



Paul Cézanne, Moulin sur la Coulevre à Pontoise, 1881, Staatliche Museen zu Berlin, Nationalgalerie

Programming in Slicer4

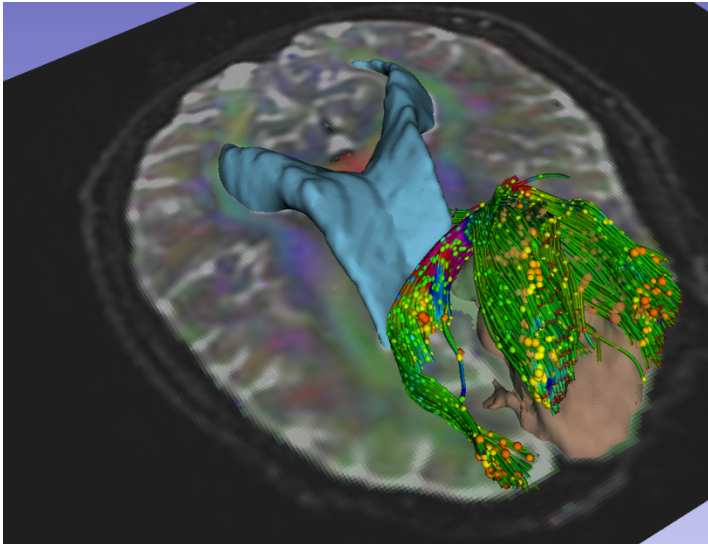
Sonia Pujol, Ph.D.
Surgical Planning Laboratory,
Harvard Medical School

Steve Pieper, Ph.D.
Isomics Inc.

The NA-MIC Kit



3D Slicer version 4 (Slicer4.5)



- An **end-user application** for image analysis
- An **open-source environment** for software development
- A software platform that is both **easy to use** for clinical researchers and **easy to extend** for programmers

Slicer Modules

- **Command Line Interface (CLI):**
standalone executable with limited input/output arguments
- **Scripted Modules (Python):**
recommended for fast prototyping
- **Loadable Modules (C++ Plugins):**
optimized for heavy computation

Slicer4 Highlights: Python

The Python console of Slicer4 gives access to

- scene objects (MRML)
- data arrays (volumes, models)
- GUI elements that can be encapsulated in a module
- Processing Libraries: numpy, VTK, ITK,CTK

Slicer4 Scripted Modules

- Python scripted modules allow more **interactive functionalities** (e.g. 'Flythrough' in Endoscopy module) and **rapid prototyping**
- GUI based on Qt libraries accessed via Python

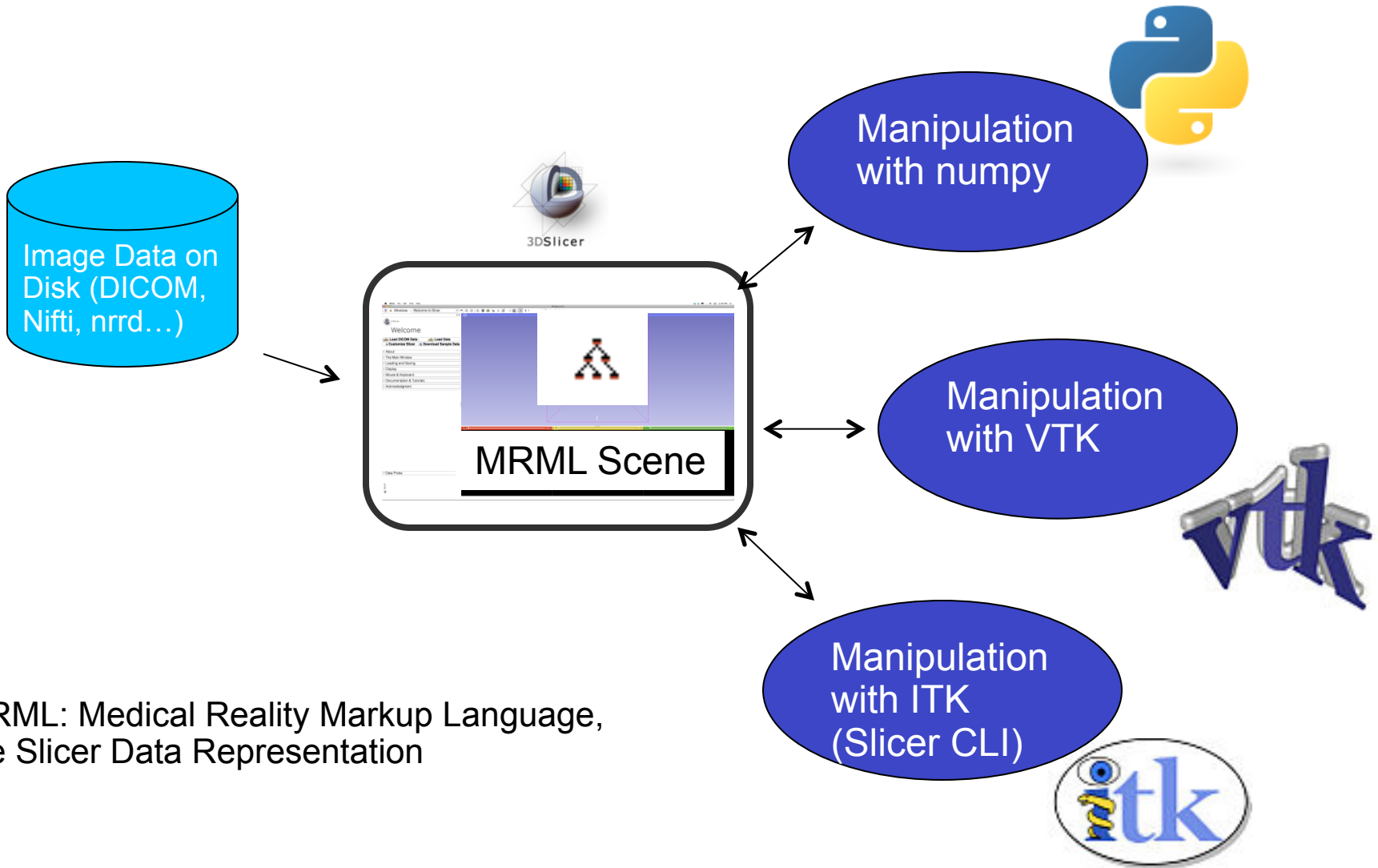


Tutorial Goal

- This tutorial guides you through the steps of programming a HelloPython scripted module for running a Laplacian filtering and sharpening.
- For additional details and pointers, visit the Slicer Documentation page

<https://www.slicer.org/slicerWiki/index.php/Documentation/4.5/Training>

Processing Examples in this Tutorial



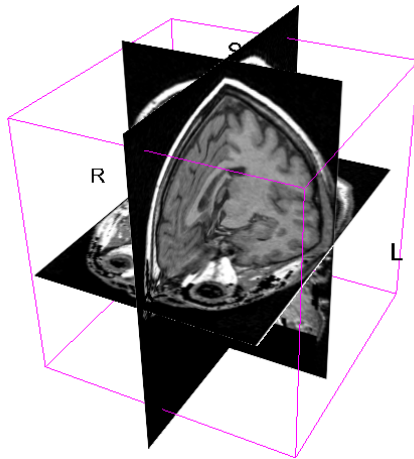
MRML: Medical Reality Markup Language,
the Slicer Data Representation

Prerequisites

- This course supposes that you have taken the tutorial: ‘Slicer4 Data Loading and Visualization’- Sonia Pujol Ph.D.
- The tutorial and HelloPython dataset are available on the Slicer4.5 compendium:
<https://www.slicer.org/slicerWiki/index.php/Documentation/4.5/Training>
- Programming experience is required, and some familiarity with Python is essential.

Course Material

Slicer4.5 version available at www.slicer.org



spgr.nhdr
spgr.raw.gz
(124 SPGR images)

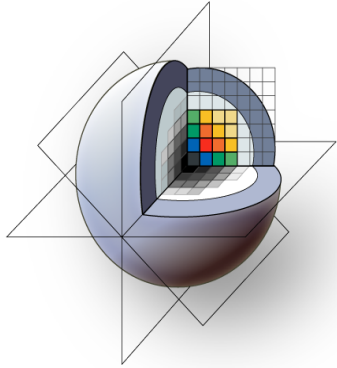


HelloPython.py
HelloLaplace.py
HelloSharpen.py

Unzip the **HelloPythonSlicer4.zip** archive

Course Overview

- Part A: Exploring Slicer via Python
- Part B: Integration of the HelloPython.py program into Slicer4
- Part C: Implementation of the Laplace operator in the HelloPython module
- Part D: Image Sharpening using the Laplace operator



3DSlicer



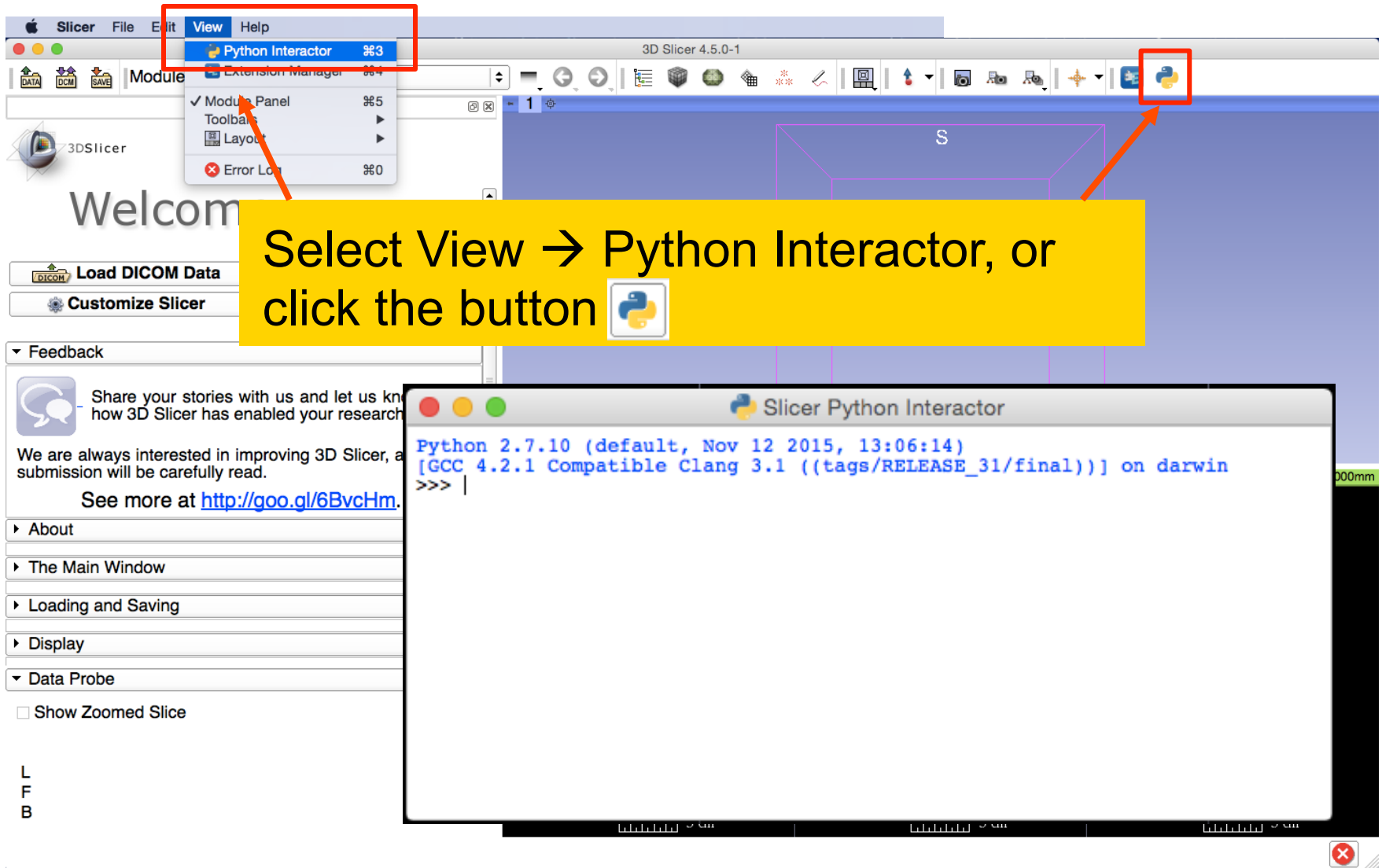
Part A: EXPLORING SLICER VIA PYTHON

Python in Slicer

Slicer 4.5 includes python 2.7.10 and a rich set of standard libraries

- *Included:*
 - **numpy, VTK, CTK, PythonQt,**
and most of standard python library
- *Not included:*
 - scipy (scientific tools for python),
 - matplotlib (python 2D plotting library),
 - ipython (interactive python)and some other popular packages that we have found difficult to package for distribution

Python Console in Slicer



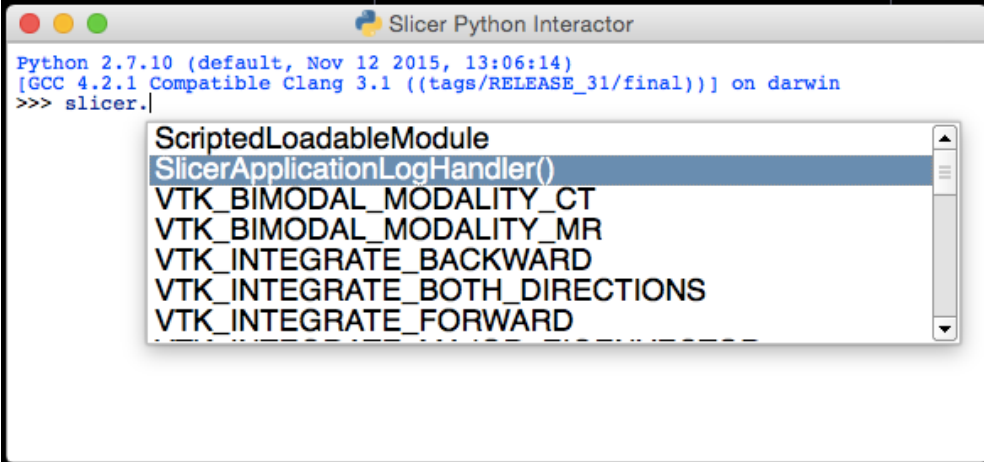
The screenshot shows the 3D Slicer 4.5.0-1 application window. The 'View' menu is open, and 'Python Interactor' is highlighted. An orange arrow points from the 'Python Interactor' menu item to a Python logo button in the top toolbar. Another orange arrow points from the same button to a yellow text box. The yellow text box contains the instruction: 'Select View → Python Interactor, or click the button' followed by a small Python logo icon. In the bottom right, a 'Slicer Python Interactor' window is open, displaying the Python 2.7.10 shell environment on a Darwin system.

Select View → Python Interactor, or click the button

```
Python 2.7.10 (default, Nov 12 2015, 13:06:14)
[GCC 4.2.1 Compatible Clang 3.1 ((tags/RELEASE_31/final))] on darwin
>>> |
```

General Python Console Features

- Command Line Editing:
 - Left/Right Arrow Keys, Home, End
 - Delete (Control-D)
- Command Completion:
 - Tab Key
- Input History:
 - Up/Down Arrow Keys



The screenshot shows a window titled "Slicer Python Interactor". The console text includes the Python version (2.7.10), GCC/Clang version (4.2.1/3.1), and the operating system (darwin). The prompt is ">>> slicer.". A completion menu is displayed, listing several modules: "ScriptedLoadableModule", "SlicerApplicationLogHandler()", "VTK_BIMODAL_MODALITY_CT", "VTK_BIMODAL_MODALITY_MR", "VTK_INTEGRATE_BACKWARD", "VTK_INTEGRATE_BOTH DIRECTIONS", and "VTK_INTEGRATE_FORWARD". The "SlicerApplicationLogHandler()" option is currently selected and highlighted in blue.

Add Volume

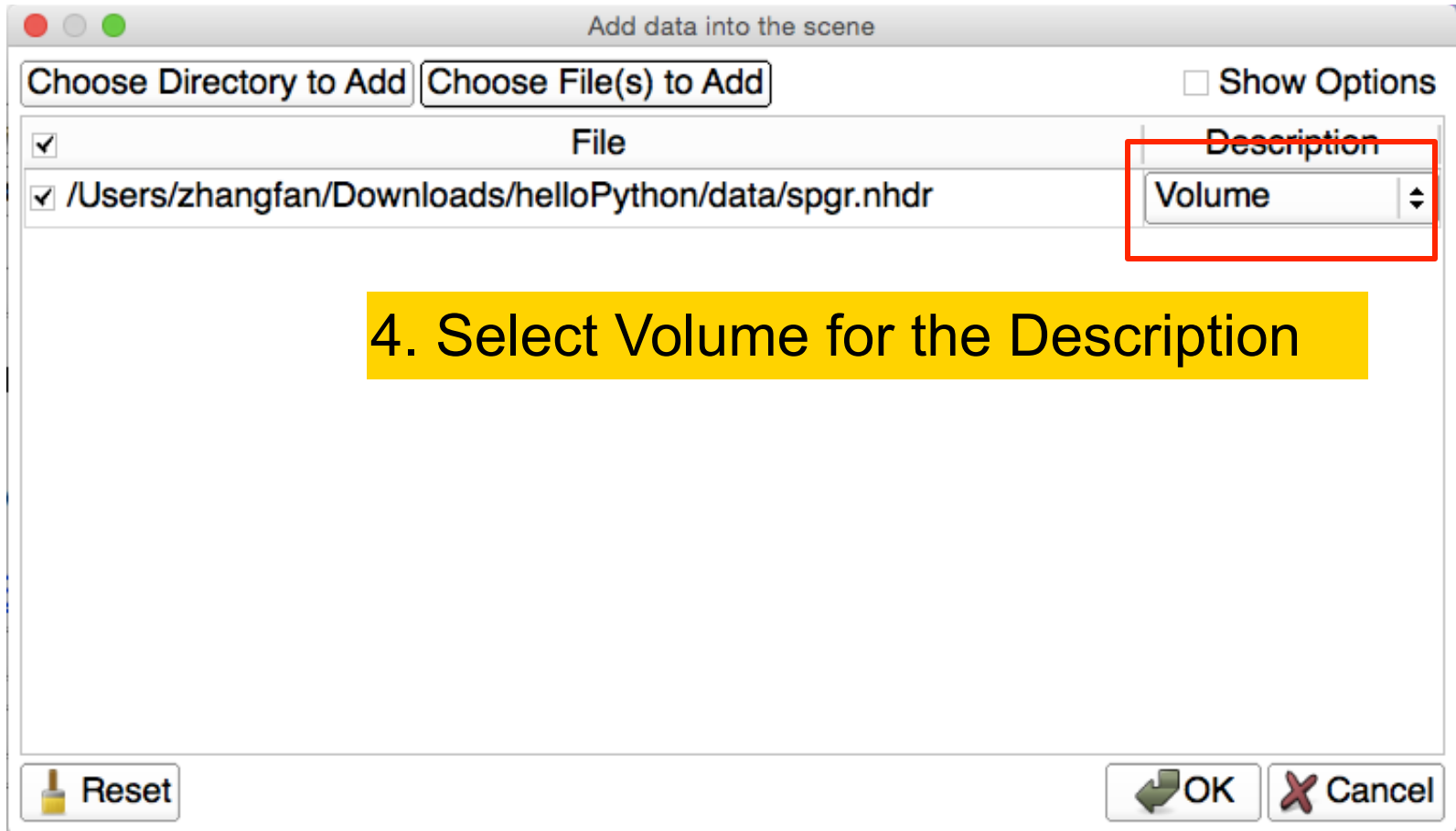
The screenshot shows the 3D Slicer 4.5.0-1 interface. The 'File' menu is open, and the 'Add Data' option is highlighted. A yellow box with the text '1. Select File → Add Data' is overlaid on the menu. Below this, another yellow box with the text '2. Select Choose File(s) to Add' is overlaid on the 'Choose File(s) to Add' button in the 'Add data into the scene' dialog. The dialog shows a file browser with the directory '/Users/zhangfan/...helloPython/data' selected. The file 'spgr.nhdr' is highlighted in the file list, and a yellow box with the text '3. Load the dataset spgr.nhdr located in the directory HelloPython/data' is overlaid on the file list. The 'File name' field in the dialog contains 'spgr.nhdr'.

1. Select File → Add Data

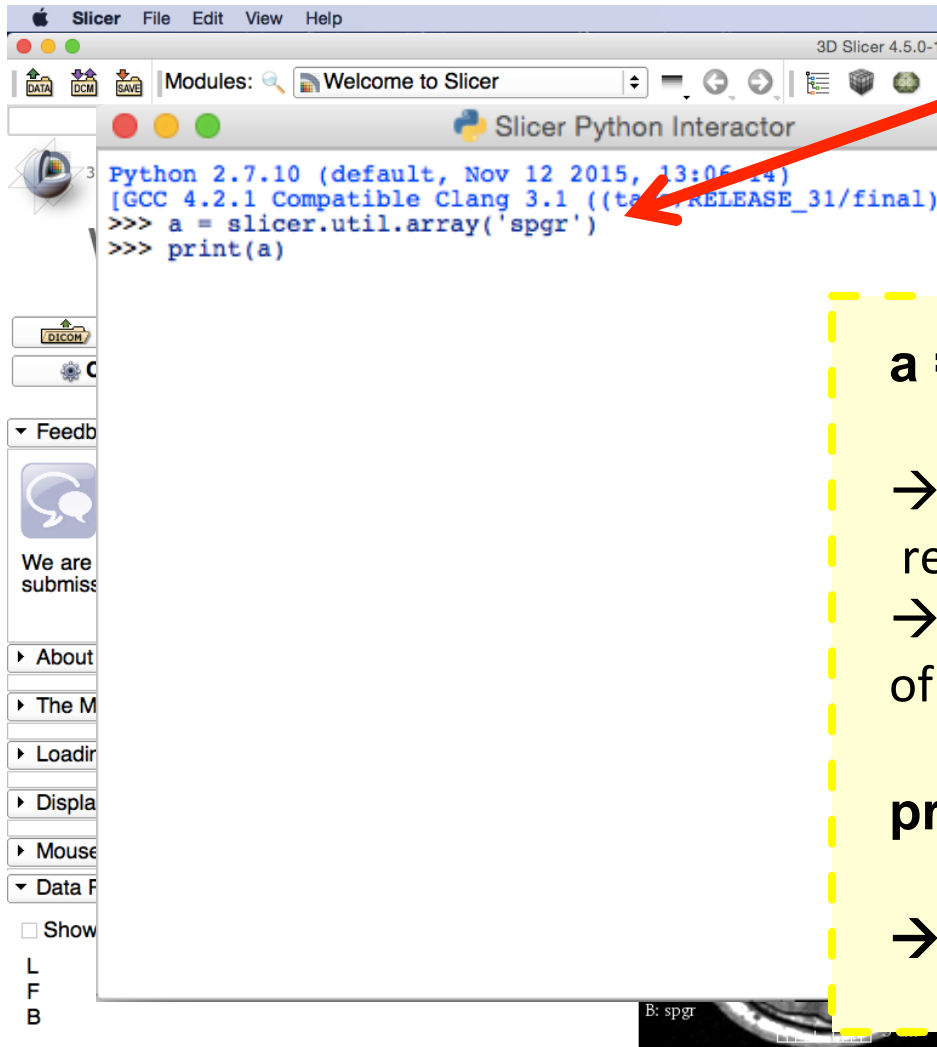
2. Select Choose File(s) to Add

3. Load the dataset **spgr.nhdr** located in the directory **HelloPython/data**

Add Volume



Access to MRML and Arrays



Run the following code in the Python console

```
a = slicer.util.array('spgr')
```

→ Uses the slicer.util package to return a numpy array of the image
→ The variable 'a' is a numpy ndarray of the volume data we just loaded

```
print( a )
```

→ Shows a shortened view of the array

Access to MRML and Arrays

The screenshot displays the 3D Slicer application interface. On the left is the 'Slicer' window with a sidebar containing buttons for 'DATA', 'DCM', 'SAVE', 'Load DICOM', and 'Customize', along with a 'Feedback' section. The main area shows a 3D view of a brain scan with axes labeled 'P' (Posterior), 'L' (Left), and 'I' (Inferior). Below the 3D view are two 2D image windows: a sagittal view on the left and a coronal view on the right, both labeled 'B: spgr' and showing a 5 cm scale bar. The right window also displays 'R: 3.050mm' and 'A: 14.238mm'. On the right side of the interface is the 'Slicer Python Interactor' window, which shows the output of a `print(a)` command. The output consists of multiple NumPy arrays representing MRML data, including intensity values for the 'spgr' image.

The intensity values of the spgr image appear in the Python console

```
>>> print(a)
[[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 1 1 ..., 3 5 1]
 ...,
 [1 0 2 ..., 1 1 2]
 [5 2 0 ..., 4 1 4]
 [4 1 1 ..., 4 1 4]]]

[[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [2 1 3 ..., 5 1 2]
 ...,
 [3 2 3 ..., 0 1 3]
 [0 1 2 ..., 1 2 3]
 [0 1 2 ..., 1 2 2]]]

[[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [3 2 1 ..., 4 1 3]
 ...,
 [3 2 3 ..., 2 3 1]
 [0 1 2 ..., 1 1 1]
 [0 1 2 ..., 1 1 1]]]

[[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [1 3 1 ..., 0 2 2]
 ...,
 [3 3 0 ..., 0 2 3]
 [2 1 1 ..., 0 1 3]
 [2 1 1 ..., 0 1 3]]]

[[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [1 3 1 ..., 2 2 2]
 ...,
 [1 1 3 ..., 1 2 1]
 [6 7 3 ..., 2 3 5]
 [5 6 3 ..., 2 3 4]]]

[[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [2 1 0 ..., 1 0 0]
 ...,
 [2 2 1 ..., 1 2 2]
 [0 4 0 ..., 0 1 3]
 [0 3 0 ..., 0 1 2]]]
>>>
```

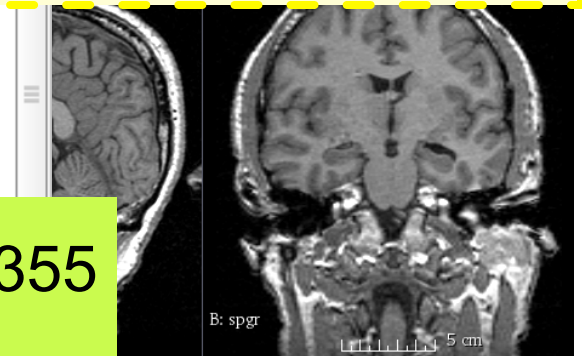
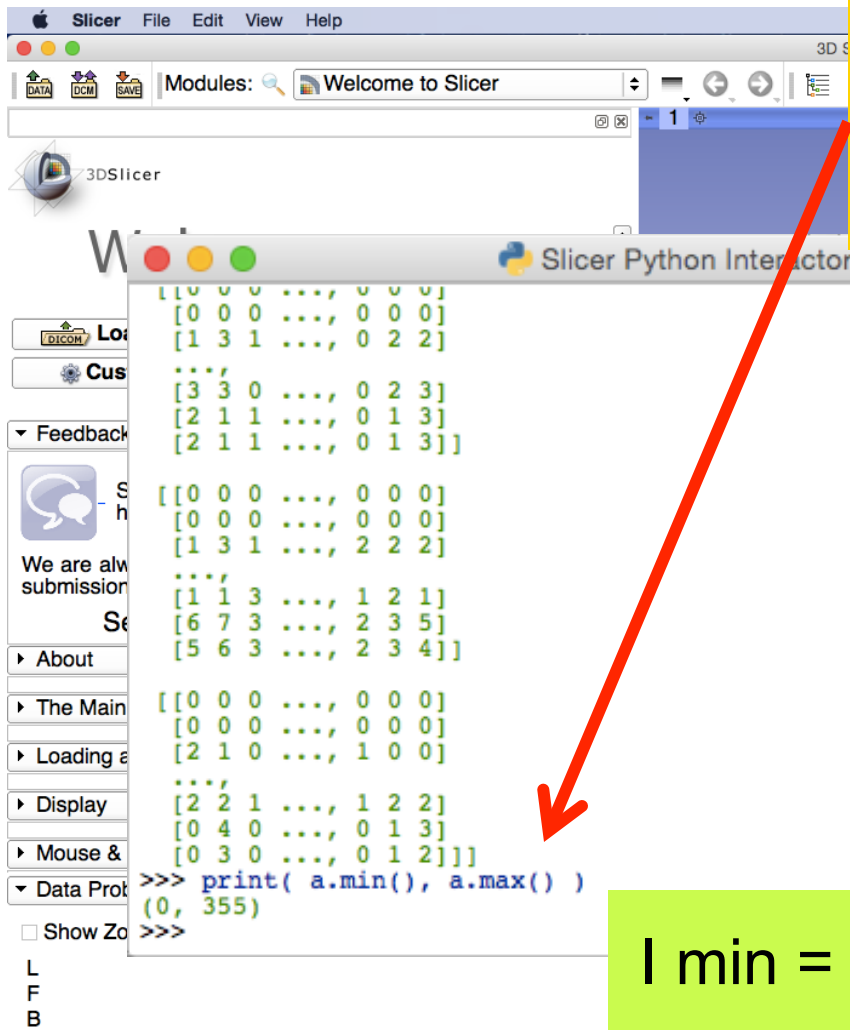
Access to MRML and Arrays

Type the following command to display the min and max intensity value of the spgr image

```
print( a.min(), a.max() )
```

→ Use numpy array methods to analyze the data

I min = 0 ; I max = 355



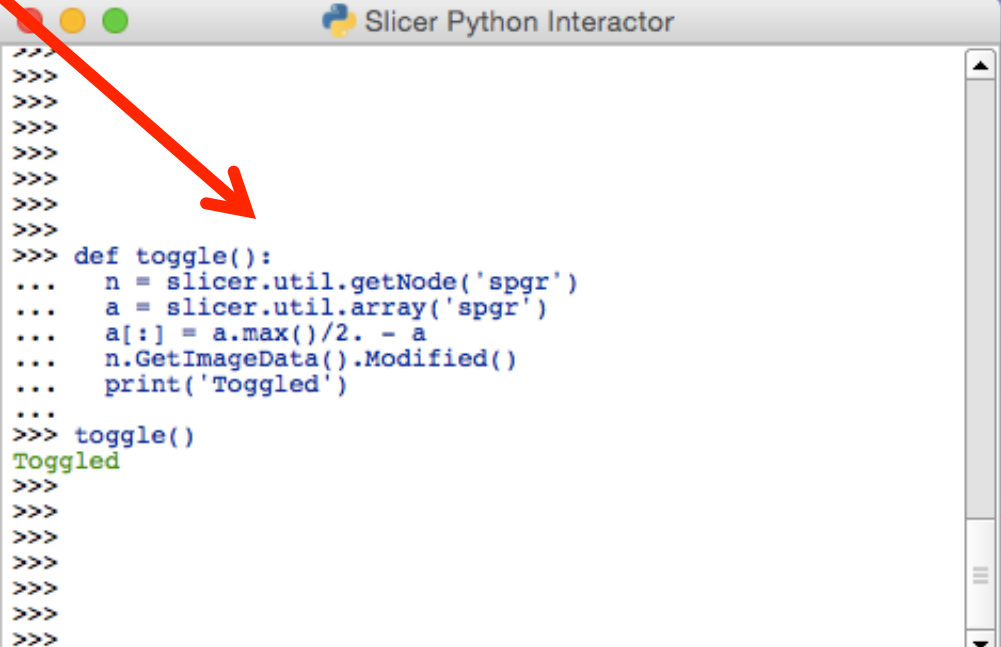
Manipulating Arrays

Run the following code in the Python console, (indent each new line with 2 spaces)

Welcome

```
def toggle():  
    n = slicer.util.getNode('spgr')  
    a = slicer.util.array('spgr')  
    a[:] = a.max()/2. - a  
    n.GetImageData().Modified()  
    print('Toggled')
```

toggle()



```
>>>  
>>>  
>>>  
>>>  
>>>  
>>>  
>>> def toggle():  
...     n = slicer.util.getNode('spgr')  
...     a = slicer.util.array('spgr')  
...     a[:] = a.max()/2. - a  
...     n.GetImageData().Modified()  
...     print('Toggled')  
...  
>>> toggle()  
Toggled  
>>>  
>>>  
>>>  
>>>  
>>>  
>>>
```

☐ Show Zoomed Slice

L
F
B

For practice: use up arrow and return keys to execute toggle() over and over

The toggle function in more detail

- **def toggle():**
 - Defines a python function
 - Body of function performs element-wise math on entire volume
 - Easy mix of scalar and volume math
- Telling slicer that the image data for node 'n' has been modified causes the slice view windows to refresh

Qt GUI in Python

Run the following code in the Python console

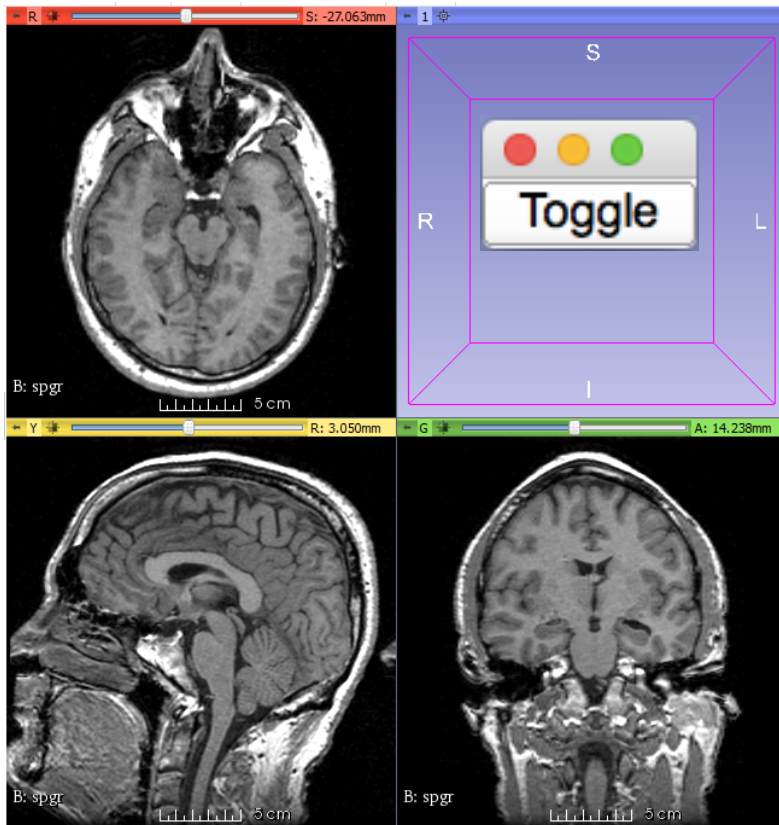
Welcome

```
b = qt.QPushButton('Toggle')  
b.connect('clicked()', toggle)  
b.show()
```

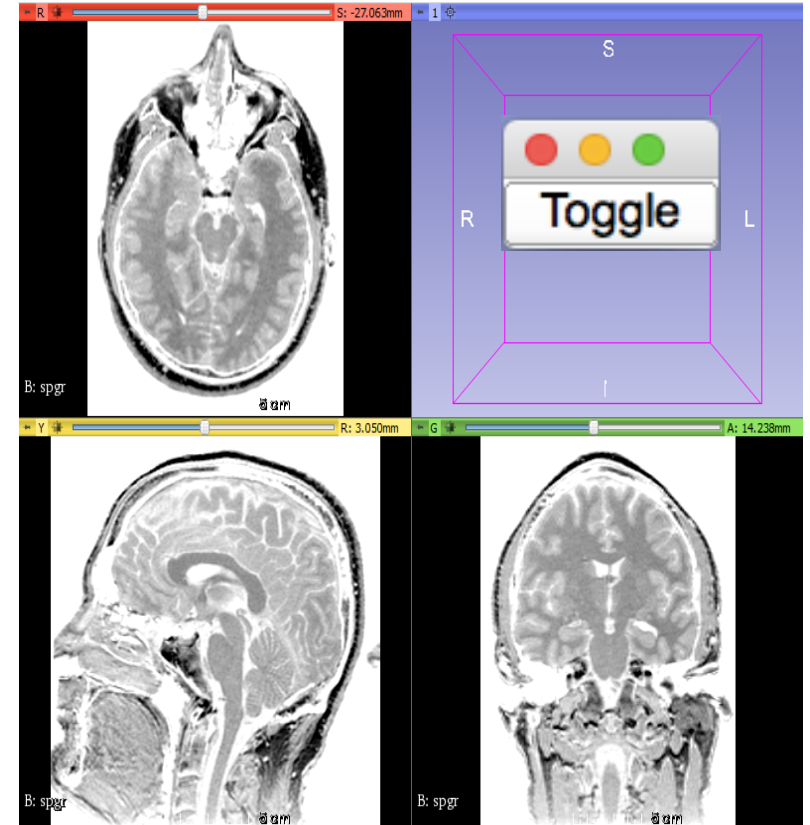
```
>>>  
>>>  
>>>  
>>>  
>>>  
>>>  
>>>  
>>>  
>>> b = qt.QPushButton('Toggle')  
>>> b.connect('clicked()', toggle)  
True  
>>> b.show()|
```

What do you think will happen when you run this code?
What about when you push the button?

Result with button toggling



Original



Result

(*) Put the slicer view windows front to see the changes

In More Detail

- Slicer uses **PythonQt** to expose the Qt library
- Sophisticated interactive modules can be written entirely with Python code calling C++ code that is wrapped in Python
 - e.g. Endoscopy, Editor, SampleData, ChangeTracker, and other slicer modules in the Slicer source code

(*) Qt: <http://qt-project.org/>

(**) PythonQt: <http://pythonqt.sourceforge.net/> (MeVis)

```

from __main__ import vtk, qt, ctk, slicer

# HelloPython
#

class HelloPython:
    def __init__(self, parent):
        parent.title = "Hello Python"
        parent.categories = ["Example"]
        parent.dependencies = []
        parent.contributors = ["Jean-Christophe Fillard-Robin (Kitware)",
                               "Steve Pieper (Isomics)",
                               "Sonia Pujol (DSH)"] # replace with "Firstname Lastname (Org)"

        parent.helpText = """
        Example of scripted loadable extension for the HelloPython tutorial.
        """
        parent.acknowledgementText = """
        This file was originally developed by Jean-Christophe Fillard-Robin, Kitware Inc.,
        Steve Pieper, Isomics, Inc., and Sonia Pujol, Brigham and Women's Hospital and was
        partially funded by NIH grant 1P41H0013215-1251 (NAC) and is part of the National Alliance
        for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the
        NIH Roadmap for Medical Research, Grant U54 EB005149. """ # replace with organization, grant and thanks.
        self.parent = parent

# qHelloPythonWidget
#

class HelloPythonWidget:
    def __init__(self, parent = None):
        if not parent:
            self.parent = slicer.qMRMLWidget()
            self.parent.setLayout(qt.QVBoxLayout())
            self.parent.setMRMLScene(slicer.mrmlScene)
        else:
            self.parent = parent
            self.layout = self.parent.layout()
        if not parent:
            self.setup()
            self.parent.show()

    def setup(self):
        # Instantiate and connect widgets ...

        # Collapsible button
        sampleCollapsibleButton = ctk.ctkCollapsibleButton()
        sampleCollapsibleButton.text = "A collapsible button"
        self.layout.addWidget(sampleCollapsibleButton)

        # Layout within the sample collapsible button
        sampleFormLayout = qt.QFormLayout(sampleCollapsibleButton)

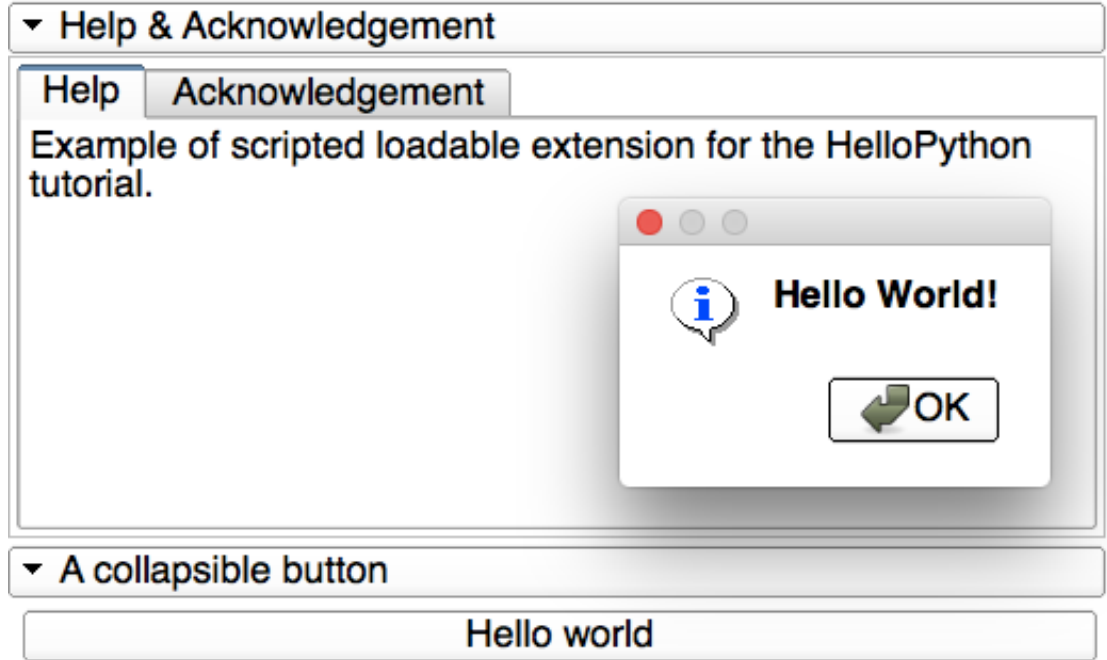
        # HelloWorld button
        # (Insert Section A text here)
        # (Be sure to match indentation of the rest of this
        # code)

        # Add vertical spacer
        self.layout.addStretch(1)

        # Set local var as instance attribute
        self.helloWorldButton = helloWorldButton

    def onHelloWorldButtonClicked(self):
        print "Hello World!"
        # (Insert Section B text here)
        # (Be sure to match indentation of the rest of this
        # code)

```



PART B: INTEGRATION OF THE HELLOPYTHON TUTORIAL TO SLICER4

HelloPython.py

Open the file **HelloPython.py**
Located in the directory
helloPython\code

Module Description

Module GUI

Processing Code

```
from __main__ import vtk, qt, ctk, slicer

#
# HelloPython
#

class HelloPython:
    def __init__(self, parent):
        parent.title = "Hello Python"
        parent.categories = ["Examples"]
        parent.dependencies = []
        parent.contributors = ["Jean-Christophe Fillion-Robin (Kitware)",
                               "Steve Pieper (Isomics)",
                               "Sonia Pujol (BWH)"] # replace with "Firstname Lastname (Org)"

        parent.helpText = """
        Example of scripted loadable extension for the HelloPython tutorial.
        """
        parent.acknowledgementText = """
        This file was originally developed by Jean-Christophe Fillion-Robin, Kitware Inc.,
        Steve Pieper, Isomics, Inc., and Sonia Pujol, Brigham and Women's Hospital and was
        partially funded by NIH grant 3P41RR013218-12S1 (NAC) and is part of the National Alliance
        for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the
        NIH Roadmap for Medical Research, Grant U54 EB005149.""" # replace with organization, grant and thanks.
        self.parent = parent

#
# qHelloPythonWidget
#

class HelloPythonWidget:
    def __init__(self, parent = None):
        if not parent:
            self.parent = slicer.qMRMLWidget()
            self.parent.setLayout(qt.QVBoxLayout())
            self.parent.setMRMLScene(slicer.mrmlScene)
        else:
            self.parent = parent
            self.layout = self.parent.layout()
        if not parent:
            self.setup()
            self.parent.show()

    def setup(self):
        # Instantiate and connect widgets ...

        # Collapsible button
        sampleCollapsibleButton = ctk.ctkCollapsibleButton()
        sampleCollapsibleButton.text = "A collapsible button"
        self.layout.addWidget(sampleCollapsibleButton)

        # Layout within the sample collapsible button
        sampleFormLayout = qt.QFormLayout(sampleCollapsibleButton)

        # HelloWorld button
        # (Insert Section A text here)
        # (be sure to match indentation of the rest of this
        # code)

        # Add vertical spacer
        self.layout.addStretch(1)

        # Set local var as instance attribute
        self.helloWorldButton = helloWorldButton

    def onHelloWorldButtonClicked(self):
        print "Hello World !"
        # (Insert Section B text here)
        # (be sure to match indentation of the rest of this
        # code)|
```

Module Description

```
class HelloPython:
    def __init__(self, parent): ← constructor
        parent.title = "Hello Python"
        parent.categories = ["Examples"]
        parent.dependencies = []
        parent.contributors = ["Jean-Christophe Fillion-Robin (Kitware)",
                               "Steve Pieper (Isomics)",
                               "Sonia Pujol (BWH)"] # replace with "Firstname Lastname (Org)"
        parent.helpText = """
        Example of scripted loadable extension for the HelloPython tutorial.
        """
        parent.acknowledgementText = """
        This file was originally developed by Jean-Christophe Fillion-Robin, Kitware Inc.,
        Steve Pieper, Isomics, Inc., and Sonia Pujol, Brigham and Women's Hospital and was
        partially funded by NIH grant 3P41RR013218-12S1 (NAC) and is part of the National Alliance
        for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through
        the NIH Roadmap for Medical Research, Grant U54 EB005149.""" # replace with organization,
        grant and thanks.
        self.parent = parent
```

This code is
provided in
the template

Module GUI

```
def setup(self):
    # Instantiate and connect widgets ...

    # Collapsible button
    sampleCollapsibleButton = ctk.ctkCollapsibleButton()
    sampleCollapsibleButton.text = "A collapsible button"
    self.layout.addWidget(sampleCollapsibleButton)

    # Layout within the sample collapsible button
    sampleFormLayout = qt.QFormLayout(sampleCollapsibleButton)
```

Add this
Text in
section A

```
# HelloWorld button
helloWorldButton = qt.QPushButton("Hello world")
helloWorldButton.setToolTip("Print 'Hello world' in standard output.")
sampleFormLayout.addWidget(helloWorldButton)
helloWorldButton.connect('clicked(bool)', self.onHelloWorldButtonClicked)
```

```
# Add vertical spacer
self.layout.addStretch(1)

# Set local var as instance attribute
self.helloWorldButton = helloWorldButton
```

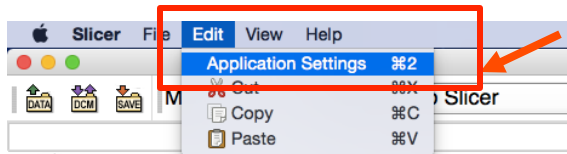
Processing Code

```
def onHelloWorldButtonClicked(self):  
    print "Hello World !"
```

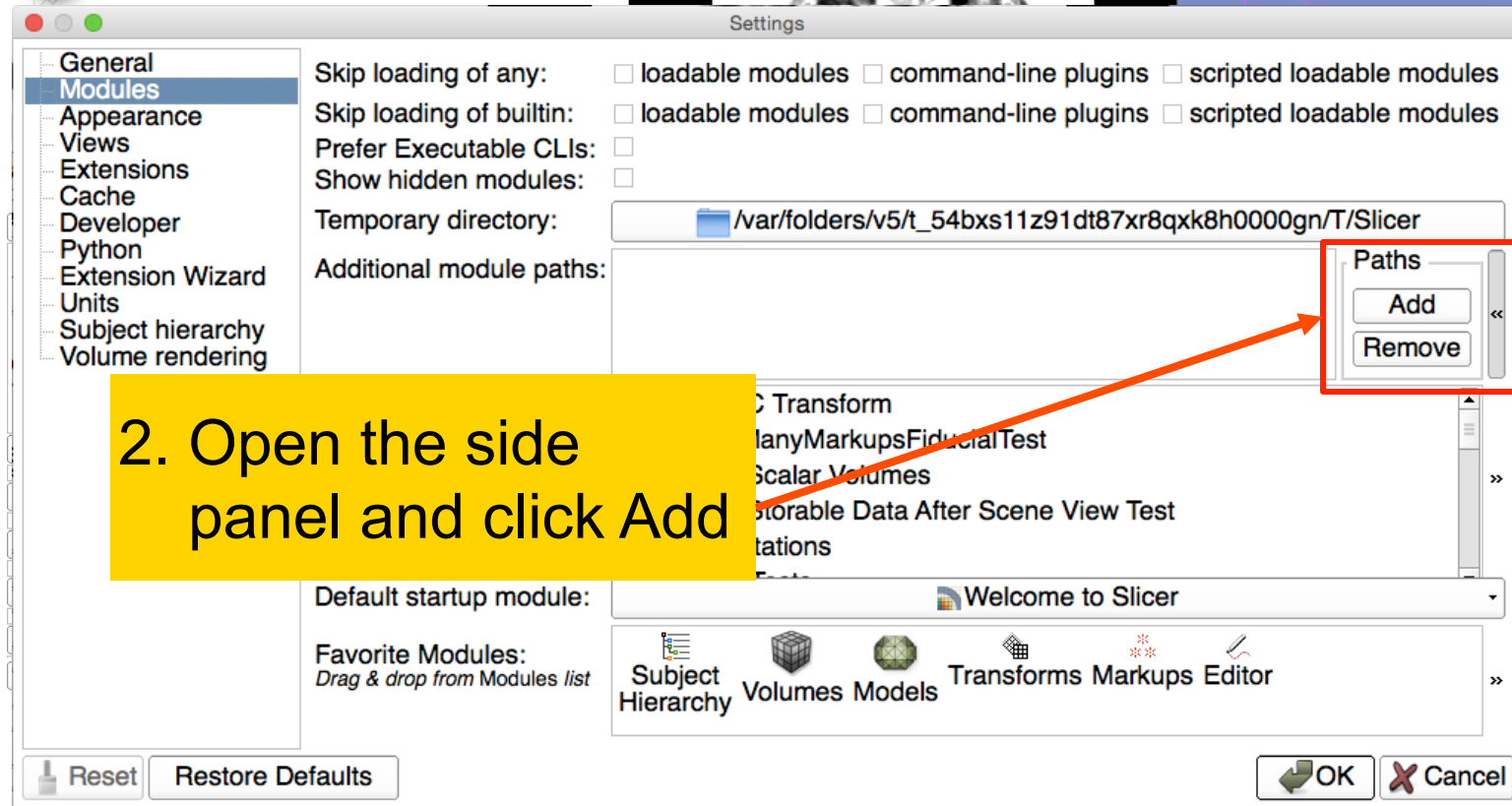
Add this
Text in
section B

```
qt.QMessageBox.information(  
    slicer.util.mainWindow(),  
    'Slicer Python', 'Hello World!')
```

Integrating HelloPython

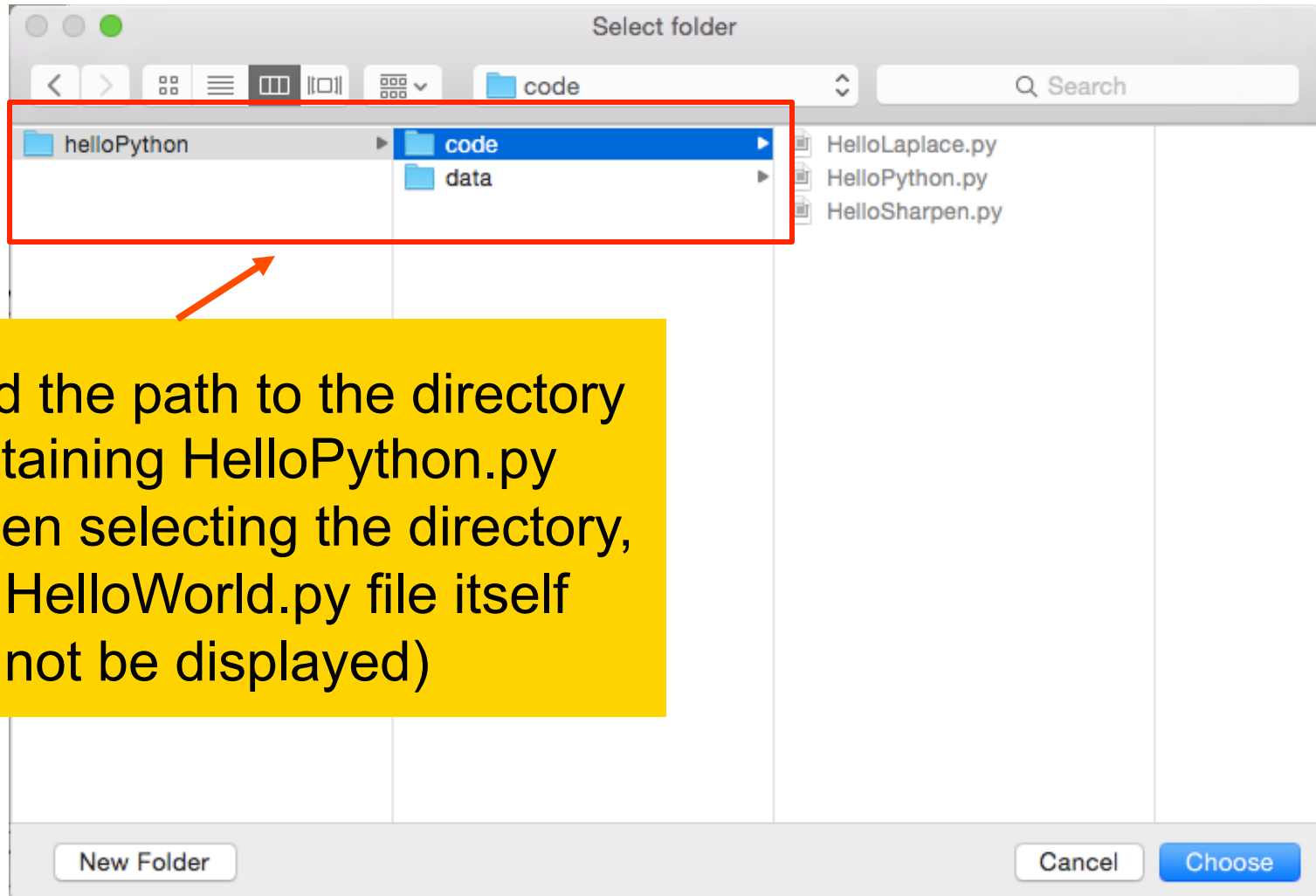


1. Select Modules from the Edit → Application Settings



2. Open the side panel and click Add

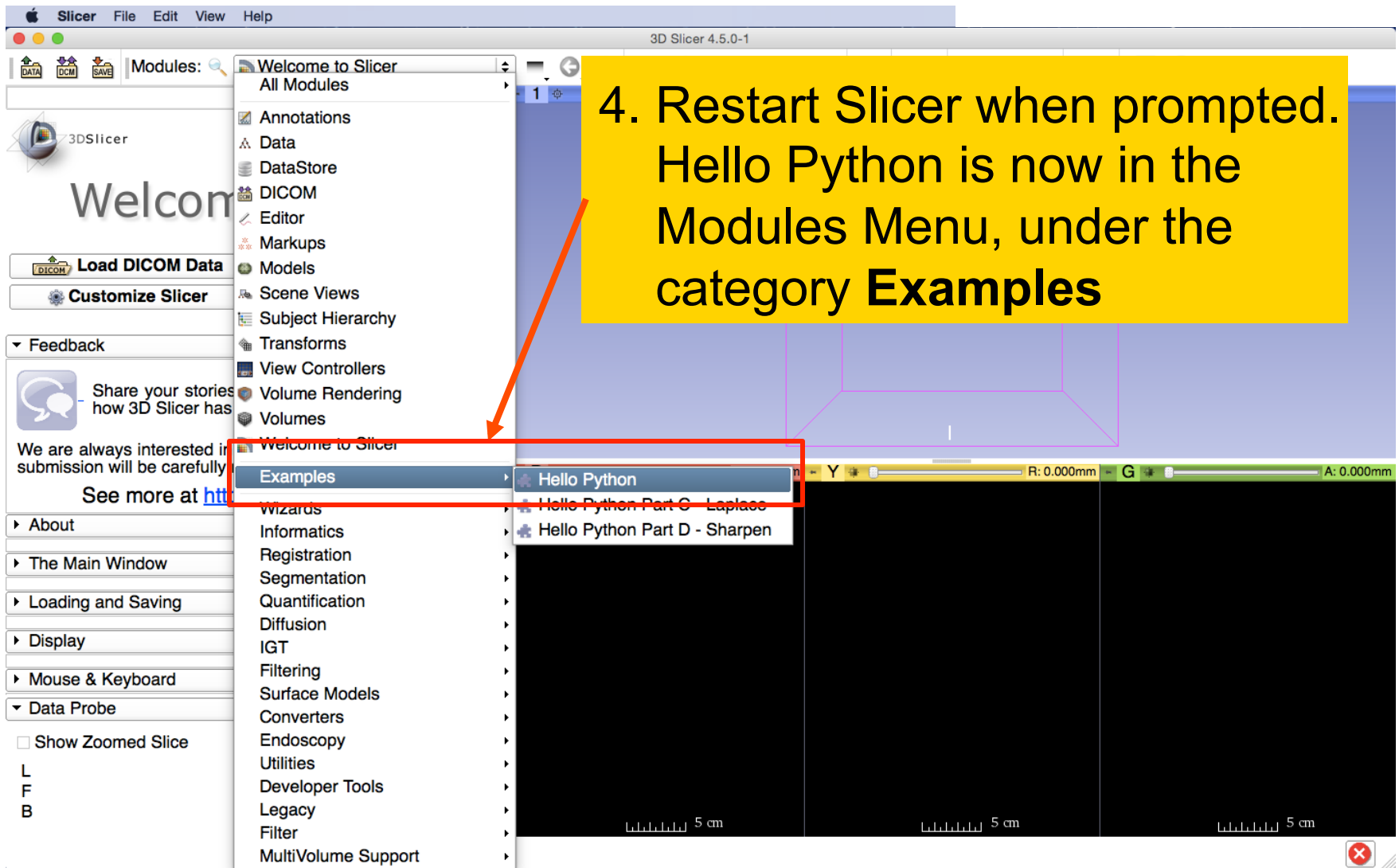
Integrating HelloPython



3. Add the path to the directory containing HelloPython.py (when selecting the directory, the HelloWorld.py file itself will not be displayed)



Integrating HelloPython



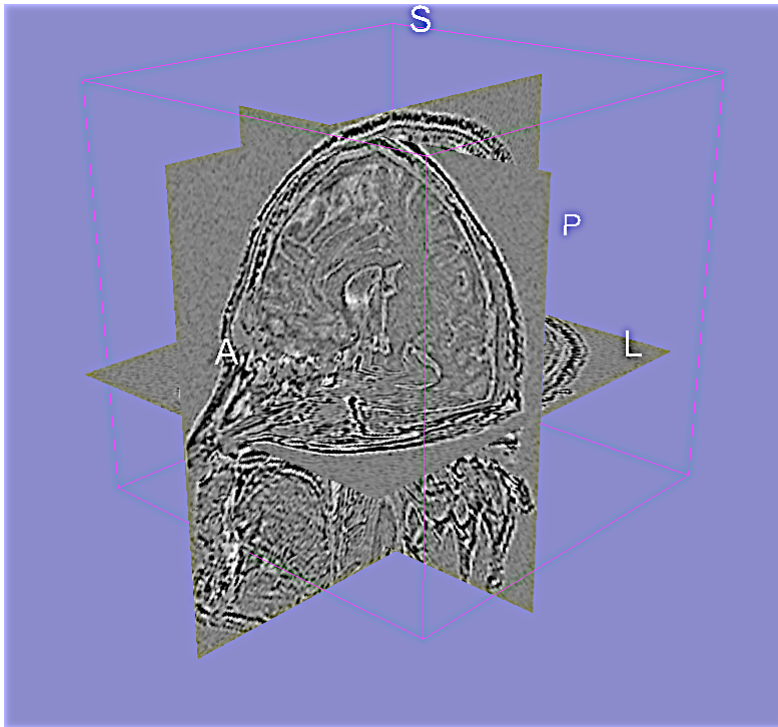


HelloPython in Slicer

The screenshot shows the 3DSlicer application window. The 'Modules' panel on the left has 'Hello Python' selected. Below it, the 'Help & Acknowledgement' section is expanded, showing a 'Help' tab with a text area containing copyright information and a 'Contributors' list. Below this is a 'A collapsible button' section with a 'Hello world' button. A yellow callout box points to the 'Help' and 'Acknowledgement' tabs with the text: 'Click on Help and Acknowledgment in the Hello Python module'. Another yellow callout box points to the 'A Collapsible button' tab with the text: 'Expand the A Collapsible button tab, and click on the Hello World button'. A small dialog box titled 'Hello World!' with an 'OK' button is visible in the center of the main 3D view area. The bottom of the window shows the 'Data Probe' panel with a 'Show Zoomed Slice' checkbox and 'L', 'F', 'B' buttons. The main 3D view area shows a wireframe box and coordinate axes (R, P, L, I) with scale bars (5 cm) and coordinate values (R: 0.000mm, G: 0.000mm, A: 0.000mm).

Click on **Help** and **Acknowledgment** in the Hello Python module

Expand the **A Collapsible button** tab, and click on the **Hello World** button



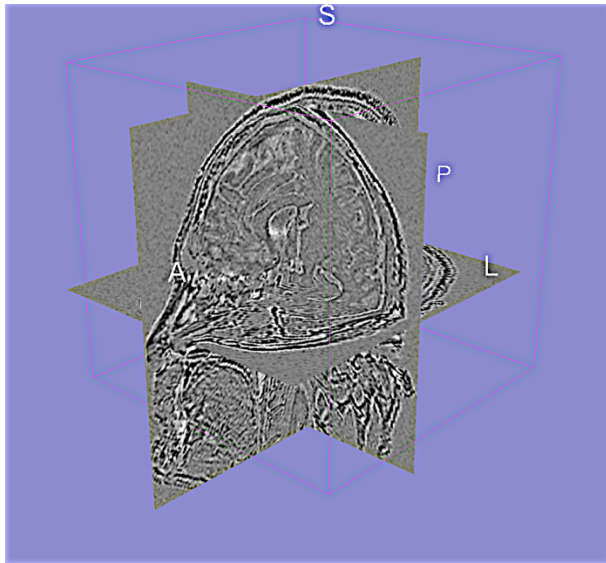
Part C:

Implementing the Laplace* Operator

*named after Pierre-Simon, Marquis de Laplace (1749-1827)

Overview

The goal of this section is to build an image analysis module that implements a Laplacian filter on volume data



- Use qMRML widgets: widgets that automatically track the state of the Slicer MRML scene
- Use VTK filters to manipulate volume data

HelloLaplace.py

```
from __main__ import vtk, qt, ctk, slicer

#
# HelloLaplace
#

class HelloLaplace:
    def __init__(self, parent):
        parent.title = "Hello Python Part C - Laplace"
        parent.categories = ["Examples"]
        parent.dependencies = []
        parent.contributors = ["Jean-Christophe Fillion-Robin (Kitware)",
                               "Steve Pieper (Isomics)",
                               "Sonia Pujol (BWH)"] # replace with "Firstname Lastname (Org)"

        parent.helpText = """
        Example of scripted loadable extension for the HelloLaplace tutorial.
        """
        parent.acknowledgementText = """
        This file was originally developed by Jean-Christophe Fillion-Robin, Kitware Inc.,
        Steve Pieper, Isomics, Inc., and Sonia Pujol, Brigham and Women's Hospital and was
        partially funded by NIH grant 3P41RR013218-12S1 (NAC) and is part of the National Alliance
        for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the
        NIH Roadmap for Medical Research, Grant U54 EB005149.""" # replace with organization, grant and thanks.
        self.parent = parent

#
# qHelloPythonWidget
#

class HelloLaplaceWidget:
    def __init__(self, parent = None):
        # Collapsible button
        self.laplaceCollapsibleButton = ctk.ctkCollapsibleButton()
        self.laplaceCollapsibleButton.text = "Laplace Operator"
        self.layout.addWidget(self.laplaceCollapsibleButton)

        # Layout within the laplace collapsible button
        self.laplaceFormLayout = qt.QFormLayout(self.laplaceCollapsibleButton)

        # the volume selectors
        self.inputFrame = qt.QFrame(self.laplaceCollapsibleButton)
        self.inputFrame.setLayout(qt.QHBoxLayout())
        self.laplaceFormLayout.addWidget(self.inputFrame)
        self.inputSelector = qt.QLabel("Input Volume: ")
        self.inputFrame
```

Open the file **HelloLaplace.py**
located in the directory **helloPython\code**

Module GUI (Part 1)

```
def setup(self):
    # Collapsible button
    self.laplaceCollapsibleButton = ctk.ctkCollapsibleButton()
    self.laplaceCollapsibleButton.text = "Laplace Operator"
    self.layout.addWidget(self.laplaceCollapsibleButton)

    # Layout within the laplace collapsible button
    self.laplaceFormLayout = qt.QFormLayout(self.laplaceCollapsibleButton)

    # the volume selectors
    self.inputFrame = qt.QFrame(self.laplaceCollapsibleButton)
    self.inputFrame.setLayout(qt.QHBoxLayout())
    self.laplaceFormLayout.addWidget(self.inputFrame)
    self.inputSelector = qt.QLabel("Input Volume: ", self.inputFrame)
    self.inputFrame.layout().addWidget(self.inputSelector)
    self.inputSelector = slicer.qMRMLNodeComboBox(self.inputFrame)
    self.inputSelector.nodeTypeNames = ( "vtkMRMLScalarVolumeNode", "" )
    self.inputSelector.addEnabled = False
    self.inputSelector.removeEnabled = False
    self.inputSelector.setMRMLScene( slicer.mrmlScene )
    self.inputFrame.layout().addWidget(self.inputSelector)
```

This code is
provided in
the template

Module GUI (Part 2)

```
self.outputFrame = qt.QFrame(self.laplaceCollapsibleButton)
self.outputFrame.setLayout(qt.QHBoxLayout())
self.laplaceFormLayout.addWidget(self.outputFrame)
self.outputSelector = qt.QLabel("Output Volume: ", self.outputFrame)
self.outputFrame.layout().addWidget(self.outputSelector)
self.outputSelector = slicer.qMRMLNodeComboBox(self.outputFrame)
self.outputSelector.nodeTypeNames = ( "vtkMRMLScalarVolumeNode", "" )
self.outputSelector.setMRMLScene( slicer.mrmlScene )
self.outputFrame.layout().addWidget(self.outputSelector)
```

```
# Apply button
laplaceButton = qt.QPushButton("Apply Laplace")
laplaceButton.setToolTip("Run the Laplace Operator.")
self.laplaceFormLayout.addWidget(laplaceButton)
laplaceButton.connect('clicked(bool)', self.onApply)
```

```
# Add vertical spacer
self.layout.addStretch(1)
```

```
# Set local var as instance attribute
self.laplaceButton = laplaceButton
```

This code is
provided in
the template

In More Detail

- **CTK** is a Qt Add-On Library with many useful widgets, particularly for visualization and medical imaging see <http://commontk.org>
- **Qt Widgets, Layouts**, and **Options** are well documented at <http://qt-project.org>
- **qMRMLNodeComboBox** is a powerful slicer widget that monitors the scene and allows you to select/create nodes of specified types (*example: here we use Volumes = vtkMRMLScalarVolumeNode*)

Processing Code

```
def onApply(self):  
    inputVolume = self.inputSelector.currentNode()  
    outputVolume = self.outputSelector.currentNode()  
    if not (inputVolume and outputVolume):  
        qt.QMessageBox.critical(slicer.util.mainWindow(),  
                                'Laplace', 'Input and output volumes are required for Laplacian')  
    return
```

Add this
code

```
    laplacian = vtk.vtkImageLaplacian()  
    laplacian.SetInputData (inputVolume.GetImageData())  
    laplacian.SetDimensionality(3)  
    laplacian.Update()  
    ijkToRAS = vtk.vtkMatrix4x4()  
    inputVolume.GetIJKToRASMatrix(ijkToRAS)  
    outputVolume.SetIJKToRASMatrix(ijkToRAS)  
    outputVolume.SetAndObserveImageData(laplacian.GetOutput())  
    # make the output volume appear in all the slice views  
    selectionNode = slicer.app.applicationLogic().GetSelectionNode()  
    selectionNode.SetReferenceActiveVolumeID(outputVolume.GetID())  
    slicer.app.applicationLogic().PropagateVolumeSelection(0)
```

In More Detail

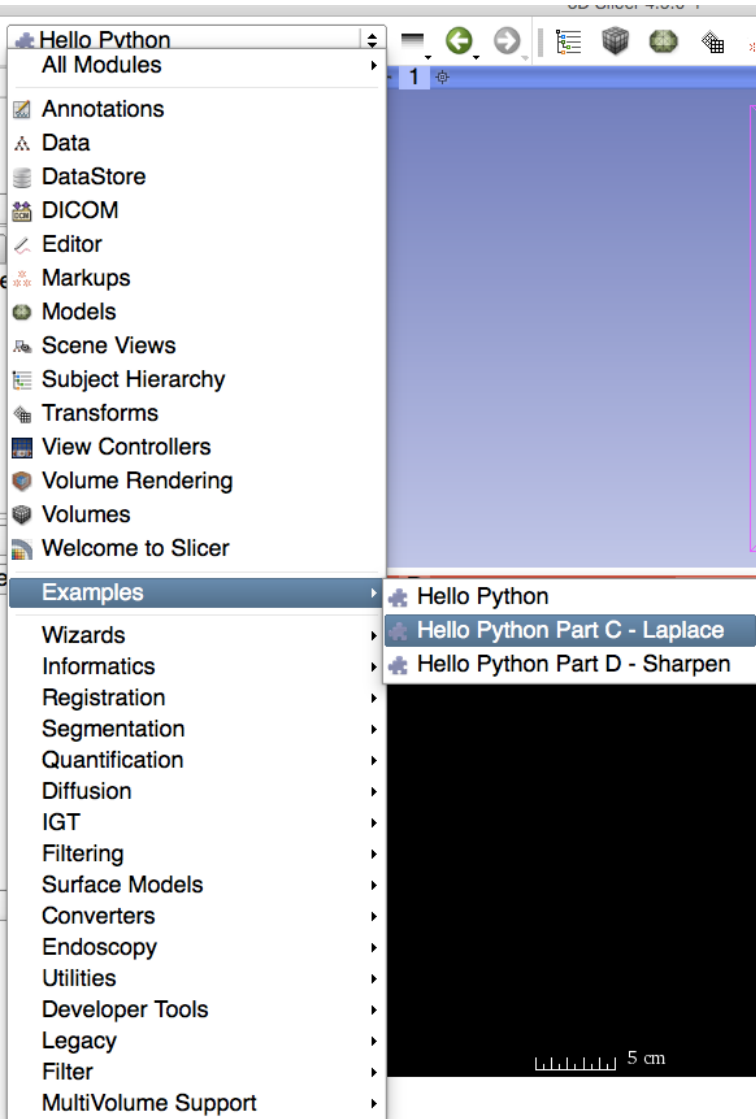
- **vtkImageLaplacian** is a **vtkImageAlgorithm** operates on **vtkImageData** (see <http://vtk.org>)
- **vtkMRMLScalarVolumeNode** is a Slicer MRML class that contains **vtkImageData**, plus orientation information **ijkToRAS** matrix (see http://www.slicer.org/slicerWiki/index.php/Coordinate_systems)

In More Detail (Continued)

- Global **licer** package gives python access to:
 - GUI (via **licer.app**)
 - modules (via **licer.modules**)
 - data (via **licer.mrmlScene**)
- **licer.app.applicationLogic()** provides helper utilities for manipulating Slicer state

Go To Laplace Module

Restart Slicer and select module.
Note that combobox is empty.



▼ Help & Acknowledgement

Help

Acknowledgement

This file was originally developed by Jean-Christophe Fillion-Robin, Kitware Inc., Steve Pieper, Isomics, Inc., and Sonia Pujol, Brigham and Women's Hospital and was partially funded by NIH grant 3P41RR013218-12S1 (NAC) and is part of the National Alliance for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

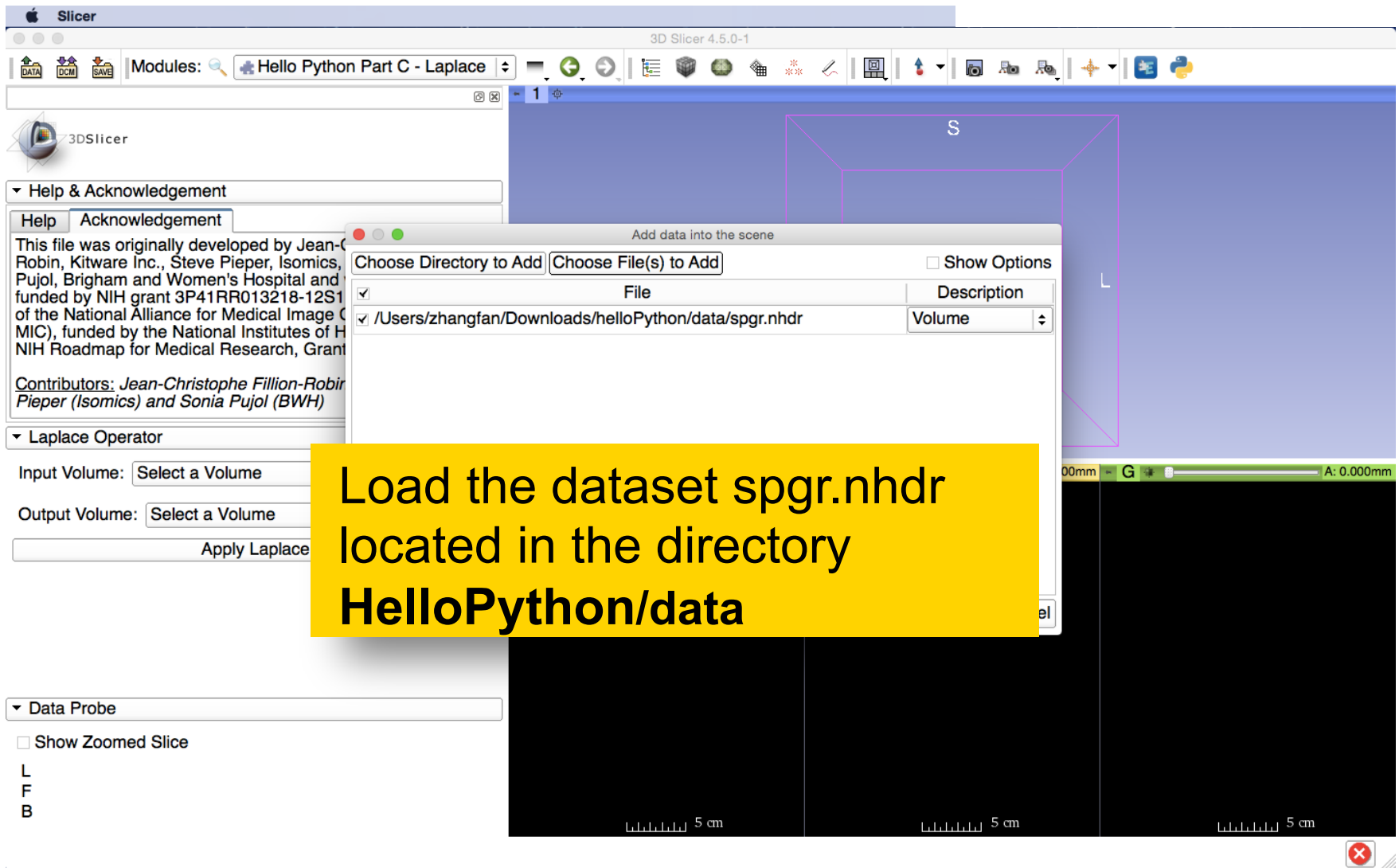
Contributors: *Jean-Christophe Fillion-Robin (Kitware), Steve Pieper (Isomics) and Sonia Pujol (BWH)*

▼ Laplace Operator

Input Volume:

Output Volume:

Add spgr.nhdr



After Adding Volume

▼ Laplace Operator

Input Volume: spgr

1. Note that Input Volume combobox autoselected new volume

Output Volume: spgr

Create new Volume
Delete current Volume

2. Create new volume for output

Output Volume: Volume

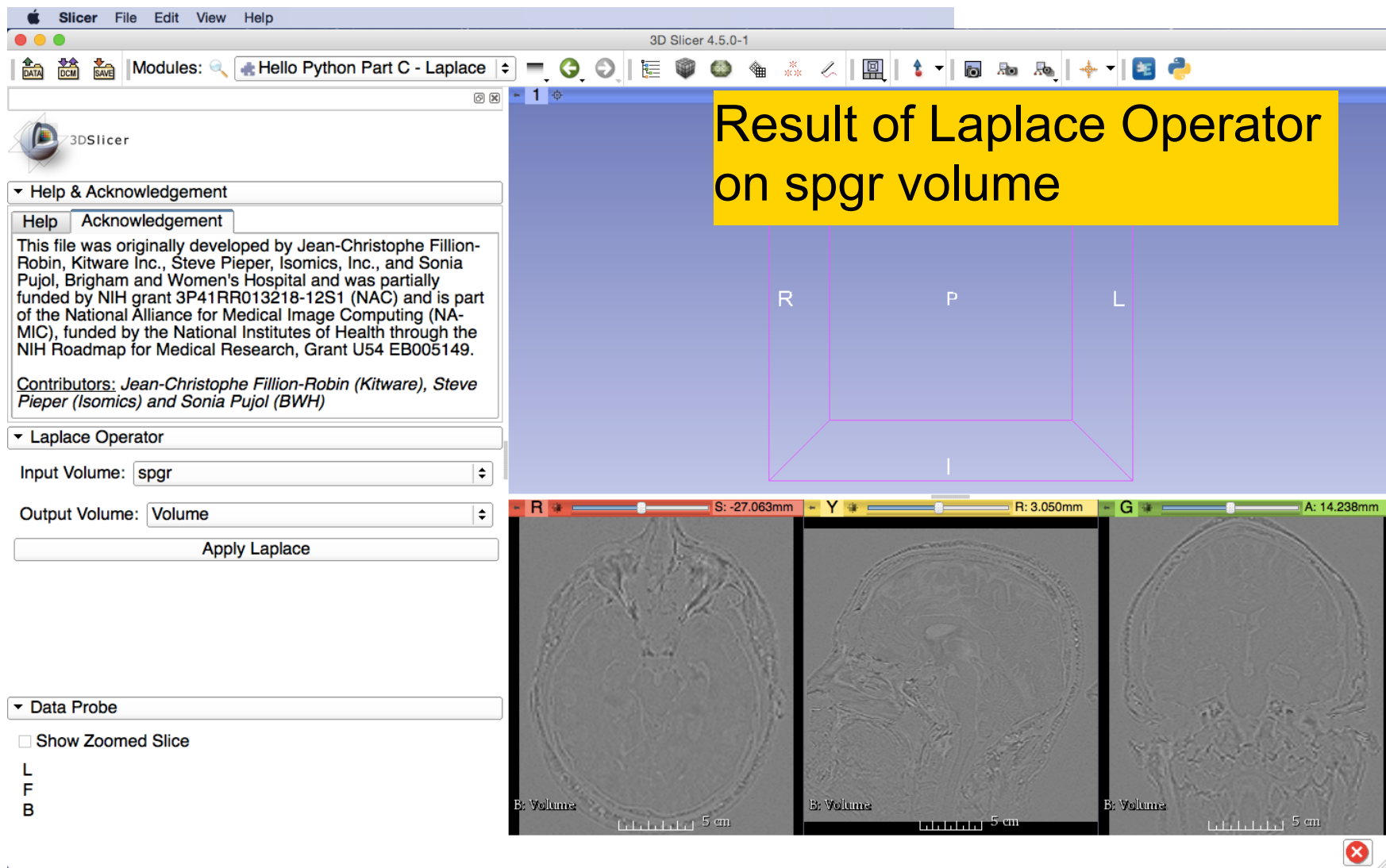
Apply Laplace

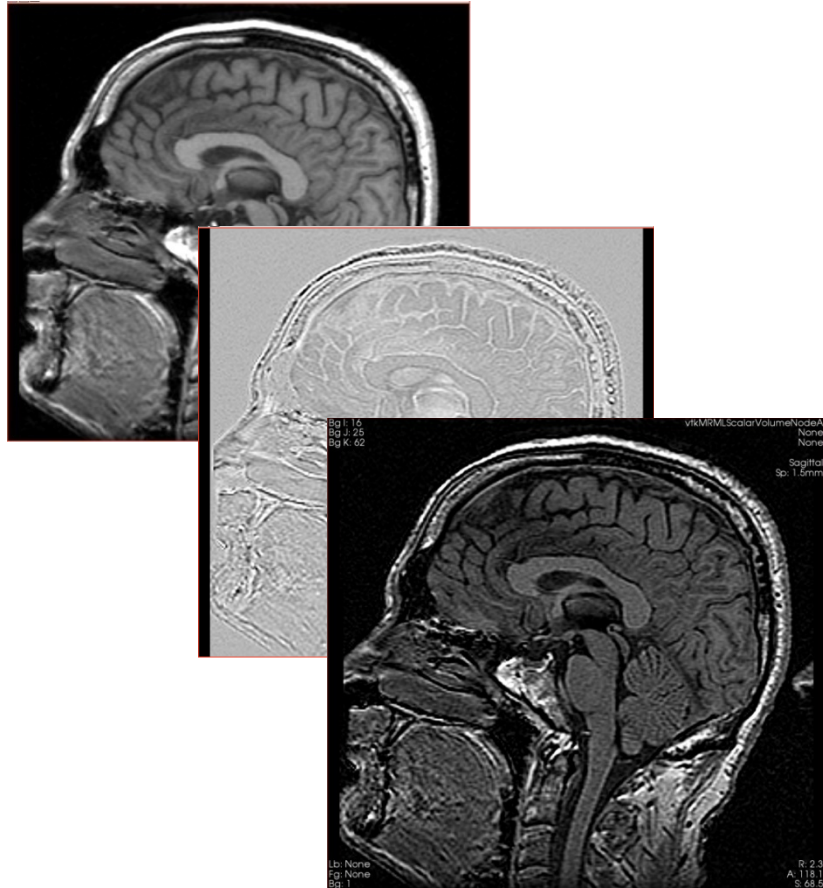
Run the Laplace
Operator.

3. Run the module



Laplace Module



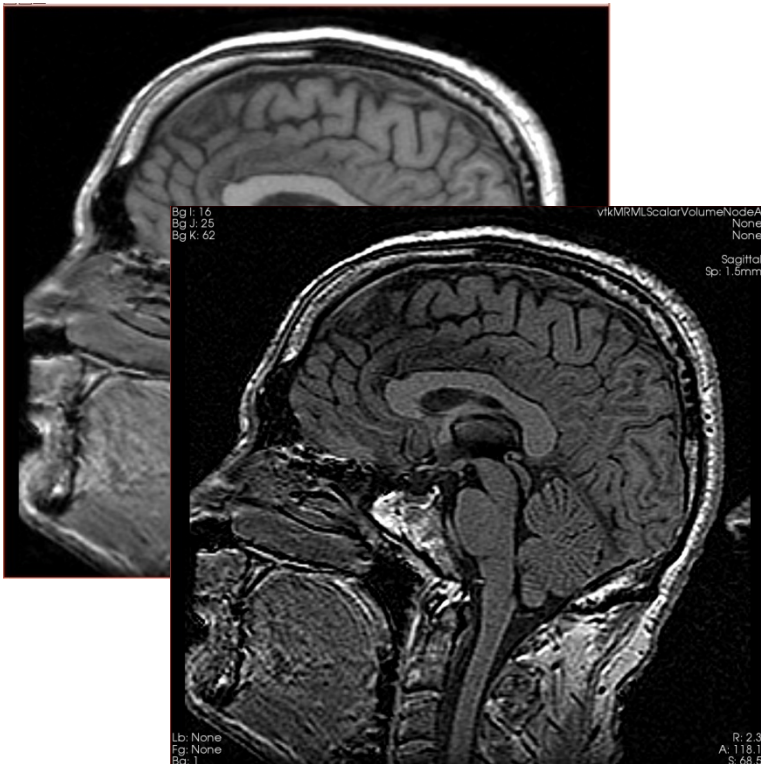


Part D:

Image Sharpening with the Laplace Operator

Overview

The goal of this section is to add a processing option for image sharpening.



- We'll implement this operation using the existing Slicer Command Line Module
- 'Subtract Scalar Volumes'

HelloSharpen.py

```
from __main__ import vtk, qt, ctk, slicer

#
# HelloSharpen
#

class HelloSharpen:
    def __init__(self, parent):
        parent.title = "Hello Python Part D - Sharpen"
        parent.categories = ["Examples"]
        parent.dependencies = []
        parent.contributors = ["Jean-Christophe Fillion-Robin (Kitware)",
                               "Steve Pieper (Isomics)",
                               "Sonia Pujol (BWH)"] # replace with "Firstname Lastname (Org)"

        parent.helpText = """
        Example of scripted loadable extension for the HelloSharpen tutorial.
        """
        parent.acknowledgementText = """
        This file was originally developed by Jean-Christophe Fillion-Robin, Kitware Inc.,
        Steve Pieper, Isomics, Inc., and Sonia Pujol, Brigham and Women's Hospital and was
        partially funded by NIH grant 3P41RR013218-12S1 (NAC) and is part of the National Alliance
        for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the
        NIH Roadmap for Medical Research, Grant U54 EB005149.""" # replace with organization, grant and thanks.
        self.parent = parent

#
# qHelloPythonWidget
#

class HelloSharpenWidget:
    def __init__(self, parent = None):
```

Open the file **HelloSharpen.py**
located in the directory **helloPython\code**

```
def setup(self):
    # Collapsible button
    self.laplaceCollapsibleButton = ctk.ctkCollapsibleButton()
    self.laplaceCollapsibleButton.text = "Sharpen Operator"
    self.layout.addWidget(self.laplaceCollapsibleButton)

    # Layout within the laplace collapsible button
    self.laplaceFormLayout = qt.QFormLayout(self.laplaceCollapsibleButton)

    #
    # the volume selectors
    #
```

Add to Module GUI

```
...  
self.outputSelector.setMRMLScene( slicer.mrmlScene )  
self.outputFrame.layout().addWidget(self.outputSelector)
```

Add this
Text in
section A

```
self.sharpen = qt.QCheckBox("Sharpen", self.laplaceCollapsibleButton)  
self.sharpen.setToolTip = "When checked, subtract laplacian from input volume"  
self.sharpen.checked = True  
self.laplaceFormLayout.addWidget(self.sharpen)
```

```
# Apply button  
laplaceButton = qt.QPushButton("Apply")  
laplaceButton.setToolTip = "Run the Laplace or Sharpen Operator."  
...
```



Add to Processing Code

Add this
Text in
section B

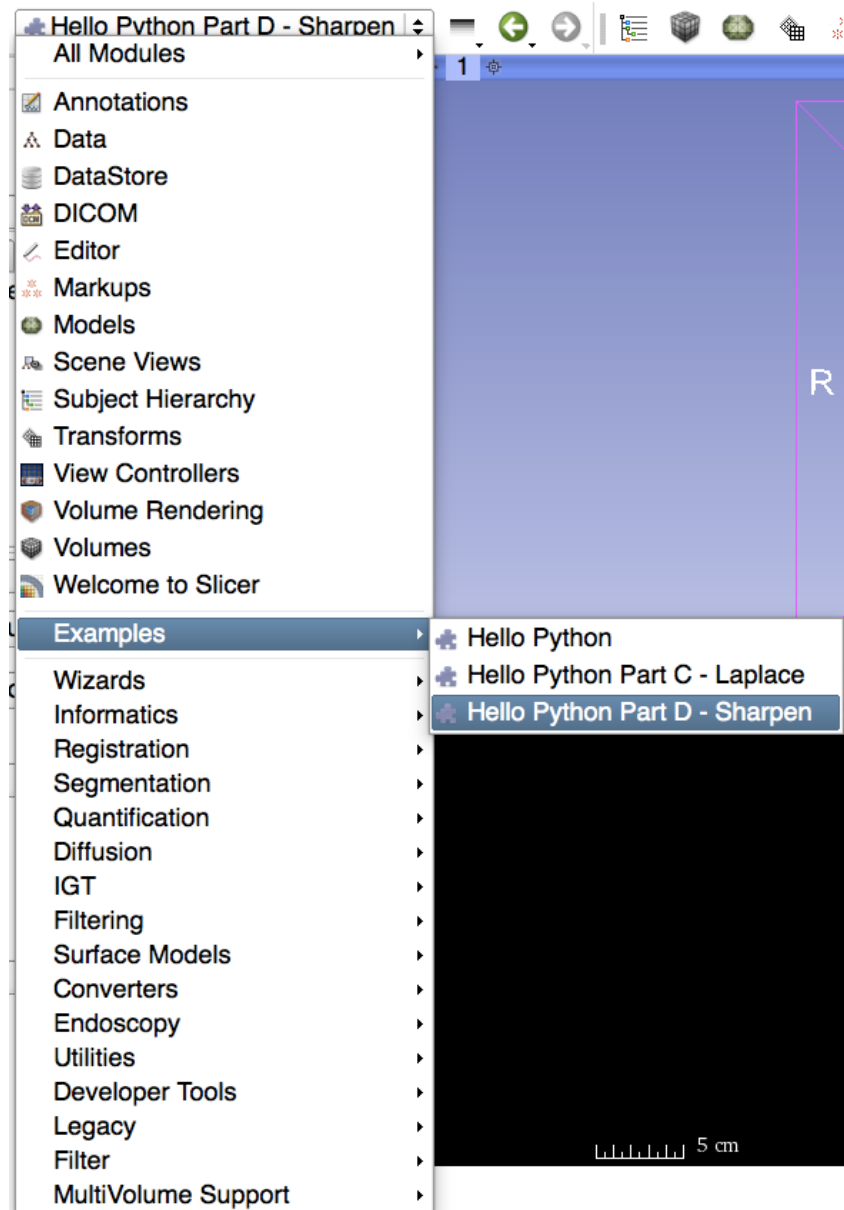
```
...
outputVolume.SetAndObserveImageData(laplacian.GetOutput())
# optionally subtract laplacian from original image
if self.sharpen.checked:
    parameters = {}
    parameters['inputVolume1'] = inputVolume.GetID()
    parameters['inputVolume2'] = outputVolume.GetID()
    parameters['outputVolume'] = outputVolume.GetID()
    slicer.cli.run( slicer.modules.subtractscalarvolumes, None,
parameters, wait_for_completion=True )
# make the output volume appear in all the slice views
selectionNode = slicer.app.applicationLogic().GetSelectionNode()

selectionNode.SetReferenceActiveVolumeID(outputVolume.GetID())
slicer.app.applicationLogic().PropagateVolumeSelection(0)
```

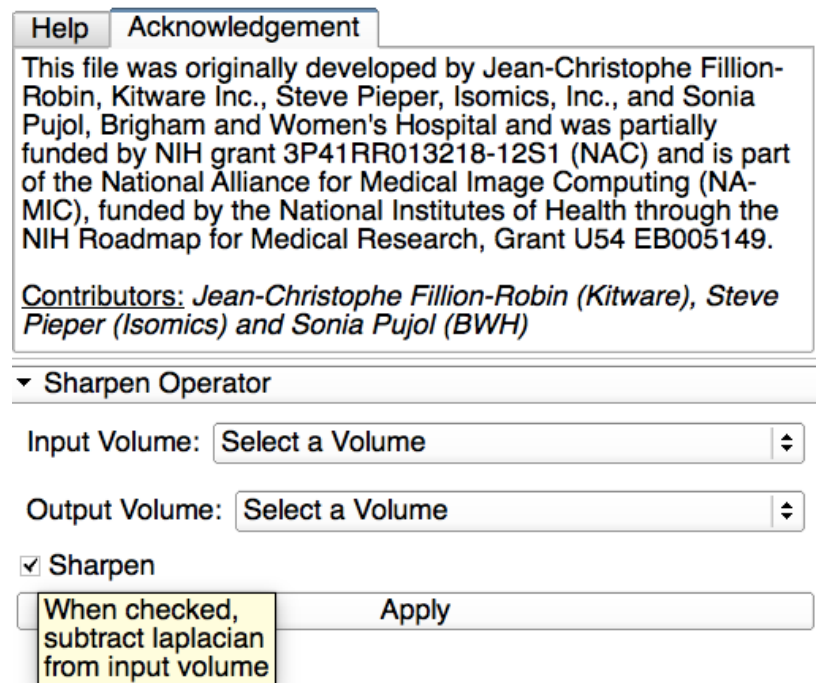
In More Detail

- **slicer.cli** gives access to Command Line Interface (CLI) modules
- CLI modules allow packaging of arbitrary C++ code (often ITK-based) into slicer with automatically generated GUI and python wrapping

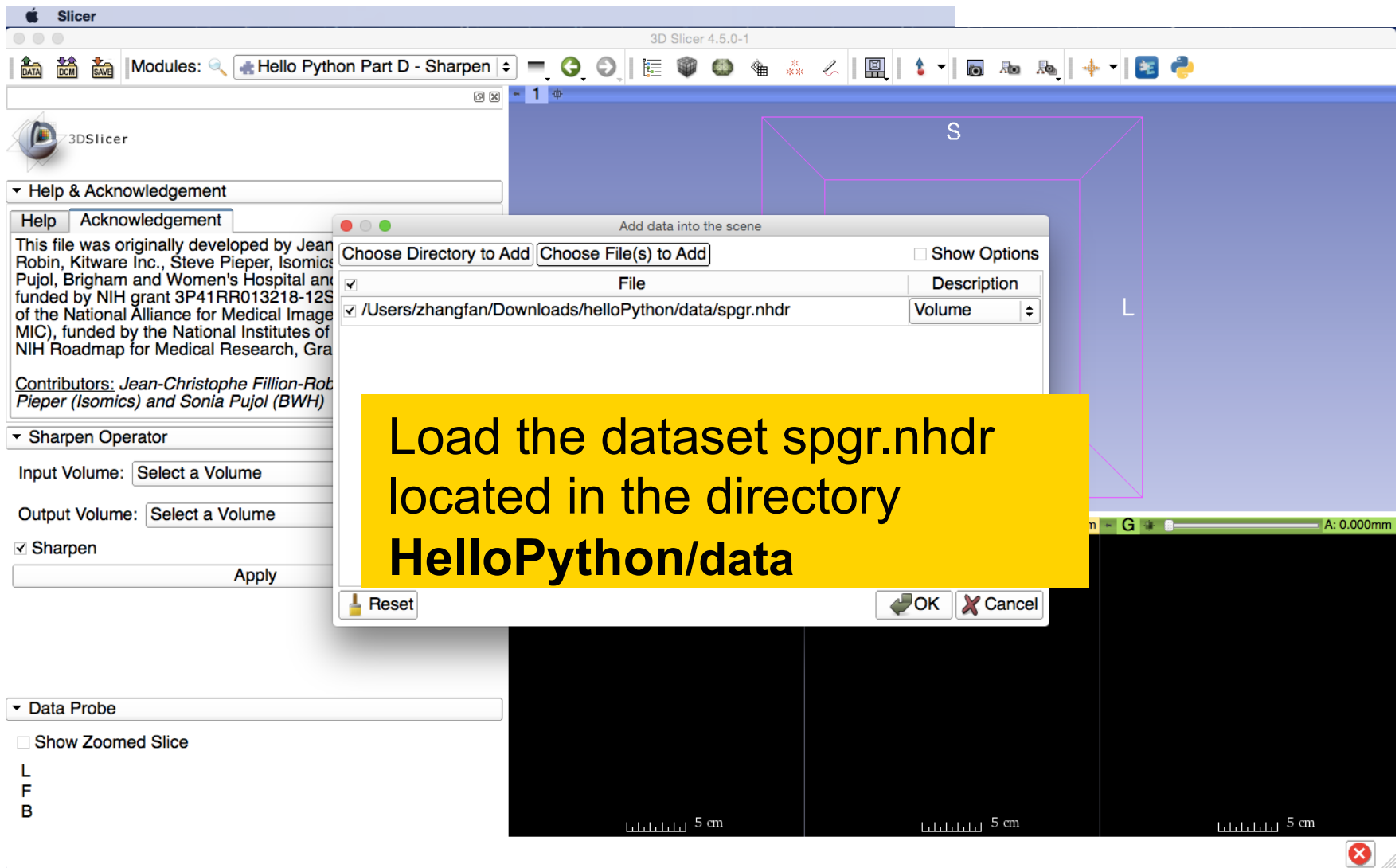
Go To Sharpen Module



Restart Slicer and select module.
Note the new sharpen check box



Add spgr.nhdr



After Adding Volume

▼ Laplace Operator

Input Volume: spgr

1. Note that Input Volume combobox autoselected new volume

Output Volume: spgr

Create new Volume
Delete current Volume

2. Create new volume for output

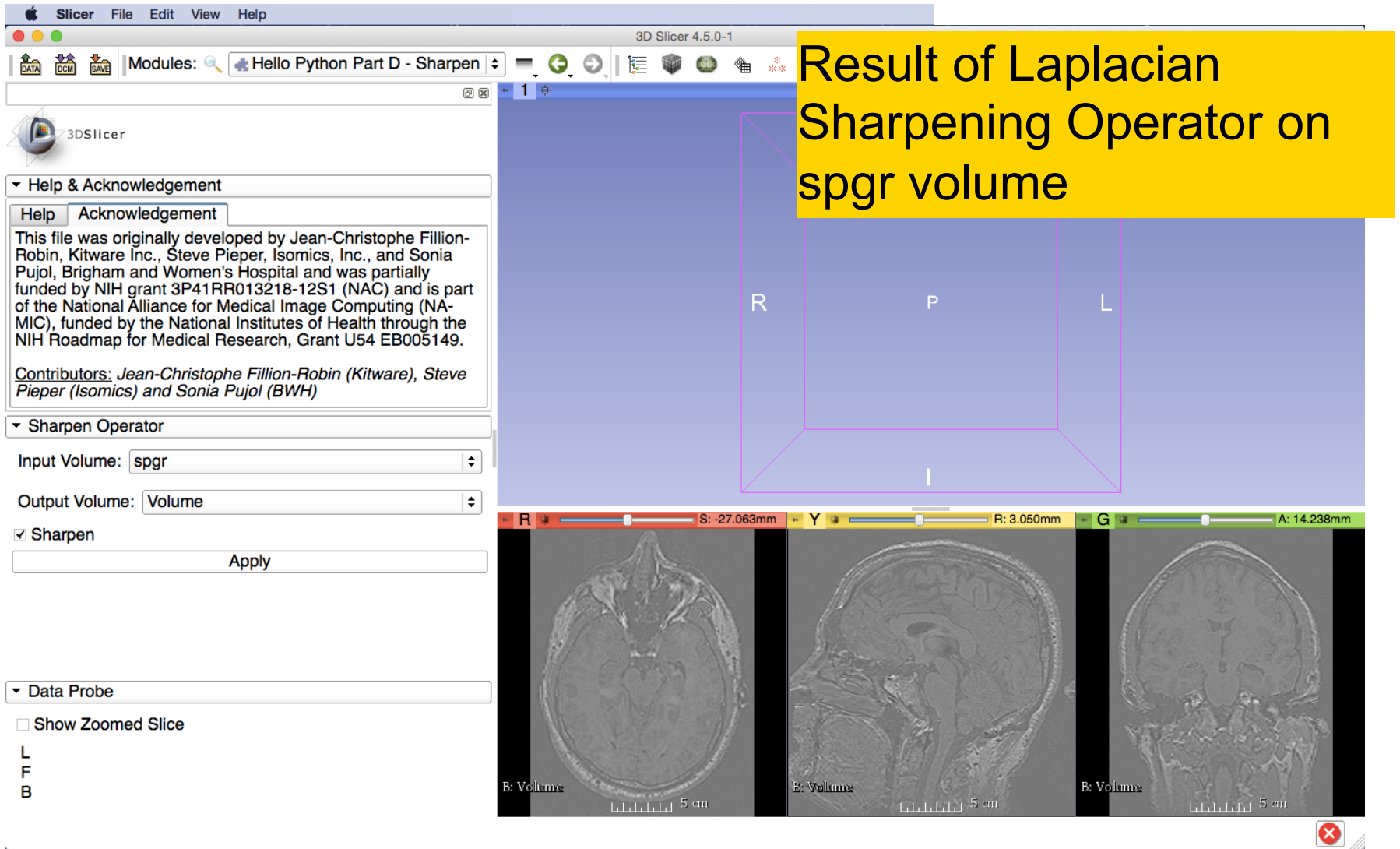
☒ Sharpen

Apply

3. Run the module

Run the Laplace or Sharpen Operator.

Sharpen Module



The screenshot displays the 3D Slicer 4.5.0-1 software interface. The top menu bar includes 'Slicer', 'File', 'Edit', 'View', and 'Help'. Below the menu, the 'Modules' panel shows 'Hello Python Part D - Sharpen' selected. The main 3D view area shows a blue wireframe box with axes labeled 'R' (Right), 'P' (Posterior), 'L' (Left), and 'I' (Inferior). A yellow text box in the upper right corner of the 3D view area reads: 'Result of Laplacian Sharpening Operator on spgr volume'. On the left side, the 'Help & Acknowledgement' panel is expanded, showing the 'Sharpen Operator' section. This section includes 'Input Volume: spgr', 'Output Volume: Volume', and a checked 'Sharpen' checkbox. Below these is an 'Apply' button. The 'Data Probe' panel at the bottom left shows 'Show Zoomed Slice' unchecked and lists 'L', 'F', and 'B' views. At the bottom of the 3D view, three orthogonal slices (axial, sagittal, and coronal) are displayed, each with a 5 cm scale bar. The axial slice is labeled 'B: Volume' and 'S: -27.063mm'. The sagittal slice is labeled 'B: Volume' and 'R: 3.050mm'. The coronal slice is labeled 'B: Volume' and 'A: 14.238mm'. A red 'X' icon is visible in the bottom right corner of the 3D view area.

Result of Laplacian Sharpening Operator on spgr volume

Input Volume: spgr

Output Volume: Volume

☒ Sharpen

Apply

▼ Data Probe

☐ Show Zoomed Slice

L
F
B

B: Volume S: -27.063mm R: 3.050mm A: 14.238mm

B: Volume B: Volume B: Volume

5 cm 5 cm 5 cm

Sharpen Module

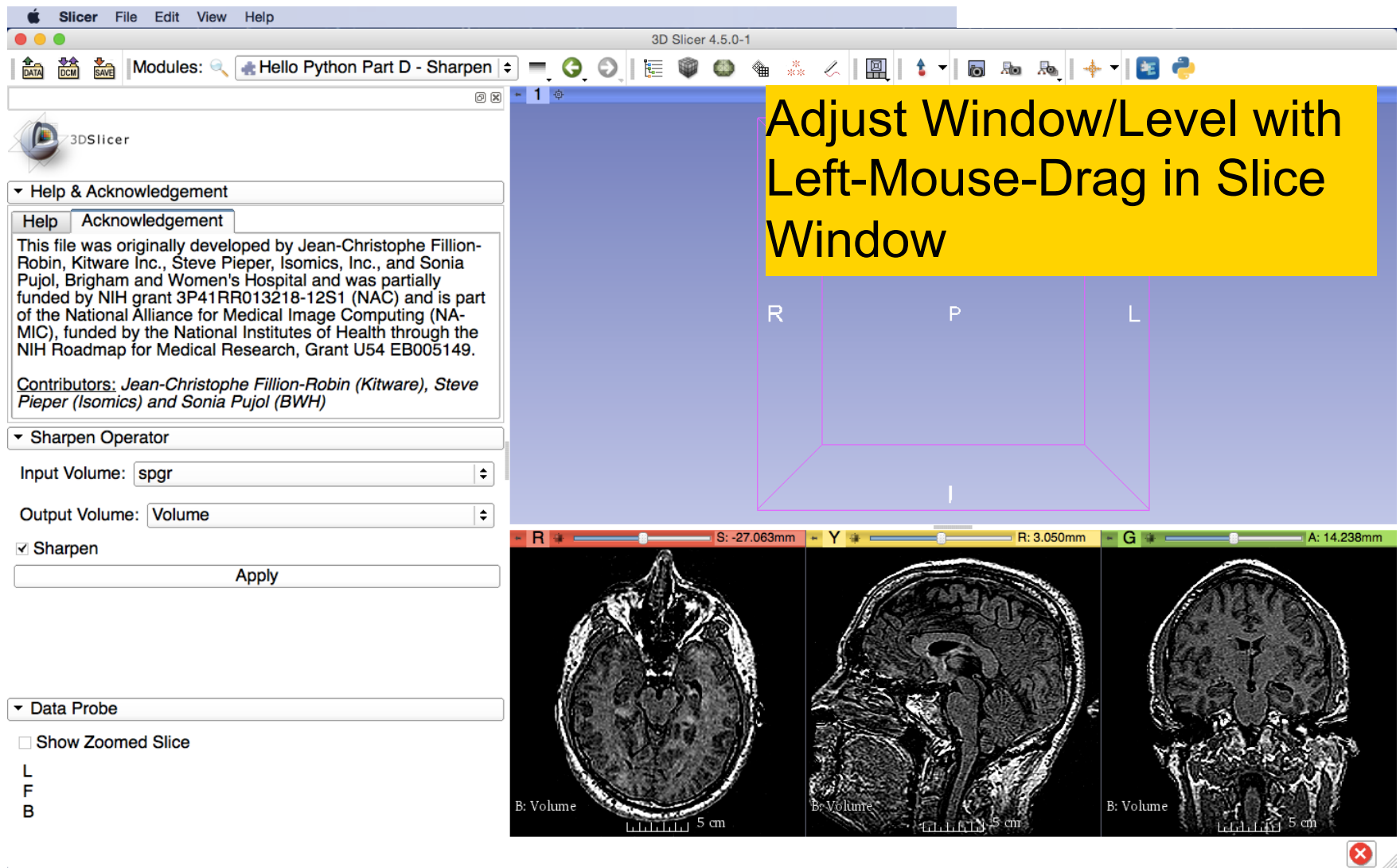
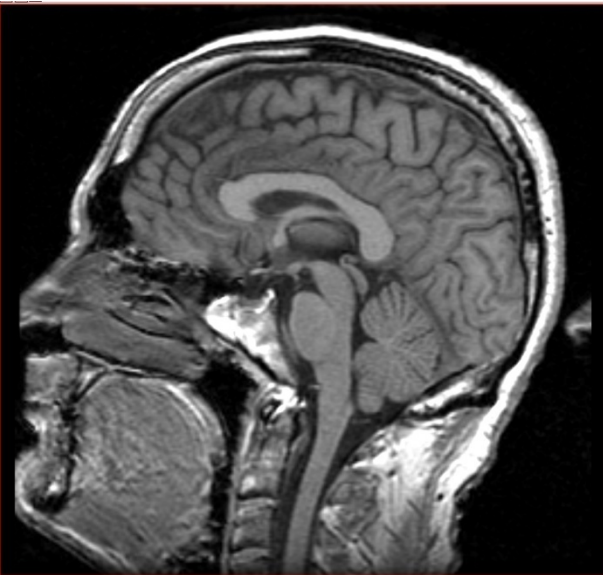
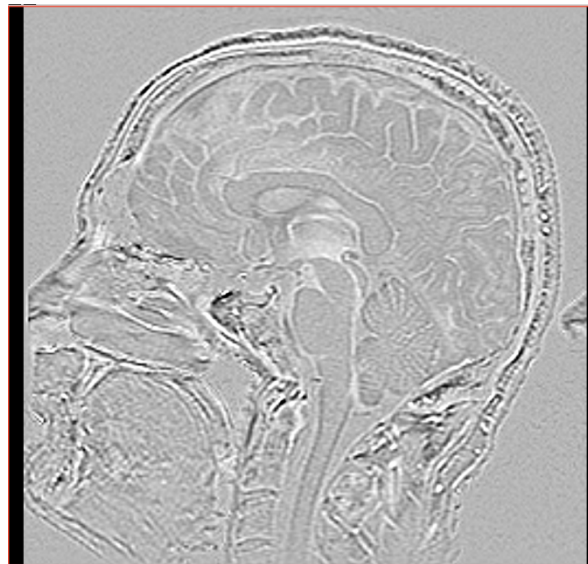


Image Sharpening

original



Laplacian



Laplacian filtered



Going Further

- Explore numpy for numerical array manipulation
- Review Endoscopy Module for interactive data exploration using MRML and VTK
- See the Editor Module for interactive segmentation examples
- Explore SimpleITK for image processing using ITK

Conclusion

This course demonstrated how to program custom behavior in Slicer with Python



Acknowledgments



National Alliance for Medical Image Computing

NIH U54EB005149



Neuroimage Analysis Center

NIH P41RR013218

Fan Zhang, M.Sc., University of Sydney