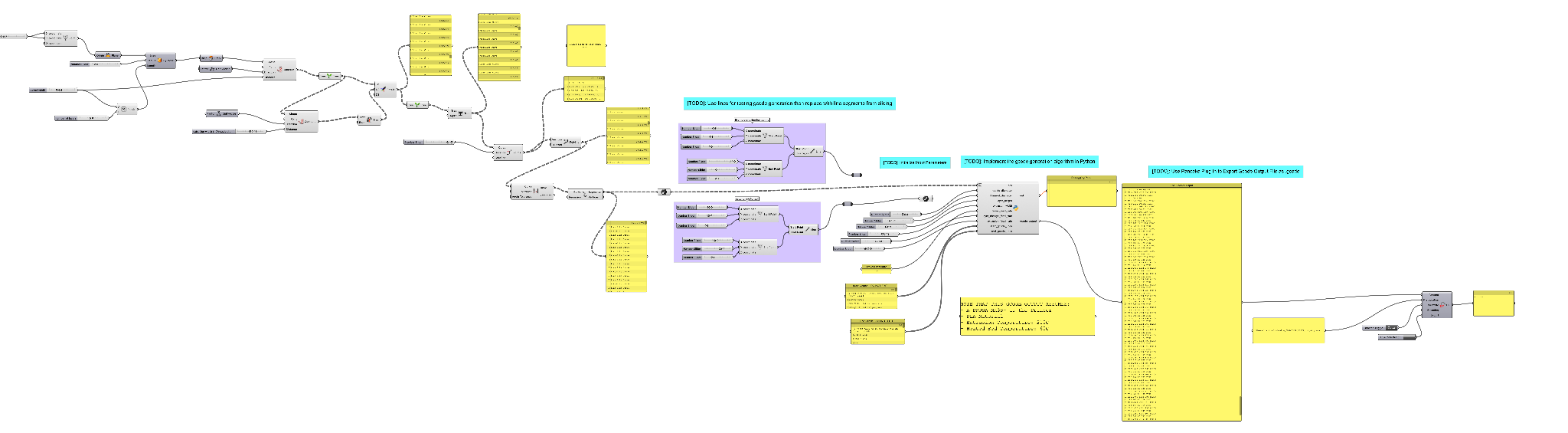
L3 — Part One [L3.gh](https://drive.google.com/file/d/1p7Kwy6Zg7T--5IBN93NTdid5pX4jXcA1/view?usp=drive_link)

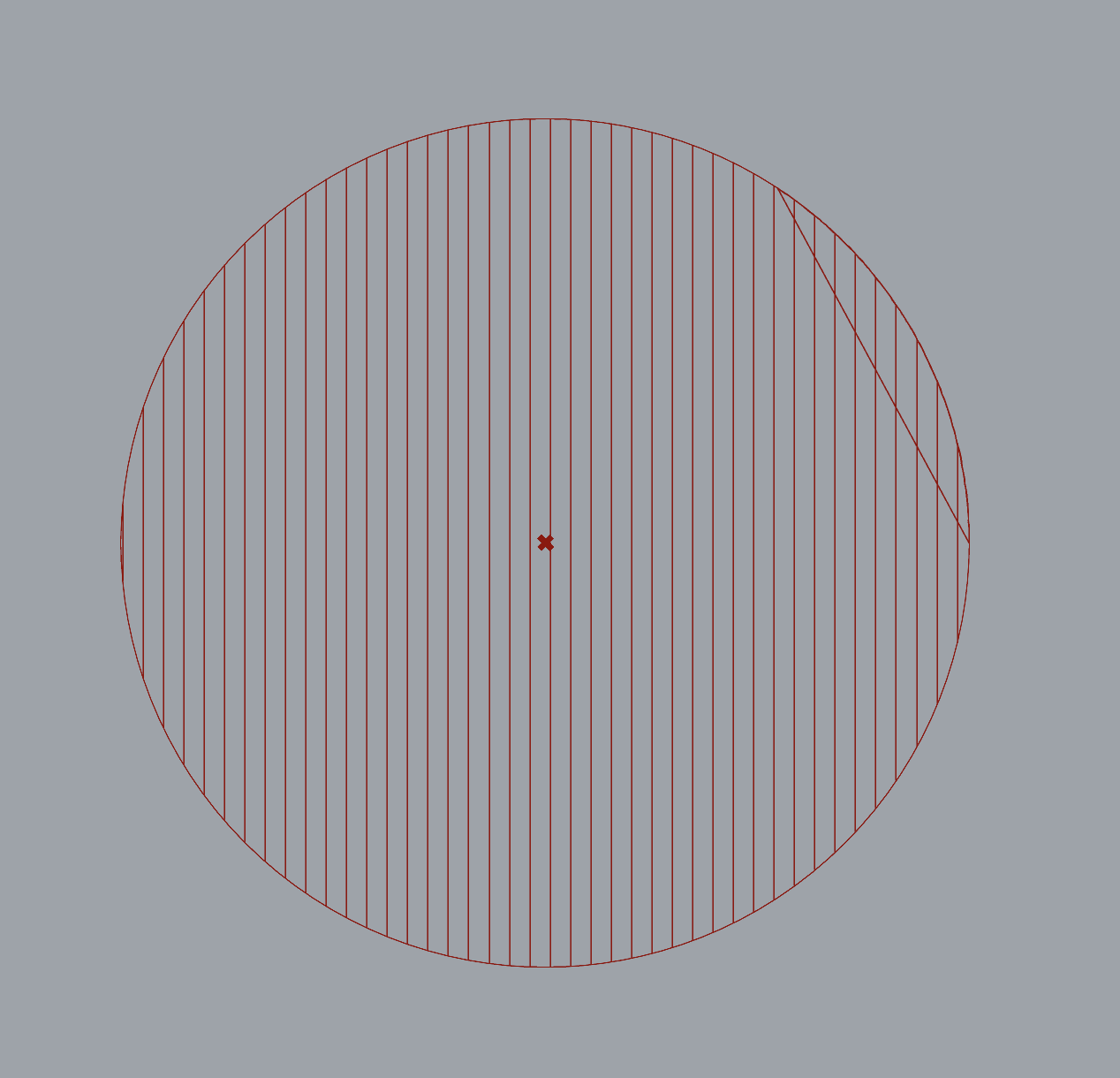
The third and final large assignment centered around 3D printers and g-code. In part one, we were tasked with creating a basic, rectilinear path generator in Grasshopper to then be translated into g-code based on a Python script nested within the Grasshopper file. The goal thereafter, in part two, is to then use our g-code to 3D print a basic cylinder. There were a few ways to approach the solution to this problem. While you could use lines, I chose to approach it using contours, and although it might be a more simple method, it is not the most straightforward.



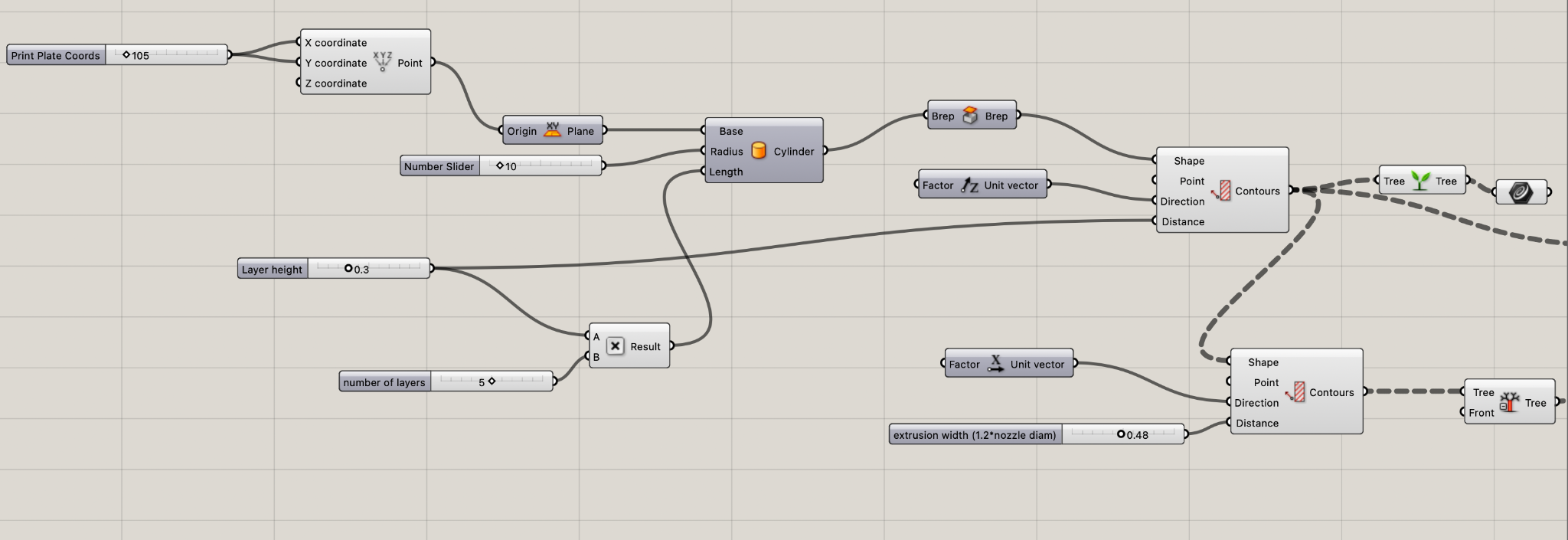
Math Used

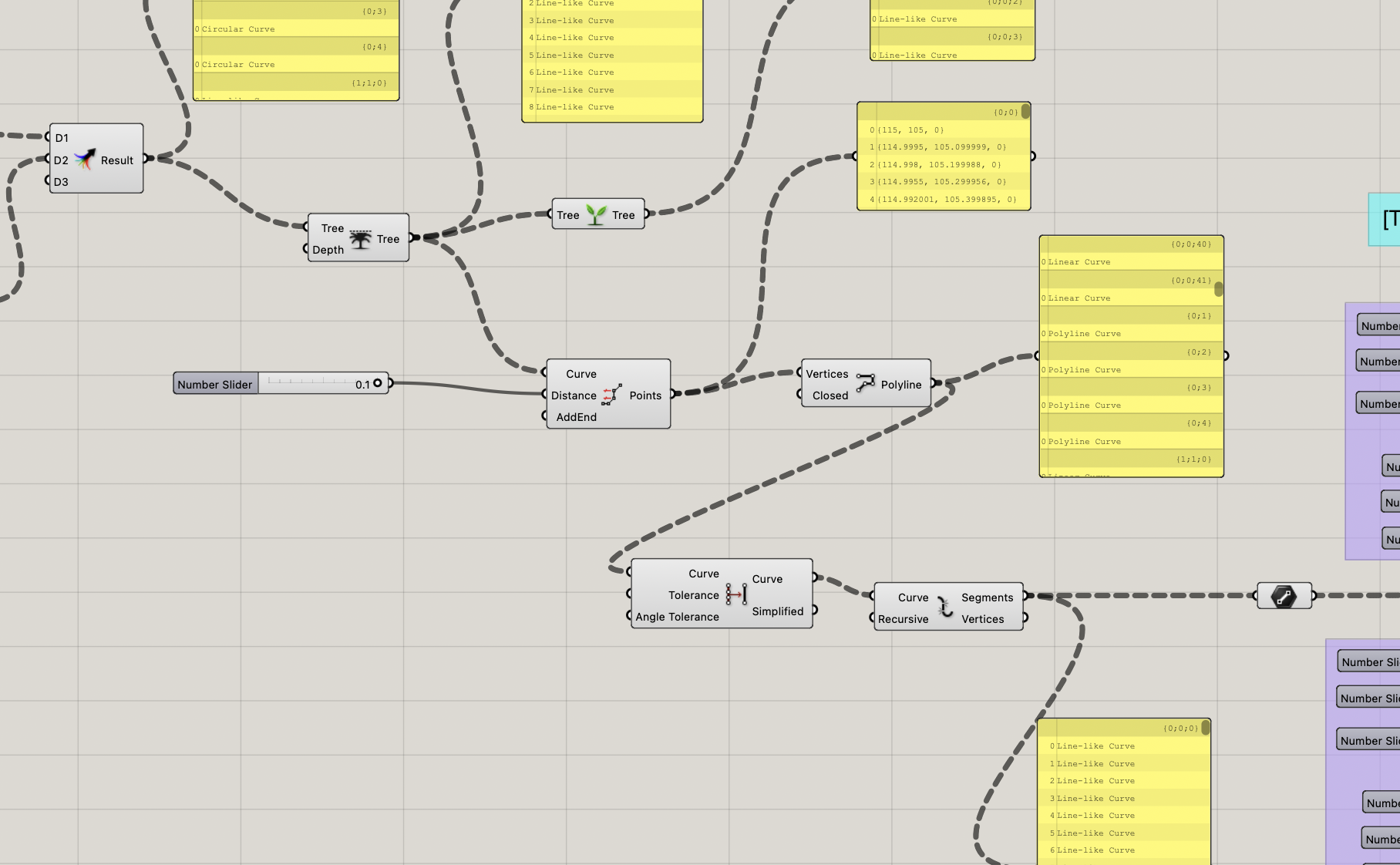
* Geometry (areas, volumes, centroids…)
* Cartesian coordinates (points, lines…)

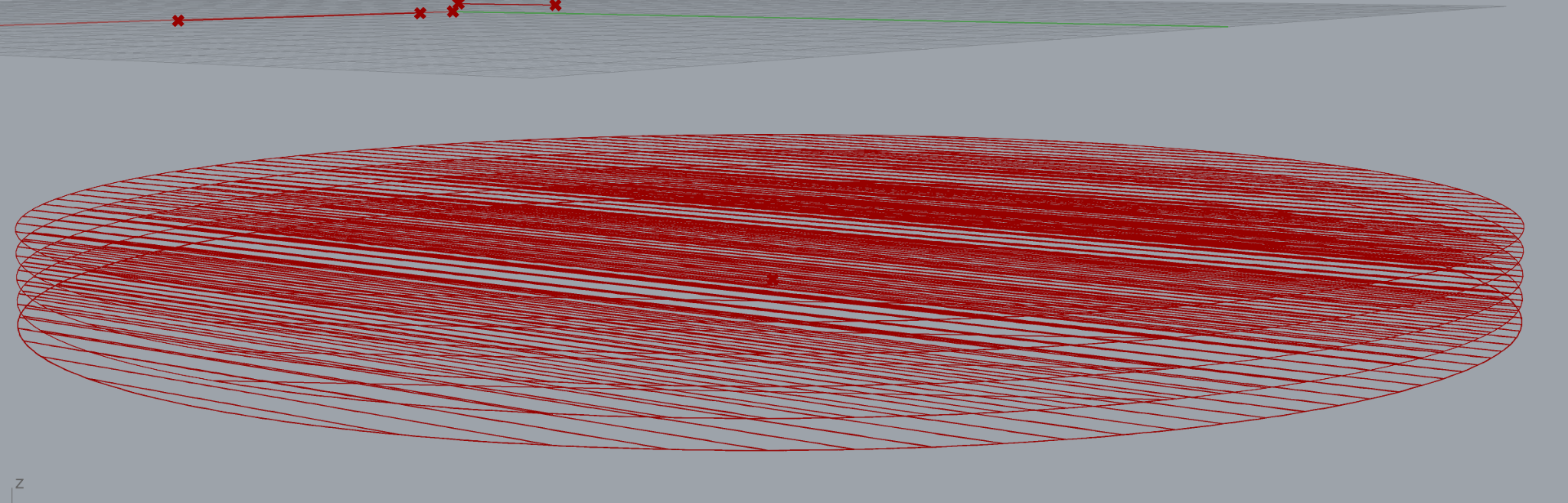
Task #1



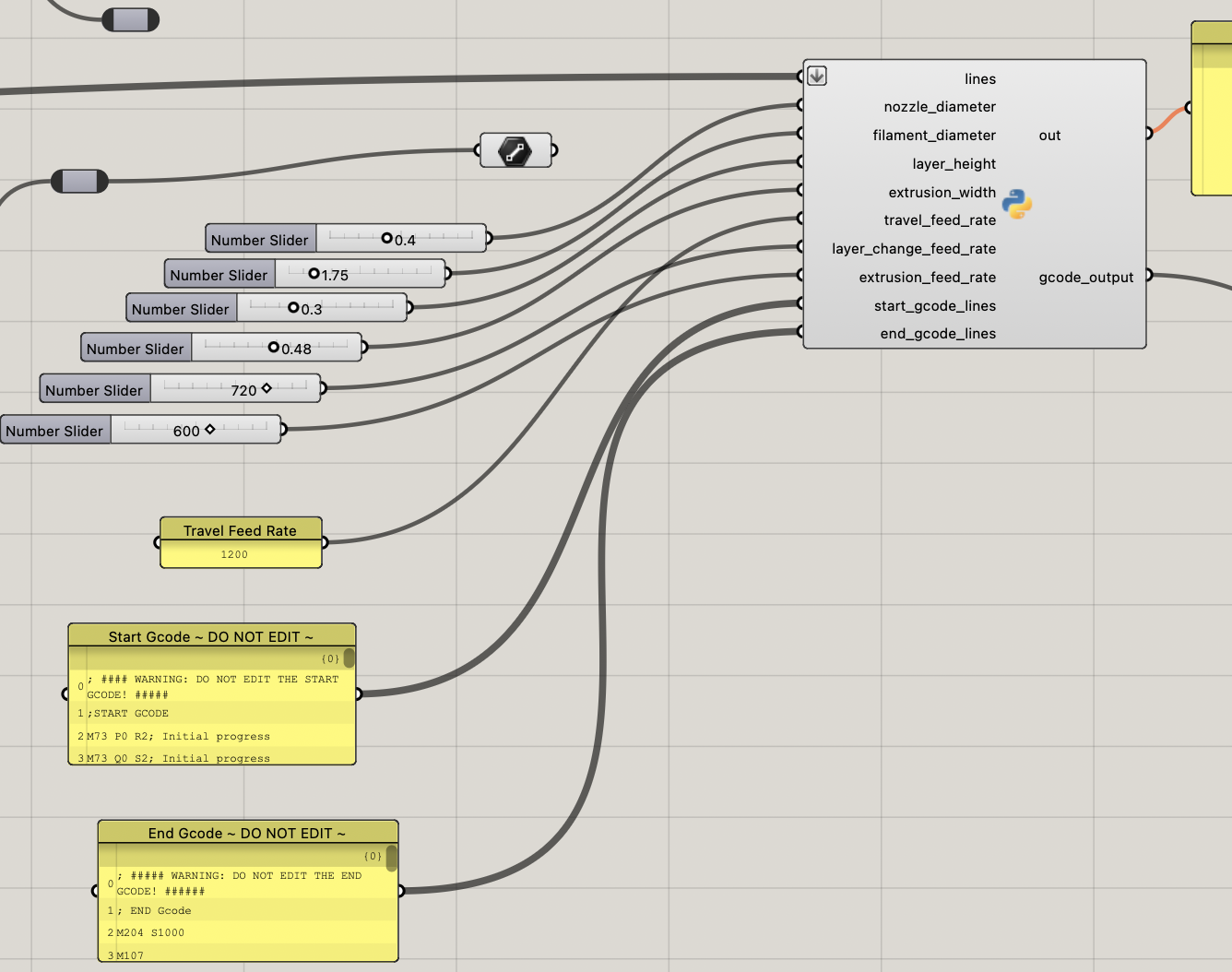
Task one appeared straightforward, but would ultimately leave me in the weeds for quite some time. The goal: build a g-code slicer for a basic cylinder with a rectilinear infill path. While line generation and bounding boxes are an option, contours made the most sense in my head. I ultimately ended up taking the contours of the cylinder on the XY-plane an extrusion layer-height apart, contouring each of those individually in the x-direction an extrusion-width apart. From there, I merged the datastreams, trimmed the tree, divided the distance, and created the polyline and segments pretty much as laid out (after much struggle explained in problems/conclusion…). Ultimately, however, to my utter dismay, I could not seem to figure out how to properly index the classes, generating each circle before its respective infill lines. Below you will find the Grasshopper workflow for the cylinder and polyline generation, as well as both the single (above) and five layer example slices:



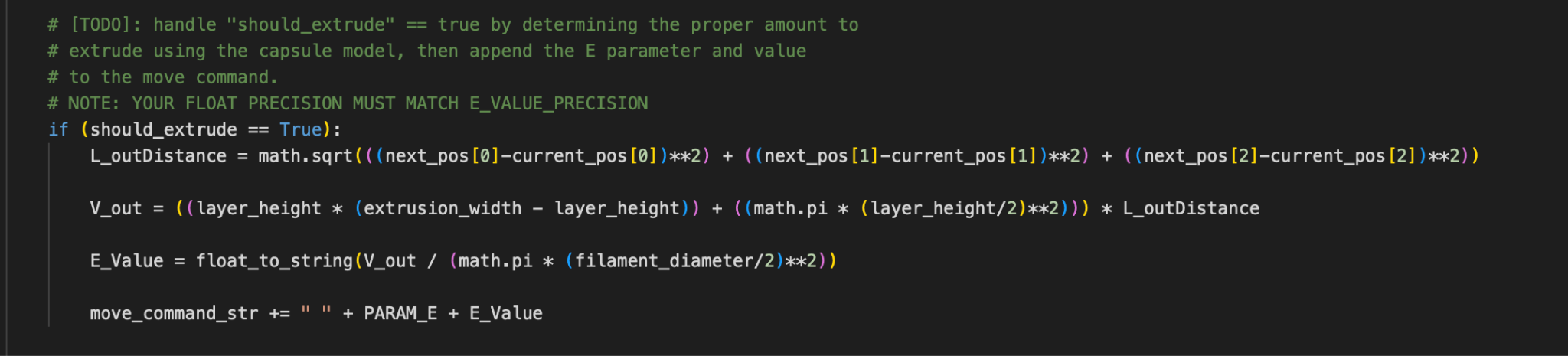


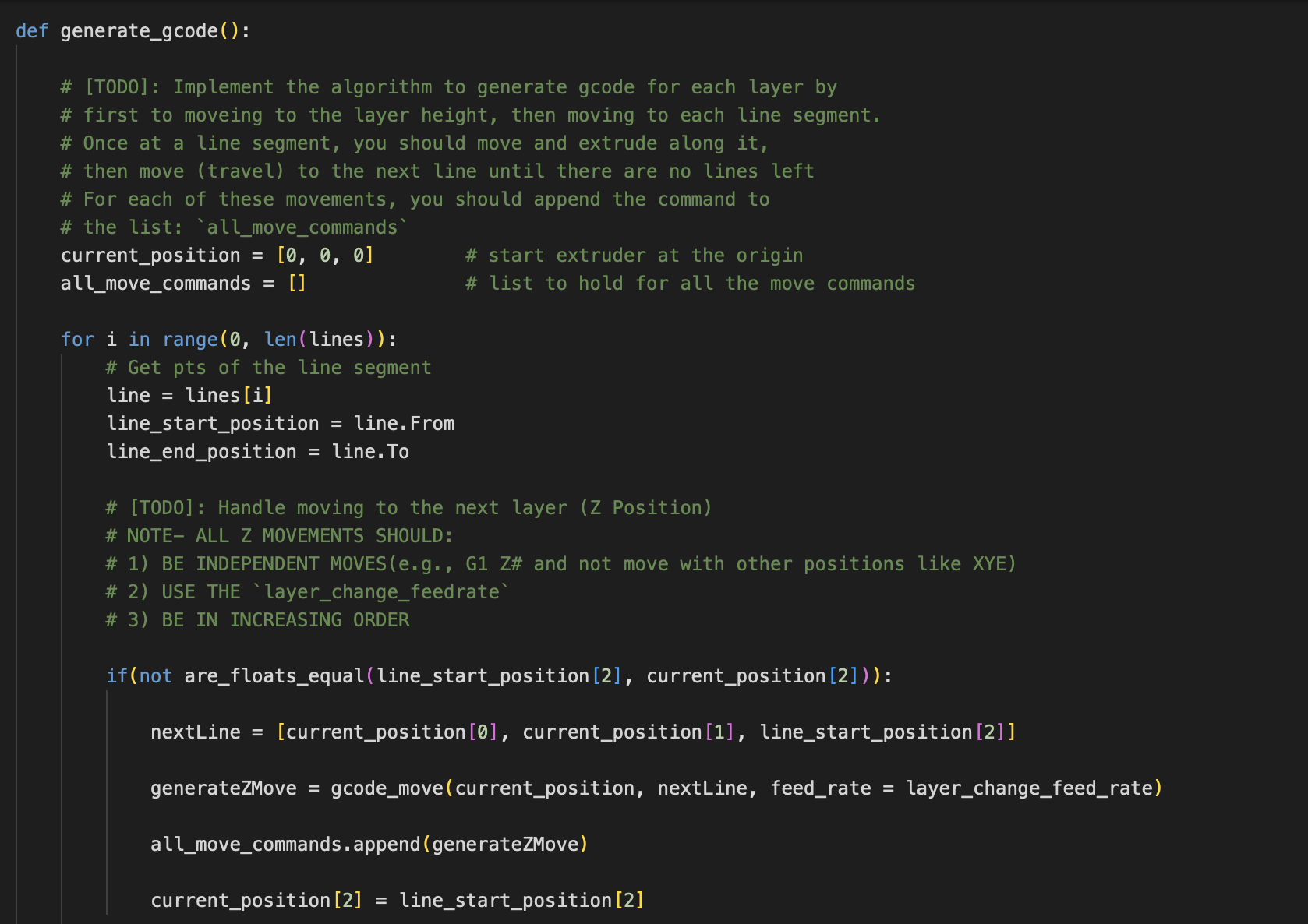


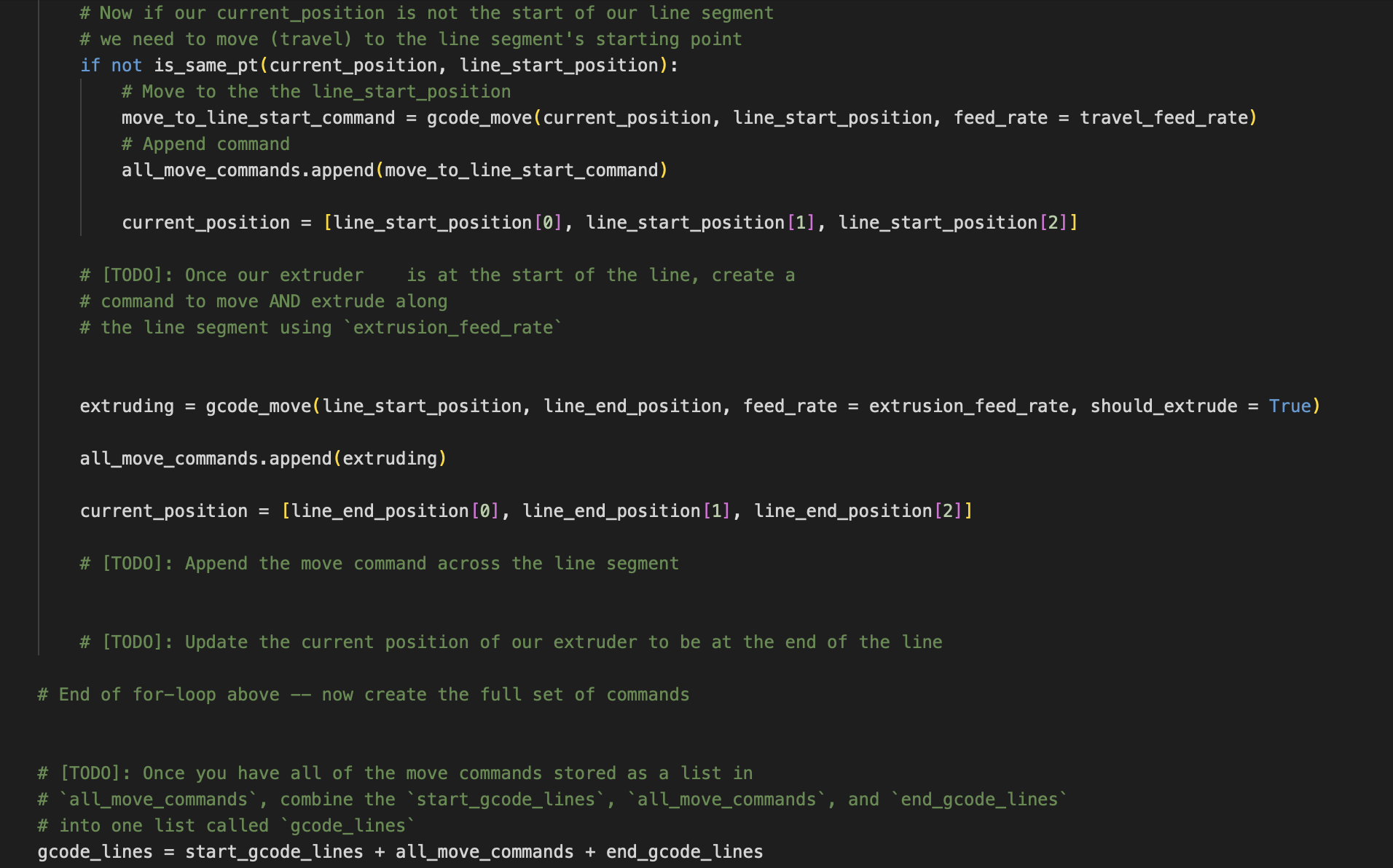
Task #2



Task two was quite simple once I wrapped my head around it. Its goal was to complete the TODOs, housed within the Python script, nested in the Grasshopper file, in order to complete the g-code generation based on the printer/extrusion parameters. First up was to complete the should\_extrude case for the gcode\_move() function. This function’s purpose was to calculate our extrusion value for the “E#” variable of the gcode. Given this was quite basic math provided on the in-class slides, this went very smoothly, and did not take long. Wrapped up with a simple append to the move command and we were finished. The next TODO was completing generate\_gcode(). Our job was to account for the positioning, in the z-direction and for travel, as well as the extrusion command. First, to account for z-movement, we compare our z-coordinate of current position and line start position. If it is different, we must start the next line. And so, we create the next line from the current position with the addition of the z-coordinate from the line start position, appending it to the list of move commands, and update the current z-coordinate to be where the new line begins. Next, we must account for our travel when the current position and the line start position are different. Here, we create the line start command movement, setting the movement from current to line start position, appending it to the list of move commands, and setting our current position to line start position so we may properly begin the new line. Finally, we had to handle the extrusion. Once the positioning was handled, I simply declared an extruding variable equal to the movement from line start to line end position using the extrusion feed rate, rather than the layer shift or travel rates of previous, and set should\_extrude to true. After appending the move command, I updated the position to line start position one final time, set the gcode\_lines in the correct order, and were complete! Below, you will find the Python script for each of the TODOs, as well as the generated gcode for the single layer and five layer cylinders:







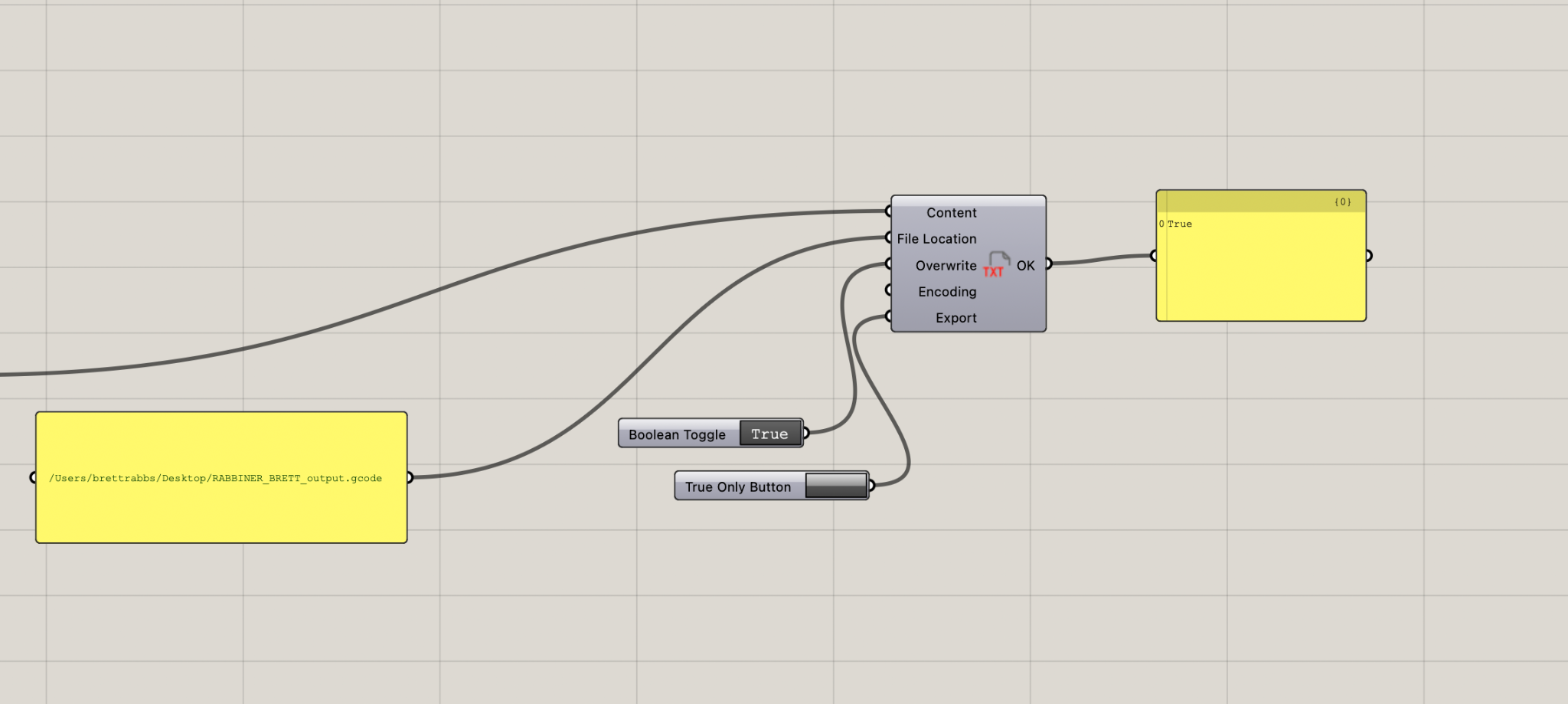
[Single Layer gcode](https://drive.google.com/file/d/1yx_yWpc3L7GoVVqq3Jr-RNTklBPc66ym/view?usp=drive_link)

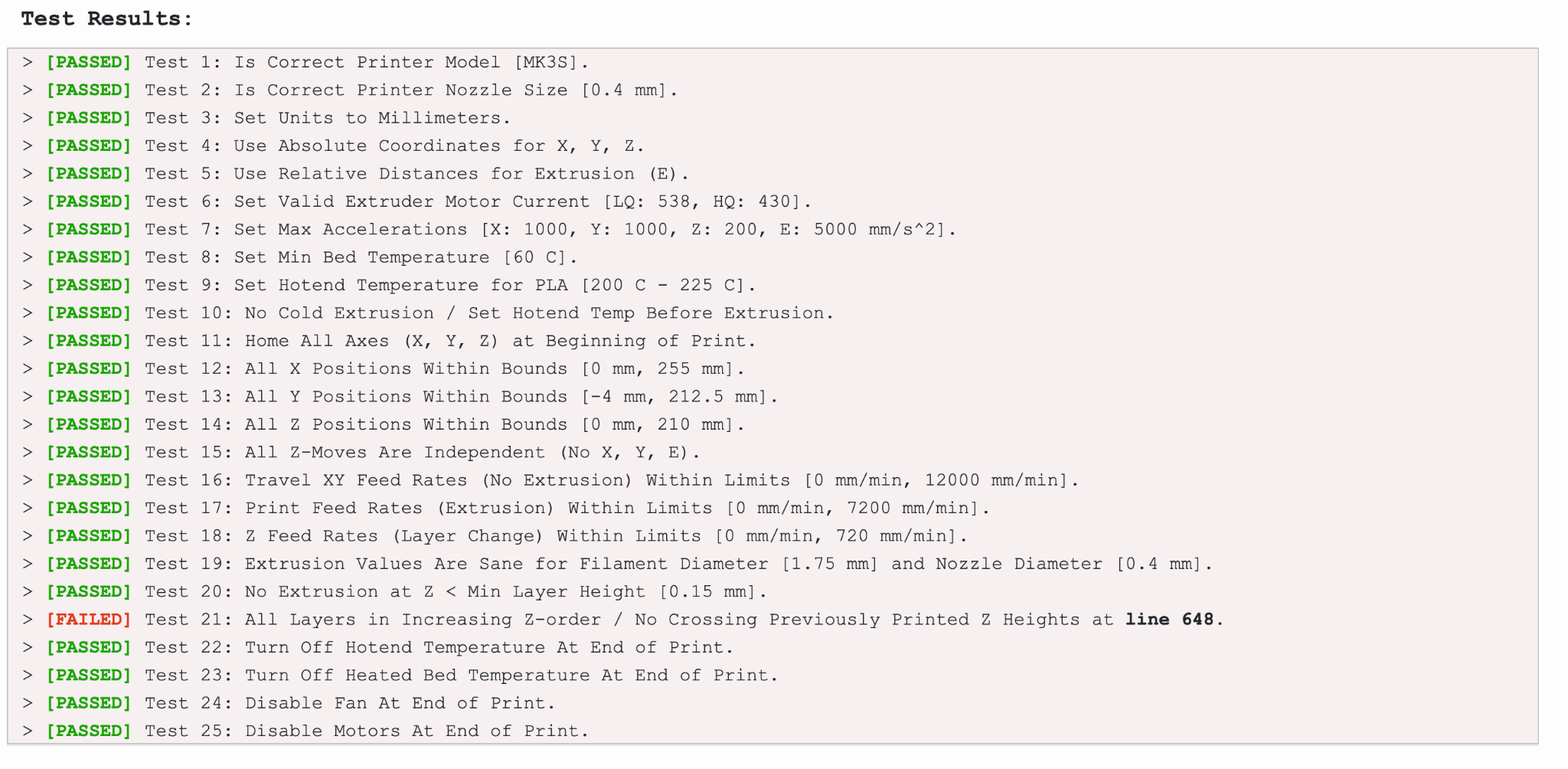
[Five Layer gcode](https://drive.google.com/file/d/1tT-0MqTFOqE57g9i0tUSQm-0nMx5HqoS/view?usp=drive_link)

Task #3

IMAGE

Task three was simple and straightforward: use Pancake, a downloadable Grasshopper plug-in, to export the gcode. After installed, and the gcode saved, we had to input our gcode into a gcode validator. Below, you will find the Pancake section of the Grasshopper file, as well as the validator test cases:





Problems/Conclusion

Overall, I have mixed feelings about part-one of this assignment. Initially, I was flying and everything was moving smoothly. I found a very easy way to generate the necessary lines using contours, thought I had a creative solution to the polyline generation, and moved decently well through the python script. However, in testing my gcode I would soon run into days worth of headaches surrounding one simple issue requiring sizable fixing. I spent countless hours going over the python script and the gh file individually. It wasn’t until I explained my issue to a friend who gave my file the quickest of glances, helping me to remember that flattening was the complete opposite of what we were supposed to do! Once this was pointed out to me, I realized my script was correct, however, my gh was not. Although I was using the contour method, I thought, *I should probably dig through the instructions more thoroughly, so as not to miss these small details*, inevitably ending my days of suffering… so I had thought. I still cannot figure out why the cylinder circumference contours do not generate just before their respective infill, but I have spent way too much time trying to figure this out, and am at least quite confident in my Python script…