Scripting with RevitPythonShell in Revit / Vasari

Iffat Mai
Design Application Development Manager at Perkins + Will

CP3837-L

Background

- Design Application Developer at Perkins + Will (NY)
- Architecture Design Background
- CAD & BIM Manager
- Research & Development in BIM Technology
- B.S. in Architecture from MIT
- Speaker at AU on Revit API for the last two years.

Who are you?

- Architecture
- Engineering
- Construction
- Education
- Software Development
- Other

Have you worked with…

- Scripting (Rhino, Max, Lisp, AutoCAD, Maya, Sketchup…)
- Python / IronPython
- Revit API (VB.net, C#)
- Other Programming Language (C++)
- Never programmed before
- Revit (Architecture, Structure, MEP)
- Vasari

Class Summary

RevitPythonShell brings scripting ability to Autodesk® Revit software and Vasari. Designers now have the ability to interactively design and manipulate Revit elements using algorithm and computational logic. We will explore the Python structures, variables, data types, and flow control, and show how to use them to create scripts to control Revit elements dynamically.

Learning Objectives

At the end of this class, you will be able to:

- Become familiar using RevitPythonShell in Revit (Vasari)
- Learn basic Python programming
- Understand Revit API elements
- Use scripts to create and change geometry in Revit (Vasari)

Agenda

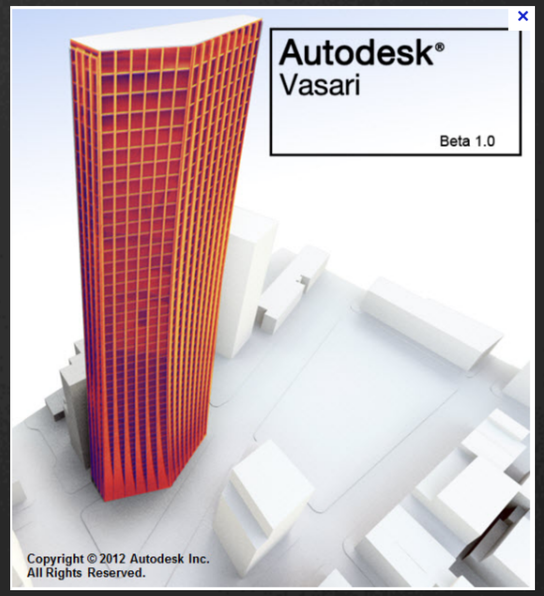
Topics	Time
Introduction	5
Python	20
Revit API	25
Revit Examples	30
Q & A	10

Software Install

- Revit 2013 or Vasari Beta
- .Net 4.0
- IronPython (2.7.3)
- RevitPythonShell for RAC2013
- RevitPythonShell for Vasari Beta
- RevitLookup tool
- Optional IDE
  - SharpDevelope 4.2
  - Python Tools For Visual Studio

Software Introduction

- Revit Architecture 2013
- Vasari Beta
- Revit API
- IronPython (Python)
- RevitPythonShell



IronPython

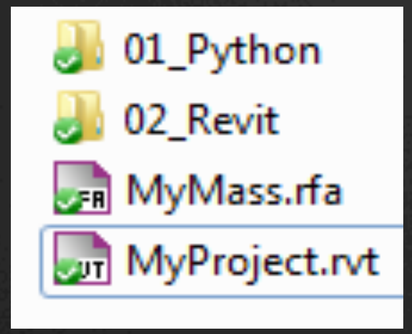
python™

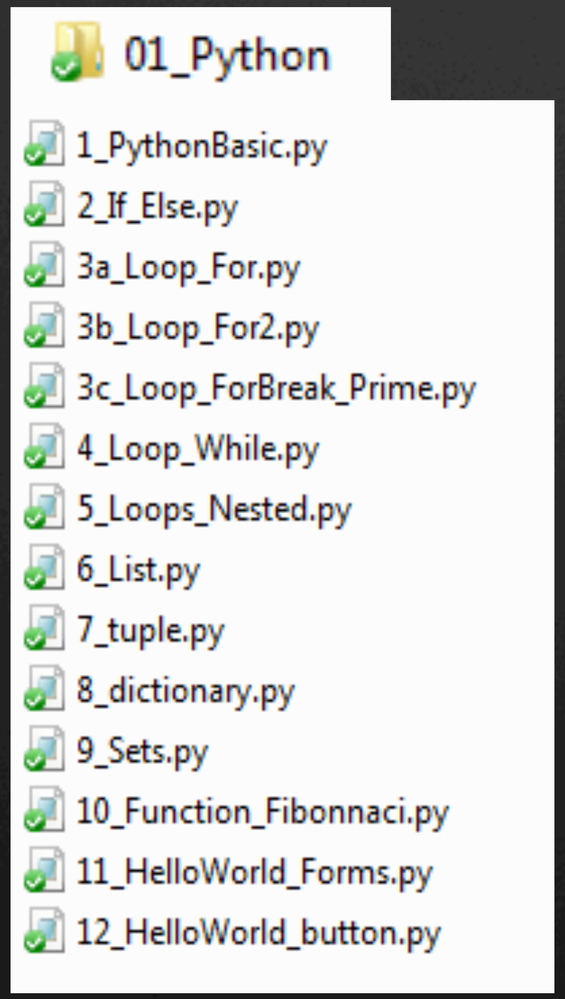
revitpythonshell

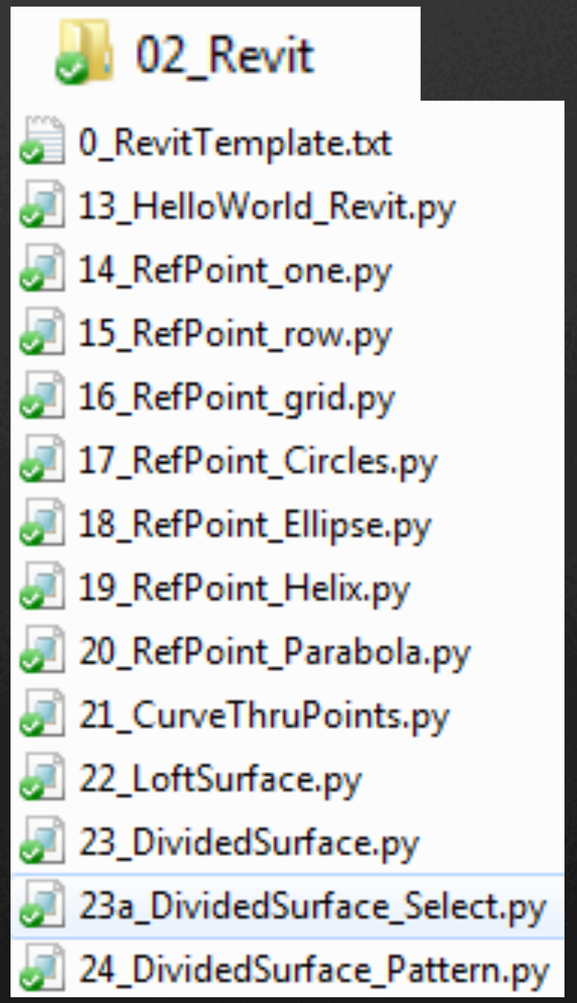
An IronPython plugin for Autodesk Revit Architecture

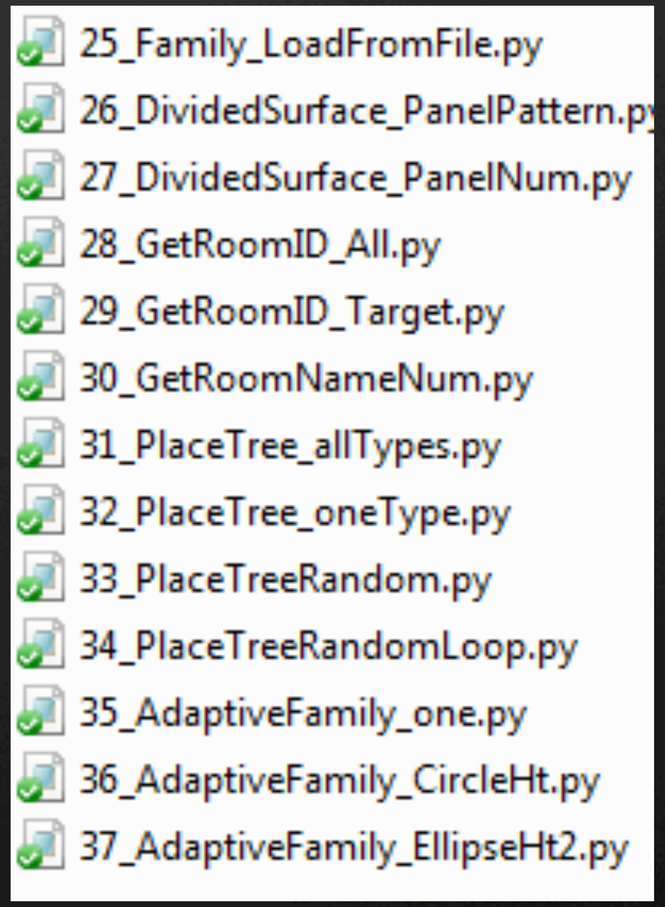
in Revit / Vasari

Dataset Folder



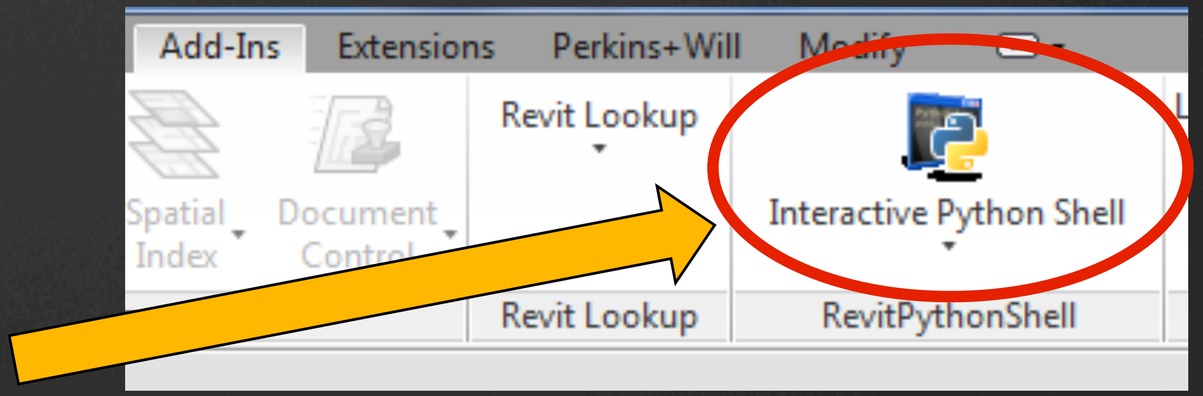


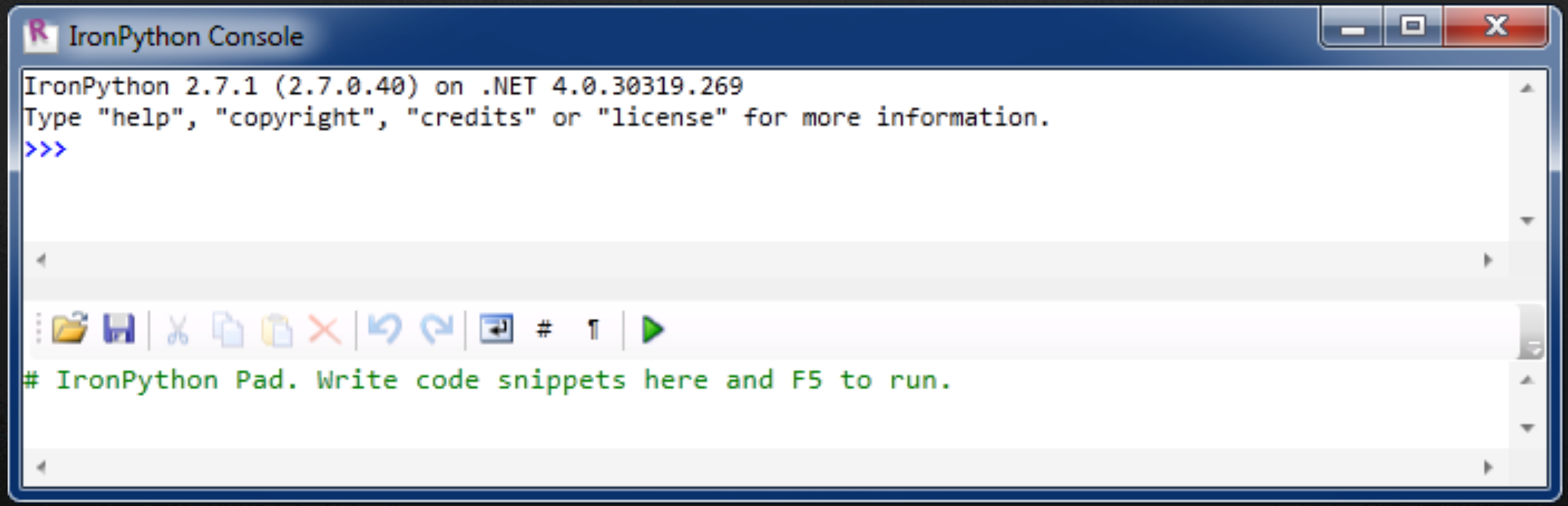




Revit Python Shell

- Open Revit (or Vasari)
- From the Add-ins tab, Select

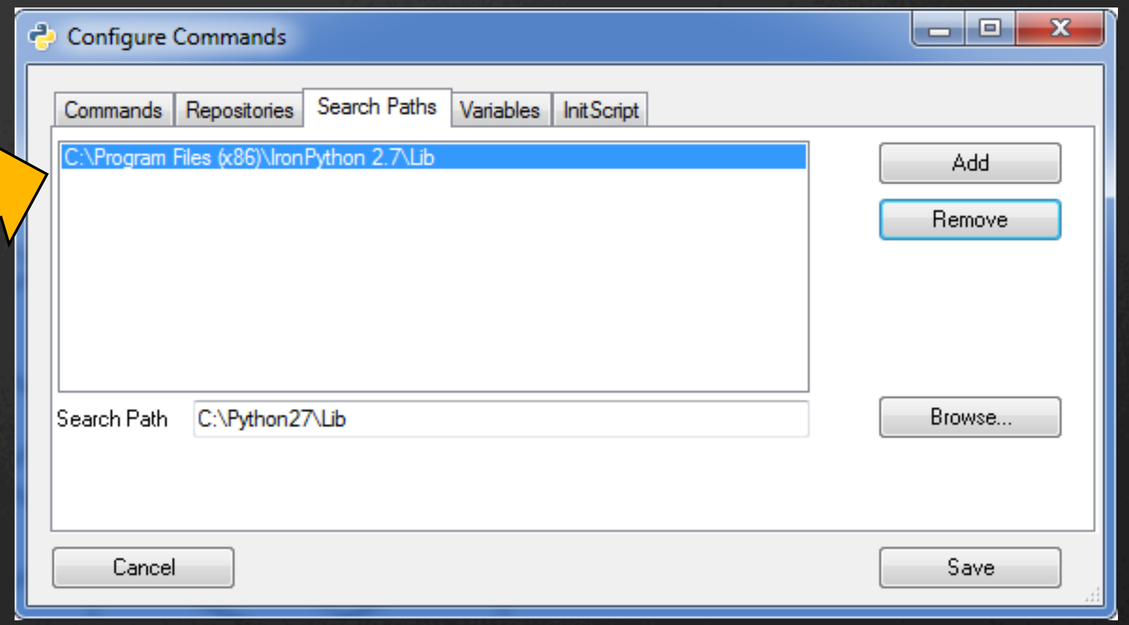




RevitPythonShell Configuration

- Select Configure
- Add IronPython library path

  C:\Program Files (x86)\IronPython 2.7\Lib



OR

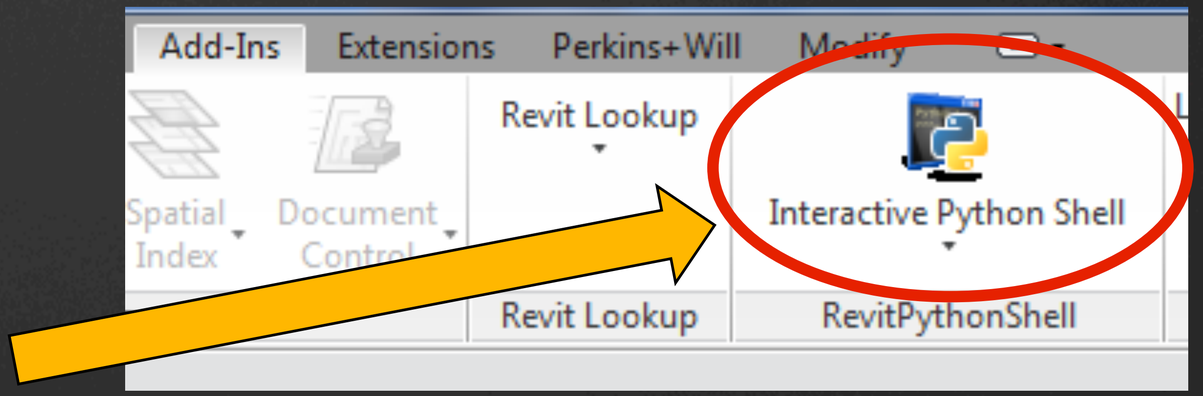
- Edit the configuration in RevitPythonShell.xml

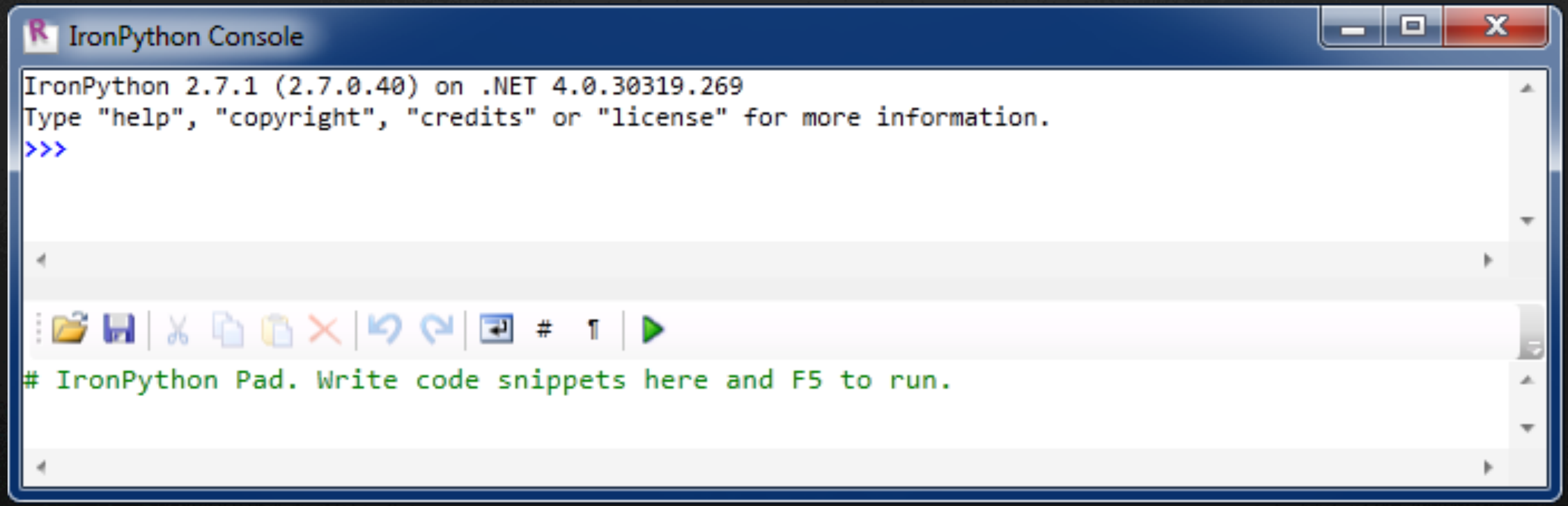
  C:\Users\YourLoginName\AppData\Roaming\RevitPythonShell2013
- Add the search path in the xml file.

  <SearchPath name="C:\Program Files (x86)\IronPython 2.7\Lib"/>

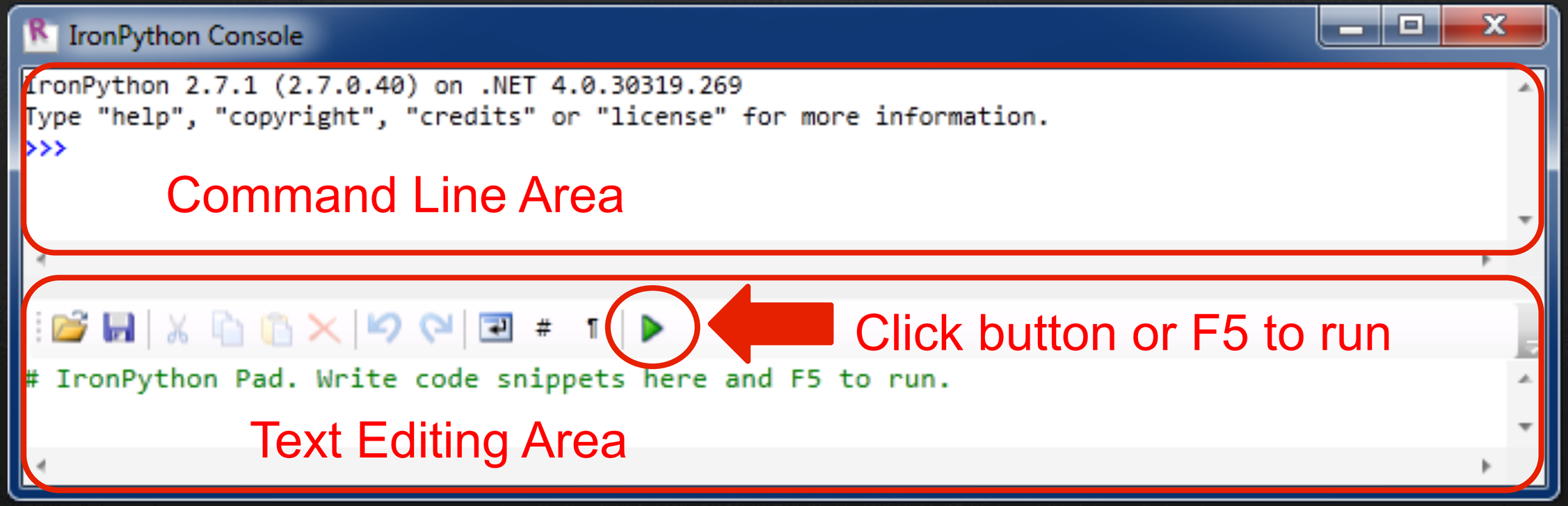
Revit Python Shell

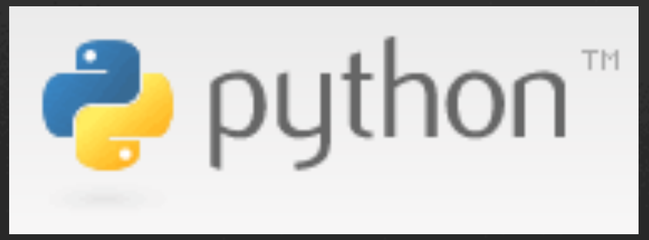
- Open Revit (or Vasari)
- From the Add-ins tab, Select





Revit Python Interactive Shell



Learn Basic  Programming

What is  ? What is IronPython ?

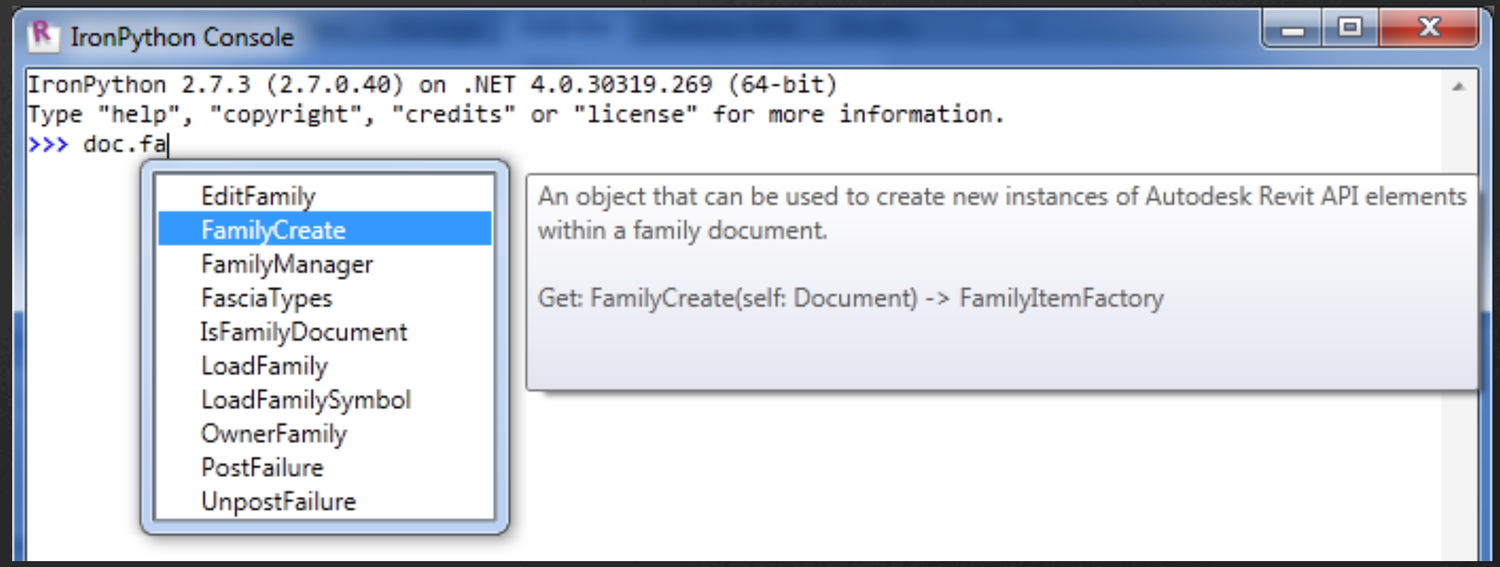
- Python is an open-source dynamic programming language that is fast and simple and runs on many different platform.
- IronPython is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework.

Python Reference

- **Python Programming Language – Official Website**
  - http://www.python.org/
- TutorialsPoint.com
  - http://www.tutorialspoint.com/python/index.htm
- IronPython – Official Website
  - http://ironpython.net/
- IronPython in Action
  - http://www.ironpython.info/index.php/Main_Page

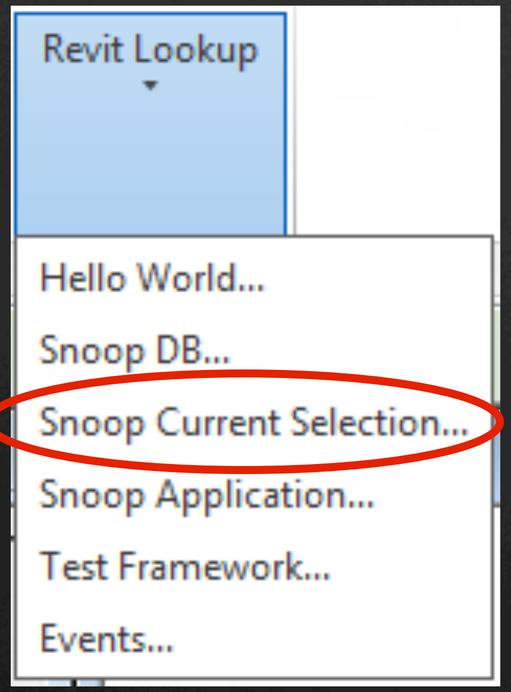
Where to find help?

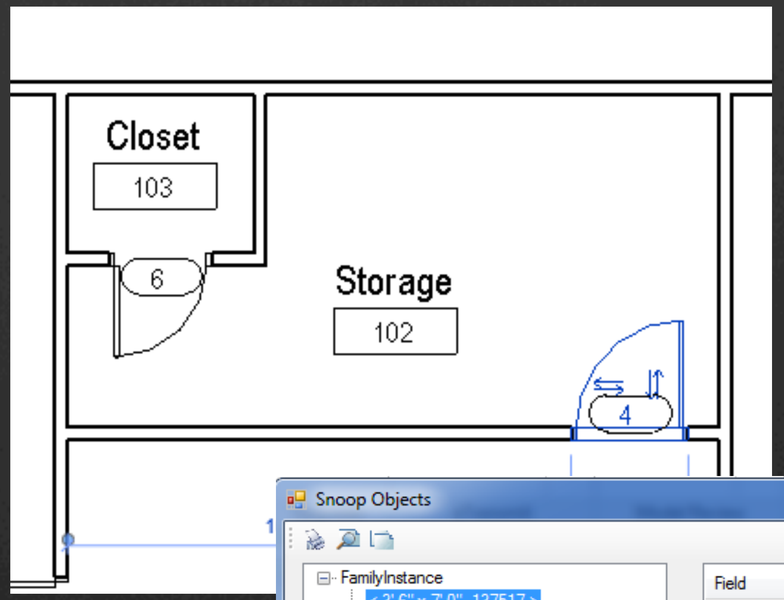
- help(object)
- dir(object)
- Activate Autocomplete* after the dot(.) using **CTRL + Spacebar**

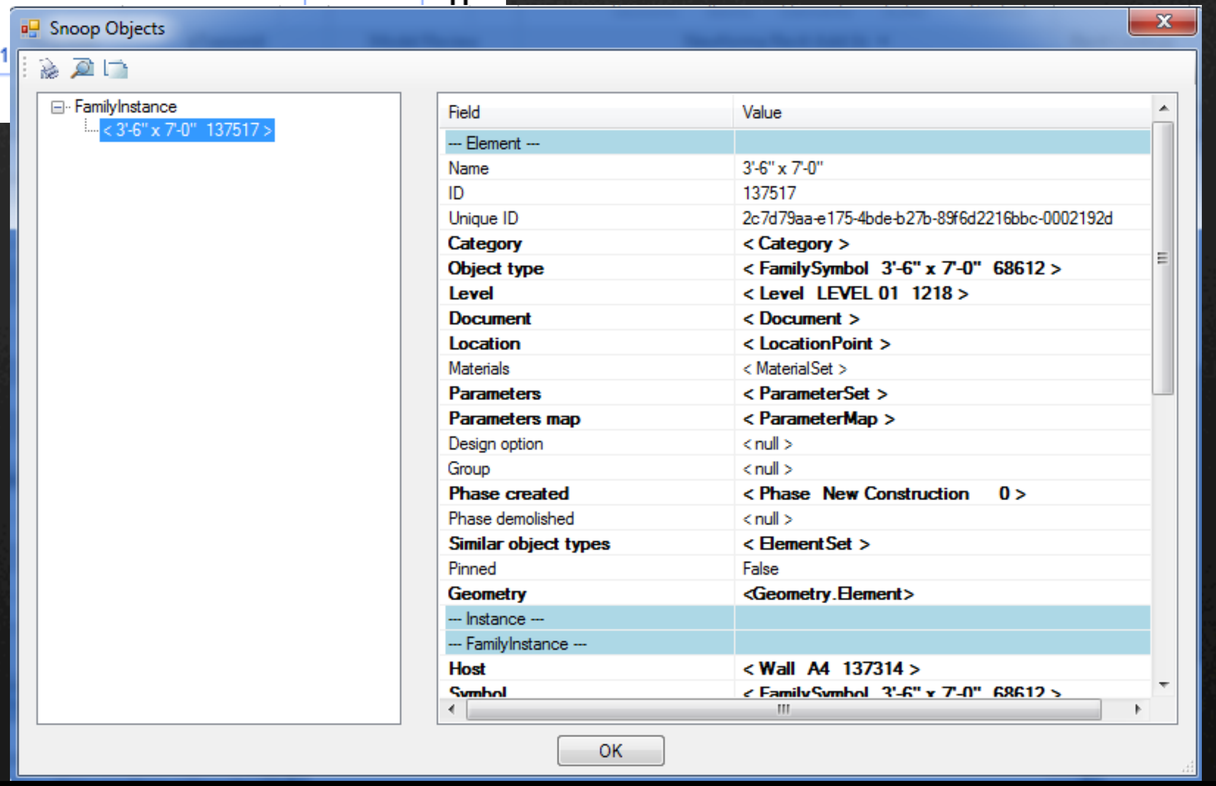


* For Autocomplete to work, make sure Ironpython search path was added to the Configuration file

Revit Lookup Tool





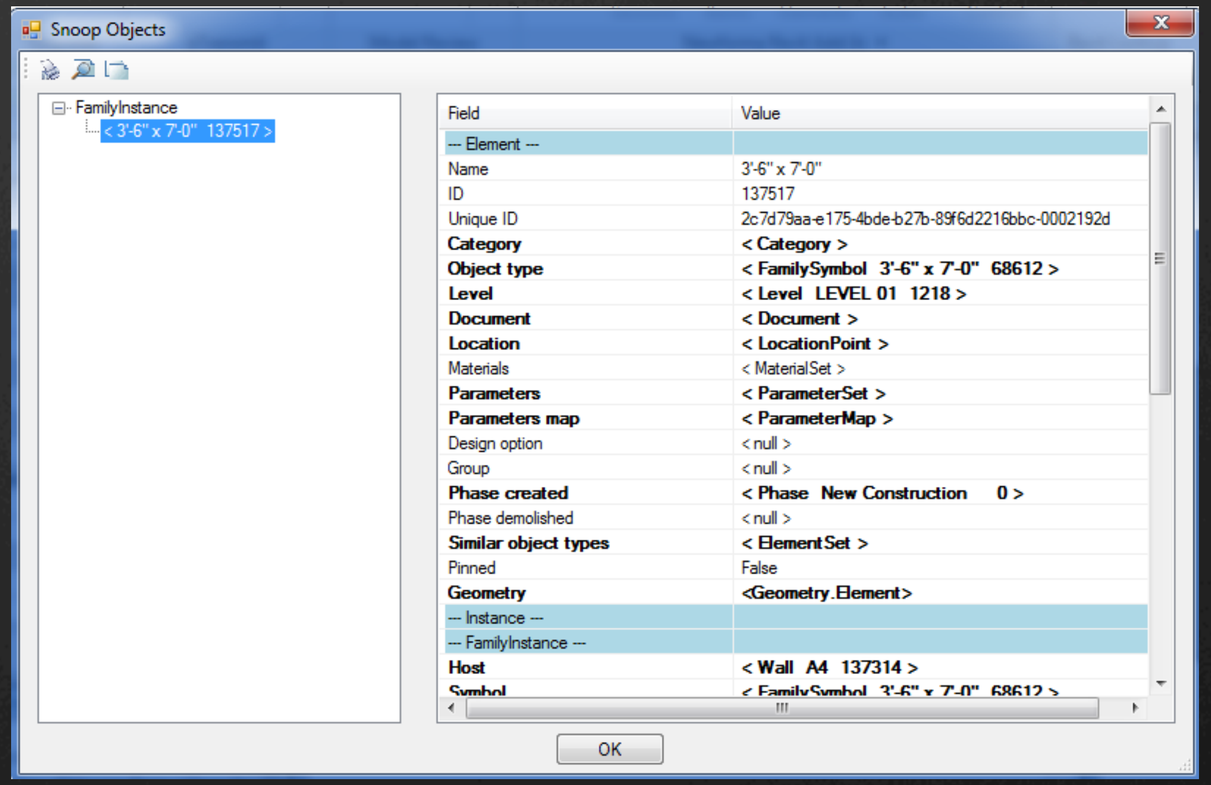


- RevitLookup.dll
- RevitLookup.addin

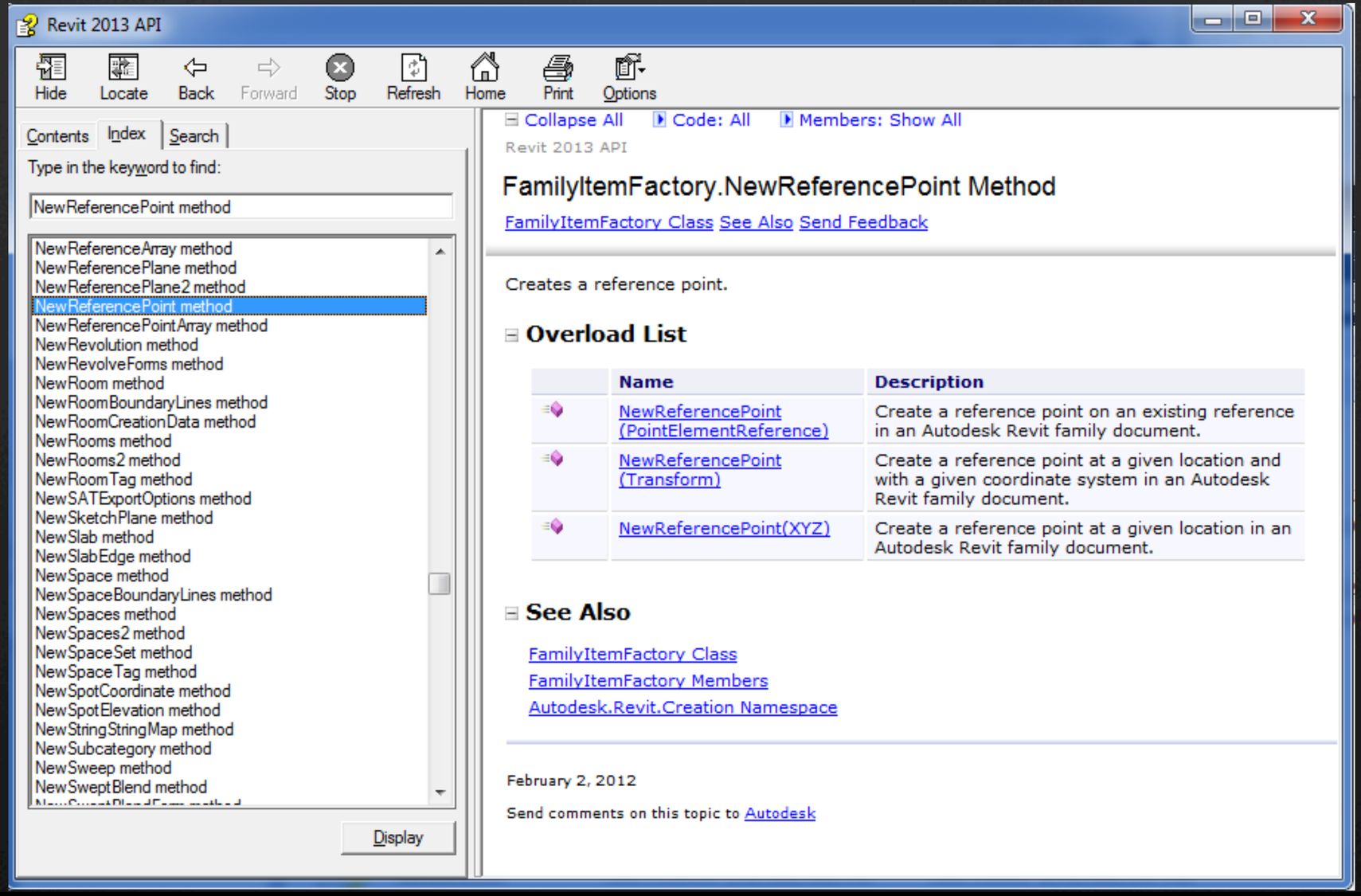
Door

Using Revit LookUp Tool to find detailed element information of an object.

- Builtin Category: OST_Doors
- Family Instance: 3’-6”x7’-0”
- Family Symbol: PW_Flush-Single
- ElementID: 137517



RevitAPI.CHM



Python Programming Syntax

- Case sensitive
  - Ex: *count* is not the same as *COUNT*
- No more if …end if, or curly brackets { …}
- Use indentation for program blocking
  - Use blank spaces or tabs for indentation
  - >>> primary prompt
  - … secondary prompt
- Use # for same line comment
- Use triple quotes (‘’’ or “””) for multiline comments
- Use CamelCase for classes
- Use lower_case_with_underscores for functions and methods.

```
>>> count = 0
>>> count=count +2
>>> print count
2
>>> print COUNT
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'COUNT' is not defined
```

```
>>> x=5
>>> if x>3:
...     y=x+8
...     print y
...
13
```

Source http://www.wordle.net/

Python Reserved Keywords

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

Python Programming Basics - Operators

Arithmetic Operators	**+, -, *, /, %(mod), **(exponent)**
Comparison Operators	== (equal), !=(not eq), <>, > , >=
Logical (or Relational) Operators	and, or, not
Assignment Operators	=, +=(increment), -+(decrement)
Membership Operators	in, not in
Identity Operators	is, is not

Python Programming Basics – Variable Types

- Dynamic typing
  - “If it quacks like a duck, it must be a duck”
- Numeric Types: Integers, float, long, complex
- To convert between types, use type name as function
  - int(x) → converts x to an integer
  - float(y) → converts y to a float
  - str(z) → converts z to a string
  - list(a) → converts a to a list
- To declare variable type
  - myList = [ ]
  - myRA = ReferenceArray()

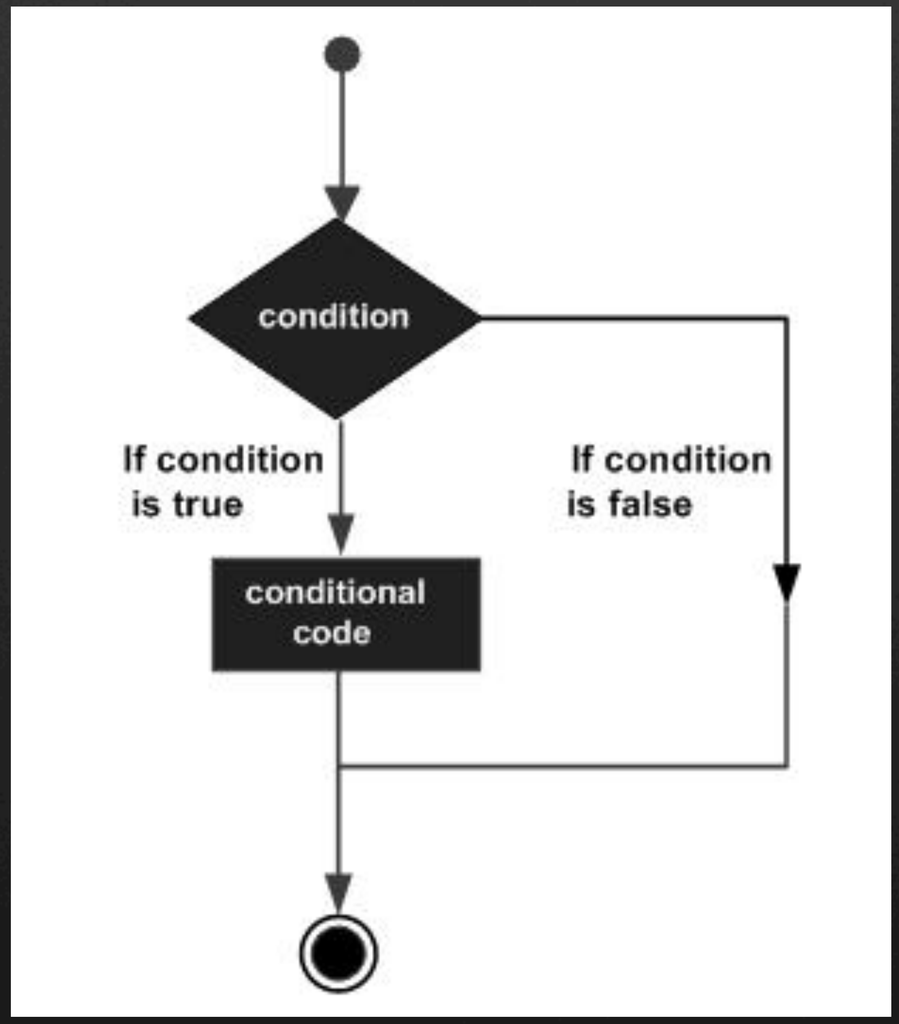
Python Programming Basics – Flow Control

- Conditionals / decision making
  - IF statement

- Loops
  - While Loop
  - For Loop
  - Nested Loop

Python Programming Basics – Conditional

- IF Statement



Source: http://www.tutorialspoint.com/python/

IF Statement

- if header line ends with a colon (:)
- The statement block must be indented.
- elif means “else if”,
- elif is followed by a colon(:)
- else: execute the final statements

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```

Python Programming Basics – Conditional

- IF Statement

```
x =100

if x> 5:
    print str(x) + ": Do Something, X is Greater than 5"
else:
    print str(x) + ": Do Something Else, X is smaller than 5"
```

```
>>>
100: Do Something, X is Greater than 5
>>>
3: Do Something Else, X is smaller than 5
>>>
```

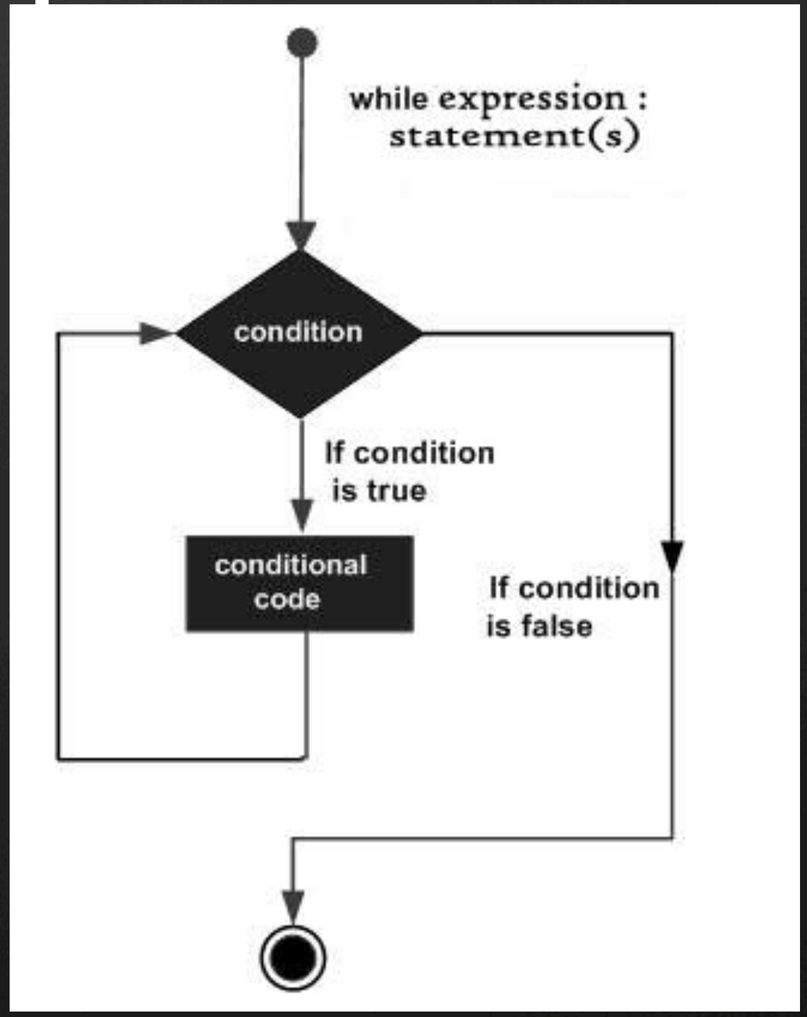
Python Programming Basics – Loops

A loop is the repeating of code lines.
Different types of loops are:

- For Loop
- While Loop
- Nested Loop

Control statements are used to control the loops:

- Break – exit loop
- Continue – jump to top of loop
- Else – statement to do when condition is false
- Pass – statement to do nothing



Source: http://www.tutorialspoint.com/python/

Python Programming Basics – Loops

- For Loop

```
for x in range(10):
    print x
```

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

```
for num in range(10,20):  #to iterate between 10 to 20
   for i in range(2,num): #to iterate on the factors of the number
      if num%i == 0:      #to determine the first factor
         j=num/i #to calculate the second factor
         print '%d equals %d * %d' % (num,i,j)
         break #to move to the next number, the #first FOR
   else:         # else part of the loop
      print num, 'is a prime number'
```

```
>>>
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
>>>
```

Python Programming Basics – Loops

- While Loop

```
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

```
>>>
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
>>>
```

Python Programming Basics – Loops

- Nested Loop

```
i = 2
while(i < 100):
    j = 2
    while(j <= (i/j)):
        if not(i%j): break
        j = j + 1
    if (j > i/j) : print i, " is prime"
    i = i + 1

print "Good bye!"
```

```
>>>
2  is prime
3  is prime
5  is prime
7  is prime
11  is prime
13  is prime
17  is prime
19  is prime
23  is prime
29  is prime
31  is prime
37  is prime
41  is prime
43  is prime
47  is prime
53  is prime
59  is prime
61  is prime
67  is prime
71  is prime
73  is prime
79  is prime
83  is prime
89  is prime
97  is prime
Good bye!
>>>
```

Python Programming Basics – Data Structures

- Lists [ ]
- Tuple ( )
- Dictionary { key : value}
- Sets([ ])

Data Structures – Lists [ ]

- List of items separated by comma between square brackets.
  - A = [1, 2, 3, 4, 5]
  - B = [‘Adam’, ‘Bob’, ‘Cindy’]
  - C = [1, 2, 3, ‘A’, ‘B’, ‘C’]
- Items in a list do not need to have the same type
- List are indexed starting with 0.
- List items can be updated, deleted
- List items can be added or appended
- You can count a list, show min and max value
- You can reverse a list, sort a list
- You can join two list using extend

Data Structures – Lists [ ]

```
# Let's talk about LIST[ ]

# List L has three items
L=['Adam', 'Bob', 'Cindy']
print L

# You can add more items to the list
L.Add('Doug')
print L

# List can have different data types(string, int, list)
L.Add(2012)
L.Add([1,2,3,4])
print L

# show item value using index
print L[3]

# Update item value using index
L[3]="Obama"
print L

# Find index of a value
print L.IndexOf('Obama')
print L.IndexOf('Romney')

# use Contains to determine membership
print L.Contains('Obama')
print L.Contains('Romney')
```

```
# use count or Length(len) to determine the number of items in a list
print L.Count
print len(L)

# use insert to add item in the middle of a list
L.insert(3, 'Zebra')
print L

# use remove to delete item by value
L.remove('Zebra')
print L

# use delete to remove an item by index
del L[5]
print L
```

Data Structures – Tuples( )

- Tuples are like lists except they are immutable.
- Tuple items cannot be updated or deleted.
- Tuple items are not indexed.
- You can use “in” to verify item membership in a tuple
- Tuple can be nested.
- Empty tuple ➔ t= ( )
- Tuple with 1 item ➔ t = (1,)
- Tuple with 3 items using parenthesis➔ t = (“A”, “B”, 666)
- Tuple without using parenthesis ( ) ➔ t = “A”, “B”, 666

Data Structure – Dictionaries { }

- A Dictionary is an unordered set of *key: value* pairs, with the requirement that the keys are unique.
- tel= {‘john': 1234, ‘Mary’: 5678}
- tel.keys( ) ➔ returns the keys in a list [key1, key2, key3]
- tel.values( ) ➔returns the values in a list [val1,val2, val3]
- tel.items( ) ➔ returns the list with pairs of key values.
- [(key1,value1), (key2,value2), (key3,value3)]
- tel.has_key(‘john’) ➔ returns True, if john is in the dictionary
- tel[‘john’]=9999 ➔ value can be updated
- del tel[‘john] ➔ removes entry john
- del tel ➔ removes the whole dictionary tel
- tel.clear() removes all items in dictionary tel

Data Structure – Sets ([ ])

- A set is an unordered collection with no duplicate elements.
  - designers = Set(['John', 'Jane', 'Jack', 'Janice'])
  - managers = Set(['Jane', 'Jack', 'Susan', 'Zack'])
  - mySet = Set([‘A’,’B’, ‘C’])
- Use union | to join 2 sets, listing unique items
  - mySet = designers | managers
- Use intersection & to find common items
  - mySet = designers & managers
- Use difference - to find the different items between sets
  - mySet = designers - managers

Data Structure – Sets ([ ])

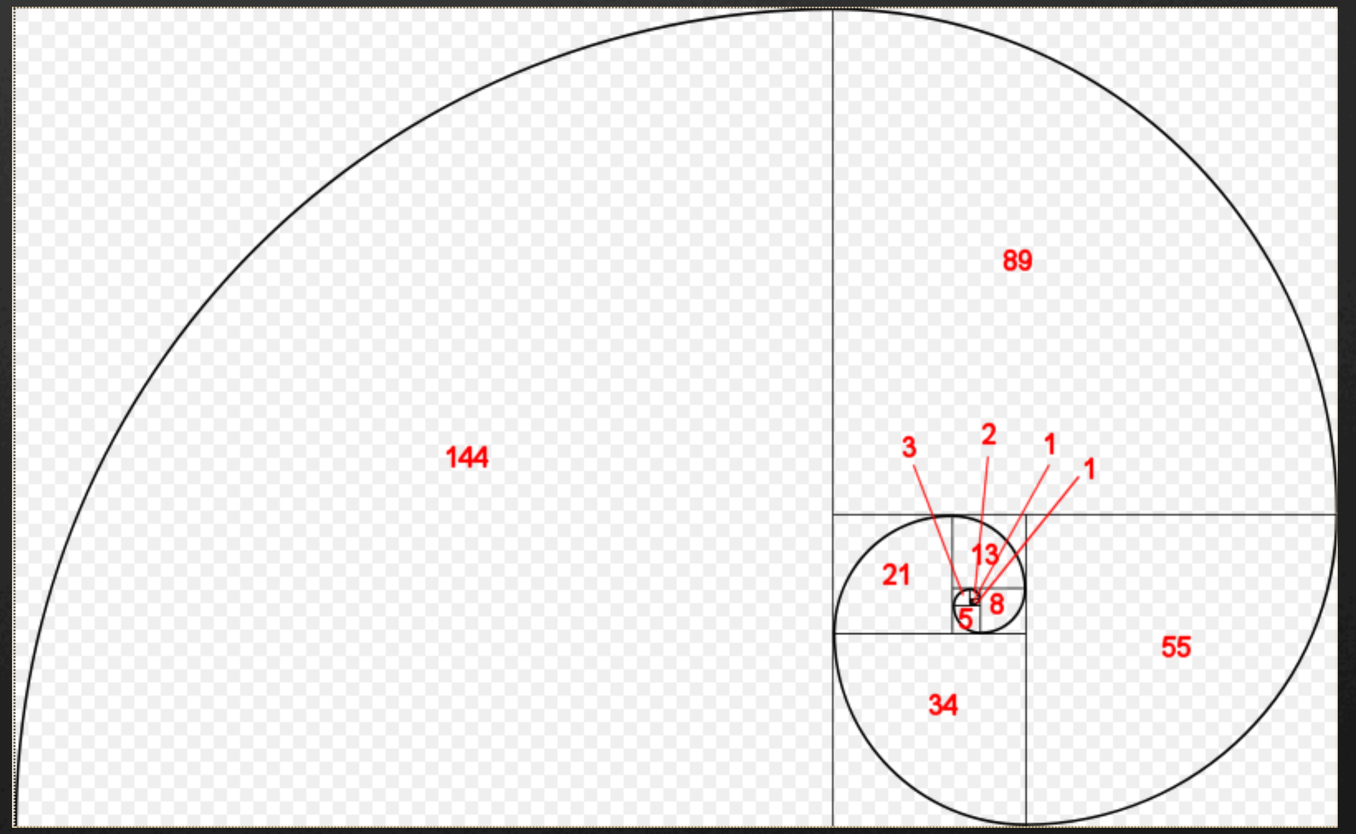
- mySet.add(x) ➔ add item to the set
- mySet.update(x) ➔ update
- mySet.issuperset(x) ➔ check if
- mySet.discard(x) ➔ discard an item from set

Python Functions

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provides better modularity for your application and a high degree of code reusing.

Fibonacci



1, 1, 2, 3, 5, 8, 13, 21, 34 55, 89, 144…..

Python Functions

```
def fib(n):
    a, b = 0, 1
    x=[]
    while b < n:
        print b
        # equivalent to temp = a; a = b; b = temp + b (aka swap pattern)
        a, b = b, a+b
        x.append(b)
    return x
```

Python Class

- **A Class is a** user-defined prototype for an object that defines a set of attributes that characterize any object of the class.
- Class attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

Basic Template for Revit API

Revit API Resource

- Autodesk Developer's Network
- Revit API Developer's Guide (Wiki)
- Revit API.chm
- Revit Lookup Tool
- Revit API Blog - The Building Coder by Jeremy Tammik
- Nathan Miller's Revit API Notebook

Basic Template for Revit API

Reference

Revit Variable Assignments

Transaction

```
#import libraries and reference the RevitAPI and RevitAPIUI
import clr
import math
clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')
from Autodesk.Revit.DB import *

#set the active Revit application and document
app = __revit__.Application
doc = __revit__.ActiveUIDocument.Document

#define a transaction variable and describe the transaction
t = Transaction(doc, 'This is my new transaction')

#start a transaction in the Revit database
t.Start()

#***** ADD YOUR OWN CODES HERE.********

#commit the transaction to the Revit database
t.Commit()

#close the script window
__window__.Close()
```

Reference

```
#import libraries and reference the RevitAPI and RevitAPIUI
import clr
import math
clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')
from Autodesk.Revit.DB import *
```

- Use the CLR(Common Language Runtime) module to reference any .NET libraries that you need
  - clr.AddReference
  - clr.AddReferenceByName
  - clr.AddReferenceByPartialName
  - clr.AddReferenceToFile
  - clr.AddReferenceToFileAndPath
- Use Import to provides access to the namespaces in .NET assemblies

Source: http://blogs.msdn.com/b/haibo_luo/archive/2007/09/25/5130072.aspx

Revit Variable Assignment

```
#set the active Revit application and document
app = __revit__.Application
doc = __revit__.ActiveUIDocument.Document
```

- __revit__ similar to CommandData in Revit API
- Assign variables to the Revit objects.
- See *RevitAPI.CHM* for all Revit API namespaces.

Transaction

```
#define a transaction variable and describe the transaction
t = Transaction(doc, 'This is my new transaction')

#start a transaction in the Revit database
t.Start()

#***** ADD YOUR OWN CODES HERE.********

#commit the transaction to the Revit database
t.Commit()
```

Transaction

- Transactions are context-like objects that encapsulate any changes to a Revit model.
- Any change to a document can only be made while there is an active transaction open for that document.
- Attempting to change the document outside of a transaction will throw an exception.
- Changes do not become a part of the model until the active transaction is committed.
- All changes made in a transaction can be rolled back either explicitly or implicitly (by the destructor).
- Only one transaction per document can be open at any given time. A transaction may consist of one or more operations.

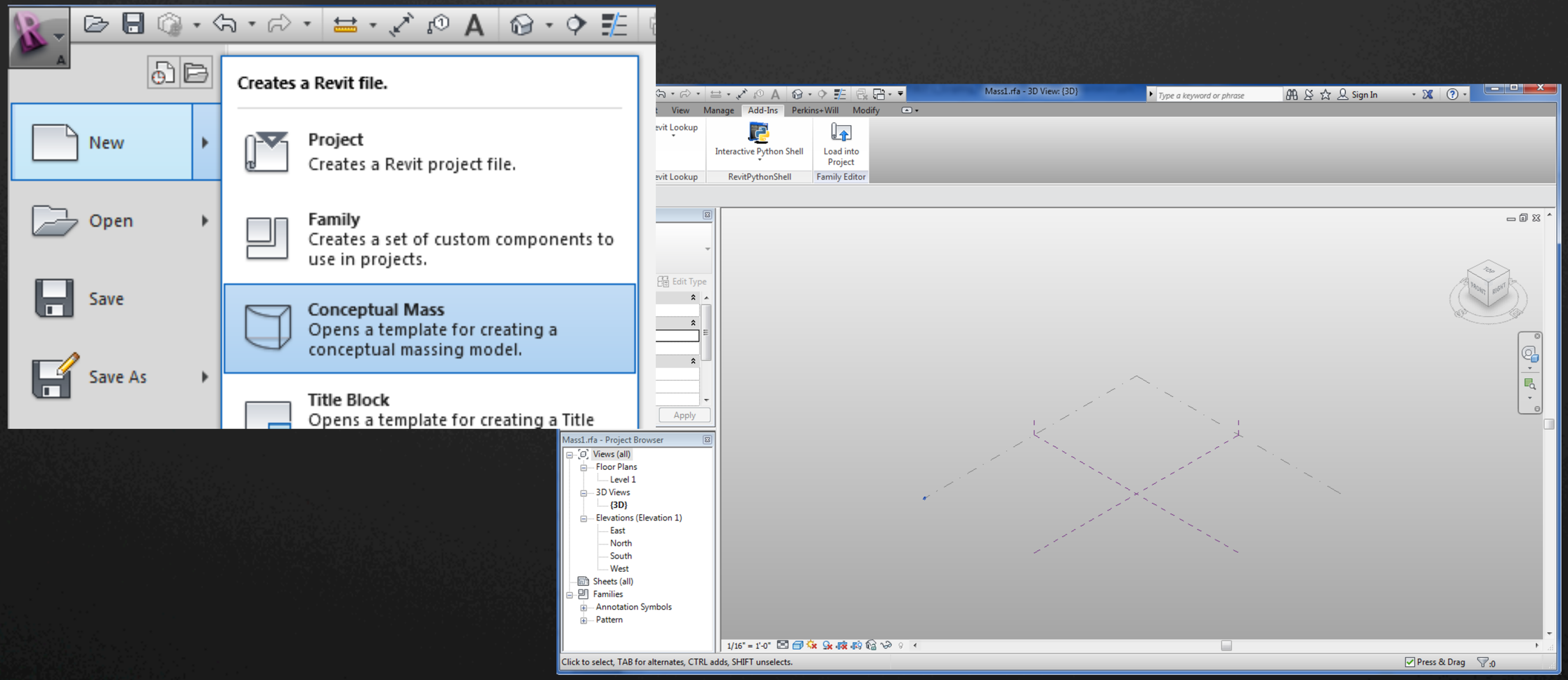
Source: Revit API Developer's Guide

Understand Revit API elements

Revit “Mode”

- Family Editor – RFA
- Project - RVT

Family Editor – Conceptual Mass



Geometric vs. Model Objects

- Geometric objects
- Non visible objects
- Defined to create other objects

- Model Objects
- Visible objects in the Revit Document
- Created using geometric object definitions

Geometric vs. Model Objects

- Geometric objects ➔app.Create
  - XYZ Point (XYZ) ➔XYZ (x,y,z)
  - Line ➔app.Create.NewLine (xyz, xyz, bool)
  - Plane ➔app.Create.NewPlane(xVec, yVec, Origin)

- Model Objects ➔doc.FamilyCreate
  - Reference Point ➔doc.FamilyCreate.NewReferencePoint(xyz)
  - Curve ➔doc.FamilyCreate.NewModelCurve(line, SketchPlane)
  - Sketch Plane ➔doc.FamilyCreate.NewSketchPlane(plane)
  - Reference Plane ➔doc.Create.NewReferencePlane(plane)

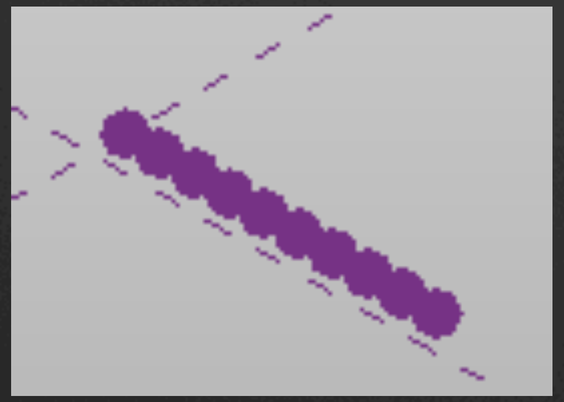
Reference Point

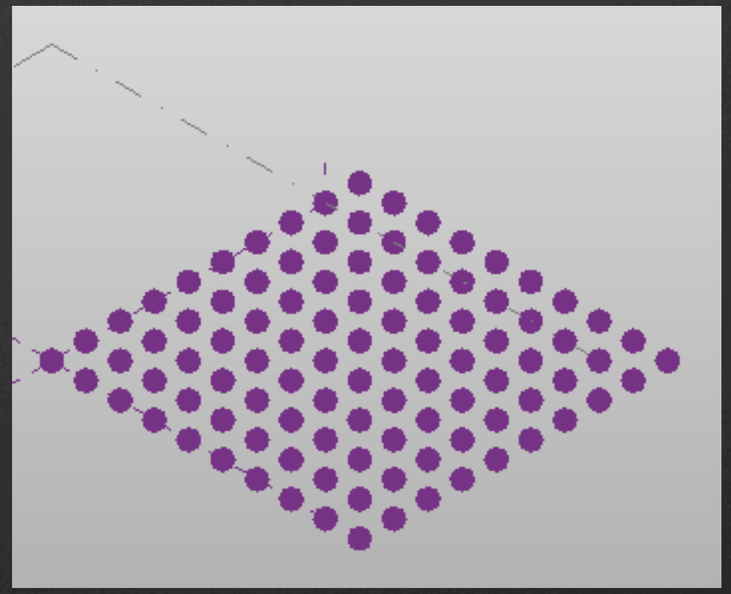
```
#p1 = XYZ(x,y,z)
p1 = XYZ(10,10,0)

#use XYZ to define a reference point
rp = doc.FamilyCreate.NewReferencePoint(p1)
```

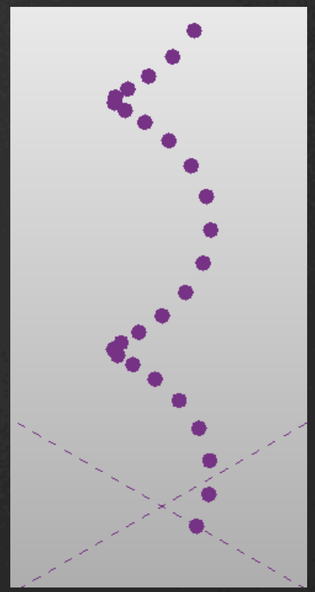
Points Equations

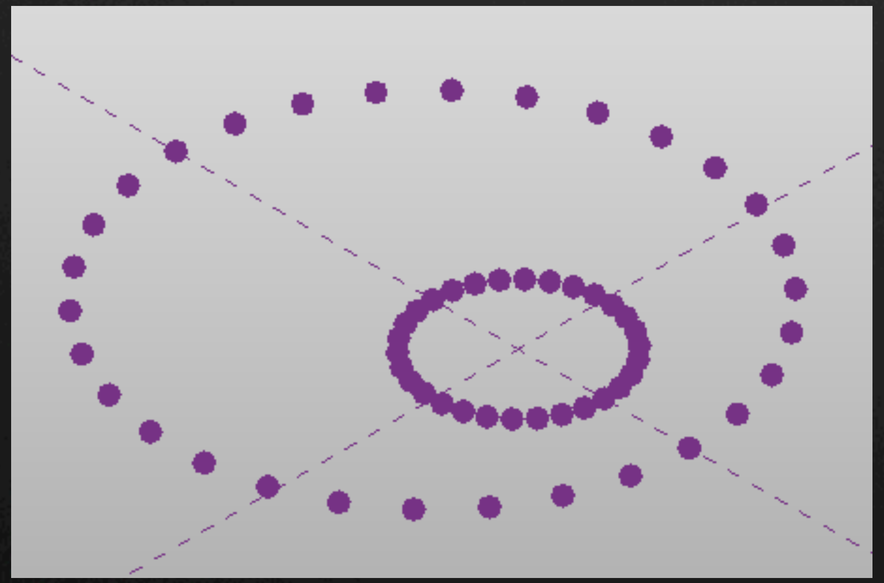
- Points in a row
- Points in a grid
- Points in a Helix
- Points in a Parabola
- Points in a Circle
- Points in an Ellipse

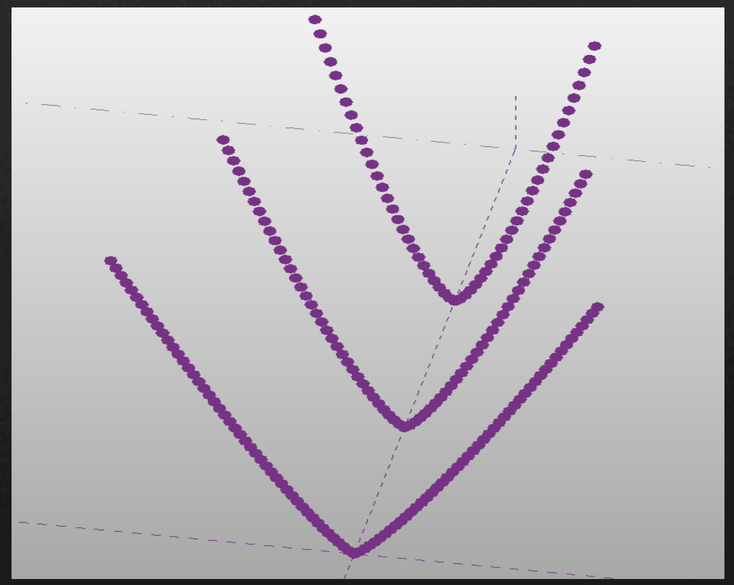
Points Equations

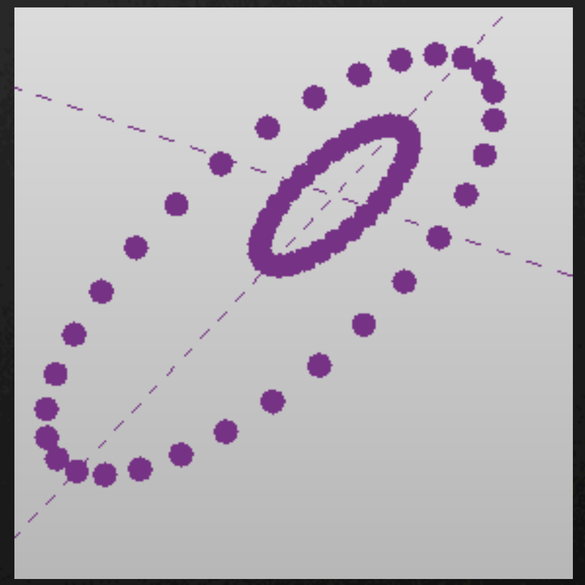












New Points Method

- NewPointOnEdge(curve.GeometryCurve.Reference, 0.5)
- NewPointOnEdge (edgeReference, LocationOnCurve)
- NewPointOnEdgeEdgeIntersection (edgeReference1, edgeReference2)
- NewPointOnEdgeFaceIntersection (edgeReference, faceReference, OrientWithFace)
- NewPointOnFace (faceReference, uv)
- NewPointOnPlane (doc, planeReference, position XYZ, xVector XYZ)
- NewPointRelativeToPoint(hostPointReference)

New Curve By Points

```
    #use XYZ to define a reference point
    refPoint = doc.FamilyCreate.NewReferencePoint(myXYZ)
    rpArray.Append(refPoint)

MyCrv = doc.FamilyCreate.NewCurveByPoints(rpArray)
```

- NewReferencePoint(XYZ)
- ReferencePointArray(ReferencePoint)
- NewCurveByPoints(ReferencePointArray)

New Loft Form

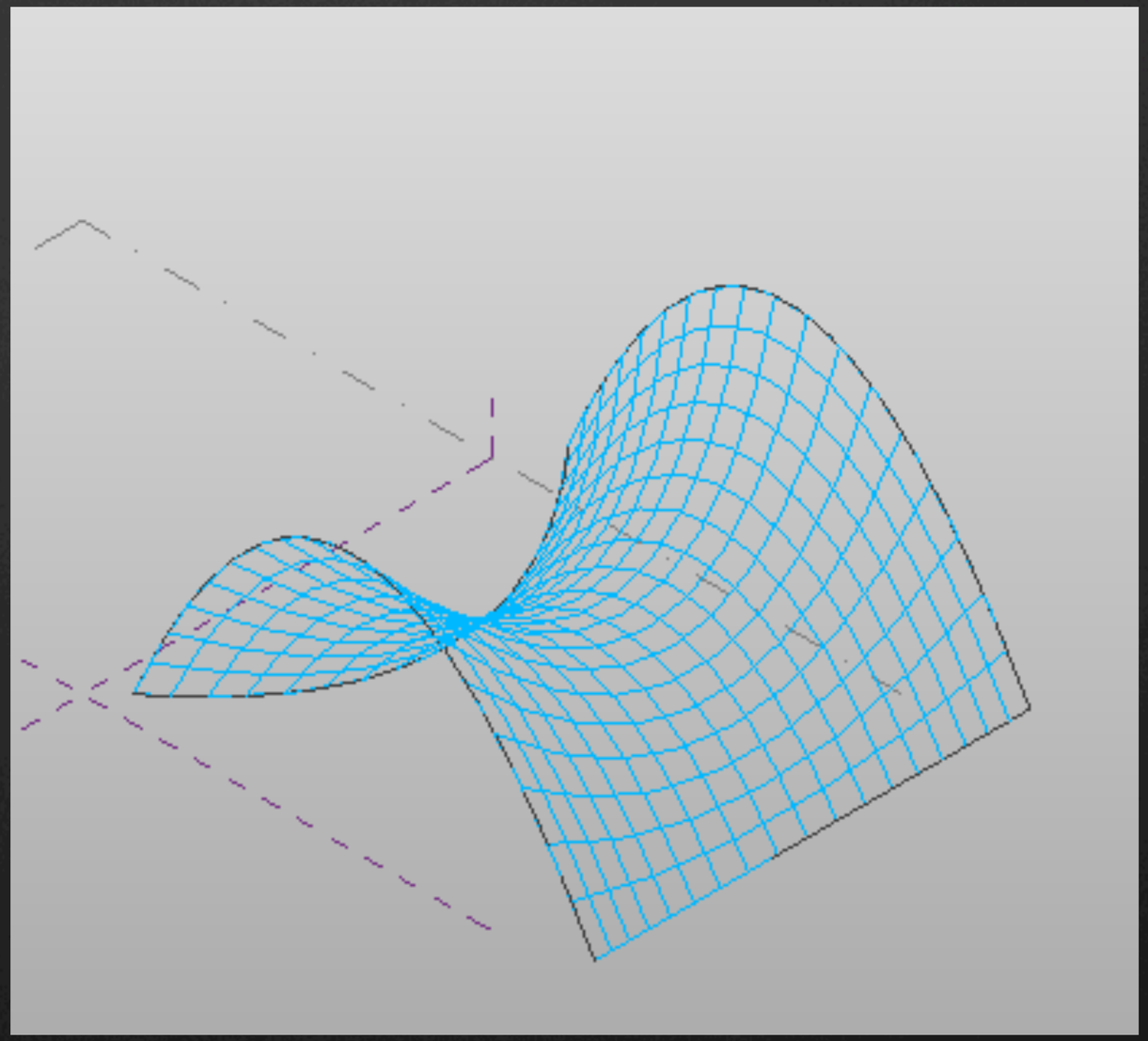
CurveReference = Curve.Geometry.Reference
ReferenceArray(CurveReference)
ReferenceArrayArray(ReferenceArray)
NewLoftForm(isSolid, ReferenceArrayArray)

```
# Create a Reference Array(RA) for each curve
# and append these RA to the profile
# Where Profile is an array of RA
# Profile = ReferenceArrayArray()
Profile = ReferenceArrayArray()

RA = ReferenceArray()
RA.Append(arc1_ref)
Profile.Append(RA)

RA = ReferenceArray()
RA.Append(arc2_ref)
Profile.Append(RA)

RA = ReferenceArray()
RA.Append(arc3_ref)
Profile.Append(RA)

#******* Create Loft ***************
doc.FamilyCreate.NewLoftForm(True, Profile)
```

Massing Forms

- Extrusion
  - NewExtrusionForm(isSolid, Profile_ReferenceArray, Direction_XYZ)
- Revolution
  - NewRevolveForm(isSolid, Profile_ReferenceArray, axis, startAng, endAng)
- Blend
  - NewBlendForm(isSolid, Profile_ReferenceArray, direction)
- Sweep
  - NewSweptBlendForm(isSolid, path, Profile_ReferenceArray)
- Other
  - NewFormByThickenSingleSurface(isSolid, form, direction)
  - NewFormByCap(isSolid, profile)

Divided Surface



- divSurf = doc.FamilyCreate.NewDividedSurface(face.Reference)

Filter

```
#Create a FilteredElementCollector
collector = FilteredElementCollector(doc)
collector.OfCategory(BuiltInCategory.OST_MassForm)
```

Iterate

```
famtypeitr = collector.GetElementIdIterator()
famtypeitr.Reset()

for item in famtypeitr:
    typeID = item
    srfObj = doc.get_Element(typeID)
```

Divided Surface – Tile Patterns Built-in

- Rectangle
- Rhomboid
- Hexagon
- HalfStep
- Arrows
- ThirdStep
- ZigZag
- Octagon
- OctagonRotate

- RectangleCheckerboard
- RhomboidCheckerboard
- Triangle_Flat
- Triangle_Bent
- TriangleStep_Bent
- TriangleCheckerboard_Flat
- TriangleCheckerboard_Bent

ChangeTypeId( )

Change the surface pattern by providing a panel ID

dividedSurface.ChangeTypeId(ElementID)

For Built-in pattern:

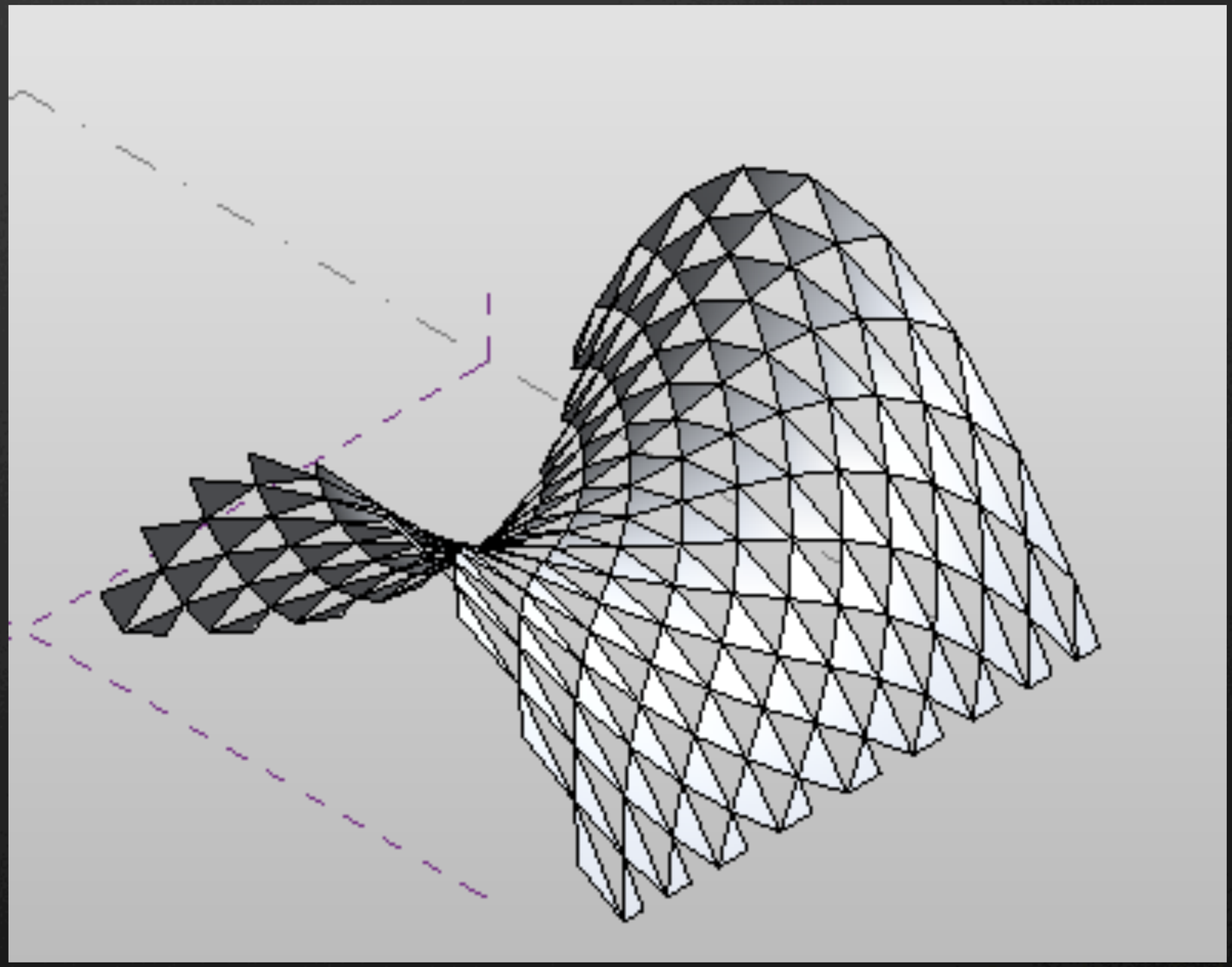
patterns.GetTilePattern(TilePatternsBuiltIn.TriangleCheckerboard_Flat).Id

For Curtain Wall Pattern:

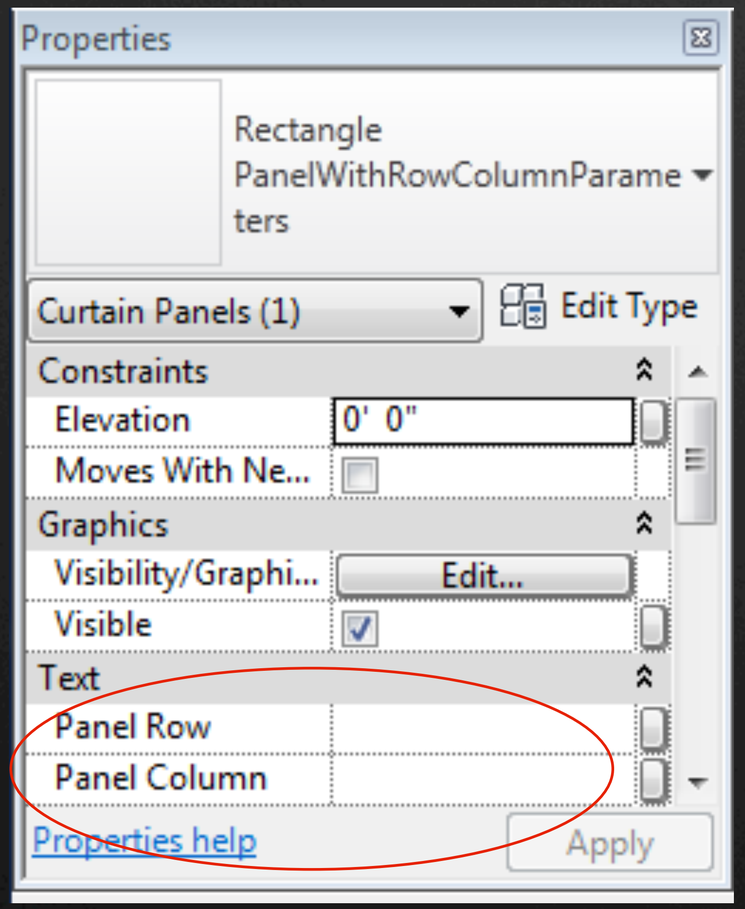
use Curtain Wall Panel’s elementID

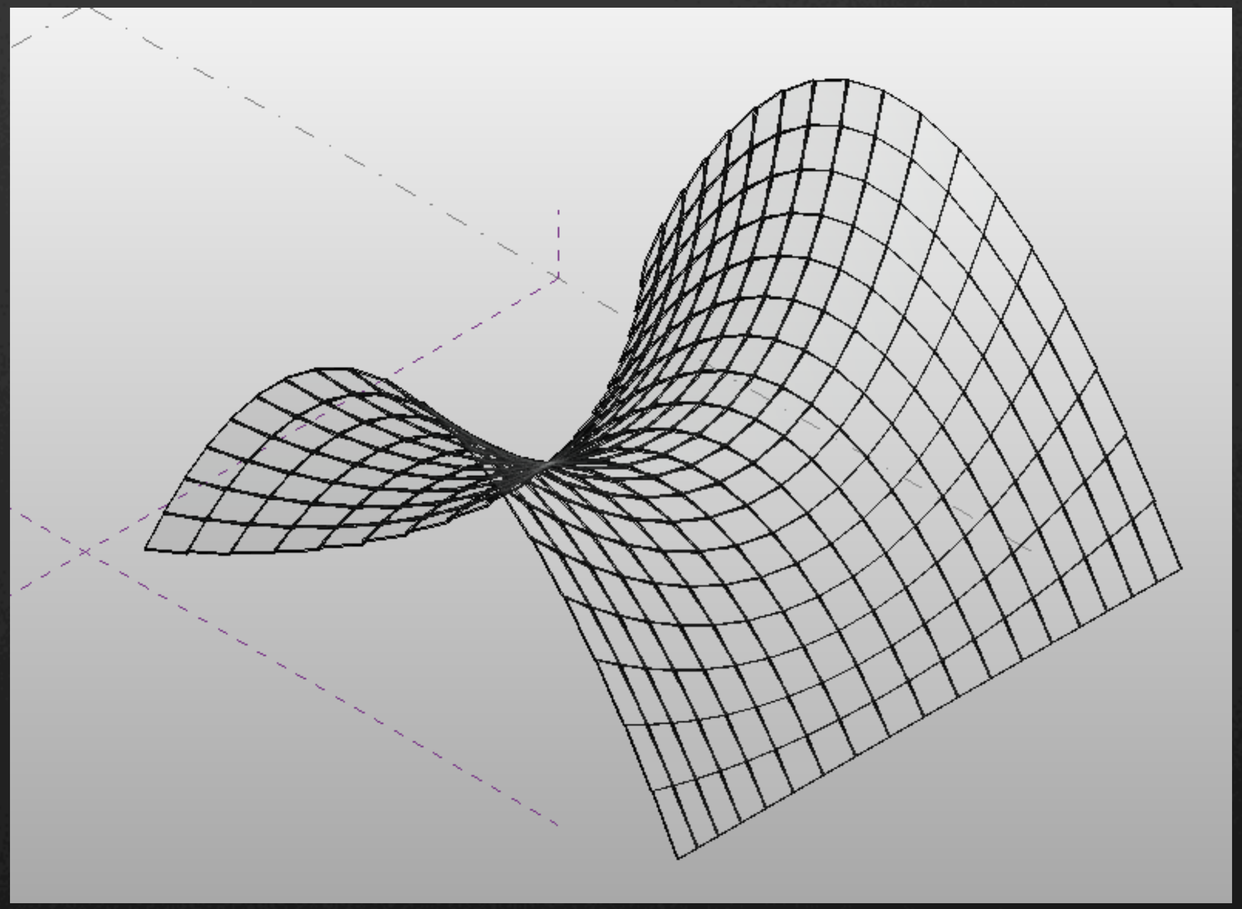
Tile Pattern

- TriangleCheckerboard_Flat



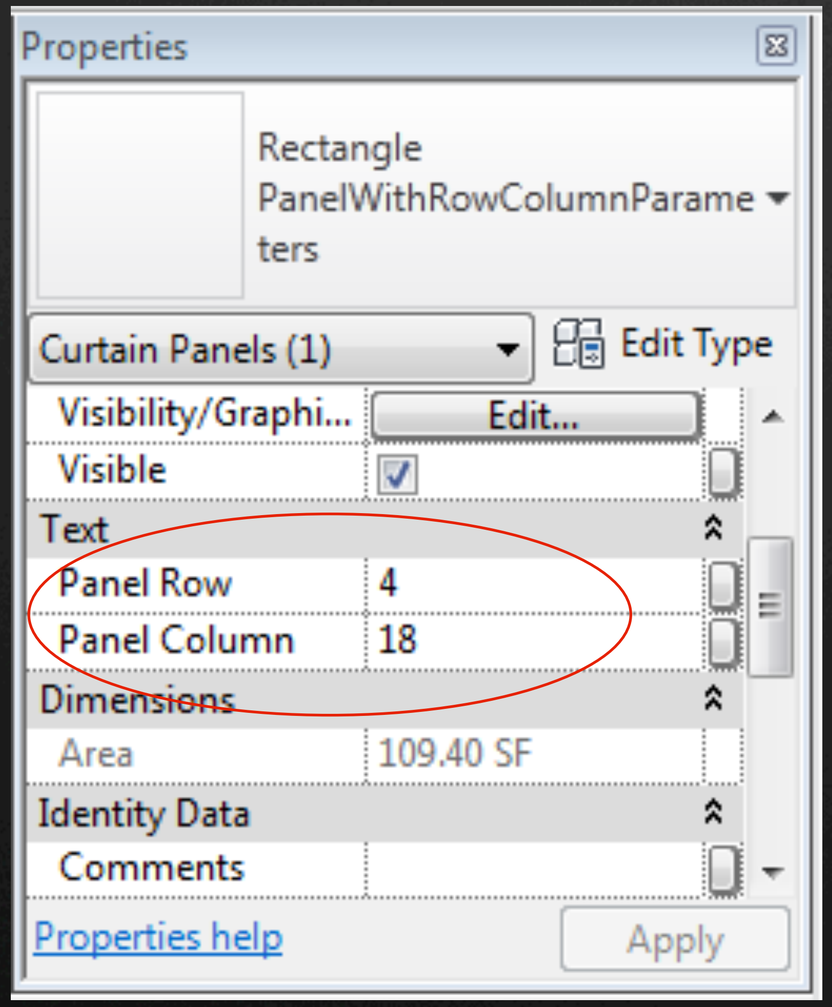
Curtain Wall Panel

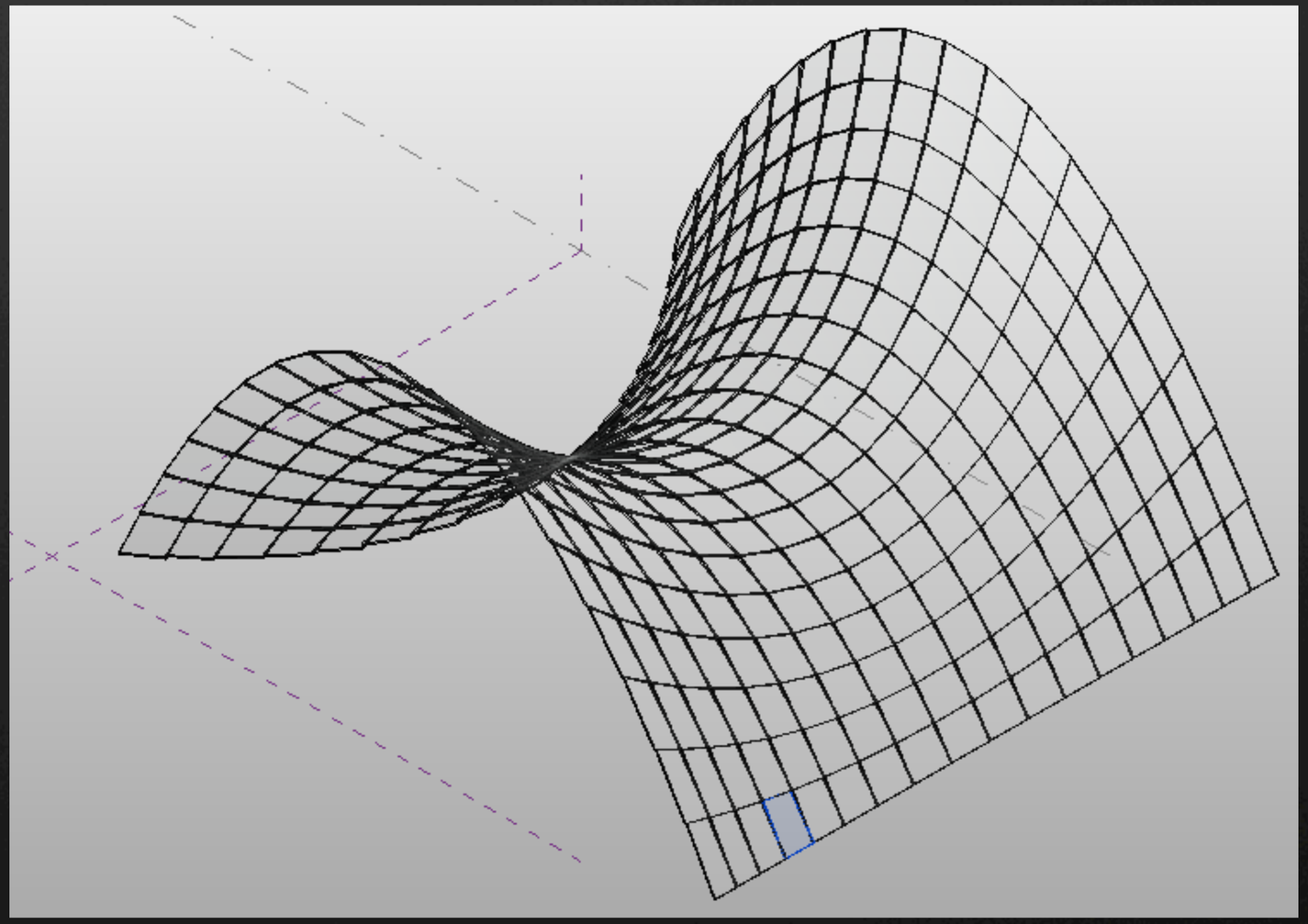




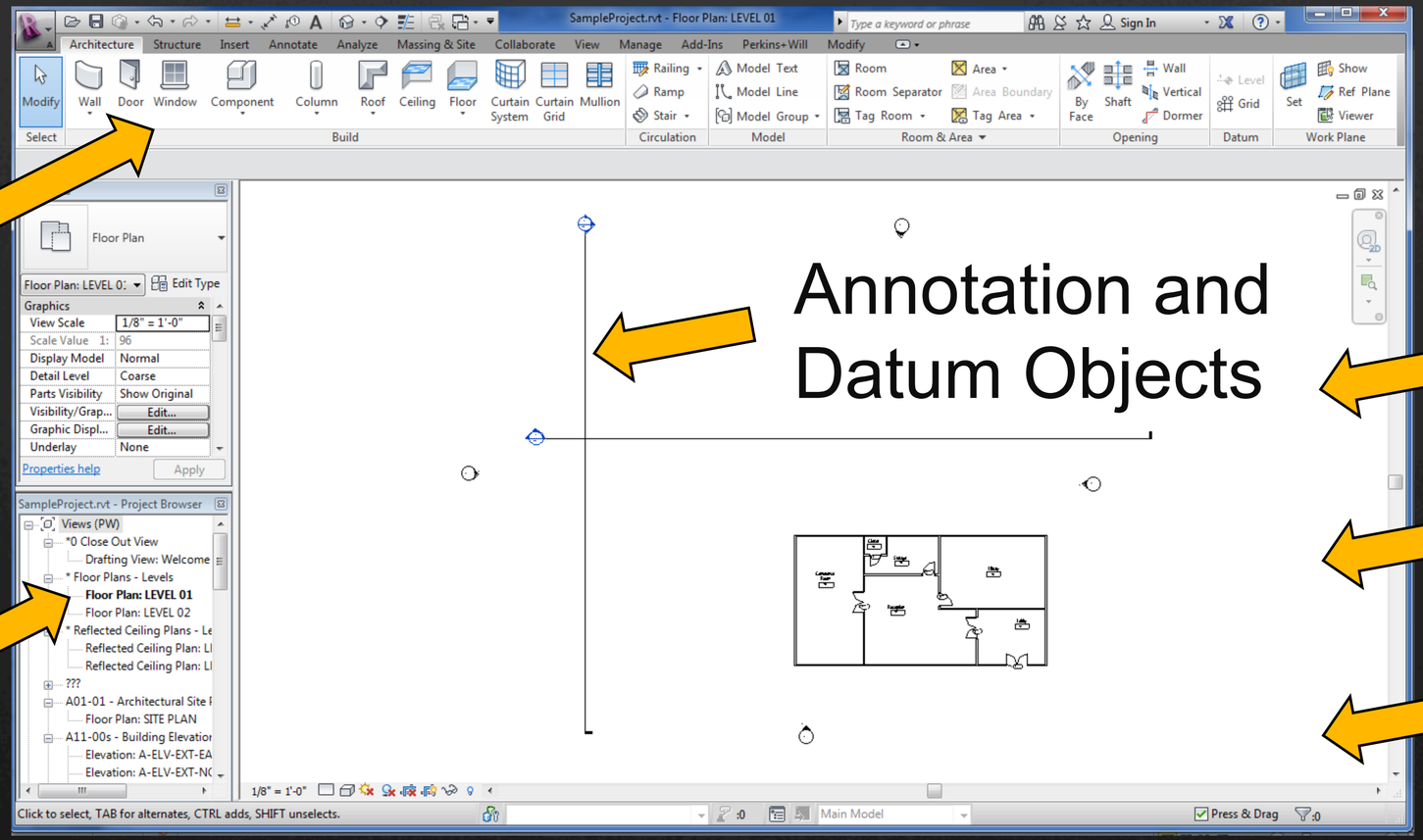
- PanelWithRowColumnsParameter.rfa

Panel Numbering





Revit Project File



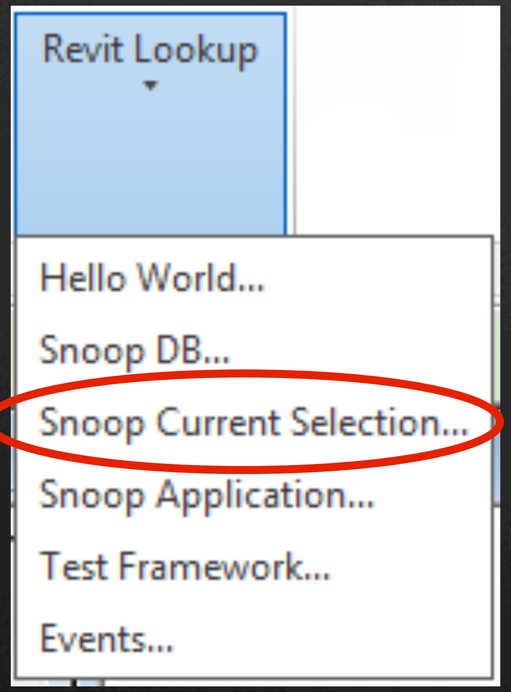
Revit Elements

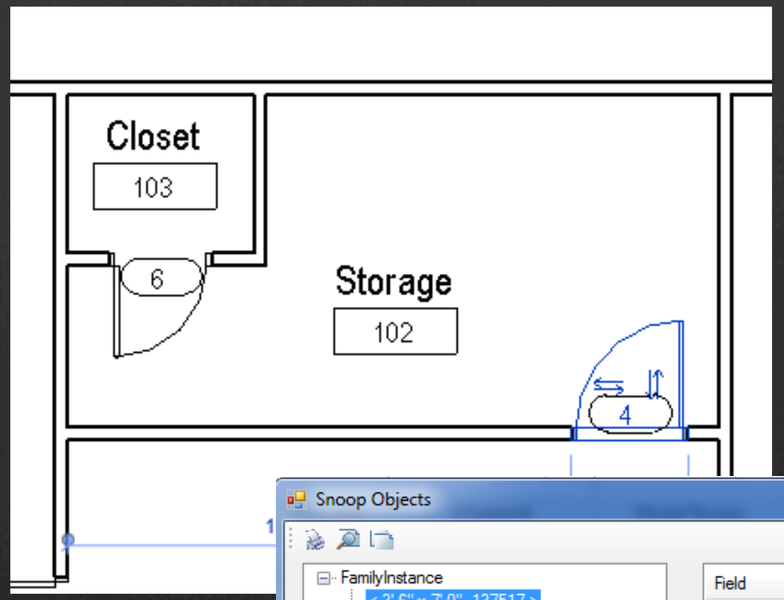
- Model Elements
- View Elements
- Group Elements
- Annotation and Datum Elements
- Sketch Elements
- Information Elements

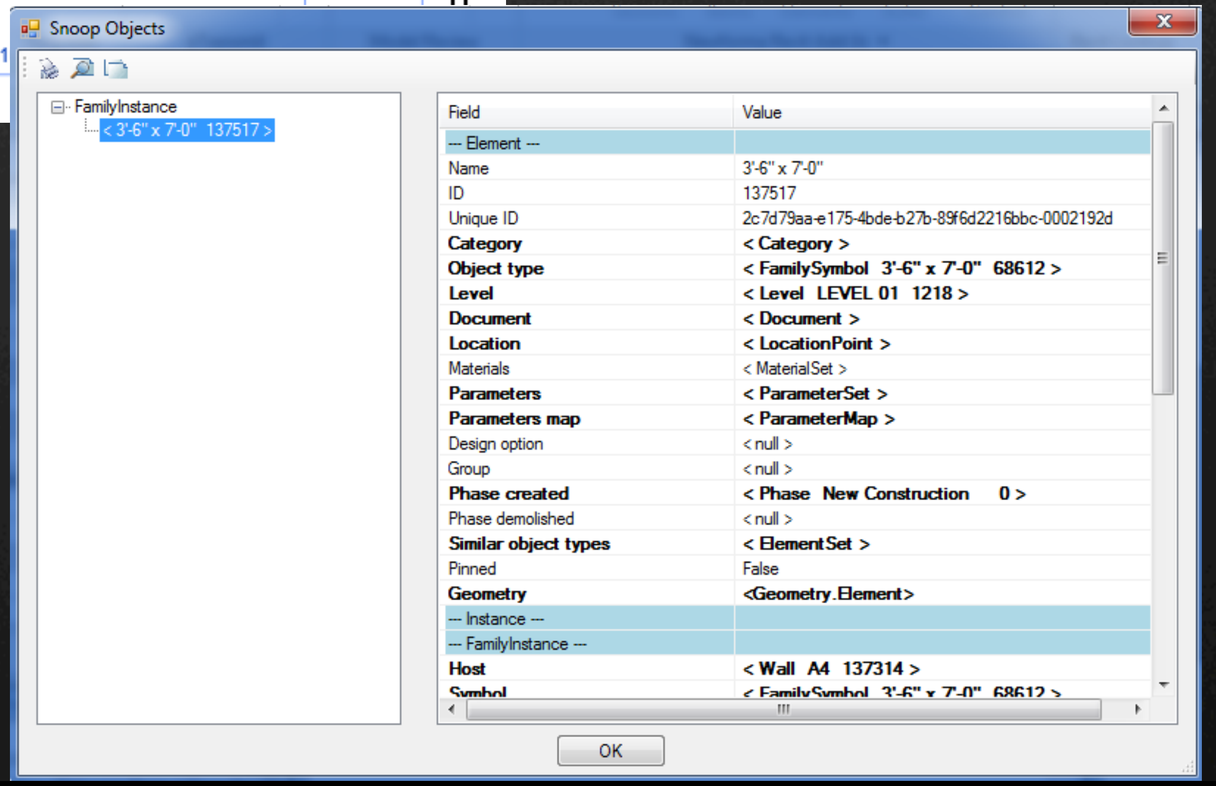
Element Classification

- Category
- Family
- Symbol
- Instance

Revit Lookup Tool





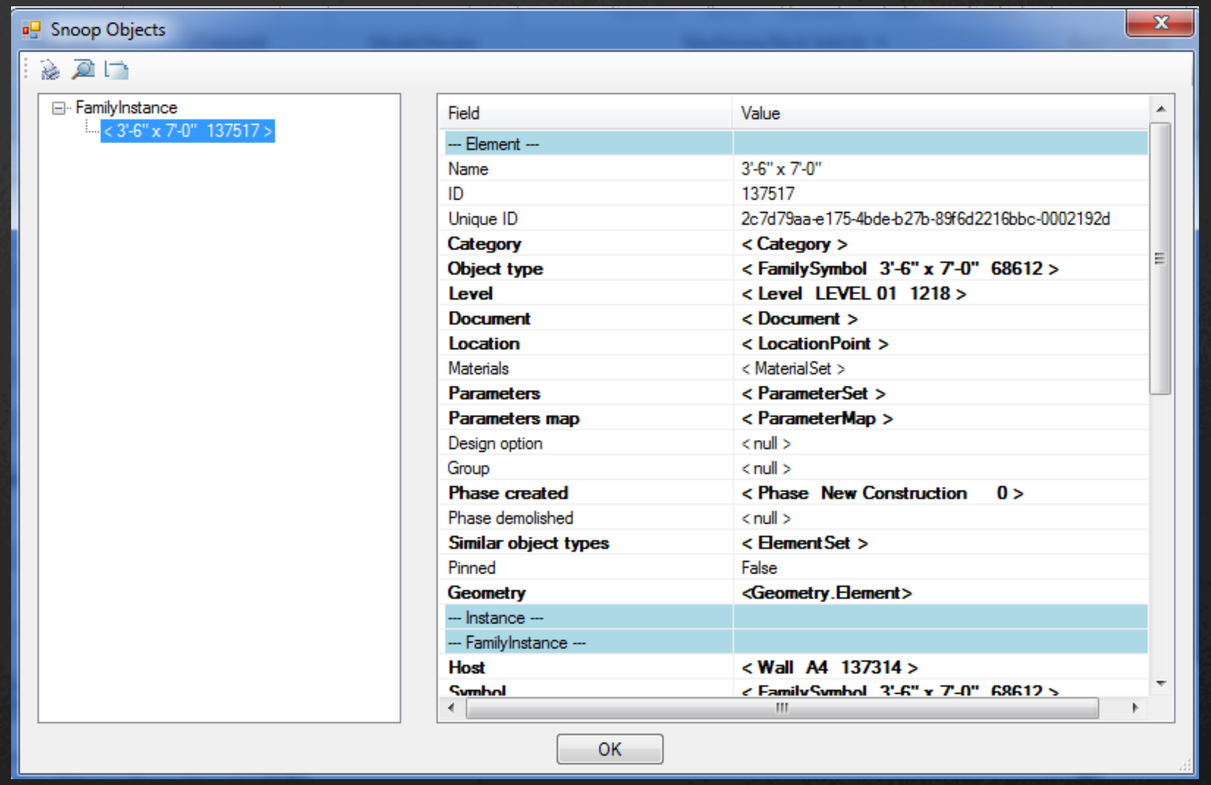


- RevitLookup.dll
- RevitLookup.addin

Door

Using Revit LookUp Tool to find detailed element information of an object.

- Builtin Category: OST_Doors
- Family Instance: 3’-6”x7’-0”
- Family Symbol: PW_Flush-Single
- ElementID: 137517



Element Retrieval

- Using Filter + Iteration
- User Selection
- ElementId
- Project Information

Use Filter to find Elements

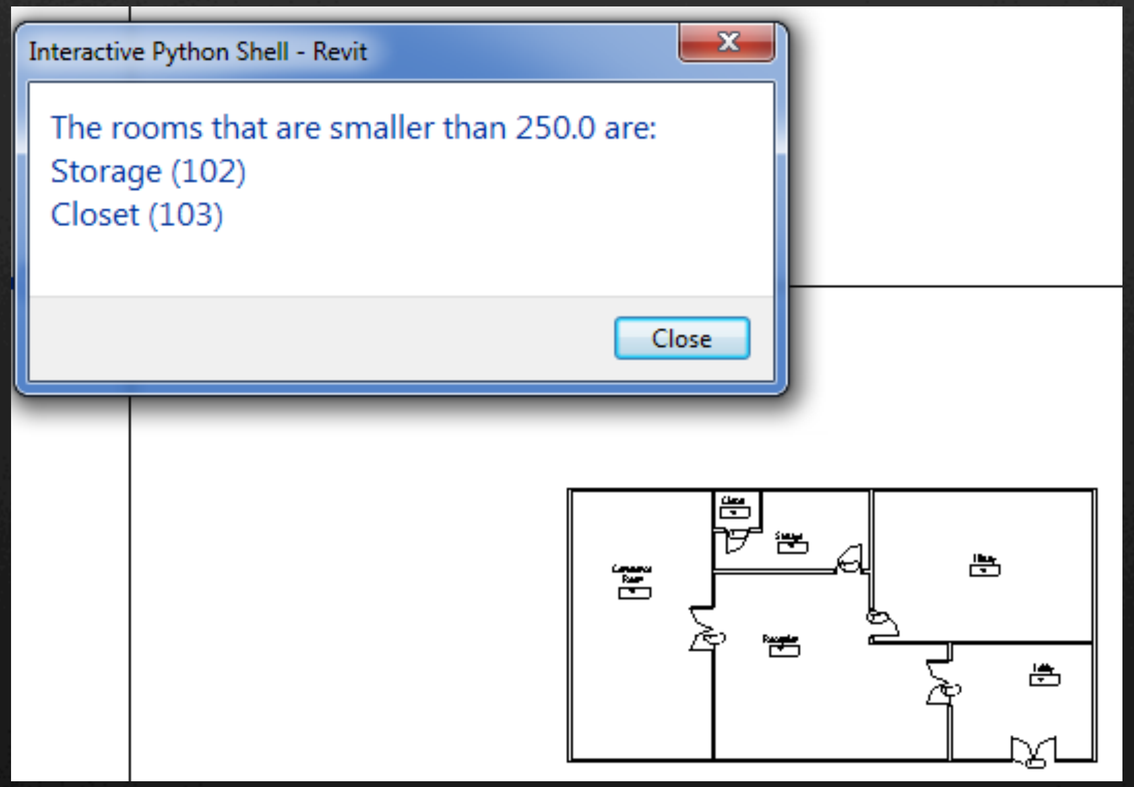
- The basic steps to get elements passing a specified filter are as follows:
  - Create a new **FilteredElementCollector**
  - Apply one or more filters to it
  - Get filtered elements or element ids (using one of several methods)

```
# create a filtered element collector set to Category OST_GenericModel
# and Class FamilySymbol
collector = FilteredElementCollector(doc)
collector.OfCategory(BuiltInCategory.OST_GenericModel)
collector.OfClass(FamilySymbol)
```

Use scripts to create and change geometry in Revit

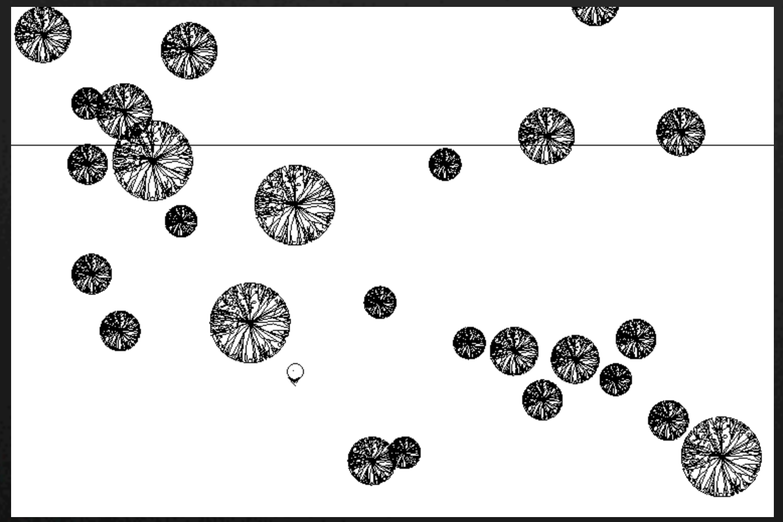
Finding Rooms

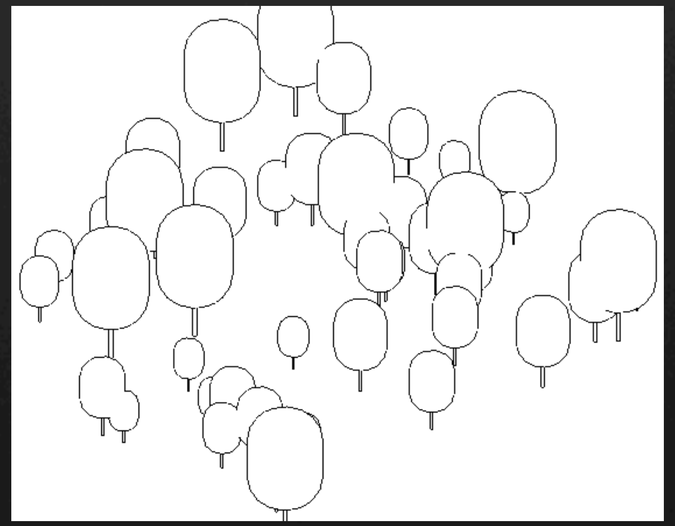
- Goal:
- Specify a target SF size, and find all rooms that are smaller than the target.



Placing Random Trees

- Task:
  - Fill out the landscape with trees randomly placed of various types within a given area.
- Given:
  - Tree family (SI_Tree) that has 5 types (20’, 25’, 30’, 35’, 50’)



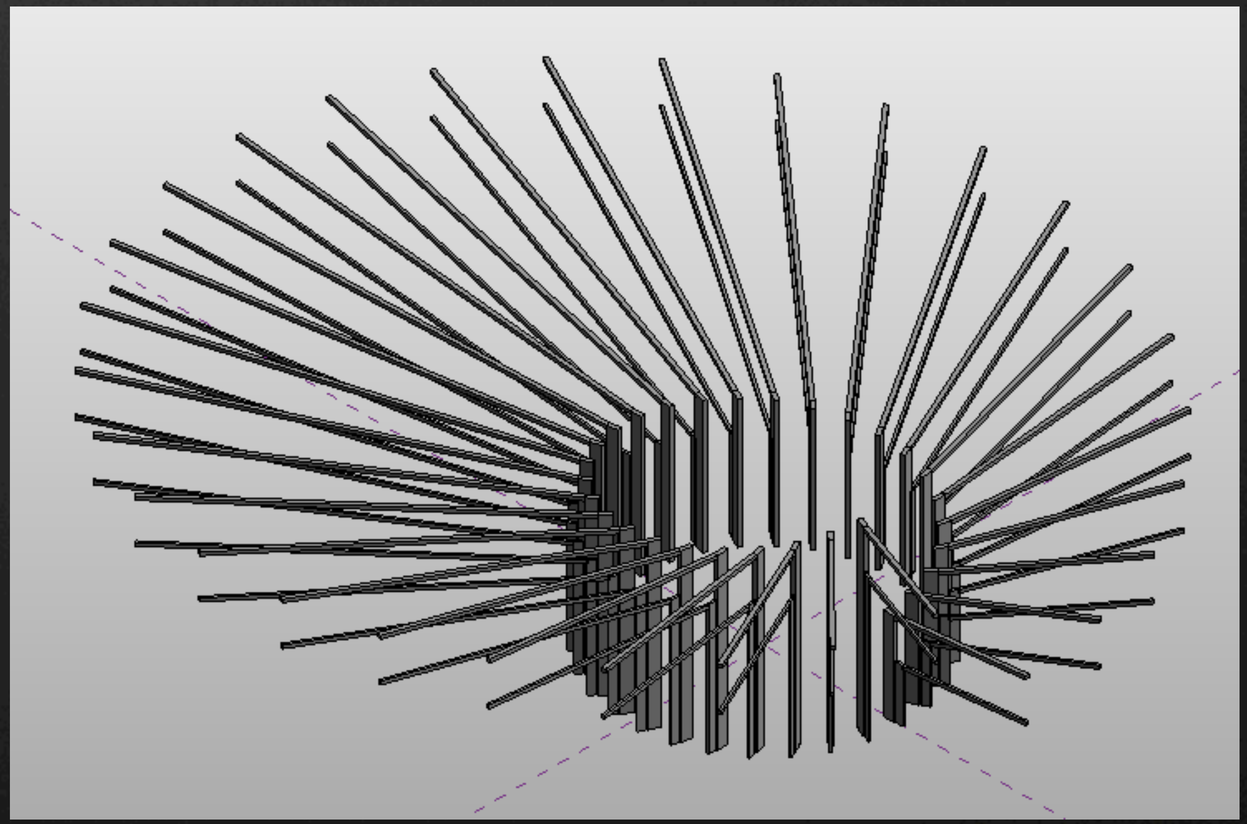


Placing Adaptive Component Family

Goal :

- Place the frames along 2 ellipses
- Adjust each frame height to be taller than the previous one, such that the last frame is twice as tall as the first one.

- *Note: Adaptive Component family (Frame_Short) has 2 placement points*



Questions?

Autodesk University Session Feedback

Your feedback is very important to Autodesk.

- ✓ You can complete the session survey on your mobile device, PC, or at a survey station.
- ✓ Each completed session survey enters you in that day's drawing for a free AU 2013 pass.
- ✓ You can help make AU 2013 better!

Complete the AU Conference Survey at a survey station and receive an AU 2012 T-Shirt.

Thank You

IffatMai@gmail.com
Iffat.mai@perkinswill.com