# Task description

## Getting familiar with the 3D Slicer environment

If not already familiar with 3D Slicer and its main components, then

* Download the latest stable release from the [3D Slicer download page](https://download.slicer.org/)
* Read the following presentations
  + [3D Slicer basics](https://github.com/PerkLab/PerkLabBootcamp/blob/master/Doc/day1_1_3DSlicerBasics.pptx)
  + [Writing correct and understandable code](https://github.com/PerkLab/PerkLabBootcamp/blob/master/Doc/day3_1_WritingCorrectAndUnderstandableCode.pptx)
* Do the following user tutorials:
  + [Data loading and visualization](https://github.com/PerkLab/PerkLabBootcamp/blob/master/Doc/day1_2_DataLoadingAndVisualizationTutorial.pptx)
  + [DICOM](https://github.com/PerkLab/PerkLabBootcamp/blob/master/Doc/day1_3_DICOMTutorial.pptx)
* Read the document next to this one called *DeveloperHowtos*
* Do the [programming tutorial](https://github.com/PerkLab/PerkLabBootcamp/blob/master/Doc/day3_2_SlicerProgramming.pptx)
  + Do not continue until the program behaves exactly as described!

## First development task

This task is about adding some initial features to an imaginary Slicer-based custom application.

If there are any questions feel free to ask!

### Build

First the application needs to be checked out and built.

* Fork the repository <https://github.com/EBATINCA/FirstTaskApp>
* Check out the source code to your local storage
* Install prerequisites for your OS based on the [Slicer build instructions](https://slicer.readthedocs.io/en/latest/developer_guide/build_instructions/index.html)
* Copy the build scripts for your OS from FirstTaskApp/Scripts to your development directory
* Edit them for your environment and start the build

(Main files to edit will be *Modules/FirstTask/FirstTask.py* and *Resources/UI/FirstTask.ui*)

### Features to implement

Open *FirstTask.ui* in Qt designer (see how to do it properly in *DeveloperHowtos* document).

Add three *ctkCollapsibleButton* objects with the following titles (as underscored):

DICOM

In this section (i.e. ctkCollapsibleButton) add one QPushButton with the text “Show DICOM browser”

Open DICOM browser:

* Setup connections (in setupConnections function):
  + Make the button toggleable: enable *checkable* property (in Qt Designer or in code)
  + Connect to the toggled(bool) signal:

self.ui.showDicomBrowserButton.toggled.connect(self.onShowDicomBrowserButtonToggled)

* Disconnect (disconnect function):  
  self.ui.showDicomBrowserButton.toggled.disconnect()
* Show DICOM browser  
   def onShowDicomBrowserButtonToggled(self, on):  
   if on:  
   dicomWidget = slicer.modules.dicom.widgetRepresentation().self()  
   slicer.modules.DICOMWidget.enter()  
   else:  
   slicer.modules.DICOMWidget.exit()
* Make sure the toolbar with the Import button is visible  
  Add this in the *setupUi function:*

dicomWidget = slicer.modules.dicom.widgetRepresentation().self()

slicer.util.findChild(dicomWidget.browserWidget, 'ToolBar').visible = True

Data

In this section of the UI add a qMRMLSubjectHierarchyTreeView object.

In the *setupUi* function add code to customize it:

# Hierarchy tree view

self.ui.SubjectHierarchyTreeView.dragDropMode = qt.QAbstractItemView.InternalMove

self.ui.SubjectHierarchyTreeView.selectionMode = qt.QAbstractItemView.ExtendedSelection

self.ui.SubjectHierarchyTreeView.setColumnHidden(self.ui.SubjectHierarchyTreeView.model().idColumn, True)

self.ui.SubjectHierarchyTreeView.setColumnHidden(self.ui.SubjectHierarchyTreeView.model().transformColumn, True)

self.ui.SubjectHierarchyTreeView.setEditTriggers(qt.QAbstractItemView.DoubleClicked)

Add function to be able to react to MRML scene changes:

In *setupConnections*

self.parent.mrmlSceneChanged.connect(self.onMrmlSceneChanged)

New function:

  def onMrmlSceneChanged(self, mrmlScene):

    self.ui.SubjectHierarchyTreeView.setMRMLScene(slicer.mrmlScene)

Volume information

* qMRMLVolumeInfoWidget should be used. It has a Qt Designer plugin so just drag&drop it in the designer from the Widget Box. Call the object VolumeInfoWidget
* The *mrmlSceneChanged* signal needs to be connected to the *setMRMLSene* slot of the widget, similarly to how it was done for the subject hierarchy (SH) tree view
* Connect the *currentItemChanged* signal of the SH tree view to a new function with the signature *onSubjectHierarchyTreeViewCurrentItemChanged(self, itemID)*
* In this function you can get the node from the item as you can find it in the script repository
* If the node is of the type *vtkMRMLScalarVolumeNode* (use the *IsA* function on the node), then set it as current node to the Volume info widget, otherwise set None