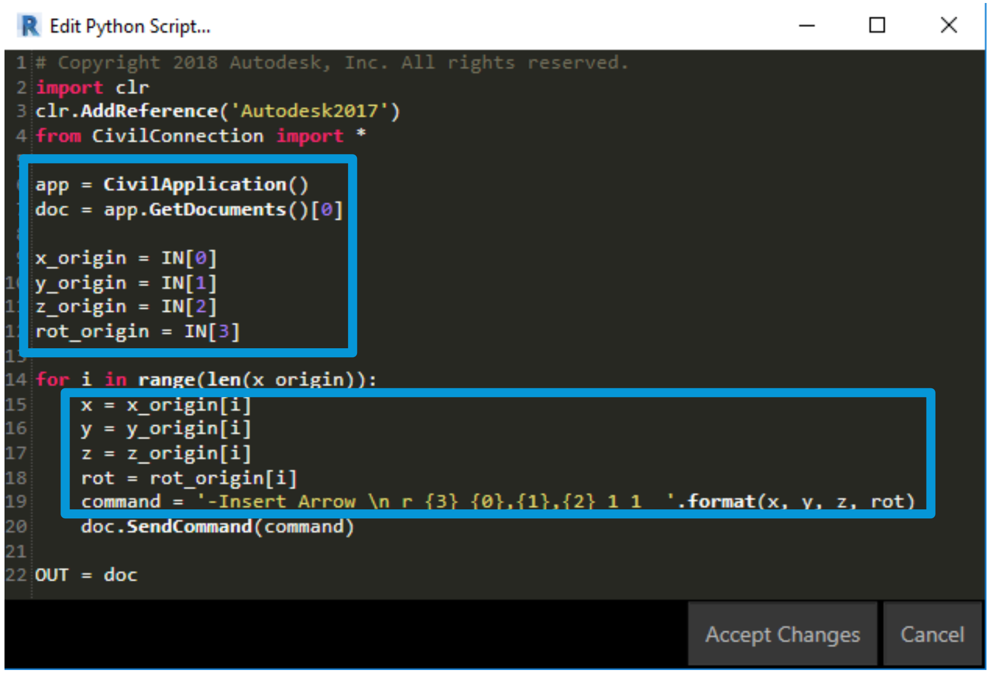
- Case sensitive
- # at the start of the line is a single line comment
- “”” …””” comments multiple lines in between
- End of the line is the end of statement
- Indentation can be done using spaces or tabs at the beginning of the line
- Add spaces for readability (i.e. x = 2 + y)



```python
# Copyright 2018 Autodesk, Inc. All rights reserved.
import clr
clr.AddReference('Autodesk2017')
from CivilConnection import *

app = CivilApplication()
doc = app.GetDocuments()[0]

x_origin = IN[0]
y_origin = IN[1]
z_origin = IN[2]
rot_origin = IN[3]

for i in range(len(x_origin)):
    x = x_origin[i]
    y = y_origin[i]
    z = z_origin[i]
    rot = rot_origin[i]
    command = '-Insert Arrow \n r {3} {0},{1},{2} 1 1  '.format(x, y, z, rot)
    doc.SendCommand(command)

OUT = doc
```

Variables

- The name of a variable is a pointer to a location in the memory
- A variable must be declared before it can be used in a given scope
- The = sign is used to assign a value to a variable
- The same variable can refer to different data types
- Keywords are not allowed as variable names and they are usually highlighted in the editor



```python
# Copyright 2018 Autodesk, Inc. All rights reserved.
import clr
clr.AddReference('Autodesk2017')
from CivilConnection import *

app = CivilApplication()
doc = app.GetDocuments()[0]

x_origin = IN[0]
y_origin = IN[1]
z_origin = IN[2]
rot_origin = IN[3]

for i in range(len(x_origin)):
    x = x_origin[i]
    y = y_origin[i]
    z = z_origin[i]
    rot = rot_origin[i]
    command = '-Insert Arrow \n r {3} {0},{1},{2} 1 1  '.format(x, y, z, rot)
    doc.SendCommand(command)

OUT = doc
```

Data Types

Integers	:	int (i.e. 1)
Numbers	:	float (i.e. 1.0)
Strings	:	str (i.e. “1.0” or ‘1.0’)
Booleans	:	**T**rue / **F**alse (i.e. 2 > 1)
Null value	:	**N**one
…		

Data Types

- Dynamo
  - Integers (i.e. 1)
  - Numbers (i.e. 1.0)
  - Strings (i.e. “1.0” or ‘1.0’)
  - Booleans (i.e. 2 > 1)
  - Null
  - …

- Python
  - int
  - float
  - str
  - True, False
  - None
  - …

Boolean Operations

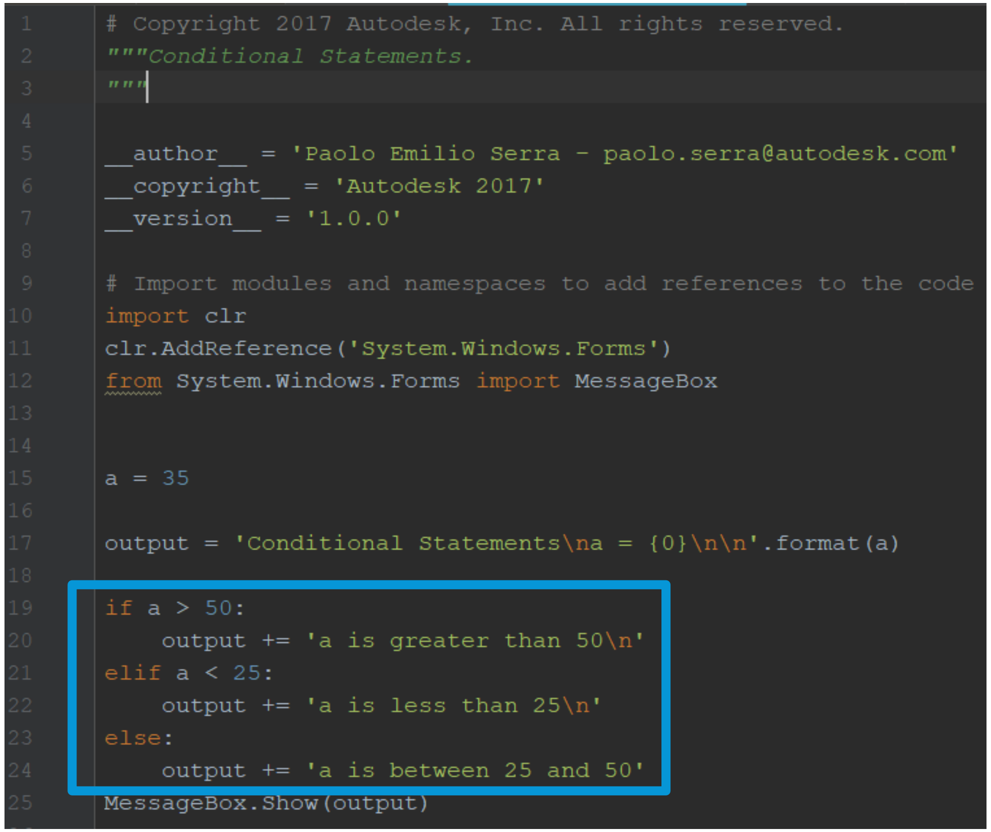
- **not** A : negates A
- A **and** B : True only if all the arguments are True
- A **or** B : True if at least one of the arguments is True
- A **==** B : equality
- A **!=** B : inequality
- A **<** B : less than
- A **<=** B : less than or equal to
- A **>** B : greater than
- A **>=** B : greater than or equal to

Collections

- Zero based collections of values
- **Array / List** defined via **[,]**
- **Typed Array** defined via **Array[T]([ ])** where T is the type
- **Tuple** defined via **(,)**
- **Set** defined via **set()**
- **Dictionary** defined as **{ key1 : value1, key2: value2, … }**
- Get an object from a collection knowing its indexed position or its key:
  - alphabet[2] = “c”

Conditional Statements

```
if Test :
        # do something
elif newTest :
        # do something different
else :
        # final case
```



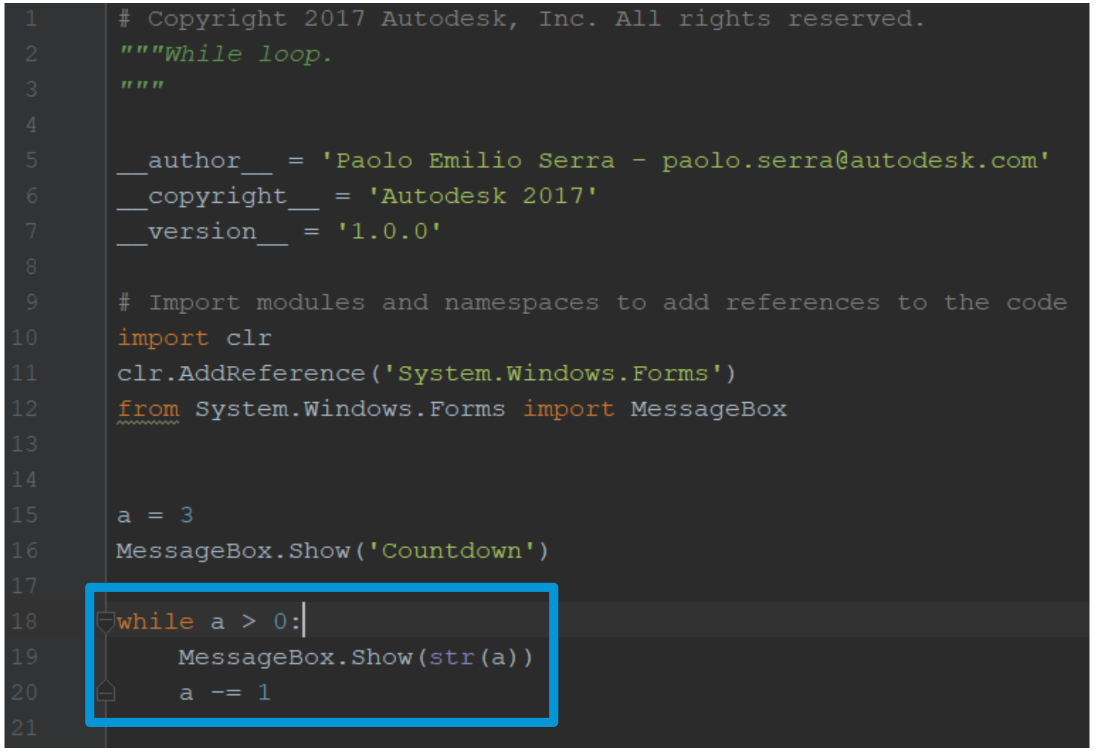
```python
# Copyright 2017 Autodesk, Inc. All rights reserved.
"""Conditional Statements.
"""

__author__ = 'Paolo Emilio Serra - paolo.serra@autodesk.com'
__copyright__ = 'Autodesk 2017'
__version__ = '1.0.0'

# Import modules and namespaces to add references to the code
import clr
clr.AddReference('System.Windows.Forms')
from System.Windows.Forms import MessageBox


a = 35

output = 'Conditional Statements\na = {0}\n\n'.format(a)

if a > 50:
    output += 'a is greater than 50\n'
elif a < 25:
    output += 'a is less than 25\n'
else:
    output += 'a is between 25 and 50'
MessageBox.Show(output)
```

Loops | While

Repeat instructions in the body until a condition is met

```
while Test :

        # do something
```

- The loop will break only when the Test returns False
- It is very easy to make a mistake and create infinite loops



```python
# Copyright 2017 Autodesk, Inc. All rights reserved.
"""While loop.
"""

__author__ = 'Paolo Emilio Serra - paolo.serra@autodesk.com'
__copyright__ = 'Autodesk 2017'
__version__ = '1.0.0'

# Import modules and namespaces to add references to the code
import clr
clr.AddReference('System.Windows.Forms')
from System.Windows.Forms import MessageBox


a = 3
MessageBox.Show('Countdown')

while a > 0:
    MessageBox.Show(str(a))
    a -= 1
```

Loops | For

Traverse a list and repeat the instructions a given amount of times

```
for iterator in collection :

        # loop instructions
```

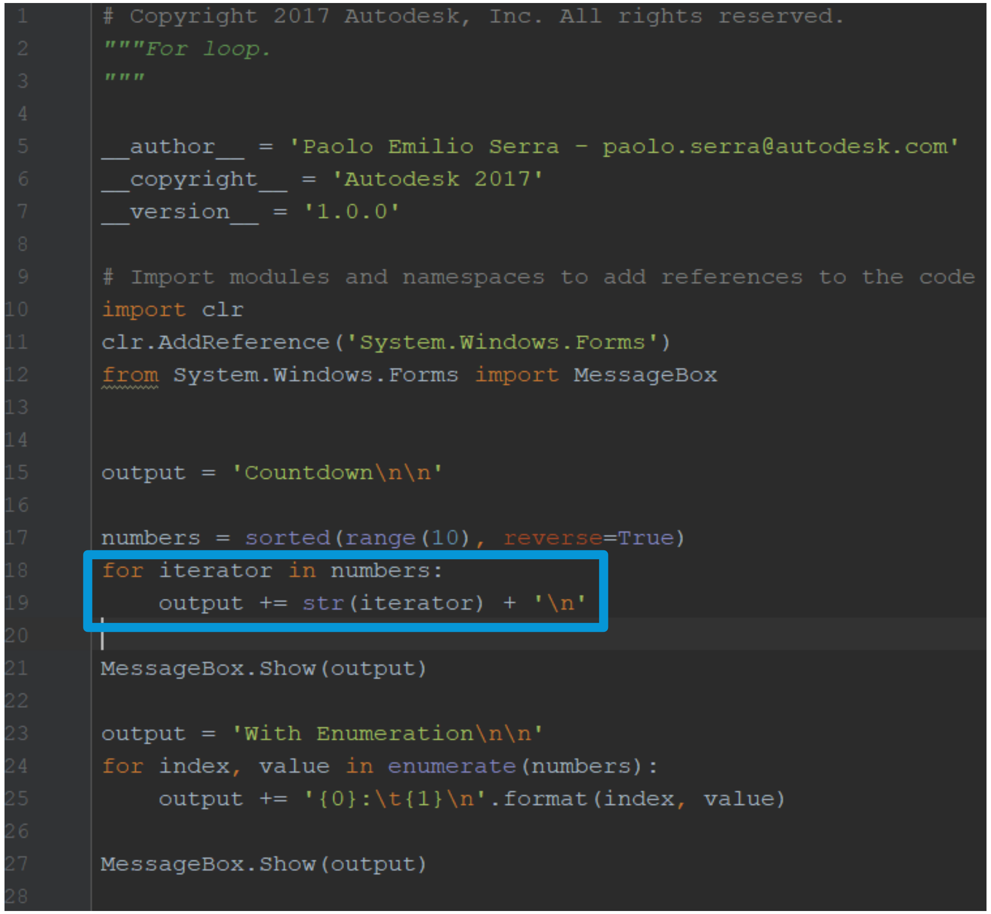
range() : creates a sequence of numbers in arithmetic progression

len() : returns the amount of elements of a collection

break : exits the smallest enclosing loop

continue : moves on to the next value of iterator

pass : does nothing



```python
# Copyright 2017 Autodesk, Inc. All rights reserved.
"""For loop.
"""

__author__ = 'Paolo Emilio Serra - paolo.serra@autodesk.com'
__copyright__ = 'Autodesk 2017'
__version__ = '1.0.0'

# Import modules and namespaces to add references to the code
import clr
clr.AddReference('System.Windows.Forms')
from System.Windows.Forms import MessageBox


output = 'Countdown\n\n'

numbers = sorted(range(10), reverse=True)
for iterator in numbers:
    output += str(iterator) + '\n'

MessageBox.Show(output)

output = 'With Enumeration\n\n'
for index, value in enumerate(numbers):
    output += '{0}:\t{1}\n'.format(index, value)

MessageBox.Show(output)
```

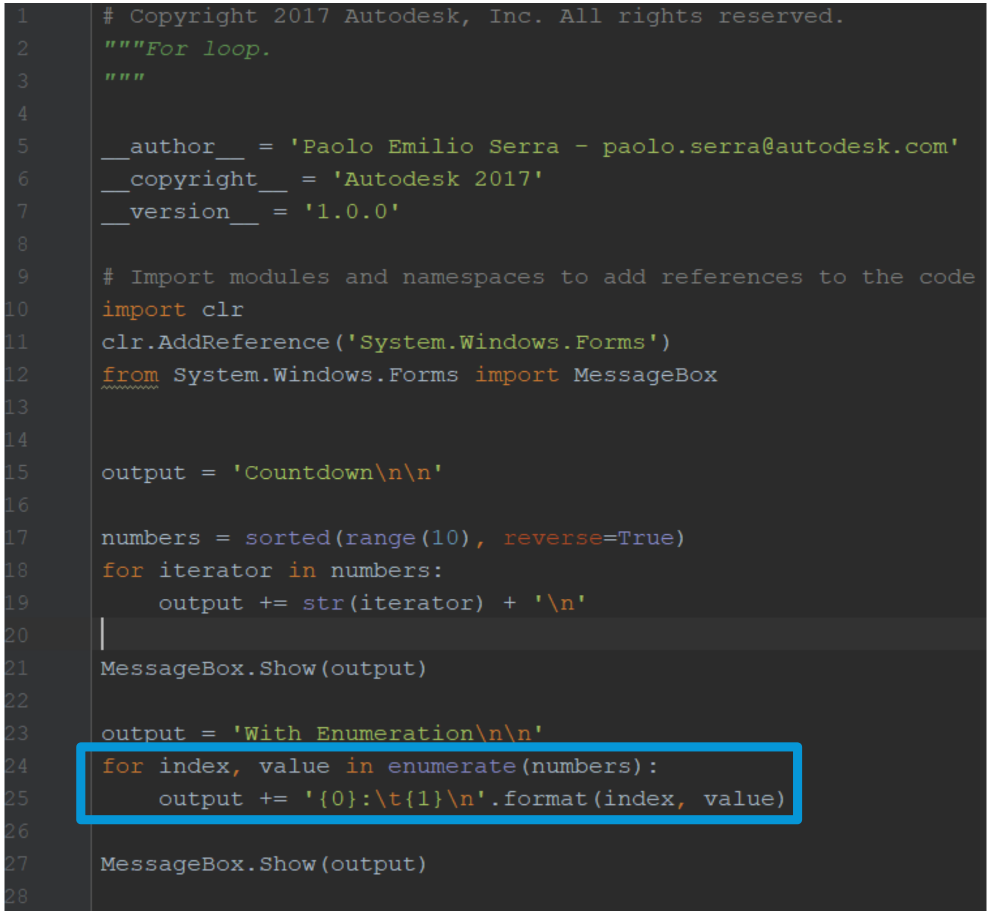
Enumerate

for key, value in **enumerate**(collection) :

# loop instructions

The enumerate() function returns simultaneously a reference to the index of an item in a collection (key) and a reference to its value

This allows to use the same index across multiple collections



```
# Copyright 2017 Autodesk, Inc. All rights reserved.
"""For loop.
"""

__author__ = 'Paolo Emilio Serra - paolo.serra@autodesk.com'
__copyright__ = 'Autodesk 2017'
__version__ = '1.0.0'

# Import modules and namespaces to add references to the code
import clr
clr.AddReference('System.Windows.Forms')
from System.Windows.Forms import MessageBox


output = 'Countdown\n\n'

numbers = sorted(range(10), reverse=True)
for iterator in numbers:
    output += str(iterator) + '\n'

MessageBox.Show(output)

output = 'With Enumeration\n\n'
for index, value in enumerate(numbers):
    output += '{0}:\t{1}\n'.format(index, value)

MessageBox.Show(output)
```

Named Function

A function is an object

- Set of instructions that can be recalled in the body of the code as many times as needed
- Easier to read and maintain
- Call itself in its definition (recursion)
- Can be used for sorting
- Arguments are optional > use defaults
- Parameters (definition), Arguments (call)
- Return is optional (None)

```
def fn_name (par0, par1, par2, …) :

          # list of instructions

          return output
```

Python Variable Reference

- In Python variables are simply names referring to objects in the memory
- Arguments passed to functions by reference
- Objects can be mutable or unmutable
  - Mutable: List, Dictionary, etc.
  - Unmutable: int, str, etc. (hashable)

```
x = [ ]

y = x

y.append(10)

print ‘X = ’ , x

print ‘Y = ‘, y
```

X is the name of the variable point to the object in the memory

Y is a new variable pointing to the same object in the memory

The List is mutable and it is possible to add an item to it

Output

```
X = [10]

Y = [10]
```

Both X and Y are pointing to the same object in the memory, the value assigned to the variables is the same

Lambda Forms

Anonymous functions

- Small functions made of a single expression

```
lambda arg0, arg1, arg2, … : # use the arguments
```

- Used whenever a function object is required
- sorted() Python Built-In sorting function

```
collection = sorted(collection, key=function,
reverse = True)
```

- Optionally lambda forms can be used as key for sorting

```
points = sorted(points, key=lambda k : k.X)
```

- Optionally the collection can be reversed

Comprehension

- Concise syntax to apply filters and functions on a collection of items
- It can return:
  - List []
  - Tuple ()
  - Dictionary {}
- Very useful in Revit API filtering / sorting
- It's the equivalent of for loops and if statements but in one line
- It can be used to "flatten" a multi-dimensional array

```
a = [ function(i) for i in collection if Test ]
```

```
mda = [[0, 1], [2, 3]]

flat = [a for row in mda for a in row]
```

Context Manager

- A construct that safely disposes an object when the focus exists the “with” scope
- Used to interact with databases (files, transactions, etc.)
- It prevents to do harm to the documents and applications the code in interacting with

```
with Object as variable :

        # do something to the Object



# the Object will be safely disposed
```

Debugging in Python for Dynamo

- Traceback call with a reference to the line containing an Error or Exception
- The name of the Exception gives an idea of what the problem might be

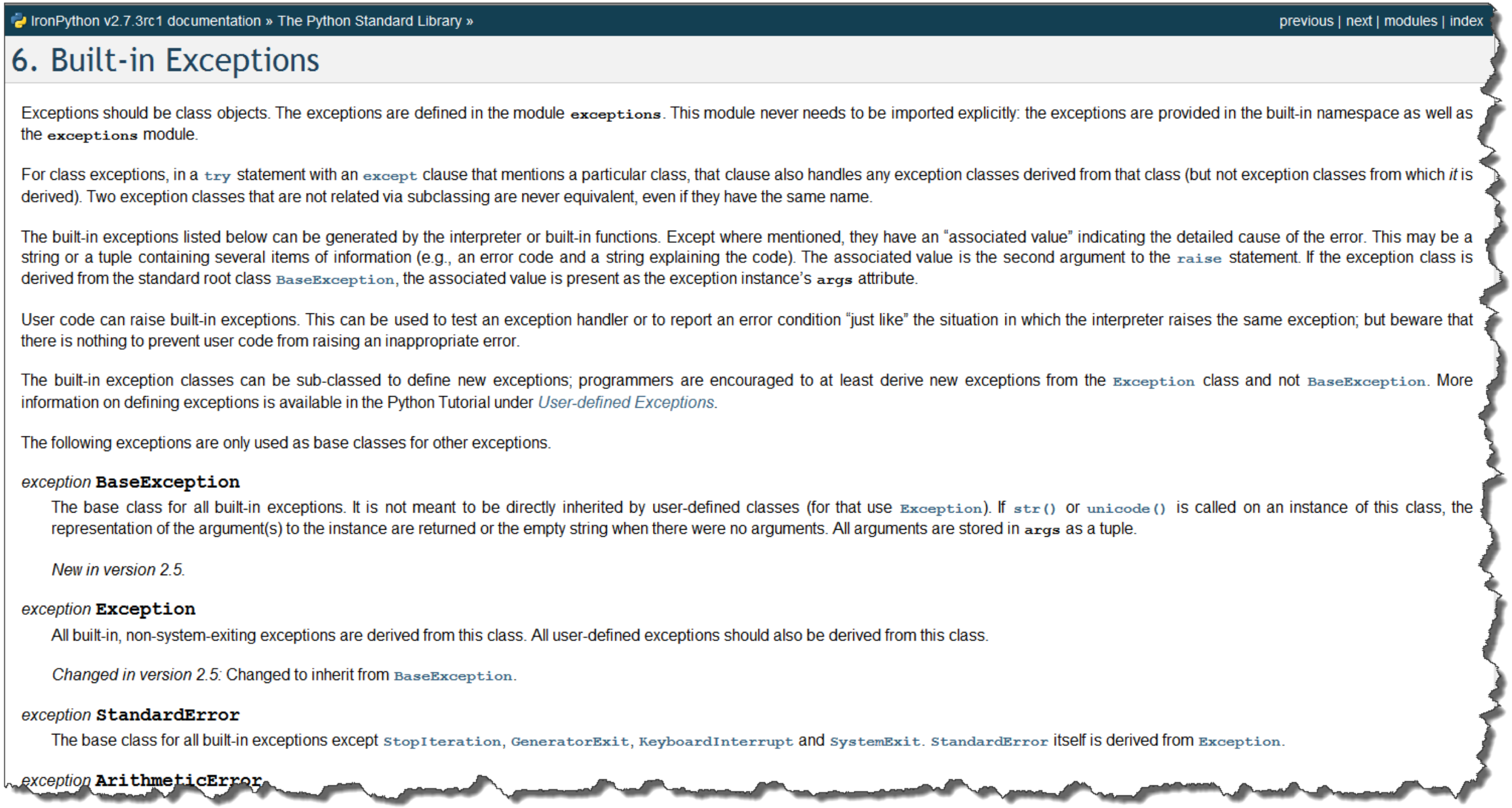
1. Syntax
2. Functions arguments
3. Instructions evaluation
4. Input values

Try / Except

- The instructions in the “try” scope may fail
- The “except” scopes can be introduced to catch and handle different scenarios
- If the error type is not specified, all sorts of errors will be caught (even typos!)
- Once the error is handled, the code can continue

```
try :
        # try to do something
except Error1 :
        # do this if Error1 is encountered
except Error2 :
        # do this if Error2 is encountered
except :
        # this catches all kinds of errors
```

Built-In Exceptions


IronPython v2.7.3rc1 documentation » The Python Standard Library »
previous | next | modules | index

6. Built-in Exceptions

Exceptions should be class objects. The exceptions are defined in the module `exceptions`. This module never needs to be imported explicitly: the exceptions are provided in the built-in namespace as well as the `exceptions` module.

For class exceptions, in a `try` statement with an `except` clause that mentions a particular class, that clause also handles any exception classes derived from that class (but not exception classes from which *it* is derived). Two exception classes that are not related via subclassing are never equivalent, even if they have the same name.

The built-in exceptions listed below can be generated by the interpreter or built-in functions. Except where mentioned, they have an "associated value" indicating the detailed cause of the error. This may be a string or a tuple containing several items of information (e.g., an error code and a string explaining the code). The associated value is the second argument to the `raise` statement. If the exception class is derived from the standard root class `BaseException`, the associated value is present as the exception instance's `args` attribute.

User code can raise built-in exceptions. This can be used to test an exception handler or to report an error condition "just like" the situation in which the interpreter raises the same exception; but beware that there is nothing to prevent user code from raising an inappropriate error.

The built-in exception classes can be sub-classed to define new exceptions; programmers are encouraged to at least derive new exceptions from the `Exception` class and not `BaseException`. More information on defining exceptions is available in the Python Tutorial under *User-defined Exceptions*.

The following exceptions are only used as base classes for other exceptions.

exception **`BaseException`**

The base class for all built-in exceptions. It is not meant to be directly inherited by user-defined classes (for that use `Exception`). If `str()` or `unicode()` is called on an instance of this class, the representation of the argument(s) to the instance are returned or the empty string when there were no arguments. All arguments are stored in `args` as a tuple.

New in version 2.5.

exception **`Exception`**

All built-in, non-system-exiting exceptions are derived from this class. All user-defined exceptions should also be derived from this class.

Changed in version 2.5: Changed to inherit from `BaseException`.

exception **`StandardError`**

The base class for all built-in exceptions except `StopIteration`, `GeneratorExit`, `KeyboardInterrupt` and `SystemExit`. `StandardError` itself is derived from `Exception`.

exception **`ArithmeticError`**

Operating With Files

- Context manager to safely interact with files on the hard drive
- OpenMode:
    - Reading ‘r’
    - Writing ‘w’ (overrides the content)
    - Append ‘a’
- ‘ab+’ means appends with a binary format and it can also read from the source (+)

```
with open(“filepath”, OpenMode) as f :

        # do something
```

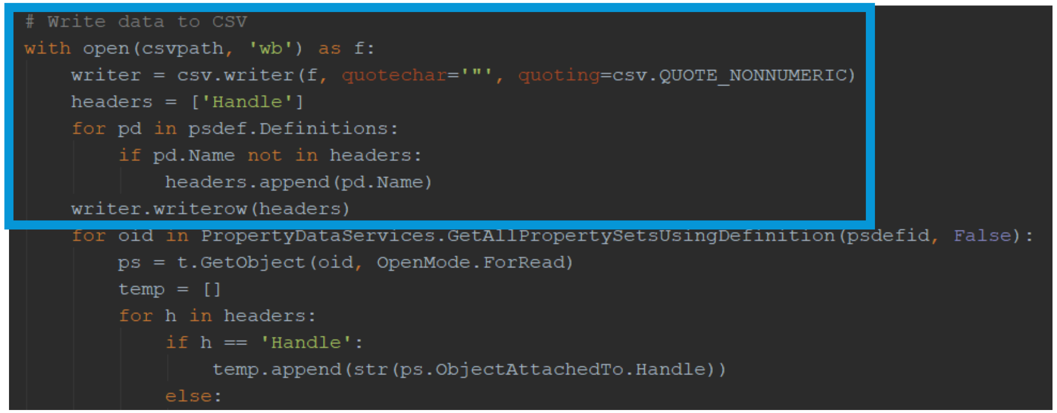
```
File.readline()

File.write()
```

Operating with CSV files

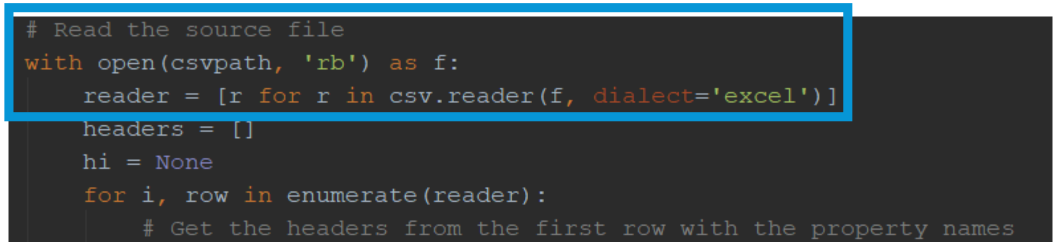
- csv module included with Python

- **csv.writer.writerow()** takes a list of values and appends a row to the CSV file

- **csv.reader** returns a rank 2 array containing the rows



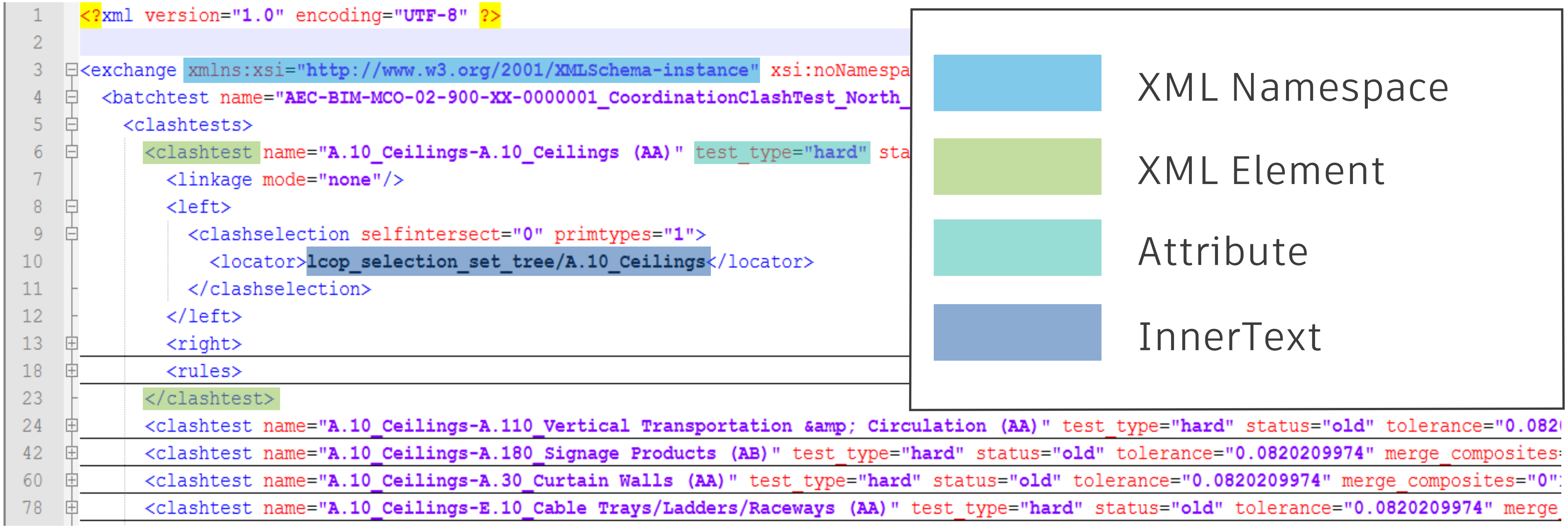
```python
# Write data to CSV
with open(csvpath, 'wb') as f:
    writer = csv.writer(f, quotechar='"', quoting=csv.QUOTE_NONNUMERIC)
    headers = ['Handle']
    for pd in psdef.Definitions:
        if pd.Name not in headers:
            headers.append(pd.Name)
    writer.writerow(headers)
    for oid in PropertyDataServices.GetAllPropertySetsUsingDefinition(psdefid, False):
        ps = t.GetObject(oid, OpenMode.ForRead)
        temp = []
        for h in headers:
            if h == 'Handle':
                temp.append(str(ps.ObjectAttachedTo.Handle))
            else:
```



```python
# Read the source file
with open(csvpath, 'rb') as f:
    reader = [r for r in csv.reader(f, dialect='excel')]
    headers = []
    hi = None
    for i, row in enumerate(reader):
        # Get the headers from the first row with the property names
```

Operating with XML

Extensible Markup Language

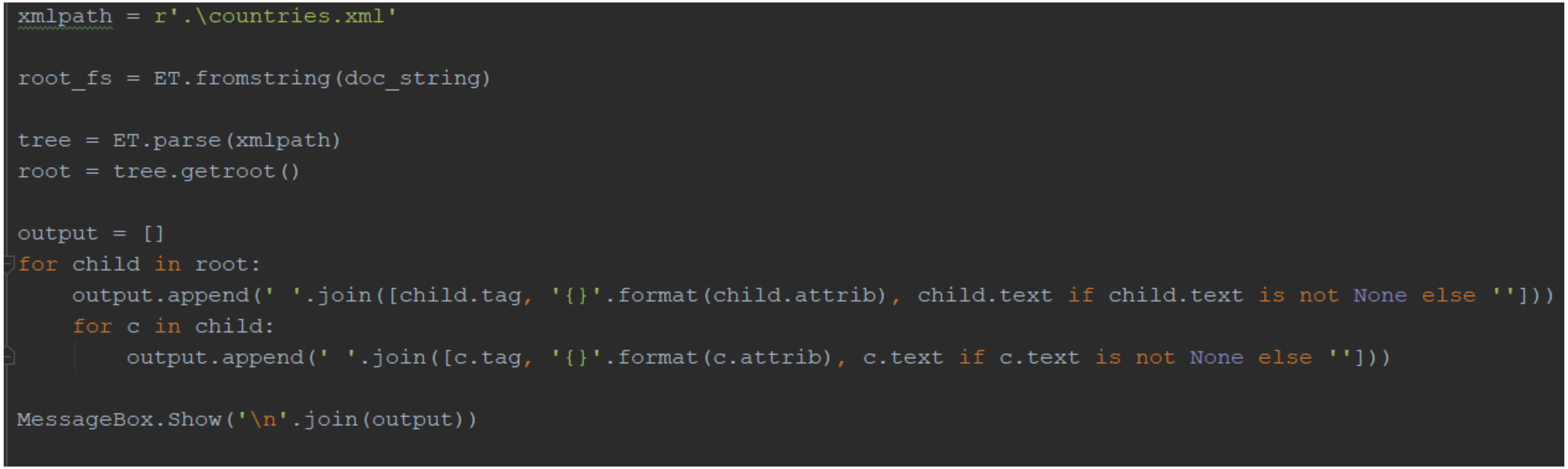


```xml
<?xml version="1.0" encoding="UTF-8" ?>

<exchange xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespa
  <batchtest name="AEC-BIM-MCO-02-900-XX-0000001_CoordinationClashTest_North_
    <clashtests>
      <clashtest name="A.10_Ceilings-A.10_Ceilings (AA)" test_type="hard" sta
        <linkage mode="none"/>
        <left>
          <clashselection selfintersect="0" primtypes="1">
            <locator>lcop_selection_set_tree/A.10_Ceilings</locator>
          </clashselection>
        </left>
        <right>
        <rules>
      </clashtest>
      <clashtest name="A.10_Ceilings-A.110_Vertical Transportation &amp; Circulation (AA)" test_type="hard" status="old" tolerance="0.082
      <clashtest name="A.10_Ceilings-A.180_Signage Products (AB)" test_type="hard" status="old" tolerance="0.0820209974" merge_composites
      <clashtest name="A.10_Ceilings-A.30_Curtain Walls (AA)" test_type="hard" status="old" tolerance="0.0820209974" merge_composites="0"
      <clashtest name="A.10_Ceilings-E.10_Cable Trays/Ladders/Raceways (AA)" test_type="hard" status="old" tolerance="0.0820209974" merge
```

🟦	XML Namespace
🟩	XML Element
🟩	Attribute
🟦	InnerText

Python XML module | Element Tree

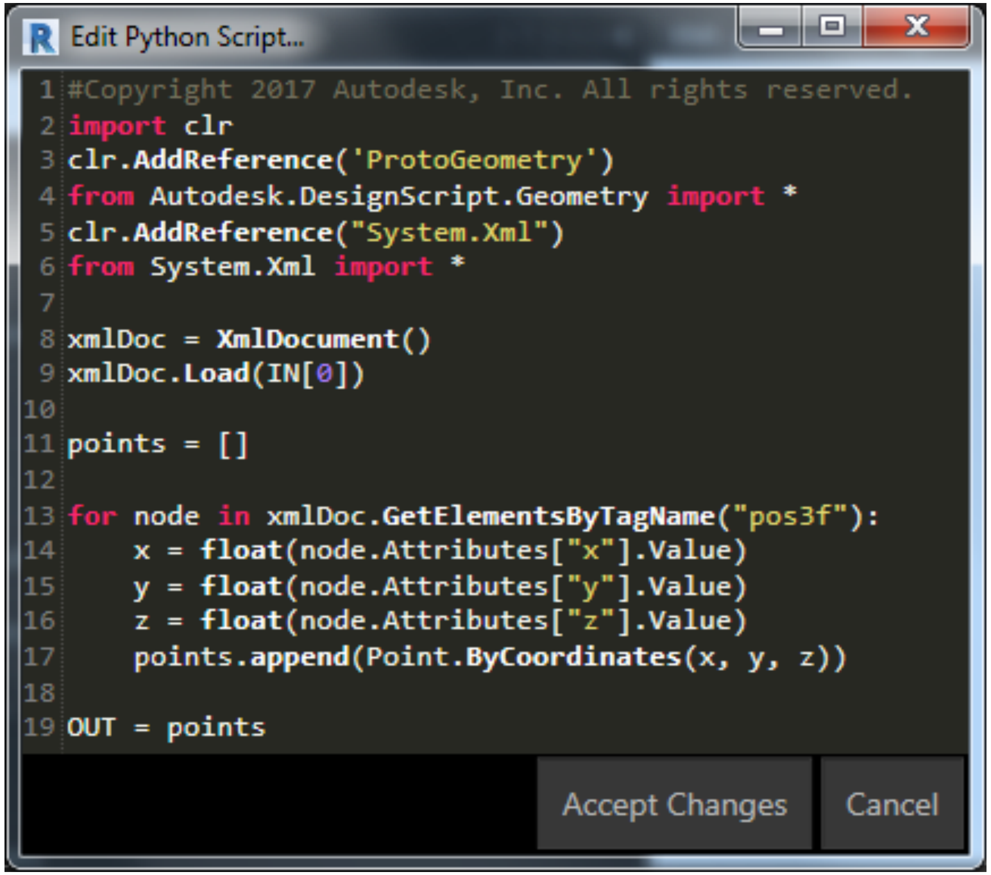
- Built-in module xml.etree.ElementTree
- Parse a file from source or string
- Find elements by tag
- Get and Set attributes of existing XML elements
- Get and Set elements text
- Create new SubElements



```
xmlpath = r'.\countries.xml'

root_fs = ET.fromstring(doc_string)

tree = ET.parse(xmlpath)
root = tree.getroot()

output = []
for child in root:
    output.append(' '.join([child.tag, '{}'.format(child.attrib), child.text if child.text is not None else '']))
    for c in child:
        output.append(' '.join([c.tag, '{}'.format(c.attrib), c.text if c.text is not None else '']))

MessageBox.Show('\n'.join(output))
```

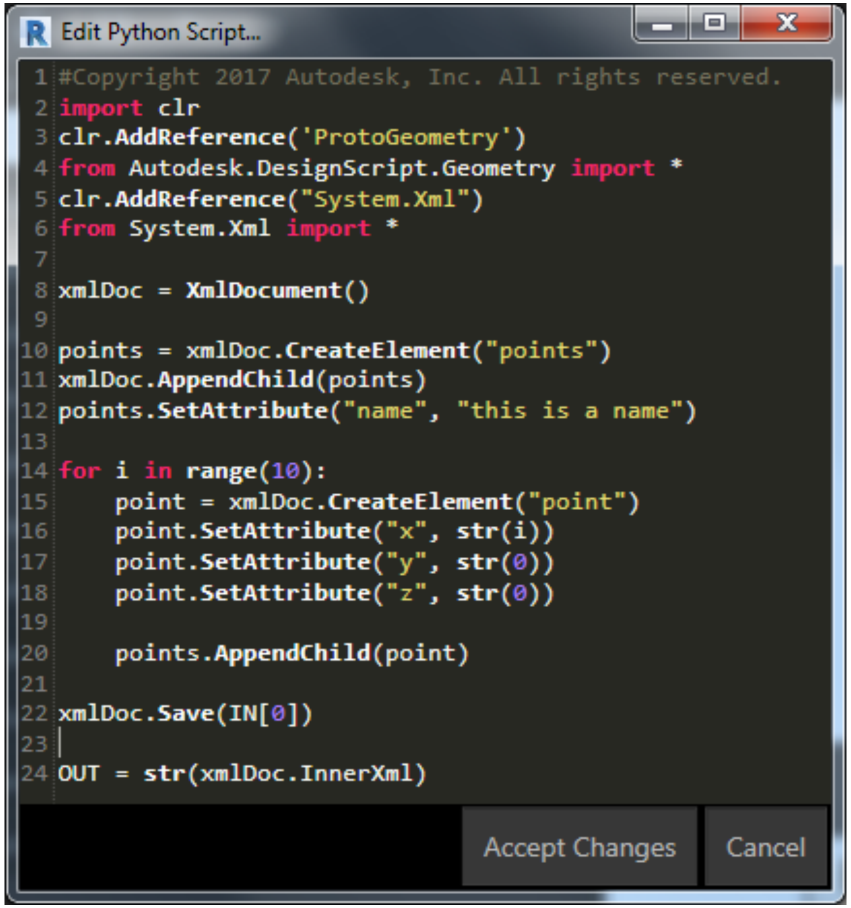
Reading and Writing XML | System.Xml

- Add reference to System.Xml
- XML Namespace, XML Document, XML Element, Attributes
- Understand / define the schema to adopt
- Select nodes using Tag name or Xpath
- Example: reading a Navisworks clash report

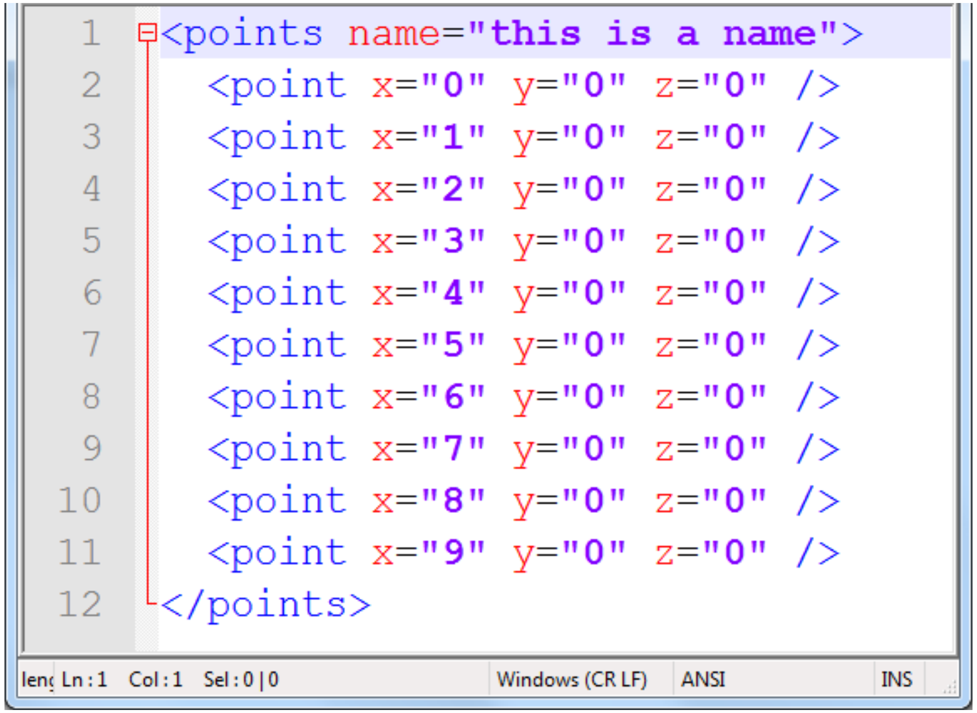


```
#Copyright 2017 Autodesk, Inc. All rights reserved.
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
clr.AddReference("System.Xml")
from System.Xml import *

xmlDoc = XmlDocument()
xmlDoc.Load(IN[0])

points = []

for node in xmlDoc.GetElementsByTagName("pos3f"):
    x = float(node.Attributes["x"].Value)
    y = float(node.Attributes["y"].Value)
    z = float(node.Attributes["z"].Value)
    points.append(Point.ByCoordinates(x, y, z))

OUT = points
```

Reading and Writing XML



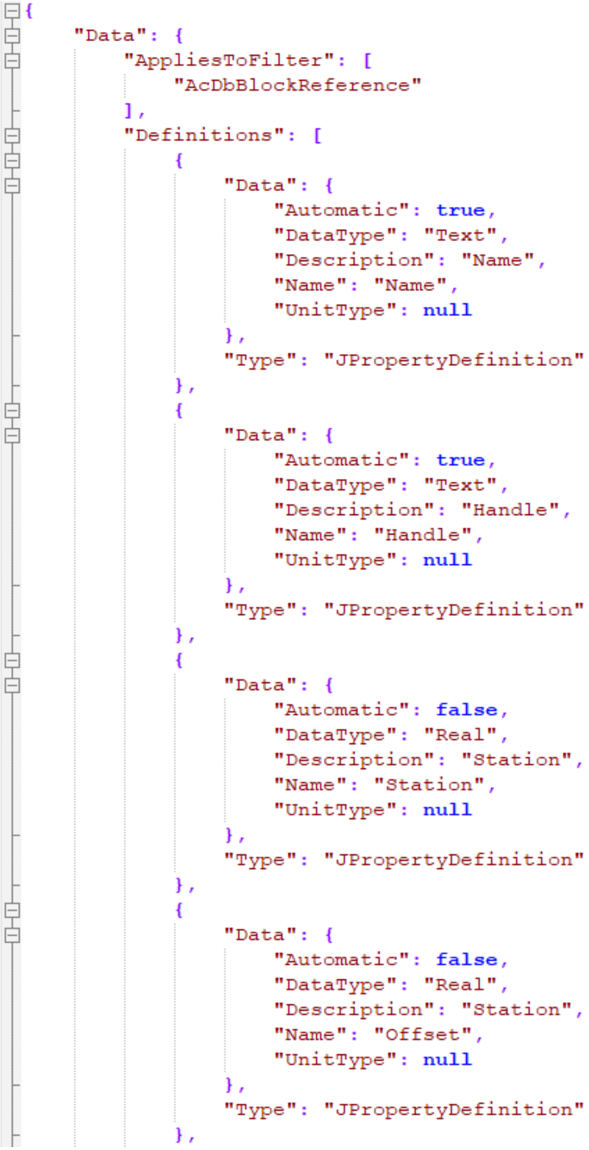
```python
#Copyright 2017 Autodesk, Inc. All rights reserved.
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
clr.AddReference("System.Xml")
from System.Xml import *

xmlDoc = XmlDocument()

points = xmlDoc.CreateElement("points")
xmlDoc.AppendChild(points)
points.SetAttribute("name", "this is a name")

for i in range(10):
    point = xmlDoc.CreateElement("point")
    point.SetAttribute("x", str(i))
    point.SetAttribute("y", str(0))
    point.SetAttribute("z", str(0))

    points.AppendChild(point)

xmlDoc.Save(IN[0])

OUT = str(xmlDoc.InnerXml)
```



```xml
<points name="this is a name">
  <point x="0" y="0" z="0" />
  <point x="1" y="0" z="0" />
  <point x="2" y="0" z="0" />
  <point x="3" y="0" z="0" />
  <point x="4" y="0" z="0" />
  <point x="5" y="0" z="0" />
  <point x="6" y="0" z="0" />
  <point x="7" y="0" z="0" />
  <point x="8" y="0" z="0" />
  <point x="9" y="0" z="0" />
</points>
```

Operating with JSON

- Java Script Object Notation
- Python built-in module json
- It is possible to read and write a file
- It supports custom Encoding and Decoding
- Very good performance, human readable



```
{
    "Data": {
        "AppliesToFilter": [
            "AcDbBlockReference"
        ],
        "Definitions": [
            {
                "Data": {
                    "Automatic": true,
                    "DataType": "Text",
                    "Description": "Name",
                    "Name": "Name",
                    "UnitType": null
                },
                "Type": "JPropertyDefinition"
            },
            {
                "Data": {
                    "Automatic": true,
                    "DataType": "Text",
                    "Description": "Handle",
                    "Name": "Handle",
                    "UnitType": null
                },
                "Type": "JPropertyDefinition"
            },
            {
                "Data": {
                    "Automatic": false,
                    "DataType": "Real",
                    "Description": "Station",
                    "Name": "Station",
                    "UnitType": null
                },
                "Type": "JPropertyDefinition"
            },
            {
                "Data": {
                    "Automatic": false,
                    "DataType": "Real",
                    "Description": "Station",
                    "Name": "Offset",
                    "UnitType": null
                },
                "Type": "JPropertyDefinition"
            },
```

Persistence

Serialization of data

- Multiple options
  - pickle (dedicated Python module)
  - XML
  - CSV
  - JSON (very popular for web)
- Restore values and objects between executions
- It can be used to store the results of expensive calculations and improve performances
- Dictionaries are the best structures to read and write data

Object Oriented Programming

Object Oriented Programming

Based on Classes (or types) and Objects (or instances)

- Classes
  - The blue-print of objects from which individual objects are created
  - Define properties and methods
  - Can be inherited from other classes
- Objects
  - They all have state (properties) and behavior (methods)
  - Objects are instances of classes

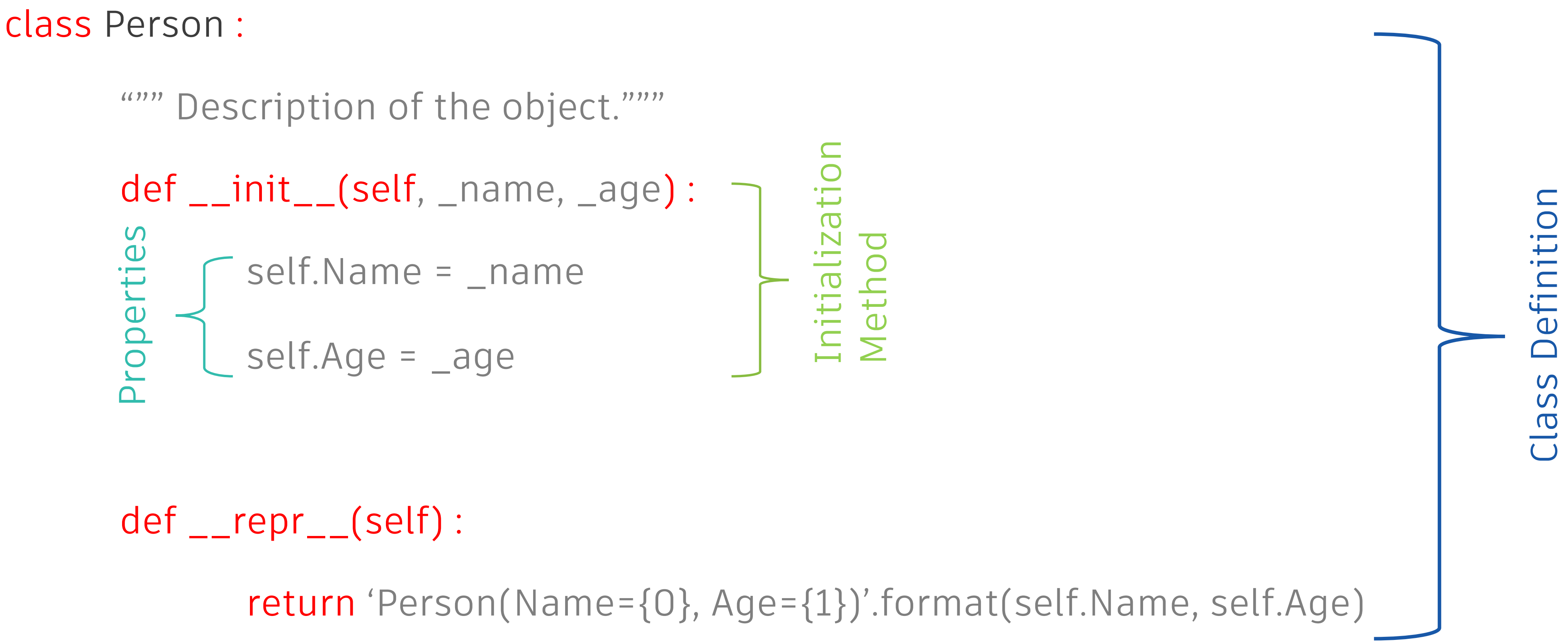
Object Oriented Programming

Basic principles

- **Inheritance**: parent class and descendants, abstract classes
- **Polymorphism**: code can be called on objects regardless they belong to parent or descendants classes
- **Encapsulation**: access modifier (public, protected, private)
- **Open Recursion**: object methods can call other methods on the same object including themselves (self)

- Class Members
  - Properties define the state of an object
  - Methods define how an object behaves
- They can be called via the “dot” notation
  - p.X # returns the x coordinate property
  - p.Add(q) # performs an action on the object

Python Classes



```
class Person :

        “”” Description of the object.”””

        def __init__(self, _name, _age) :

                self.Name = _name

                self.Age = _age


        def __repr__(self) :

                return ‘Person(Name={0}, Age={1})’.format(self.Name, self.Age)
```

Properties

Initialization Method

Class Definition

Python Classes

- The keyword class defines a class
- The keyword self is used to refer to an object (an instance of the class) in its definition
- Use the keyword self to refer to properties and methods defined in the class
- Define attributes and members to enable common behaviors such as:
  - String representation
  - Comparison
- Use dir(object) to access the class members

Decorators

- A design pattern that takes a function and wraps it into another function
- Used to customize the behaviors of class members at runtime
- Define read only properties
- Define class and static methods
- @decorator

Namespaces

- An organized collection of classes
- Compiled in .DLL files or in the GAC
- Defined in Python (.py) files
- IronPython add reference
- Import classes to make their names available in the code (either all or a subset of those present in the namespace)
- Define aliases for disambiguation

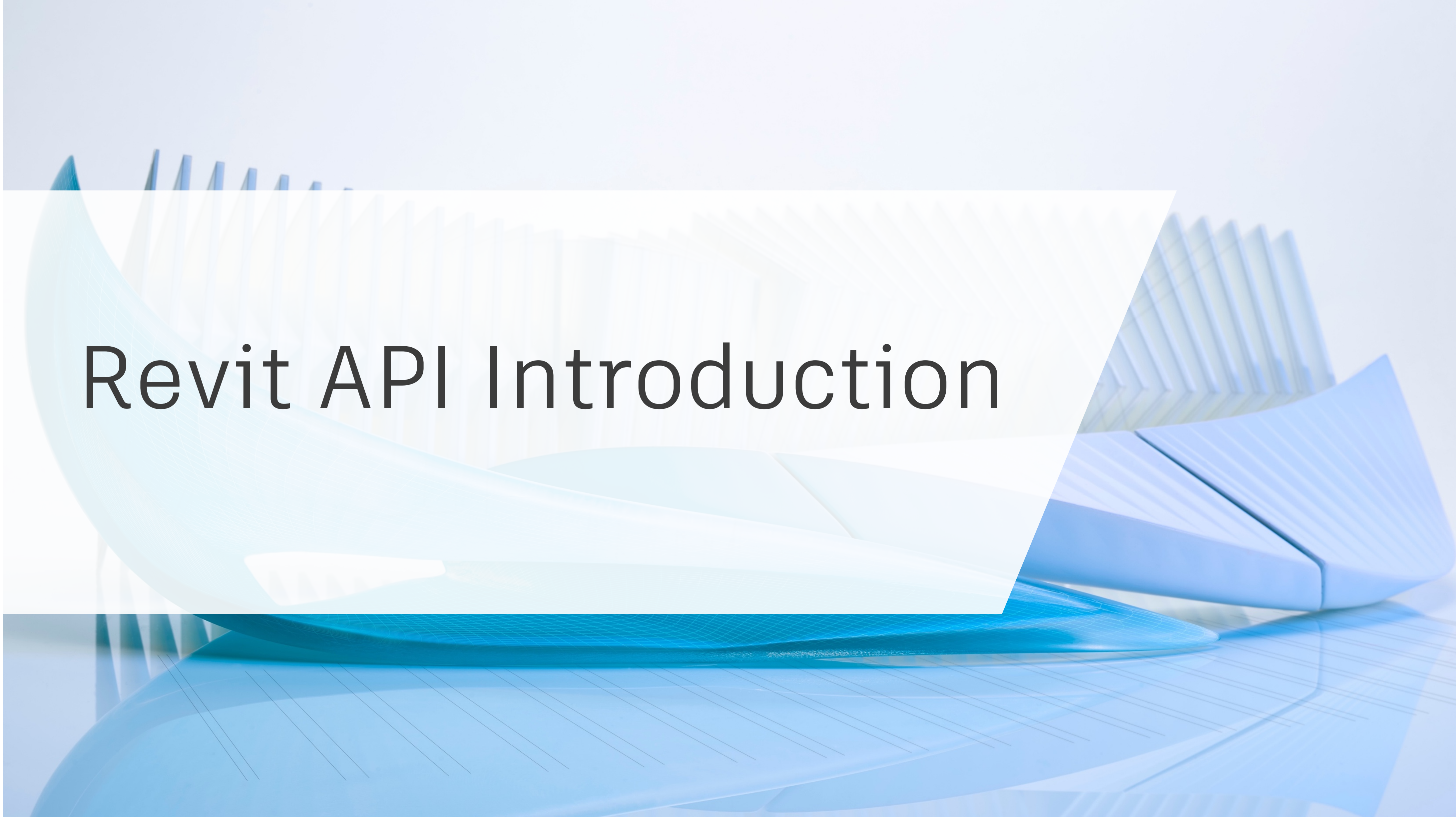
```
import System

from Autodesk.Revit.DB import *

from Autodesk.Revit.UI import TaskDialog, Selection

from Autodesk.DesignScript.Geometry import Point as DSPoint
```

Python Modules

- A Python file can contain multiple classes definitions and can be referenced in another file
- __all__ is a list of names of the objects defined in a Python file that are available when using import *
- Defining a folder with a name and a __init__.py file defines a Python module
- Python modules can be downloaded to expand the available classes (numpy, shapely, etc.)

Python PEP8 Style Guide

- Standard for Python code development
- Ensures readability
- Facilitate maintenance
- Helps in understanding better a code
- Conventions for naming files, constants, variables, functions, parameters, classes, methods, properties, etc.
- Documentation strings to be included in the definitions (`__doc__`)
- Link



Revit API Introduction

Revit API | Resources

- Software Development Kit (SDK)
  - RevitAPI.chm
- The Building Coder (Jeremy Tammik)
- Revit LookUp add-in

Revit API | Main Namespaces

- Autodesk.Revit.DB
  - Access to the Database object and its children
    - Document
    - Elements
- Autodesk.Revit.UI
  - Access to objects that allows the user interaction with Revit
    - Selection
    - Messages

Revit API | Application

- An object that represents the instance of the Revit application that is running
- It contains a reference to the Documents loaded (active projects and links)
- It allows to access the Revit application settings

Revit API | UIDocument

- An object that represents the current active project presented to the user

- It contains the methods necessary to interactively select items, the result is a collection of Reference objects

- It allows to present messages to the user via TaskDialog (also useful for debugging)

Revit API | Document

- An object that represents a Revit file (project .rvt or family .rfa)
- It has properties to define the file (Title, PathName, etc.)
- It has methods to retrieve objects and modify the content
  - GetElement(), FilteredElementCollector(), …

Revit API | Element

- The base object for most of the Revit items (inheritance and polymorphism)
  - Instance
  - ElementType
  - Wall
  - …

Revit API | Parameters

- Every Element object has a specific set of properties that help describe the object in more detail

- They can be Built-In or added via Shared Parameters file

- Object parameters are retrieved not in any particular order so it is important to be able to filter and sort them

Revit API | Parameters

- Parameters value can be read-only and of a specific data type (StorageType)
- The internal units may differ from the Display Unit Type
- To set parameter values there is need for an open Transaction

Revit API | Transaction

- Allows to safely access to the Document database
- The name in the “undo” history
- The transaction can Start, Commit if successful or Rollback if not
- In Python use the with statement to be sure to handle the database correctly

Transaction example

```
with Transaction(doc, “TransactionName”) as t :

        t.Start()

        # do something

        t.Commit()
```

Transaction.Rollback() restores the Document to the same state as before the Transaction started.

If the code fails in the **with** scope Rollback is used.

Revit API | Transaction Groups

- An object to execute multiple Transactions without cluttering the Undo commands list

- The transaction can Start and Assimilate the internal Transactions

- In Python use the with statement to be sure to handle the database correctly

Transaction Group example

```
with TransactionGroup(doc, “GroupName”) as tg :

        tg.Start()

        with Transaction(doc, “TransactionName”) as t:

                t.Start()

                # do something

                t.Commit()

                t.Start()

                # do something else

                t.Commit()

        tg.Assimilate()
```

Revit API | Create

- Document.Create for project
  - Walls
  - Floors
  - FamilyInstances
  - …
- Document.FamilyCreate for family
  - Extrusions
  - Sweeps
  - Lofts
  - …

Revit API | GeometryElement

- To create, change or delete a geometry object in Revit API it is not necessary to open a Transaction

- GeometryElement is the geometrical representation of a Revit object

- An element can contain more than one GeometryObject

Revit API | GeometryObject

- Quite often in Revit API it is necessary to create the GeometryObject that defines a Revit object to create it
- The kind of GeometryObject needed may vary depending on the nature of the Revit object
- It is possible to use Dynamo geometry objects for this purpose but they must be converted first

Revit to Dynamo | Conversions

- Revit
  - XYZ (point)
  - XYZ (vector)
  - GeometryObject
  - Revit ElementType

- Dynamo
  - ToPoint()
  - ToVector()
  - ToProtoType()
  - ToDSType()

Dynamo to Revit | Conversions

- Dynamo
  - Point
  - Vector
  - Geometry Object
  - RevitNodes

- Revit
  - ToXyz()
  - ToXyz()
  - ToRevitType()
  - UnwrapElement(element)

Next Steps

GitHub / DynamoDS

- Open project
- Wiki pages (lost of pieces are missing)
- Examples and resources
- Report a bug
- Propose a different approach

Zero Touch Essentials

- Load a DLL into one session
- New shelf in the Dynamo Library
- Written using C#
- There is no “new” keyword
- All the method are static

- Multi-return nodes using Dictionary<string, object>
- Pay attention when creating geometry objects (dispose the variables or encapsulate in “using”)
- Objects life cycle for larger applications

Garbage Collector

- Dynamo GC is different from .NET GC

- Dynamo GC delete objects at the end of a cycle and it calls the Dispose() method if it is implemented

- .NET GC frees memory when it is necessary

- Dynamo GC comes before .NET's GC (except in case of crash or errors)

Trace and Element Binding

- In general at each iteration objects are deleted and recreated and this could be an issue
  - An object can affect other objects
  - Changing an object could be cheaper than a new creation cycle
- Trace register the ID of an object in the Thread Local Storage (TLS) and lookup to it to re-associate the object of previous executions
- Mark attribute [RegisterForTrace(ID)] on the method

Custom Nodes UI

- Implement a NodeModel object
- Override the methods
- WPF to generate the node interface
- Create ViewExtensions to customize menus, node appearance and behavior or introduce new features

Myths and Truths

- Don’t need to know how to code to use Dynamo
- Dynamo is only for modeling
- Dynamo = Revit API
- Once you go .NET you never go back

- Need to understand how to build logical structures
- Dynamo geometry engine is very powerful but it can be used for data mining also
- Not all the methods are available
- It works with the localized version of Revit parameters names must be in the same language
- Revit is just the first big application but it’s not the only one
- The more one knows the better
- Broader choice of tools to work with

AUTODESK®

Make anything™