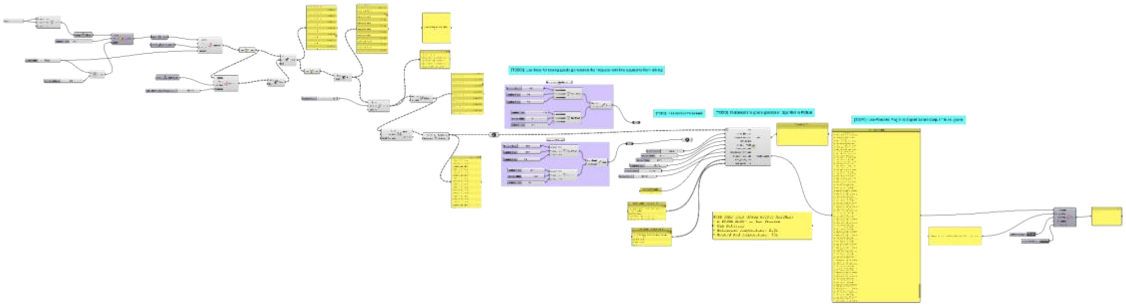
L3 — Part One

L3.gh

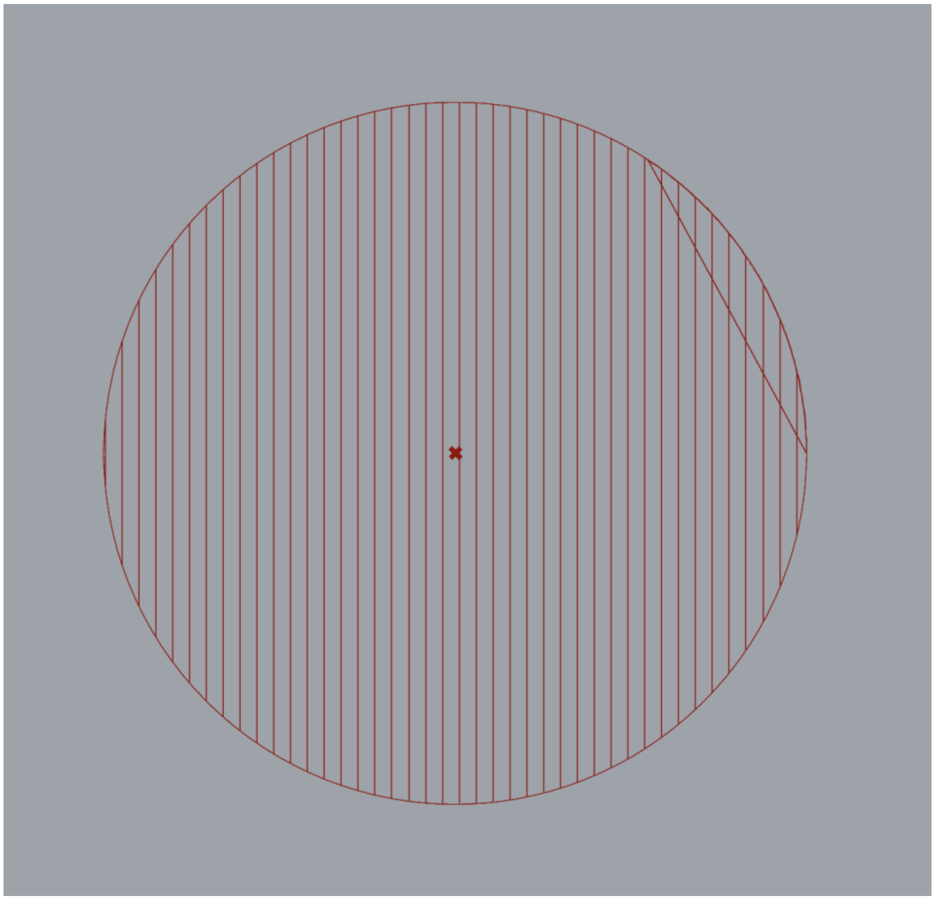
The third and final large assignment centered around 3D printers and g-code. In part one, we were tasked with creating a basic, rectilinear path generator in Grasshopper to then be translated into g-code based on a Python script nested within the Grasshopper file. The goal thereafter, in part two, is 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.



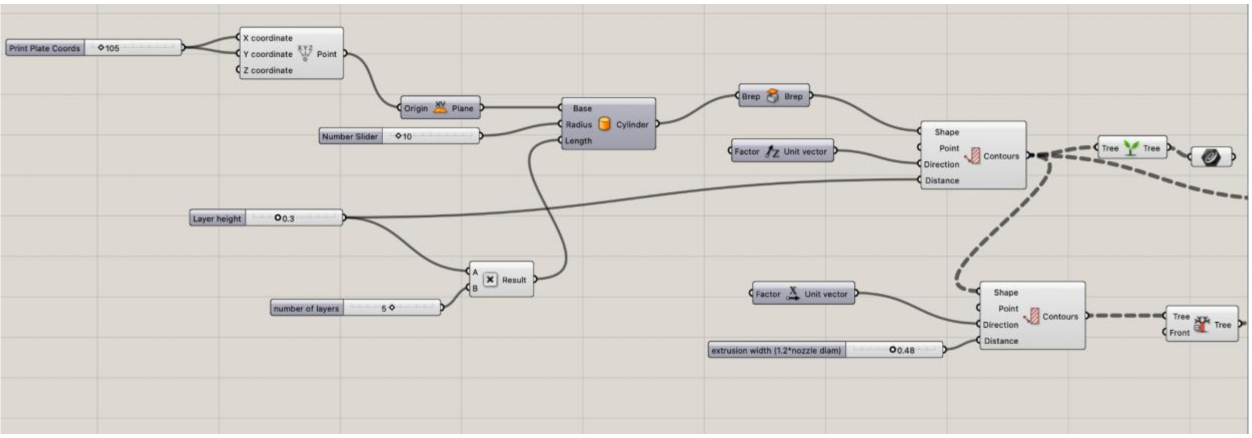
Math Used

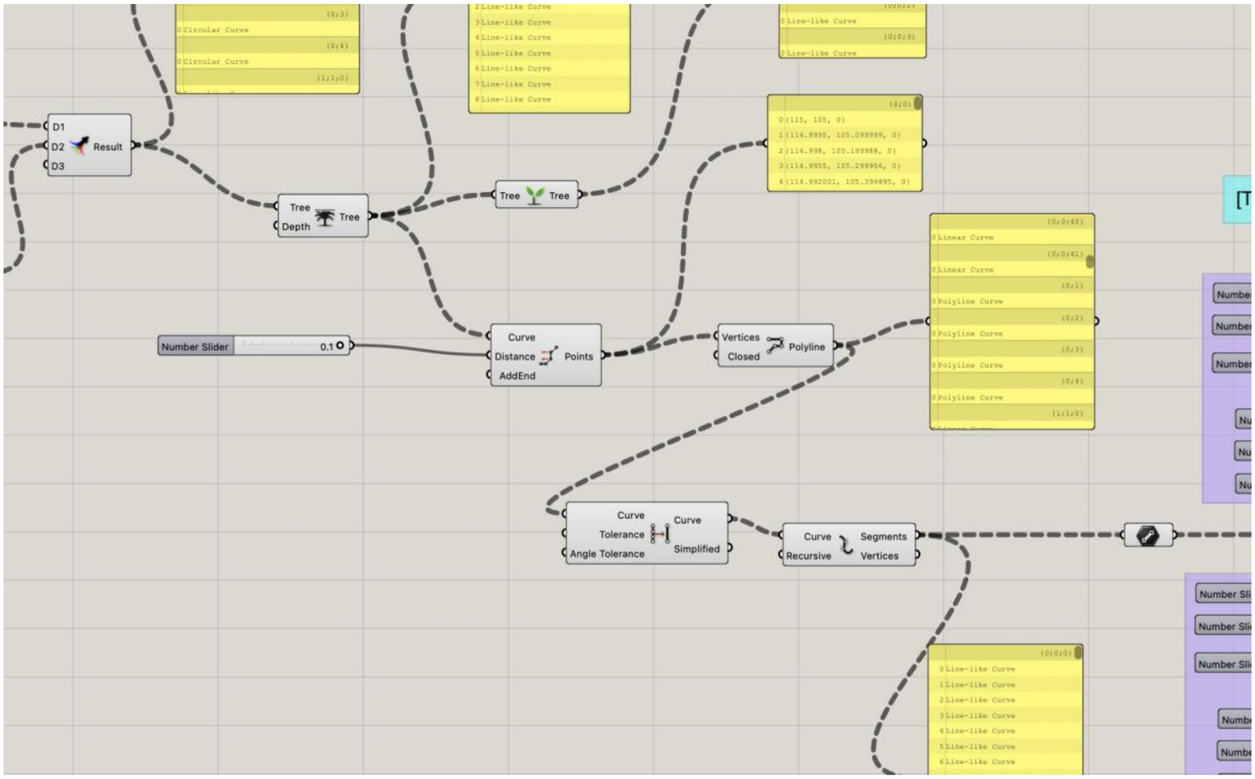
- Geometry (areas, volumes, centroids...)
- Cartesian coordinates (points, lines...)

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:





```
(0;3)
0 Circular Curve
(0;4)
0 Circular Curve
(1;1;0)
```

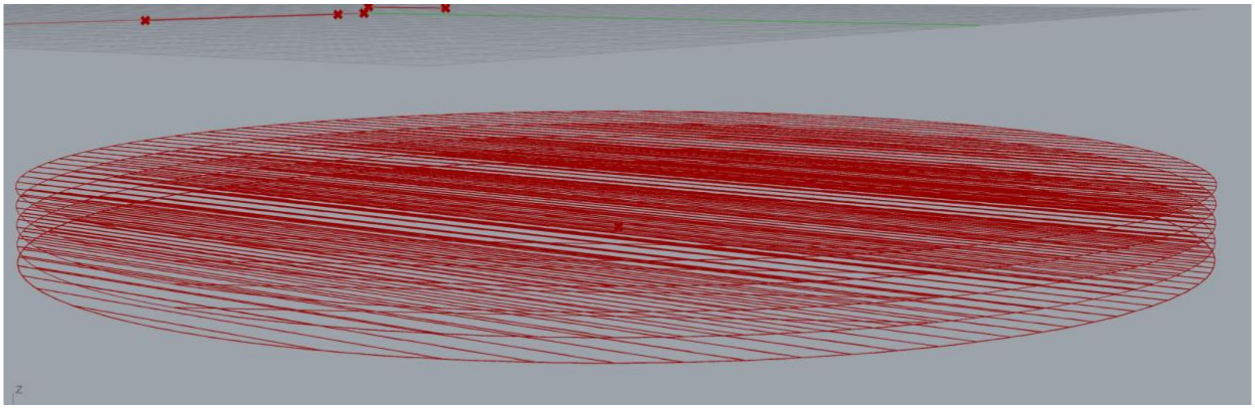
```
2 Line-like Curve
3 Line-like Curve
4 Line-like Curve
5 Line-like Curve
6 Line-like Curve
7 Line-like Curve
8 Line-like Curve
```

```
(0;0;2)
0 Line-like Curve
(0;0;3)
0 Line-like Curve
```

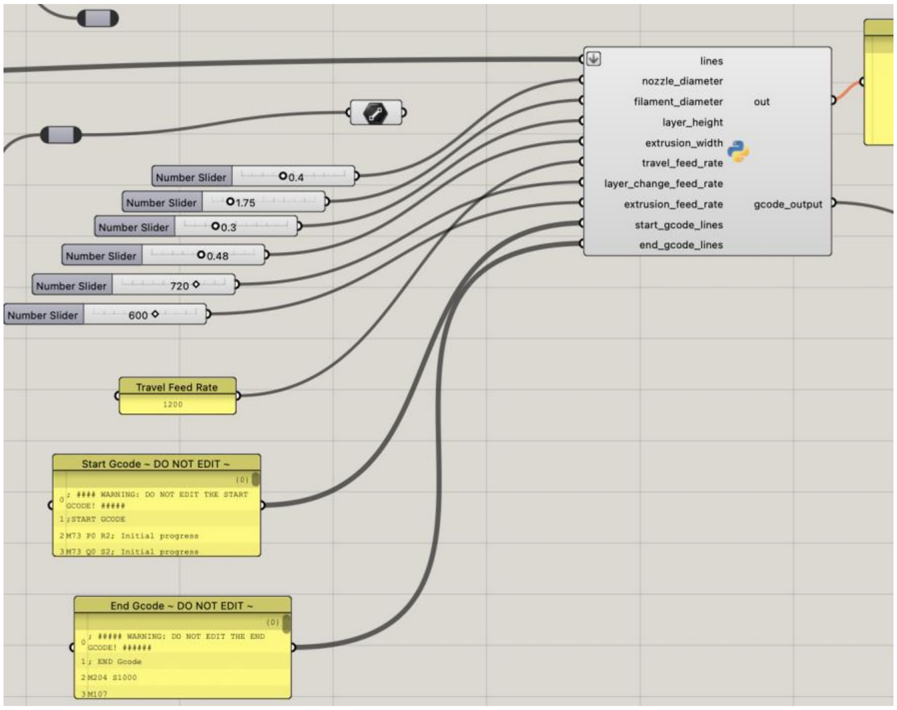
```
(0;0)
0 {115, 105, 0}
1 {114.9995, 105.099999, 0}
2 {114.998, 105.199988, 0}
3 {114.9955, 105.299956, 0}
4 {114.992001, 105.399895, 0}
```

```
(0;0;40)
0 Linear Curve
(0;0;41)
0 Linear Curve
(0;1)
0 Polyline Curve
(0;2)
0 Polyline Curve
(0;3)
0 Polyline Curve
(0;4)
0 Polyline Curve
(1;1;0)
```

```
(0;0;0)
0 Line-like Curve
1 Line-like Curve
2 Line-like Curve
3 Line-like Curve
4 Line-like Curve
5 Line-like Curve
6 Line-like Curve
```



Task #2



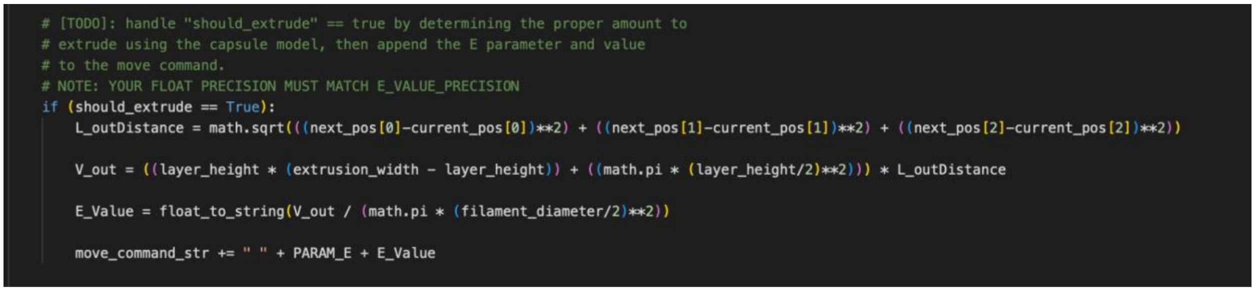
Start Gcode ~ DO NOT EDIT ~

```
{0}
0 ; #### WARNING: DO NOT EDIT THE START
  GCODE! #####
1 ;START GCODE
2 M73 P0 R2; Initial progress
3 M73 Q0 S2; Initial progress
```

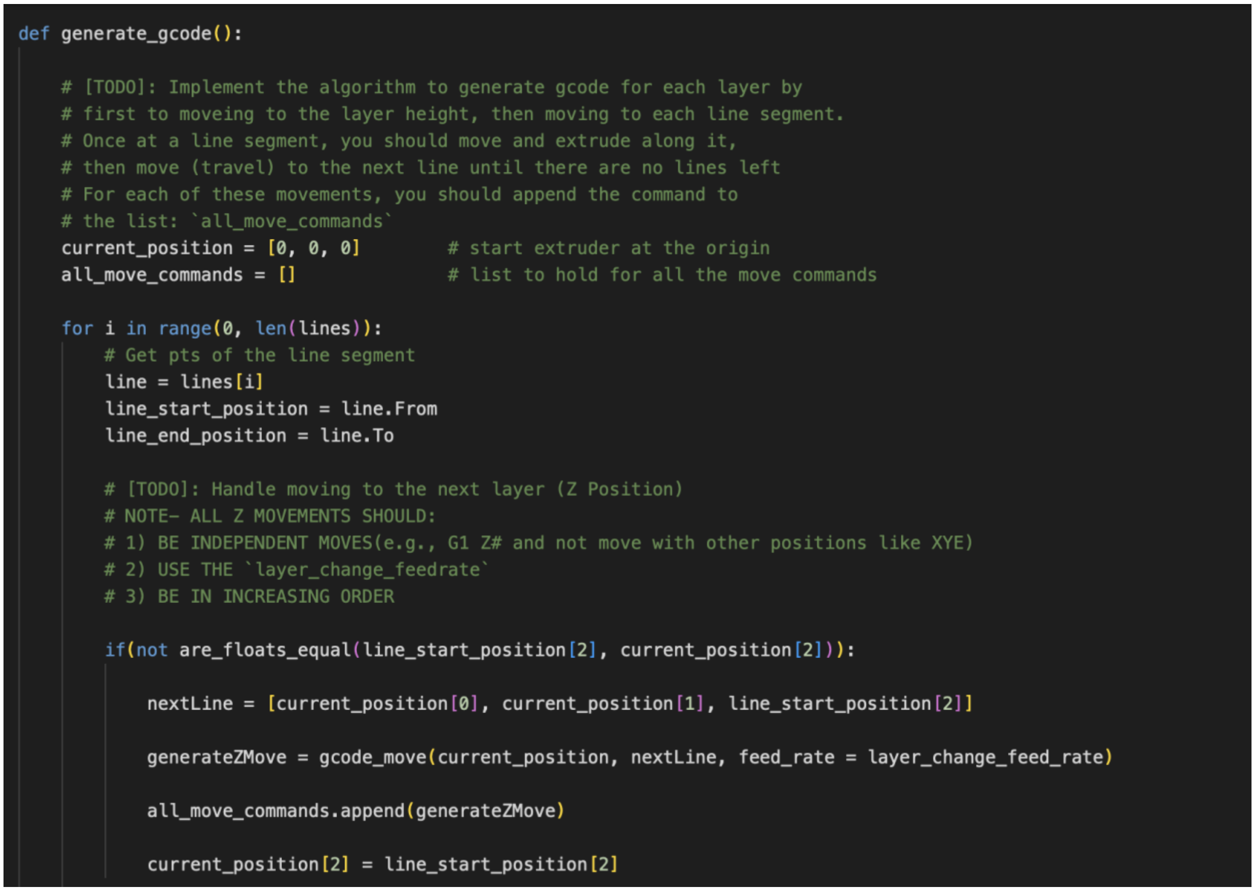
End Gcode ~ DO NOT EDIT ~

```
{0}
0 ; ##### WARNING: DO NOT EDIT THE END
  GCODE! ######
1 ; END Gcode
2 M204 S1000
3 M107
```

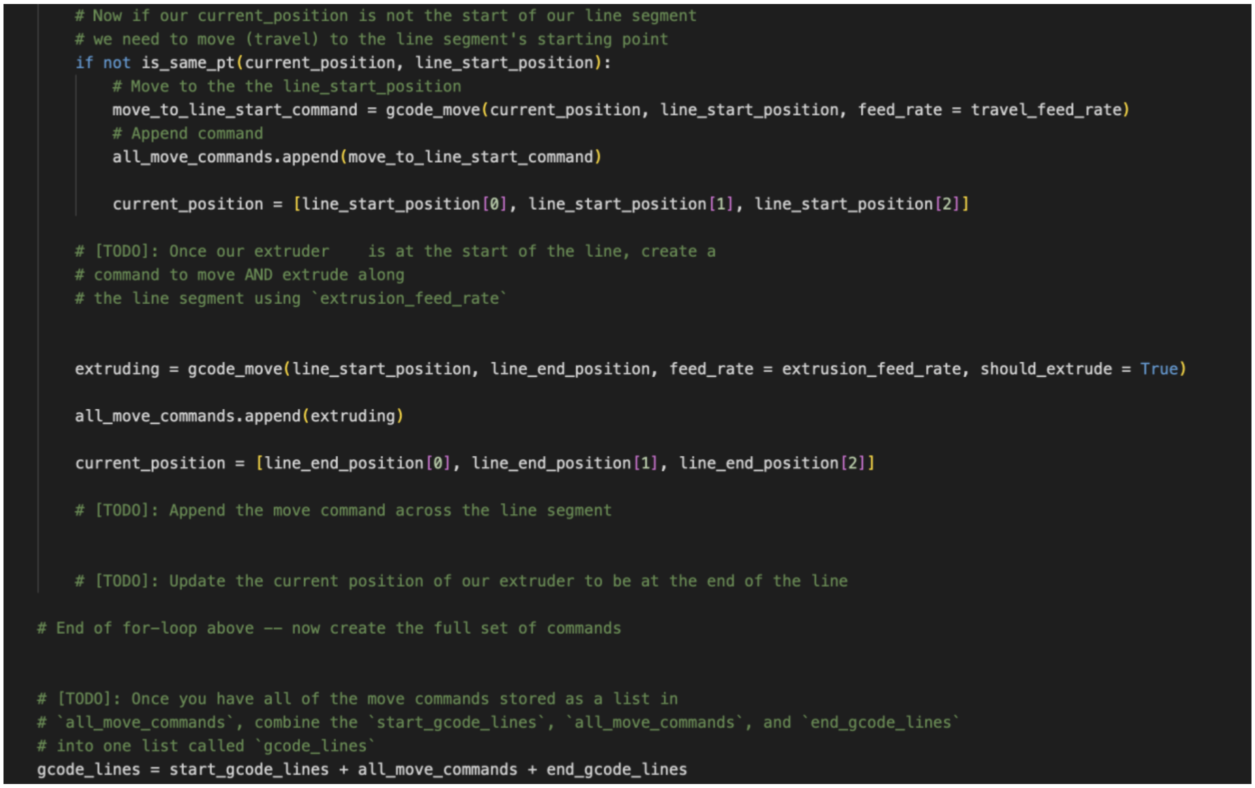
Task two was quite simple once I wrapped my head around it. Its goal was to complete the TODOs, housed within the Python script, nested in the Grasshopper file, in order to complete the g-code generation based on the printer/extrusion parameters. First up was to complete the should_extrude case for the 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:



```python
# [TODO]: handle "should_extrude" == true by determining the proper amount to
# extrude using the capsule model, then append the E parameter and value
# to the move command.
# NOTE: YOUR FLOAT PRECISION MUST MATCH E_VALUE_PRECISION
if (should_extrude == True):
    L_outDistance = math.sqrt(((next_pos[0]-current_pos[0])**2) + ((next_pos[1]-current_pos[1])**2) + ((next_pos[2]-current_pos[2])**2))

    V_out = ((layer_height * (extrusion_width - layer_height)) + ((math.pi * (layer_height/2)**2))) * L_outDistance

    E_Value = float_to_string(V_out / (math.pi * (filament_diameter/2)**2))

    move_command_str += " " + PARAM_E + E_Value
```



```python
def generate_gcode():

    # [TODO]: Implement the algorithm to generate gcode for each layer by
    # first to moveing to the layer height, then moving to each line segment.
    # Once at a line segment, you should move and extrude along it,
    # then move (travel) to the next line until there are no lines left
    # For each of these movements, you should append the command to
    # the list: `all_move_commands`
    current_position = [0, 0, 0]        # start extruder at the origin
    all_move_commands = []              # list to hold for all the move commands

    for i in range(0, len(lines)):
        # Get pts of the line segment
        line = lines[i]
        line_start_position = line.From
        line_end_position = line.To

        # [TODO]: Handle moving to the next layer (Z Position)
        # NOTE- ALL Z MOVEMENTS SHOULD:
        # 1) BE INDEPENDENT MOVES(e.g., G1 Z# and not move with other positions like XYE)
        # 2) USE THE `layer_change_feedrate`
        # 3) BE IN INCREASING ORDER

        if(not are_floats_equal(line_start_position[2], current_position[2])):

            nextLine = [current_position[0], current_position[1], line_start_position[2]]

            generateZMove = gcode_move(current_position, nextLine, feed_rate = layer_change_feed_rate)

            all_move_commands.append(generateZMove)

            current_position[2] = line_start_position[2]
```



```
    # Now if our current_position is not the start of our line segment
    # we need to move (travel) to the line segment's starting point
    if not is_same_pt(current_position, line_start_position):
        # Move to the the line_start_position
        move_to_line_start_command = gcode_move(current_position, line_start_position, feed_rate = travel_feed_rate)
        # Append command
        all_move_commands.append(move_to_line_start_command)

        current_position = [line_start_position[0], line_start_position[1], line_start_position[2]]

    # [TODO]: Once our extruder    is at the start of the line, create a
    # command to move AND extrude along
    # the line segment using `extrusion_feed_rate`


    extruding = gcode_move(line_start_position, line_end_position, feed_rate = extrusion_feed_rate, should_extrude = True)

    all_move_commands.append(extruding)

    current_position = [line_end_position[0], line_end_position[1], line_end_position[2]]

    # [TODO]: Append the move command across the line segment


    # [TODO]: Update the current position of our extruder to be at the end of the line

# End of for-loop above -- now create the full set of commands


# [TODO]: Once you have all of the move commands stored as a list in
# `all_move_commands`, combine the `start_gcode_lines`, `all_move_commands`, and `end_gcode_lines`
# into one list called `gcode_lines`
gcode_lines = start_gcode_lines + all_move_commands + end_gcode_lines
```

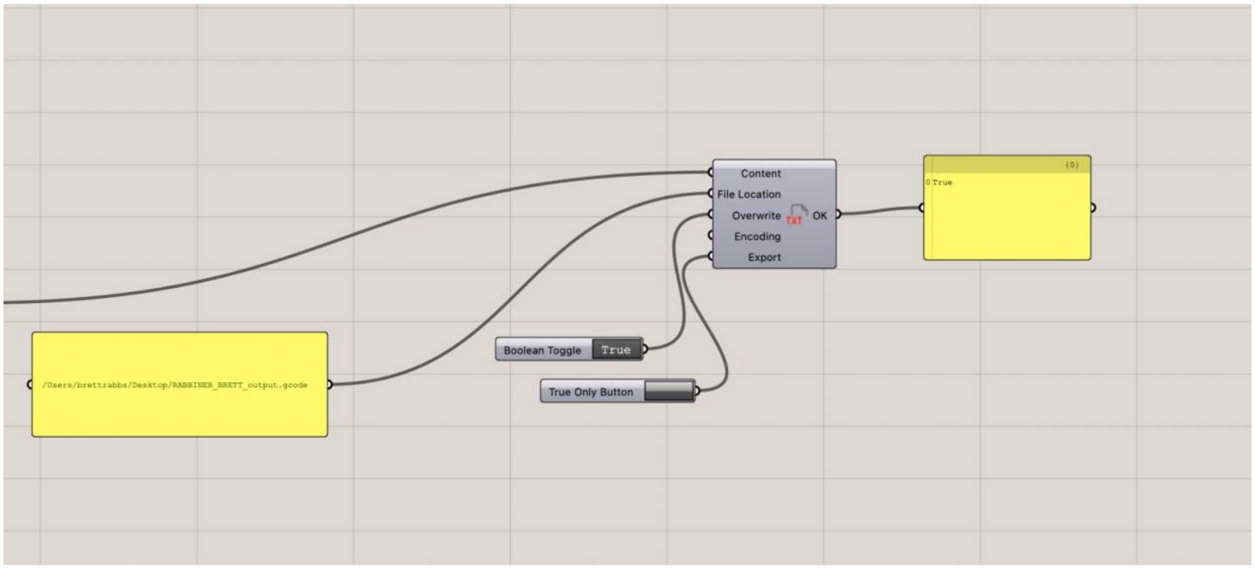
Single Layer gcode

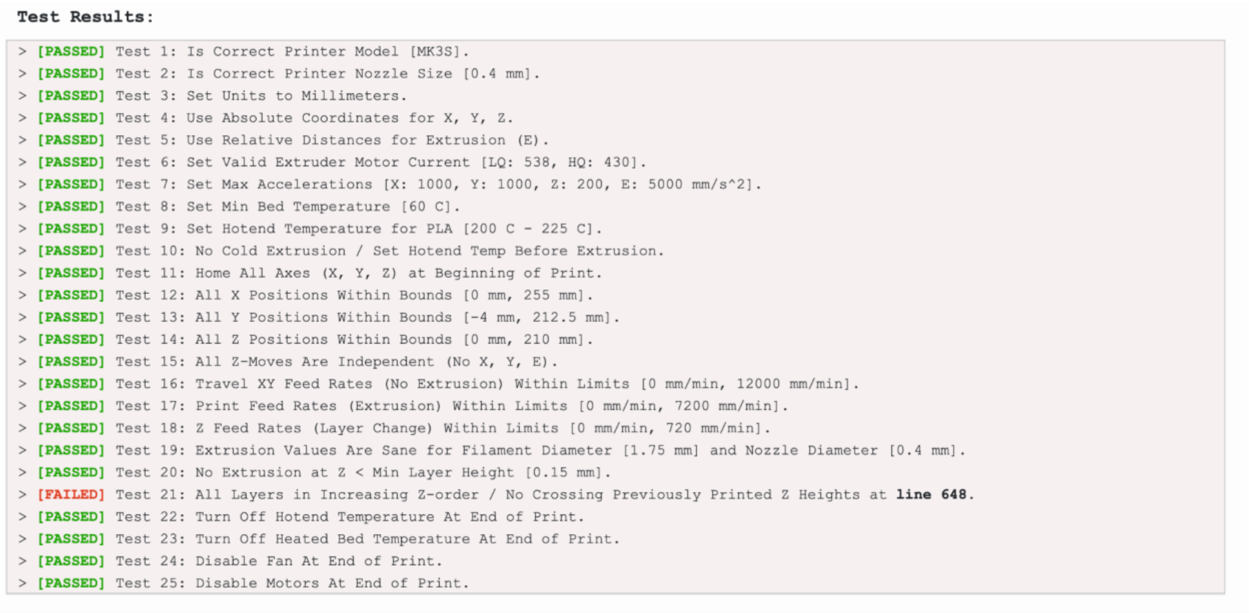
Five Layer gcode

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:





Test Results:

```
> [PASSED] Test 1: Is Correct Printer Model [MK3S].
> [PASSED] Test 2: Is Correct Printer Nozzle Size [0.4 mm].
> [PASSED] Test 3: Set Units to Millimeters.
> [PASSED] Test 4: Use Absolute Coordinates for X, Y, Z.
> [PASSED] Test 5: Use Relative Distances for Extrusion (E).
> [PASSED] Test 6: Set Valid Extruder Motor Current [LQ: 538, HQ: 430].
> [PASSED] Test 7: Set Max Accelerations [X: 1000, Y: 1000, Z: 200, E: 5000 mm/s^2].
> [PASSED] Test 8: Set Min Bed Temperature [60 C].
> [PASSED] Test 9: Set Hotend Temperature for PLA [200 C - 225 C].
> [PASSED] Test 10: No Cold Extrusion / Set Hotend Temp Before Extrusion.
> [PASSED] Test 11: Home All Axes (X, Y, Z) at Beginning of Print.
> [PASSED] Test 12: All X Positions Within Bounds [0 mm, 255 mm].
> [PASSED] Test 13: All Y Positions Within Bounds [-4 mm, 212.5 mm].
> [PASSED] Test 14: All Z Positions Within Bounds [0 mm, 210 mm].
> [PASSED] Test 15: All Z-Moves Are Independent (No X, Y, E).
> [PASSED] Test 16: Travel XY Feed Rates (No Extrusion) Within Limits [0 mm/min, 12000 mm/min].
> [PASSED] Test 17: Print Feed Rates (Extrusion) Within Limits [0 mm/min, 7200 mm/min].
> [PASSED] Test 18: Z Feed Rates (Layer Change) Within Limits [0 mm/min, 720 mm/min].
> [PASSED] Test 19: Extrusion Values Are Sane for Filament Diameter [1.75 mm] and Nozzle Diameter [0.4 mm].
> [PASSED] Test 20: No Extrusion at Z < Min Layer Height [0.15 mm].
> [FAILED] Test 21: All Layers in Increasing Z-order / No Crossing Previously Printed Z Heights at line 648.
> [PASSED] Test 22: Turn Off Hotend Temperature At End of Print.
> [PASSED] Test 23: Turn Off Heated Bed Temperature At End of Print.
> [PASSED] Test 24: Disable Fan At End of Print.
> [PASSED] Test 25: Disable Motors At End of Print.
```

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…