Emile Goulard
uID: u1244855
CS3200 - Assignment 7 Report
11/2/2022

Question 1

For seeing the results of the circular mesh, open the “A7_Question1.m” file in matlab. Refer to the README file for more information.

The methods used for dividing and conquering squares to create a circular mesh started with gathering the circle’s data: radius, theta, x, and y and then using those to create the plot of the circle. Then I began gathering the square data such as defining points in a square, and then began recursion from there. After the recursion process, I then begin defining the mesh data with each square to create the triangles that divide each square into two triangles. This is achieved by iterating through each point in the square, gathering the points, and then placing those points in an array called ‘numOfTriangles’ which will be used for patching the mesh together.

The recursive process began by creating arguments that store the current square to recurse on, the points in both the x and y axes for said square, and the current depth in recursion. The square holds the 4 corner points that will be split through this algorithm to achieve results in creating a circular mesh. In order to do this, I also create global variables that store the total number of points now created after splitting the squares into 4 more squares and so on. The conditions needed to follow if a square was completely inside or outside the circle, or if it was partially inside the circle. To check if it was completely inside, I used the all 4 points from the current square and checked if any of them landed inside or outside the circle. If it was completely inside the circle, I store those points into my global variable to store all points during recursion and then return. If it was completely outside, I simply return as well. If the diagonal distance between points in the square are greater than the radius of the circle, then I know that it’s partially inside the circle. When it’s partially inside the circle, I redefine the points in the current square to be each of the previous square’s point coordinates divided by 2. Then I simply recurse on that new square with those points.

The terminating cases that would stop recursion were if the current depth is equal to 6 and if the current area of the square is less than the minimum area that I defined as 0.015625. This way I can stop recursion when squares become really small.

These were the results of my algorithm. Many problems arised, I tried my best to get the algorithm working in MATLAB.

Figure 1 - Entire Mesh

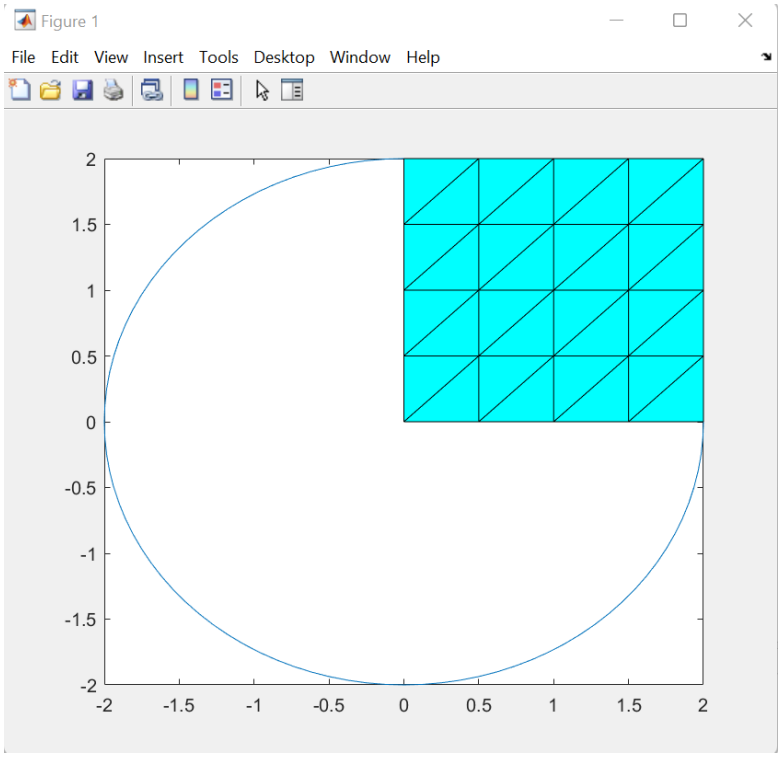
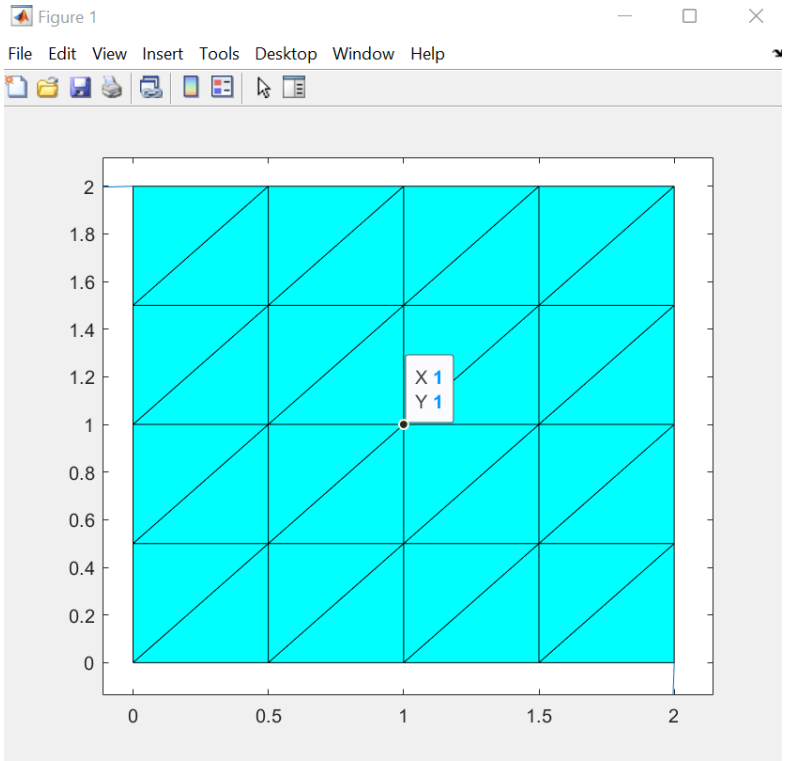


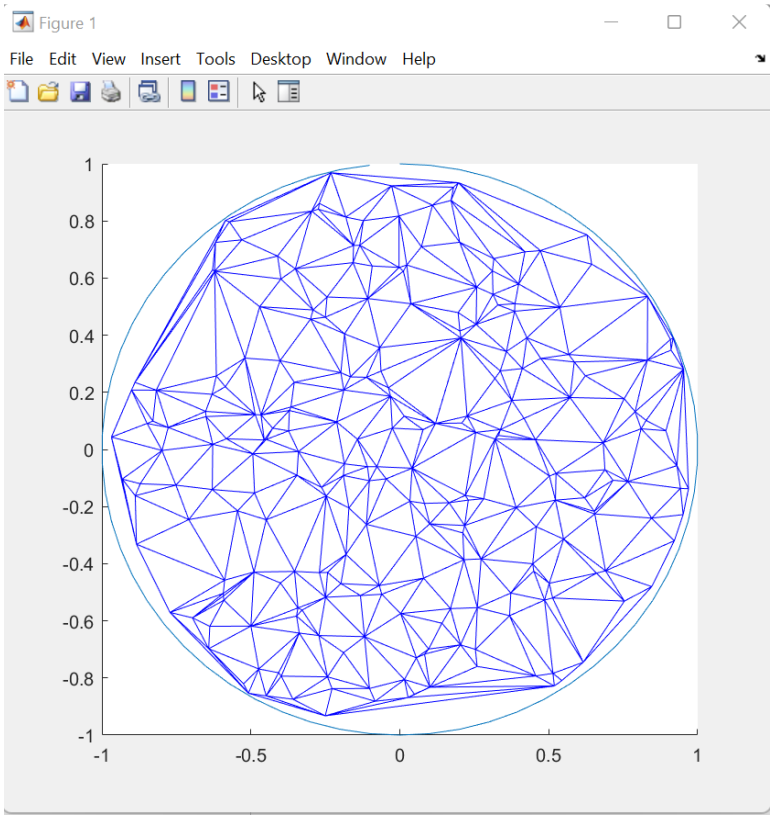
Figure 2 - 1st Quadrant



Question 2

For seeing the results of the randomly distributed triangulated mesh, open the "A7_Question2.m" file in matlab. Refer to the README file for more information.

These were the resulting visuals of adding 250 random points within the circle boundaries given by the file "circle2d-outer.node". Just to show randomness, I will be displaying 3 images after running the code 3 individual times.



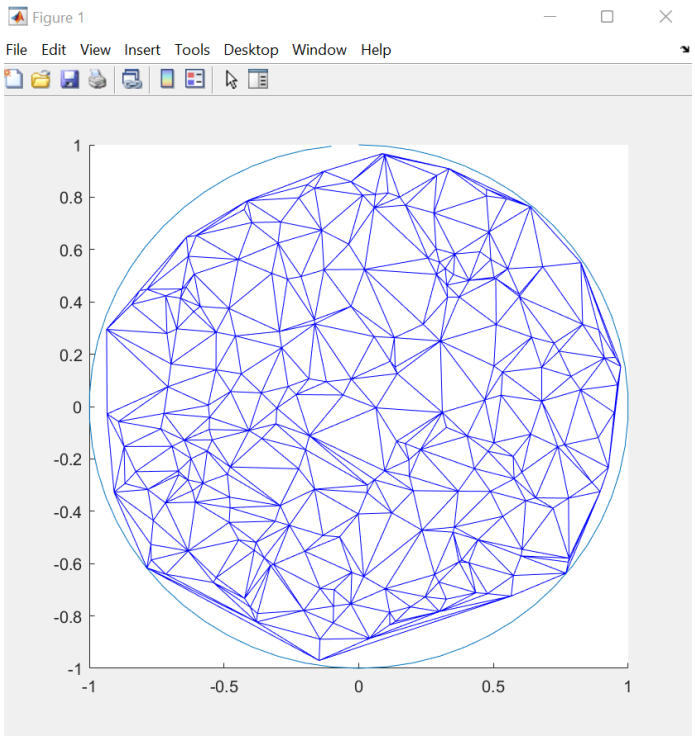


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.5
0
0.5
1

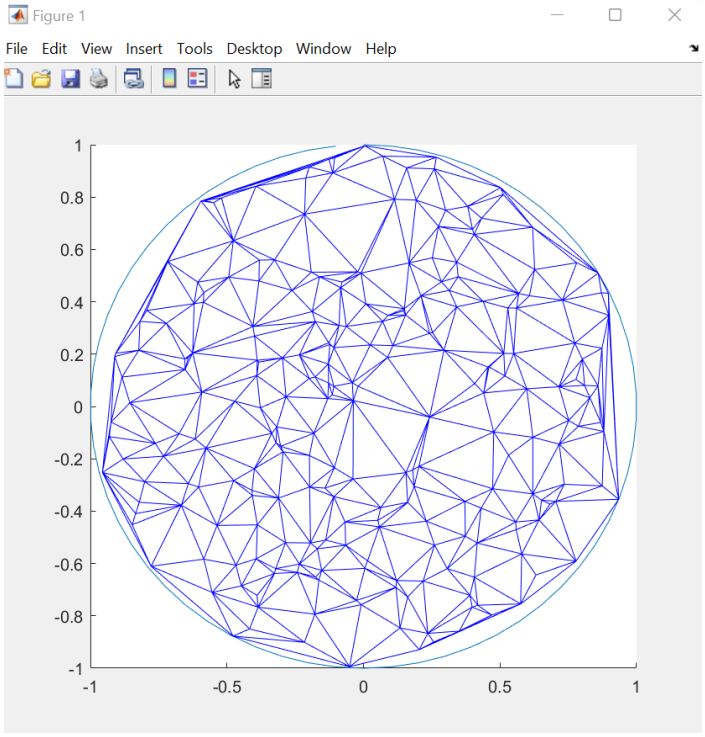
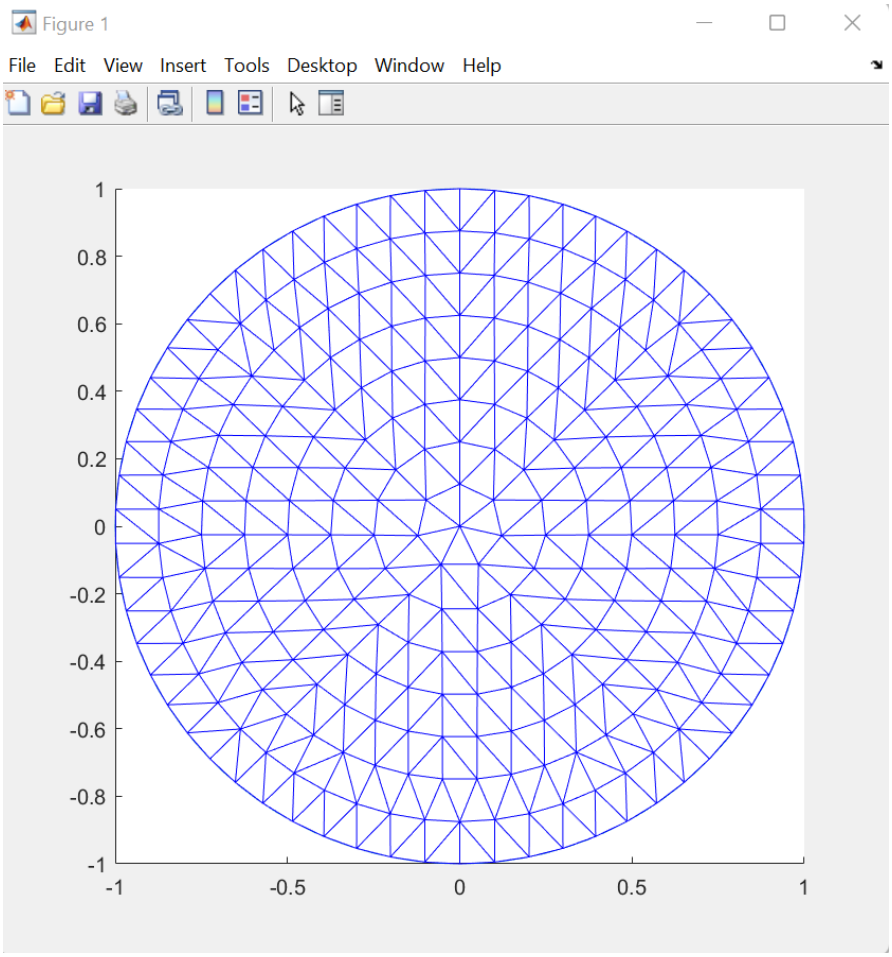


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.5
0
0.5
1

Question 3

For seeing the results of the triangulated mesh, open the “A7_Question3.m” file in matlab. Refer to the README file for more information.

Code from Question 2 carries over into Question 3, except now I use the points from the file “circle2d.node” as my x and y coordinates for generating the triangulated mesh. Below is the result:

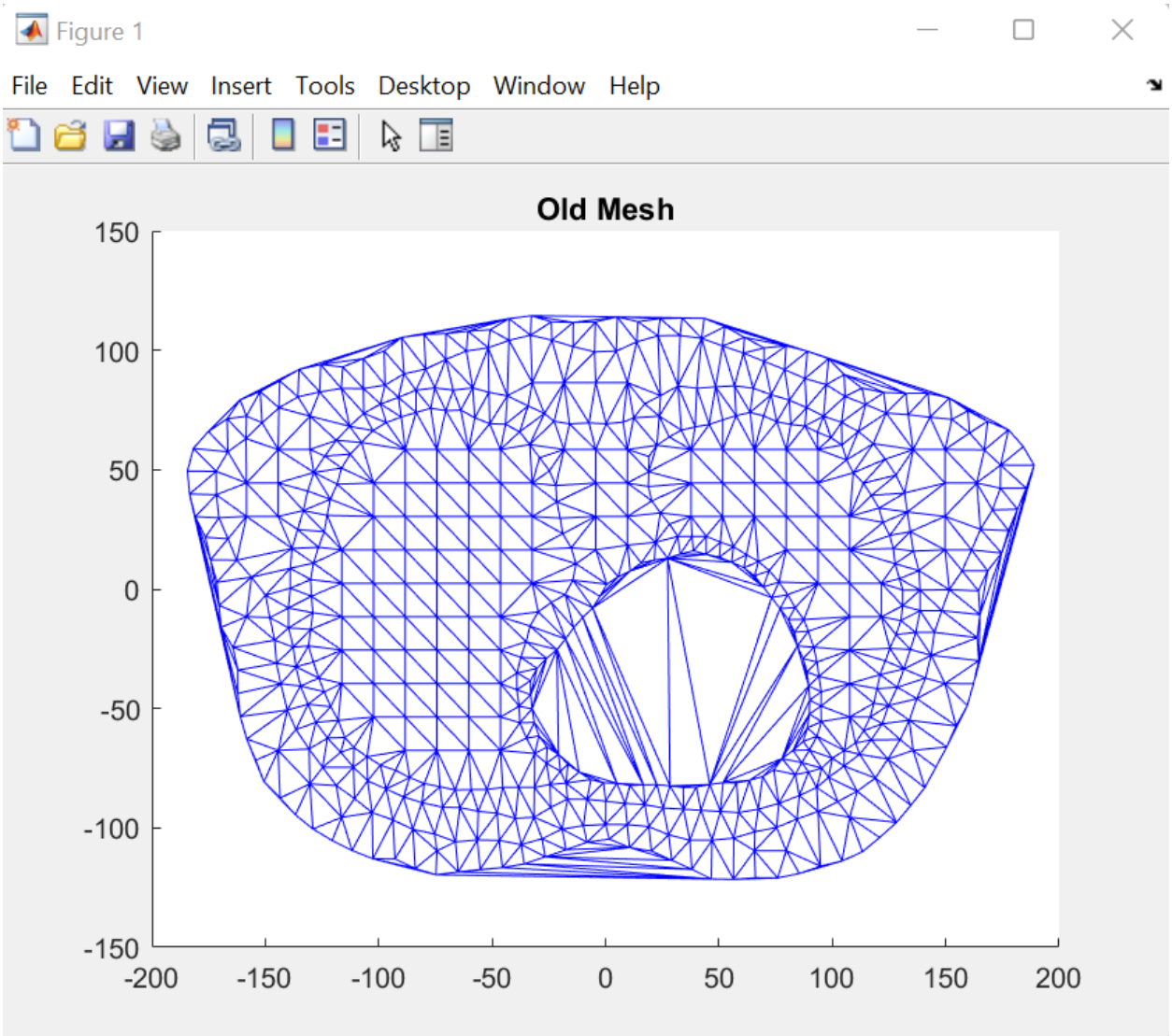


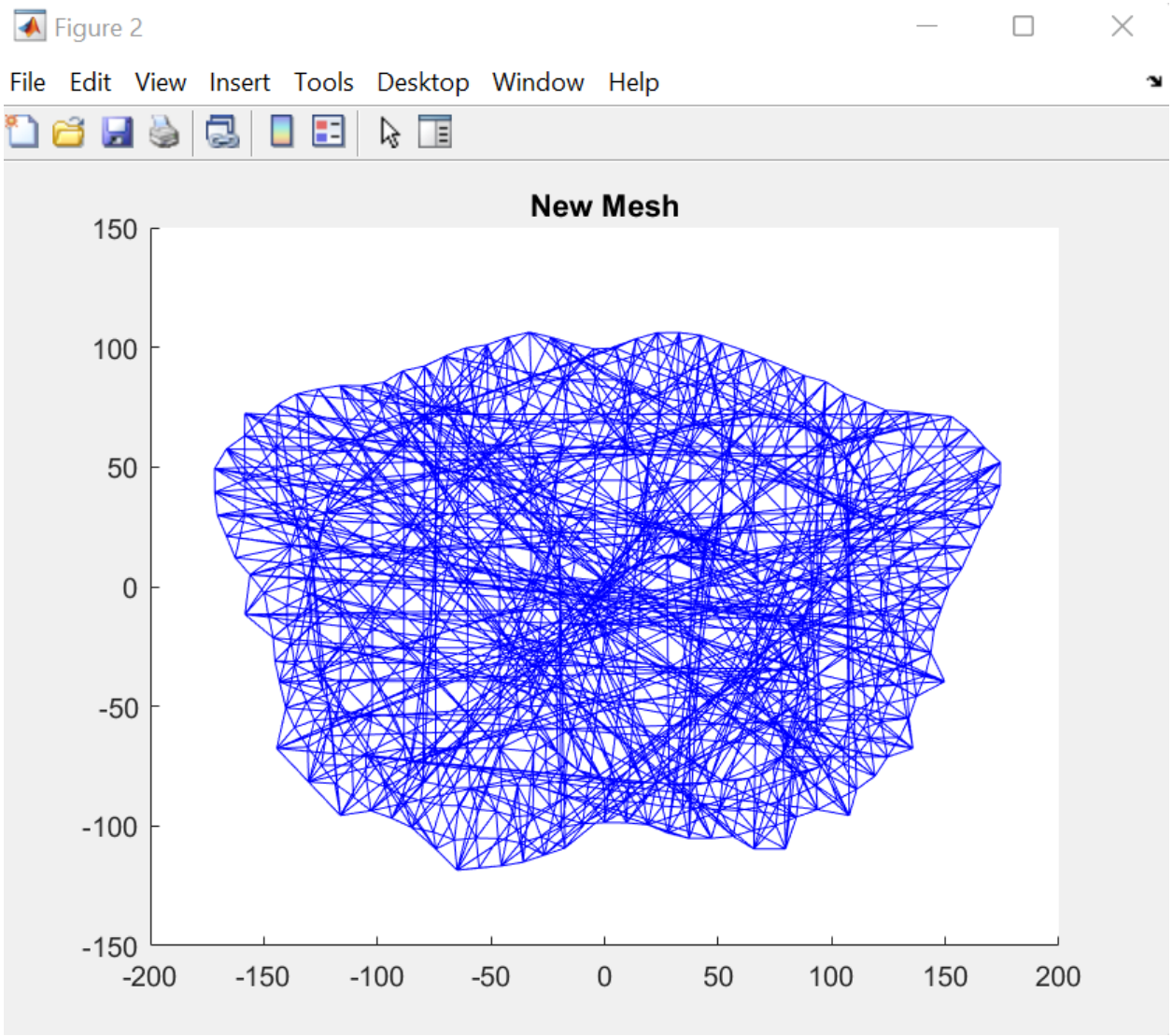
Question 4 - Extra Credit

For seeing the results of the triangulated mesh on the torso points, open the “A7_ExtraCredit.m” file in matlab. Refer to the README file for more information.

The program I wrote that attempts to remove the triangles inside the ‘heart’ region and the outer bounds started by gathering the delaunay triangulation using the points provided in “2D-torso.node” and then finding the connectivity list. What I do to try and find triangles and

remove them is simply iterating through the entire connectivity list and checking if values in any of those columns storing that edge data is less than a certain threshold. I set this threshold to 100. Then, if that coordinate is less than the threshold, I clamp their values to be a randomly generated number between 300 and 500. This way the average connectivity of the list remains in tact. While not perfect, it was a result that did technically remove the triangles on the outside and inside, to the detriment of the rest of the mesh. Below are the results of the old mesh and the new mesh:





Sources

1. https://www.mathworks.com/matlabcentral/answers/72915-creating-random-points-in-a-circle
2. https://my.eng.utah.edu/~cs3200/Meshing_Part_1.pdf
3. https://my.eng.utah.edu/~cs3200/Meshing_Part_2.pdf
4. https://www.mathworks.com/help/matlab/ref/delaunay.html
5. https://www.mathworks.com/help/matlab/ref/triangulation.size.html
6. https://www.cdslab.org/matlab/notes/program-units/variable-scope/index.html