Ifc rules

Advanced Building Information Modelling

11034 Autumn 2020 DTU Byg

Tim Pat McGinley



Contents

[Introduction 4](#_Toc55207093)

[Ifc Dashboard focussed on the ‘Gather’ BIM Use Case 5](#_Toc55207094)

[What we learnt from the IFC Dashboard assignment 6](#_Toc55207095)

[Advanced BIM is more about testing than modelling 6](#_Toc55207096)

[Assignment 2 7](#_Toc55207097)

[IFC Rules: BIM Use Cases example rules 8](#_Toc55207098)

[Python 11](#_Toc55207099)

[Appendix A: RWTH Download and Installation instructions 12](#_Toc55207100)

[Appendix B: BIM Use Case Wheel and Segments 13](#_Toc55207101)

[Appendix C: Remaining use cases with Rule suggestions 14](#_Toc55207102)

[Appendix D: Level of Development (LOD) Definition from DiKon 15](#_Toc55207103)

[Appendix E: Setting up Python / IfcOpenShell on your machine 16](#_Toc55207104)

[Step 1: Check if python installed on your machine and what version it is. 16](#_Toc55207105)

[Step 2: jump into and out of python from the command line / terminal (optional) 16](#_Toc55207106)

[Step 3: run your first python program. 17](#_Toc55207107)

[Step 4 add IfcOpenShell to python so that you can include it in your program. 18](#_Toc55207108)

[Step 5: Test your setup: 19](#_Toc55207109)

[Appendix F: Advanced Python scripts 20](#_Toc55207110)

[Example 1: Get the property sets of an element 20](#_Toc55207111)

[Example 2: Get the doors that bound a space (BoundedBy) - HARD 20](#_Toc55207112)

[Example 3: Get the doors that bound a space (BoundedBy) 0](#_Toc55207113)

[Example 4: Define a class and function to load models (Hard) 1](#_Toc55207114)

[Example 5: Compare geometry in different models 1](#_Toc55207115)

[Example 5a: Other collision functions to explore 1](#_Toc55207116)

[Example 6: Super fast property queries using selector (Currently working on this) 1](#_Toc55207117)

[Example 7a: Check the NUMBER of stories in different models 2](#_Toc55207118)

[Example 7b: Compare the storey ELEVATIONS in different models 2](#_Toc55207119)

[Example 7c: Are the ELEVATIONS in diff models the same? 1](#_Toc55207120)

[Example 8a: Property check 1](#_Toc55207121)

[Example 8b: Generic Property list 1](#_Toc55207122)

[Example 8c: Find entities based on a singlevalue property 1](#_Toc55207123)

[Example 9a: Door code check 2](#_Toc55207124)

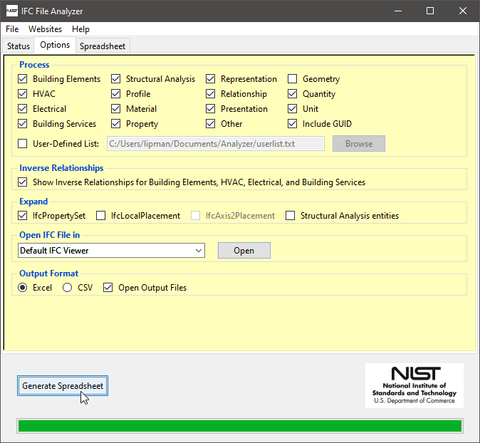
[APPENDIX G: Glossary of terms (Work in progress) 1](#_Toc55207125)

# Introduction

In the IFC Dashboard exercise we explored how to get information from [IFC files](https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/) without any traditional BIM software, imagining that we were in a situation where we did not have access to these due to loss of internet / licence and still needed to find out about the example BIM model.

To simulate this scenario / use case we used the [IFA tool](https://www.nist.gov/services-resources/software/ifc-file-analyzer) from NIST developed by Robert Lipman to convert the IFC file into an excel Spreadsheet. Our BIM use case in this example was to *model* ‘headline’ information about the project that might be of interest to other stakeholders in the project from information that was available in the file.

To achieve this we needed to:

1. [](https://www.nist.gov/services-resources/software/ifc-file-analyzer)Consider what information we wanted to present in the IFC Dashboard
2. Identify the information that we had available in the IFC
3. Work out how to use the IFC information to describe what we wanted to present in the dashboard
4. Try to use functions in excel to support this (although copy and paste is acceptable)

The process that we went through in this initial exercise is the same one that we will go through in most exercises working with IFC tools. Although sometimes we may not be sure about the potential use cases so may do step 1 before step 2.

For this exercise we used the ‘Duplex’ IFC example which is a [common example](https://www.nibs.org/page/bsa_commonbimfiles#project1) provided by [BuildingSmart](https://www.nibs.org/page/bsa_commonbimfiles#project1). There is a report on the Duplex model [here](https://portal.nibs.org/files/wl/?id=o5aj18qAVsjgrGdMP7YSEbkC1CijV7Mz). And the revit file is also available [here](https://portal.nibs.org/files/wl/?id=wQJuc4eUqn3icukr8TzuQ7wtBJovWHv1). There are more IFC examples [here.](http://www.ifcwiki.org/index.php/Examples)

# Ifc Dashboard focussed on the ‘Gather’ BIM Use Case

The Ifc Dashboard assignment enabled us to explore the following use cases:

**Quantify** Floor area

We were able to get the area property of the IfcSpace objects and add these up to get the total GFA of the building. This assumes that the GFA of the building is equal to the sum of the IfcSpaces. This is not necessarily true but the idea here is to be able to get specific information based on the

**Monitor** Materials

Generate a list of the materials used in the project for specific elements.

**Quantify** Materials

This use case extends the previous case, by seeking to quantify the volume of the slabs for instance to determine the amount of material to be used in the project.

**Capture** Material costs

Estimates for costs can be provided if the material and element.

**Monitor** Costs

Based on the Quantify materials and Monitor Materials use cases it is then possible to Monitor the costs for the project.

These use cases can be combined together to make new use cases as can be seen in the figure below, so to solve your BIM use case, you may need to first solve other ‘sub’ use cases.

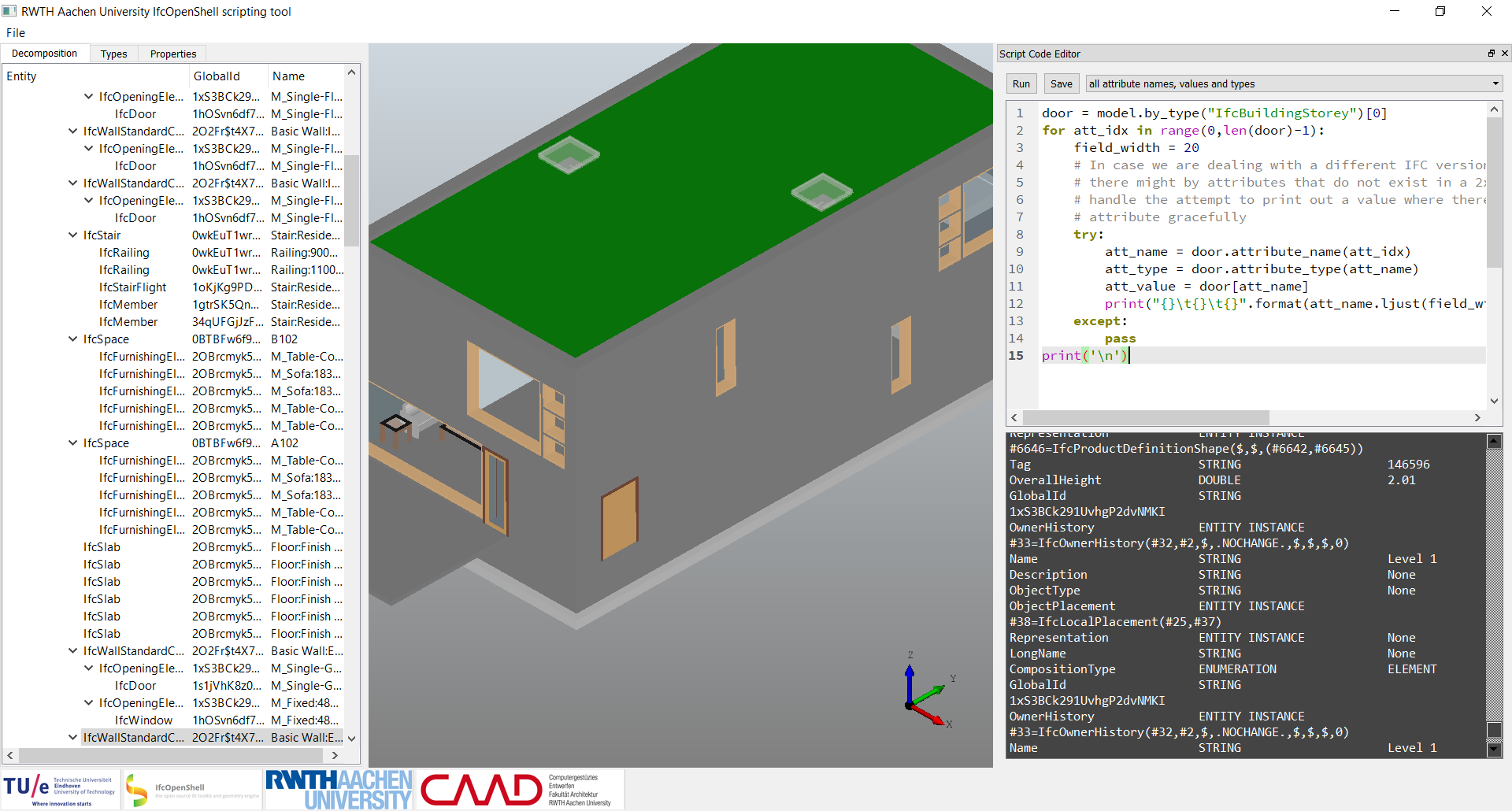
# What we learnt from the IFC Dashboard assignment

1. It was difficult to find the information and navigate the PSets etc - There must be a simpler way to do this.
2. In excel we could model the data, but it was difficult to ‘query’ it.
3. It took a long time to get the results and the number of use cases that we could support in that time was limited.
4. It is a great idea to have a format like IFC that supports interoperability in the AEC but working with IFC is complicated, however if we can get our heads around this complexity

# Advanced BIM is more about testing than modelling

An alternative approach is to model the IFC data so that we can query it. An example is the query function in the RWTH viewer (RWTH Viewer install instructions ). Also you can view the [video here](https://web.microsoftstream.com/video/533e22e3-5719-4101-9893-0d78a9da38e9) if you are enrolled on this semester’s course.

**Query Window**



**Result Window**

In the RWTH viewer example we can see that we can type code ‘snippets’ these are small bits of code that we can use to query the model. The viewer comes with a set of different snippets for you to try from the dropdown menu at the top of the query window. If you then click run in the top left hand of the query window you will see the result of your query in the Result window. Some queries run on a selection you have made in the 3d window (i.e*. print properties of current selection*) and some select all instances of an entity (*print all wall ids*).

# Assignment 2

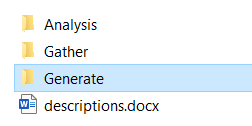
To design and write an IFC rule in Python for each of the Gather, Generate and Analyze use case segments (3 rules in total). The assignment can be done just using the RWTH viewer. However for some cases like when we try and use more than one model we need to setup our own python / ifcopenshell development environment.

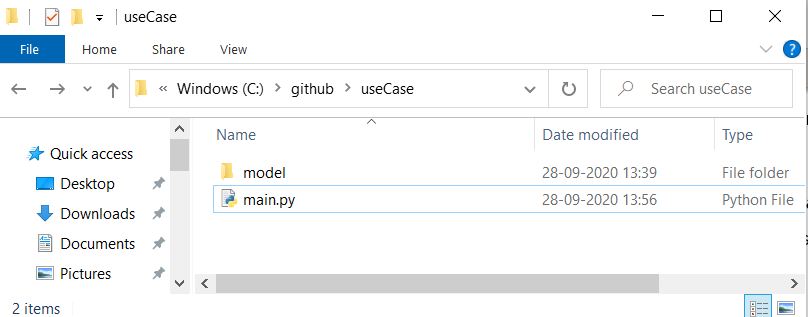
*Therefore the core of the assignment is:* For each of the three rules, produce 1 x A4 description and 1 a4 that include the code snippet. Therefore, the assignment report should include 6 pages in total.

|  |
| --- |
| page 1: Selected Gather case description |
| page 2: Selected Gather case snippet |
| page 3: Selected Generate case description |
| page 4: Selected Generate case snippet |
| page 5: Selected Analyse case description |
| page 6: Selected Analyse case snippet |

This should be submitted as a pdf / doc / docx / ods / odt Please be careful with the snippets and test them. I should be able to copy and paste them into my RWTH viewer - please test this! I Have about 120 of these to test so if we have to do the indenting for every rule it will take forever ☹

Because of this and because some of you may be using the python command line rather than RWTH you also have the option to include code folders for each of your use cases. similar to the image below. So your final submission would be a zip folder that would include 3 code folders and your report.



Then inside each folder you would save a main.py file and your models folder, that would look something like this.

Even if you are just using the RWTH viewer you could copy your snippet code from RWTH into a text / .py file and put this into the code folder. Then I would be able to easily check the code without having to guess the indentation 😊

# IFC Rules: BIM Use Cases example rules

We can then use the queries as rules to find out something about the data. In this assignment we are looking to write one rule for each of the three use case segments in Penn State’s uses of BIM. The relative difficulty of creating the rule is also given from easy to hard. The communicate and realize uses are probably not appropriate for this assignment as we either don’t have the required information or they are beyond what could be expected at this stage. However, if you have an idea for them you are welcome to try.

|  |  |  |
| --- | --- | --- |
| Gather | | |
| **CAPTURE**  [EASY] | *Represent or preserve the current status of the facility and facility elements* | |
| Rule: | Check the number of spaces in the 3.  **Example 1a**: *loop through the entities and then add one to spaces\_in\_model each time we find an instance of that entity* |
| spaces\_required = 21  spaces\_in\_model = 0  for entity in model.by\_type("IfcSpace"):  spaces\_in\_model+=1  print("\nThere are "+str(spaces\_in\_model)+" spaces in the model")  if (spaces\_required == spaces\_in\_model):  print ('RESULT: The number of spaces is correct')  else:  print ('RESULT: The number of spaces is wrong') |
| Example 1b: *using len() to count the number of entities (without having to loop through all of them).* |
| spaces\_required = 21  spaces\_in\_model = len(model.by\_type("IfcSpace"))  print("\nThere are "+str(spaces\_in\_model)+" spaces in the model")  if (spaces\_required == spaces\_in\_model):  print ('RESULT: The number of spaces is correct')  else:  print ('RESULT: The number of doors is wrong') |
| RESULT: |
| There are 21 spaces in the model  RESULT: The number of spaces is correct |

|  |  |  |
| --- | --- | --- |
| **QUANTIFY**  [MEDIUM] | *Express or measure the amount of a facility element* | |
| Rule: | In this example we will try to quantify the length of the beam. We will base this on the for loop we defined in example 1A.  Example 2: Quantify use case code example |
| total\_beam\_Lenght = 0  for entity in model.by\_type("IfcBeam"):  #we need to get the attributes  for relDefinesByProperties in entity.IsDefinedBy:  for prop in relDefinesByProperties.RelatingPropertyDefinition.HasProperties:  #and then get the attribute we are looking for  if prop.Name == 'Length':  #add the length to the total length  total\_beam\_length += prop.NominalValue.wrappedValue  print("\nThere are "+str(total\_beam\_length)+" meters of beam in the model") |
| RESULT: |
| There are 43.8523340981 meters of beam in the model |
| **MONITOR**  [HARD] | *Collect information regarding the performance of facility elements and systems* | |
| Rule: | This is hard because it is difficult to think of a way we could make it work in our example file, but it is not necessarily impossible … |
| **QUALIFY**  [HARD] | *Characterize, or identify facility elements status* | |
| Rule: | Is the element at least **Level of Development (LOD)** 200 [DiKon]. Hint: LOG is hard to calculate but LOI might be a bit easier – if basing LOI test on the real DiKon specification it still counts as HARD! – but it is possible.  RESULT: The LOI of the slabs is 325 |

|  |  |  |
| --- | --- | --- |
| Generate | | |
| **PRESCRIBE**  [MEDIUM] | *Determine the need for and select specific facility elements.* | |
| Rule: | Test to see if required elements are present in the model, for instance, are there basins in the WCs?  RESULT: You have WCs, don’t forget the Basins! |
| **ARRANGE**  [MEDIUM] | *Determine location and placement of facility elements.* | |
| Rule: | Are the elements placed on the correct storey / level or in the correct room? – what other examples can you think of?  RESULT: The basins are placed in WCs – good job! |

|  |  |  |
| --- | --- | --- |
| **SIZE**  [HARD] | *Determine the magnitude and scale of facility elements.* | |
| Rule: | Examples: check the dimensions of spaces, the shape of a steel beam, or the size of ductwork. This is similar to the work we did in IFC Dashboard.  REQUIRES: Specify the minimum depth of a slab.  RESULT: The depth of slabs is OK! |

|  |  |  |
| --- | --- | --- |
| Analyze | | |
| **COORDINATE**  [EASY] | *Ensure the efficiency and harmony of the relationship of facility elements* | |
| Rule: | Check the storey elevations and names are the same between the disciplinary models. |
| **FORECAST**  [HARD] | *Predict the future performance of the facility and facility elements* | |
| Rule: | Calculate if the energy use for instance exceeds the project budget for a year. This may also mean that you need to define this budget.  REQUIRES: Specify predicted energy use per m2/yr (you will not be able to get this directly form the IFC)  RESULT: The forecasted energy use for the building for a year. |
| **VALIDATE**  [HARD] | *Check or prove accuracy of facility information and that is logical and reasonable* | |
| Rule: | Check the produced information (IFC files) against the requirements for the project, this could include regulatory requirements. Can you for instance identify a clause in [BR18](https://bygningsreglementet.dk/) that you could test the model for? |

So the task for IFC Rules is to select 3 from the Gather, Generate and Analyze use case segments. For interest the last two segments are included in Appendix B: BIM Use Case Wheel and Segments, however it is not anticipated that it will be possible to use these in the IFC Rules assignment, although we would be interested if you think you could 😊. We started to look at BIM use cases in last week’s lecture and we will revisit them in the next assignment. Each use case on the wheel has 4 characteristics.

Facility Element

The system of the facility on which the BIM Use will be implemented.

Facility Phase

The point in the facility’s lifecycle at which the BIM Use will be implemented.

Discipline

The party by whom the BIM Use will be implemented. The models are typically divided into different disciplinary models, for instance Architecture and MEP.

Level of Development

The degree of granularity to which the BIM Use will be implemented. For more information on Level of Development see **Appendix D:** LOD

# Python

*Further thinking….*

Thomas Krijnen the author of IfcOpenShell has recently produced this awesome [web viewer](https://view.ifcopenshell.org/) that enables us to view IFC files on the web.

How might this example be developed in the future to support the use cases we identified in the first assignment?

The code snippets that you can see in the RWTH view are written in Python. Python is the most popular programming language in the world and is highly regarded for its scientific libraries that make it easier to use advanced approaches such as machine learning and natural language processing in your code.

Excitingly IfcOpenShell is also possible to access in python as evidenced in the previous example. The RWTH viewer is a great starting point but the really interesting thing is using Python directly to develop your queries. This is what we will aim to do in this assignment. See Appendix A: RWTH Download and Installation instructions Appendix A: RWTH Download and Installation instructions

Once you have the viewer running and you have familiarised yourself with the interface and how to run the different snippets, please make your way through the following tutorials.

1. The [byte of python book](https://python.swaroopch.com/first_steps.html), is a great to start to learn more about the language. You can skip the installation stage unless you want to run scripts from the console – which would be awesome, but is not necessary to complete the assignment., so here I have provided the link to the getting started chapter.
2. This is a great [tutorial](https://github.com/jakob-beetz/IfcOpenShellScriptingTutorial/wiki/00:-Hello-IFC-world) to start with working with RWTH
3. [Working with the IFC model](file:///C:\Users\admin-timmc\Downloads\Working%20with%20the%20IFC%20model)
4. [Inspecting IFC instance objects](https://github.com/jakob-beetz/IfcOpenShellScriptingTutorial/wiki/02:-Inspecting-IFC-instance-objects) (gets us the attributes but not properties)
5. [Selecting and visualizing](https://github.com/jakob-beetz/IfcOpenShellScriptingTutorial/wiki/03:-Selecting-and-visualizing) (for more interactivity in the rules)

# Appendix A: RWTH Download and Installation instructions

The link below provides an overview of how to get the RWTH up and running on your computer, unfortunately this is currently only for windows machines but we are working on developing a version that also runs on mac in the future. I could write out the instructions but Jakob Beetz has already done an amazing job of this below. I have included a table below, to answer some of the common questions we get about setting up the RWTH viewer.

<https://github.com/jakob-beetz/IfcOpenShellScriptingTutorial>

|  |  |  |
| --- | --- | --- |
|  | Zip folder\*  <http://caad.arch.rwth-aachen.de/download/rwth_viewer.zip> | snippets.conf file\*  <https://raw.githubusercontent.com/jakob-beetz/IfcOpenShellScriptingTutorial/master/src/snippets.conf> |
|  | The zip folder that contains the application and all its interdependencies |  |
|  |  |  |
| Where to place it on your computer | It looks like this can be placed anywhere (not downloads 😊) as long as you unzip it. | The location of this file is important, please follow the instructions on the RWTH viewer page. |

\* If the links are broken please go the [IfcOpenShellScriptingTutorial](https://github.com/jakob-beetz/IfcOpenShellScriptingTutorial) tutorial on github

# Appendix B: BIM Use Case Wheel and Segments

7aA close up of text on a white background

Description automatically generated

# Appendix C: Remaining use cases with Rule suggestions

|  |  |  |
| --- | --- | --- |
| Communicate | | |
| **VIZUALIZE**  [N/A] | *Form a realistic representation of a facility or facility elements* | |
| Rule: | Check (possibly using the RWTH viewer) what elements are modelled. is something missing – this requires your interpretation. It might not be relevant in this assessment |
| **TRANSFORM**  [N/A] | *Objective:* to modify information and translate it to be received by another process. | |
| Rule: | Elements transformed accurately between the files? Revit / IFC mapping…? |
| **DRAW**  [N/A] | *Objective:* to make a symbolic representation of the facility and facility elements. | |
| Rule: | Drawings / Symbolic representations are up to date? |
| **DOCUMENT**  [N/A] | *Objective:* to create a record of facility information including the information necessary to precisely specify facility elements | |
| Rule: | Are elements specified in the BIM model? |

|  |  |  |
| --- | --- | --- |
| Realize | | |
| **FABRICATE**  [N/A] | *Objective:* to use facility information to manufacture the elements of a facility. | |
| Rule: | Could get the geometry of the element and test if it could be produced automatically using robots / CNC for instance (out of scope of this assignment) |
| **ASSSEMBLE**  [N/A] | *Objective:* to use facility information to bring together the separate elements of a facility. | |
| Rule: | Check the assembly of different disciplinary models including construction sequence. |
| **CONTROL**  [N/A] | *Objective:* to use facility information to physically manipulate the operation of executing equipment. | |
| Rule: | Devise a test to understand the occupancy of the finished building under different use conditions. |
| **REGULATE**  [N/A] | *Objective:* to use facility information to inform the operation of a facility element. | |
| Rule: | Identify entities, attributes and properties that could be linked in the future to real time sensing in the building. How could you test that the BIM model is able to support this use case? |

# 

# Appendix D: Level of Development (LOD) Definition from DiKon

Level of Development (LOD) describes explicitly which information about model elements must be present in the BIM at different stages during the design and construction process. In Denmark this can be defined using [DiKon](https://www.dikon.info/download/specification-of-building-parts_en_v-3/?wpdmdl=2424&masterkey=5d9c53033ddf4). In this assignment LOD is a potential approach for the Qualify use case. LOD is the result of LOG (Level of Geometry), LOI (Level of Information and

In a recent thesis exploration we realized that LOG (Level of Geometry) is difficult to assess but LOI is easier to build rules for

LOD for building parts is comprised of:

Level of Reliability (LOR) describes the reliability of the information provided for the building part and it’s properties.

**Level of Geometry (LOG)** describes the building parts’ geometric representations and the extent of secondary components/parts.

**Level of Information (LOI)** describes the building parts’ properties contained in, linked to, or in some other way connected.

In a recent thesis we realised that LOI is relatively easy to assess in a model based on DiKon but LOG and LOR are more difficult. If you select the Quality use case you could focus on how to get the LOI of the element and check this against the DiKon specification.

# Appendix E: Setting up Python / IfcOpenShell on your machine

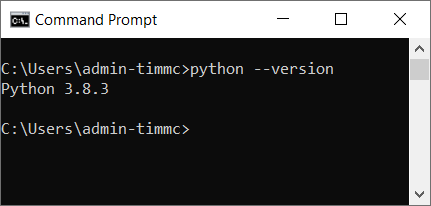
We have been using the RWTH viewer, which is great to get us started, but we do not always need the graphical interface and you might like to run python form the command line / terminal. The advantage to this approach is you can have better control of the script and the libraries that you are including. Ok now we have Python set up we can add the IfcOpenShell Library.

## Step 1: Check if python installed on your machine and what version it is.

To do this we need to add download the correct version of IfcOpenShell which depends on your operating system and the version of python you are using. To find this out.

1. Go to the command line in windows or the Terminal in mac and type:

python --version



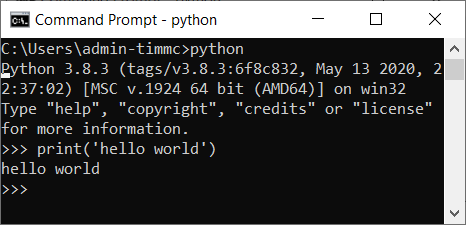
This should tell you the version of python that you are using. For instance you can see here that I am using 3.8.3. So I can download the correct version from <http://ifcopenshell.org/python>.

## Step 2: jump into and out of python from the command line / terminal (optional)

You can see that this time I have typed

python

And it has given me the python interpreter >>>

I can now type python commands into this and it will give me an output. for instance:

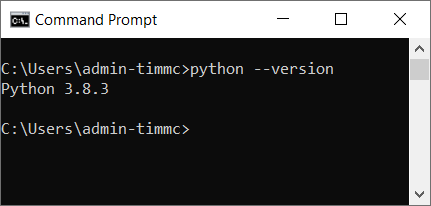
print(‘hello world’)

Produces the result

hello world

To get out of the python interpreter type

exit()

So you now shouldn’t be able to see the >>> prompt and it should instead be something more like  or the equivalent on your machine.

## Step 3: run your first python program.

It is a bit annoying to have to write out each line separately into the python interpreter, to get around this we can write a python script in a text editor and then save this a .py file.

So first we set up a location for your program. I am using c:/github as a root location on my machine, it is up to you where you choose, but it should be somewhere easy to find.

#### Step 3.1 creating a .bat file to get to your location (optional)

This is an optional step, but I find it makes it easier - I don’t have a mac equivalent yet, and it is not an essential step. so create a new file on your desktop and rename it to [name of the program].bat for instance mine is called rule.bat.

Open that file in a text editor for instance notepad++ or sublime and copy in the following.

@ECHO OFF

cmd.exe /K "cd C:\github\rule && c:"

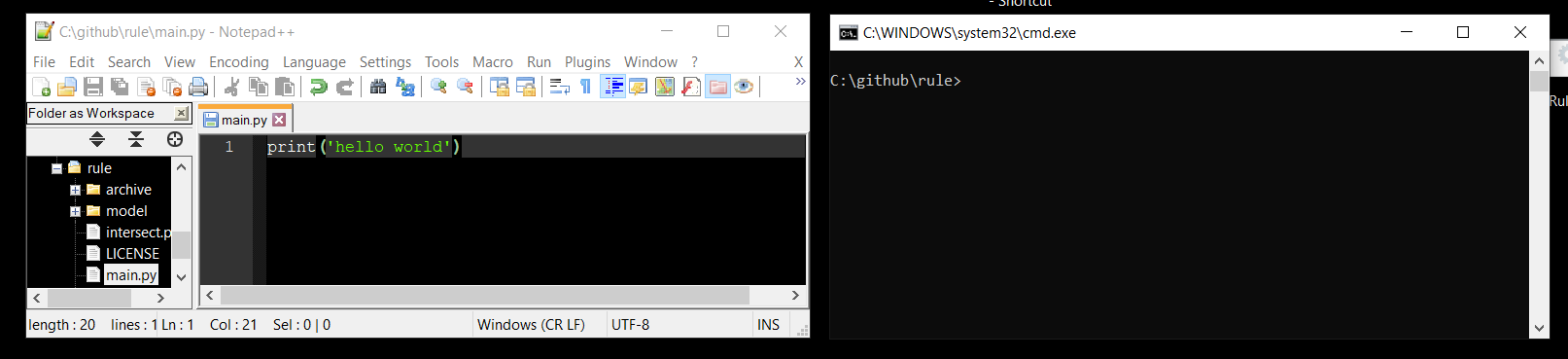
Save it and then when you double click it on your desktop it will automatically open the cmd line in the right place. Please note that the bat file uses backslashes.



Ok so now we have set that up go to the program folder (in our case c:/github/rule). In that folder create a new file called main.py. Open up that folder in a text editor (notepad++ for instance) and in the file type:

print (‘hello world’)

Now save the file - you can keep the text editor open so that you have the text editor and cmd line side by side for instance…

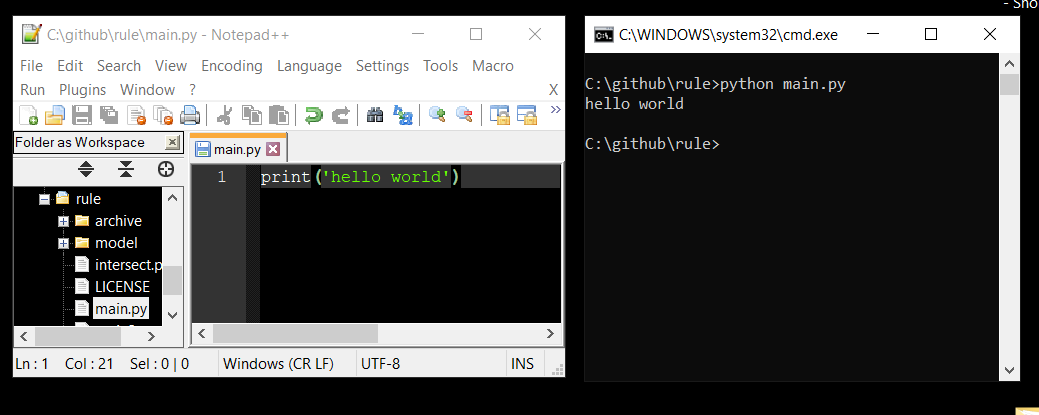


#### Step 3.2

Ok now, type into the cmd line

python main.py

This will run your program and you will see the result in the command line.



Cool eh? you just wrote and ran your first proper program in Python - good job!

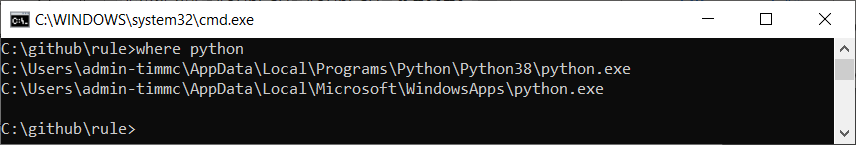
## Step 4 add IfcOpenShell to python so that you can include it in your program.

Having worked out what version of Python you have in step 1 we now need to work out where it is installed so that we can add the ifcOpenShell folder you downloaded in Step 1. (<http://ifcopenshell.org/python>). On a windows machine I will type …

where python

on a mac I can type:

which python

into Terminal which should give a similar result. On my machine it says two places, I choose the first one.

So I will go to

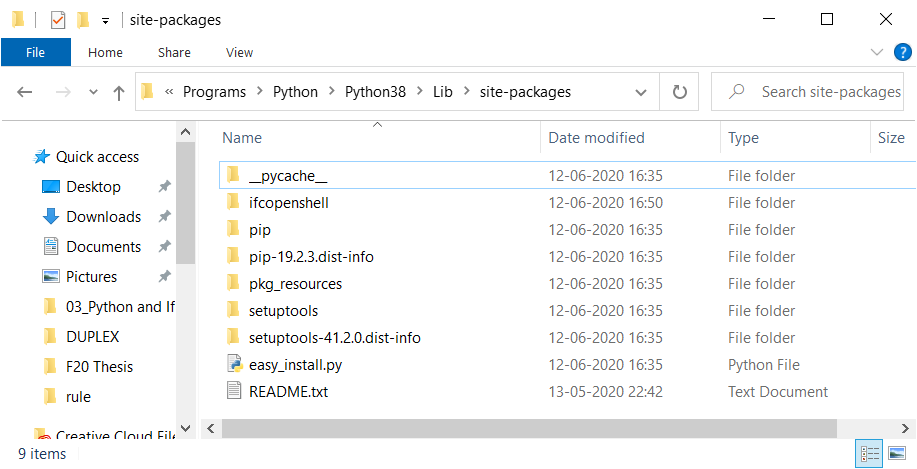
C:\Users\XXXX-XXXX\AppData\Local\Programs\Python\Python38\

n.b your location will not be XXXX-XXXX - this is an example!

and from there I will open the Lib/site-packages so I end up at

C:\Users\*admin-timmc*\AppData\Local\Programs\Python\Python38\Lib\site-packages

The downloaded ifcopenshell folder should go in there.



## Step 5: Test your setup:

Open up your program folder for me it was (c:/github/rule), but this could be anywhere for you. In the text editor open up the main.py file in your folder. Now in that folder type the following

import ifcopenshell

model = ifcopenshell.open('model\Duplex\_A\_20110907.ifc')

for obj in model.by\_type('IfcSlab'):

print(obj.Name)

Save that and then run it from the command line following Step 3.2 if this works, have a go at some of the code examples in the code examples document.

# Appendix F: Advanced Python scripts

*Appendix Firstly, as always massive thanks to Thomas Krijnen for most of this source, which has just been commented here, but is mostly intact from Thomas’s original examples from different locations.*

## Example 1: Get the property sets of an element

Only include the first chunk of text if you are NOT doing this in the RWTH viewer, in the viewer the import of the ifcopenshell library and getting the file has already been done for you and declared as model. Therefore in these examples we are also going to use model.

# ########### this is required if running from console ############

import ifcopenshell

model = ifcopenshell.open('model\Duplex\_A\_20110907.ifc')

# ########### end of required if running from console #############

# ############## code below needed in both cases ##################

# this just gets you the entity, defined here as wall

# feel free to change this to your needs

wall = model.by\_type('IfcWall')[0]

for definition in wall.IsDefinedBy:

# To support IFC2X3, we need to filter our results.

if definition.is\_a('IfcRelDefinesByProperties'):

property\_set = definition.RelatingPropertyDefinition

# Might return Pset\_WallCommon

print(property\_set.Name)

# ###################### end of example ###########################

## Example 2: Get the doors that bound a space (BoundedBy) - HARD

This example works to get you the doors (line 13) that bound the space (line 8)

# ########### this is required if running from console ############

import ifcopenshell

model = ifcopenshell.open('model\Duplex\_A\_20110907.ifc')

# ########### end of required if running from console #############

# ############## code below needed in both cases ##################

for space in model.by\_type("IfcSpace"):

near = space.BoundedBy

print("\n\t####{}\n".format(space.Name))

for objects in near:

if (objects.RelatedBuildingElement != None):

if (objects.RelatedBuildingElement.is\_a('IfcDoor')):

print(objects.RelatedBuildingElement.Name)

# ###################### end of example ###########################

## Example 3: Get the doors that bound a space (BoundedBy)

For this example we have to include an additional library, but it provides a really cool approach. Also please note that this example uses the optimized version of the Duplex model. This is also available in your models folder. Optimised versions of files are much smaller, they are optimized using a great tool ([Solibri IFC Optimizer](https://www.solibri.com/solibri-ifc-optimizer)) from Solibri. The idea is that it can be used to make IFC files easier to share.

# We need all this code and we can’t run it from RWTH viewer

import ifcopenshell

# That was normal the new bit is this geom lib below

import ifcopenshell.geom

# ok, so we are calling it fn (for file name here) - lets stick to that

fn = "model/Duplex\_A\_20110907\_optimized.ifc"

# based on the fn we can now create the model which is called f

f = ifcopenshell.open(fn)

# a specific wall is defined here based on its GlobalID

# we are working with standard files so you should also be able to find this.

# the small diff is that this is working on the optimized version

wall = f["2O2Fr$t4X7Zf8NOew3FLPP"]

# This is the magic code that loads the geometry for the models into its own model - so that we can query the geometry (and in this case the

tree\_settings = ifcopenshell.geom.settings()

tree\_settings.set(tree\_settings.DISABLE\_OPENING\_SUBTRACTIONS, True)

t = ifcopenshell.geom.tree(f, tree\_settings)

~~# you need the code below in both RWTH and if running it directly.~~

~~for space in model.by\_type("IfcSpace"):~~

~~near = space.BoundedBy~~

~~print("\n\t####{}\n".format(space.Name))~~

~~for objects in near:~~

~~if (objects.RelatedBuildingElement != None):~~

~~if (objects.RelatedBuildingElement.is\_a('IfcDoor')):~~

~~print(objects.RelatedBuildingElement.Name)~~

## Example 4: Define a class and function to load models (Hard)

For this example we will work with classes and functions to load the model, the reason for this is it will make it much more simple when we try and load multiple models in the next example.

# We need all this code and we can’t run it from RWTH viewer

import ifcopenshell

import ifcopenshell.geom

# we need this module to tell us how long our code took to run

import time

start\_time=time.time()

# This is the tree settings - you shouldn't need to change this. keep it as is.

# it is a smart way to structure the geometry of the model,

# so that we can do super fast 'bounding box' tests, where the

# smallest possible box isnt drawn around the object.

# the we can then do fast clash detection on those boxes.

# i.e. does the bounding box of the slab intersect the bounding box of any columns?

tree\_settings = ifcopenshell.geom.settings()

tree\_settings.set(tree\_settings.DISABLE\_OPENING\_SUBTRACTIONS, True)

# This is a class, - your first one! good job!

# it defines an object (Model) that has attributes (i.e. name)- cool eh?

# using this means we can get attribute info about the geometry model

class Model:

def \_\_init\_\_(self,name, file, geometry, load\_time):

self.name = name

self.file = file

self.geometry = geometry

self.load\_time = load\_time

# this is a function, we use functions when we want to repeat the same process in the code

# in python we need to define (def) the function before we can call it.

# we call the function below with model=getGeometry('ARCHI',f\_ifc, tree\_settings)

# the function makes an instance of the model class defined above and 'returns'

# this to us in.

def getGeometry(model\_name,file,tree\_settings):

start\_time=time.time()

a = ifcopenshell.geom.tree()

a.add\_file(file, tree\_settings)

load\_time = time.time()-start\_time

mod = Model(model\_name,file,a,load\_time)

return mod

# now that we have done all that work above defining classes and functions,

# the code below can be quite simple (it is only 3 lines.

# you can see that it is important to name the functions and classes in a meaningful way to

# contribute to the legibility of your code.

f\_ifc = ifcopenshell.open("model/Duplex\_A\_20110907\_optimized.ifc")

f\_geo = getGeometry('ARCHI',f\_ifc, tree\_settings)

print("\n\t{} Model took {:06.2f} seconds to load".format(f\_geo.name,f\_geo.load\_time))

## Example 5: Compare geometry in different models

This code enables you to load in different models into the same geometry model / tree (line 16). The arch model is added on line 20 and the MECH model is added on line 27. Line 31 and 32 define the Ifc classes that you will use for your clash detection, in this example we are identifying clashes between IfcSpace and IfcFlowSegment as the IfcFlowSegment is something that definitely appears in the MECH model.

For more code examples for detection check out this awesome page

import ifcopenshell

import ifcopenshell.geom

import time

# setup

start\_time=time.time()

tree\_settings = ifcopenshell.geom.settings()

tree\_settings.set(tree\_settings.DISABLE\_OPENING\_SUBTRACTIONS, True)

# this gets the architectural model

a\_ifc = ifcopenshell.open("model/Duplex\_A\_20110907\_optimized.ifc")

# this gets the mechanical model

m\_ifc = ifcopenshell.open("model/Duplex\_M\_20111024\_optimized.ifc")

# this is a sperate geometry model tree

t = ifcopenshell.geom.tree()

start\_time=time.time()

# this adds the architecture geometry to the tree

t.add\_file(a\_ifc, tree\_settings)

load\_time = time.time()-start\_time

print("\n\t{} Model took {:06.2f} seconds to load".format('ARCH',load\_time))

start\_time=time.time()

# this adds the mechanical geometry to the tree

t.add\_file(m\_ifc, tree\_settings)

load\_time = time.time()-start\_time

print("\n\t{} Model took {:06.2f} seconds to load".format('MECH',load\_time))

total\_clashes = 0

obj1 = "IfcSlab"

obj2 = "IfcFlowSegment"

start\_time=time.time()

# ok so in this example I want to take

clashed = a\_ifc.by\_type(obj1)

for clash\_object in clashed:

#print('### {}'.format(space.Name))

for obj in t.select\_box(clash\_object):

if (obj.is\_a(obj2)):

#print ('\t - {}'.format(obj.Name))

total\_clashes+=1

print("\n\tTotal clashes between {} and {} : {:6} ".format(obj1,obj2,total\_clashes))

load\_time = time.time()-start\_time

print("\n\tClash detection took {:06.6f} seconds to complete".format(load\_time))

## Example 5a: Other collision functions to explore

Extend the previous examples with the following commands ….

# extend the bounding box of the object being tested -> wall in this case

t.select\_box(wall, extend=1.)

# extend the bounding box of the object being tested

# and state that it has to be completely inside

t.select\_box(wall, extend=0.001, completely\_within=True)

# get the geometry in t that clash with point 0,0,0

t.select((0.,0.,0.))

# get the geometry in t that intersect a bounding box defined by (-1.,-1.,-1.) -> (1.,1.,1.)

t.select\_box(((-1.,-1.,-1.),(1.,1.,1.)))

# get the t geometry that intersect a bounding box defined by (-2.,-2.,-2.) -> (10.,10.,10.)

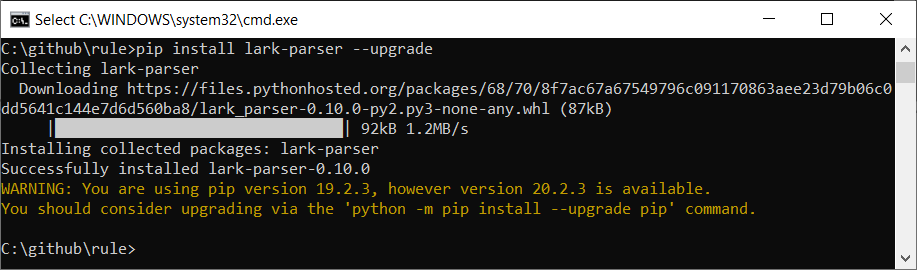
# and is completely contained by it

t.select\_box(((-2.,-2.,-2.),(10.,10.,10.)), completely\_within=True)

## Example 6: Super fast property queries using selector (Currently working on this)

Ok, so this is pretty straight forward, the only thing that might mess this is up, is that selector has a dependency called lark, to install lark you can follow the guide [here](https://github.com/lark-parser/lark). Or if feeling brave, just get the command line open and type:

pip install lark-parser --upgrade



You can look at the code examples here, to see what would be possible with it.

Examples for how to use it:

#import the module and setup

import ifcopenshell.util  
from ifcopenshell.util.selector import Selector  
model = ifcopenshell.open('model\Duplex\_A\_20110907.ifc')  
 selector = Selector()

#this is equivalent to model.by\_type(‘IfcWallStandardCase’)

walls = selector.parse(model, '.IfcWallStandardCase')  
 print(walls)  
 noWalls = len(walls)  
 print("There are {} walls in the model".format(noWalls))

#this is equivalent to searching by GlobalId

wall = selector.parse(model, '.IfcWallStandardCase'[GlobalId = "2O2Fr$t4X7Zf8NOew3FLOH"]')   
 print(wall)

#you can search for elements based on their properties  
#here you find all the external walls

extWalls = selector.parse(model, '.IfcWallStandardCase'[Pset\_WallCommon.IsExternal = "True"]')  
 noExtwalls = len(extWalls)  
 print("There are {} external walls in the model".format(noExtwalls))

#here you find all the spaces on level 1

spaces = selector.parse(model, '.IfcSpace[PSet\_Revit\_Constraints.Level = "Level 1"]')  
 noSpaces = len(spaces)  
 print("There are {} spaces on level 1".format(noSpaces))

#find all walls with a volume above 5

wallsVol = selector.parse(model, '.IfcWallStandardCase[PSet\_Revit\_Dimensions.Volume > "5"]')  
 noWallsVol = len(wallsVol)  
 print("{} walls out of {} walls in the model have a volume above 5".format(noWallsVol,noWalls))

## Example 7a: Check the NUMBER of stories in different models

This example checks to see if different models have the same number of stories.

import ifcopenshell

# this gets the architectural model

a\_ifc = ifcopenshell.open("model/Duplex\_A\_20110907\_optimized.ifc")

# this gets the mechanical model

m\_ifc = ifcopenshell.open("model/Duplex\_M\_20111024\_optimized.ifc")

# Ok so above we loaded 2 models...

a\_stories = a\_ifc.by\_type('IfcBuildingStorey')

m\_stories = m\_ifc.by\_type('IfcBuildingStorey')

**# first we need to check if they have the same number of stories**

**# check to see if the number of stories is the same in the different models...**

if (len(a\_stories) == len(m\_stories)):

print ('\n\tnumber of stories matches')

**# else if that is not true ... is a BIGGER than m ...**

elif (len(a\_stories) > len(m\_stories)):

print ('\n\tmodel\_a has more stories than model\_b')

**# else if that is not true ... is a SMALLLER than m ...**

elif (len(a\_stories) < len(m\_stories)):

print ('\n\tmodel\_a has less stories than model\_b')

## Example 7b: Compare the storey ELEVATIONS in different models

We use a **while** loop here which iterates through the len of a\_stories. This could cause a problem if there are less b\_stories than a\_stories. So we use the **try** command here, this enables us to try a piece of code, and if it doesn’t work it will trigger **except**, and enable us to define an error message to help us debug our program or provide feedback to the user.

import ifcopenshell

# this gets the architectural model

a\_ifc = ifcopenshell.open("model/Duplex\_A\_20110907\_optimized.ifc")

# this gets the mechanical model

m\_ifc = ifcopenshell.open("model/Duplex\_M\_20111024\_optimized.ifc")

# Ok so above we loaded 2 models...

a\_stories = a\_ifc.by\_type('IfcBuildingStorey')

m\_stories = m\_ifc.by\_type('IfcBuildingStorey')

# we are creating a counter here to help us iterate through the stories

count = 0

# we use a while loop here

while count < len(a\_stories):

# try is really cool and stops your code blowing up!

try:

print ('for {} level: a is at {:04.2f} m, and b is at {:04.2f} m'.format(count,a\_stories[count].Elevation,m\_stories[count].Elevation))

# each time this loop runs it adds one to count and gets

# closer to the len of m\_stories

except:

print ('something went wrong')

count+= 1

## Example 7c: Are the ELEVATIONS in diff models the same?

This example is a combination of the logic in 7a and 7b

import ifcopenshell

# this gets the architectural model

a\_ifc = ifcopenshell.open("model/Duplex\_A\_20110907\_optimized.ifc")

# this gets the mechanical model

m\_ifc = ifcopenshell.open("model/Duplex\_M\_20111024\_optimized.ifc")

# Ok so above we loaded 2 models...

a\_stories = a\_ifc.by\_type('IfcBuildingStorey')

m\_stories = m\_ifc.by\_type('IfcBuildingStorey')

# we are creating a counter here to help us iterate through the stories

count = 0

# we use a while loop here

while count < len(a\_stories):

# try is really cool and stops your code blowing up!

try:

# here we grab the elevations and also round them up to 3 decimal places

a\_elev= round(a\_stories[count].Elevation,3)

m\_elev= round(m\_stories[count].Elevation,3)

# we can then print the elevations as we did in example 7A

print ('\n[{} level] a : {:04.3f}m | b : {:04.3f}m'.format(count,a\_elev,m\_elev))

# test if the rounded values are equal

if (a\_elev == m\_elev):

print ('- PASS these match!')

else:

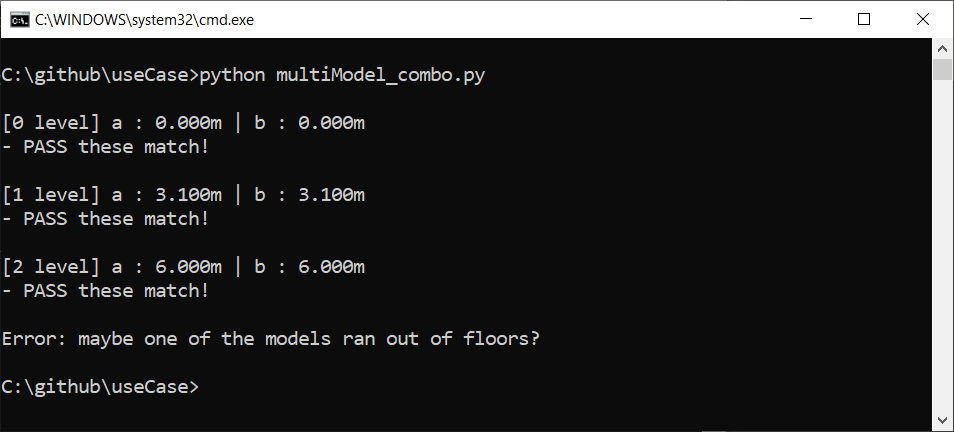
print ('- FAIL these do not match')

except:

print ('\nError: maybe one of the models ran out of floors?')

count+= 1

Example output from 7c - does this match what you got? - why does it error? is the answer in 7a?!?



## Example 8a: Property check

Descirption and comments to follow

import ifcopenshell

model = ifcopenshell.open("model\Duplex\_A\_20110907.ifc")

for entity in model.by\_type("IfcFooting"):

ele\_at\_bottom = False

#we need to get the attributes

for relDefinesByProperties in entity.IsDefinedBy:

for prop in relDefinesByProperties.RelatingPropertyDefinition.HasProperties:

#and then get the attribute we are looking for

if prop.Name == 'Elevation at Bottom':

#add the length to the total length

ele\_at\_bottom = True

if (ele\_at\_bottom):

print ('[X] {}'.format(entity.Name))

else:

print ('[ ] {}'.format(entity.Name))

## Example 8b: Generic Property list

OK so this is my best bet its not perfect but you should be able to adapt it to your needs and it works for windows and doors 😊

import ifcopenshell

model = ifcopenshell.open("model\Duplex\_A\_20110907.ifc")

print ('\n## search all properties of a type ##\n')

# lets search all property sets

for pset in model.by\_type("IfcPropertySet"):

# and all single values within those property sets (that have props)

for prop in pset.HasProperties:

# and check if that property matches the one we are looking for...

if prop.is\_a("IfcPropertySingleValue"):

obj = model.get\_inverse(pset)

# ok cool so now get the related objects

# so then we get the entity related to that property

for part in obj[0].RelatedObjects:

# then we check if the entity is a window

if (part.is\_a('IfcWindow')):

# print the property sets that you have found

print ('{} : {}'.format(prop.Name,prop.NominalValue.wrappedValue))

## Example 8c: Find entities based on a singlevalue property

I think this one is pretty cool, I was trying to write an example for 8b and ended up with this. The logic is that we define the property and value we are looking for , get it and then seeks its inverse entities.

import ifcopenshell

model = ifcopenshell.open(" model\Duplex\_A\_20110907.ifc")

propName ='Unbounded Height'

value= 2.6

print ('\n## search entity by property and value ##\n')

print ('Searching for...')

print ('\tIfcPropertySingleValue = [ {} : {} ]'.format(propName,value))

# lets search all property sets

count = 0

print ('Results:')

for pset in model.by\_type("IfcPropertySet"):

# and all single values within those property sets (that have props)

for prop in pset.HasProperties:

# and check if that property matches the one we are looking for...

if prop.is\_a("IfcPropertySingleValue") and prop.Name == propName and prop.NominalValue.wrappedValue == value:

# then get the objects inverse relations

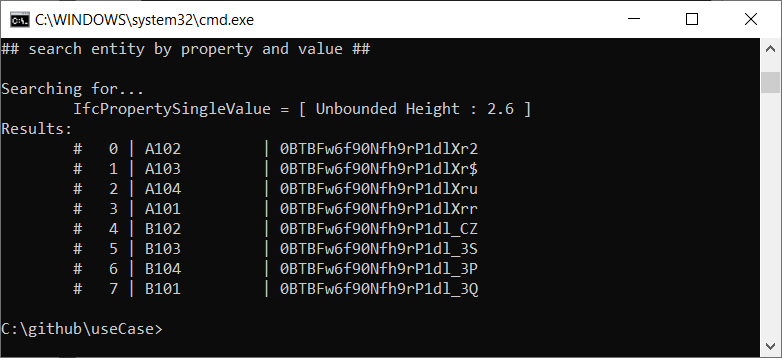
obj = model.get\_inverse(pset)

# ok cool so now get the related object …

for part in obj[0].RelatedObjects:

print('\t# {:3} | {:12} | {}'.format(count,part.Name,part.GlobalId))

count+=1



## 

## Example 9a: Door code check

This is an edit of Kallina’s door code check, its a good example.

import ifcopenshell

model = ifcopenshell.open("model\Duplex\_A\_20110907.ifc")

###Doors###

doors\_required = 14 ### <- Expected value of doors ###

doors\_in\_model = len(model.by\_type("IfcDoor"))

min\_width\_door = 0.77

valid\_doors=0

invalid\_doors=0

print ('\n')

# initial check to establish if we have the 'correct' number of doors

if doors\_required == doors\_in\_model:

print("Result matches expected value ({})".format(doors\_required))

elif doors\_required > doors\_in\_model:

print("There are more doors than expected")

elif doors\_required < doors\_in\_model:

print("There are less doors than expected")

print ('\n')

# check each door to see if it complies and count the valid ones

for door in model.by\_type("IfcDoor"):

print ("door with width: "+str(door.OverallWidth))

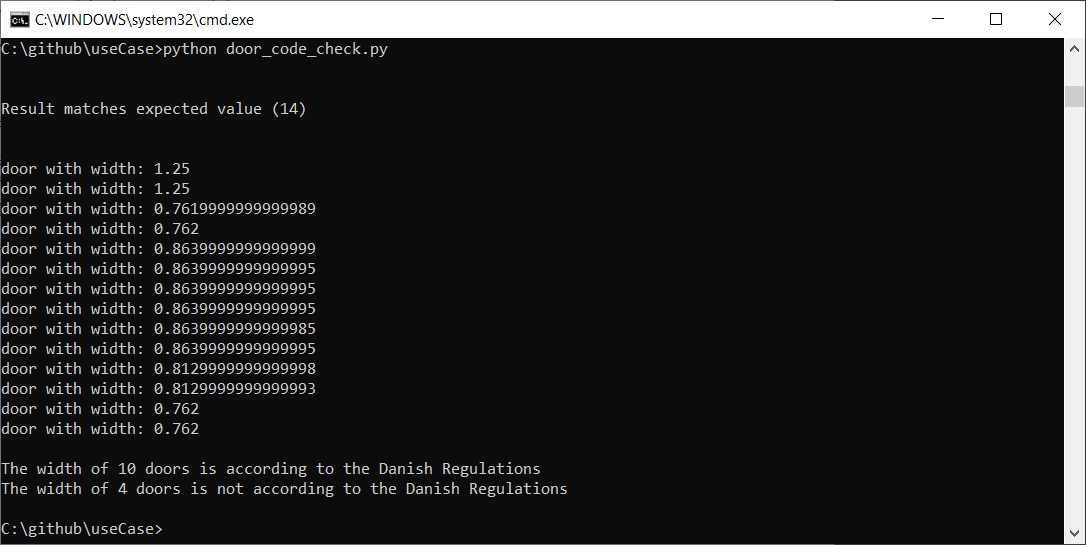
if door.OverallWidth>=min\_width\_door:

valid\_doors+=1

# now we have finished the counting we can pull back the indents and print the result

print("\nThe width of {} doors is according to the Danish Regulations".format(valid\_doors))

print("The width of {} doors is not according to the Danish Regulations".format(doors\_in\_model-valid\_doors))



# APPENDIX G: Glossary of terms (Work in progress)

**Format** - we use this to make the output statements pretty and reduce the amount of logic that we need to include in the print statement -> check it out here <https://pyformat.info/>

*Example*

print('-\t{}'.format(obj.Name))

**Input** - Get more information -> <https://www.w3schools.com/python/ref_func_input.asp>

*Example*

print

Function - sometimes we might have to define a function for code that we will reuse in our script for instance <https://www.tutorialspoint.com/python/python_functions.htm>