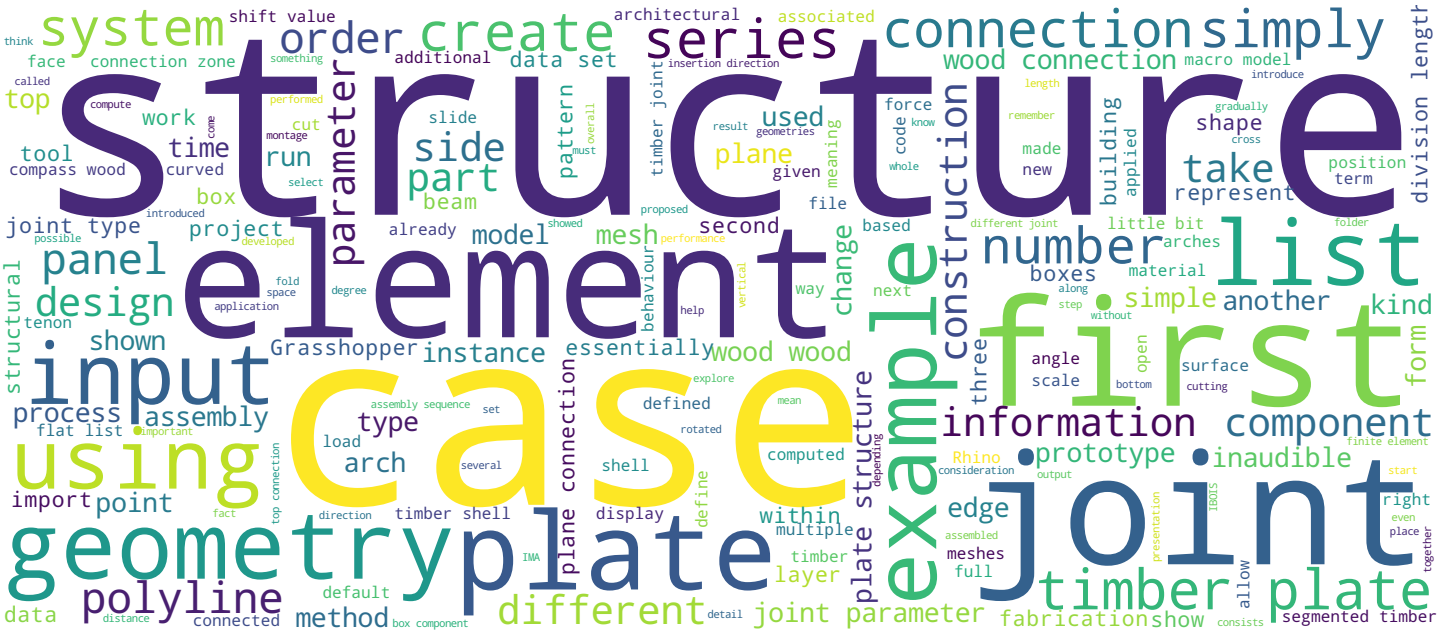
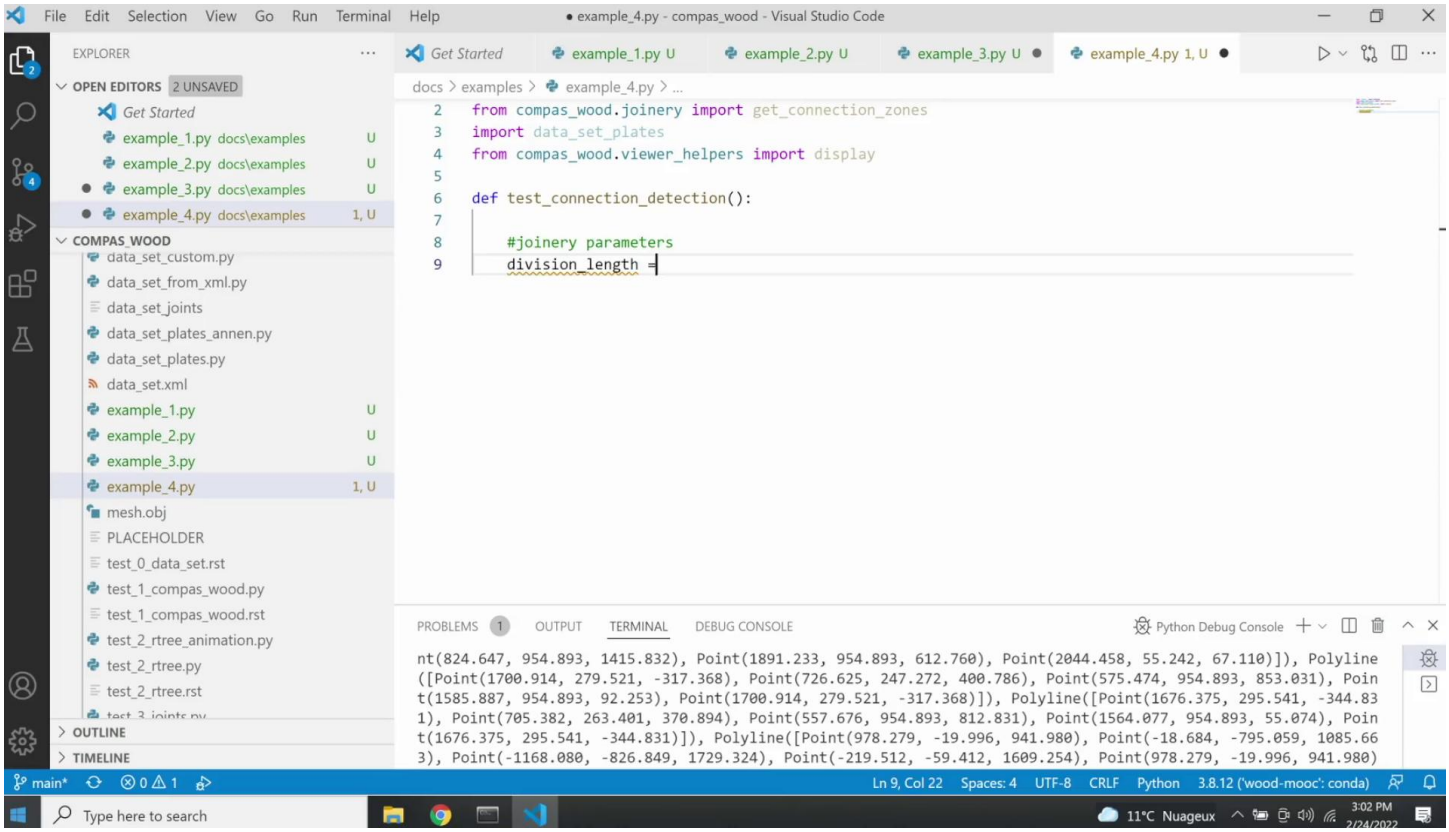

EPFL
IBOIS
Double-layered
Timber Plate Shell
Tutorial Python Part. 2
Petras Vestartas, Ph.D.


structure
element
case
joint
example
first
input
using
geometry
plate
timber plate
system
order
create
series
connection
simply
parameter
construction
number
list
panel
design
information
component
prototype
inaudible
polyline
different
joint parameter
fabrication
show
segmented timber
Search MOOC
Video
EPFL

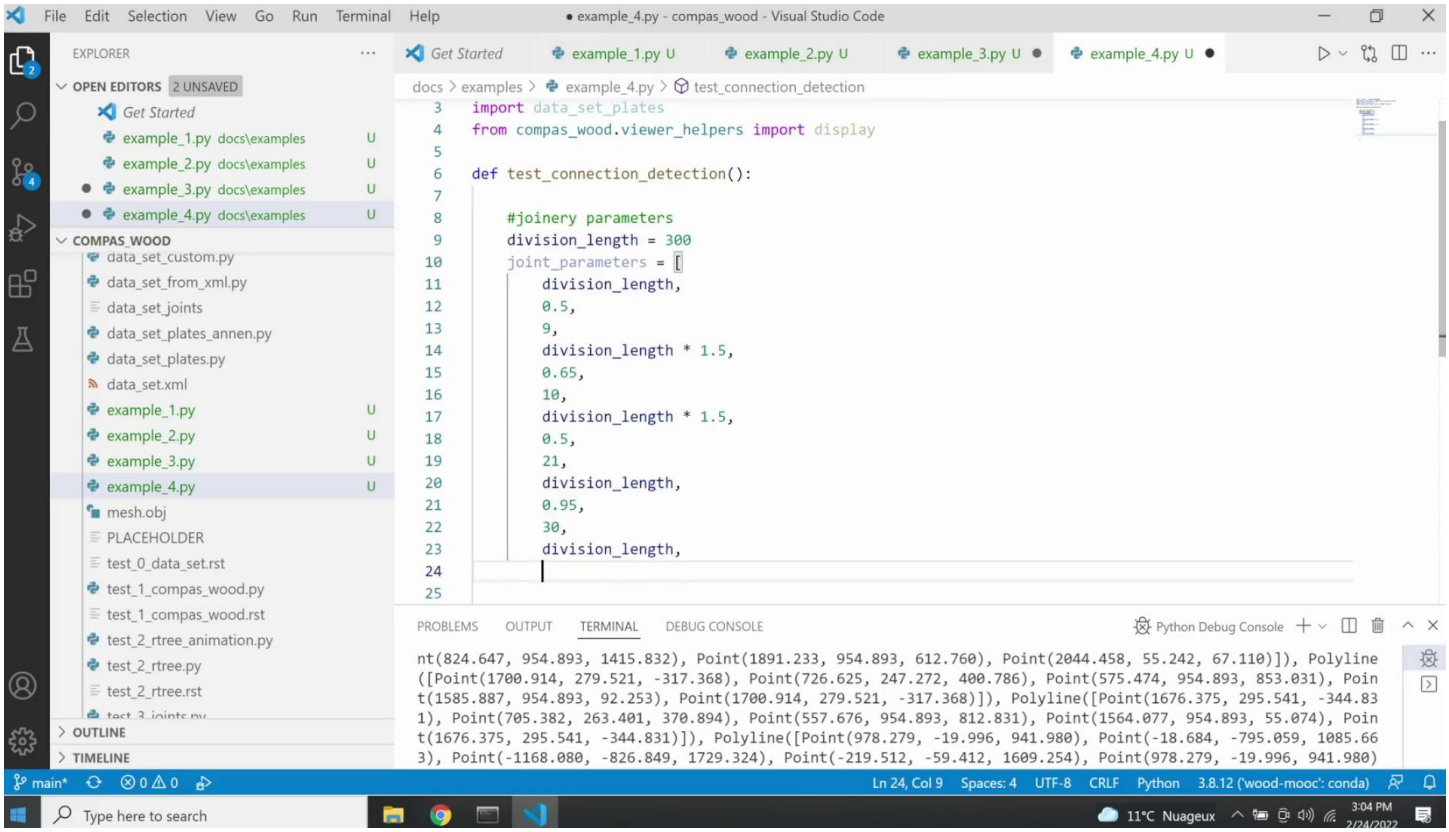


The last example that I'm going to show today is an actual timber joint generation for segmented timber shell. I will create the fourth example. Say example four [inaudible 00:00:19]. Now we'll run a bit more longer example but essentially it will help to create timber joints. The procedure is the same. We import compass wood joinery, then we import get connection zones method. That is the main method for generating joints. You're going to import the data set of plates and then as before, we are going to import the compass [inaudible 00:00:49] compass wood, viewer helpers, import display. So these are the imports and then we are going to define a method called the test connection detection. Now we need to do a few things. We need to first get the inputs and then gradually run it in the get connection zone method. So the very first thing we need to have the joint parameters. So joinery parameters and it is kind of the same as was in Grasshopper. I don't know if it's the very best design decision but simply I hard code every single joint parameter that is possible to detect in this joint resolver. So let's say the division length of the joint will be around 300 units.

Notes

Summary

0m 04s

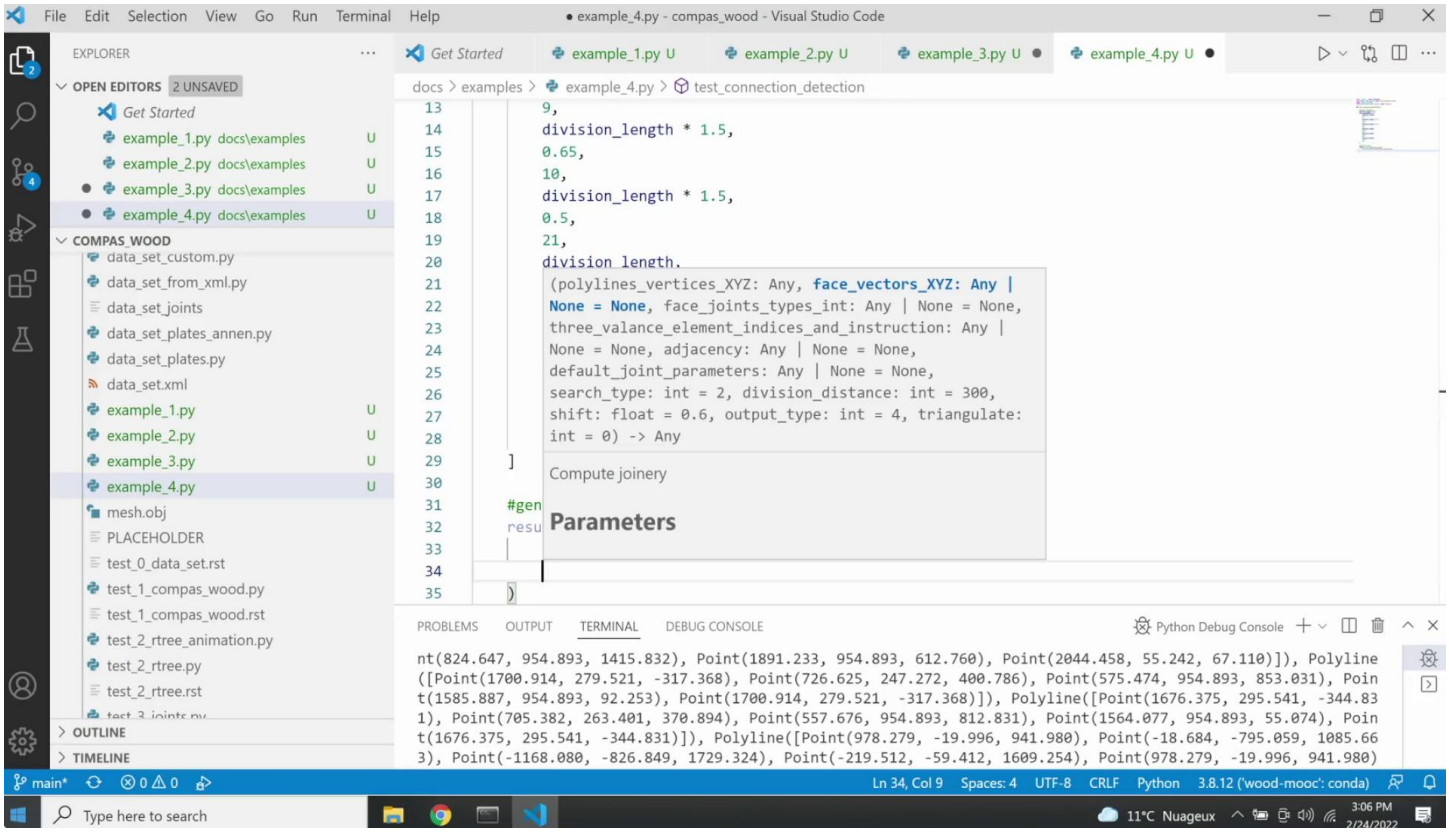


And now we create a joint parameters list that actually have a list of numbers that represents different joint parameters. The first one is usually division length as it is in Grasshopper. Then it is the shift value. Then there is the index of the tile joint in the tile dictionary. So this is for the in-plane joint. Then we have a division length for out-of-plane joint which let's say would be a little bit longer. Then it has a shift value of 0.65 because we are going to represent the dovetail joint and the type of that joint would be number 10. And then we have a out-of-plane connection which is [inaudible 00:02:41] connection that is also, let's say, a little bit bigger and it's not going to have any shift value. The tile name for the anon is 21, [inaudible 00:02:58] joint, a simple one is 20 and for anon case, it's 21 because it's a special joint type. And then for the rest, we simply can leave it as a default. It does not really matter because in this example the loose joints will not really be computed. But let's fill the list of the joint parameters. Say 0.95, 40, then division length again, 0.95, and 50.

Notes

Summary

1m 50s

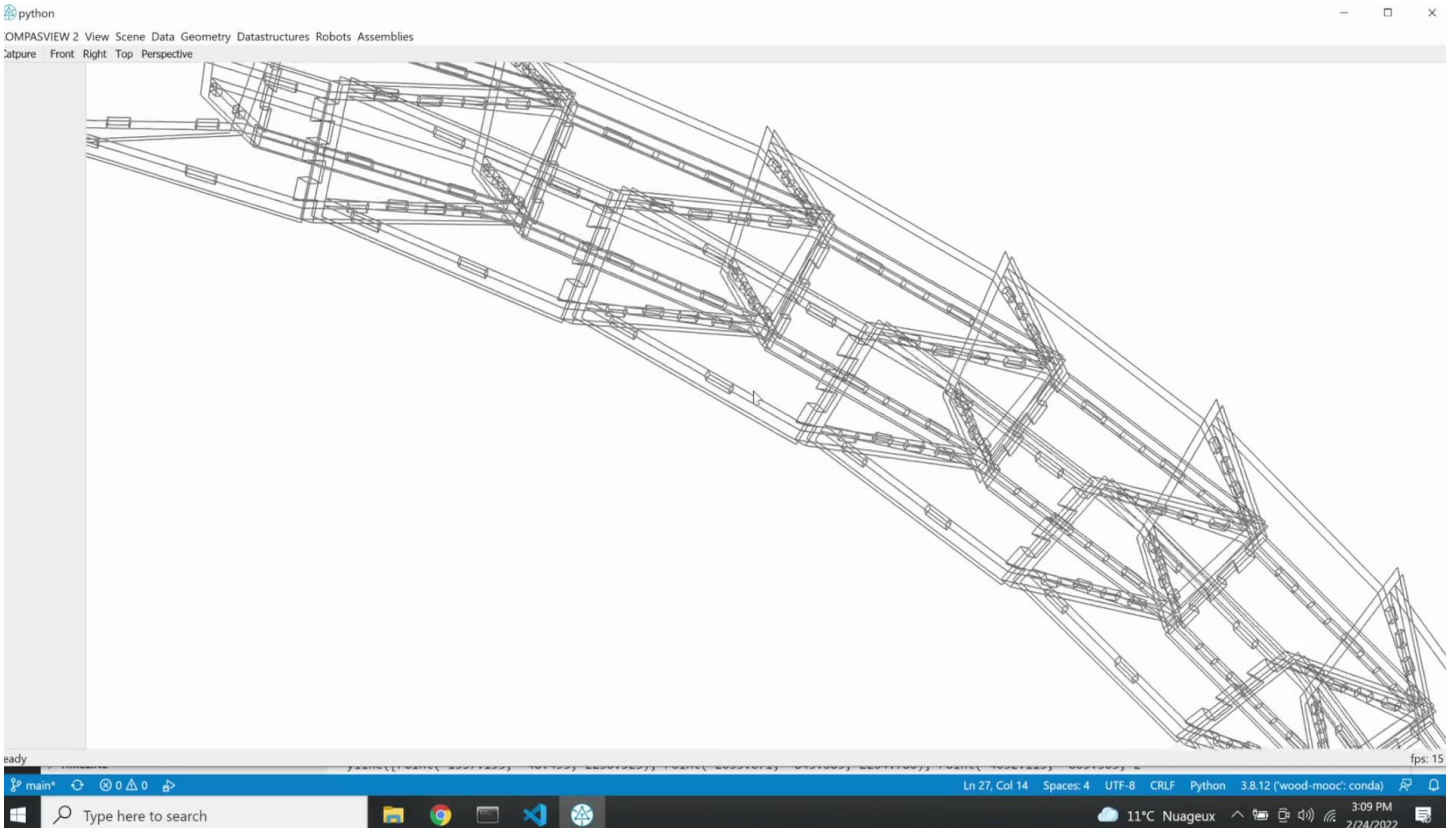


So these were the number of parameters essentially for in-plane connections, out-of-plane connections, side-to-top connections, and then cases for top-to-top connection which I think is here and then also rotation case when elements are rotated here. In case you are not aware of these parameters, you can go to the description and actually this is the list of joints that are currently available. So this is the joint type and these are different joints that actually available in the joint resolver. So there are multiple joints. You can also implement your own example as well. It's not necessarily limited to these number of examples and these are kind of generic parameters that I'm using it for controlling the joints. So now since we have the parameters, we can actually generate joints. Now we can simply run this method and this time, we'll have a bit more parameters as we saw before, call this connection zone method and it requires a series of inputs. The first one is the body line input that we can find in data sets, say anon, small polylines which represents a series of body line pairs. Then as you remember before, we need to have insertion directions, so we need to have these hard coded insertion directions.

Notes

Summary

3m 37s

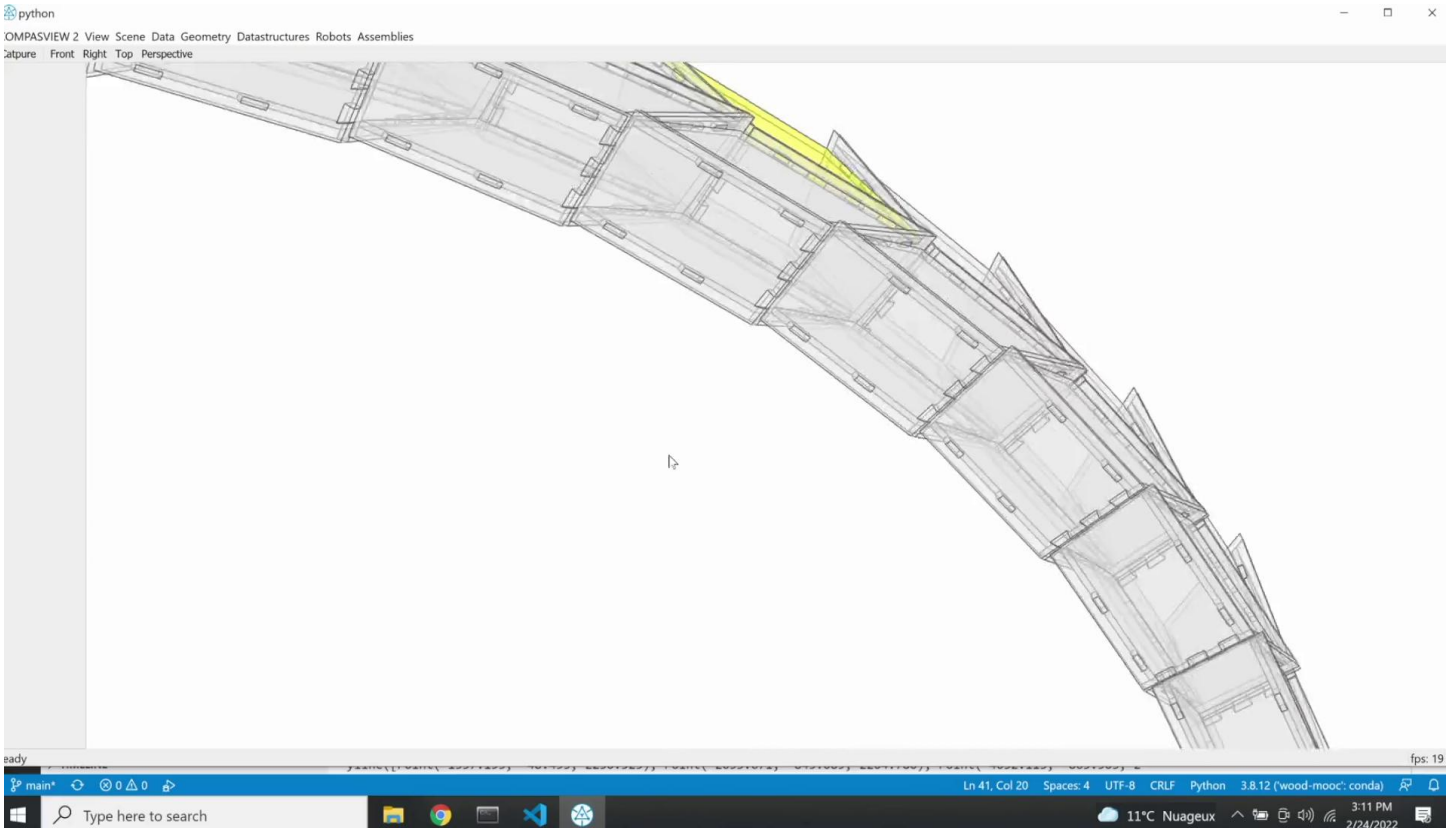


Then we need to specify the joint types for each plate. So on an edge joints and then lastly, as you remember, we need to add the free balance connection which is the special case for the presentation I showed before. You can also not use air tree, you can hard code adjacency if you have any. In this case I will have nothing and then I input the joint parameters which was defined before. So this is the full input. Now what we need to do is we simply need to display the information. So first, let's flatten a list of polylines for display case. So let's say result and flat list, let's write a small end-of-case how to unflatten the sub list. This is how you create [inaudible 00:06:58] the list of lists to one single flat list. Then we gradually display the geometry. Let's say display, result flat list and these are some, again, default parameters that were existing before. Yes, and now we can simply call this function test connection detection. This one simple sample. If there are no really big mistakes in typing this sample, maybe I'll decrease the size of the code and let's try to run it to see if it works. Okay, you see the joints generated, so. This is the case of the timber joiner.

Notes

Summary

5m 35s

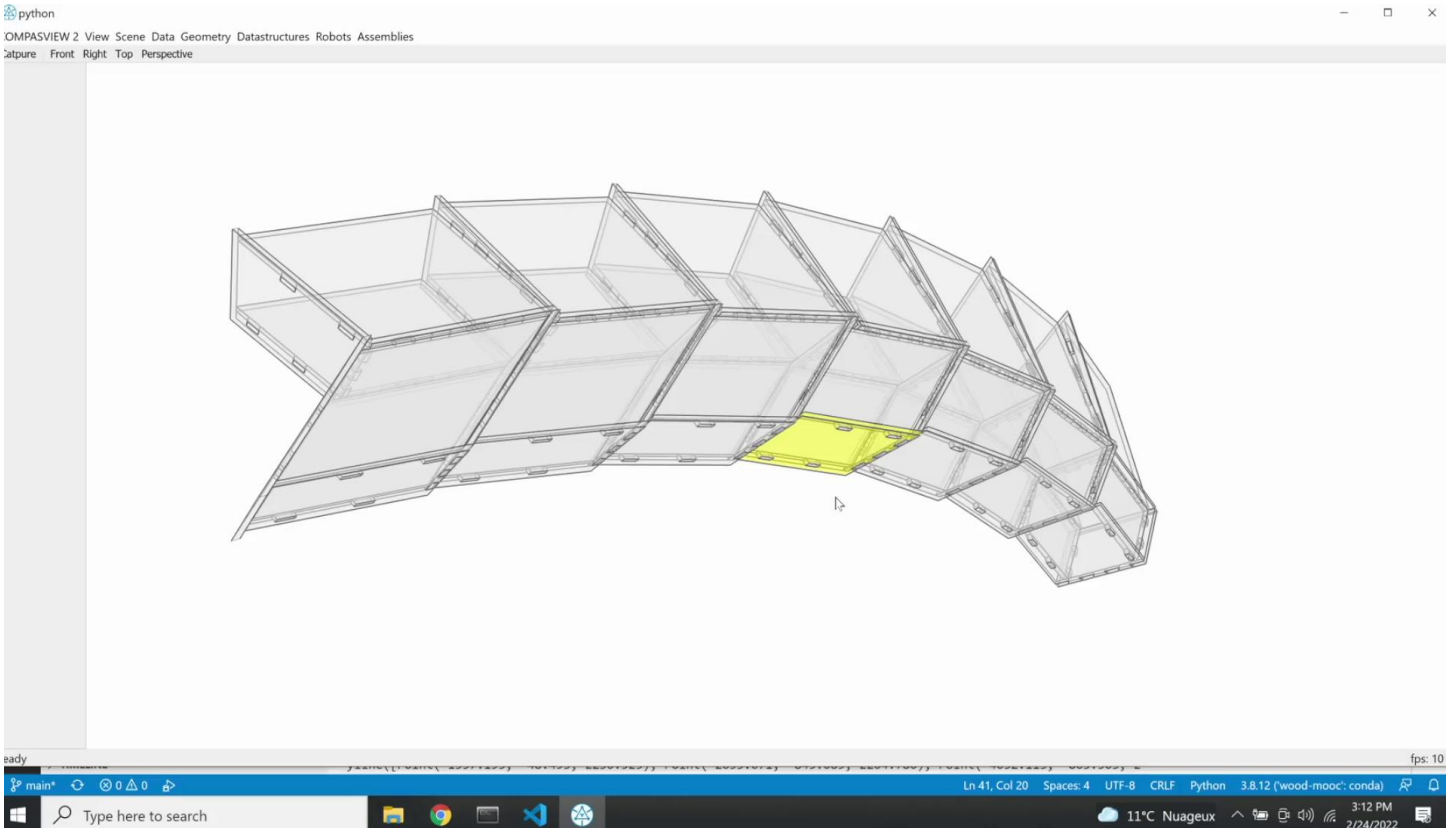


And one last step that you would like maybe to have is how to measure these geometries because these geometry sign general polylines for the holes and there's no really specific method to do this in this, and in order to do that, we can simply display the meshes. So let's create that snippet to do that. So let's mesh polylines and say meshes equals a list and then for i in range and then results, so it's like that. Let's get a length of total numbers of polylines and let's try to mesh it. So mesh result closed close mesh from polylines which again comes from compass woods and then the results [inaudible 00:08:59]. Can be the case is that the mesh is invalid because compass viewers likes only the valid well-oriented meshes. So sometimes you need to check that as well in case your geometry can be wrong. Let's create a mesh like this and then input the mesh list into the free parameter, represents a list of meshes. So let's try to run it and you see that right now, all these meshes are displayed. I can actually select and see that the mesh geometry taken into account and these convex, concave geometries, including the holes inside and other properties.

Notes

Summary

7m 54s



You can explore other examples, as was shown in the beginning, but this is the last recording of the first session, which finalises the examples in Rhino and Grassropper environment, but also finalises the Python example, how to create a segmented timber shell from theoretical geometrical description to practical assignments.

Notes

Summary

10m 00s