



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

Programming in Slicer4

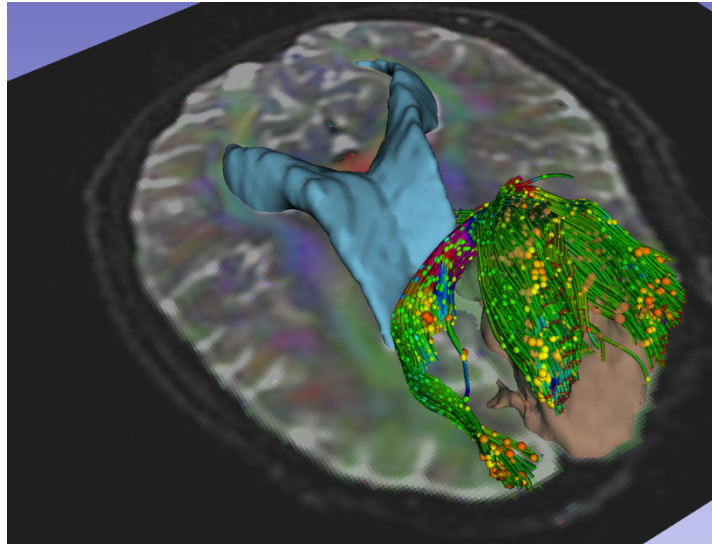
Sonia Pujol, Ph.D.
Surgical Planning Laboratory,
Harvard Medical School

Steve Pieper, Ph.D.
Isomics Inc.

The NA-MIC Kit



3D Slicer version 4 (Slicer4.4)



- 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

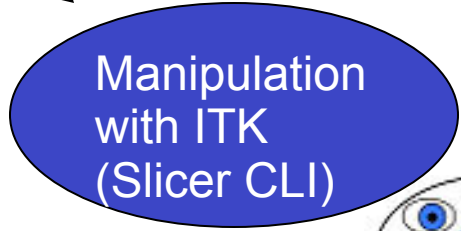
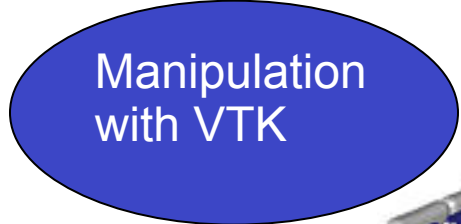
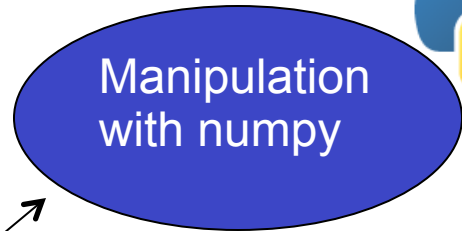
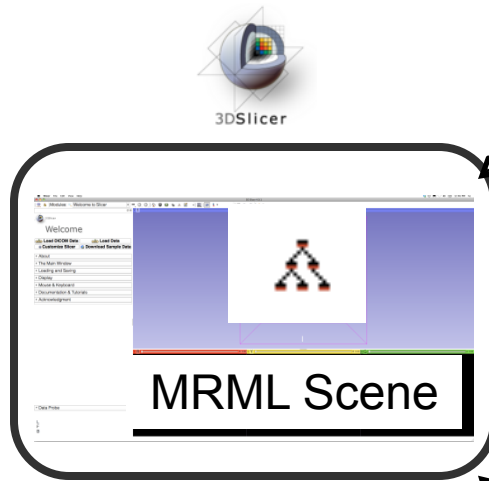
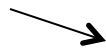
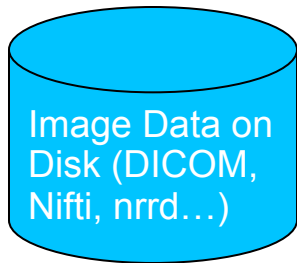
- Python scripted modules allow more **interactive functionalities** (e.g. 'Flythrough' in Endoscopy module) and **rapid prototyping**
- GUI based on Qt libraries accessed via Python



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
<http://wiki.slicer.org/slicerWiki/index.php/Documentation/Nightly>

Processing Examples in this Tutorial



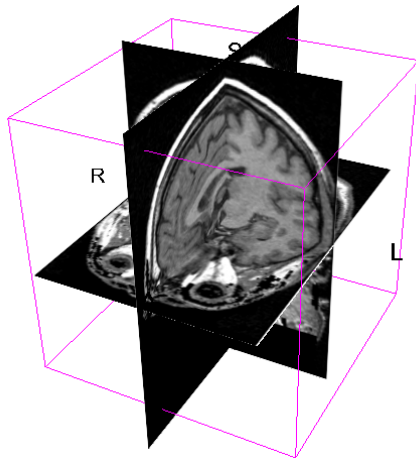
MRML: Medical Reality Markup Language, the Slicer Data Representation

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 Slicer4.4 compendium:
<http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Training>
- Programming experience is required, and some familiarity with Python is essential.

Course Material

Slicer4.4 version available at www.slicer.org



```
class HelloPython:
    def __init__(self, parent):
        self.parent = parent
        self.layout.addWidget(self.helloPythonWidget)

class HelloLaplace:
    def __init__(self, parent):
        self.parent = parent
        self.layout.addWidget(self.helloLaplaceWidget)

class HelloSharpen:
    def __init__(self, parent):
        self.parent = parent
        self.layout.addWidget(self.helloSharpenWidget)

class HelloPythonSlicer4:
    def __init__(self, parent):
        self.parent = parent
        self.layout.addWidget(self.helloPythonWidget)
        self.layout.addWidget(self.helloLaplaceWidget)
        self.layout.addWidget(self.helloSharpenWidget)

def main():
    app = QApplication(sys.argv)
    window = HelloPythonSlicer4()
    window.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```

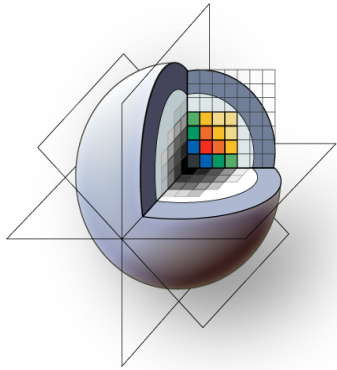
spgr.nhdr
spgr.raw.gz
(124 SPGR images)

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 4.4 includes python 2.7.3 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

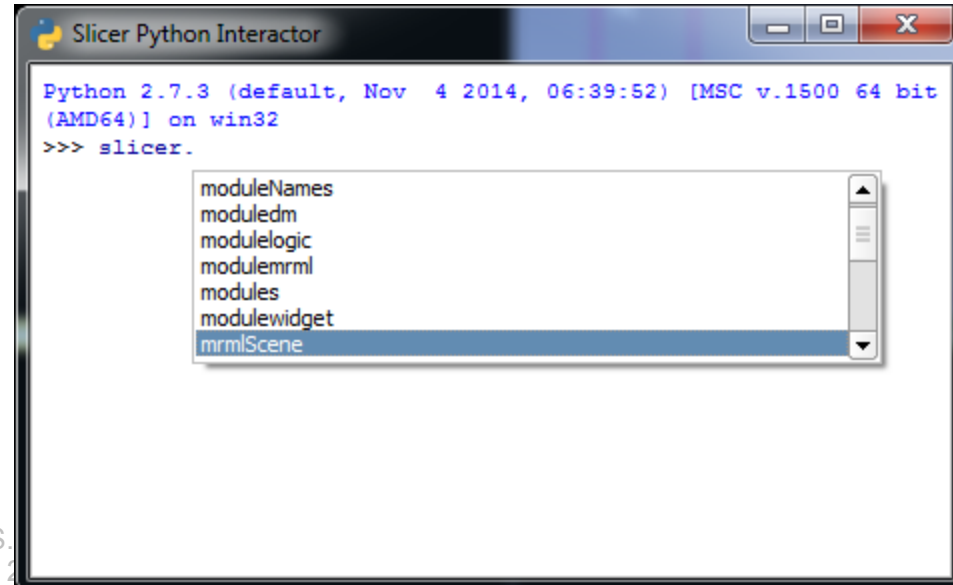
The screenshot shows the 3D Slicer 4.4.0-2014-11-02 interface. The 'View' menu is open, highlighting 'Python Interactor' (Ctrl+3). A red box highlights the 'Python Interactor' button in the top toolbar. A yellow text box with a Python logo icon contains the instruction: 'Select View → Python Interactor, or click the button'. A separate window titled 'Slicer Python Interactor' is open, displaying the Python 2.7.3 shell environment.

Select View → Python Interactor, or click the button 

```
Python 2.7.3 (default, Nov 4 2014, 06:39:52) [MSC v.1500 64 bit (AMD64)] on win32
>>>
```

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 text inside the window reads: "Python 2.7.3 (default, Nov 4 2014, 06:39:52) [MSC v.1500 64 bit (AMD64)] on win32". Below this, the command prompt shows ">>> slicer.". A list of completion options is displayed in a scrollable box: "moduleName", "moduledm", "modulelogic", "modulemri", "modules", "modulewidget", and "mriScene". The "mriScene" option is currently selected and highlighted in blue.

Add Volume

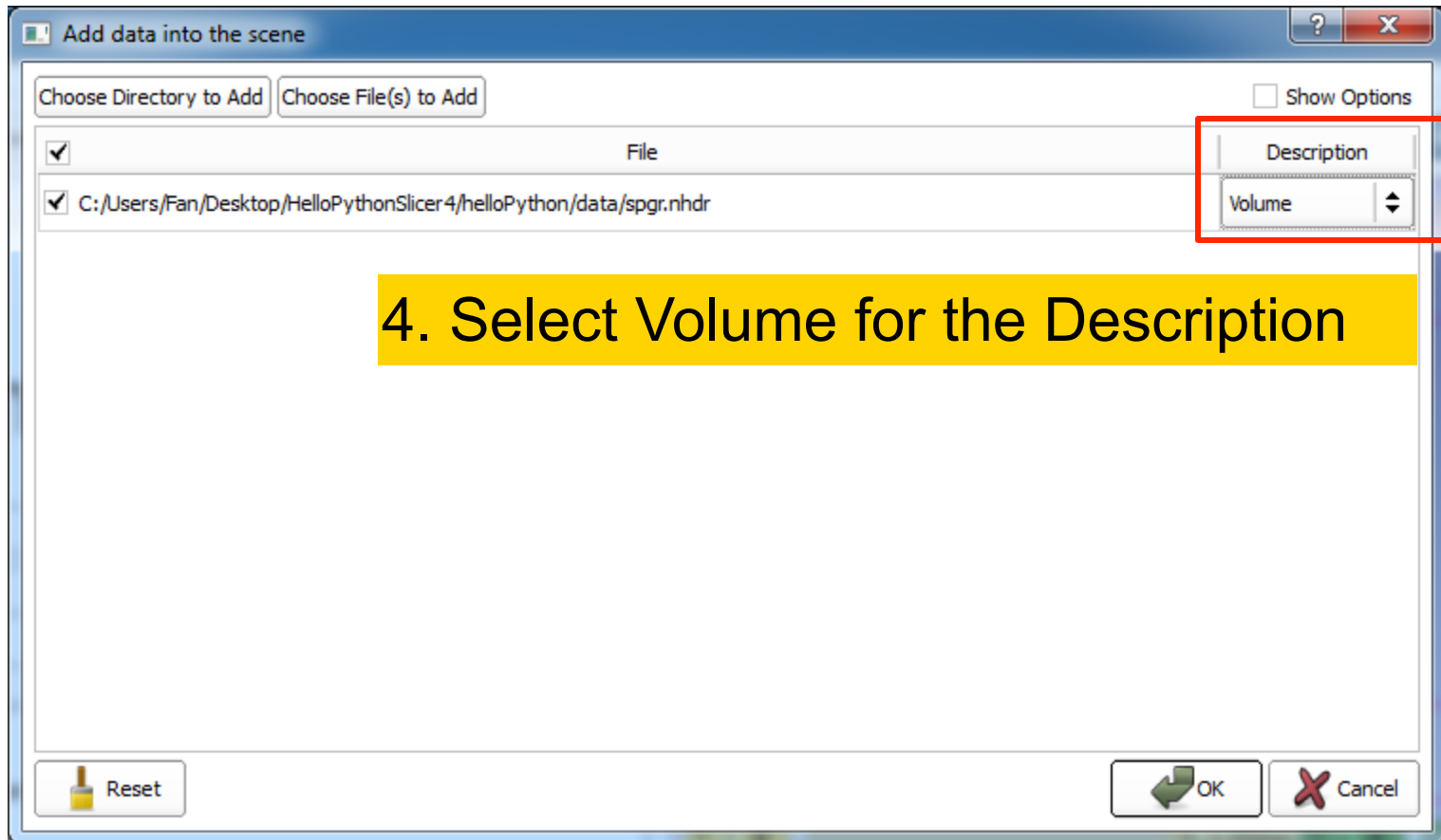
The screenshot shows the Slicer software interface with three yellow callout boxes and red boxes highlighting specific UI elements. The background shows a 3D view of a volume with a purple bounding box and axes labeled S, P, L, and I. The interface includes a menu bar (File, Edit, View, Help), a sidebar with buttons like 'Load DICOM Data' and 'Customize Slicer', and a main workspace. Three windows are overlaid: 1. The 'File' menu is open, with 'Add Data' highlighted. 2. The 'Add data into the scene' dialog is open, with 'Choose File(s) to Add' selected. 3. The 'Open' file dialog is open, showing the directory 'C:\Users\Fan\Desktop\HelloPythonSlicer4\helloPython\data' and the file 'spgr.nhdr' selected.

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

```
Python 2.7.3 (default, Nov 4 2014, 06:39:52) [MSC v.1500 64 bit  
(AMD64)] on win32  
>>> a = slicer.util.array('spgr')  
>>> print(a)
```

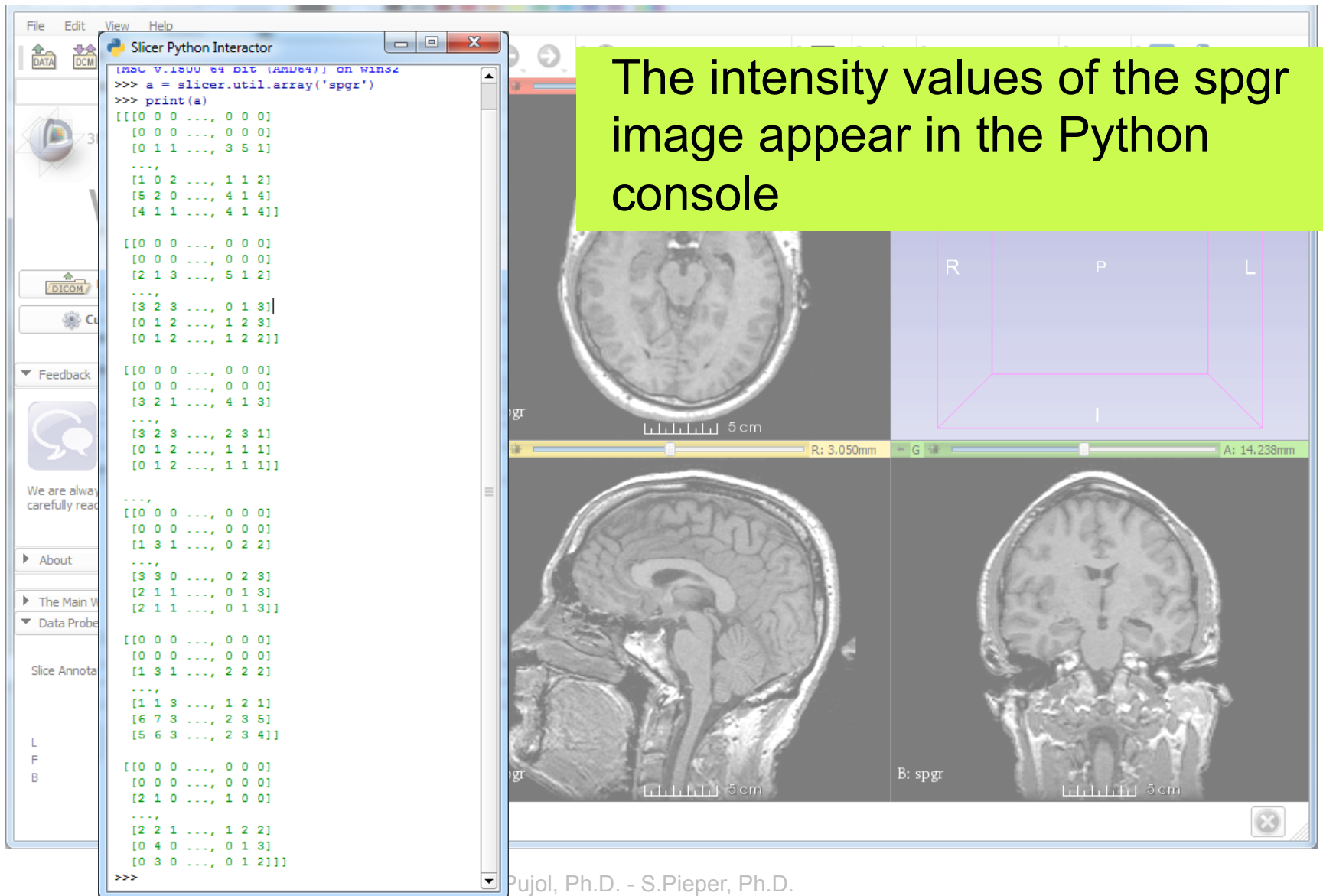
```
a = slicer.util.array('spgr')
```

→ 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 shows the Slicer Python Interceptor window on the left, displaying the following code and output:

```
[PASC V.1800 64 BIT (AMD64)] on Win32
>>> a = slicer.util.array('spgr')
>>> 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]]]]
>>>
```

The main Slicer window on the right displays three MRI slices: an axial slice (top), a sagittal slice (bottom left), and a coronal slice (bottom right). A yellow text box is overlaid on the top right of the Slicer window, containing the text: "The intensity values of the spgr image appear in the Python console". The coronal slice is labeled "B: spgr" and has a 5 cm scale bar. The interface also shows a DICOM viewer, a feedback button, and a data probe panel.

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

The screenshot shows the Slicer Python Interactor window. The code editor contains a 5x5 numpy array and a command to print its minimum and maximum values. A red arrow points from the text above to the command. Below the code editor, there are two MRI brain scan images. The left image is a sagittal view labeled 'B: spgr' and the right image is a coronal view also labeled 'B: spgr'. Both images have a 5cm scale bar.

Access to MRML and Arrays

The screenshot displays the 3D Slicer interface. A 'Slicer Python Interactor' window is open, showing the following code and output:

```
.....  
[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]]]  
  
>>> print(a.min(), a.max())  
(0, 355)  
>>> |
```

A yellow callout box highlights the text: **I min = 0 ; I max = 355**

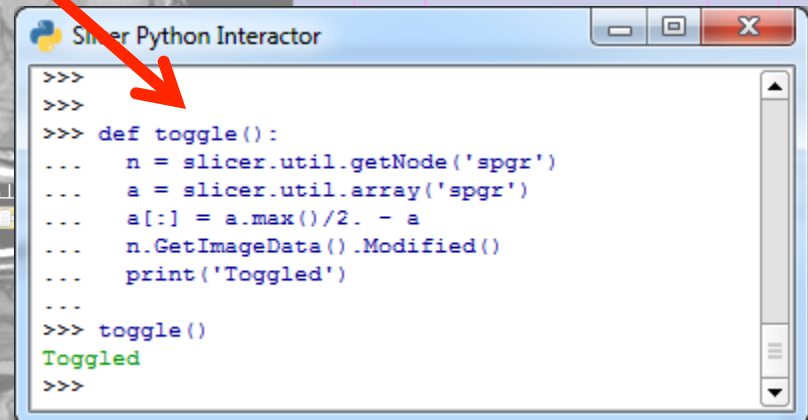
The main interface shows three MRI slices: a top axial slice, a bottom sagittal slice, and a bottom coronal slice. A pink wireframe box is overlaid on the top slice, with axes labeled R (Right), P (Posterior), L (Left), and I (Inferior). Below the slices are sliders for Y, R (3.050mm), G, and A (14.238mm). A 5 cm scale bar is visible under each slice. The bottom-left corner shows 'B: spgr' and 'L F B' orientation markers.

Manipulating Arrays

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

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

```
toggle()
```



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

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

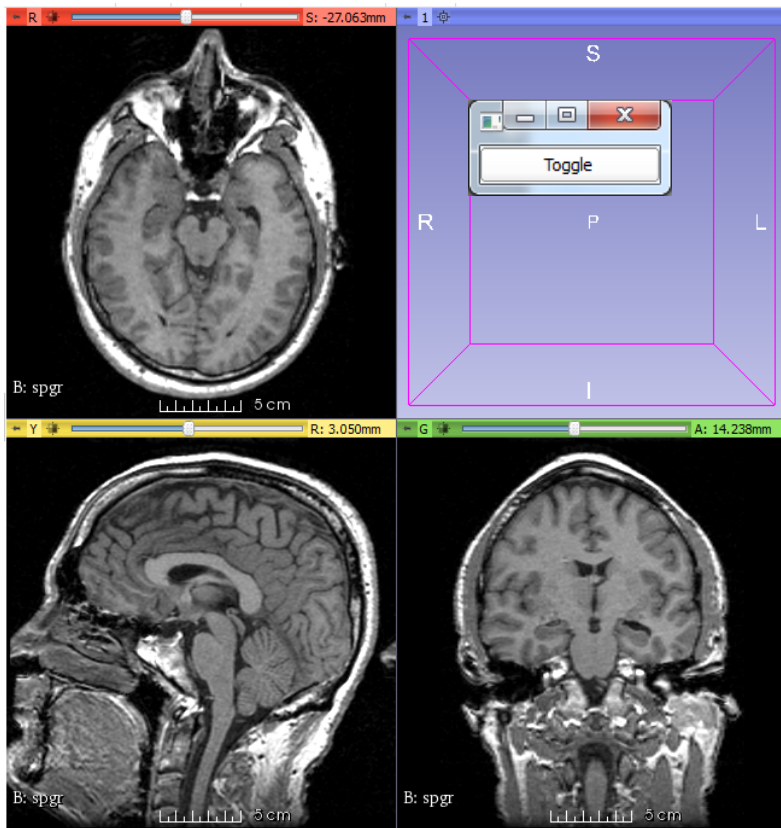
Run the following code in the Python console

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

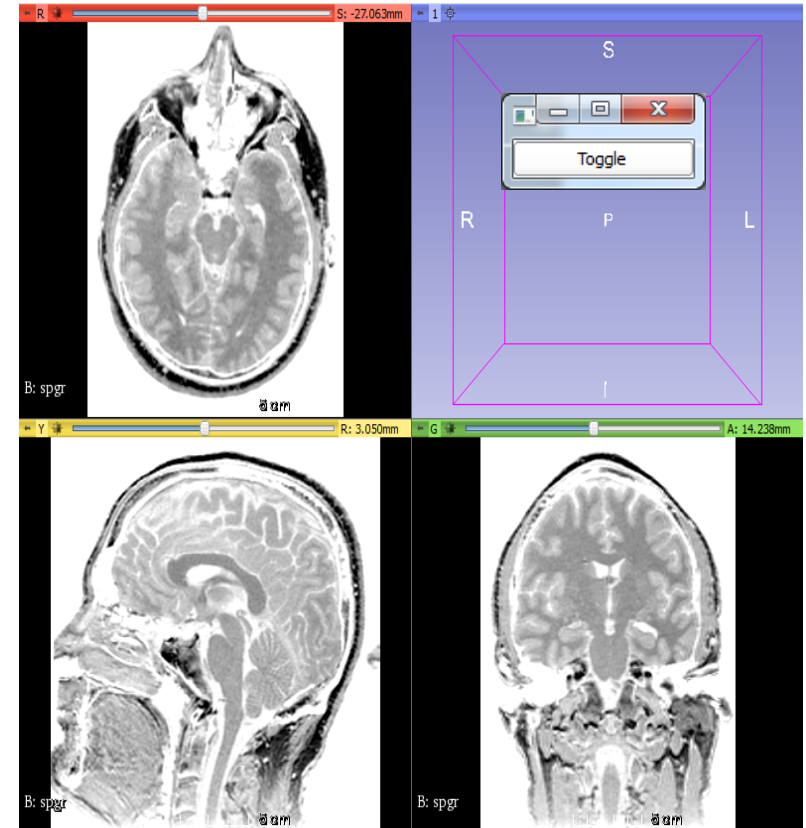
```
Slicer Python Interactor
-- n.GetImageData().Modified()
-- print('Toggled')
--
>>> toggle()
Toggled
>>> b = qt.QPushButton('Toggle')
>>> b.connect('clicked()', toggle)
True
>>> b.show()
```

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)

```
File Edit Format Run Options Windows Help
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 (MNI)"] # 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 3P41R013218-12J1 (NAC) and is part of the National Alliance
for Medical Sleep 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)

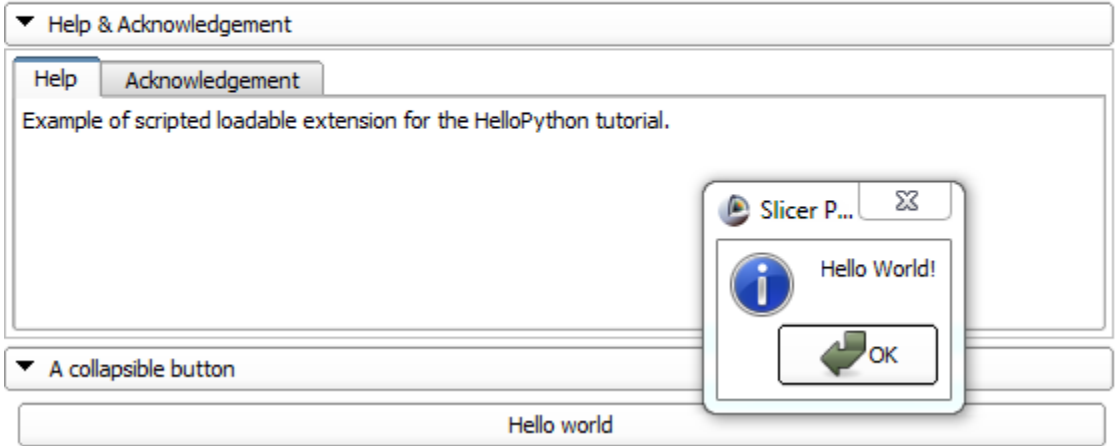
    # Hello World button
    # (Insert Section A text here)
    # (Be sure to match indentation of the rest of this
    # code)

    # 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

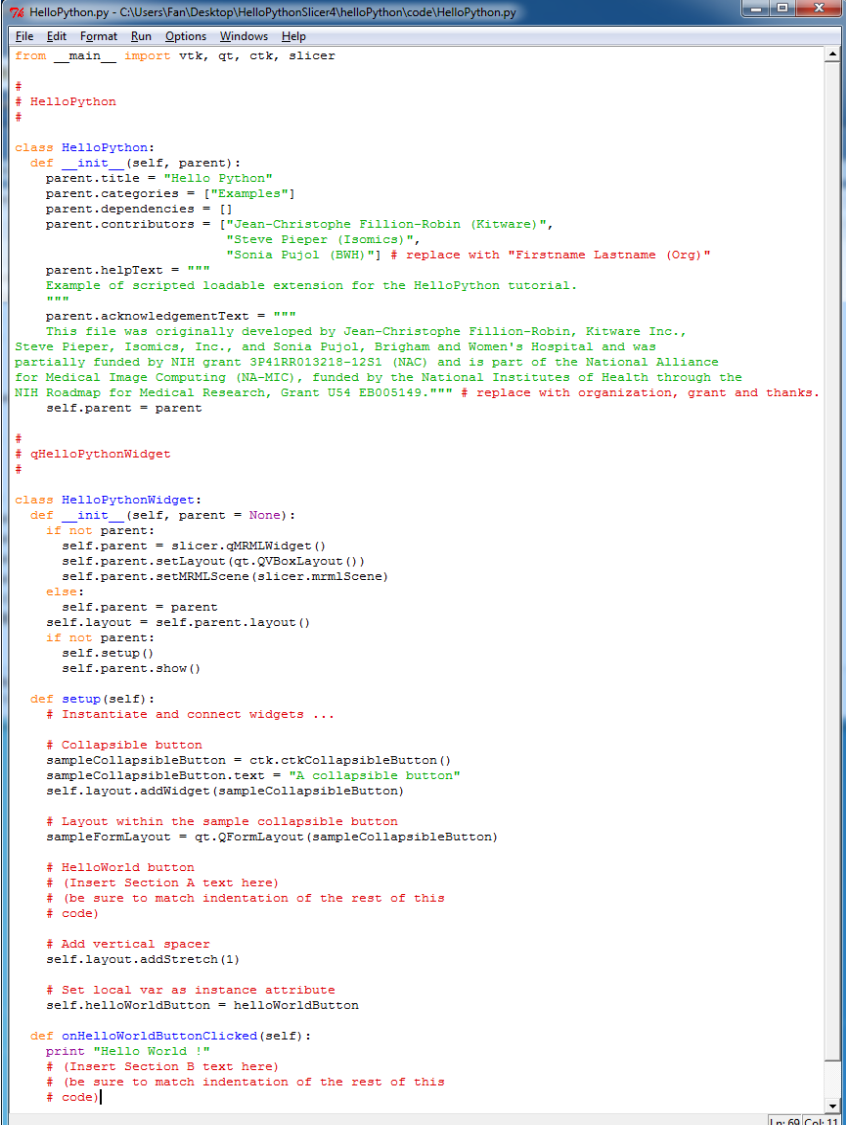
HelloPython.py

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

Module
Description

Module GUI

Processing
Code



```
74 HelloPython.py - C:\Users\Fan\Desktop\HelloPythonSlicer4\helloPython\code\HelloPython.py
File Edit Format Run Options Windows Help
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)|

Ln: 69 Col: 11
```

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)
```

```
# 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
```

Add this
Text in
section A

Processing Code

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

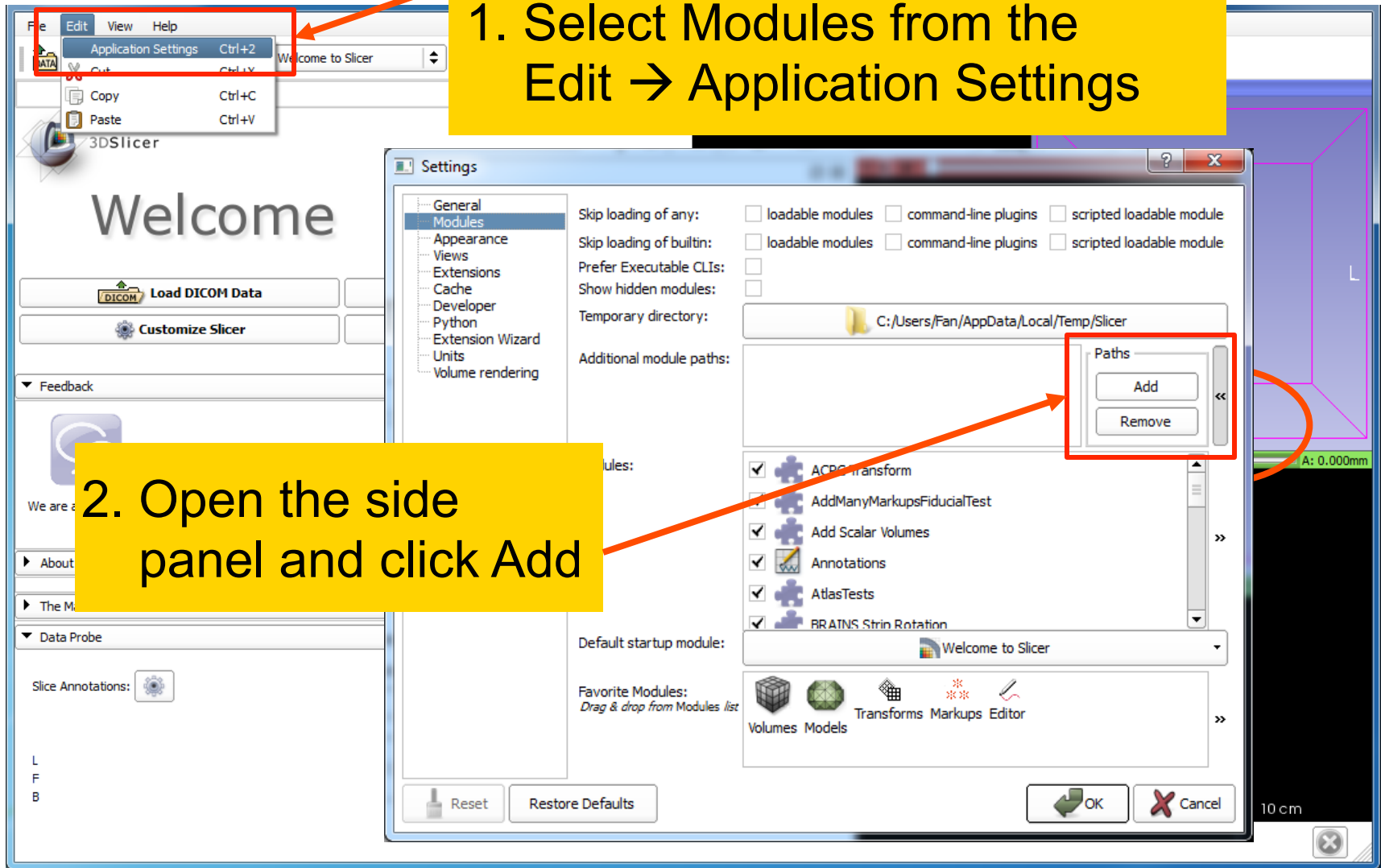
```
qt.QMessageBox.information(  
    slicer.util.mainWindow(),  
    'Slicer Python', 'Hello World!')
```

Add this
Text in
section B

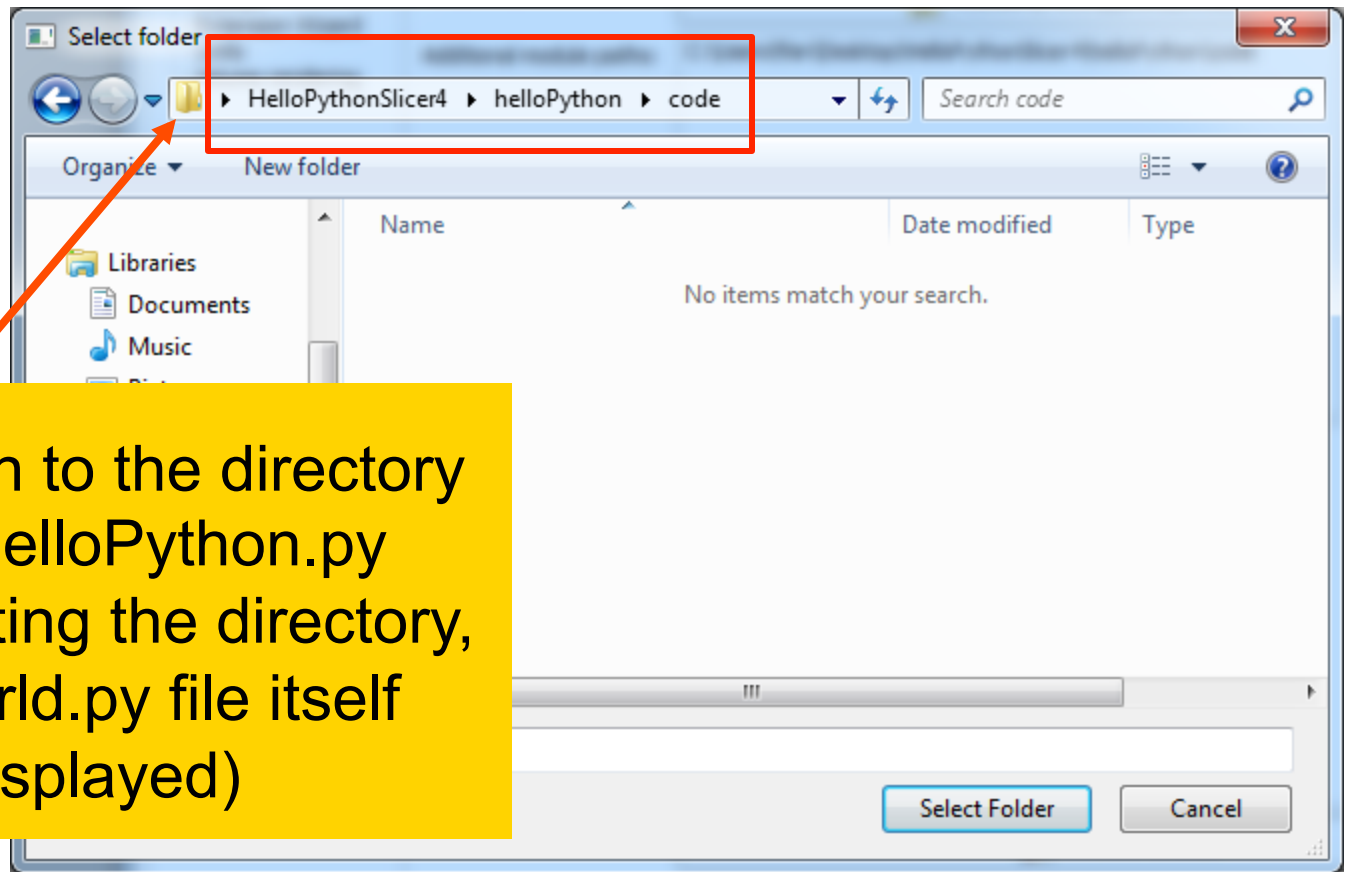
Integrating HelloPython

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)

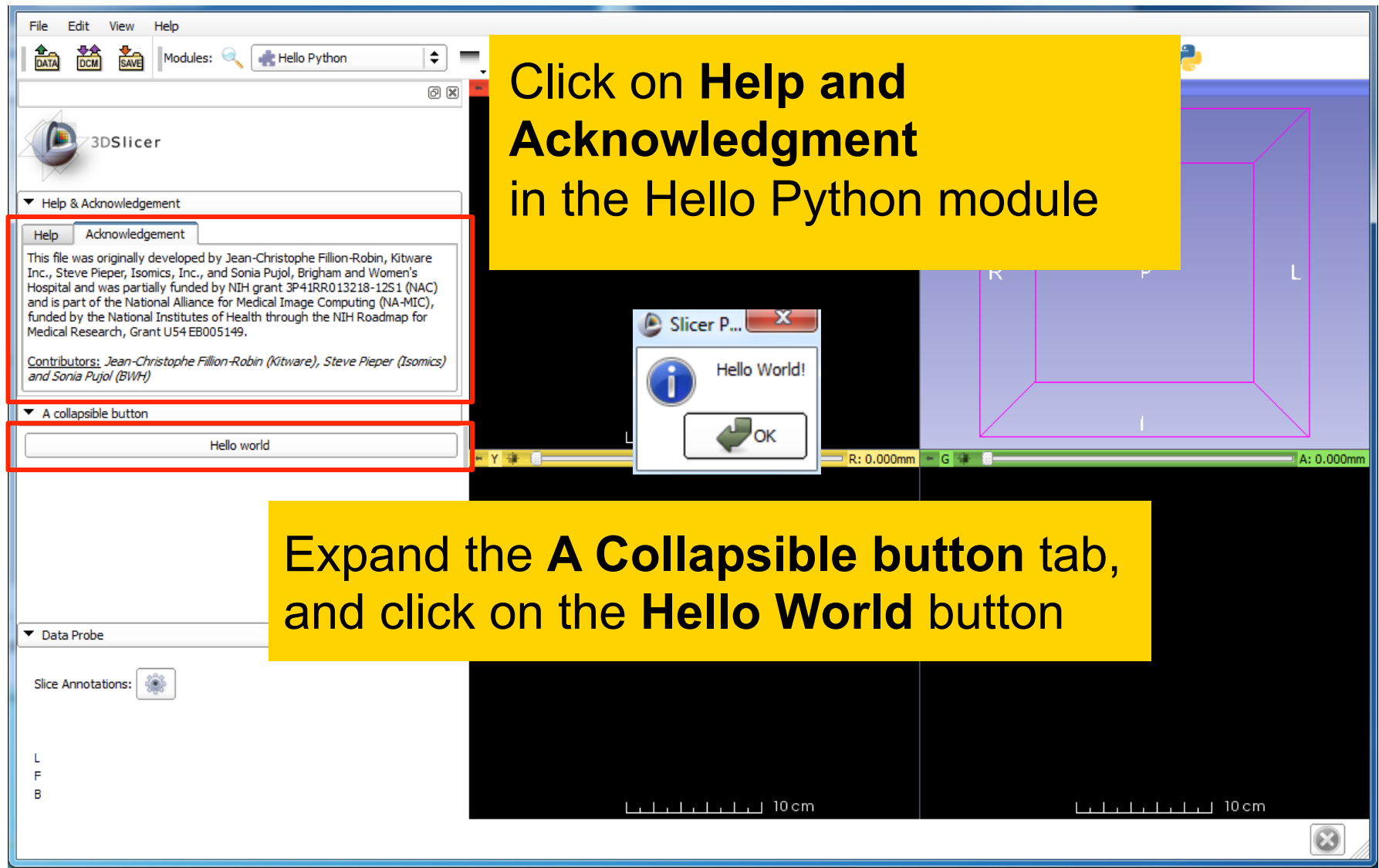


Integrating HelloPython

4. Restart Slicer when prompted. Hello Python is now in the Modules Menu, under the category **Examples**

The screenshot shows the 3DSlicer interface with the 'Modules' menu open. The 'Examples' category is highlighted with a red box, and the 'Hello Python' module is visible within it. A yellow callout box contains the text '4. Restart Slicer when prompted. Hello Python is now in the Modules Menu, under the category Examples'. An orange arrow points from the callout box to the 'Hello Python' module in the menu.

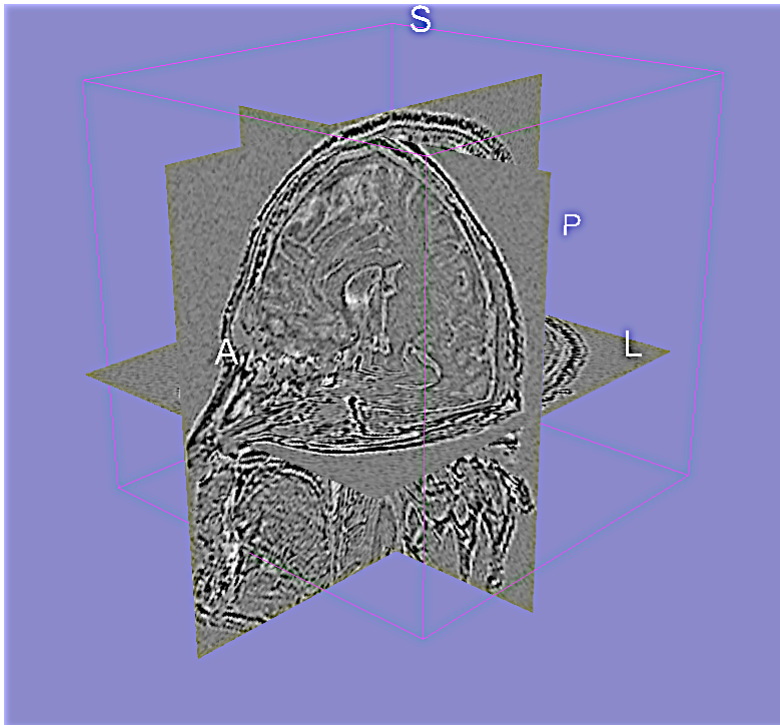
HelloPython in Slicer



The screenshot shows the 3DSlicer application window. The 'Hello Python' module is active. A yellow box highlights the 'Help & Acknowledgement' section, which contains text about the software's development and contributors. A red box highlights the 'A collapsible button' section, which contains a button labeled 'Hello world'. A dialog box titled 'Slicer P...' with the text 'Hello World!' and an 'OK' button is open in the center. Another yellow box highlights the 'Data Probe' section, which contains a 'Slice Annotations' button. The main 3D view shows a purple wireframe box on a black background.

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

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



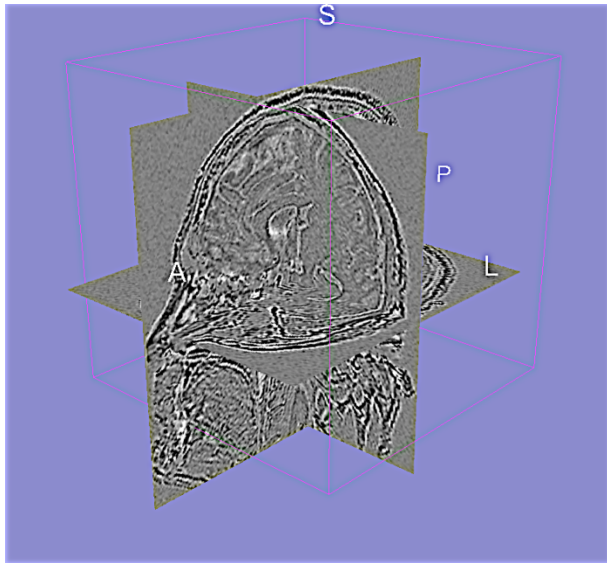
Part C:

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

```
74 HelloLaplace.py - CAUsers\Fan\Desktop\HelloPythonSlicer4\helloPython\code\HelloLaplace.py
File Edit Format Run Options Windows Help
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)

Ln: 55 Col: 53
```

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://commonstk.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
```

```
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)
```

Add this
code

In More Detail

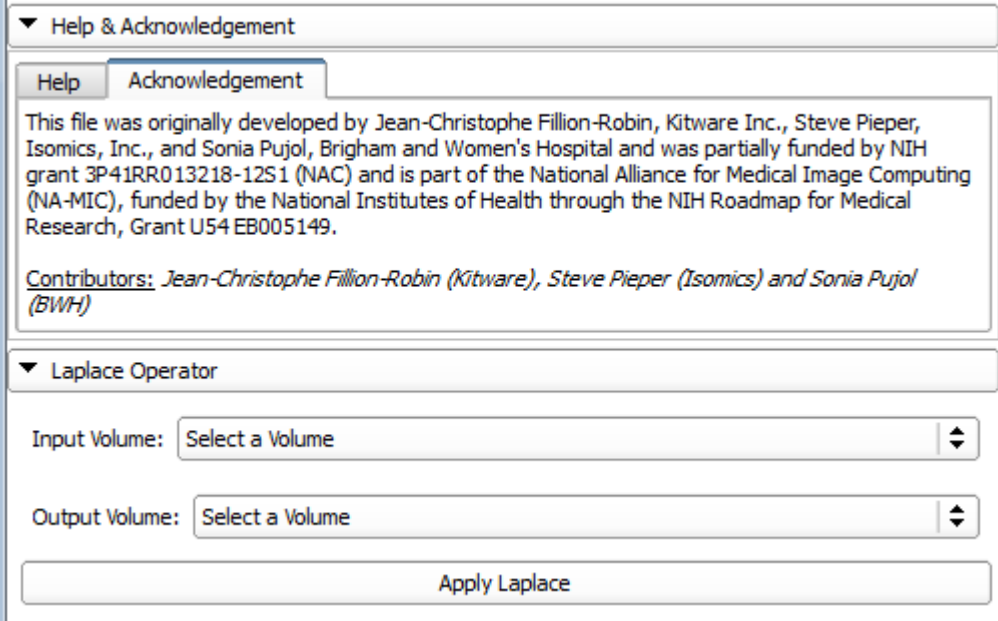
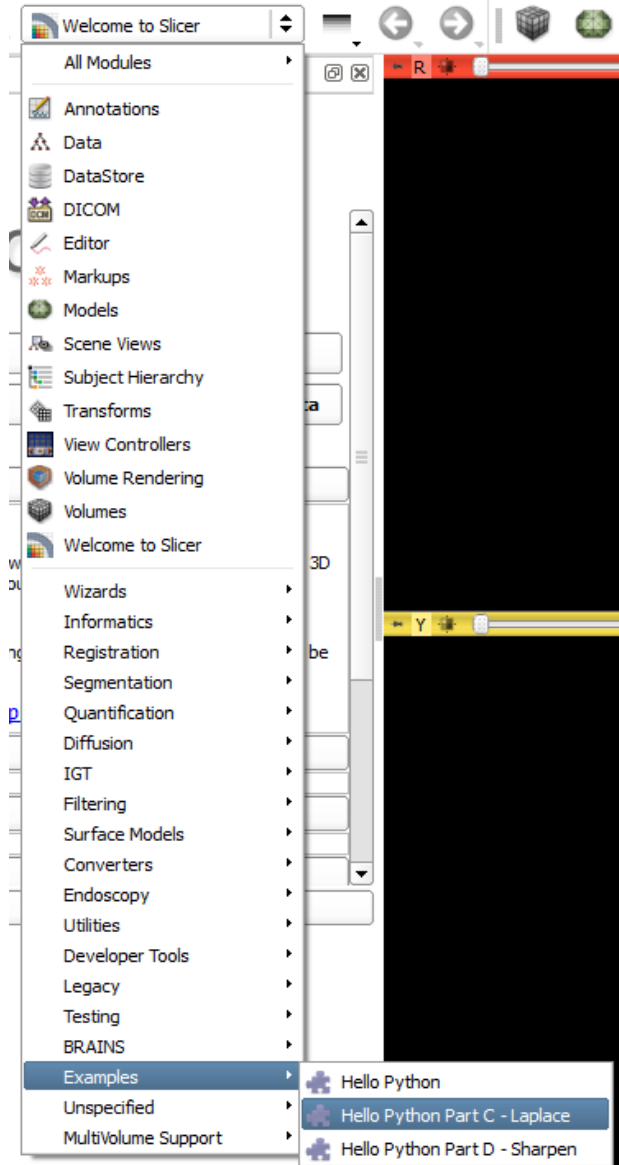
- **vtkImageLaplacian** is a `vtkImageAlgorithm` operates on `vtkImageData` (see <http://vtk.org>)
- **vtkMRMLScalarVolumeNode** is a Slicer MRML class that contains `vtkImageData`, plus orientation information `ijkToRAS` matrix (see http://www.slicer.org/slicerWiki/index.php/Coordinate_systems)

In More Detail (Continued)

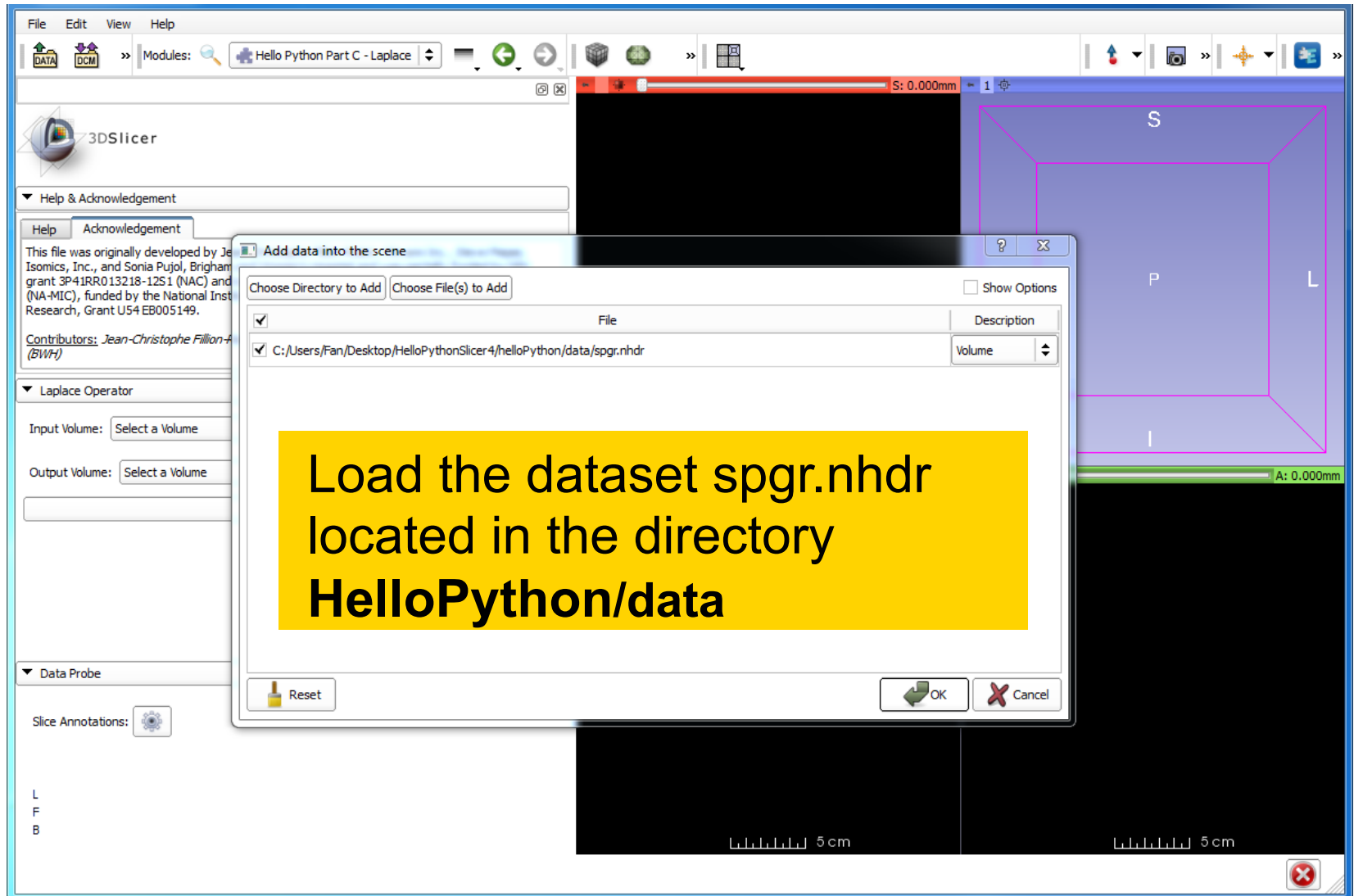
- Global **licer** package gives python access to:
 - GUI (via **licer.app**)
 - modules (via **licer.modules**)
 - data (via **licer.mrmlScene**)
- **licer.app.applicationLogic()** provides helper utilities for manipulating Slicer state

Go To Laplace Module

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



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

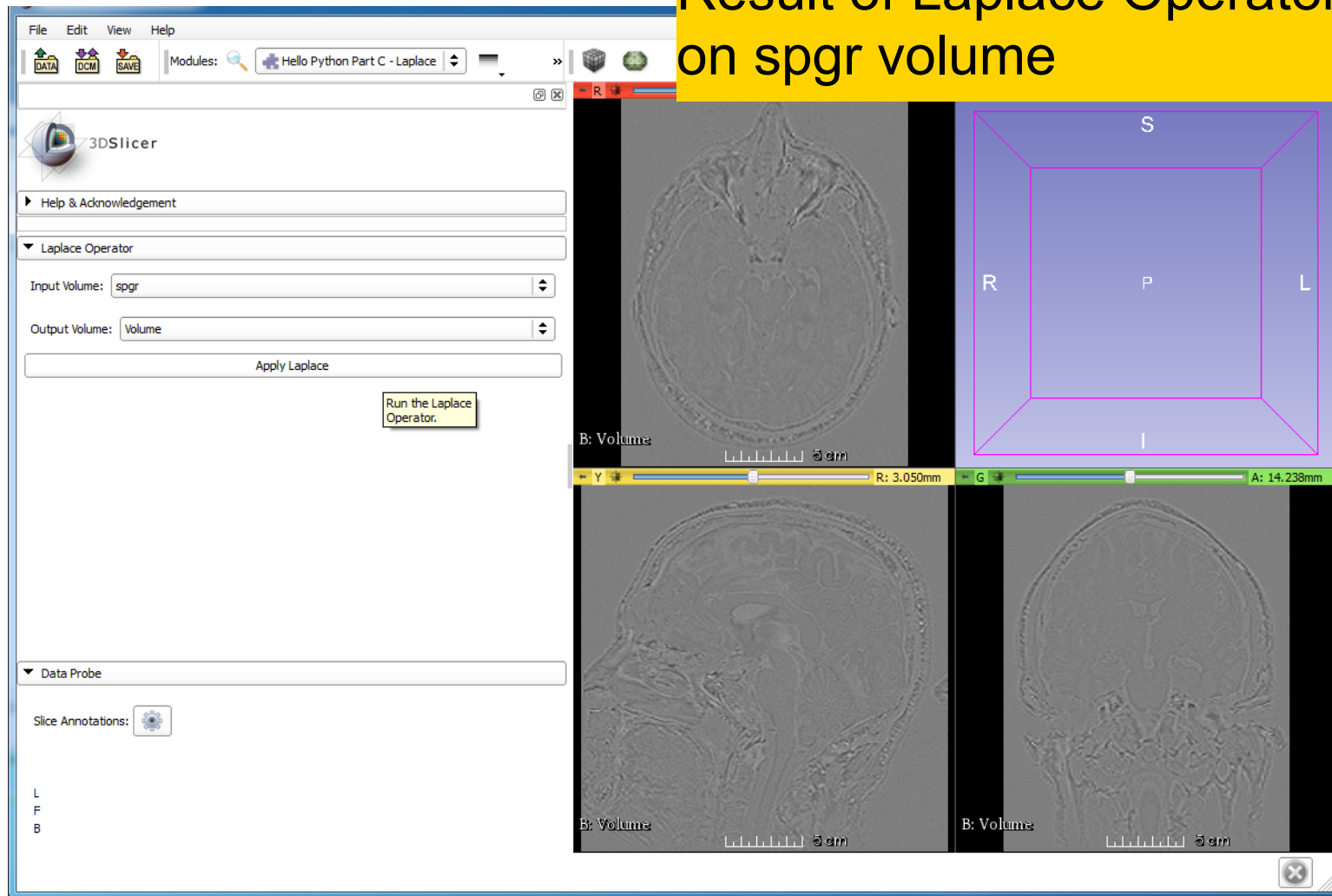
Apply Laplace

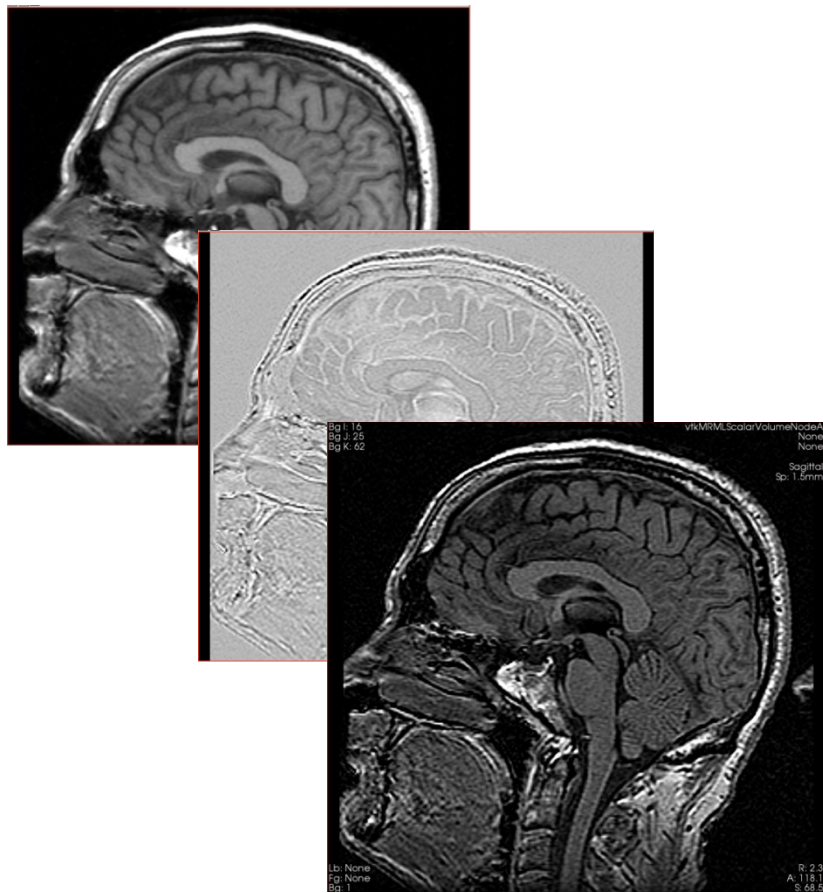
Run the Laplace Operator.

3. Run the module

Laplace Module

Result of Laplace Operator on spgr volume



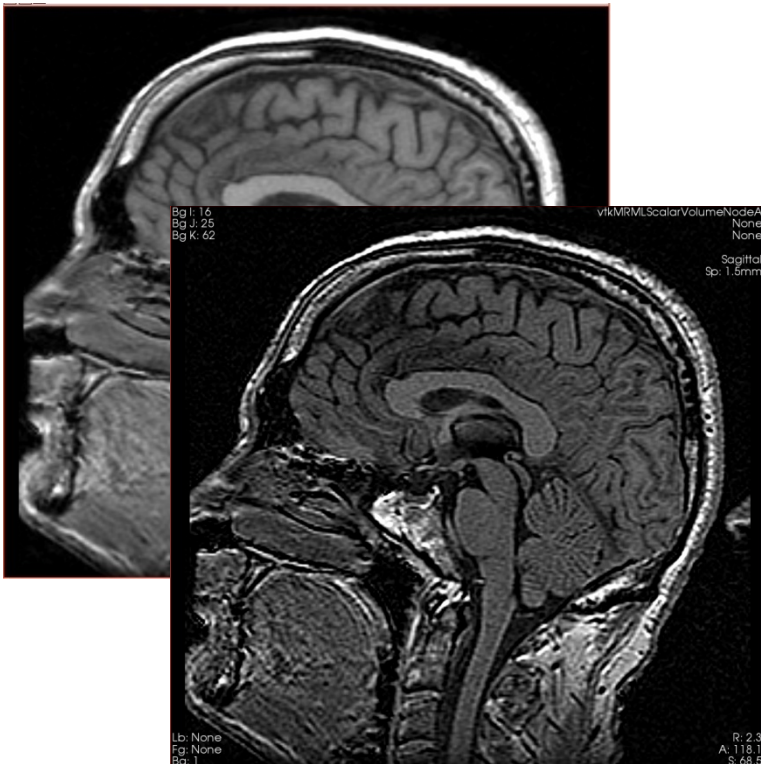


Part D:

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

```
76 HelloSharpen.py - C:\Users\Fan\Desktop\HelloPythonSlicer4\helloPython\code\HelloSharpen.py
File Edit Format Run Options Windows Help
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):

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
    #

S. Pujol, Ph.D. - S. Pieper, Ph.D. Ln: 34 Col: 0
```

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

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

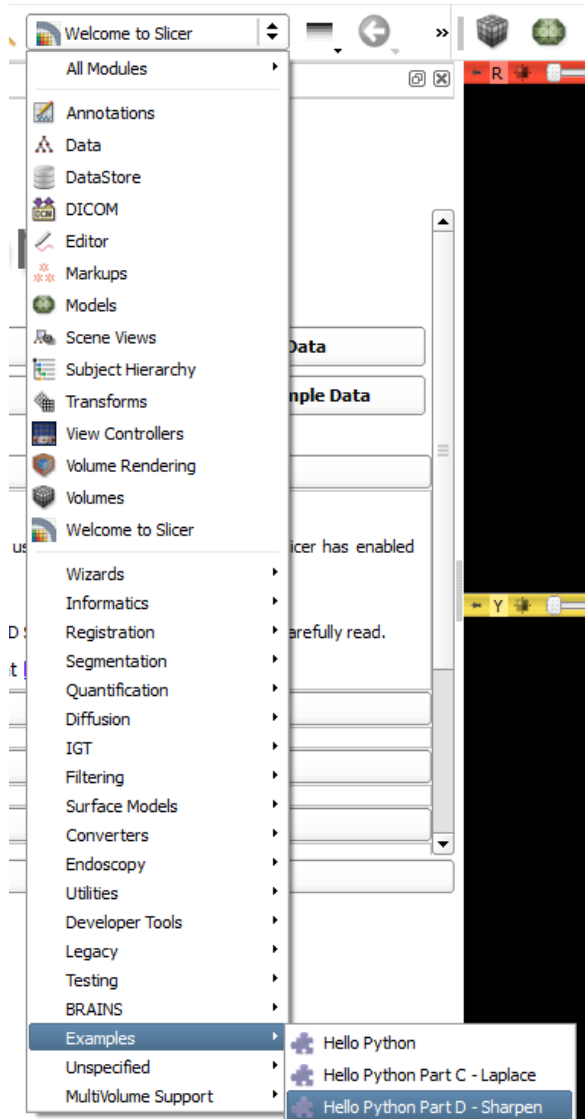
```
...
outputVolume.SetAndObserveImageData(laplacian.GetOutput())
# optionally subtract laplacian from original image
if self.sharpen.checked:
    parameters = {}
    parameters['inputVolume1'] = inputVolume.GetID()
    parameters['inputVolume2'] = outputVolume.GetID()
    parameters['outputVolume'] = outputVolume.GetID()
    slicer.cli.run( slicer.modules.subtractsclavolumes, 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)
```

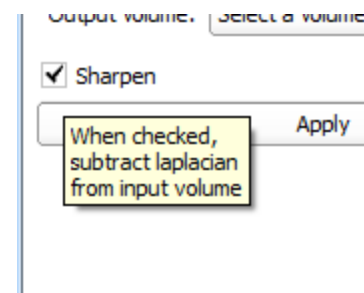
In More Detail

- **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

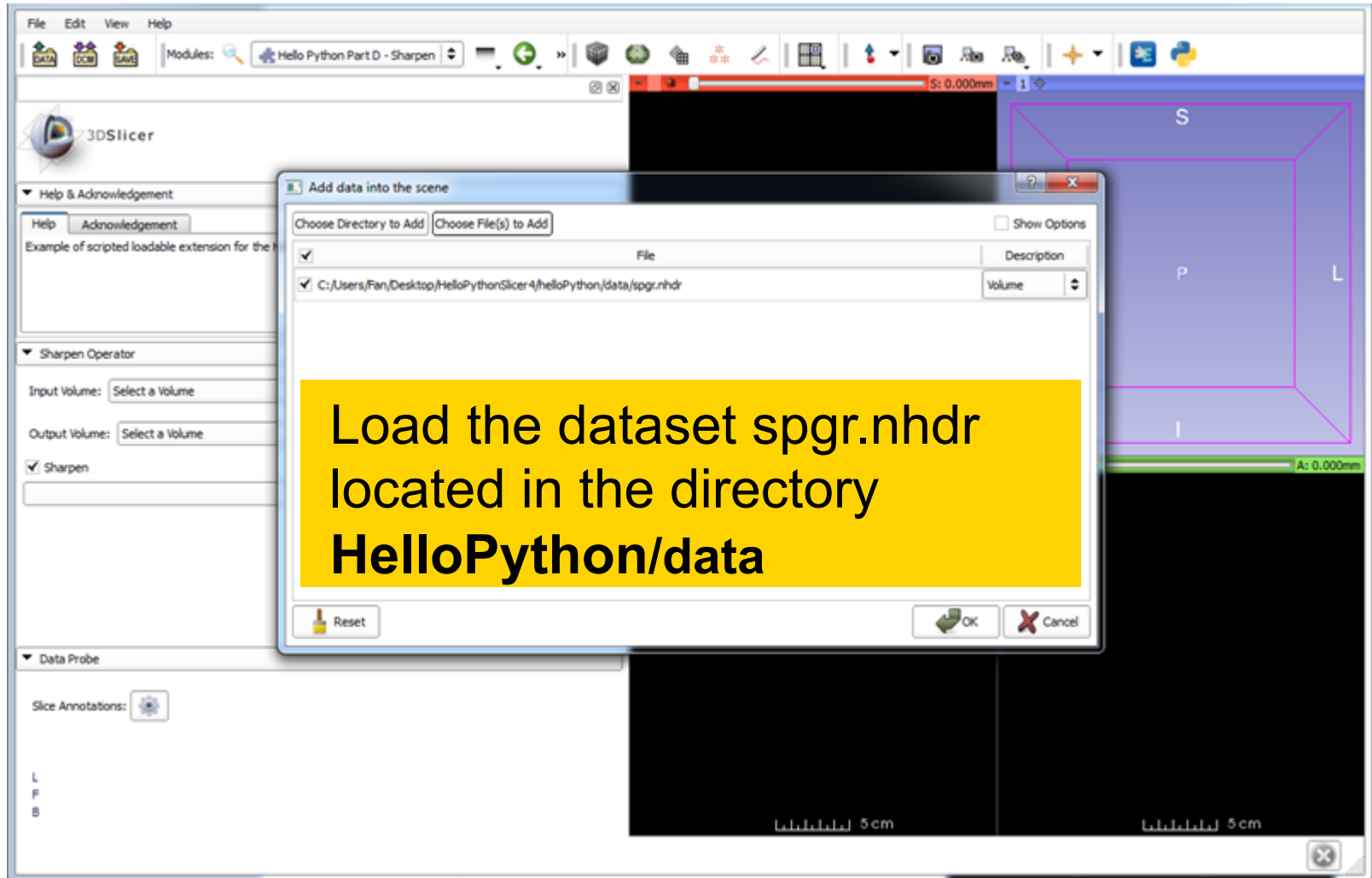
Go To Sharpen Module



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

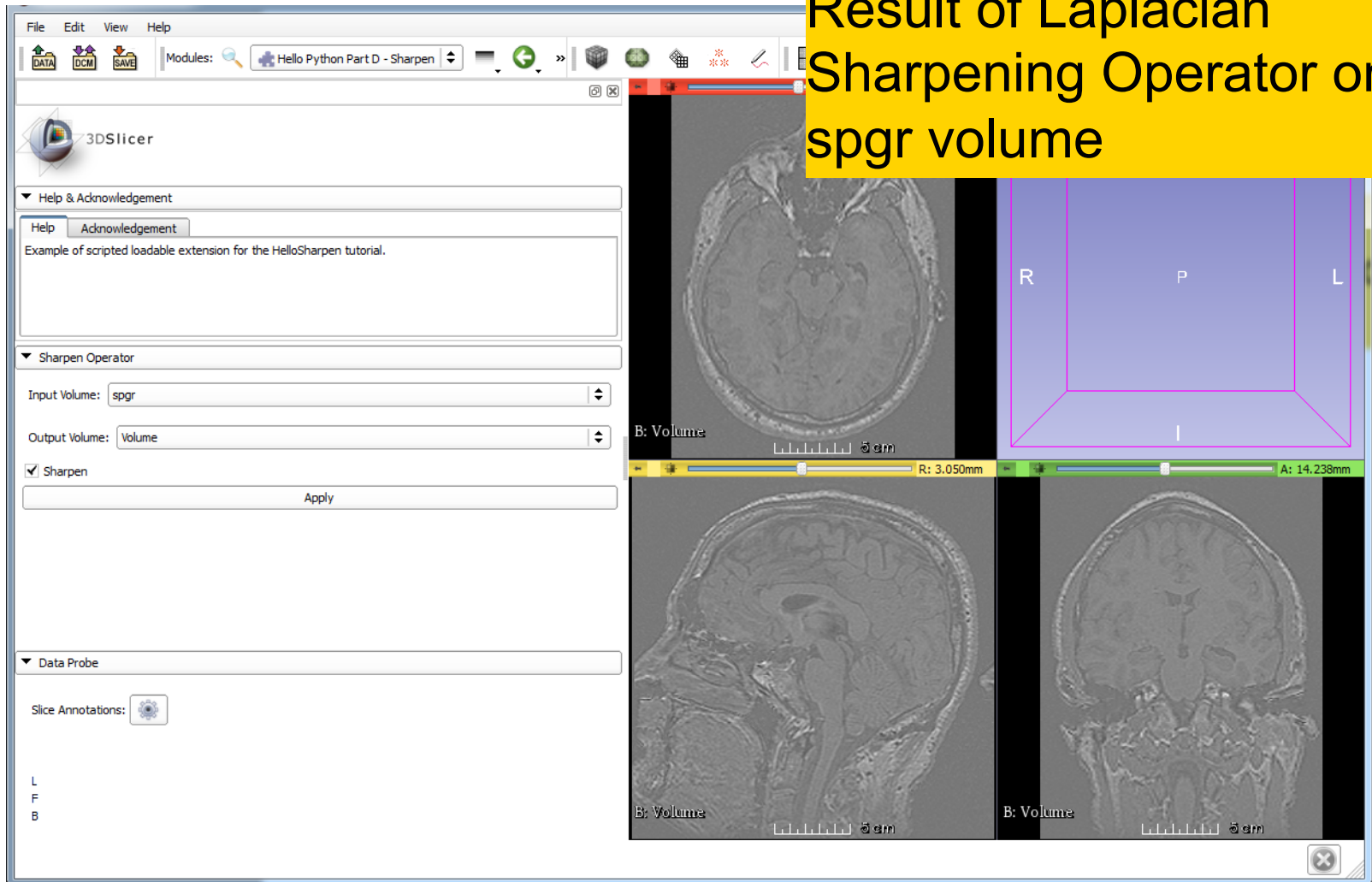
Apply

3. Run the module

Run the Laplace or Sharpen Operator.

Sharpen Module

Result of Laplacian
Sharpening Operator on
spgr volume



Sharpen Module

Adjust Window/Level with
Left-Mouse-Drag in Slice
Window

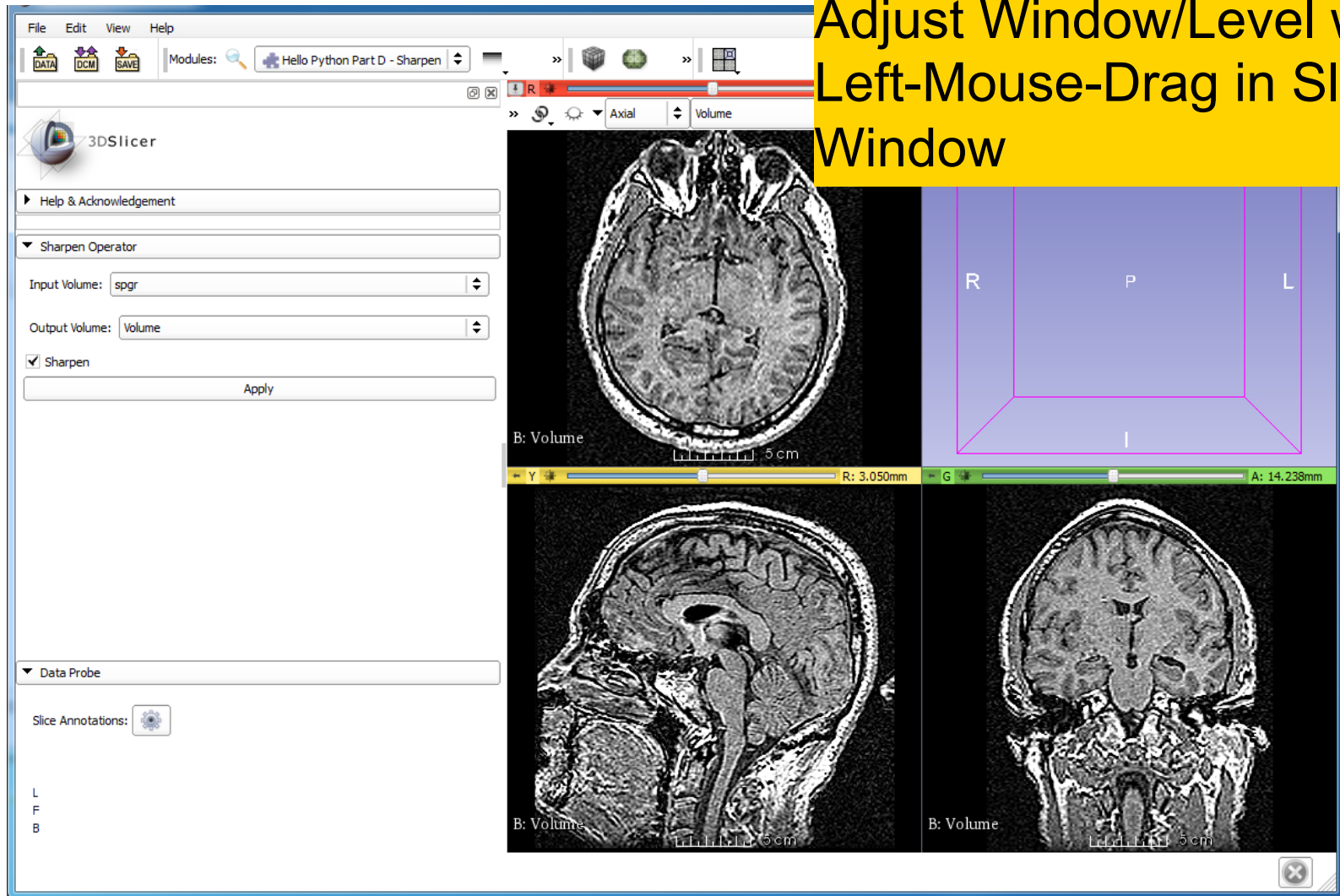
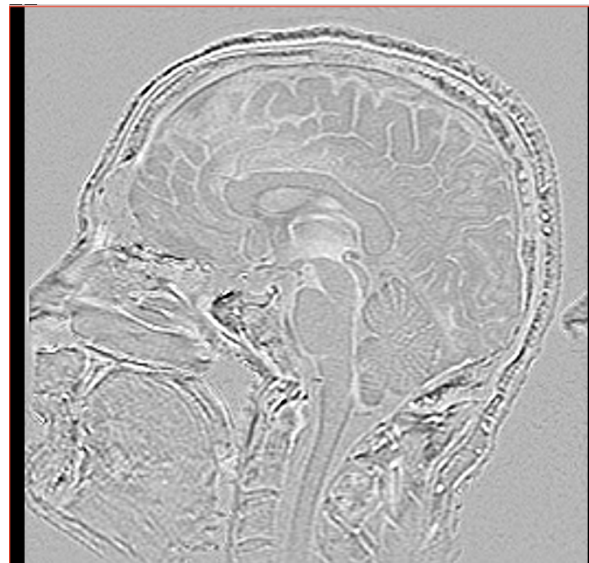
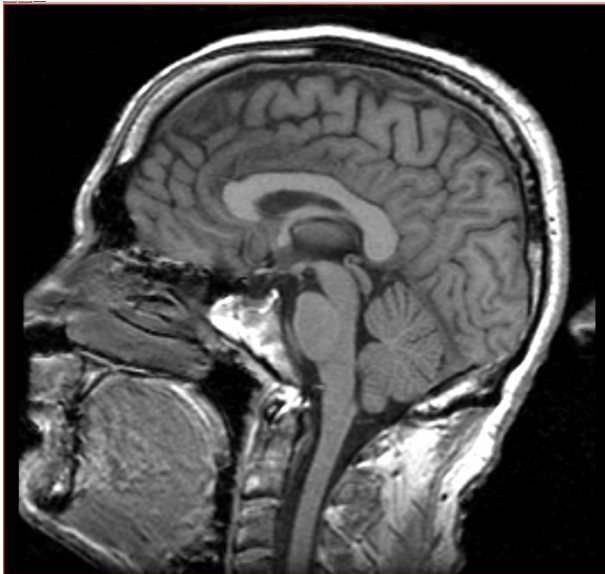


Image Sharpening

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



Sidong Liu and Fan Zhang, The University of Sydney