

AS224711-L

# Hack it 'til You Crack It: An Introduction to Python in Dynamo for Revit Users

John Pierson

**Parallax** Team, Inc.

## Magnificent Lab Assistants:

Sol Amour, Jason Boehning, Patrick Podeyn, Marcello Sgambelluri, Carl Storms

### Learning Objectives

- **Gain** an introductory understanding of Python in Dynamo.
- **Learn** how to translate and understand Python terms in a Revit way.
- **Understand** how to implement standard Python methods in Dynamo.
- **Learn** how to deconstruct other's Python code for additional learning and modification.

### Description

With Python, Dynamo users can access the Revit API (application programming interface) in a whole new way and create additional functionality to make the making of things easier. This lab will not be an overly complex look at Python from a computer programmer's point of view. Instead, it will introduce the Python environment within Dynamo from a Revit user's perspective. As a Dynamo user, the speaker has struggled with the same scenarios and has used Python in ways never anticipated. This lab will cover key concepts for Python, including syntax, indentation, object types, and iteration, while relating it all back to Revit terms.

### Speaker



John Pierson is a Design Technology Specialist at Parallax Team, a full-service Implementation firm. Specializing in the creation of custom tools and workflows for the AEC industry, John has become well known for his Dynamo-based solutions. Currently, John manages several Dynamo packages that are available to the Dynamo community. This includes Rhythm, Bang! and Duct Tape totaling over 45,000 downloads.

In addition to the creation of automated workflows and custom tools, John is a Revit certified professional for all disciplines. As a Revit certified professional, he provides custom content creation, project-based modelling, and Revit training.

John is an avid contributor to the Dynamo community, being a moderator on the Dynamo forum and a contributor to the Revit Forum. John is also a top speaker at many events including, Autodesk University, BiLT Conferences and various user groups everywhere. Find him at [sixtysecondrevit.com](http://sixtysecondrevit.com), [@60secondrevit](https://twitter.com/60secondrevit) or on [parallaxteam.com](http://parallaxteam.com)

## Introduction

This class is meant to be an introduction to Dynamo to get the average Revit user up and running as quickly as possible. I structured this class based on my own experiences hacking away at Python until it worked for me. I wanted to give you some useful tips and comments I have picked up over the years.

### There's a **slight** chance that...

- This class will get some terms incorrect. It happens to all of us. Python is a pretty big topic to cover when you are a Revit user first.
- We will reference more refined Python resources for the next steps or for answers to our questions.
- We will venture into some “questionable” territory from the point-of-view of “real programmers”. Don't worry though, it's going to be okay. 😊

### There's a **100%** chance that you...

- Will have fun learning about Python and gaining some insight to get started.
- Will make awesome stuff that can be used in Dynamo.
- Will learn at least one thing that you did not know before opening this handout.
- Will find out about great resources to venture down your Dynamo journey and grow.

## Resources (from people way smarter than me)

- [Diving Deeper – A Beginners Look at Python in Dynamo – Sol Amour](#)

Sol is a Python artist. His class from AU London really breaks down the concepts and introduces them in a very deep way. In addition to his AU handout, Sol started a Community Python Resource on Github that has been contributed to by many people. Maybe after this class you can be the next!

<https://github.com/Amoursol/dynamoPython>

- [Untangling Python: A Crash Course on Dynamo's Python Node - Gui Talarico](#)

Gui's class is an amazing resource on using Python for Revit to its fullest. He covers deep Python concepts and provides info on how to utilize outside development environments. He is also an active contributor to the AEC community when it comes to Python and the API. His API resources are something I use on the daily.

<http://www.revitapidocs.com/> & <https://apidocs.co/>

- [Python Next Steps – Danny Bentley](#)

Danny's YouTube is an amazing resource for next steps in your Python journey. He has everything on there from sample Python workflows to how to get Python for Dynamo to work in Visual Studio Code, (A very popular IDE), enabling additional predictive text for coding.

[Danny's YouTube](#) & [Danny's Twitter](#)

## But, what if my code is a mess and not perfect?

If we worry about having perfect code that will in fact prohibit us from getting better and growing. Ian and Anthony are great people who do awesome things and they have gone down this path as well.



**Anthony Hauck**  
@anthonyhauck

Following

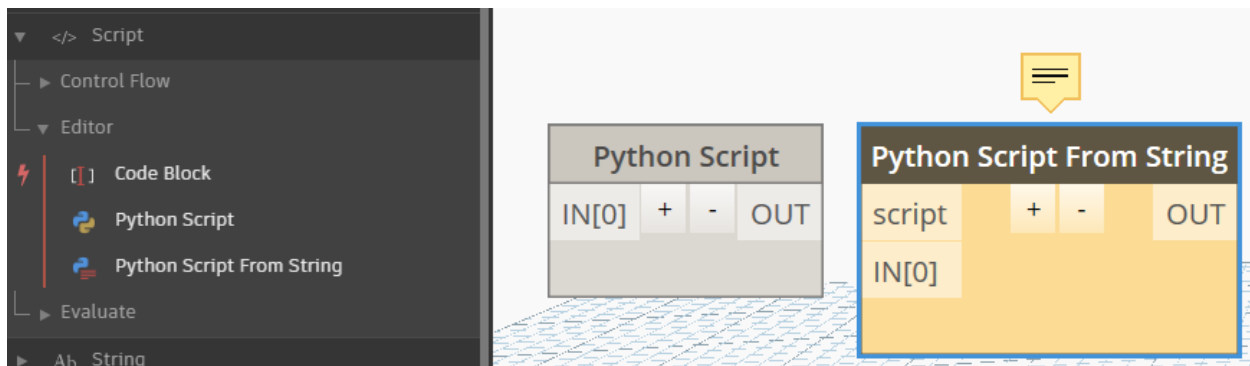
Replying to @didonenov @60secondrevit @arch\_laboratory

We should be open about these experiences.  
@ikeough helped when he said to me,  
"Almost every bug fix ends with the coder thinking he or she is an idiot for missing the obvious." It's discouraging for people to think everyone else is a genius creating prize-worthy code.

# HYPAR

## Why Python?

Python has been included in Dynamo for quite some time. This is achieved through special nodes that allow you to edit Python code on the fly.



## What is Python?

Python is an open-source, general-purpose programming language. Since Python does not require compiling to be tested, it is an excellent candidate for Dynamo use as it provides instant gratification. [Revit Comparison: The process of using Python is like editing families, loading into a project for testing and repeating the process.](#)

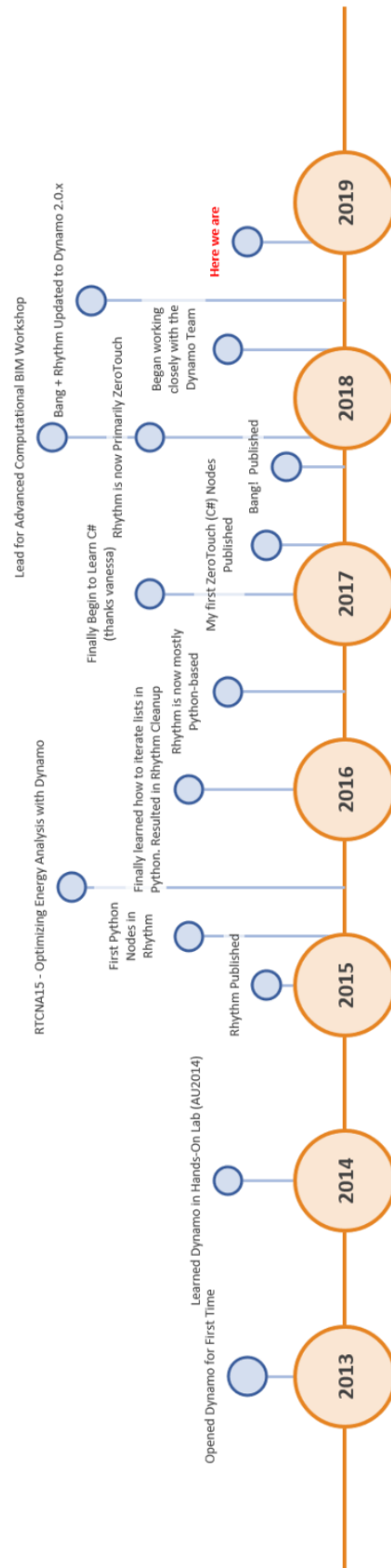
### Iron Python?

Iron Python is a version of Python that was created in C#. This allows for use of the Common Language Runtime (CLR) to speak to other .Net applications and libraries.

In Dynamo 2.0.1 we can use Iron Python 2.7.7 and the editor window allows us to really execute code on the fly.

## **A Natural Progression**

Every Dynamo user has their own version of their “Dynamo Journey”. Often a user learns Dynamo and builds quite a few scripts utilizing OOTB nodes and third-party packages from the package manager. Many of these packages are built utilizing Python, so there are a lot of examples. As a beginner Dynamo user becomes an intermediate-to-advanced user they long to enable more functionality in Dynamo or simply eliminate dependencies on third-party packages. Using the guidance, they have available and the rapid editability of the node, Python is most often the natural next step. I understand this journey just as much as you while you are reading this as I started out as a beginner as well. When I learned Dynamo in 2014, I began creating custom nodes and sought out the next steps.



## Getting Started

Dropping a Python node on the Dynamo canvas is the first step towards making your own custom functionality. Once we do that we can either double click or right click and edit the node.

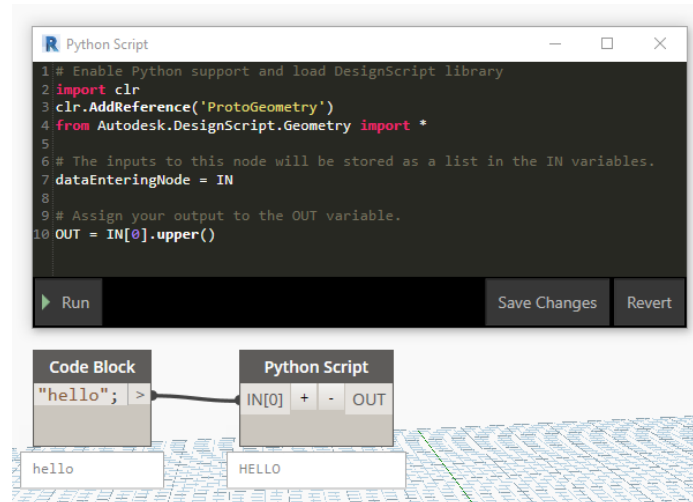
## String Modification

Not only can we add Revit functionality with Python, we can use a variety of libraries already available for us. To get started, we will modify strings (text) values in Python on top of Dynamo.

A simple google search for string methods in Python can yield this result.

[https://www.tutorialspoint.com/python/string\\_upper.htm](https://www.tutorialspoint.com/python/string_upper.htm)

This link tells us that we can take a string and apply the method “upper()” to it. This is how that looks in Python in Dynamo.

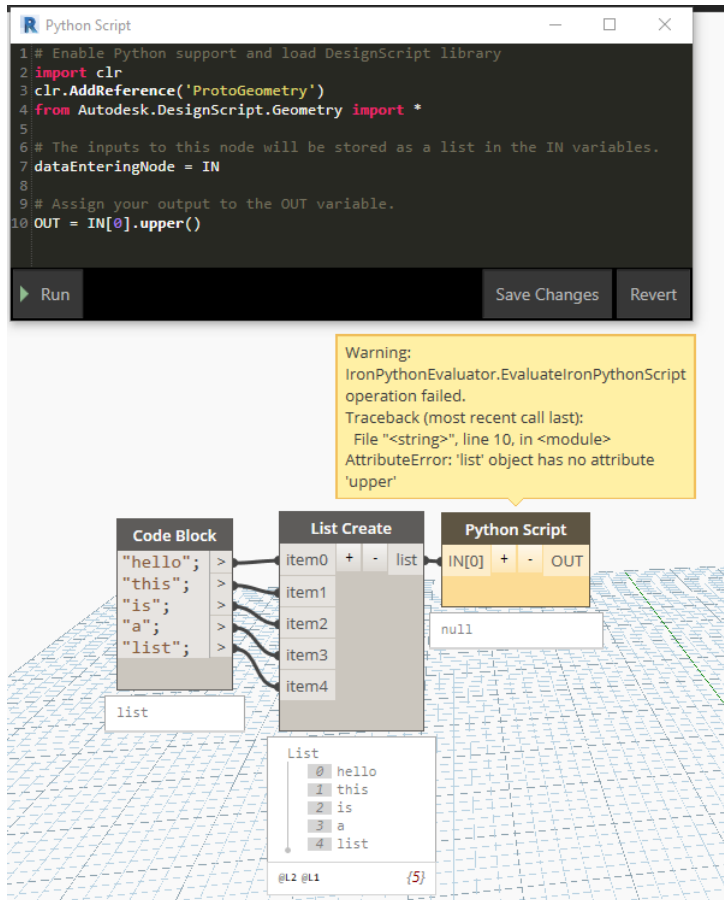


Using the code from the link we can see that it works as expected in Dynamo when used on a single string of “hello”.

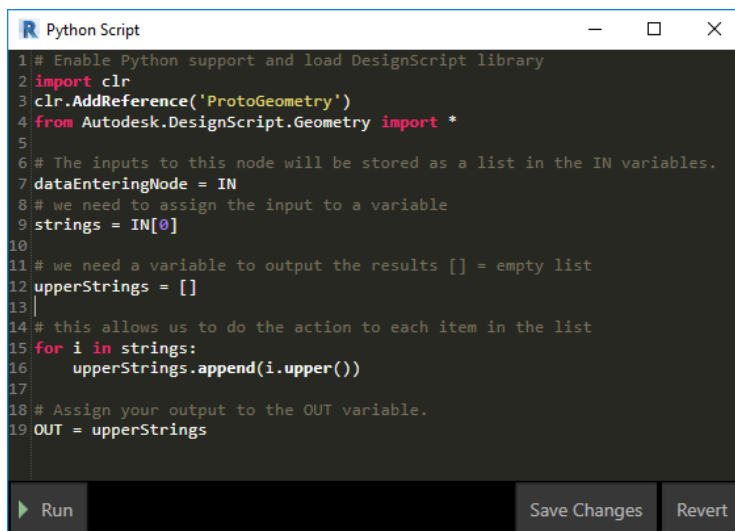
### Example Making a Single String Upper

## String Modification on Multiple

Reusing the previous code on a list of strings, we can see that we have an error. The reason this occurs is because we need to tell Python to “walk” along the list and do the action to each thing as an individual.



Error on Lists.



We can think of this as the same thing as selecting several families in Revit and attempting to look at their instance parameters. If you do this in Revit it will not give you a result unless they are all the same. The solution is to make an itemized schedule that looks at each item as an individual.

## Working with Lists

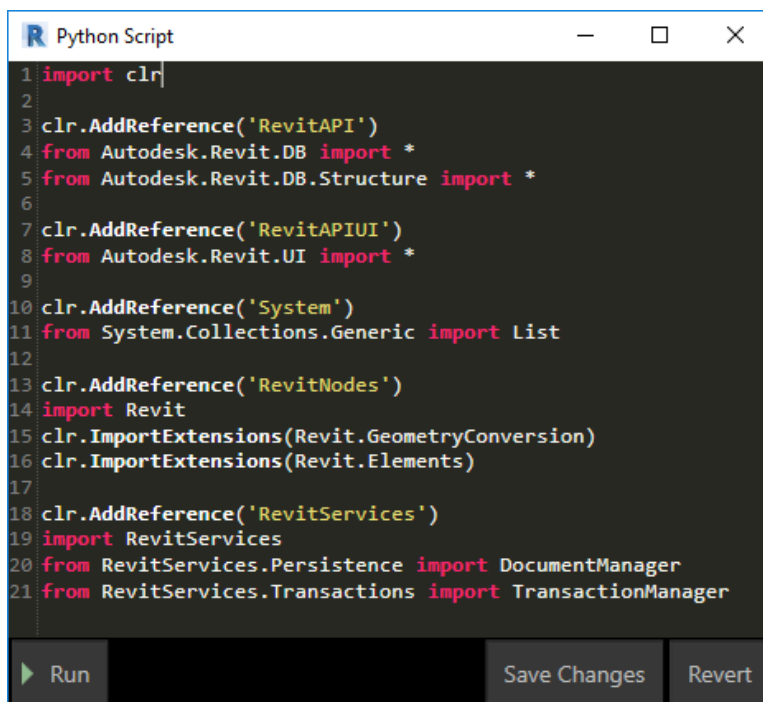
To work with lists in Python we can use a process called iteration or a “for each” statement. Essentially, we are telling Python to do an action to each thing in the input list. In the case of this example we are saying, “For each **string**,(i) in the **list of strings (strings)**, make them uppercase.”

## Importing Libraries to Do Awesome Things.

Imports are a way to load additional libraries into your code. [For a Revit user, this can be compared to loading content into your Revit file for your use.](#) Imports appear at the top of your Python window and can encompass a variety of libraries.

### Importing Revit Related Functionality

To access the Revit API and add additional functionality to Dynamo, we need to reference the Revit API. This is done by telling Dynamo where to look and what modules we are using. If you want to just import all the modules you can simply use an asterisk (\*) to do so. Unfortunately, this is considered not the “best” practice because you are bringing in a lot of stuff you may not need. [Revit comparison: This is like loading an entire family type catalogue when you only need one family type from it.](#) Even though this may not be the best practice, I find that this is a great way to get started as I would be crazy to expect everyone to understand what specific modules they need to reference. We can worry about that once you break through being a beginner Python user. Next, we will break down what each import is.



```

1 import clr
2
3 clr.AddReference('RevitAPI')
4 from Autodesk.Revit.DB import *
5 from Autodesk.Revit.DB.Structure import *
6
7 clr.AddReference('RevitAPIUI')
8 from Autodesk.Revit.UI import *
9
10 clr.AddReference('System')
11 from System.Collections.Generic import List
12
13 clr.AddReference('RevitNodes')
14 import Revit
15 clr.ImportExtensions(Revit.GeometryConversion)
16 clr.ImportExtensions(Revit.Elements)
17
18 clr.AddReference('RevitServices')
19 import RevitServices
20 from RevitServices.Persistence import DocumentManager
21 from RevitServices.Transactions import TransactionManager
  
```

*Line 3-5: add a reference to the Revit API and bring in functionality for Revit and Structural.*

*Lines 7-8: Bring in UI Related functionality. (Dialogue boxes, etc.)*

*Lines 10-11: Bring in list related functionality.*

*Lines 13-15: This loads the libraries from the Revit category in Dynamo. Essentially this allows you to use the nodes already made in Dynamo within Python. (E.g. GetParameterValueByName)*

*Lines 18-21: Load the Dynamo libraries for making changes to the Revit document. This is known as a transaction.*

Typical Imports for Revit Interaction.



## Transactions?

Transactions can be conceptualized in the same manner that they obtain their namesake from. Think about buying a candy bar at a candy store. For you to obtain the candy bar, you need to start a transaction with the cashier. Within that transaction certain things take place. This includes, finding out the total cost, handing over the money, ringing up the item and closing out the sale. The same thing goes for Revit. We start the transaction, do some things and apply the changes to the Revit model. Note: A transaction in Dynamo will almost always result in having an undo available in Revit.



Charlie Knows About Transactions!

## Imports are cool! But do I have to memorize that?

Absolutely not! With Dynamo 2.0.x, we now can define Python templates to load on the placement of a Python node. This further enables us to prototype with Python in a very quick way. (Shout out to Radu Gidei for implementing this!)

To add a python template, you can reference the following link,  
[http://dynamoprimer.com/en/10\\_Custom-Nodes/10-6\\_Python-Templates.html](http://dynamoprimer.com/en/10_Custom-Nodes/10-6_Python-Templates.html)

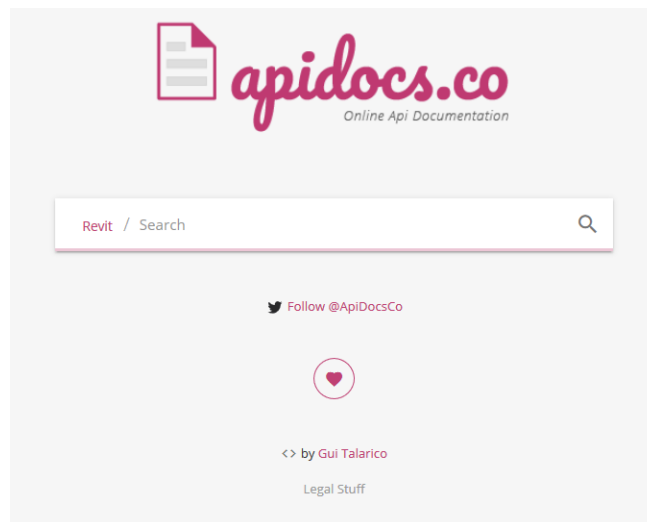
This process does require you to know your current username and path. Luckily, we can use Dynamo to obtain that and send the value right to the clipboard. Use the file, **AS224711-00-GetCurrentUserPythonTemplatePath\_end.dyn** in the data set.

## Finding Functionality to Add

Whenever a Dynamo user finds themselves wanting to learn Python, they often have an idea or goal in mind. Typically, you want to access or change data on a Revit element and simply could not find a node for it in other custom packages. Other times, users want to eliminate package dependencies altogether. (*Python nodes are unique in the sense that they are “embedded” in the DYN*)

### Find Properties and Methods of Revit Elements

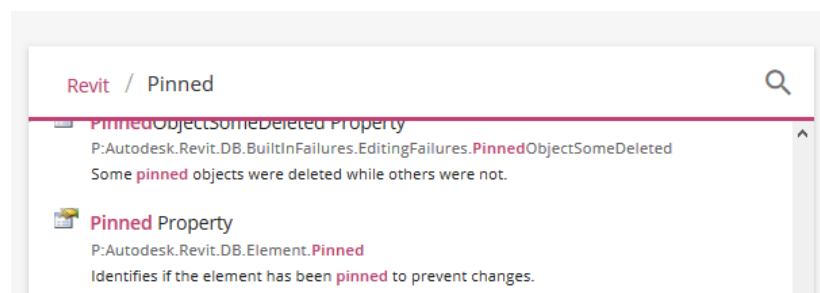
The absolute easiest method of finding out what you can do to a Revit element is to use the website, [APIDocs](https://apidocs.co). APIDocs is built by [Gui Talarico](https://github.com/GuiTalarico) and provides instant access to various API documentation for Revit, Rhino and Grasshopper. (*API is Application Programming Interface and is a set of definitions for a program*)



Apidocs.co landing page.

### Finding Element Properties

On APIDocs, you can search for properties based on what you want to achieve. In this example we want to access the **Pinned** property of elements with Python. Since this is possible using the UI, it should be possible through the API as well. Searching on API docs yields the following result.



We can see that Element has a property called pinned.

## What is “Autodesk.Revit.DB.Element”?

In Dynamo, there is a concept of a user-understandable Revit element and an API-understandable version. These are referred to as “Wrapped” and “Unwrapped” respectively. Let’s break that down a bit more by looking at a burrito.



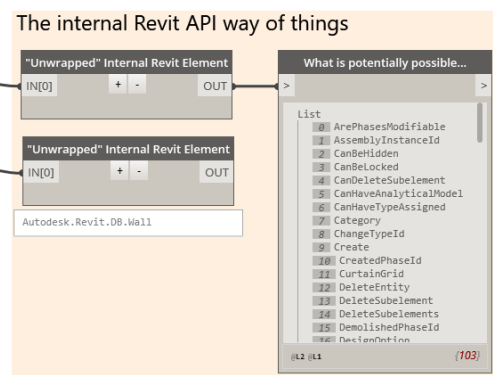
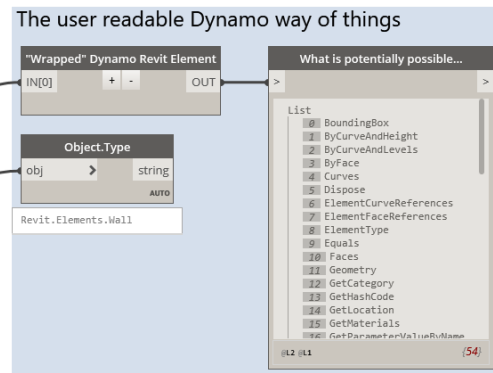
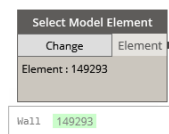
MMMmm.

As we can see on the outside of the “wrapped” burrito, we can only gain so much knowledge about it. Once we unwrap or cut open the burrito, we learn a whole lot more about it.

Now to see this in Dynamo terms.

In the Dynamo example to the right, we have the concept of a “wrapped” element known as **Revit.Elements.Wall**. This is a class of object and has its own properties and methods that have been exposed by the Dynamo team.

*(These are the nodes under the Revit bin in the library. They can also be used in a code block with Design Script)*



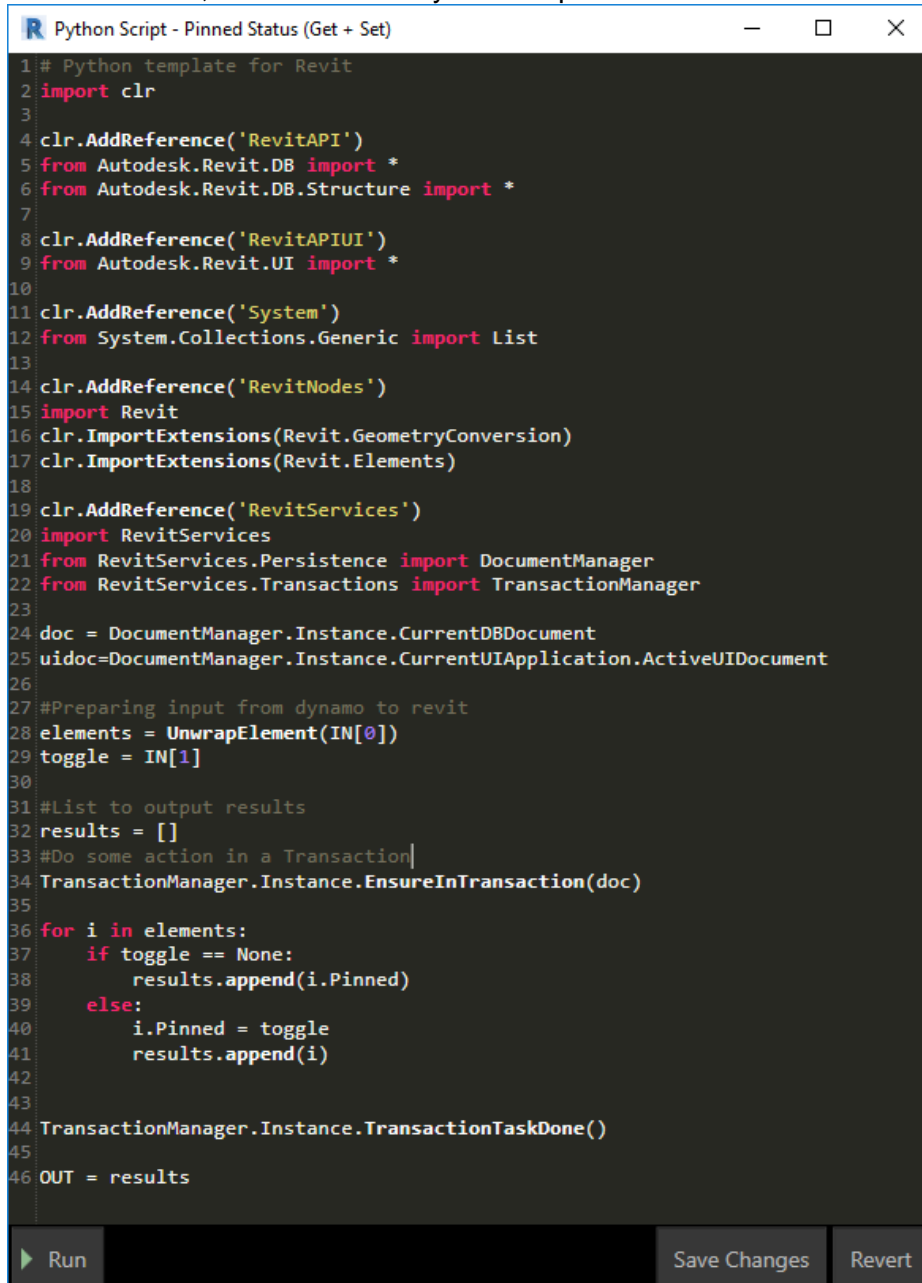
We also have the concept of an “unwrapped” element known as **Autodesk.Revit.DB.Wall**. This is the internal class for the wall element and has the API properties that we can use.

## Examples

Documentation for all examples from this class

### Element Pinned Status

Knowing if an element is pinned or changing its pinned status is not accessible via OOTB nodes at this time. So, we can build a Python script to do this.



```

1 # Python template for Revit
2 import clr
3
4 clr.AddReference('RevitAPI')
5 from Autodesk.Revit.DB import *
6 from Autodesk.Revit.DB.Structure import *
7
8 clr.AddReference('RevitAPIUI')
9 from Autodesk.Revit.UI import *
10
11 clr.AddReference('System')
12 from System.Collections.Generic import List
13
14 clr.AddReference('RevitNodes')
15 import Revit
16 clr.ImportExtensions(Revit.GeometryConversion)
17 clr.ImportExtensions(Revit.Elements)
18
19 clr.AddReference('RevitServices')
20 import RevitServices
21 from RevitServices.Persistence import DocumentManager
22 from RevitServices.Transactions import TransactionManager
23
24 doc = DocumentManager.Instance.CurrentDBDocument
25 uidoc=DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument
26
27 #Preparing input from dynamo to revit
28 elements = UnwrapElement(IN[0])
29 toggle = IN[1]
30
31 #List to output results
32 results = []
33 #Do some action in a Transaction
34 TransactionManager.Instance.EnsureInTransaction(doc)
35
36 for i in elements:
37     if toggle == None:
38         results.append(i.Pinned)
39     else:
40         i.Pinned = toggle
41         results.append(i)
42
43 TransactionManager.Instance.TransactionTaskDone()
44
45 OUT = results
  
```

The python script. (Note: you can copy from the dataset if need be)

## Show/Hide Grid Bubbles in View

Batch changing which side a grid bubble is showing on is awesome. It's even awesome-er when you build the Python yourself with switchable logic.

```

Python Script
1 # Python template for Revit
2 import clr
3
4 clr.AddReference('RevitAPI')
5 from Autodesk.Revit.DB import *
6 from Autodesk.Revit.DB.Structure import *
7
8 clr.AddReference('RevitAPIUI')
9 from Autodesk.Revit.UI import *
10
11 clr.AddReference('System')
12 from System.Collections.Generic import List
13
14 clr.AddReference('RevitNodes')
15 import Revit
16 clr.ImportExtensions(Revit.GeometryConversion)
17 clr.ImportExtensions(Revit.Elements)
18
19 clr.AddReference('RevitServices')
20 import RevitServices
21 from RevitServices.Persistence import DocumentManager
22 from RevitServices.Transactions import TransactionManager
23
24 doc = DocumentManager.Instance.CurrentDBDocument
25 uidoc=DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument
26
27 activeView = doc.ActiveView
28
29 #Preparing input from dynamo to revit
30 elements = UnwrapElement(IN[0])
31 toggle = IN[1]
32
33 #We need a transaction because we are making a change to the Revit model
34 TransactionManager.Instance.EnsureInTransaction(doc)
35
36 for i in elements:
37     if toggle == True: #allows us to swap between the options
38         i.ShowBubbleInView(DatumEnds.End1,activeView) #Show end 1 in the view
39         i.HideBubbleInView(DatumEnds.End0,activeView) # hide end 0 in the view
40     else:
41         i.ShowBubbleInView(DatumEnds.End0,activeView) #Show end 0 in the view
42         i.HideBubbleInView(DatumEnds.End1,activeView) # hide end 1 in the view
43
44 TransactionManager.Instance.TransactionTaskDone()
45
46 OUT = elements

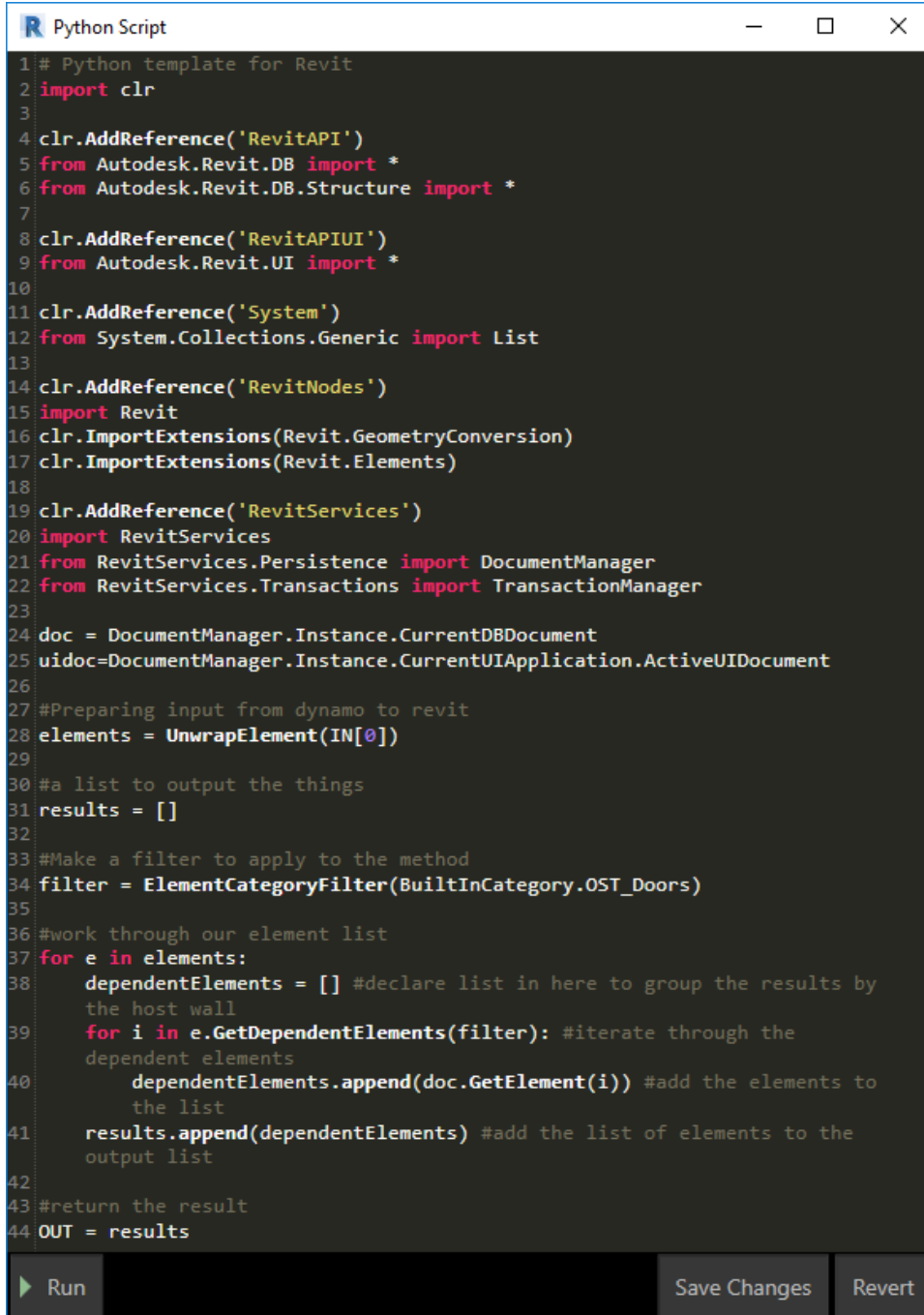
```

Run Save Changes Revert

The python script. (Note: you can copy from the dataset if need be)

## Get Doors in Walls Using New API Methods

I have seen several ways to get door counts in walls. This includes intersection tests, use of the “GetInserts” method and more. With Revit 2018.1 we were given access to a “dependent elements” property. So, we can use that to very quickly get doors.



```
1 # Python template for Revit
2 import clr
3
4 clr.AddReference('RevitAPI')
5 from Autodesk.Revit.DB import *
6 from Autodesk.Revit.DB.Structure import *
7
8 clr.AddReference('RevitAPIUI')
9 from Autodesk.Revit.UI import *
10
11 clr.AddReference('System')
12 from System.Collections.Generic import List
13
14 clr.AddReference('RevitNodes')
15 import Revit
16 clr.ImportExtensions(Revit.GeometryConversion)
17 clr.ImportExtensions(Revit.Elements)
18
19 clr.AddReference('RevitServices')
20 import RevitServices
21 from RevitServices.Persistence import DocumentManager
22 from RevitServices.Transactions import TransactionManager
23
24 doc = DocumentManager.Instance.CurrentDBDocument
25 uidoc=DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument
26
27 #Preparing input from dynamo to revit
28 elements = UnwrapElement(IN[0])
29
30 #a list to output the things
31 results = []
32
33 #Make a filter to apply to the method
34 filter = ElementCategoryFilter(BuiltInCategory.OST_Doors)
35
36 #work through our element list
37 for e in elements:
38     dependentElements = [] #declare list in here to group the results by
39                             #the host wall
40     for i in e.GetDependentElements(filter): #iterate through the
41         dependentElements.append(doc.GetElement(i)) #add the elements to
42         the list
43     results.append(dependentElements) #add the list of elements to the
44     output list
45
46 #return the result
47 OUT = results
```

Run Save Changes Revert

The python script. (Note: you can copy from the dataset if need be)

## Conclusion

Well, there we have it. All the info that I could cram in regarding Python based on my experience as a Revit and Dynamo user. I hope that this material helps you in your Dynamo journey and gave you at least a few new ideas/tools to work with. I hope that you go off and do awesome things in Dynamo with Python now and share them with your firms and the AEC community. I also wanted to thank you for considering my class and for making all this worth it. Go do awesome things!



**John Pierson**

**Design Technology Specialist**

**PARALLAX TEAM**

[johnPierson@parallaxTeam.com](mailto:johnPierson@parallaxTeam.com)

[sixtysecondrevit@gmail.com](mailto:sixtysecondrevit@gmail.com)

@60secondrevit