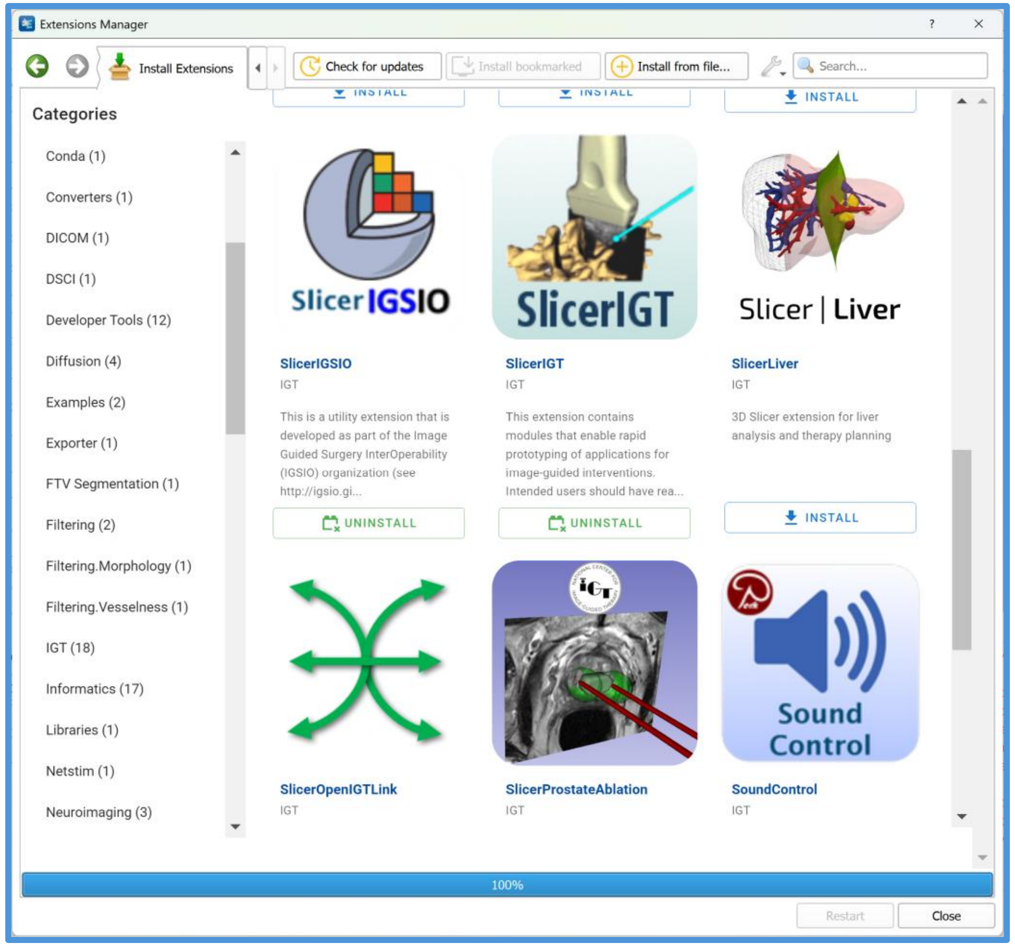
3D Slicer Module Implementation in Python

Mónica Sevilla García
Alicia Pose-Díez de la Lastra
Lucía Cubero Gutiérrez
Sergio Carreras Salinas

MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES
Cofinanciado por la Unión Europea
AGENCIA ESTATAL DE INVESTIGACIÓN
Comunidad de Madrid

3D Slicer Architecture: Extensions and Modules

- Functionality is provided through modules
- Modules can be grouped by purpose into Extensions (Segmentation, Registration, Visualization, etc.)
- 3D Slicer is a modular platform designed for extensibility:
    - Pre-installed modules
    - Available from the Extension Manager
    - Option to create your own modules to extend Slicer's capabilities



SlicerIGSIO
IGT
This is a utility extension that is developed as part of the Image Guided Surgery InterOperability (IGSIO) organization (see http://igsio.gi...

SlicerIGT
IGT
This extension contains modules that enable rapid prototyping of applications for image-guided interventions. Intended users should have rea...

SlicerLiver
IGT
3D Slicer extension for liver analysis and therapy planning

SlicerOpenIGTLink
IGT

SlicerProstateAblation
IGT

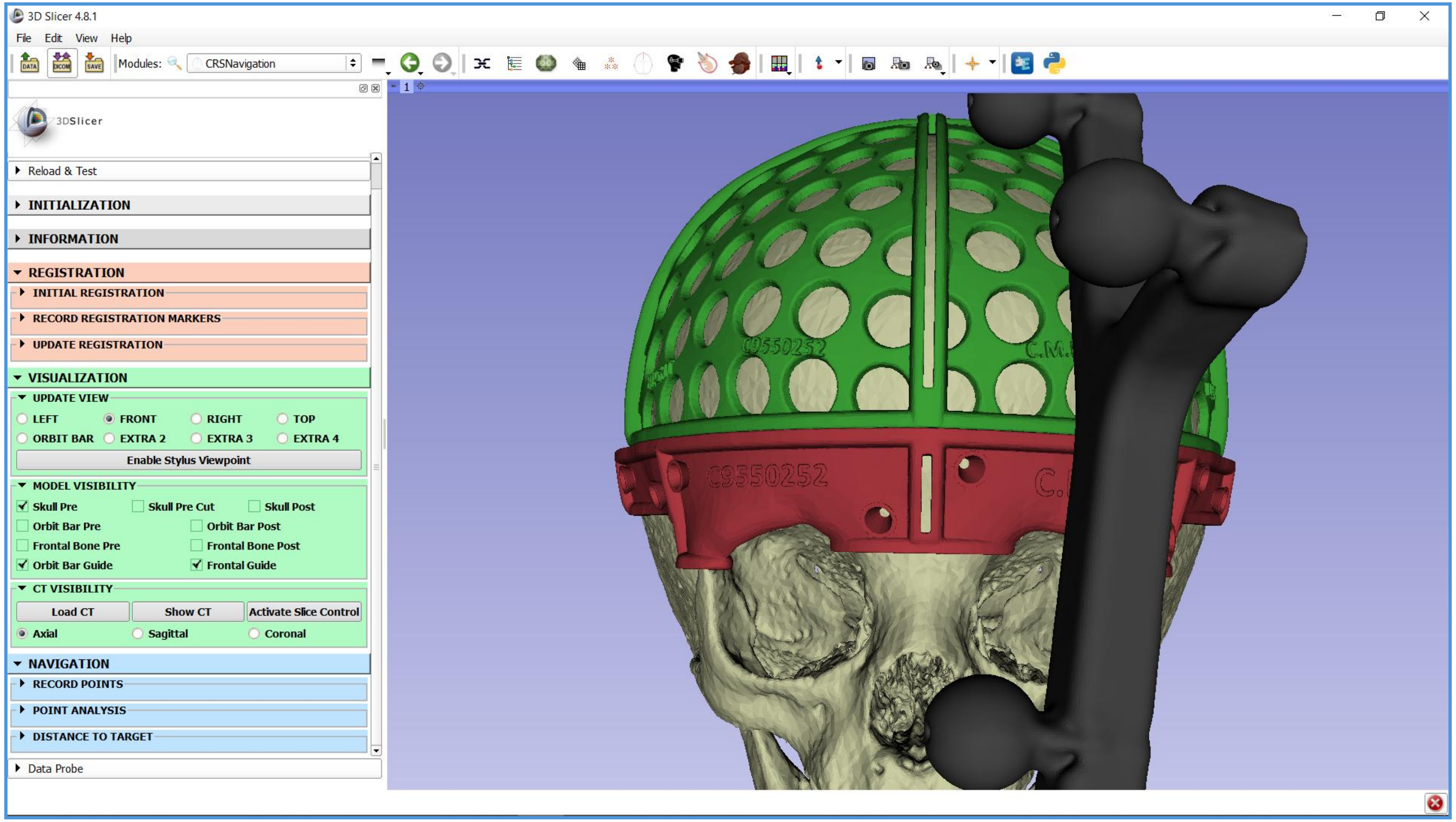
SoundControl
IGT

Why develop your own module?

- Existing tools may not meet specific project or research needs
- Automate repetitive tasks (loading, processing, exporting data)
- Integrate your own image processing or AI algorithms
- Enable reproducible workflows to share with collaborators
- Use in research, teaching or clinical applications

Examples of custom modules

- Surgical navigation module for craniosynostosis surgery



Modules: CRSNavigation

○ LEFT ● FRONT ○ RIGHT ○ TOP

○ ORBIT BAR ○ EXTRA 2 ○ EXTRA 3 ○ EXTRA 4

☑ Skull Pre □ Skull Pre Cut □ Skull Post

□ Orbit Bar Pre □ Orbit Bar Post

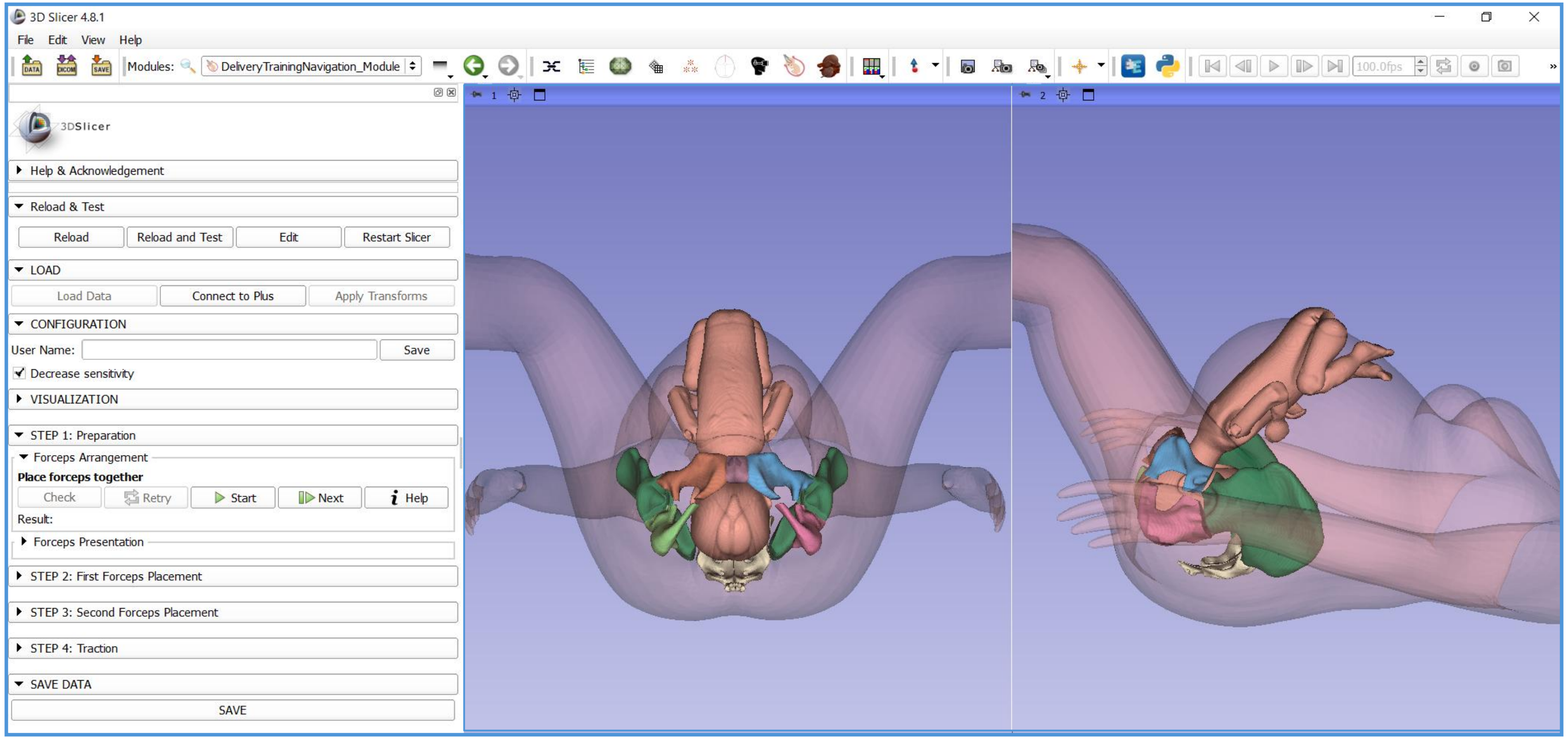
□ Frontal Bone Pre □ Frontal Bone Post

☑ Orbit Bar Guide ☑ Frontal Guide

● Axial ○ Sagittal ○ Coronal

Examples of custom modules

- Training module for birth delivery

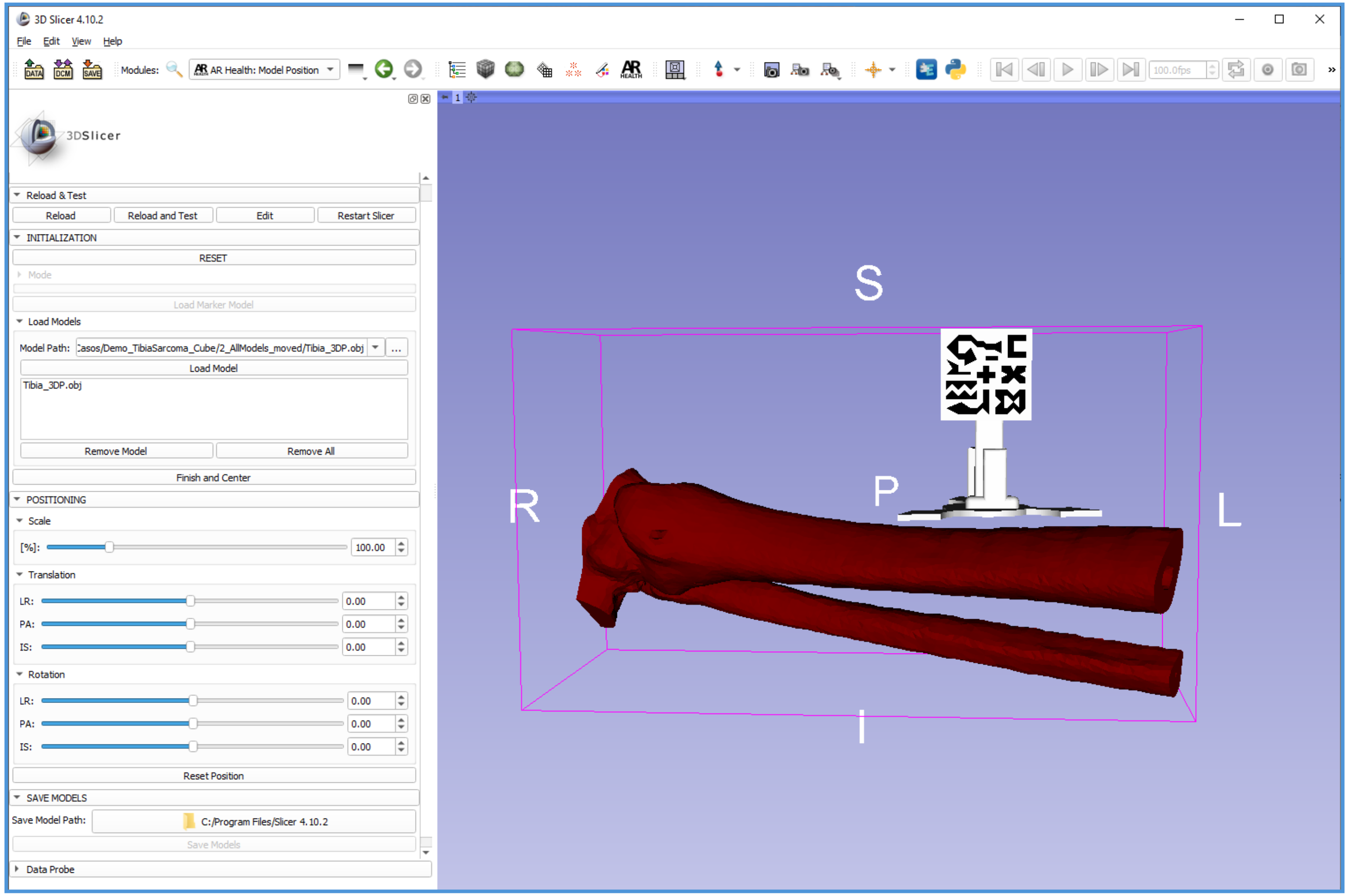


☑ Decrease sensitivity

Examples of custom modules

3D Slicer SOFA

- Module for building custom AR applications in healthcare



3D Slicer 4.10.2

Modules: AR Health: Model Position

Model Path: :asos/Demo_TibiaSarcoma_Cube/2_AllModels_moved/Tibia_3DP.obj

Tibia_3DP.obj

Scale

[%]: 100.00

Translation

LR: 0.00

PA: 0.00

IS: 0.00

Rotation

LR: 0.00

PA: 0.00

IS: 0.00

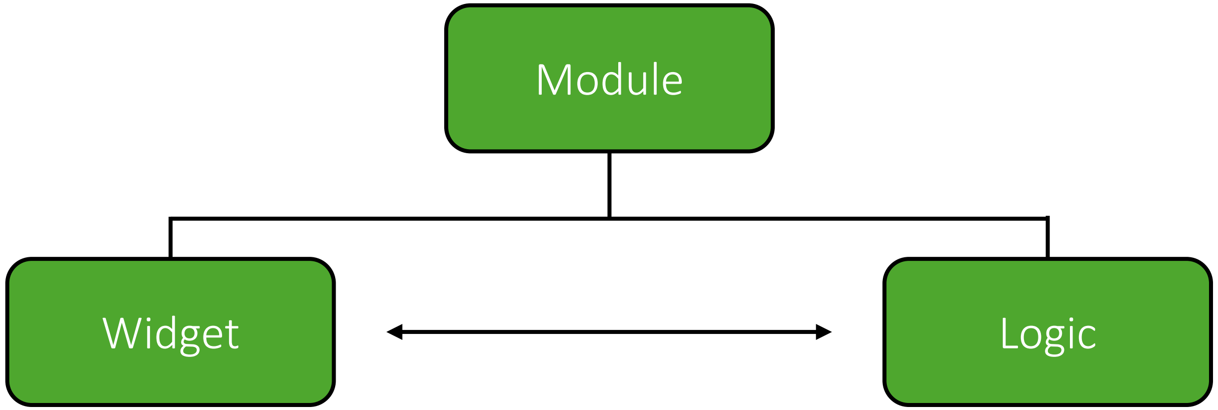
Save Model Path: C:/Program Files/Slicer 4.10.2

Types of modules in 3D Slicer

- CLI modules (Command Line Interface):
  - Use external executables or Python scripts
  - Good for batch processing or headless execution
- Scripted modules (Python):
  - Entirely written in Python
  - Easy to develop and test
  - Full access to GUI and MRML scene
- Loadable modules (C++):
  - Highest performance
  - Complex to build and maintain (requires compilation)

Anatomy of scripted module

- Scripted modules are typically a single *.py* file
- Key components:
  - Module class: defines metadata (name, category)
  - Module Widget class: builds the graphical interface (interactions with buttons)
  - Module Logic class: implements the core functionality and processing

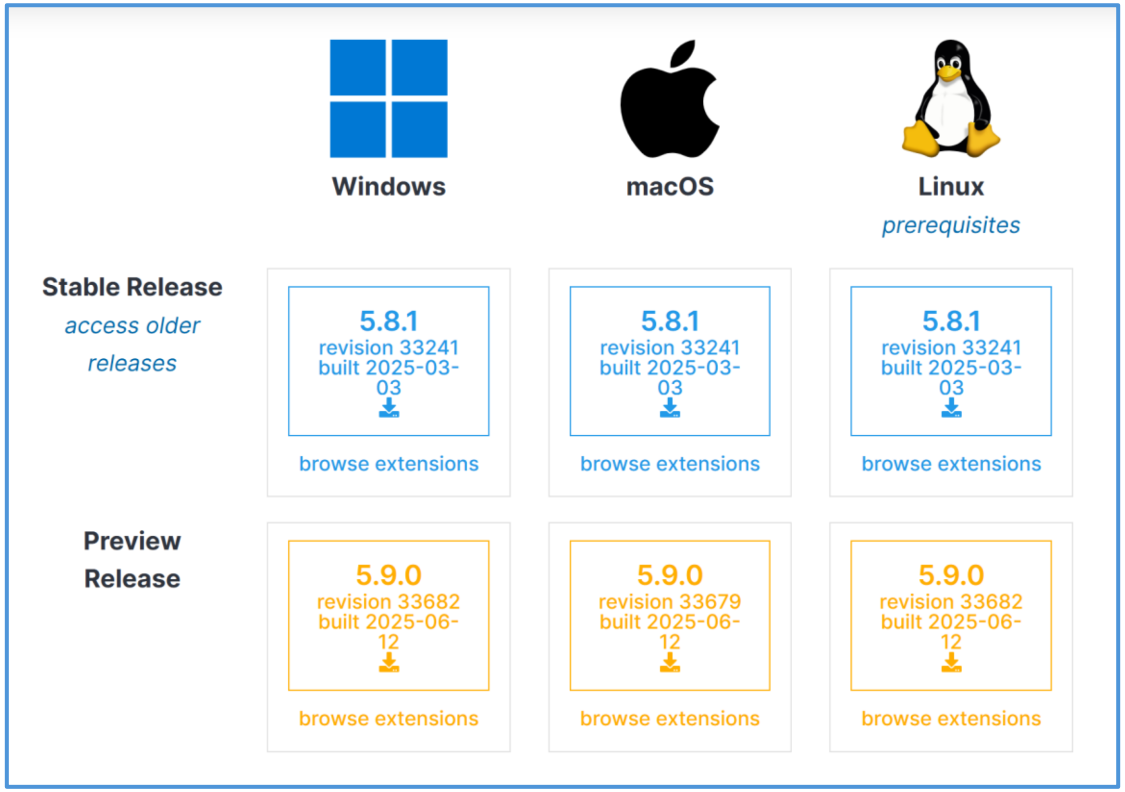


Design of the User Interface (UI):

- The GUI is built using Qt Designer
- Already embedded into Slicer → no need for installation

Installation requirements

- Download and install the latest version of **3D Slicer** from the following link: https://download.slicer.org/



	Windows	macOS	Linux *prerequisites*
Stable Release *access older releases*	5.8.1 revision 33241 built 2025-03-03 browse extensions	5.8.1 revision 33241 built 2025-03-03 browse extensions	5.8.1 revision 33241 built 2025-03-03 browse extensions
Preview Release	5.9.0 revision 33682 built 2025-06-12 browse extensions	5.9.0 revision 33679 built 2025-06-12 browse extensions	5.9.0 revision 33682 built 2025-06-12 browse extensions

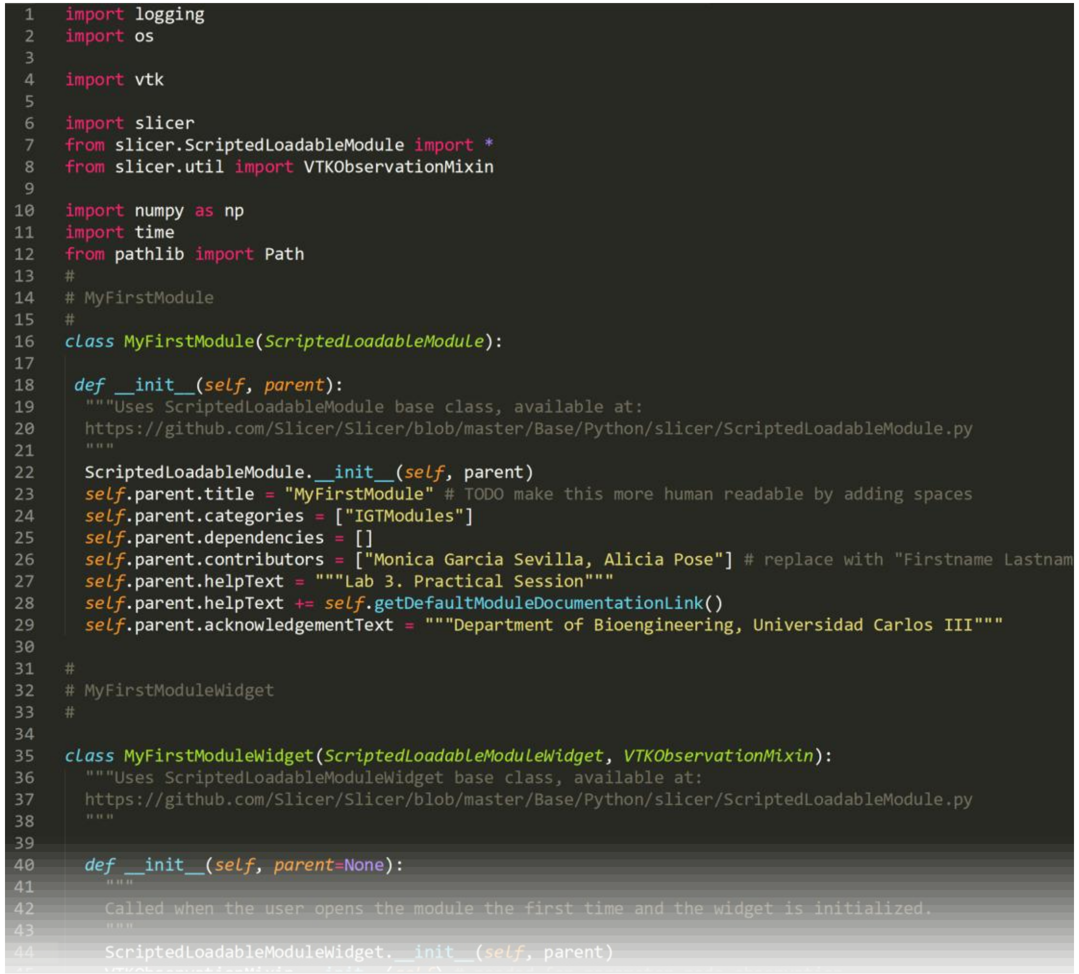
Installation requirements

- Install a suitable code editor:
  - We recommend Visual Studio Code (free, cross-platform)
  - Alternatives: PyCharm (Community), Sublime Text, etc.

Visual Studio Code

PyCharm

Sublime Text



```
import logging
import os

import vtk

import slicer
from slicer.ScriptedLoadableModule import *
from slicer.util import VTKObservationMixin

import numpy as np
import time
from pathlib import Path
#
# MyFirstModule
#
class MyFirstModule(ScriptedLoadableModule):

 def __init__(self, parent):
  """Uses ScriptedLoadableModule base class, available at:
  https://github.com/Slicer/Slicer/blob/master/Base/Python/slicer/ScriptedLoadableModule.py
  """
  ScriptedLoadableModule.__init__(self, parent)
  self.parent.title = "MyFirstModule" # TODO make this more human readable by adding spaces
  self.parent.categories = ["IGTModules"]
  self.parent.dependencies = []
  self.parent.contributors = ["Monica Garcia Sevilla, Alicia Pose"] # replace with "Firstname Lastnam
  self.parent.helpText = """Lab 3. Practical Session"""
  self.parent.helpText += self.getDefaultModuleDocumentationLink()
  self.parent.acknowledgementText = """Department of Bioengineering, Universidad Carlos III"""

#
# MyFirstModuleWidget
#

class MyFirstModuleWidget(ScriptedLoadableModuleWidget, VTKObservationMixin):
  """Uses ScriptedLoadableModuleWidget base class, available at:
  https://github.com/Slicer/Slicer/blob/master/Base/Python/slicer/ScriptedLoadableModule.py
  """

  def __init__(self, parent=None):
    """
    Called when the user opens the module the first time and the widget is initialized.
    """
    ScriptedLoadableModuleWidget.__init__(self, parent)
```

3D Slicer

SOFA

Hands-on: Building your own module in 3D Slicer

Part 1: Create your module

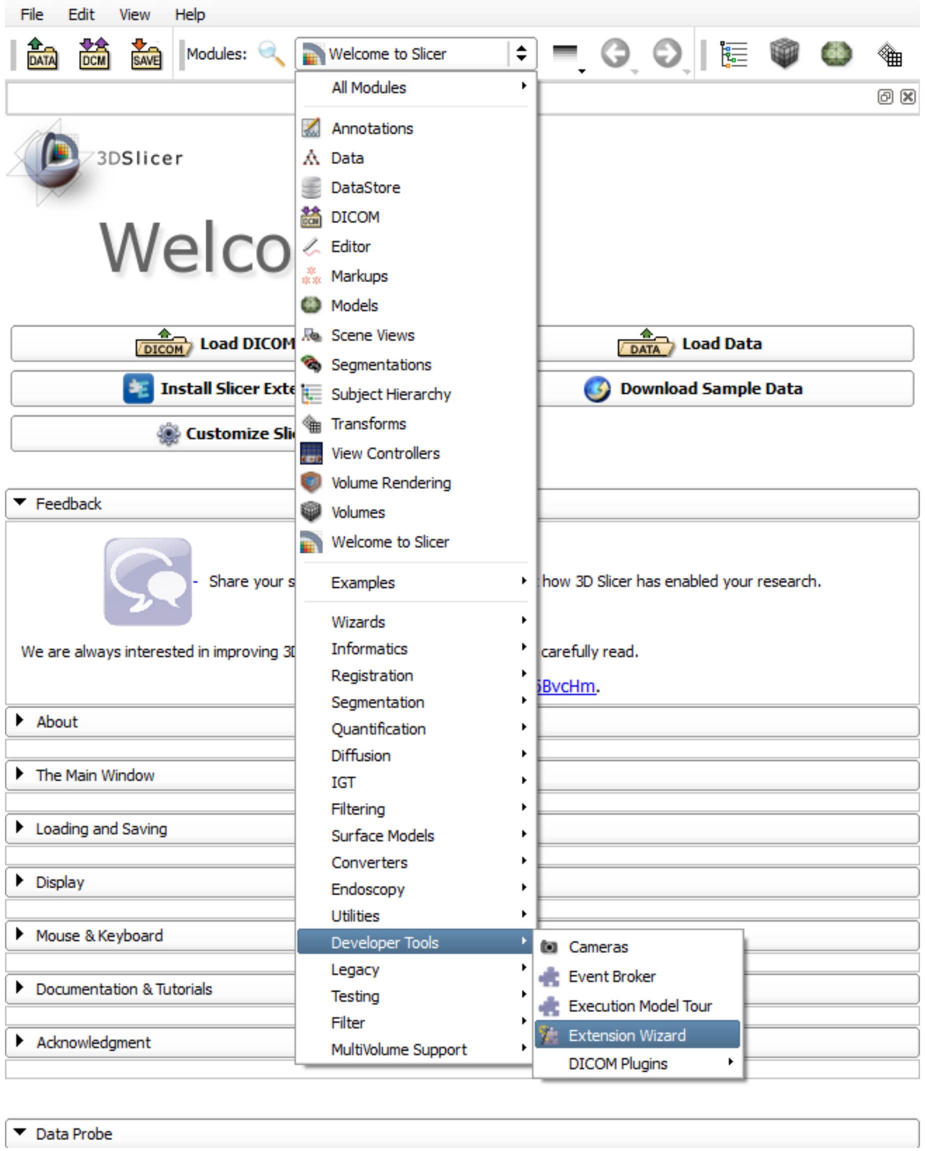
1. Create module

3D Slicer includes a module, **Extension Wizard**, to create modules or extensions.

The initial module code already contains a template with some buttons and functions as an example, making it easier to start.

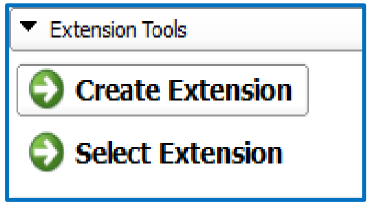
Steps:

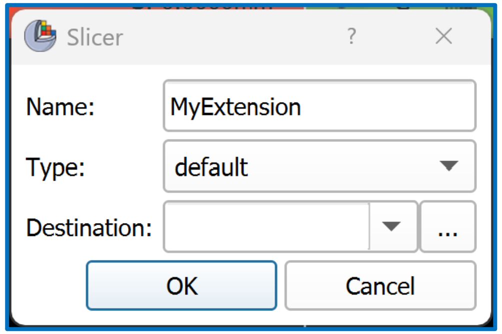
- Open Slicer.
- Click on Modules: “Welcome to Slicer”, select “Developer Tools” from the list and choose the module “Extension Wizard”.



1. Create module

- Click on “Create Extension” and specify the name of your extension. For this workshop you should call it *MyModule.*
- In “Destination” click ‘…’ to choose the directory where you will store the module.
- We recommend creating a folder in disk *C* called *Slicer* and storing the module in→ *C:/Slicer.*
- Click OK.

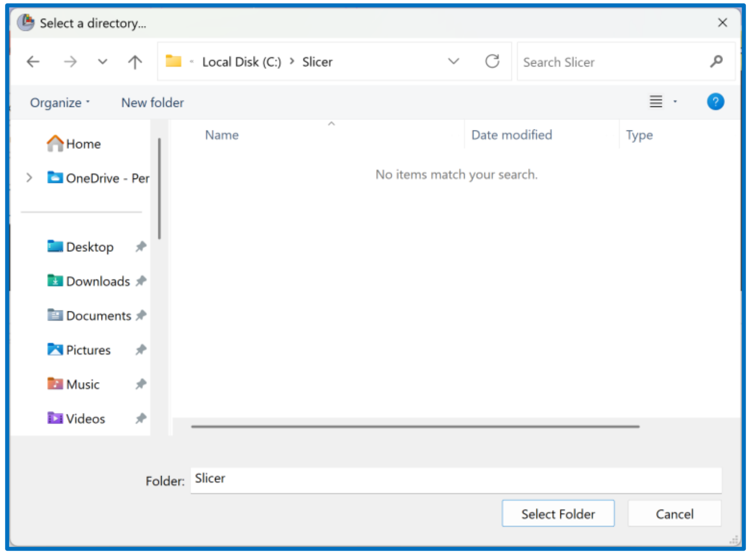




Name: MyExtension

Type: default

Destination:



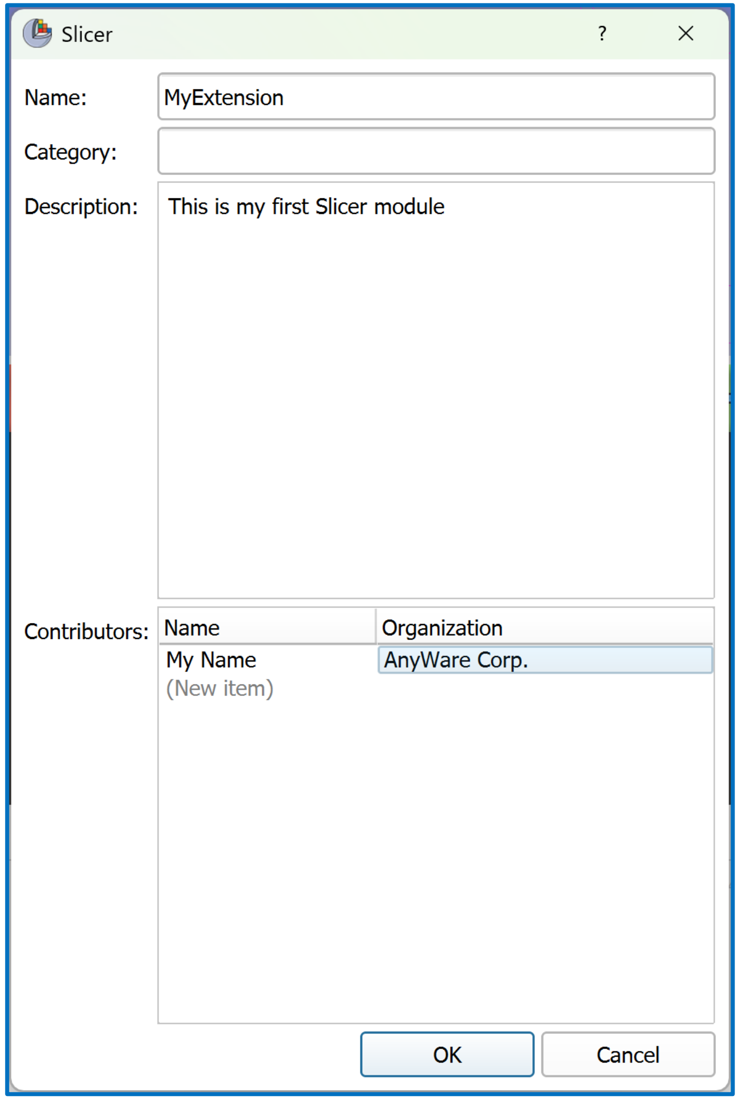
No items match your search.

Folder: Slicer

1. Create module

- You can add extra information about the extension (authors, organization, a description,…).
- Click OK.

IMPORTANT! Never use special characters like accents in paths, code or text (including the description of the module). This will result in errors and the module will not be loaded



Name: MyExtension

Category:

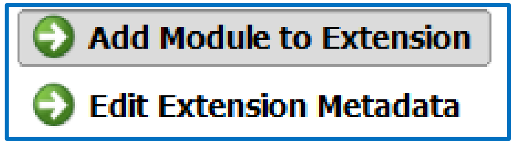
Description: This is my first Slicer module

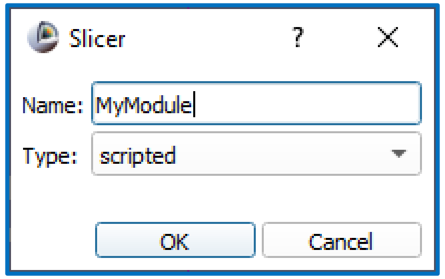
Contributors:

Name	Organization
My Name	AnyWare Corp.
(New item)	

1. Create module

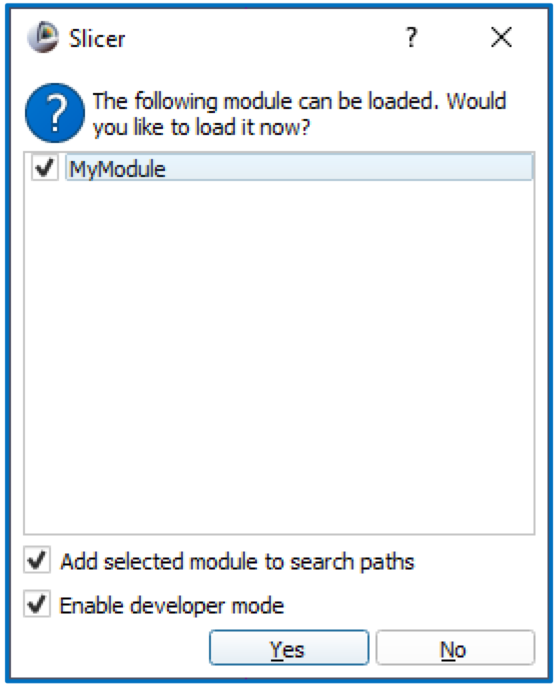
- Select “Add Module to Extension” to create your module.
- Input the name (*MyModule*), in Type select *scripted* and press OK.
- Activate the checkbox “Add selected module to search paths” to load your module every time Slicer is opened.
- The module has now been created!
- However, we will do some additional steps.





Name: MyModule

Type: scripted



The following module can be loaded. Would you like to load it now?

☑ MyModule

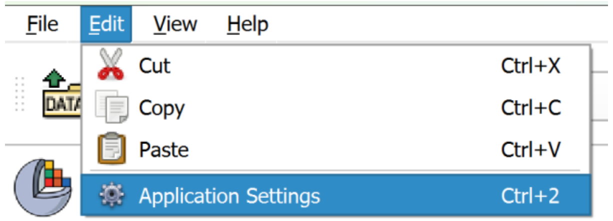
☑ Add selected module to search paths

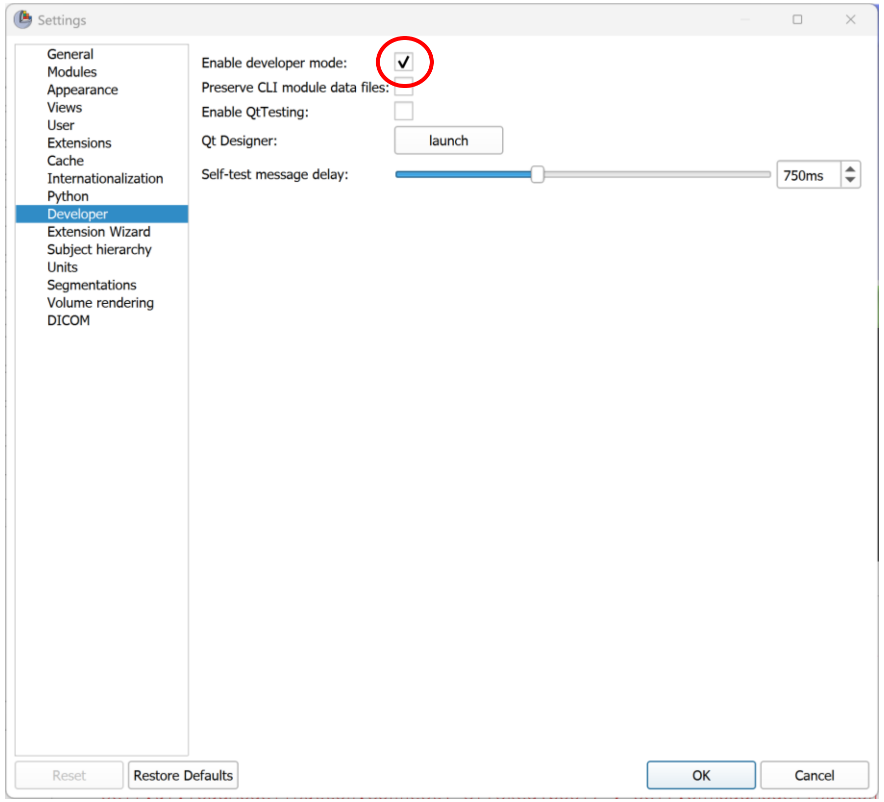
☑ Enable developer mode

1. Create module: developer mode

As we will be using 3D Slicer as developers, we have to enable the developer mode. That way we can access some additional and useful functions.

- Click on Edit → Application Settings.
- Choose "Developer" from the left panel and check "Enable developer mode".
- Click OK and **restart** Slicer.





Enable developer mode: ☑
Preserve CLI module data files: □
Enable QtTesting: □
Qt Designer: launch
Self-test message delay: 750ms

1. Create module: using the template

- If you open the directory where you created your module (*C:/Slicer/MyExtension/MyModule*) you will find inside a file called *MyModule.py.* Open it using your code editor (Visual Studio, Sublime Text,…)
- You will find that there is already some code, which is an initial template generated by Slicer. You could start with this, but to simplify the implementation of the module for this tutorial we will **substitute it by the file *MyModule_TemplateForStudents.py* given to you in this workshop.**
- Close the file in your code editor.

1. Create module: using the template

- Delete the current file *MyModule.py* from your folder and paste the provided one: *MyModule_TemplateForStudents.py*.
- **IMPORTANT:** Change the name of the template file to *MyModule.py* so that it has the name of your module. Otherwise Slicer won't be able to load it.
- Restart Slicer.
- Click on "Welcome to Slicer" drop down menu to show all modules. You will find your module now under "MyExtension" category (the extension you created).
- Open it.
- Now let's take a look at what each section represents…

1. Create module

Show/Hide Python console

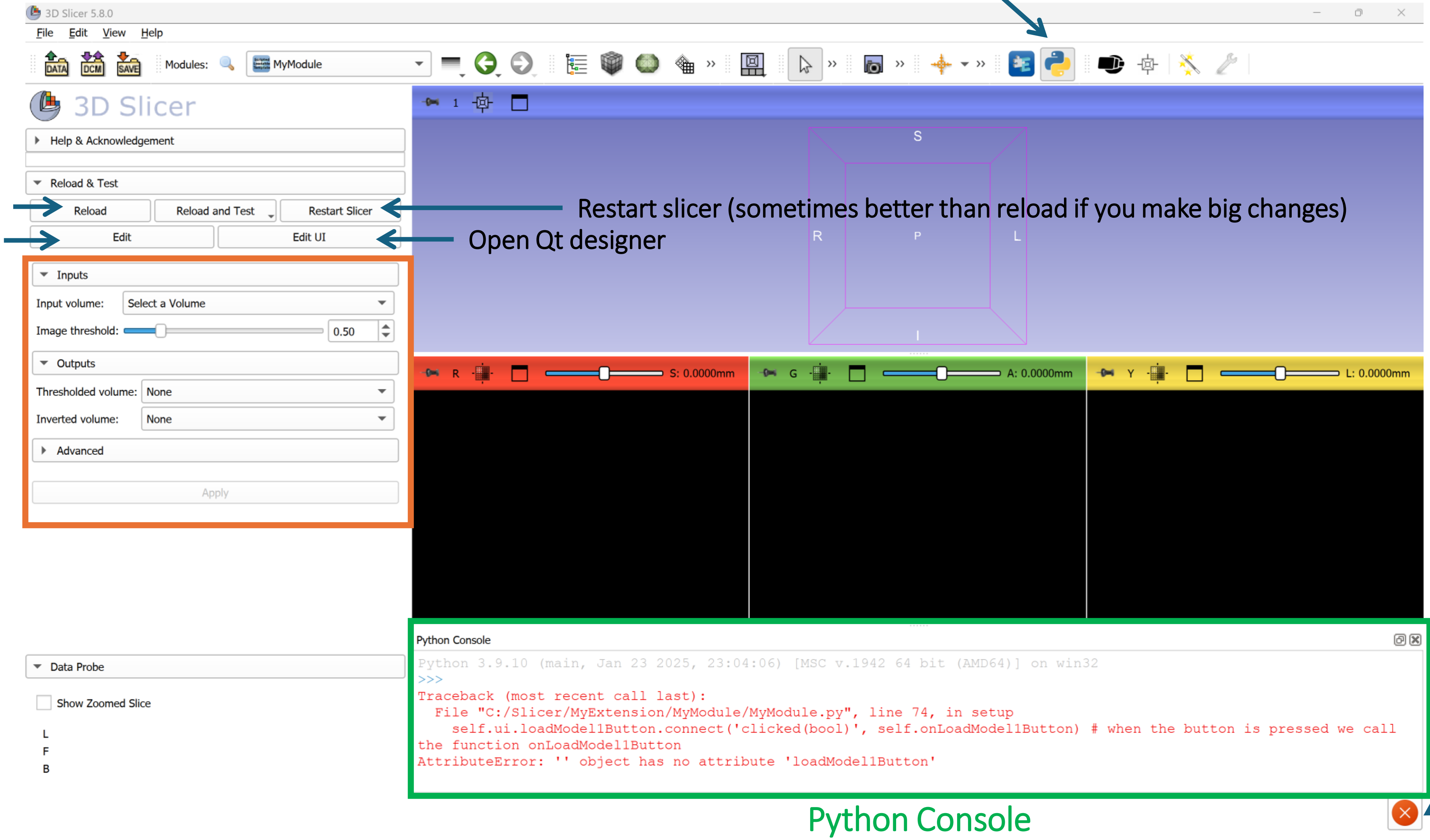
Reload to see module changes

Open code editor

Restart slicer (sometimes better than reload if you make big changes)

Open Qt designer

Your Module



```
Python 3.9.10 (main, Jan 23 2025, 23:04:06) [MSC v.1942 64 bit (AMD64)] on win32
>>>
Traceback (most recent call last):
  File "C:/Slicer/MyExtension/MyModule/MyModule.py", line 74, in setup
    self.ui.loadModel1Button.connect('clicked(bool)', self.onLoadModel1Button) # when the button is pressed we call
the function onLoadModel1Button
AttributeError: '' object has no attribute 'loadModel1Button'
```

Python Console

Error Log

1. Create module: add data

- Open again your module’s folder (*C:/Slicer/MyExtension/MyModule*).
- Inside the Resources directory of your module, create a new folder and call it *Data.*
- Copy the files *transform.h5*, *Model1.stl* and *Model2.stl* (also provided in this workshop) inside the *Data* folder.

Storing your data inside your module’s folders makes it much easier to access it from code.

3D Slicer + SOFA

Part 2: Design UI

2. Design UI

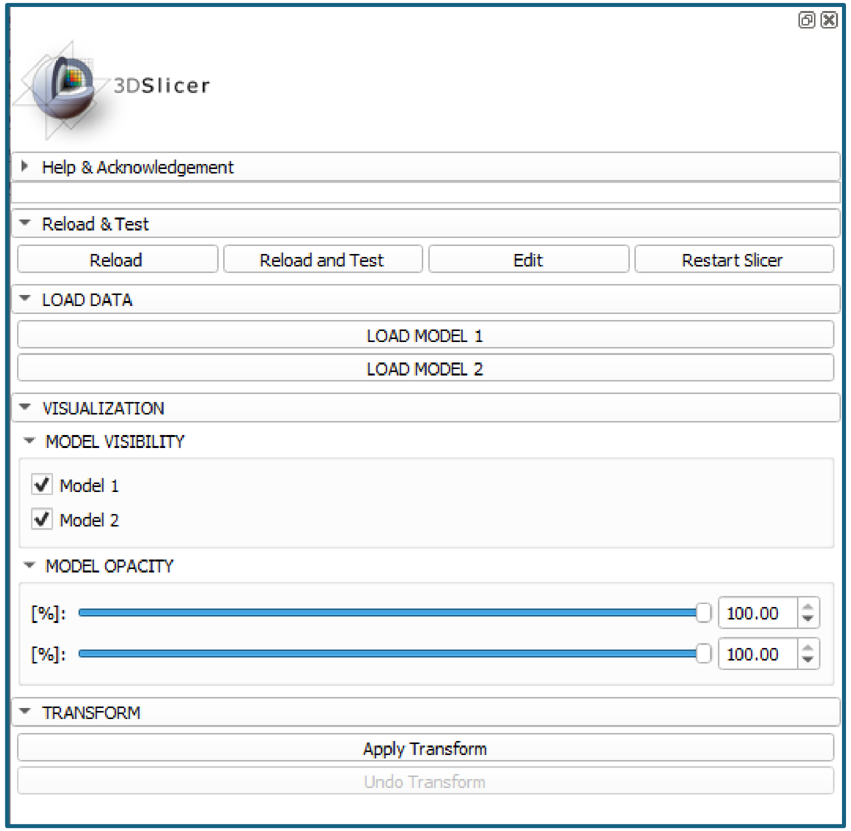
- To start, you have to create this UI with Qt Designer.
- The structure is the following:

  LOAD DATA: ← Collapsible Button
  - Load Model 1 ← Push Button
  - Load Model 2 ← Push Button

  VISUALIZATION: ← Collapsible Button
  - Model Visibility ← ctkCollapsibleGroupBox
    - Show Model 1 ← Check Box
    - Show Model 2 ← Check Box
  - Model Opacity ← ctkCollapsibleGroupBox
    - Change opacity of Model 1 ← ctkSliderWidget
    - Change opacity of Model 2 ← ctkSliderWidget

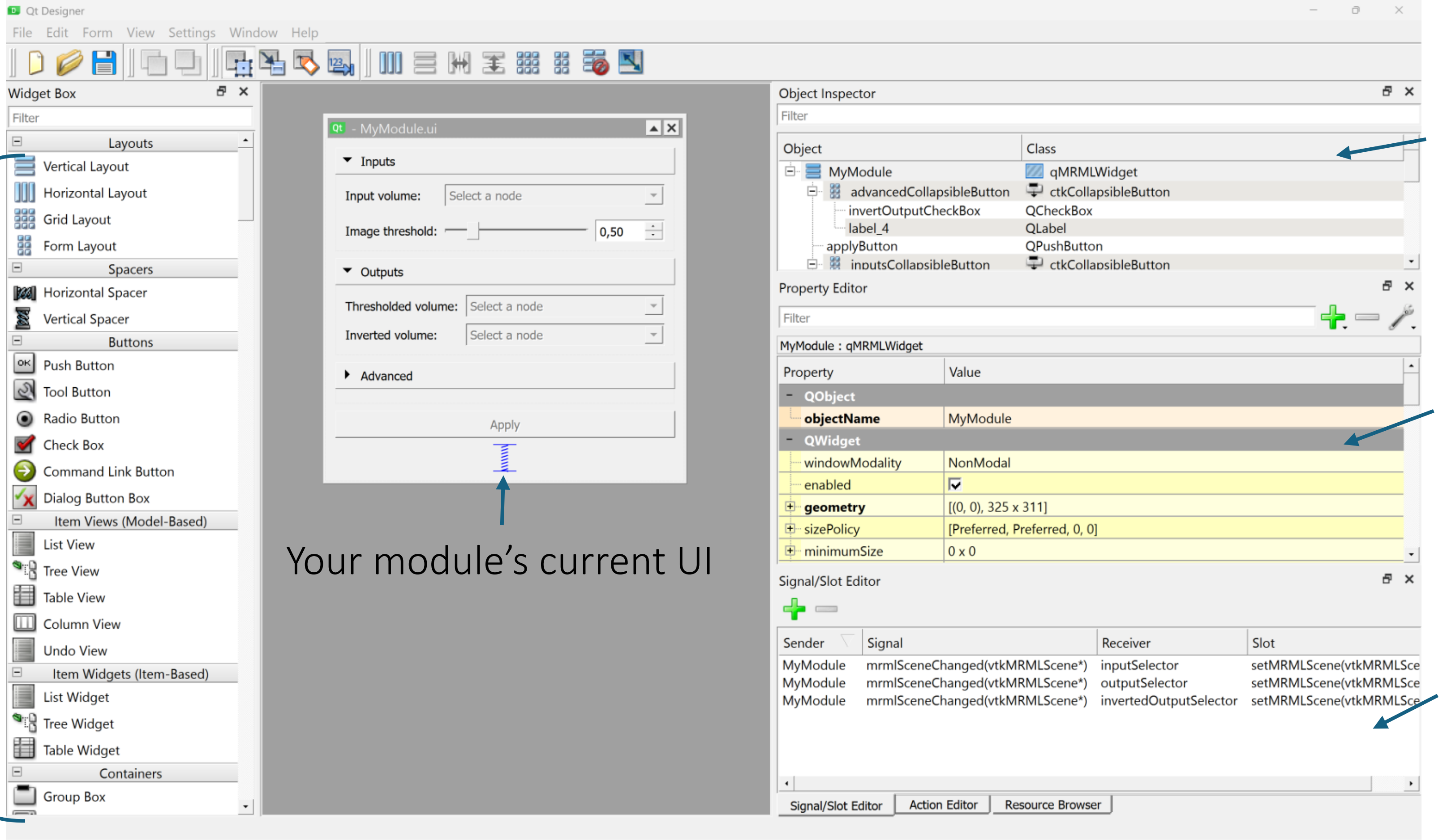
  TRANSFORM: ← Collapsible Button
  - Apply Transform ← Push Button
  - Undo Transform ← Push Button

Expected Final UI



2. Design UI

- Click on the Edit UI button of your module to open Qt Designer.



Object	Class
MyModule	qMRMLWidget
advancedCollapsibleButton	ctkCollapsibleButton
invertOutputCheckBox	QCheckBox
label_4	QLabel
applyButton	QPushButton
inputsCollapsibleButton	ctkCollapsibleButton

Property	Value
QObject	
objectName	MyModule
QWidget	
windowModality	NonModal
enabled	☑
geometry	[(0, 0), 325 x 311]
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0

Sender	Signal	Receiver	Slot
MyModule	mrmlSceneChanged(vtkMRMLScene*)	inputSelector	setMRMLScene(vtkMRMLSce
MyModule	mrmlSceneChanged(vtkMRMLScene*)	outputSelector	setMRMLScene(vtkMRMLSce
MyModule	mrmlSceneChanged(vtkMRMLScene*)	invertedOutputSelector	setMRMLScene(vtkMRMLSce

Your module's current UI

Widget Box:

contains all the available Qt widgets that you can use to build your interface.
Widgets are organized by category (e.g., Buttons, Display Widgets, Input Widgets).
You can drag and drop widgets from the Widget Box directly into your UI layout.

Object Inspector:

Shows the hierarchical structure of your widgets in your interface. It allows you to view and select UI elements based on their parent-child relationships, making it easier to organize and manage the layout of your module.

Property Editor:

Allows you to view and modify the properties of the selected widget.
These properties include object names, geometry, fonts, colors, tooltips, visibility, and more.

Signal/Slot Editor:

enables you to create connections between signals (such as a button click) and slots (methods that handle the action).
It provides a graphical way to link user interactions with specific behaviors in your application, without needing to write code manually.

2. Design UI

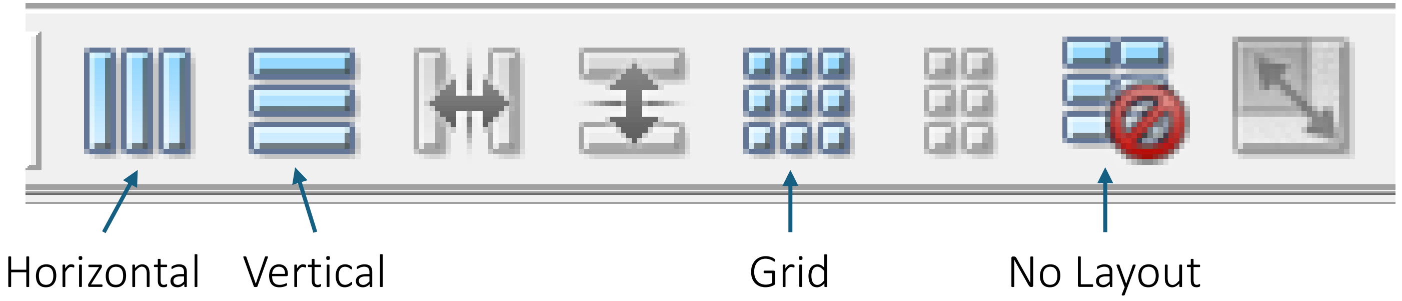
- Try to replicate the expected UI by adding the corresponding widgets.
- Start by changing the names of the 3 collapsible buttons (Inputs, Outputs, Advanced) to the expected ones (LOAD DATA, VISUALIZATION, TRANSFORM). You can do this by double clicking on the collapsible or looking for the “text” field in the Property Editor after clicking on it.
- For the TRANSFORM collapsible button, make sure you uncheck the collapsed option from the Property Editor.
- Then drag and drop the corresponding elements from the Widget Box to the UI and remove the original ones. Input the correct text in each widget.

NOTE: Please refer to slide 23 to check each widget type.

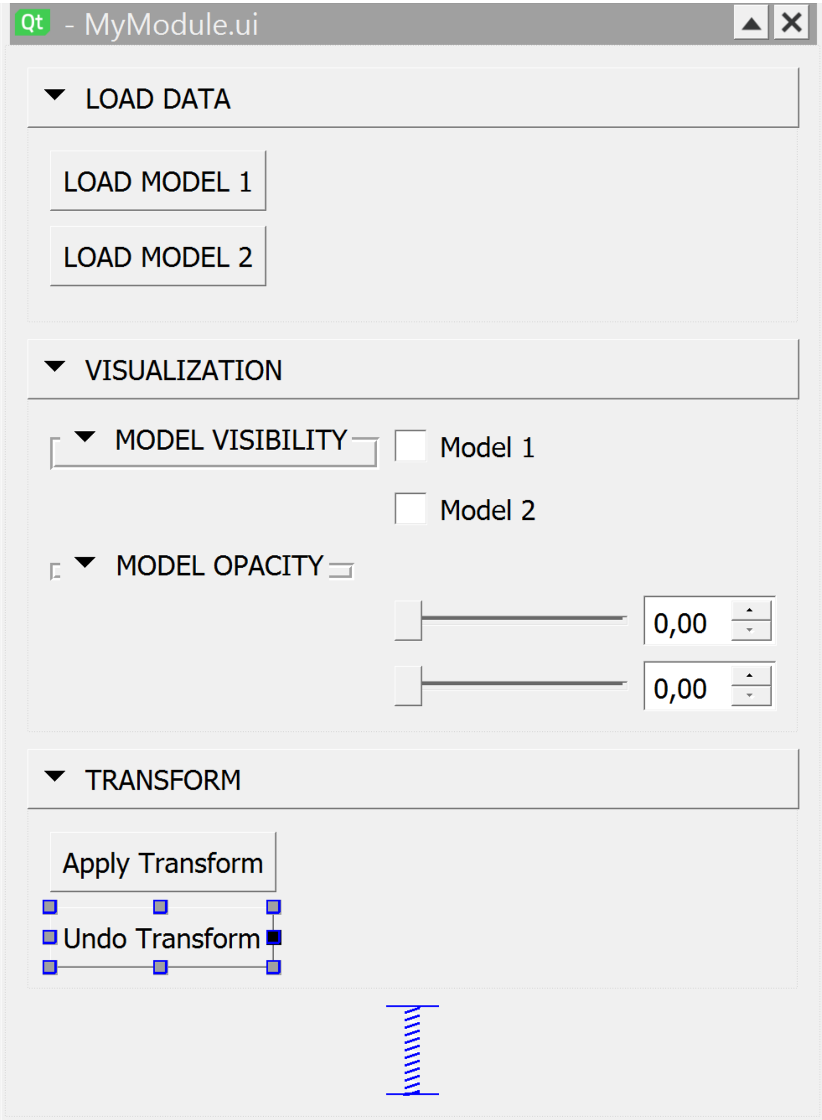
IMPORTANT: Don’t worry about the layout for now, we will arrange everything in the next step.

2. Design UI

- You might have something like this now →
- To arrange the widget elements, click on every collapsible button and choose the correct layout from the top menu:



- For the ctkCollapsibleGroupBox widgets, drag and drop at least one element on top of them to be able to select the layout.



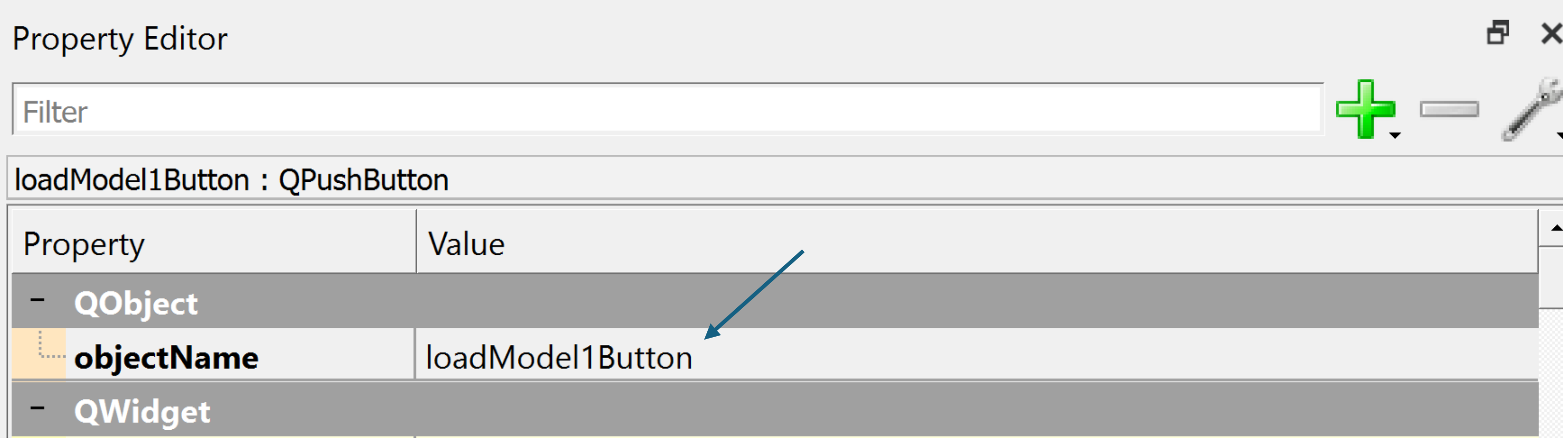
2. Design UI

Now we are going to give names to every widget element to be able to access them from code later.

- Select each widget, go to the Property Editor and input a name in *objectName*.

We suggest you use the following names, as the template code already uses them:

- loadModel1Button
- loadModel2Button
- model1_checkBox
- model2_checkBox
- opacityValueSliderWidget_1
- opacityValueSliderWidget_2
- applyTransformButton
- undoTransformButton



Property Editor

Filter

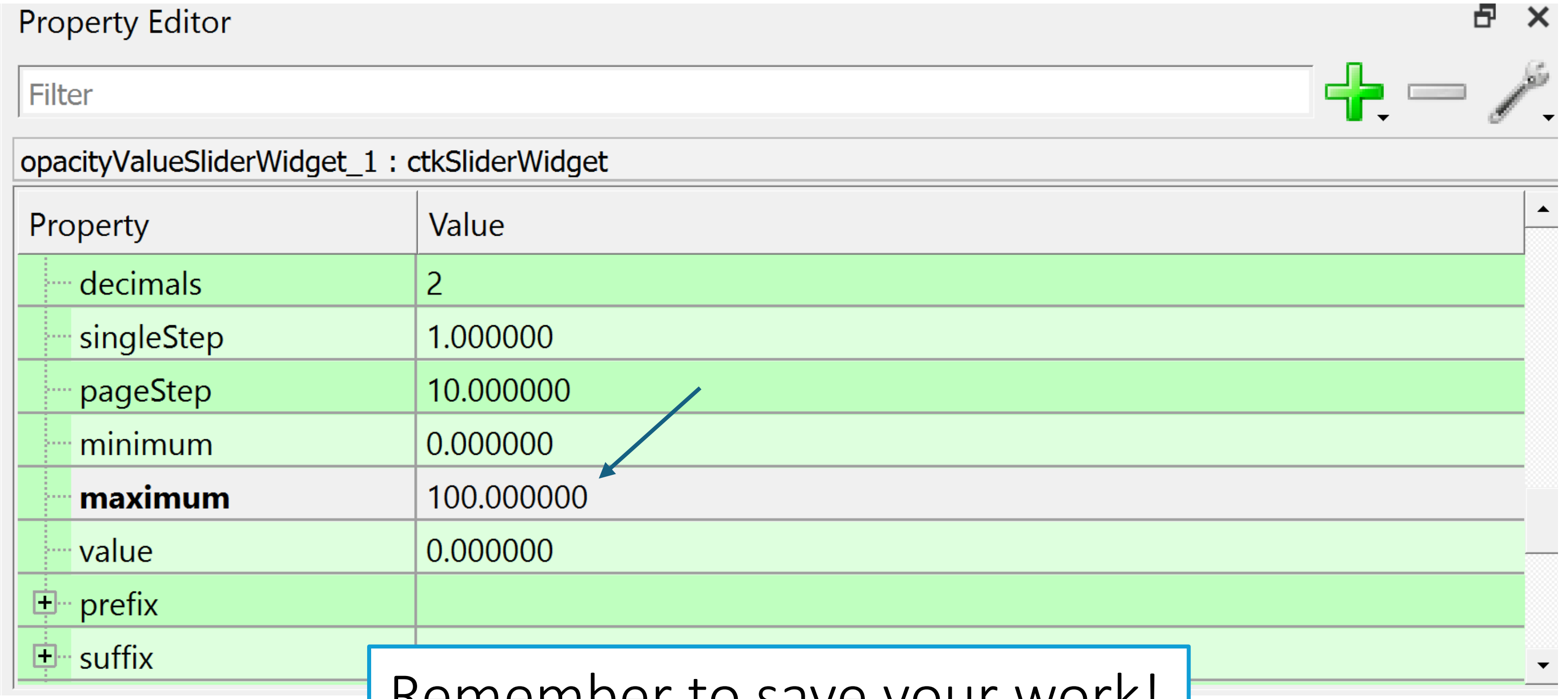
loadModel1Button : QPushButton

Property	Value
QObject	
objectName	loadModel1Button
QWidget	

2. Design UI

Set the maximum value of the ctkSliderWidgets to 100. The minimum should already be 0.

- To do this, select them and look for the field *maximum* in the Property Editor.



Property Editor

Filter

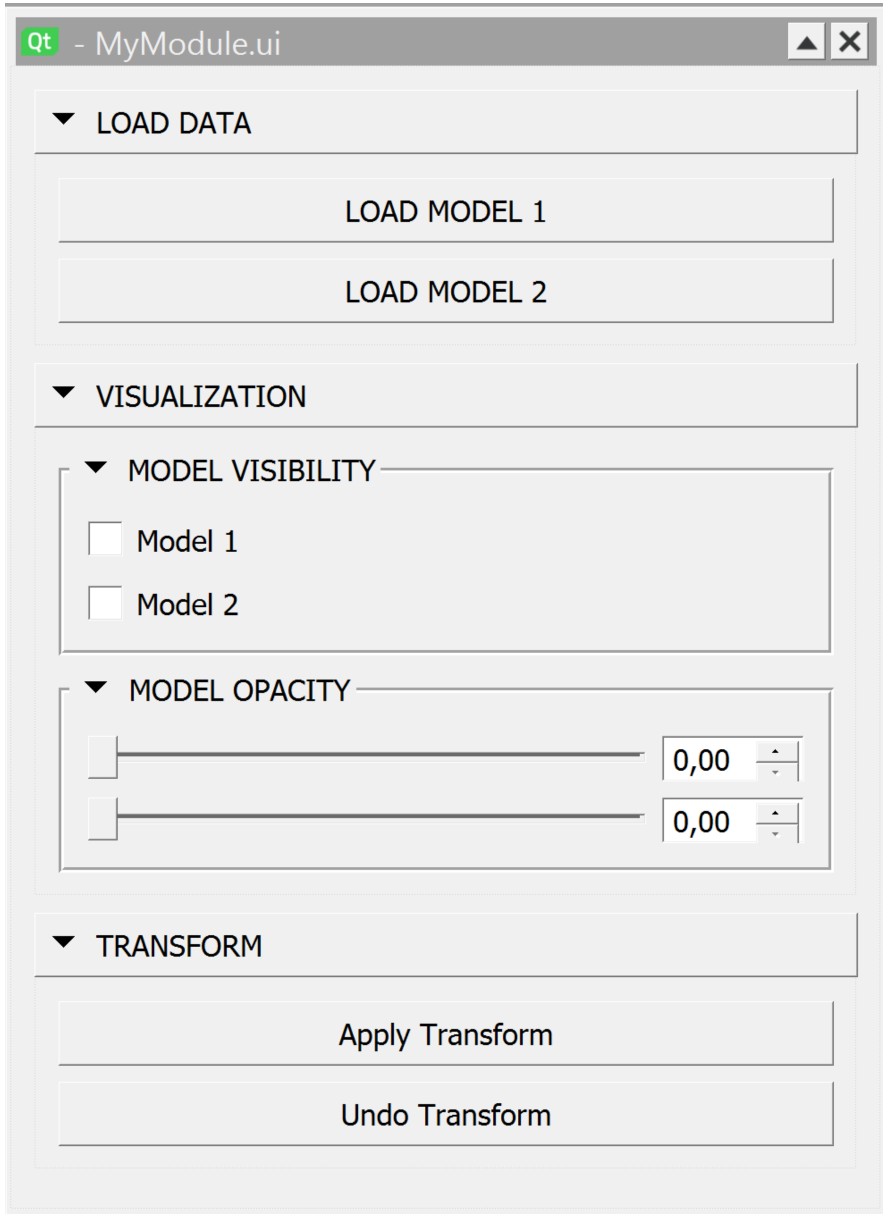
opacityValueSliderWidget_1 : ctkSliderWidget

Property	Value
decimals	2
singleStep	1.000000
pageStep	10.000000
minimum	0.000000
maximum	100.000000
value	0.000000
⊞ prefix	
⊞ suffix	

Remember to save your work!

2. Design UI

- Expected result:



Part 3: Python review

3. Python review

For this tutorial, we asume you have some experience in programming.

If you have experience with Python language you can move to “Part 4: Code implementation”. If not, the following slides review some key concepts you need to know before starting.

You can also follow this link: https://www.geeksforgeeks.org/statement-indentation-and-comment-in-python/

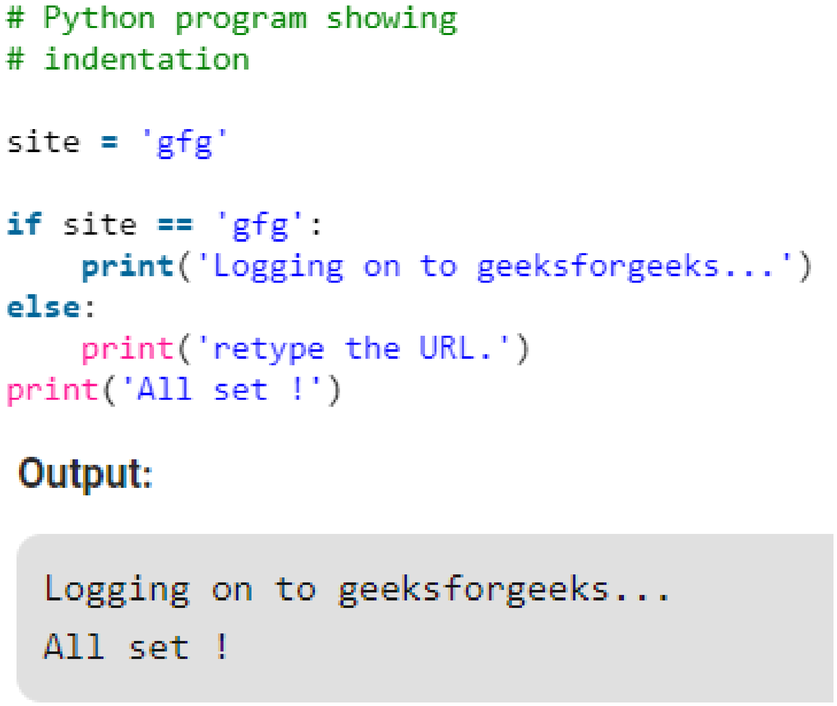
3. Python review

Indentation

Most programming languages like Java, C or C++ use braces {} to define a block code (a for loop, a function,…).

In Python, indentation is used to identify blocks. The block will start after “:” and an indentation and will end with the first unindented line.

You can either indent using two spaces or four, but you should choose one and be consistent.



```python
# Python program showing
# indentation

site = 'gfg'

if site == 'gfg':
    print('Logging on to geeksforgeeks...')
else:
    print('retype the URL.')
print('All set !')
```

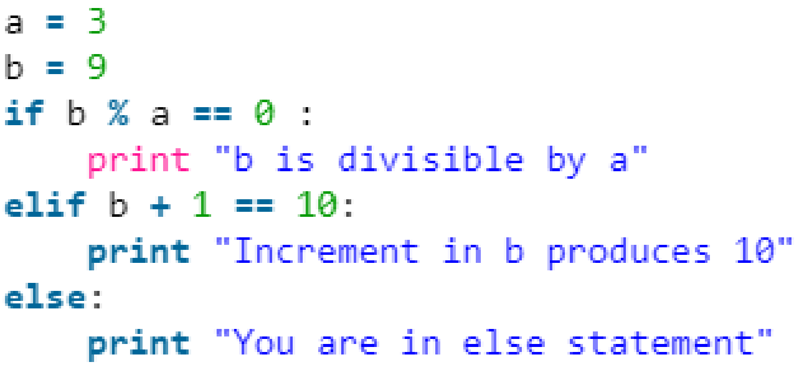
Output:

```
Logging on to geeksforgeeks...
All set !
```

3. Python review

- **Conditions**

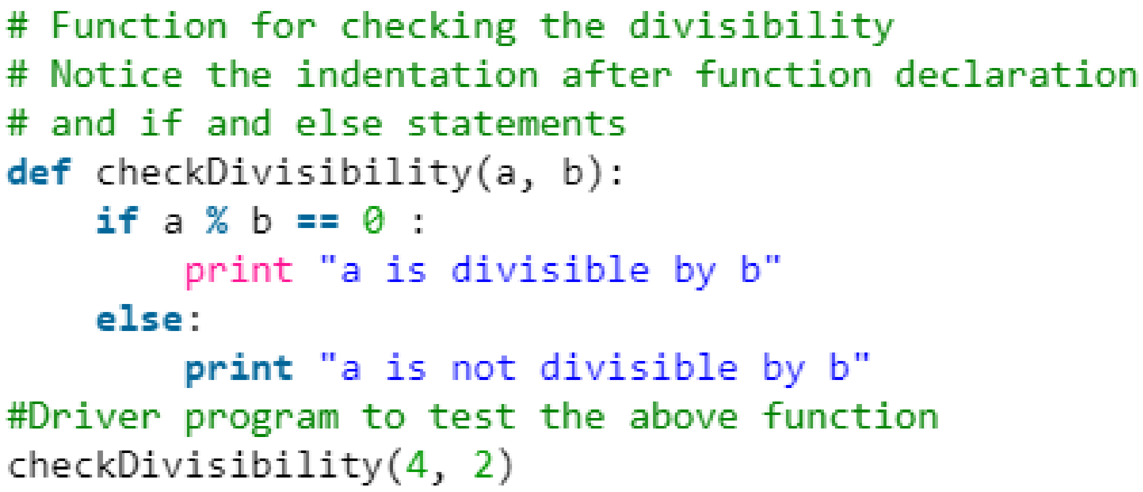
If, elif and else:



```
a = 3
b = 9
if b % a == 0 :
    print "b is divisible by a"
elif b + 1 == 10:
    print "Increment in b produces 10"
else:
    print "You are in else statement"
```

- **Functions**

def *myFunction*(param1, …, paramN):

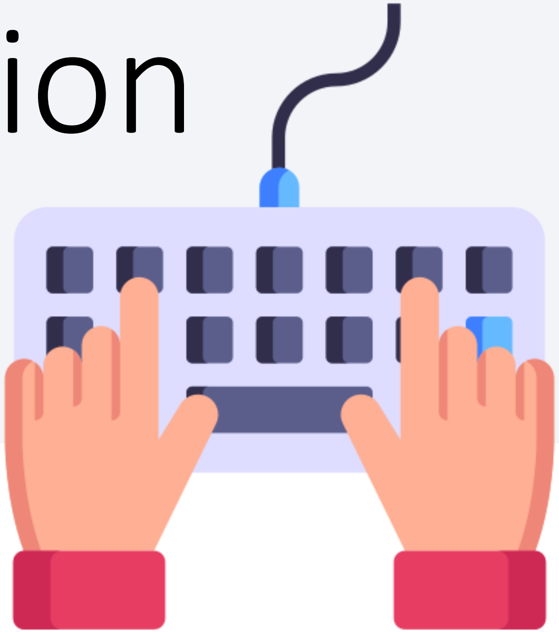


```
# Function for checking the divisibility
# Notice the indentation after function declaration
# and if and else statements
def checkDivisibility(a, b):
    if a % b == 0 :
        print "a is divisible by b"
    else:
        print "a is not divisible by b"
#Driver program to test the above function
checkDivisibility(4, 2)
```

3D Slicer

SOFA

Part 4: Code implementation

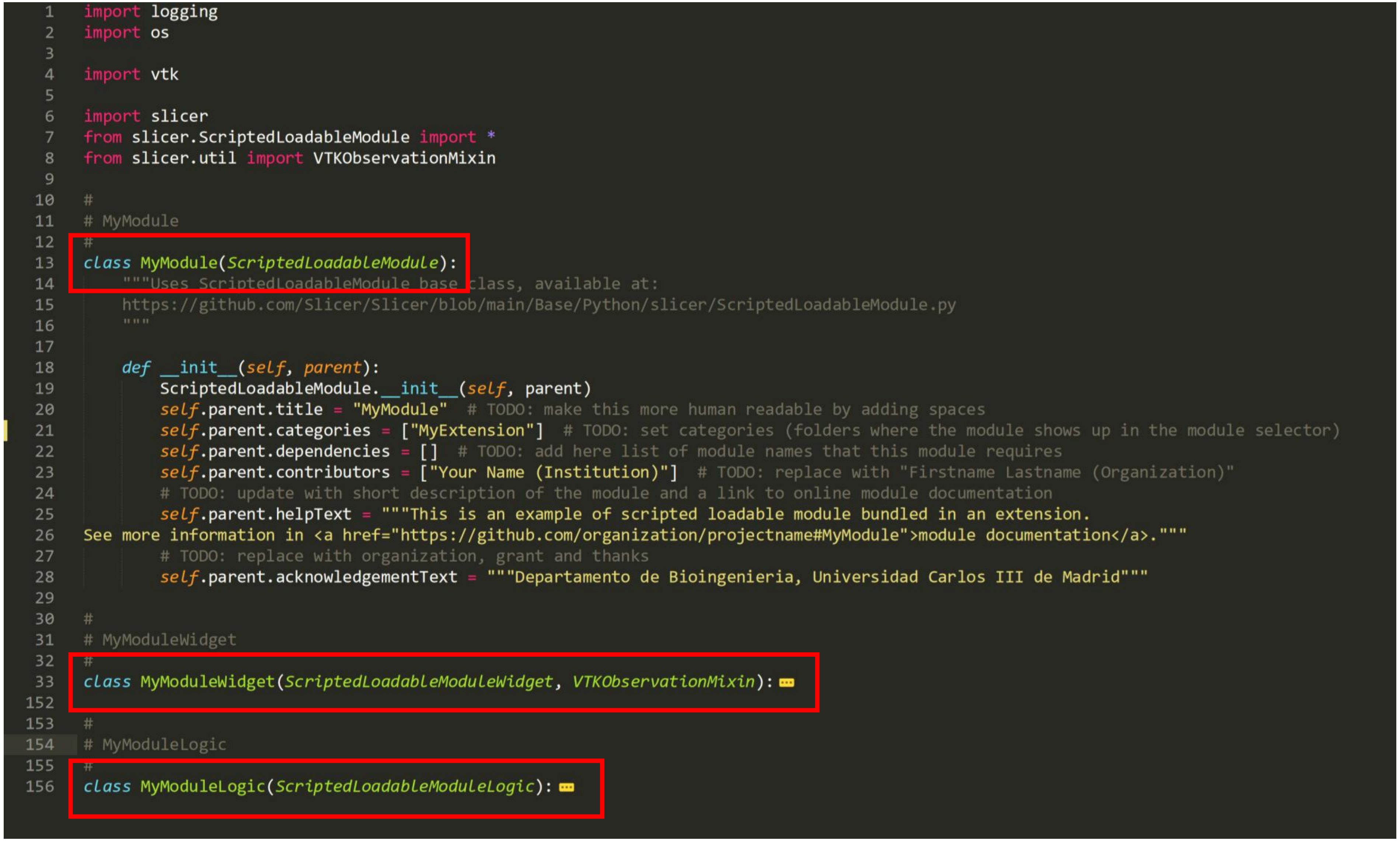


Before we start coding…

Before starting with the implementation of our module, let’s remember how the code is structured:

- Every module is composed of three classes:
  - ***Module***: Contains metadata of your module (categories, contributors, etc.)
  - ***Widget***: Defines the graphical interface (buttons, layouts,…) and captures the interaction of the user with these items.
  - ***Logic***: Defines the logic of the module. For example, it defines what action should be performed when a certain button is clicked. The corresponding function of the button in the widget will capture the interaction and call some function inside logic to do a certain action.
- Open your module’s code file (*MyModule.py*) and check this.

Before we start coding…



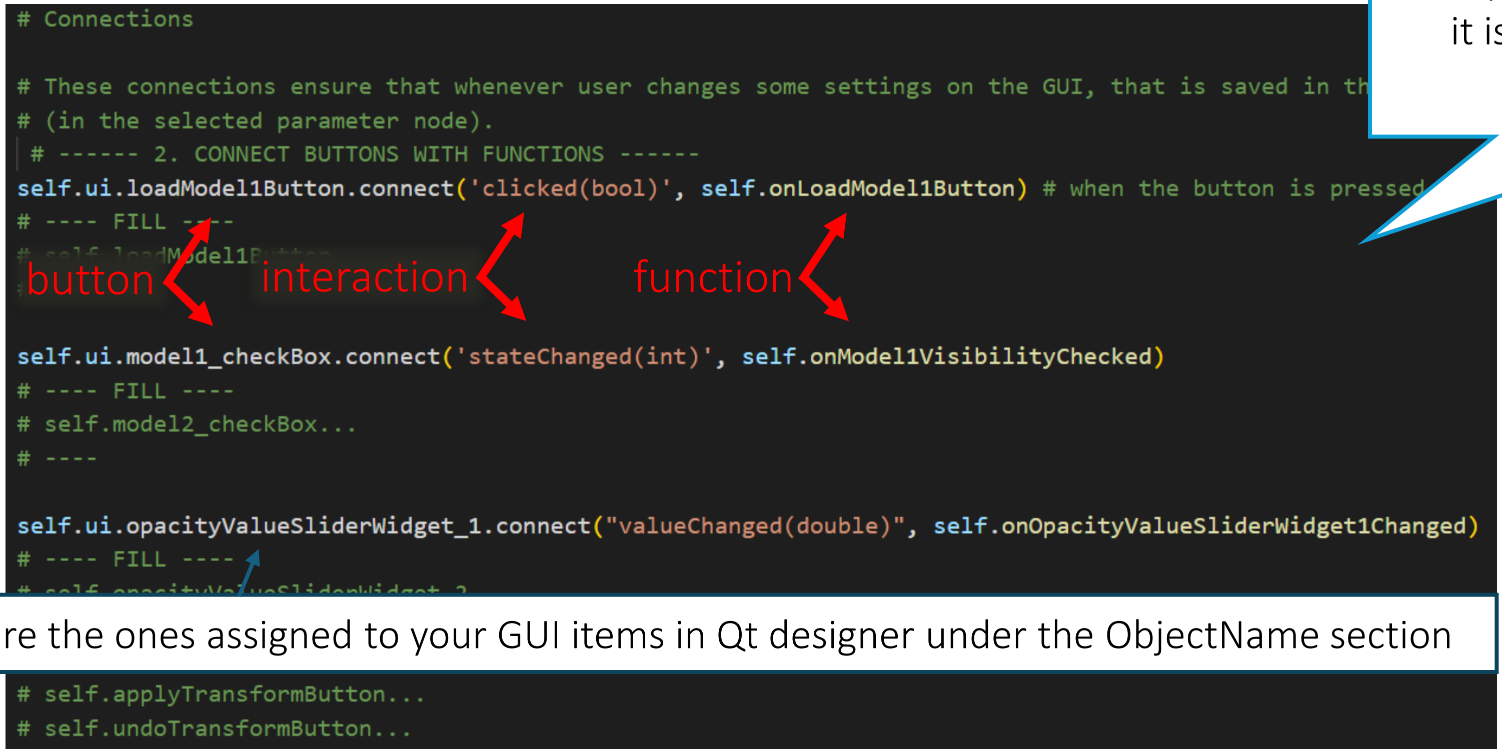
```python
import logging
import os

import vtk

import slicer
from slicer.ScriptedLoadableModule import *
from slicer.util import VTKObservationMixin

#
# MyModule
#
class MyModule(ScriptedLoadableModule):
    """Uses ScriptedLoadableModule base class, available at:
    https://github.com/Slicer/Slicer/blob/main/Base/Python/slicer/ScriptedLoadableModule.py
    """

    def __init__(self, parent):
        ScriptedLoadableModule.__init__(self, parent)
        self.parent.title = "MyModule"  # TODO: make this more human readable by adding spaces
        self.parent.categories = ["MyExtension"]  # TODO: set categories (folders where the module shows up in the module selector)
        self.parent.dependencies = []  # TODO: add here list of module names that this module requires
        self.parent.contributors = ["Your Name (Institution)"]  # TODO: replace with "Firstname Lastname (Organization)"
        # TODO: update with short description of the module and a link to online module documentation
        self.parent.helpText = """This is an example of scripted loadable module bundled in an extension.
See more information in <a href="https://github.com/organization/projectname#MyModule">module documentation</a>."""
        # TODO: replace with organization, grant and thanks
        self.parent.acknowledgementText = """Departamento de Bioingenieria, Universidad Carlos III de Madrid"""

#
# MyModuleWidget
#
class MyModuleWidget(ScriptedLoadableModuleWidget, VTKObservationMixin):⋯

#
# MyModuleLogic
#
class MyModuleLogic(ScriptedLoadableModuleLogic):⋯
```

4. Code implementation: widget

Buttons, sliders and other elements of the interface are usually assigned a function that is executed when the user interacts with it.

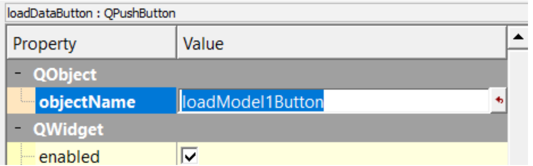
This assignment is performed as shown below:

The type of interaction is associated with the type of widget you add. For buttons it is ‘clicked(bool)’, for check boxes ‘stateChanged(int)’,…



```
# Connections

# These connections ensure that whenever user changes some settings on the GUI, that is saved in th
# (in the selected parameter node).
 # ------ 2. CONNECT BUTTONS WITH FUNCTIONS ------
self.ui.loadModel1Button.connect('clicked(bool)', self.onLoadModel1Button) # when the button is pressed
# ---- FILL ----
# self.loadModel1Button
#

self.ui.model1_checkBox.connect('stateChanged(int)', self.onModel1VisibilityChecked)
# ---- FILL ----
# self.model2_checkBox...
# ----

self.ui.opacityValueSliderWidget_1.connect("valueChanged(double)", self.onOpacityValueSliderWidget1Changed)
# ---- FILL ----
# self.opacityValueSliderWidget_2

# self.applyTransformButton...
# self.undoTransformButton...
```

These names are the ones assigned to your GUI items in Qt designer under the ObjectName section

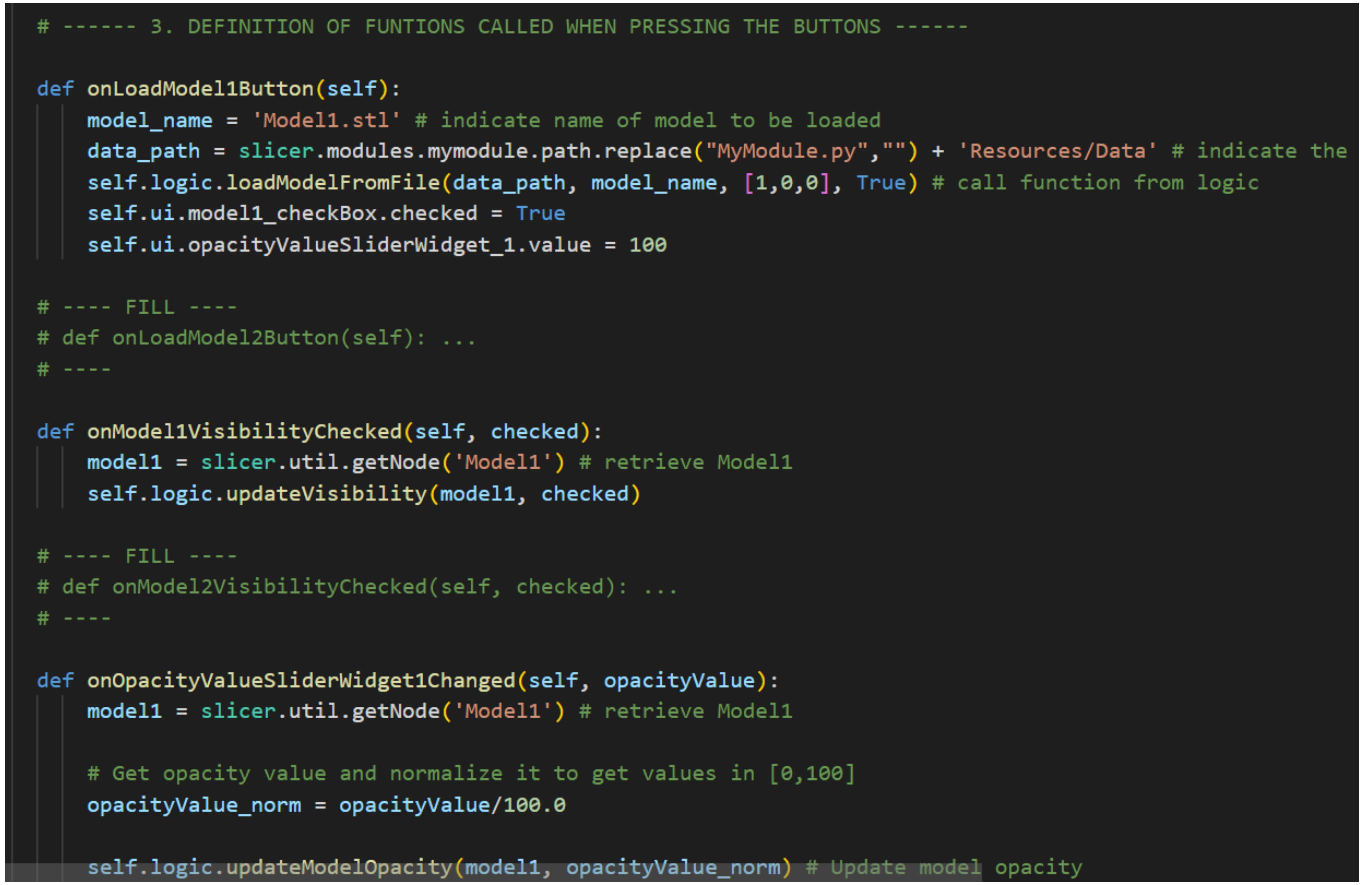


loadDataButton : QPushButton

Property	Value
- QObject	
objectName	loadModel1Button
- QWidget	
enabled	☑

4. Code implementation: widget

In this section of the code you see the definition of all functions called when an interaction with a button (or other interface element) is performed.

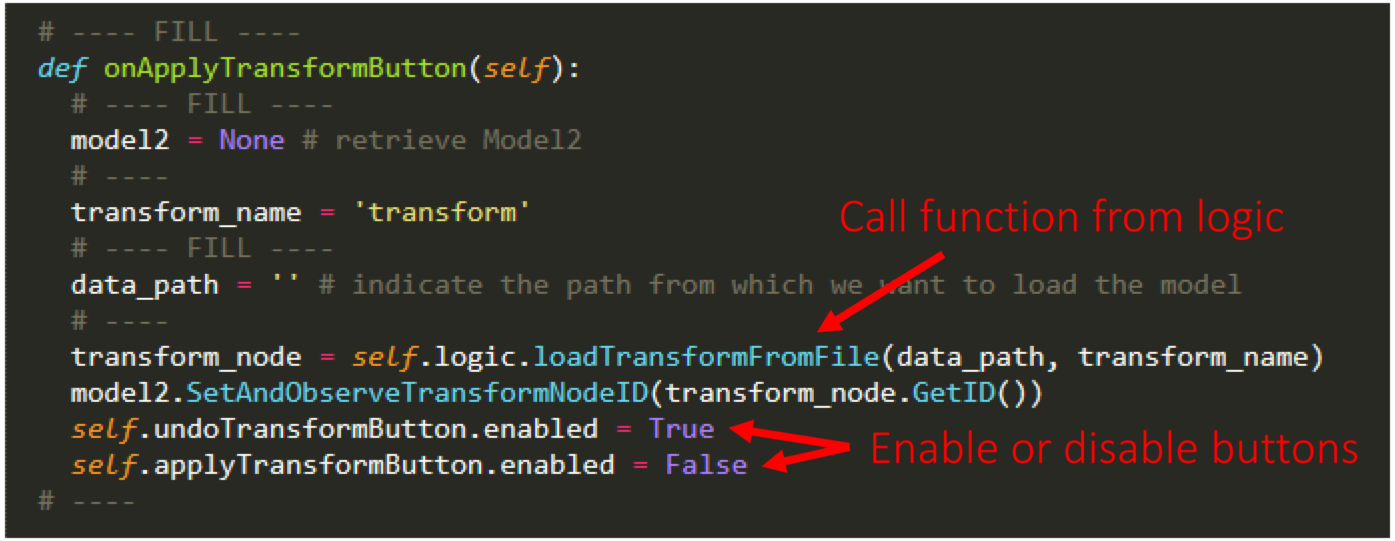


```python
# ------ 3. DEFINITION OF FUNTIONS CALLED WHEN PRESSING THE BUTTONS ------

def onLoadModel1Button(self):
    model_name = 'Model1.stl' # indicate name of model to be loaded
    data_path = slicer.modules.mymodule.path.replace("MyModule.py","") + 'Resources/Data' # indicate the
    self.logic.loadModelFromFile(data_path, model_name, [1,0,0], True) # call function from logic
    self.ui.model1_checkBox.checked = True
    self.ui.opacityValueSliderWidget_1.value = 100

# ---- FILL ----
# def onLoadModel2Button(self): ...
# ----

def onModel1VisibilityChecked(self, checked):
    model1 = slicer.util.getNode('Model1') # retrieve Model1
    self.logic.updateVisibility(model1, checked)

# ---- FILL ----
# def onModel2VisibilityChecked(self, checked): ...
# ----

def onOpacityValueSliderWidget1Changed(self, opacityValue):
    model1 = slicer.util.getNode('Model1') # retrieve Model1

    # Get opacity value and normalize it to get values in [0,100]
    opacityValue_norm = opacityValue/100.0

    self.logic.updateModelOpacity(model1, opacityValue_norm) # Update model opacity
```

4. Code implementation: widget

These functions:

- modify everything directly related with the interface (enable buttons, hide or show a collapsible layout, etc.)
- Call functions from the logic.



```
# ---- FILL ----
def onApplyTransformButton(self):
  # ---- FILL ----
  model2 = None # retrieve Model2
  # ----
  transform_name = 'transform'
  # ---- FILL ----
  data_path = '' # indicate the path from which we want to load the model
  # ----
  transform_node = self.logic.loadTransformFromFile(data_path, transform_name)
  model2.SetAndObserveTransformNodeID(transform_node.GetID())
  self.undoTransformButton.enabled = True
  self.applyTransformButton.enabled = False
# ----
```

Call function from logic

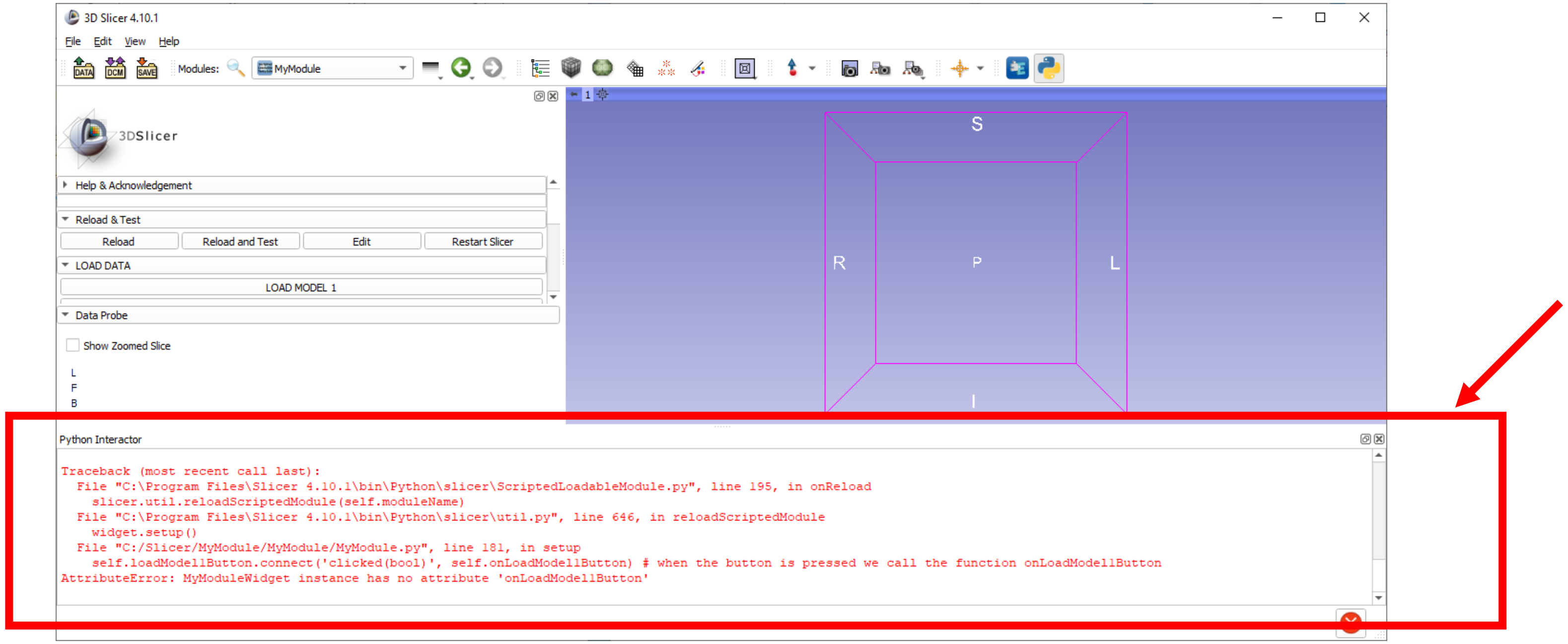
Enable or disable buttons

HINT

While you are implementing code, it is a good idea to check the result every time you finish a step (definition of a function, creation of a button,...)

For that, you can *Reload* or *Restart* Slicer

If your code has errors, they will appear in the Python console



```
Traceback (most recent call last):
  File "C:\Program Files\Slicer 4.10.1\bin\Python\slicer\ScriptedLoadableModule.py", line 195, in onReload
    slicer.util.reloadScriptedModule(self.moduleName)
  File "C:\Program Files\Slicer 4.10.1\bin\Python\slicer\util.py", line 646, in reloadScriptedModule
    widget.setup()
  File "C:/Slicer/MyModule/MyModule/MyModule.py", line 181, in setup
    self.loadModel1Button.connect('clicked(bool)', self.onLoadModel1Button) # when the button is pressed we call the function onLoadModel1Button
AttributeError: MyModuleWidget instance has no attribute 'onLoadModel1Button'
```

4. Code implementation: widget

- Now that you know all this…

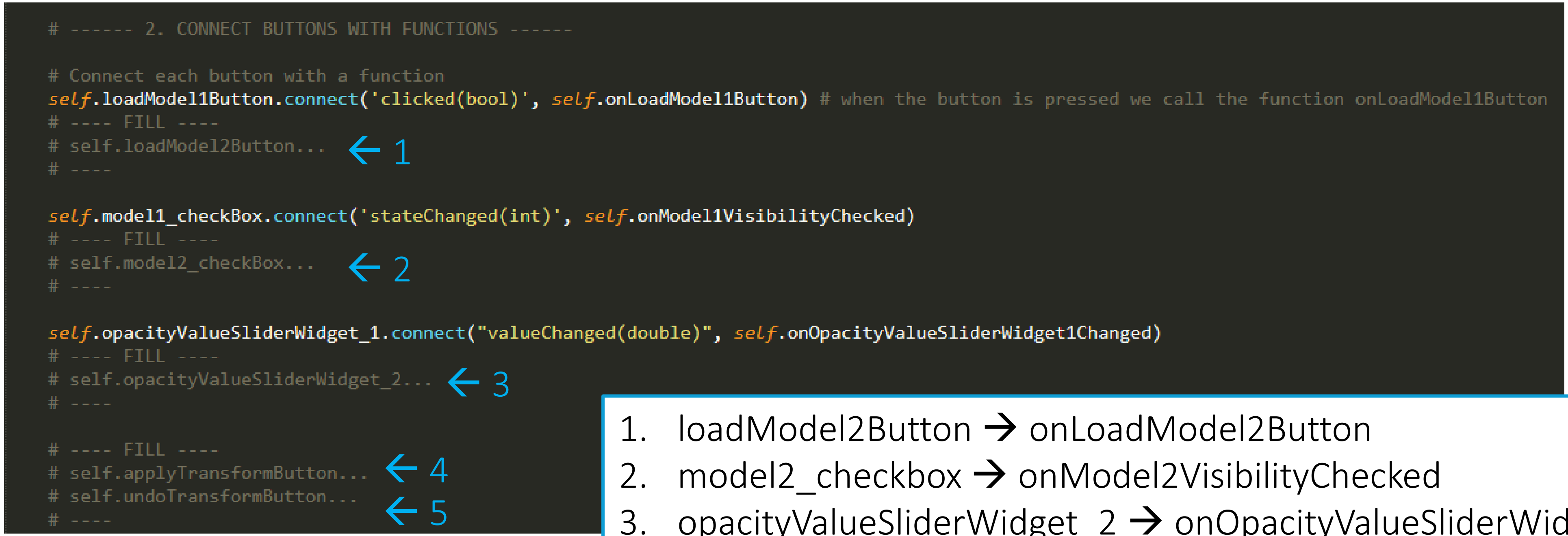
Let’s start implementing our module!

The sections of the code you have to fill are indicated the following way:

```
---- FILL ----
Code to implement
----
```

4. Code implementation: widget

We’ll start with the connections. You have already some of them in the template. Connect the remaining buttons with the following functions:

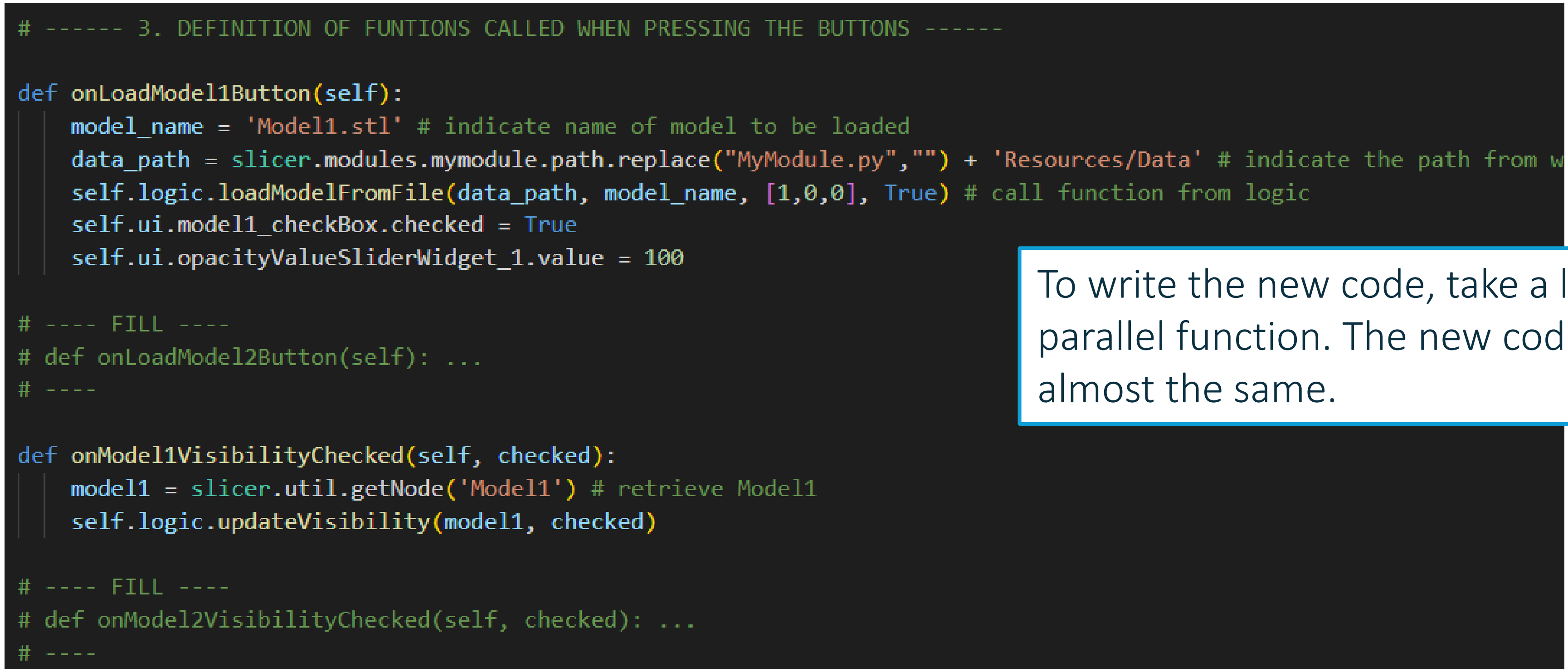


```
# ------ 2. CONNECT BUTTONS WITH FUNCTIONS ------

# Connect each button with a function
self.loadModel1Button.connect('clicked(bool)', self.onLoadModel1Button) # when the button is pressed we call the function onLoadModel1Button
# ---- FILL ----
# self.loadModel2Button...
# ----

self.model1_checkBox.connect('stateChanged(int)', self.onModel1VisibilityChecked)
# ---- FILL ----
# self.model2_checkBox...
# ----

self.opacityValueSliderWidget_1.connect("valueChanged(double)", self.onOpacityValueSliderWidget1Changed)
# ---- FILL ----
# self.opacityValueSliderWidget_2...
# ----

# ---- FILL ----
# self.applyTransformButton...
# self.undoTransformButton...
# ----
```

1. loadModel2Button → onLoadModel2Button
2. model2_checkbox → onModel2VisibilityChecked
3. opacityValueSliderWidget_2 → onOpacityValueSliderWidget2Changed
4. applyTransformButton → onApplyTransform
5. undoTransformButton → onUndoTransform

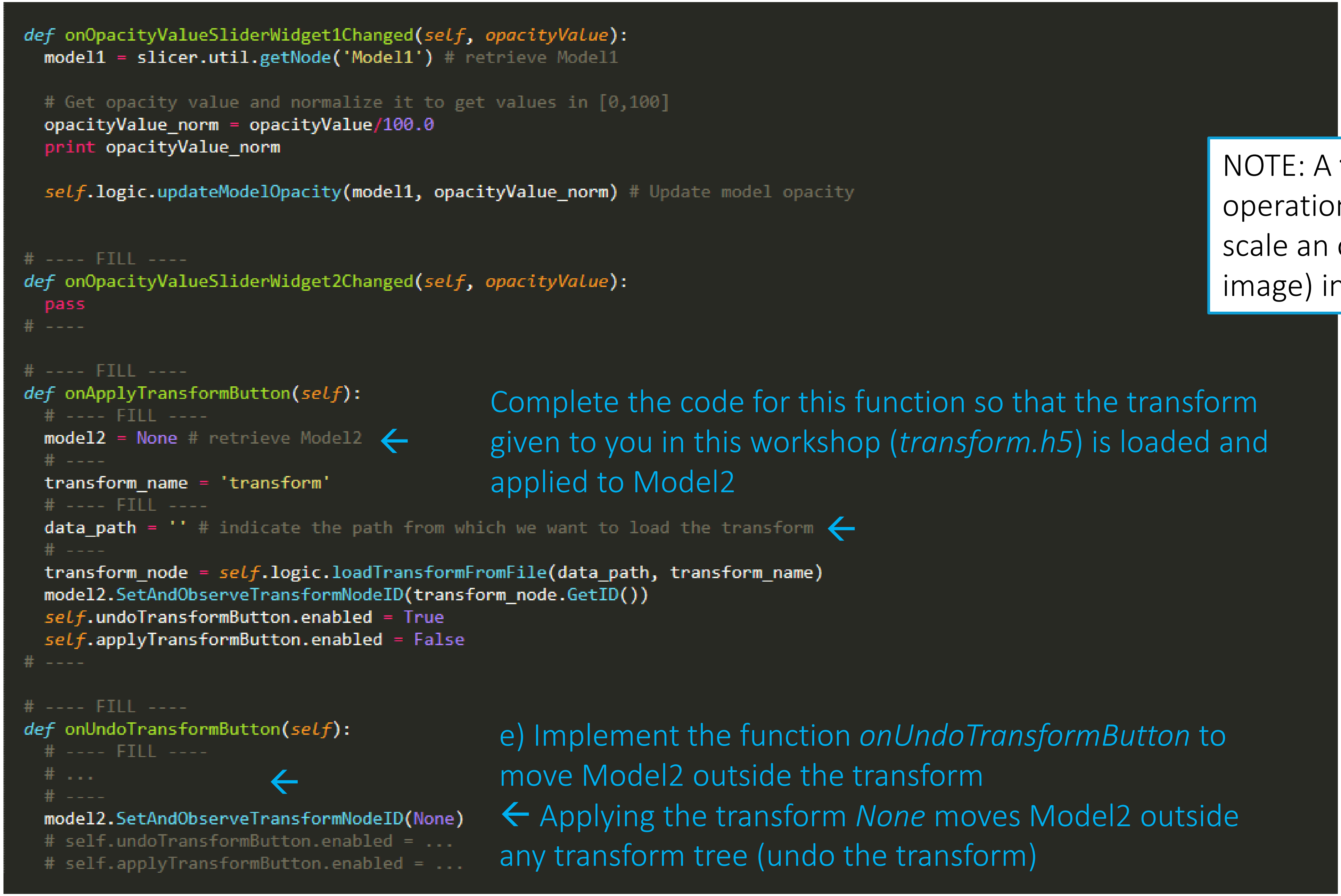
4. Code implementation: widget

Now we have to implement the code for each of the funtions we have associated with the widgets:



```
# ------ 3. DEFINITION OF FUNTIONS CALLED WHEN PRESSING THE BUTTONS ------

def onLoadModel1Button(self):
    model_name = 'Model1.stl' # indicate name of model to be loaded
    data_path = slicer.modules.mymodule.path.replace("MyModule.py","") + 'Resources/Data' # indicate the path from w
    self.logic.loadModelFromFile(data_path, model_name, [1,0,0], True) # call function from logic
    self.ui.model1_checkBox.checked = True
    self.ui.opacityValueSliderWidget_1.value = 100

# ---- FILL ----
# def onLoadModel2Button(self): ...
# ----

def onModel1VisibilityChecked(self, checked):
    model1 = slicer.util.getNode('Model1') # retrieve Model1
    self.logic.updateVisibility(model1, checked)

# ---- FILL ----
# def onModel2VisibilityChecked(self, checked): ...
# ----
```

To write the new code, take a look at the parallel function. The new code will be almost the same.



```
def onOpacityValueSliderWidget1Changed(self, opacityValue):
  model1 = slicer.util.getNode('Model1') # retrieve Model1

  # Get opacity value and normalize it to get values in [0,100]
  opacityValue_norm = opacityValue/100.0
  print opacityValue_norm

  self.logic.updateModelOpacity(model1, opacityValue_norm) # Update model opacity


# ---- FILL ----
def onOpacityValueSliderWidget2Changed(self, opacityValue):
  pass
# ----

# ---- FILL ----
def onApplyTransformButton(self):
  # ---- FILL ----
  model2 = None # retrieve Model2  ←
  # ----
  transform_name = 'transform'
  # ---- FILL ----
  data_path = '' # indicate the path from which we want to load the transform  ←
  # ----
  transform_node = self.logic.loadTransformFromFile(data_path, transform_name)
  model2.SetAndObserveTransformNodeID(transform_node.GetID())
  self.undoTransformButton.enabled = True
  self.applyTransformButton.enabled = False
# ----

# ---- FILL ----
def onUndoTransformButton(self):
  # ---- FILL ----
  # ...           ←
  # ----
  model2.SetAndObserveTransformNodeID(None)
  # self.undoTransformButton.enabled = ...
  # self.applyTransformButton.enabled = ...
```

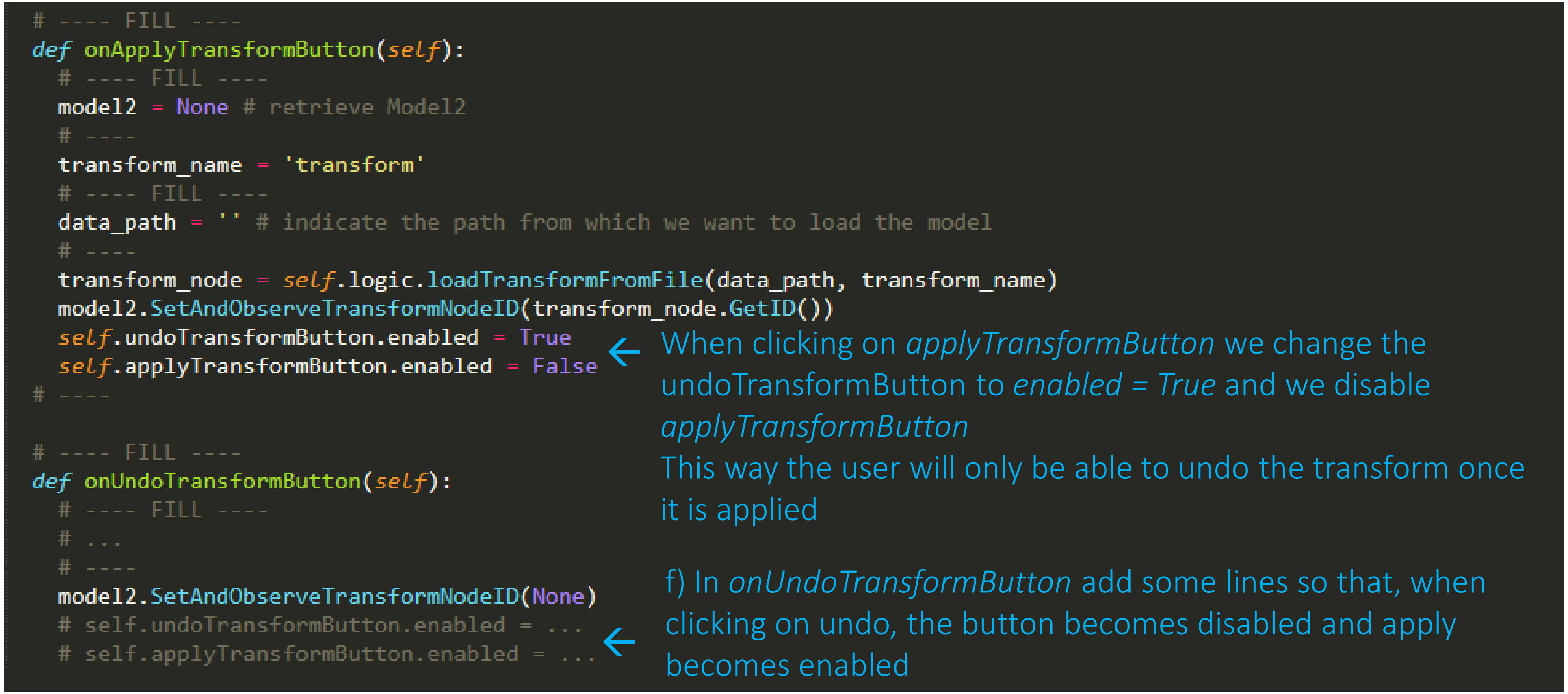
Complete the code for this function so that the transform given to you in this workshop (*transform.h5*) is loaded and applied to Model2

e) Implement the function *onUndoTransformButton* to move Model2 outside the transform

← Applying the transform *None* moves Model2 outside any transform tree (undo the transform)

NOTE: A **transform** is a mathematical operation used to move, rotate, or scale an object (such as a model or image) in space.

4. Code implementation: widget



```
# ---- FILL ----
def onApplyTransformButton(self):
  # ---- FILL ----
  model2 = None # retrieve Model2
  # ----
  transform_name = 'transform'
  # ---- FILL ----
  data_path = '' # indicate the path from which we want to load the model
  # ----
  transform_node = self.logic.loadTransformFromFile(data_path, transform_name)
  model2.SetAndObserveTransformNodeID(transform_node.GetID())
  self.undoTransformButton.enabled = True
  self.applyTransformButton.enabled = False
# ----

# ---- FILL ----
def onUndoTransformButton(self):
  # ---- FILL ----
  # ...
  # ----
  model2.SetAndObserveTransformNodeID(None)
  # self.undoTransformButton.enabled = ...
  # self.applyTransformButton.enabled = ...
```

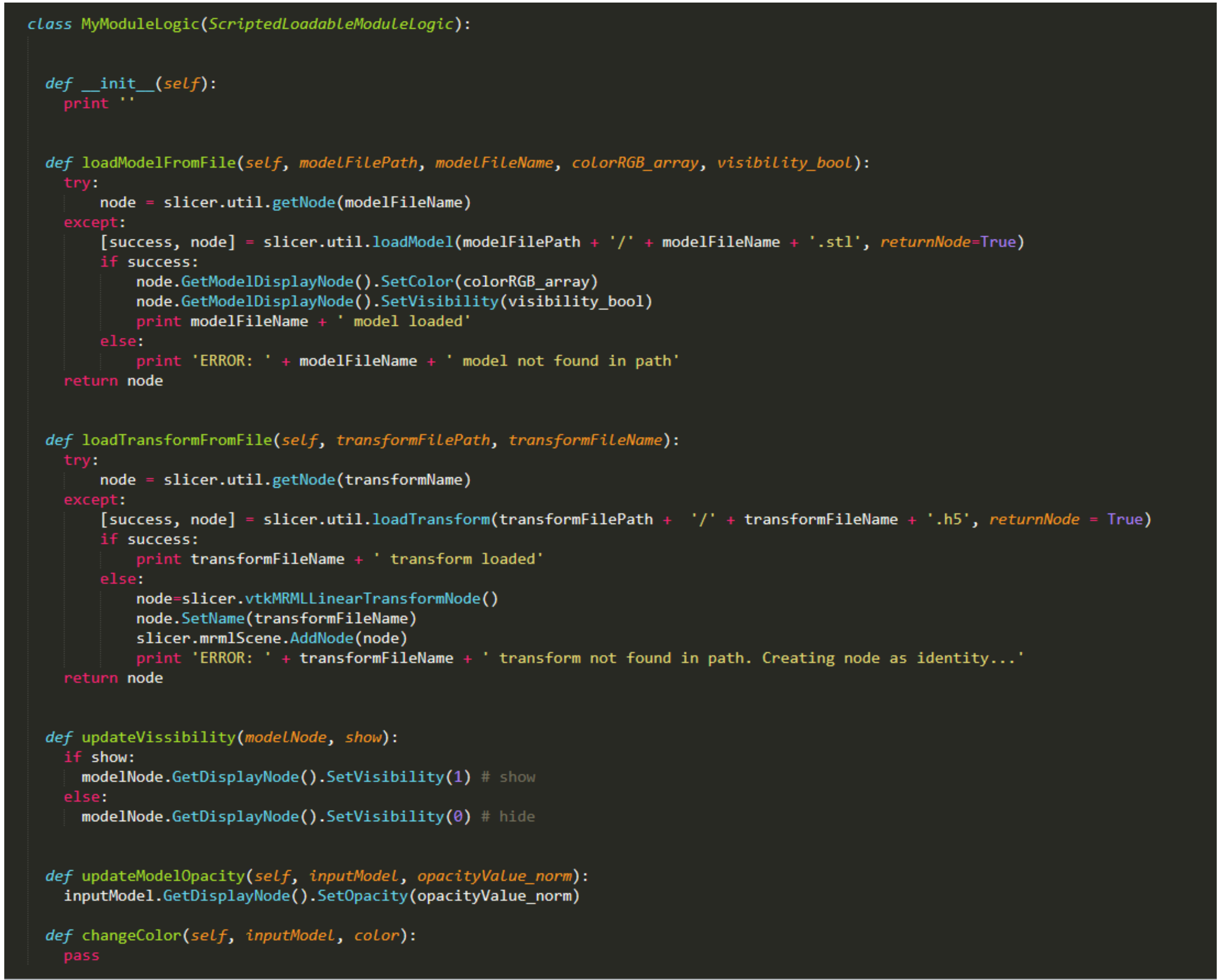
When clicking on *applyTransformButton* we change the undoTransformButton to *enabled = True* and we disable *applyTransformButton*
This way the user will only be able to undo the transform once it is applied

f) In *onUndoTransformButton* add some lines so that, when clicking on undo, the button becomes disabled and apply becomes enabled

You can access other parameters of the widgets from the functions to modify their appereance, text, if they are enabled,… and also if a collapsible button is collapsed.
OPTIONAL: Make the section LOAD DATA collapse after loading Model 2.

4. Code implementation: logic

Finally, lets take a look at the functions implemented in the Logic class:

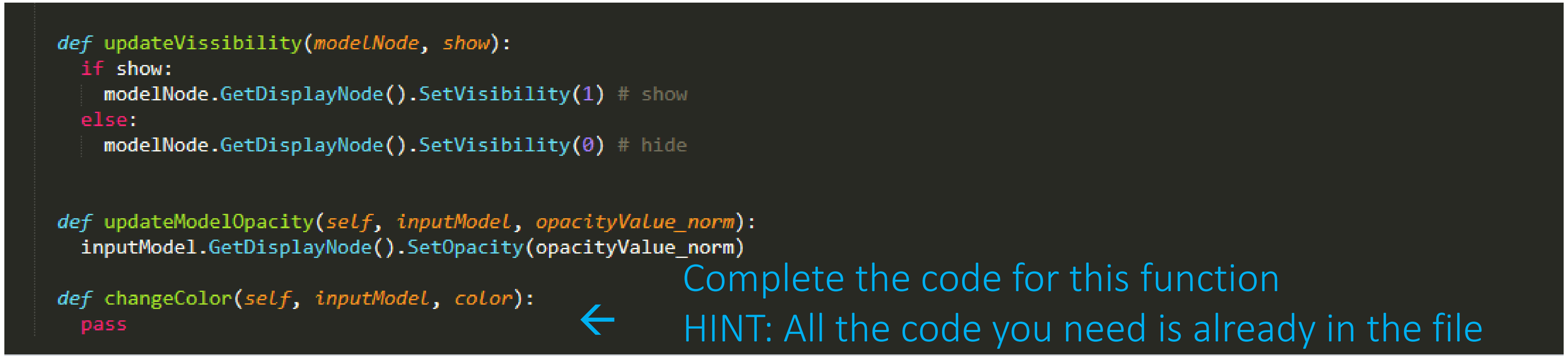


```
class MyModuleLogic(ScriptedLoadableModuleLogic):

 def __init__(self):
  print ''

 def loadModelFromFile(self, modelFilePath, modelFileName, colorRGB_array, visibility_bool):
  try:
    node = slicer.util.getNode(modelFileName)
  except:
    [success, node] = slicer.util.loadModel(modelFilePath + '/' + modelFileName + '.stl', returnNode=True)
    if success:
      node.GetModelDisplayNode().SetColor(colorRGB_array)
      node.GetModelDisplayNode().SetVisibility(visibility_bool)
      print modelFileName + ' model loaded'
    else:
      print 'ERROR: ' + modelFileName + ' model not found in path'
  return node

 def loadTransformFromFile(self, transformFilePath, transformFileName):
  try:
    node = slicer.util.getNode(transformName)
  except:
    [success, node] = slicer.util.loadTransform(transformFilePath +  '/' + transformFileName + '.h5', returnNode = True)
    if success:
      print transformFileName + ' transform loaded'
    else:
      node=slicer.vtkMRMLLinearTransformNode()
      node.SetName(transformFileName)
      slicer.mrmlScene.AddNode(node)
      print 'ERROR: ' + transformFileName + ' transform not found in path. Creating node as identity...'
  return node

 def updateVissibility(modelNode, show):
  if show:
   modelNode.GetDisplayNode().SetVisibility(1) # show
  else:
   modelNode.GetDisplayNode().SetVisibility(0) # hide

 def updateModelOpacity(self, inputModel, opacityValue_norm):
  inputModel.GetDisplayNode().SetOpacity(opacityValue_norm)

 def changeColor(self, inputModel, color):
  pass
```

4. Code implementation: logic

Until now we have only implemented functions in the widget which capture the interaction of the user with the buttons and call functions already defined in the logic.

In this step we are going to define a new function in the logic to change the color of the models.



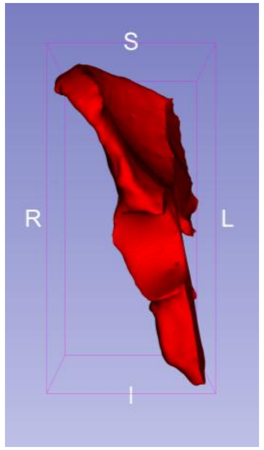
```
def updateVissibility(modelNode, show):
  if show:
    modelNode.GetDisplayNode().SetVisibility(1) # show
  else:
    modelNode.GetDisplayNode().SetVisibility(0) # hide


def updateModelOpacity(self, inputModel, opacityValue_norm):
  inputModel.GetDisplayNode().SetOpacity(opacityValue_norm)

def changeColor(self, inputModel, color):
  pass
```

Complete the code for this function

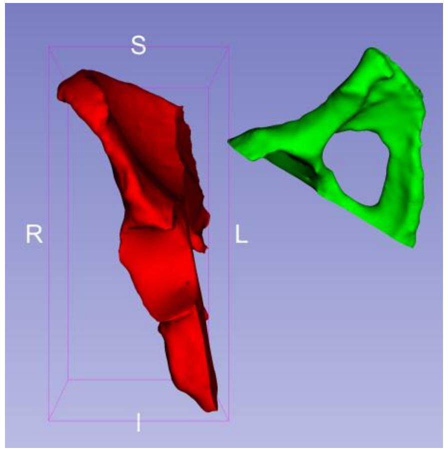
HINT: All the code you need is already in the file

4. Code implementation: logic

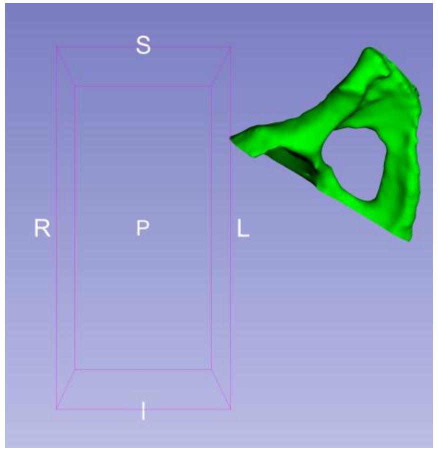
- Now load your module in Slicer and check every button works.



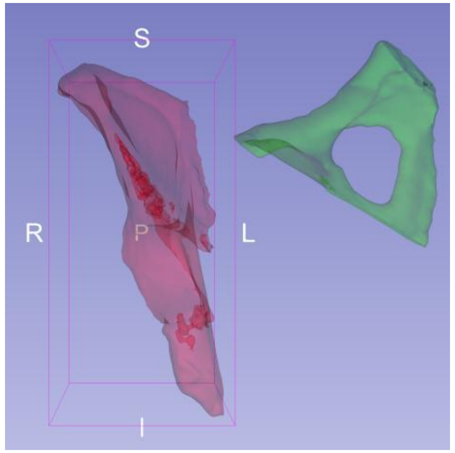
Load Model 1



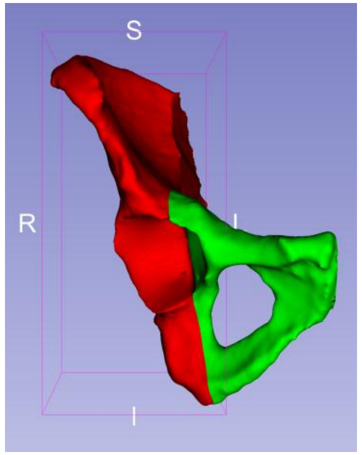
Load Model 2



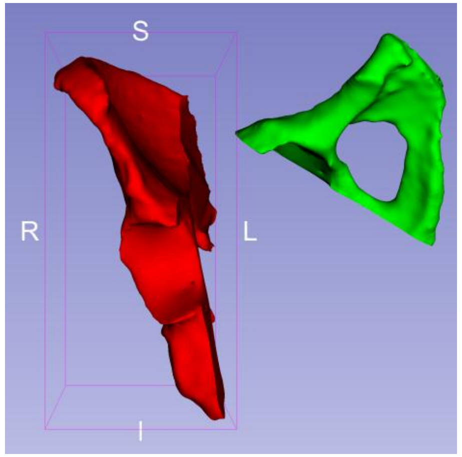
Hide Model 1/2



Change Opacity



Apply Transform



Undo Transform

TUTORIAL COMPLETED!

Now you know the basics to create a module in 3D Slicer for whatever application you want.

Enjoy exploring all the possibilities!