For this practical, you will be designing several hard constraints and designing an appropriate order. It is recommended you do Task 1-3 for one constraint (to understand the workflow) first.

**Data Set Description**

I have provided two data sets, TestSet is smaller and it is recommend during initial design. BrainParcellation should be used for evaluating runtime in Task 3 and 4. FakeBrain.nii.gz is the original image, and each other image is a binary mask, the name corresponds to the structure.

Target Points (targets.fcsv) and Entry points (entries.fcsv) are [MarkupFiducials](https://apidocs.slicer.org/v4.8/classvtkMRMLMarkupsNode.html#a98e21fa54198551a3884267d83e3c24a) listing possible entry and target points, you can pair these to create possible trajectories to consider.

The number of points in each fiducial can be found by

#how to list the number of points

fiducialNode = slicer.util.getNode('entries')

fiducialNode.GetNumberOfMarkups()

576

# how to iterate over the fiducials and get the real world coordinates, the forth dimension is time

for x in range(0, fiducialNode.GetNumberOfMarkups()) :

world =[0,0,0,0]

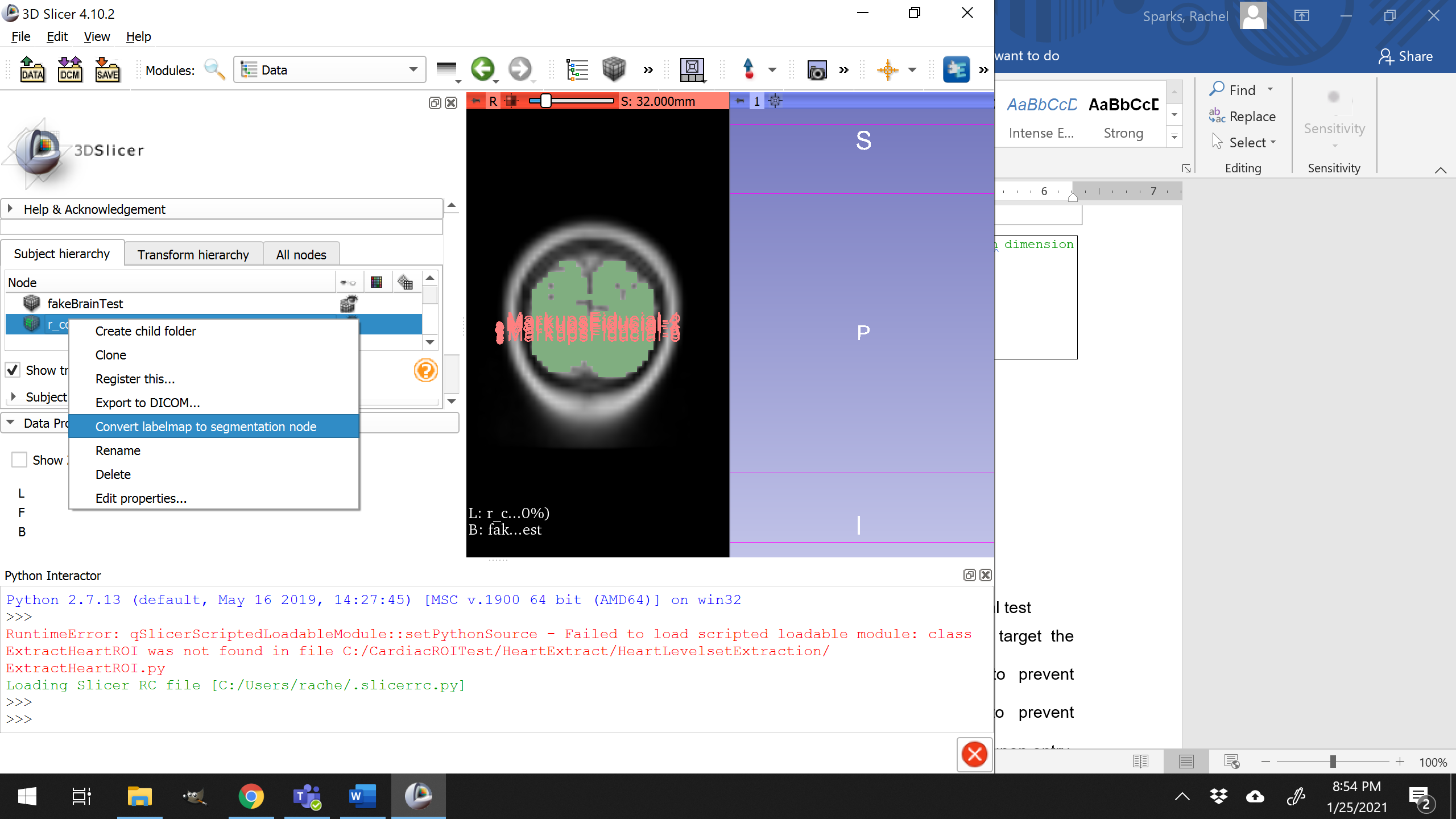
fiducialNode.GetNthFiducialWorldCoordinates(x, world)

print(world)

[207.927, 120.111, 130.003, 1.0]

Task 0.

1. Load all the files located in the TestSet.
2. In the python console use slicer.util to assign a variable to each node
3. Identify the size of the data in each node.
   1. What are the dimensions of the image
   2. What are the number of points in the entries and targets fiducials
   3. For the binary images – use the convert label map to segmentation node (as shown below) to create a surface node. What is the bounding box of each surface node.
   4. Try to identify a way to convert a label map to a segmentation node from the command prompt.

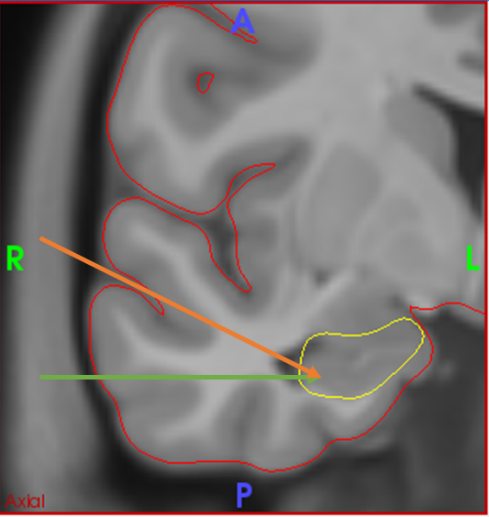


1. Repeat these tasks for the files in BrainParcellation. Note how the sizes differ.

**Task Description**

Task 1. Given the following statements, write a corresponding formal mathematical test

1. Placement of an electrode for recording temporal lobe epilepsy must target the hippocampus (r\_hippo.nii.gz)
2. Placement must avoid penetrating the ventricles (ventricles.nii.gz) to prevent cerebrospinal fluid leakage.
3. Placement must avoid hitting any blood vessels (vessels.nii.gz) to prevent haemorrhage.
4. Electrodes may deflect if not perpendicular (<55°) to cortex (cortex.nii.gz) upon entry. I.E. tool placement should be placed similar to green trajectory, the orange trajectory is too shear:



Task 2. For each mathematical test in Task 1 design a simple algorithm on paper that can perform the check. First identifying the input and output datatypes and then determining how the information should flow through the program to give the desired output. Determine its Big O Notation

Task 3. For each algorithm in Task 2, write a python function in 3D Slicer.

1. Use the Extension Wizard found under “Developer Tools” to create your own module (I suggest naming it “PathPlanning”).
2. Open the file in your favourite python editor and identify
   1. Where the portions of the code related to its layout on the screen are
   2. Where the functions within the module are
   3. Where the test call is
3. For each algorithm create a new function to perform the given task.
4. Determine its Big O notation – is this the same as Task 2. If not, what identify the changes made to the algorithm and why.
5. Evaluate how long each algorithm takes. This can be done with the following code

import time

startTime = time.time()

# put RunMyAwesomeAlgorithm() here

endTime = time.time()

print('RunMyAwesomeAlgorithm took', endTime-startTime, 'seconds')

('RunMyAwesomeAlgorithm took', 0.006000041961669922, 'seconds')

1. Determine how many trajectories can be rejected for each test out of the total number of points

Task 4. Based on the Task 3 identify the order of operations each test should run, provide a justification based on theoretical and practical (time and search space reduction) considerations. Implement the end-to-end algorithm; report on the final runtime performance. Note this should be faster than the summative test in Task 3.

Image Filter References for Inspiration

[sitkDanielssonDistanceMapImageFilter](https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1DanielssonDistanceMapImageFilter.html)

[sitkSignedMaurerDistanceMapImageFilter](https://itk.org/SimpleITKDoxygen/html/classitk_1_1simple_1_1SignedMaurerDistanceMapImageFilter.html)

[sitkXorImageFilter](https://itk.org/SimpleITKDoxygen/html/sitkXorImageFilter_8h_source.html)

[sitkBinaryNotImageFilter](https://itk.org/SimpleITKDoxygen/html/sitkBinaryNotImageFilter_8h_source.html)

[itkOctree](https://itk.org/Doxygen/html/classitk_1_1Octree.html)

[vtkMarchingCubes](https://vtk.org/doc/nightly/html/classvtkMarchingCubes.html)

[vtkOBBTree](https://vtk.org/doc/release/5.2/html/a00908.html)

[vtkHyperOctree](https://vtk.org/doc/release/7.1/html/classvtkHyperOctree.html)