Autodesk University 2012

Scripting with RevitPythonShell in Project Vasari

Iffat Mai – Perkins+Will

CP3837-L

RevitPythonShell brings scripting ability to Autodesk® Revit software and Project 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
- Learn basic Python programming
- Understand Revit API elements
- Use scripts to create and change geometry in Revit

About the Speaker

Iffat Mai has more than 20 years of experience working in the field of digital and computational design technology in Architecture. Iffat has managed and trained hundreds of users in some of the most prestigious architectural firms in New York City. Her expertise is in developing custom tools and solutions to automate and streamline design processes and improve project productivity and efficiency. Before rejoining Perkins and Will as their firm wide Design Application Development Manager, Iffat held the position of Senior Research and Development Manager at SOM. Iffat was a speaker on Revit API Customization at Autodesk University for the last few years. Iffat holds a Bachelor of Science degree in Architecture from Massachusetts Institute of Technology.

iffatmai@gmail.com

iffat.mai@perkinswill.com

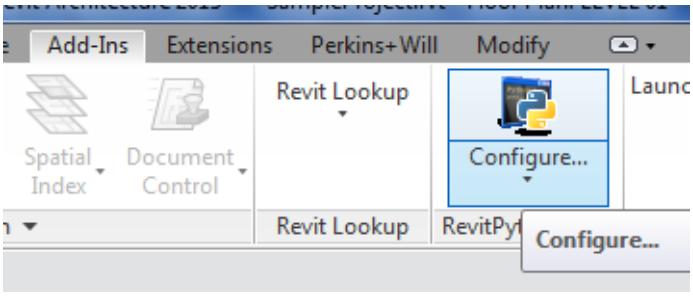
Software Introduction

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

Our primary environment will be in Revit Architecture 2013. The Vasari Beta version functions similarly to Revit's Massing Family Editor, we will be using Revit Architecture 2013 in both Project and Family Editor mode. For many years, the Revit Application Programming Interface (API) has provided a compiled method for programmers to communicate with Revit using .NET languages. However, the process is quite complicated and does not allow direct interaction with Revit. Thanks to Daren Thomas, who created the RevitPythonShell which exposes Revit API to python scripting. Now Revit users can create simple scripts in python to manipulate their Revit database.

Become familiar using RevitPythonShell in Revit

RevitPythonShell Configuration



The RevitPythonShell can be accessed from within Revit Architecture's Add-Ins tab. It contains two options, Configure and Interactive Python Shell.

First, you must configure the Revit Python Shell and add the library path of the IronPython as a search path:

- 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"/>

Once the configuration is saved, you must exit Revit, and restart Revit for the changes to take effect.

Interactive Python Shell

The Interactive Python Shell contains two sections. The top section is the command prompt area. The bottom section is the python editor. In the command prompt area, you will see >>> as the primary prompt and … as the secondary prompt when you write multiline codes. Codes entered in the command prompt area will be executed immediately once you hit the enter key. In the lower editor section, you can type new codes or load an existing codes from any text file.

Run the codes by hitting F5 or the play button  . Please note that in the editor, you can only SAVE to a file, there is no SAVE AS button. When you load an existing file and make changes to it, you can only save and overwrite the original code, you don't have the option to SAVE AS a different file. If you want to start a new code using an existing code as base, copy and rename the file in Window's explorer first before loading it into the Python Editor.

Learn basic Python programming

What is Python and IronPython?

Python is an open-source dynamic programming language that is fast and simple and runs on many different platforms. IronPython is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework. IronPython makes all .NET libraries easily available to Python programmers, while maintaining compatibility with existing Python code.

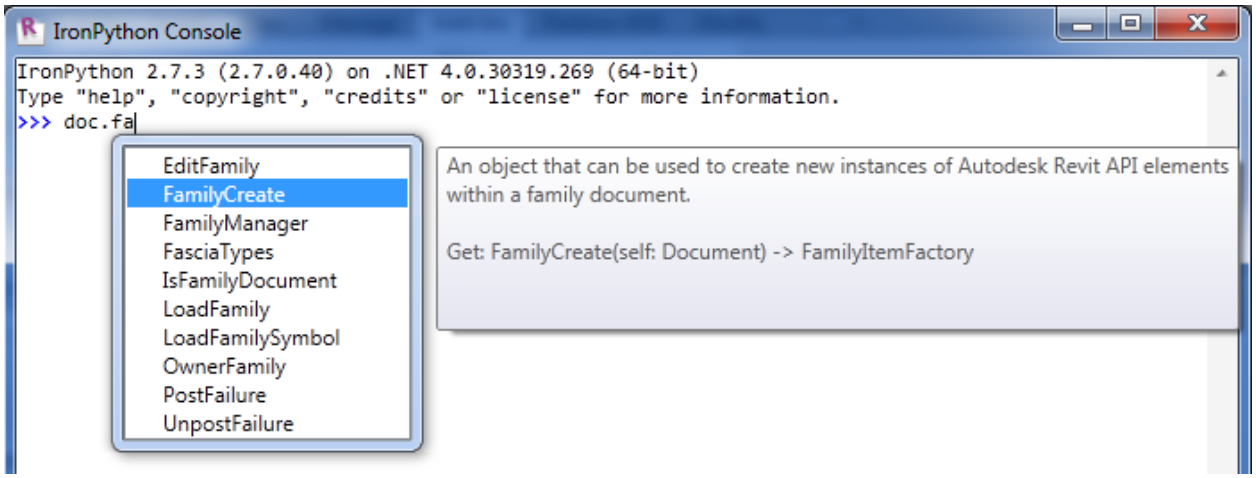
Where to find help with Python?

In the command prompt area, type in **help(X)** to get the built-in help system to print the help page of that object X in the Console. Type in keyword **dir(X)** with the object X enclosed in the parenthesis, and you will get a listing of all the names (variables, modules, functions) that this

object defines. Autocomplete only works in the command prompt area, and you can invoke it by typing the dot(.) after an object and hold down the **CTRL** and **Spacebar** keys together.

- help(object)
- dir(object) or dir() for currently defined objects
- Activate Autocomplete* : after the dot(.) press **CTRL + Spacebar**

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



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 single line comment
- Use triple quotes (‘’’ or “””) for multiline comments
- Use CamelCase for classes
- Use lower_case_with_underscores for functions and methods.

*See files/Scripts/1_PythonBasic.py for examples.

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 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 Variable Types

Python offers dynamic variable typing. Unlike other structured programming languages, in python, you don't need to explicitly declare variable types. Python figures out what variable type it should be by the type of objects that you assign it to. This is often referred to as "duck typing", that "If it quacks like a duck, it must be a duck". The principal types that are built-in are numerics, sequences, mappings, files, classes, instances and exceptions.

- Numerics : Integer, float, long and complex types.
- Sequences: str, list, tuple, xrange, bytearray, buffer, unicode
- Boolean: True, False

To convert between types, use type name as function

- int(x) ➔ converts x to an integer
- float(x) ➔ converts x to a float
- str(x) ➔ converts x to a string
- list(x) ➔ converts x to a list

To declare variable type

- myList = [ ]
- myRA = ReferenceArray()

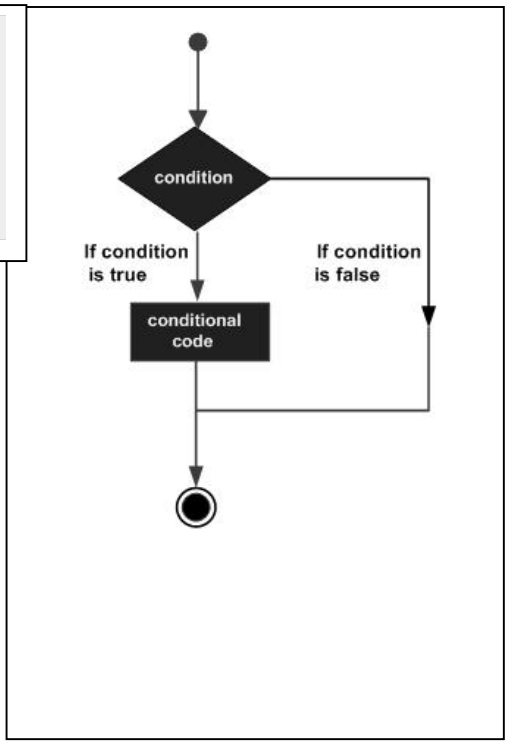
Python Flow Control

Conditionals / decision making - IF statement

For conditional flow control, use the 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)
```




```
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"
```

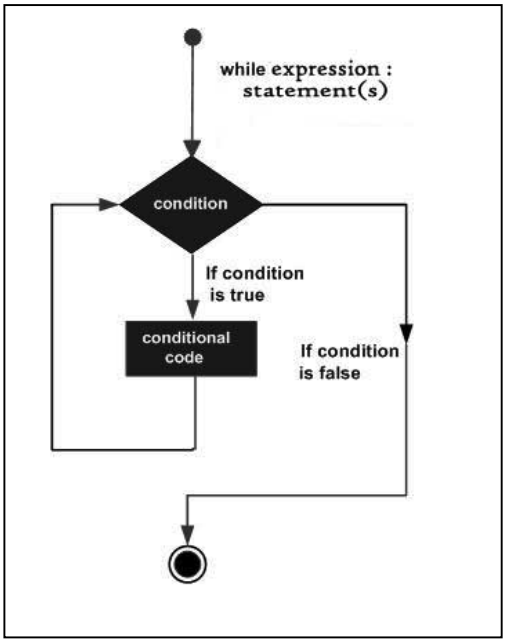
```
>>>
100: Do Something, X is Greater than 5
>>>
3: Do Something Else, X is smaller than 5
>>>
```

Loops

A loop is the repeating of code lines.

Different types of loops are:

- For Loop
- While Loop
- Nested Loop



For Loop

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

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

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!
>>>
```

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!
>>>
```

Control Statements

Use control statements together with the loop to complete the loop.

- Break statement – to exit the loop (Loop_ForBreak_Prime.py)
- Continue statement – jump to the top of the loop
- Else statement – execute when condition is false.
- Pass statement – command to do nothing.

Python Data Structures

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

Lists

- A sequence 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
- Lists 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

```
# 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
```

Tuple

- 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

Dictionary

- 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

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
- 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]
```

```
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.

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary
```

Revit API

Open and run HelloWorld_Revit.py

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

app = __revit__.Application
doc = __revit__.ActiveUIDocument.Document

t = Transaction(doc, 'What the script does')

t.Start()

TaskDialog.Show("Revit", "Hello World")


t.Commit()

__window__.Close()
```

Parts of Revit API Template

There are three main parts in the basic Revit API template. The reference section, the Revit objects variable assignments and Transactions.

Reference Section

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

The Reference section should include imports of different modules needed. Use the *CLR* (Common Language Runtime) module to reference any .NET libraries that you might need.

- clr.AddReference
- clr.AddReferenceByName
- clr.AddReferenceByPartialName
- clr.AddReferenceToFile
- clr.AddReferenceToFileAndPath

To access the RevitAPI assemblies, you need to add references to two DLL libraries, Revit API.dll and RevitAPIUI.dll. Then use import to access the Namespace in these assemblies.

Revit Objects Variable Assignments

- ***__revit__*** :class name similar to CommandData object in Revit API
- Assign ***app*** variable to the Revit Application.
- Assign ***doc*** variables to the Current Active Document object.
- Both variables are already declared in the init-script of the configuration file.
- Refer to RevitAPI.CHM for all the name spaces in the Revit API

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

Transaction

According to the Revit API Developer's Guide:

"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."

```
#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()
```

window

This window refers to the python script window. When running the python code as a shortcut button, you will need to close the window automatically. When using this interactively, __window__ will not be available.

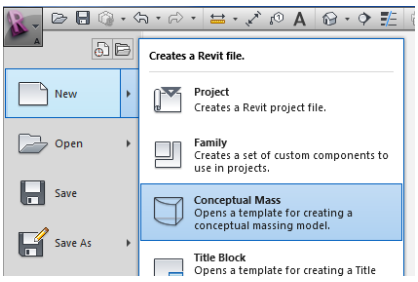
```
#import libraries and reference the RevitAPI and RevitAPIUI
import clr
import math
clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')
from Autodesk.Revit.DB import *

#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()
```

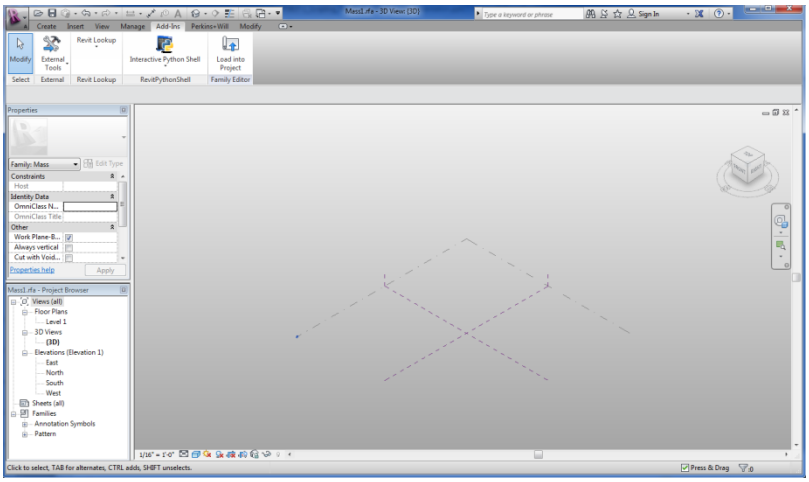
Revit Mode

Revit basically has two modes, one for Revit Family, and the other for Revit Project.

Let's first take a look at Revit Conceptual Massing Family editor.

Open Massing1.rfa (or start a new Conceptual Mass)





Conceptual Massing Family Editor

Geometric vs. Model Objects

Geometric objects	*Model Objects*
Non visible objects	Visible objects in the Revit Document
Defined to create other objects	Created using geometric object definitions

Geometric objects types	*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 types	*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)

An XYZ point

- MyPoint = XYZ(10, 20, 0)

NewLine()

- BoundLine = app.Create.NewLine(startPoint, endPoint, True)
- UnboundLine = app.Create.NewLine(startPoint, endPoint, False)

NewPlane()

- MyPlane = app.Create.NewPlane(xVec, yVec, origin)
- MyPlane = app.Create.NewPlane(normal, origin)
- MyPlane = app.Create.NewPlane(CurveArray)

A Reference point

- MyRefPoint = doc.FamilyCreate.NewReferencePoint(XYZ)

NewCurve()

- MyCurve = doc.FamilyCreate.NewModelCurve(Line, SketchPlane)

NewSketchPlane()

- MySketchPlane = doc.FamilyCreate.NewSketchPlane(Plane)
- NewReferencePlane()
- MyRefPlane = doc.Create.NewReferencePlane(bubbleEnd, freeEnd, cutVec, document.ActiveView)

Curve

- 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)

Curve and Reference Array

- NewReferencePoint(XYZ)
- ReferencePointArray(ReferencePoint)
- NewCurveByPoints(ReferencePointArray)
- CurveReference = Curve.Geometry.Reference
- ReferenceArray(CurveReference)
- ReferenceArrayArray(ReferenceArray)

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)

Sketch

Extrusion	NewExtrusion(bool isSolid, CurveArrArray profile, SketchPlane sketchPlane, double end)

Revolution	NewRevolution(bool isSolid, CurveArrArray profile, SketchPlane sketchPlane, Line axis, double startAngle, double endAngle)
Sweep	NewSweep(isSolid, path, profiles)
Blend	NewBlend(isSolid, profile, direction)

Element Retrieval

ElementId

If the ElementId of the element is known, the element can be retrieved from the document.

Element filtering and iteration

This is a good way to retrieve a set of related elements in the document.

Selected elements

Retrieves the set of elements that the user has selected

Specific elements

Some elements are available as properties of the document

Use Filter to find Elements

The basic steps to get elements passing a specified filter are as follows:

First create a new *FilteredElementCollector*. Next apply one or more filters to the collection. Finally, get filtered elements or element ids (using one of several methods)

Divided Surface

Divided Surface Tile Patterns

- **Built-ins**
  - Rectangle
  - Rhomboid
  - Hexagon
  - HalfStep
  - Arrows
  - ThirdStep
  - ZigZag
  - Octagon
  - OctagonRotate
  - RectangleCheckerboard
  - RhomboidCheckerboard
  - Triangle_Flat
  - Triangle_Bent
  - TriangleStep_Bent
  - TriangleCheckerboard_Flat
  - TriangleCheckerboard_Bent

Understand Revit API elements

Revit Elements

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

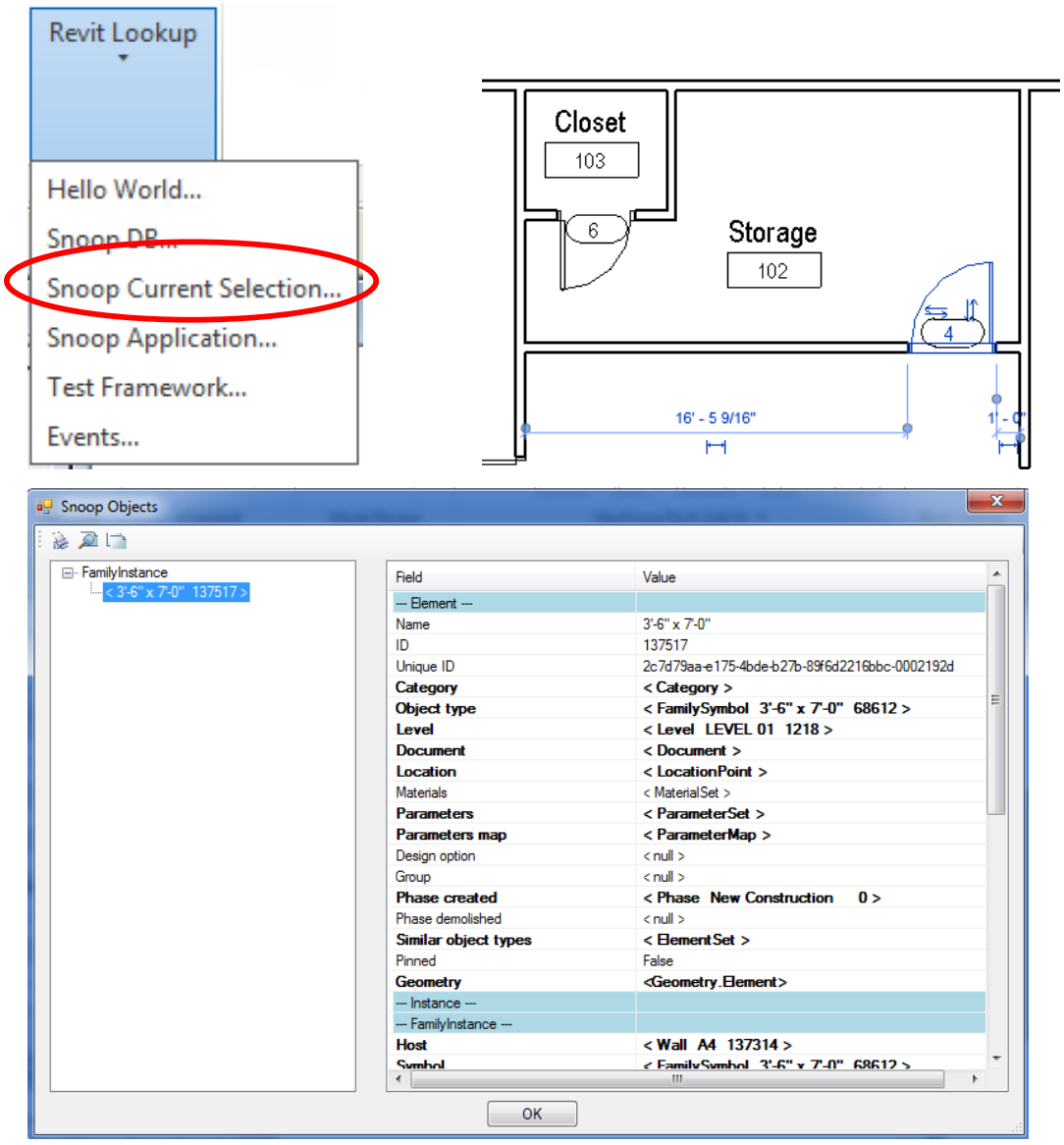
Element Classification

- Category
- Family
- Symbol
- Instance

Revit Lookup Tool

Use Revit Lookup Tool to find the classification of element that you need. Revit lookup tool can be installed by adding RevitLookup.dll and RevitLookup.addin into your Revit Addins folder.

Example: Select a door, then select from the add-ins tab, Revit Lookup > Snoop Current Selection… to see all the metadata associated with this door object.



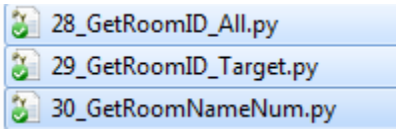
Using Revit LookUp Tool to find detailed element information of an object.

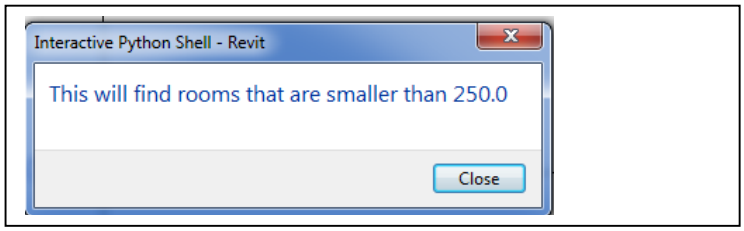
- Builtin Category: OST_Doors
- Family Instance: 3’-6”x7’-0”
- Family Symbol: PW_Flush-Single
- ElementID: 137517

Use scripts to create and change geometry in Revit

Find Rooms that are smaller than a certain square feet

Specify a target size, then iterate through all the rooms and find the rooms that are smaller than the target size.





Start by setting the target and communicate with the user through a dialog box.

```
Target = 250.0
TaskDialog.Show("Revit", 'This will find rooms that are smaller than ' + str(Target))
```

```
# Use a RoomFilter to find all room elements in the document.
Rmfilter = RoomFilter();

#Apply the filter to the elements in the active document
collector = FilteredElementCollector(doc);
collector.WherePasses(Rmfilter);
```

Next use a room filter through the Collector and get all the rooms in the project file.

Use the iterator to go through each room

```
#Get results as ElementId iterator
roomIdItr = collector.GetElementIdIterator()
roomIdItr.Reset()
prompt = 'The rooms that are smaller than ' + str(Target) + ' are:'
count = 0
while (roomIdItr.MoveNext()):
    roomId = roomIdItr.Current;
    #Warn rooms smaller than Target SF
    room = doc.GetElement(roomId)
```

Use if statement to check the room area size against the target size

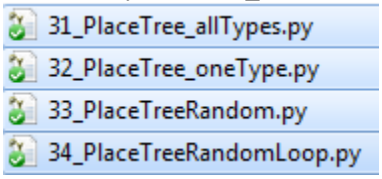
```
rmname= room.get_Parameter('Name').AsString()
print rmname
if (room.Area < Target):
    prompt += '\n\t' + rmname + ' (' + room.Number + ')'
    count = count + 1
```

In case no smaller rooms were found, return “No room Found”

```
if count == 0:
    prompt = 'No room found'
```

Placing Family Instance and types

Place a number of trees randomly of different types within a given area.
Tree family name, SI_Tree, has 5 types (20', 25', 30', 35', 50')



Start with some variable definition.

```
#Family symbol name to place.
symbName = 'SI_Tree'
count = 10
min = 10
max = 200
z = 0
MyTrees=[]
```

Use FilteredElementCollection with Category and Class filters to get all the family symbols.

```
collector = FilteredElementCollector(doc)
collector.OfCategory(BuiltInCategory.OST_Planting)
collector.OfClass(FamilySymbol)
```

Iterate through the collection until a matching family symbol is found and append it to the MyTrees List.

```
famtypeitr = collector.GetElementIdIterator()
famtypeitr.Reset()
#Search Family Symbols in document.
for item in famtypeitr:
    famtypeID = item
    famsymb = doc.get_Element(famtypeID)
    #If the FamilySymbol is the name we are lo
    if famsymb.Family.Name == symbName:
        #location to place family
        MyTrees.append(famsymb)
```

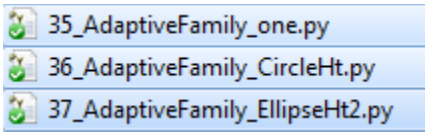
Once all the family symbol types have been appended to the MyTrees list, iterate through each type and place a tree for each count and each type.

```
for i in range(count):
    for tree in MyTrees:
        x = random.randint(min, max)
        y = random.randint(min, max)
        loc = XYZ(x,y,z)
```

Placing Adaptive Component Family

Place the frames_short along two concentric 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.

Frame_Short.rfa is an adaptive component family with 2 placement points



Start with some basic variable assignment.

```
#Family symbol name to place.
MySymbName = 'Frame_Short'

#Total number of points/family to place
TotalPoints = 31
```

Create a function that draws the points on an Ellipse and returns the ellipsePoints list.

```
#Function for points on an Ellipse
def DrawEllipsePoints(points, radius1, radius2, center):
    slice = 2 * math.pi / points
    ellipsePoints=[]
    for i in range(points):
        angle = slice * i
        newX = (center.X + radius1 * math.cos(angle))
        newY = (center.Y + radius2 * math.sin(angle))
        p = XYZ(newX, newY, 0)
        ellipsePoints.append(p)
        print p
        #refPoint = doc.FamilyCreate.NewReferencePoint(p)
    return ellipsePoints
```

Next define a function that will place an adaptive component with 2 placement points.

```
#CreateAdaptiveComponentInstance
def PlaceAdpComp(document, symbol, pt1, pt2, h):
    # Create a new instance of an adaptive component family
    instance = AdaptiveComponentInstanceUtils.CreateAdaptiveComponentInstance(document, symbol)

    # Change the height of the family
    param = instance.get_Parameter('Height1')
    param.Set(h)

    # Get the placement points of this instance
    placePointIds = []
    placePointIds = AdaptiveComponentInstanceUtils.GetInstancePlacementPointElementRefIds(instance)

    # Set the position of each placement point
    refpt1 = document.get_Element(placePointIds[0])
    refpt2 = document.get_Element(placePointIds[1])

    #New Points for the Adaptive Points to be placed
    newPt1 = pt1
    newPt2 = pt2

    #Find the difference between the 2 points as the translation for MoveElement()
    trans1 = newPt1.Subtract(refpt1.Position)
    trans2 = newPt2.Subtract(refpt2.Position)

    #Place the Adaptive Component using MoveElement()
    ElementTransformUtils.MoveElement(doc, placePointIds[0], trans1)
    ElementTransformUtils.MoveElement(doc, placePointIds[1], trans2)
```

Next use the filteredElementCollector and iterate through the collection.

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

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

#Search for Family Symbols in document.
for item in famtypeitr:
    famtypeID = item
    famsymb = doc.get_Element(famtypeID)
```

Use the DrawEllipsePoints function to generate the points on the ellipses.

```
#If the FamilySymbol is same name, create a new instance.
if famsymb.Family.Name == MySymbName:
    #Get points on inner ellipse (# of points, radius1, radius2, center)
    epts1=DrawEllipsePoints(TotalPoints, 10, 5, XYZ(0,0,0))

    #Get points on outer ellipse (# of points, radius1, radius2, center)
    epts2=DrawEllipsePoints(TotalPoints, 20, 10, XYZ(5,0,0))
```

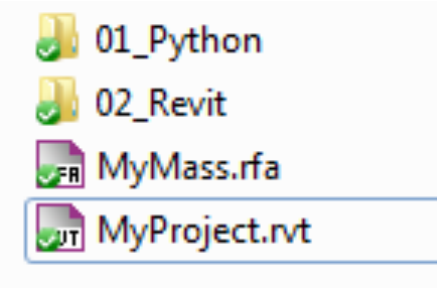
Set a default height, and an increment height based on height divided by TotalPoints (in feet).

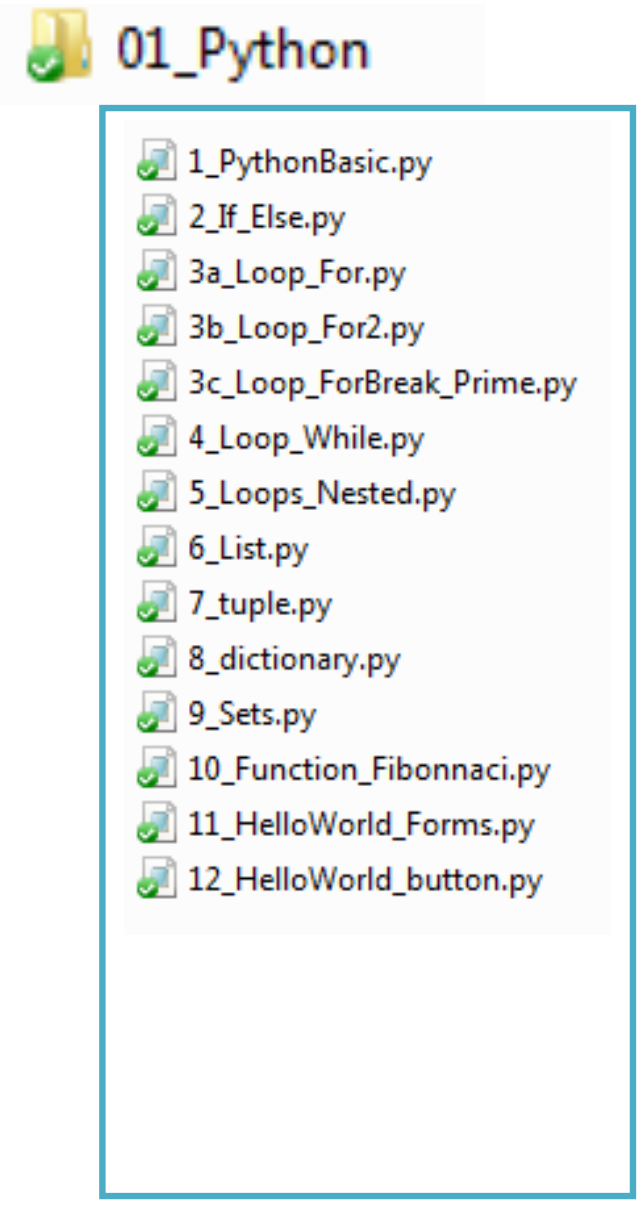
```
# Set default height
ht = 7.0
# For end height to be twice the start height
# setting height increment for each family
ht_inc = ((ht*12)/TotalPoints)/12
print ht_inc
```

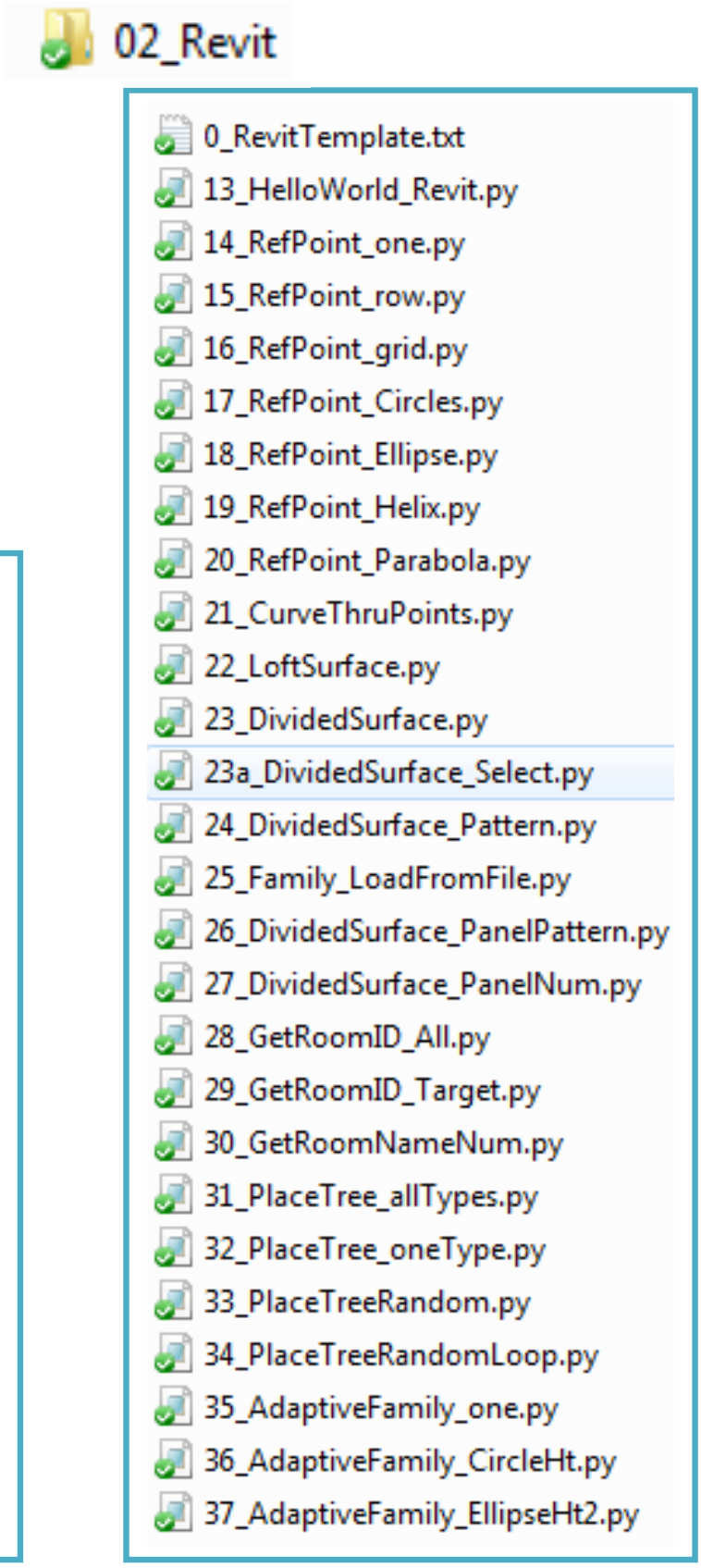
Use a for loop and the PlaceAdpComp function to place the frame_short family using two points, one on each ellipse.

```
#for i in range(TotalPoints):
for i in range(5):
    pt1 = epts1[i]
    pt2 = epts2[i]

    #location to place Adaptive Component family
    PlaceAdpComp(doc, famsymb, pt1, pt2, ht)
    ht += ht_inc
```

Datasets > Files Folder







Resources

Software Installation

The following software should be installed.

- IronPython-2.7.3.msi
  http://ironpython.codeplex.com/downloads/get/423690)
- Setup_RevitPythonShell_2013_r155.msi
  http://revitpythonshell.googlecode.com/files/Setup_RevitPythonShell_2013_r155.msi)
- Setup_RevitPythonShell_Vasari_Beta1.exe
  http://revitpythonshell.googlecode.com/files/Setup_RevitPythonShell_Vasari_Beta1.exe)
- Install RevitLookUp.dll (see RevitPython\RevitAPI\RevitLookup)
  Copy RevitLookup.dll and RevitLookup.addin to your local ProgramData folder
  (C:\ProgramData\Autodesk\Revit\Addins\2013)

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

Revit API Reference

- Autodesk Developer's Network
  - http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=19118898
- Revit API Developer's Guide (Wiki)
  - http://wikihelp.autodesk.com/enu?adskContextId=revitapi_landing&language=ENU&release=2013&product=Revit
- Revit API.chm
- Revit Lookup Tool
- Revit API Blog - The Building Coder by Jeremy Tammik
  - http://thebuildingcoder.typepad.com/blog/
- Nathan Miller's Revit API Notebook
  - http://theprovingground.wikidot.com/revit-api