SD322695

EXPLORING PYTHON NODES IN DYNAMO (WIP)

Tadeh Hakopian
HKS
BIM Manager and Design Technology Specialist

Learning Objectives

- Learn about the Python node in Dynamo and how to configure it
- Learn how to execute code in Python with statements and conditions
- Learn how to debug your Python script when errors appear
- Learn how to create Python scripts for Dynamo that can create new editing possibilities unavailable with regular Dynamo or Revit tools

Description

Learn how to use Python in Dynamo to enable more editing options than ever before. Python is one of the fastest-growing scripting languages and can be used for daily tasks in your own projects. Dynamo can let you directly input Python code into your scripts to do things regular nodes can’t. This course will lead you through how to plan, edit, and execute your own scripts with Python for Dynamo. Learn about the essentials of setting up your own Python script, and edit geometry, sort data lists, write content to Revit software, and much more. With Python, you can unleash the potential in your projects so come and see what’s possible.

Speaker(s)

Tadeh Hakopian leverages BIM, VDC and Design Technology to provide his teams with impactful tools for project success. He has over 8 years of experience in the AEC field developing methods and practices to enhance project outcomes. With a background in Architecture he has worked with designers, engineers and contractors in all phases of building design and construction. Over the years he has been a part of large, complex projects in Commercial, Sports, Education, Healthcare and Residential sectors. His current focus is on design automation, data insights in projects and comprehensive workflows that come full circle in planning project life cycles. He is an active speaker at conferences including Trimble Dimensions, ATG Midwest University, BILT NA, BIM Forum and his local community meetups. Current Professional Goals Help move the AEC profession into new horizons using value driven solutions and innovative research.

TABLE OF CONTENTS

Table of Contents

About this Class

Everyone should be able to try out coding so I created this class as a starting point. Sometimes the barrier feels high and you don't know where to start which is where Dynamo comes into the picture.

For those in the AEC field who are interested in coding you can use the Dynamo plug in which is a great utility that makes the most out of visual scripting with Revit.
With Dynamo you can create your own scripts and execute the commands and write to Revit. This includes Python language support which is easy to learn and has an active community around it.

This class is meant to be a a showcase of how Python can be used in Dynamo for practical everyday tasks. This includes design exploration, modifying your model, integrating software tools, data reporting, document setup and more. After taking this class for yourself you will see what it takes to get started with Python including the essentials of syntax and deploying code.

It's wonderful that we have all these tools available for us and a lot of open source content to go with it. Hopefully with this class you can get a good start and a sense of what is possible for your own projects. Trust me it's easy.

Learning Objectives

Introduction

This class is meant to orient a novice with using Python code for Dynamo.
That includes project setup, configuring your test environment, working through examples and additional context for problem solving.

Learn about the Python node in Dynamo and how to configure it

The Python node has virtually all the functionality of the Revit API at its disposal. This class will show you what the properties of the Python node and how you can use it in the Dynamo environment.

Learn how to debug your Python script when errors appear

Coding comes with debugging. We will go through examples of what goes right and wrong in code and typical fixes. Debugging is an important part of the process as errors are routine and problem solving is essential to successful coding.

Learn how to execute code in Python with statements and conditions

Like all coding languages you have to set certain parameters to execute your script. Python makes this process straight forward and intuitive. We'll dive into how to set up statements and how to use them in a comprehensive way with improvised and boiler plate code.

Learn how to create Python scripts for Dynamo that can create new editing possibilities unavailable with regular Dynamo or Revit tools

The point of this class is to take all that Python coding knowledge to make your own scripts and solve problems in your projects. We will go into different use cases of Python in Dynamo, solutions to modeling and editing dilemmas and what is possible with Python that might not be so easy with typical Dynamo and Revit workflows.

Prerequisites

Ideally you would be prepared with some knowledge of how Dynamo and Python coding functions before taking this class. The training includes basic examples of each. To get the most out of your time try out these courses to get a grounding in the class before starting.

- Python Basics from Google - https://developers.google.com/edu/python/set-up
- Dynamo Training courses - https://www.autodesk.com/autodesk-university/class/Dynamo-Dummies-Intro-Dynamo-and-How-It-Interacts-Revit-2014
- Python in Dynamo basics courses - https://www.autodesk.com/autodesk-university/class/Untangling-Python-Crash-Course-Dynamos-Python-Node-2017

Computer and Software Setup

This is class is structured as a demonstration but you can follow along with your own computer. It is recommended to do this after the course since it will be recorded and meant to be repeatable.

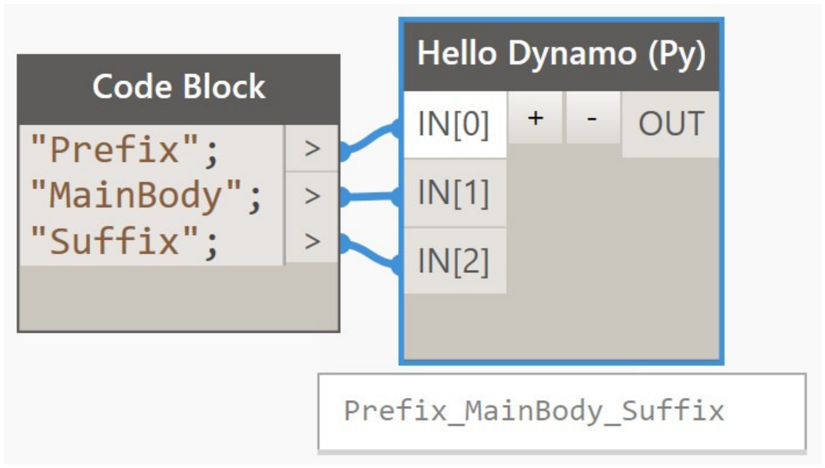
a. Revit 2019 or higher and a computer that can run the software.
b. Dynamo 2.0 or later
c. Dynamo Packages (as described in examples)
d. Python 3.0 or later
install - http://purcellconsult.com/python-installation-tutorial/
install 2 - https://github.com/pasadena-python-lab/pylab-classes/blob/master/installation_guide.md
e. Python IDLE

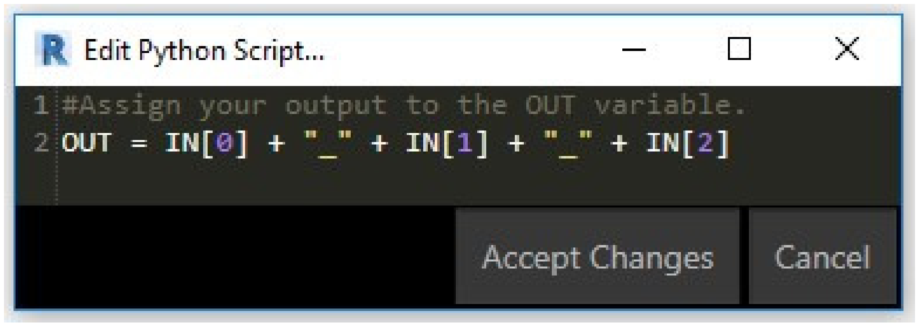
Exercise 0 - Warmup

This exercise will orient us to the Python node and how it works.
Open Dynamo and create a code block and a Python Node.
Python Node needs inputs from external nodes to work so you have to write them into the node.

We want to concatenate (combine) strings to make one string.
In this example our code block has text strings as inputs into each section of the Python node.
In Python all you have to do is list the output based on inputs.

By creating the instructions in Python you are all set to add your strings and get a combined result. This process is the most basic example of how all scripting in Python works in Dynamo.

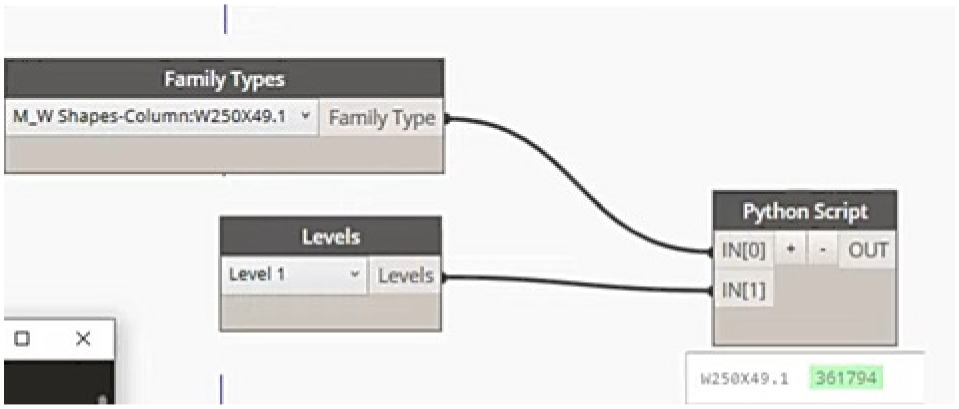


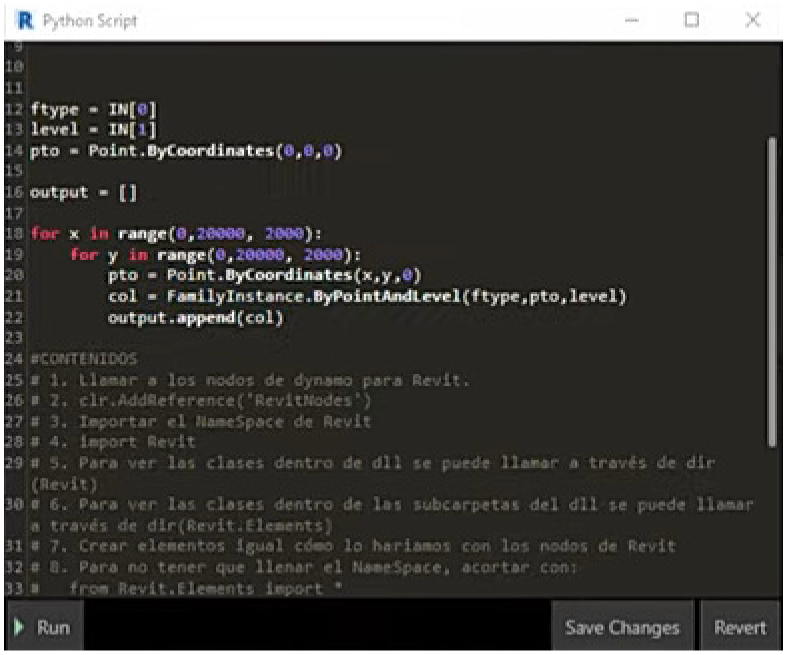


```
#Assign your output to the OUT variable.
OUT = IN[0] + "_" + IN[1] + "_" + IN[2]
```

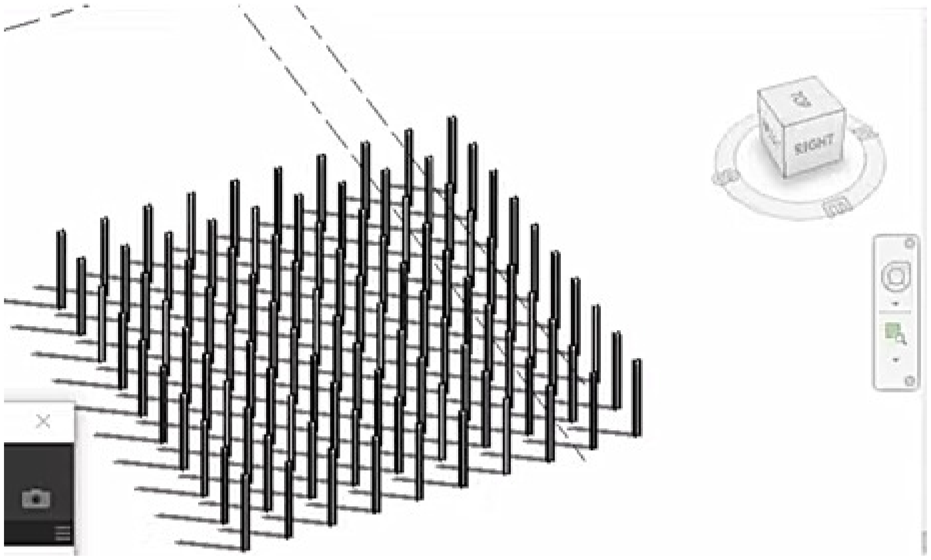
Exercise 1 – Write to Revit

Dynamo is great for creating content that can go into Revit.
With the Python node you can create inputs to set typical content in a given project.
In this example you input columns as the family and the level as the horizontal plane.
From here all you need is a for loop to run some commands.
The for loop will iterate through a range that you state and create a series of points in a grid arrangement. Those grid points will then be used to set your families into place.



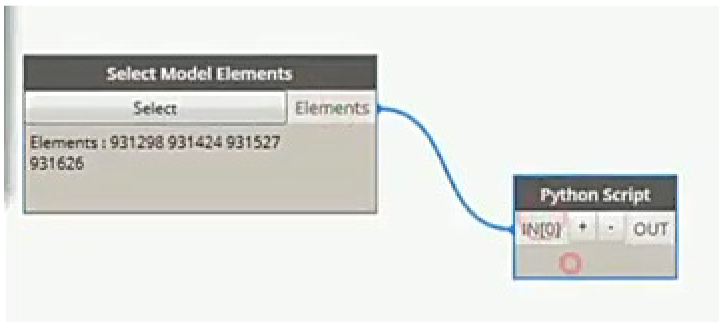


```
ftype = IN[0]
level = IN[1]
pto = Point.ByCoordinates(0,0,0)

output = []

for x in range(0,20000, 2000):
    for y in range(0,20000, 2000):
        pto = Point.ByCoordinates(x,y,0)
        col = FamilyInstance.ByPointAndLevel(ftype,pto,level)
        output.append(col)

#CONTENIDOS
# 1. Llamar a los nodos de dynamo para Revit.
# 2. clr.AddReference('RevitNodes')
# 3. Importar el NameSpace de Revit
# 4. import Revit
# 5. Para ver las clases dentro de dll se puede llamar a través de dir
(Revit)
# 6. Para ver las clases dentro de las subcarpetas del dll se puede llamar
a través de dir(Revit.Elements)
# 7. Crear elementos igual cómo lo hariamos con los nodos de Revit
# 8. Para no tener que llenar el NameSpace, acortar con:
#    from Revit.Elements import *
```

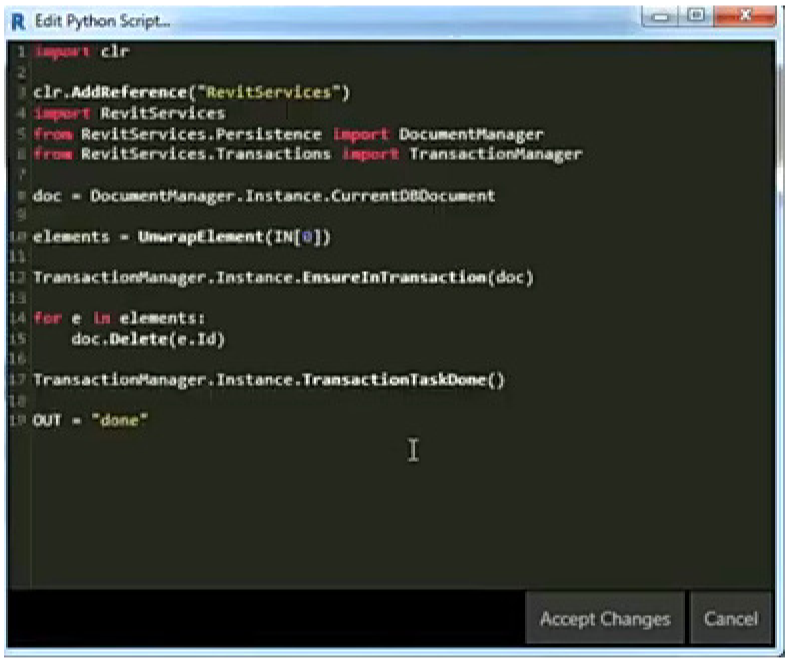

RIGHT

Exercise 2 – Expedite Modifications

Using the Revit API with Python nodes can save you a lot of time and effort.
One example of this is taking a basic command and giving it a ‘boost’. For example we can create a script that can bypass additional steps and get to the point. In this case it will be the delete tool. By importing modules and unwrapping elements you can set up a node that can bypass pin locks. In the demonstration below the columns are selected and input to the Python node. Run the script and all the pinned columns are deleted.



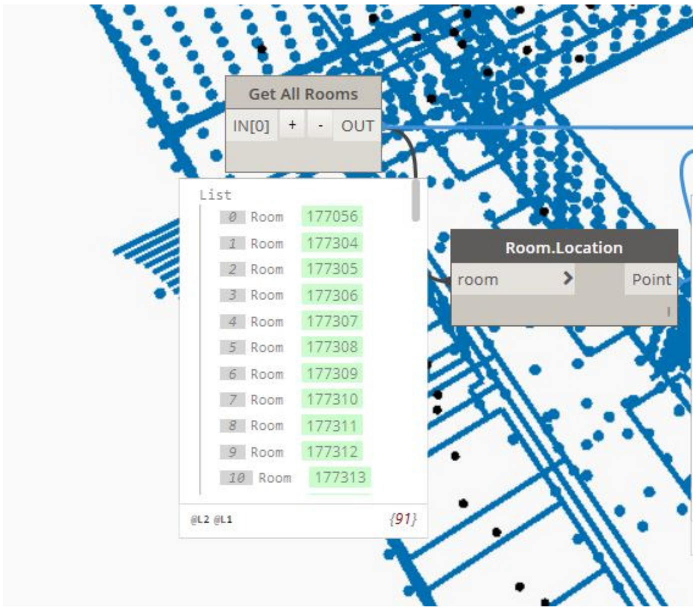
Elements : 931298 931424 931527 931626



```
import clr

clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager

doc = DocumentManager.Instance.CurrentDBDocument

elements = UnwrapElement(IN[0])

TransactionManager.Instance.EnsureInTransaction(doc)

for e in elements:
    doc.Delete(e.Id)

TransactionManager.Instance.TransactionTaskDone()

OUT = "done"
```

Exercise 3 – Data Reporting from Your Model

Data is a critical part of any BIM model. However it is often underutilized since accessing the data isn't always easy. If you can make the data access more convenient then there is a lot of opportunity to explore with that content. One of the most data rich assets is the Room elements in Revit. With a Python script you can get all the room location for a start.

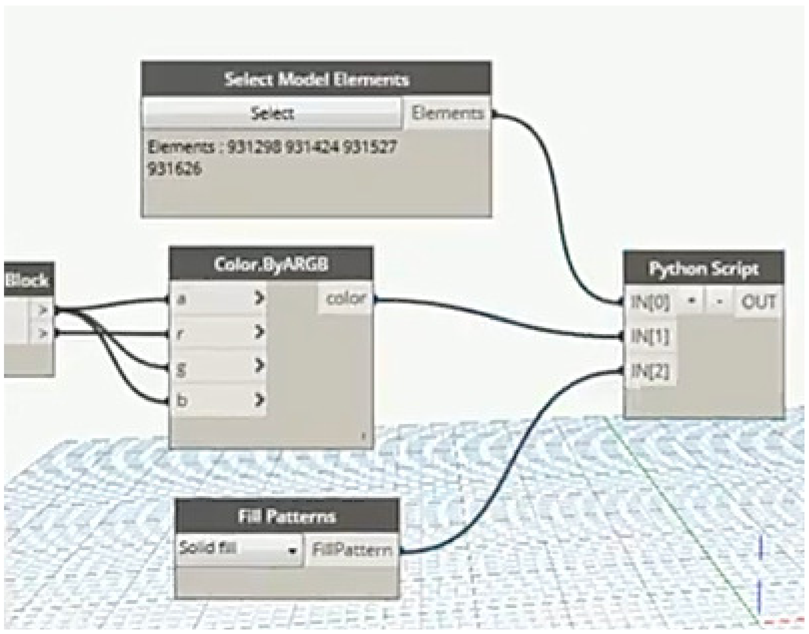


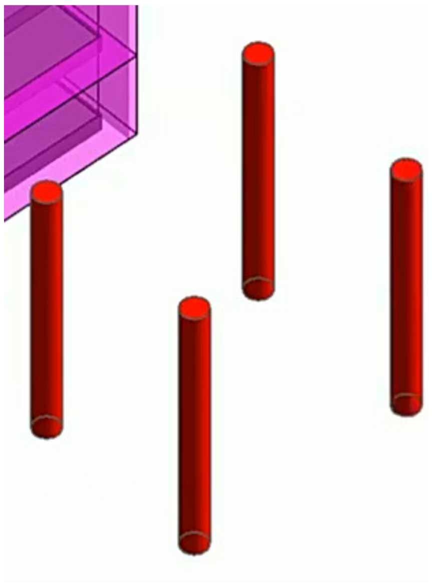
```
List
0 Room 177056
1 Room 177304
2 Room 177305
3 Room 177306
4 Room 177307
5 Room 177308
6 Room 177309
7 Room 177310
8 Room 177311
9 Room 177312
10 Room 177313
@L2 @L1 {91}
```

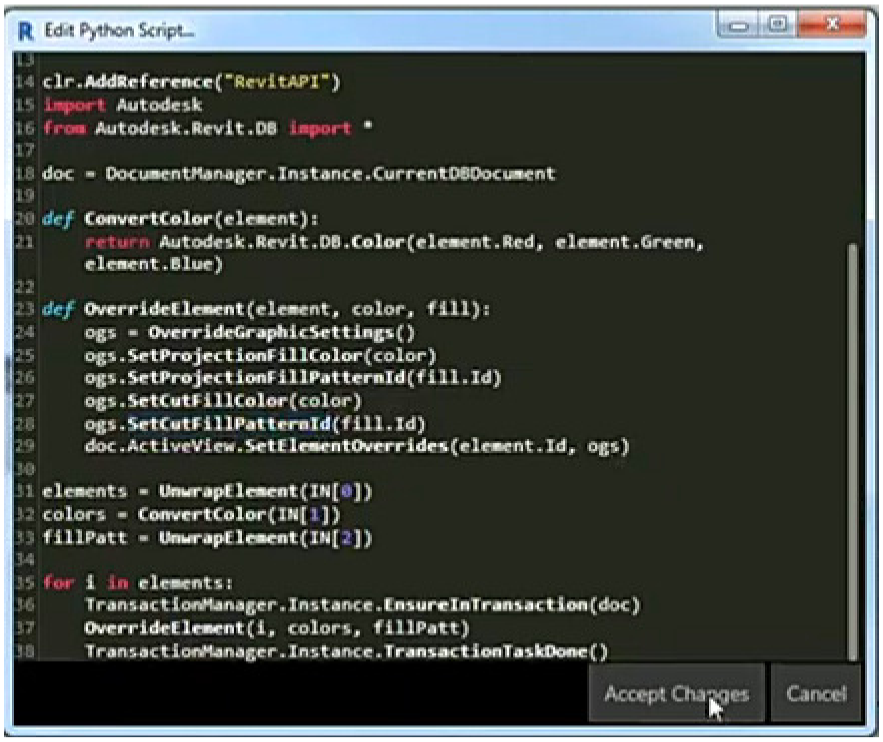
```
import clr

clr.AddReference("RevitServices")
from RevitServices.Persistence import DocumentManager
doc = DocumentManager.Instance.CurrentDBDocument

clr.AddReference("RevitAPI");
from Autodesk.Revit.DB import FilteredElementCollector
from Autodesk.Revit.DB import SpatialElement
from Autodesk.Revit.DB.Architecture import Room

collector = FilteredElementCollector(doc)

collection = collector.OfClass(SpatialElement).ToElements()
rooms = []
for element in collection:
        if (element.GetType().Equals(Room)):
                rooms.append(element)

OUT = rooms
```

Exercise 4 – Change Model Settings

Consolidating changes to your model can be a great time saver. Long scripts can be difficult to interpret and custom nodes may not always work. With the Python code language you can list all the changes you want to make to your model within in the same script. In this example we take the input of several elements and change the color pattern on them.



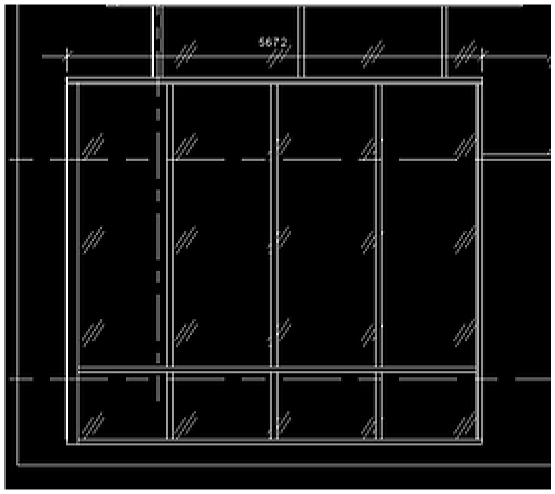


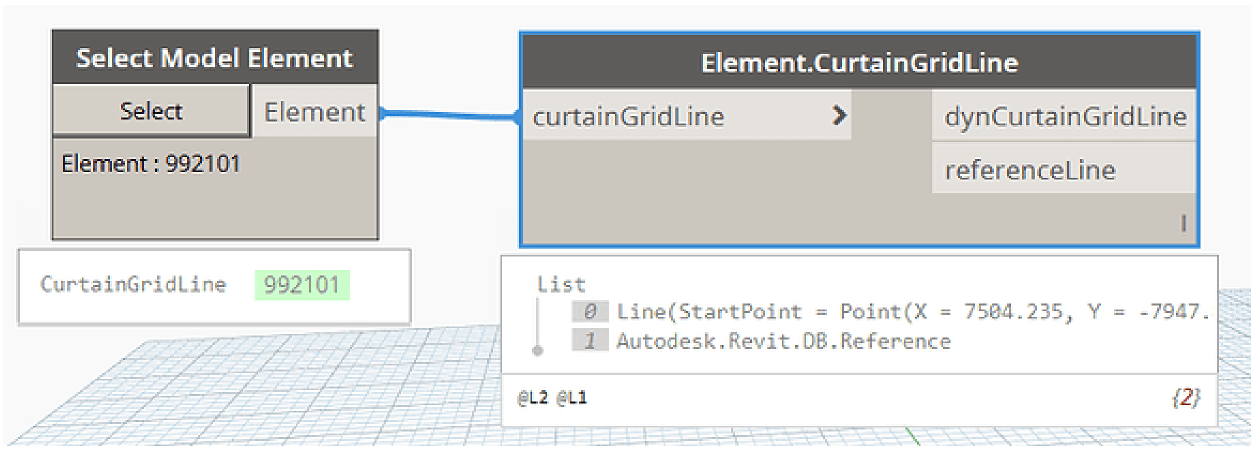


```
clr.AddReference("RevitAPI")
import Autodesk
from Autodesk.Revit.DB import *

doc = DocumentManager.Instance.CurrentDBDocument

def ConvertColor(element):
    return Autodesk.Revit.DB.Color(element.Red, element.Green,
    element.Blue)

def OverrideElement(element, color, fill):
    ogs = OverrideGraphicSettings()
    ogs.SetProjectionFillColor(color)
    ogs.SetProjectionFillPatternId(fill.Id)
    ogs.SetCutFillColor(color)
    ogs.SetCutFillPatternId(fill.Id)
    doc.ActiveView.SetElementOverrides(element.Id, ogs)

elements = UnwrapElement(IN[0])
colors = ConvertColor(IN[1])
fillPatt = UnwrapElement(IN[2])

for i in elements:
    TransactionManager.Instance.EnsureInTransaction(doc)
    OverrideElement(i, colors, fillPatt)
    TransactionManager.Instance.TransactionTaskDone()
```

Additional Examples

The possibilities with coding go on. You can even use it to model for you. With a Python script along with curtain grids you can dimension the model automatically.

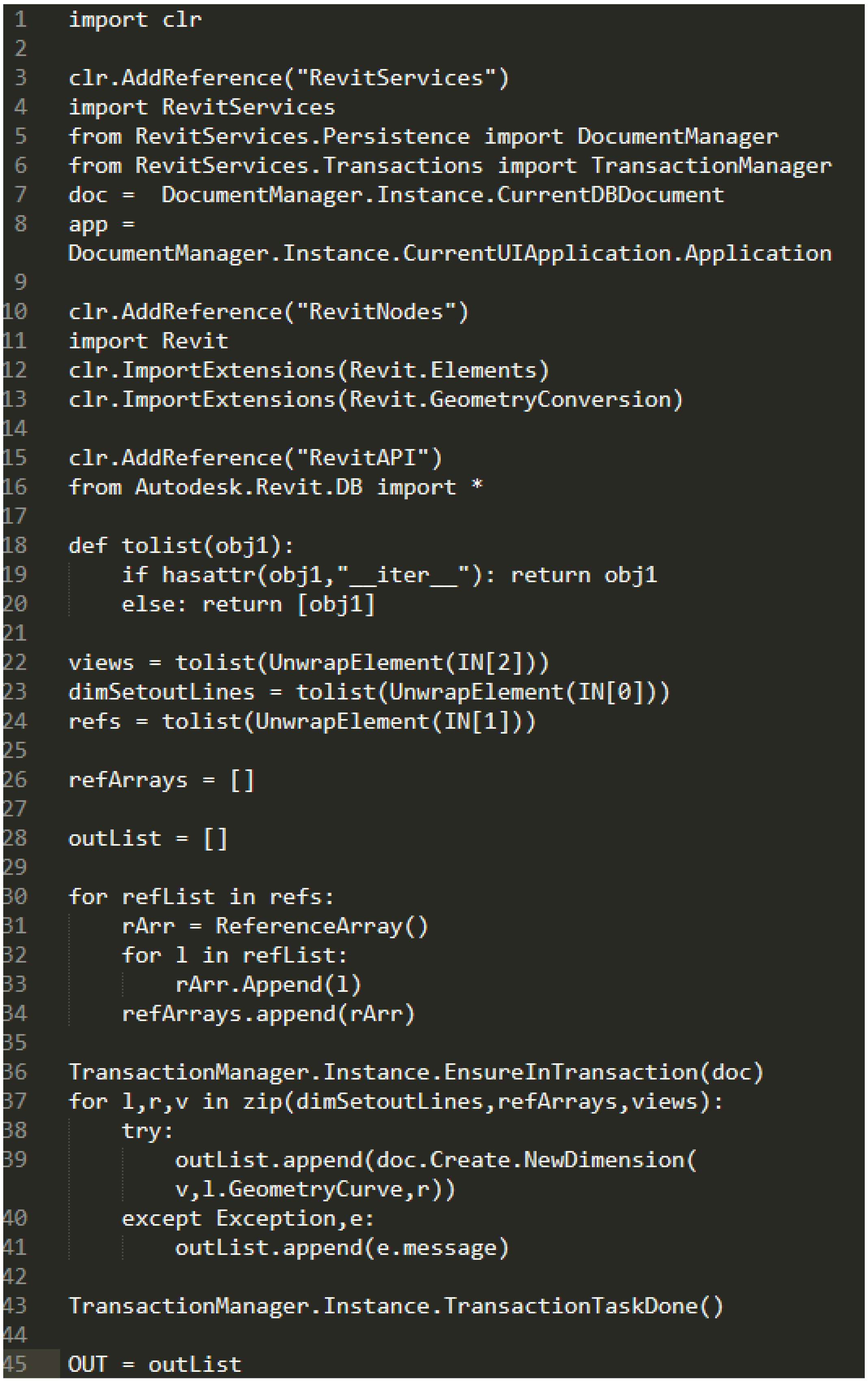




Element : 992101

CurtainGridLine 992101

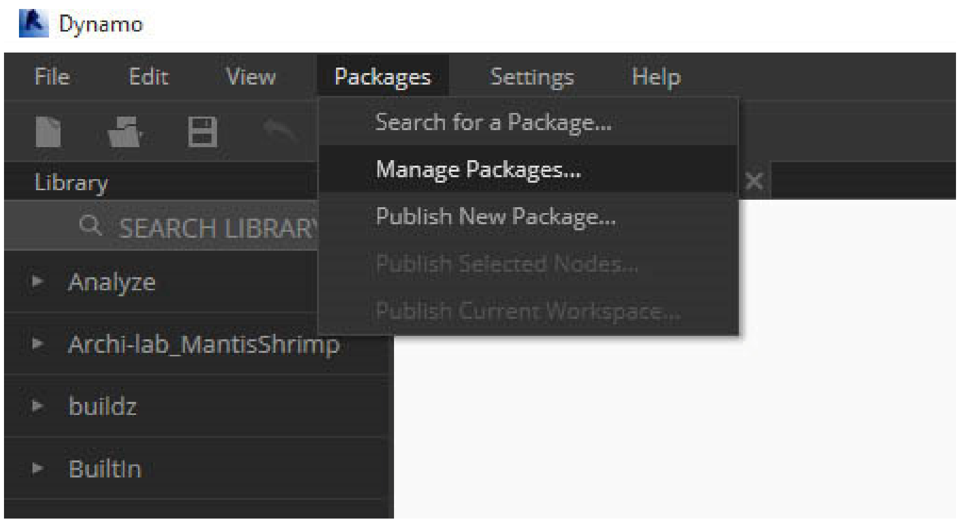
```
List
 0 Line(StartPoint = Point(X = 7504.235, Y = -7947.
 1 Autodesk.Revit.DB.Reference
@L2 @L1                                        {2}
```

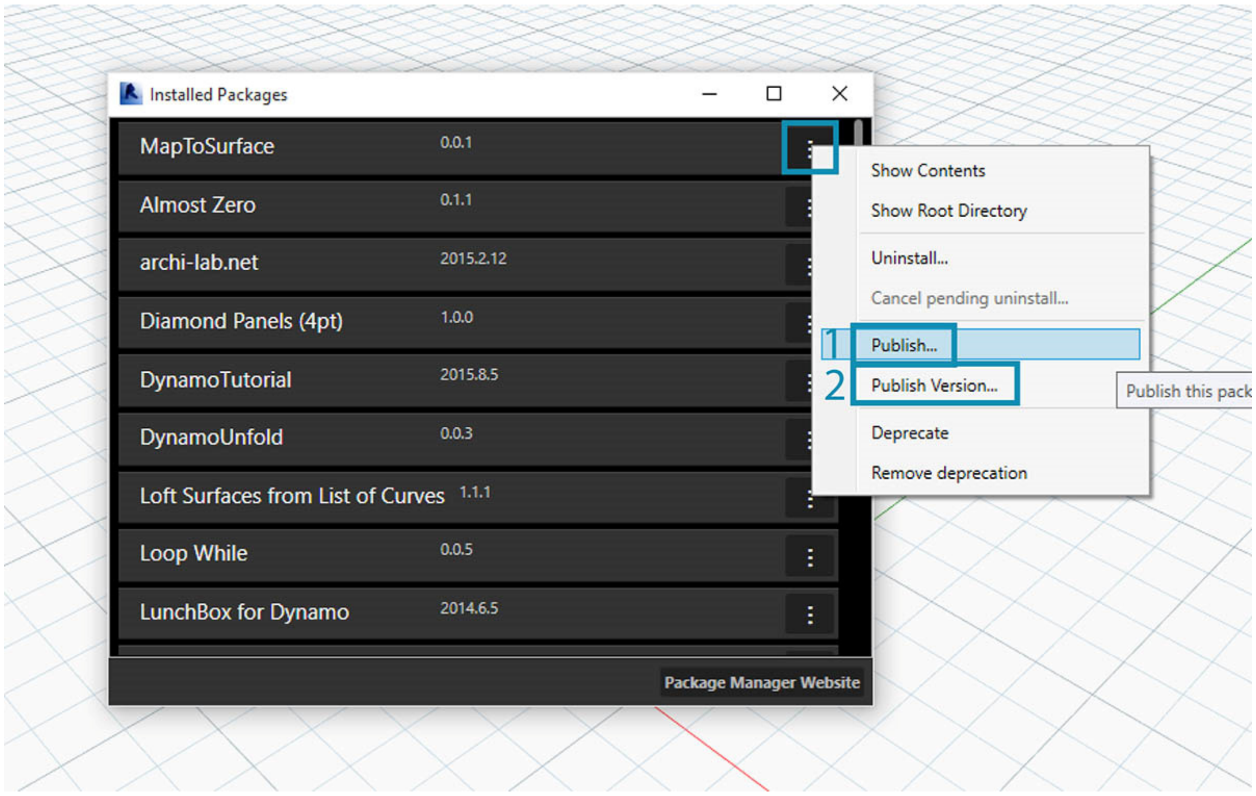


```python
import clr

clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager
doc =  DocumentManager.Instance.CurrentDBDocument
app = DocumentManager.Instance.CurrentUIApplication.Application

clr.AddReference("RevitNodes")
import Revit
clr.ImportExtensions(Revit.Elements)
clr.ImportExtensions(Revit.GeometryConversion)

clr.AddReference("RevitAPI")
from Autodesk.Revit.DB import *

def tolist(obj1):
    if hasattr(obj1,"__iter__"): return obj1
    else: return [obj1]

views = tolist(UnwrapElement(IN[2]))
dimSetoutLines = tolist(UnwrapElement(IN[0]))
refs = tolist(UnwrapElement(IN[1]))

refArrays = []

outList = []

for refList in refs:
    rArr = ReferenceArray()
    for l in refList:
        rArr.Append(l)
    refArrays.append(rArr)

TransactionManager.Instance.EnsureInTransaction(doc)
for l,r,v in zip(dimSetoutLines,refArrays,views):
    try:
        outList.append(doc.Create.NewDimension(v,l.GeometryCurve,r))
    except Exception,e:
        outList.append(e.message)

TransactionManager.Instance.TransactionTaskDone()

OUT = outList
```

Package and Deploy your own Code

Don't let all this code sit around your computer. Upload it and distribute with custom packages. Dynamo has a built in package manager to help you do this.





Limitations

As you continue to use Python and Dynamo you will be able to create your own code from the ground up. It is worth mentioning that not everything is going to be easy to solve. Sometimes what you want to do might end up being more complex than you realize.

Here's some things of what you can expect:

Dynamo Iterative

With every release or update of Revit the API can change a little bit. For anyone used to coding plugins or scripts for Revit this is a known issue. If the API function changes then your scripts have to be updated in order to work with newer releases.

Python in Dynamo relies on the Revit API

Dynamo lets you interface with the Revit API and Python nodes can let you use the API functions directly in the code. As much previously the API content can change and your code will need updates. However if you are coding then you always need to keep up to date on the Revit API releases and how they function.

Python node script is different from regular Python script

While mostly the same there are a few limitations with Python in Dynamo compared to regular scripting. First there is no 'print' command just an 'OUT' command. Because this is all visual scripting all content made has to exist in a node which has inputs and outputs. Python is no exception so there is an OUT which you need to address. Not a big hurdle you just have to remember to have a different execution process with Python in Dynamo but that can lead to more debugging issues.

Script deployment can be an issue at larger organizations

Making scripts for yourself or a small group is easy to maintain. The problem is when you have hundreds of people across multiple computer types and offices. You may need to create script that are adapted to scaling in an environment like that.

You may be creating this code solo within a given team

There is a big community online of people who are coding with Dynamo but you may be a community of one in your office. If scripts don't work that usually means you are the only one fixing problems so be prepared for that workload.

Additional Learning

Dynamo Primer

Dynamo Forums

Autodesk University

LinkedIn Learning

YouTube Channels

Automate the boring stuff with Python

Contact Information

Twitter: https://twitter.com/tadeh_hakopian

Linkedin: https://twitter.com/tadeh_hakopian

Github: https://twitter.com/tadeh_hakopian