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

Programming in Slicer4

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

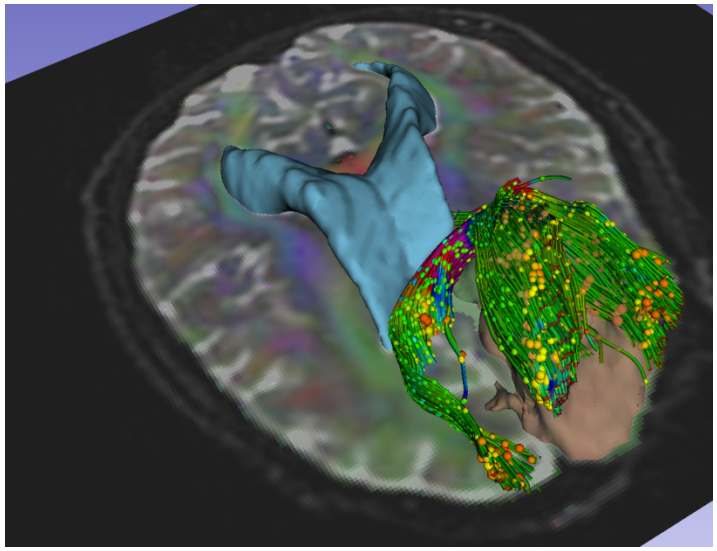
Steve Pieper, Ph.D.
Isomics Inc.

The NA-MIC Kit





3D Slicer version 4 (Slicer4.7.0-2017-07-12)



- 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 heaving 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 (Qt)
- 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**

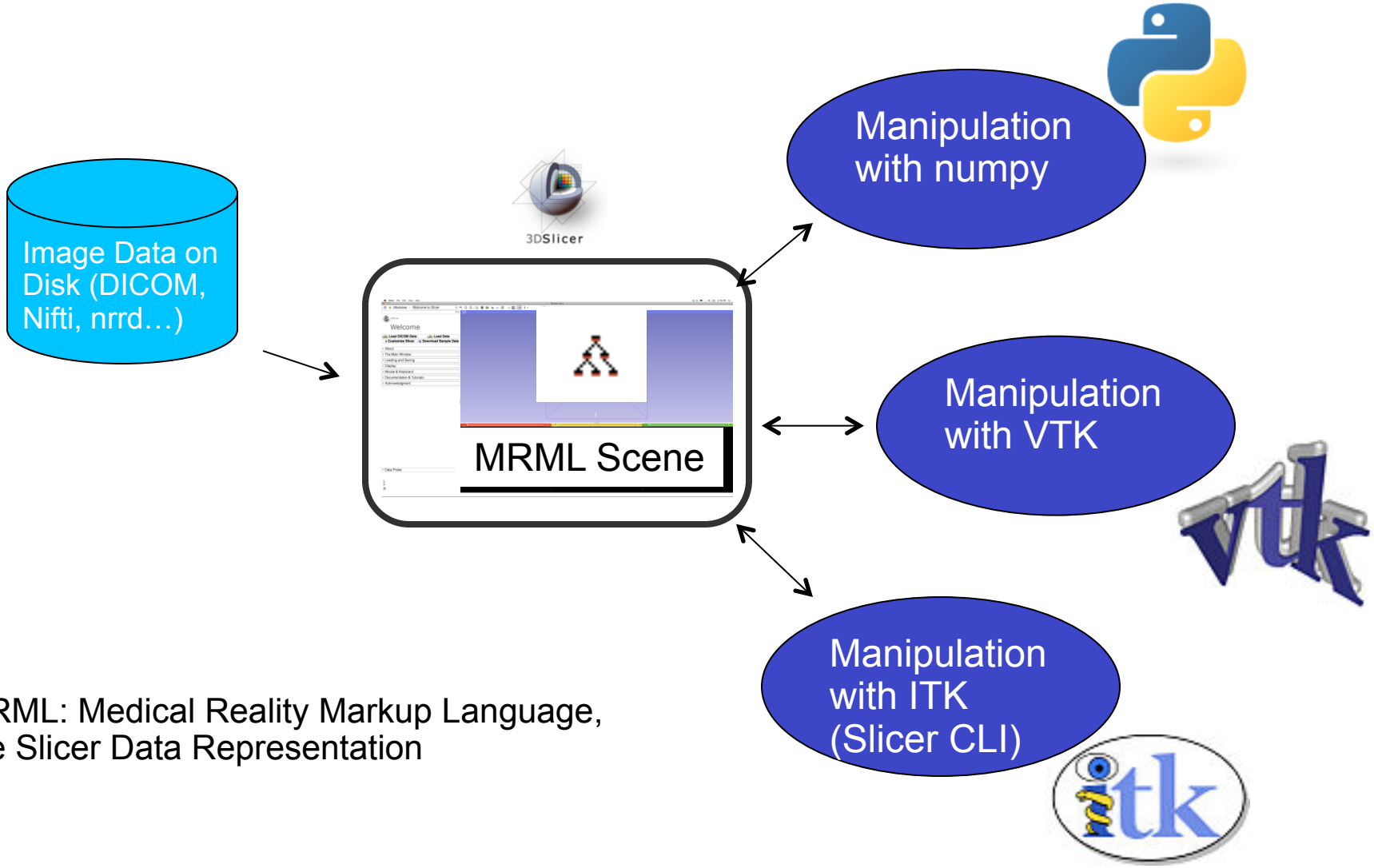
- GUI based on Qt libraries accessed via Python

Tutorial Goal

- This tutorial guides you through the steps of creating a very simple “Hello World” Python scripted module and two small modules for performing 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



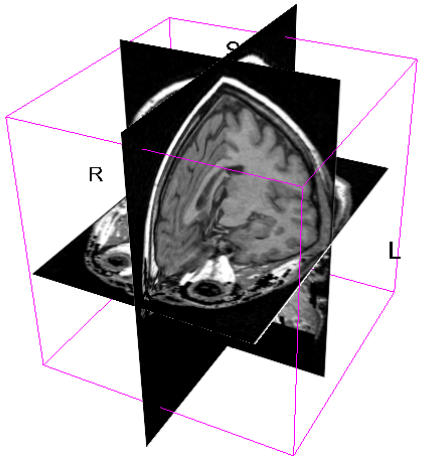
MRML: Medical Reality Markup Language, the Slicer Data Representation

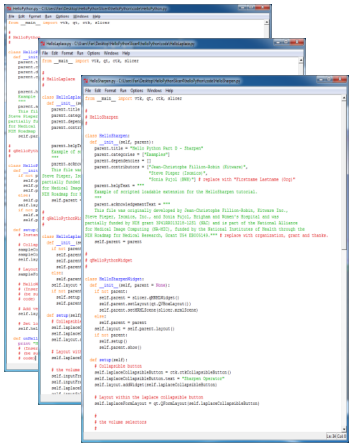
Prerequisites

- 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 in the Slicer training compendium
- Programming experience is required, and some familiarity with Python is essential.

Course Material

www.slicer.org → Slicer training → **Slicer4 Programming Tutorial**





Sample data set: spgr.nrrd (124-slice MRI volume)

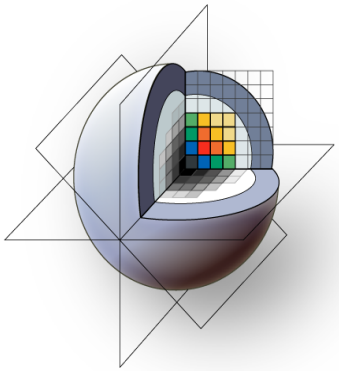
Note: other volumes (for example, provided by SampleData module) can be used just as well.

Completed programming examples: HelloPython, HelloLaplace, HelloSharpen

Unzip the **HelloPython_Nightly.zip** archive

Course Overview

- Part A: Exploring Slicer via Python
- Part B: “Hello world” simple Python scripted module (HelloPython)
- Part C: Implementation of Laplace operator for a 3D volume (HelloLaplace)
- Part D: Image sharpening using Laplace operator and subtraction (HelloSharpen)



Part A: EXPLORING SLICER VIA PYTHON

S.Pujol, Ph.D. - S.Pieper, Ph.D.
NA-MIC ARR 2012-2017

Python in Slicer

Slicer includes python 2.7 and a rich set of standard libraries

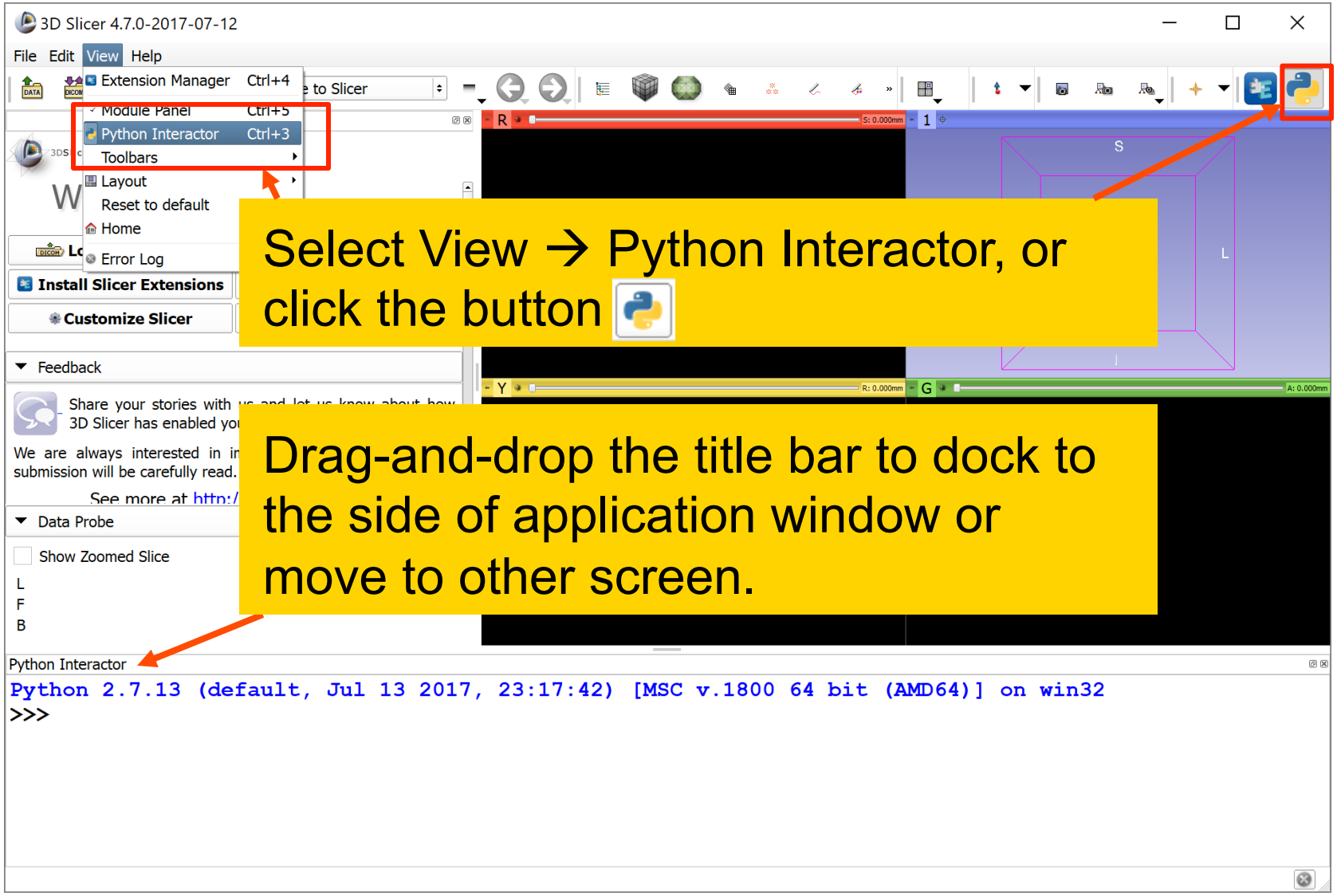
- *Included*:
  - **numpy, VTK, CTK, PythonQt**,

  and most of standard python library

- *Not included*:
  - scipy (scientific tools for python),
  - matplotlib (python 2D ploting library),
  - ipython (interactive python)

  and some other popular packages that we have found difficult to package for distribution

Python Console in Slicer

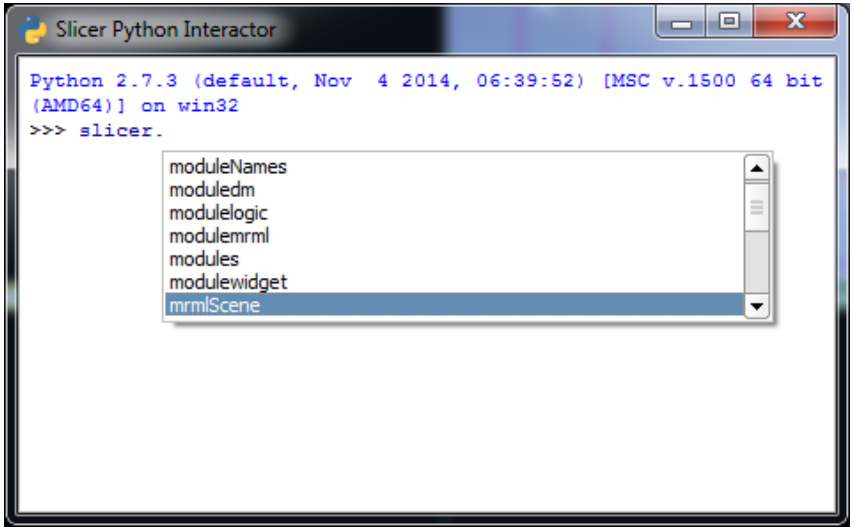


Select View → Python Interactor, or click the button

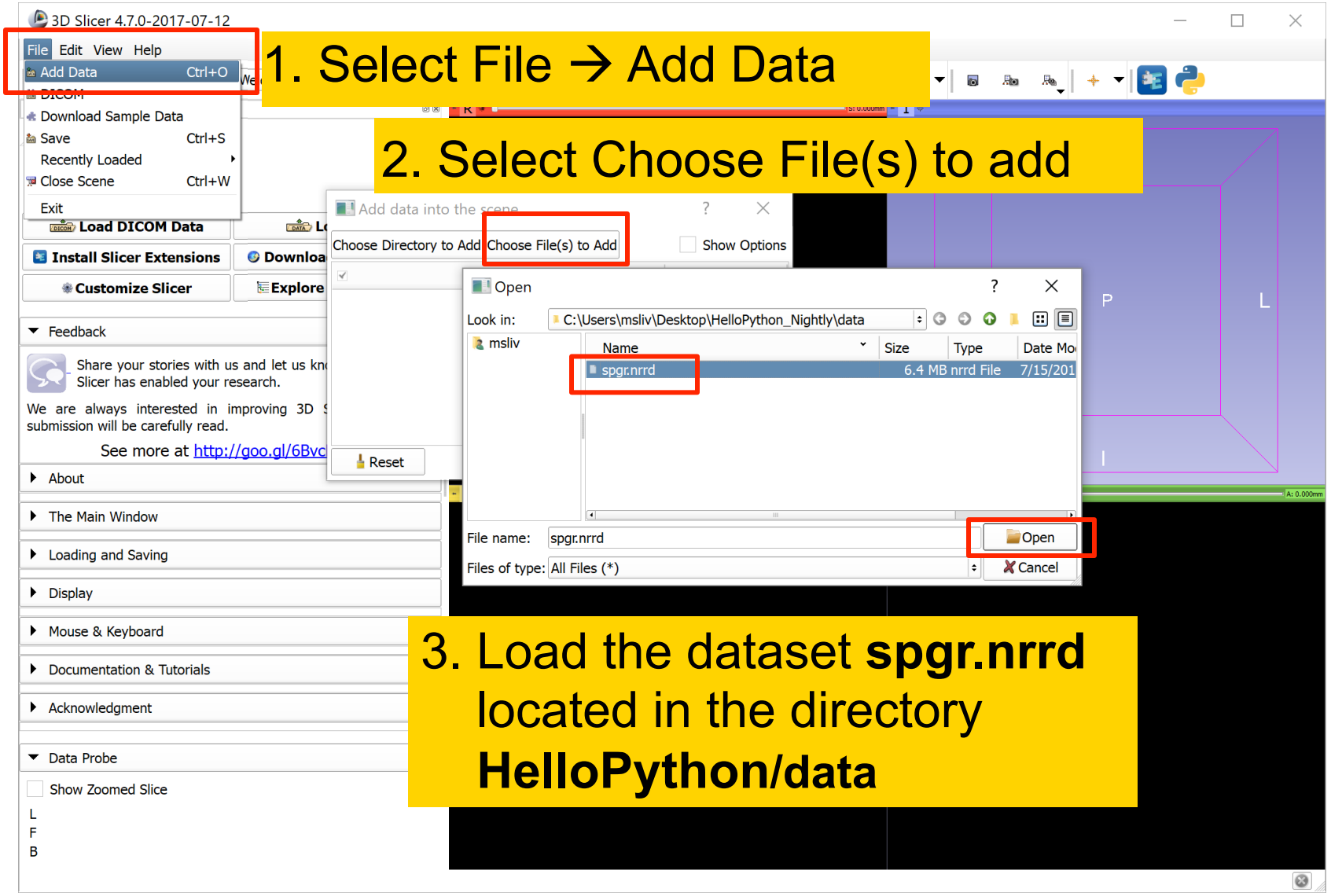
Drag-and-drop the title bar to dock to the side of application window or move to other screen.

General Python Console Features

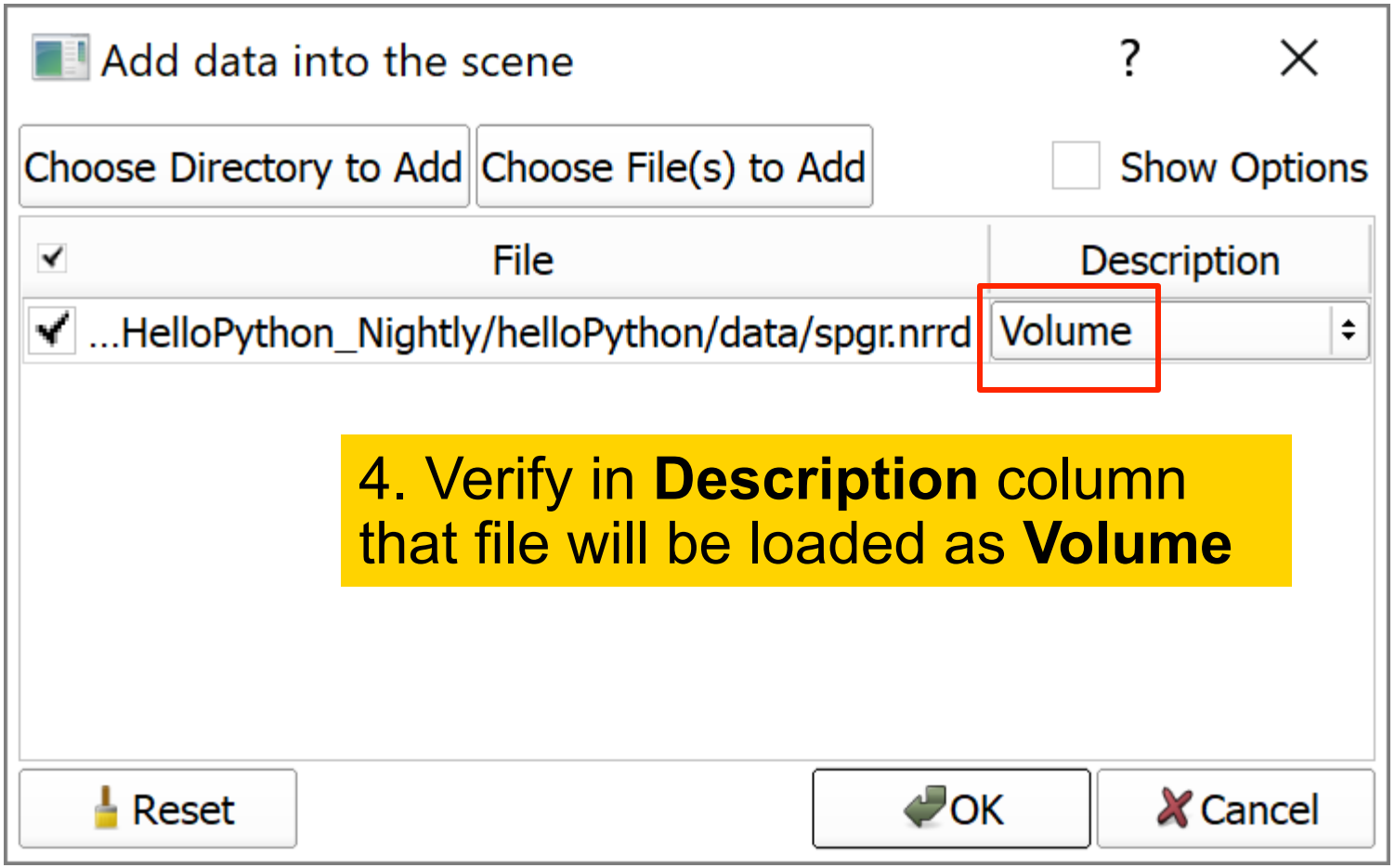
- Command Line Editing:
  - Left/Right Arrow Keys, Home, End
  - Delete (Control-D)
  - Copy/Paste
- Command Completion:
  - Tab Key
- Input History:
  - Up/Down Arrow Keys



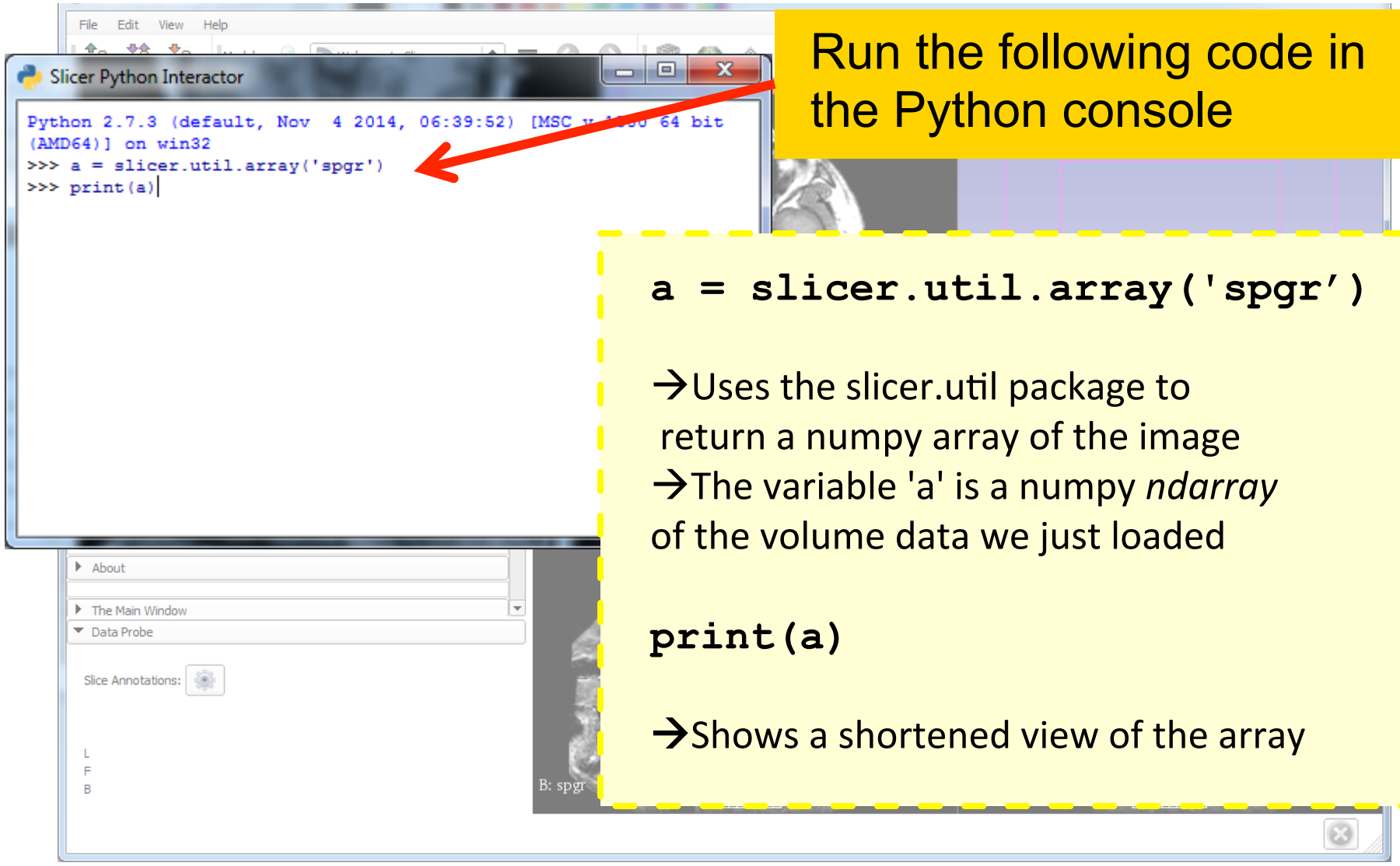
Add Volume


3D Slicer 4.7.0-2017-07-12
File Edit View Help
Add Data Ctrl+O
DICOM
Download Sample Data
Save Ctrl+S
Recently Loaded
Close Scene Ctrl+W
Exit
1. Select File → Add Data
2. Select Choose File(s) to add
Add data into the scene
Choose Directory to Add
Choose File(s) to Add
Show Options
Reset
Open
Look in: C:\Users\msliv\Desktop\HelloPython_Nightly\data
msliv
Name Size Type Date Mo
spgr.nrrd 6.4 MB nrrd File 7/15/201
File name: spgr.nrrd
Open
Files of type: All Files (*)
Cancel
Load DICOM Data
Install Slicer Extensions
Customize Slicer
Feedback
Share your stories with us and let us kn
Slicer has enabled your research.
We are always interested in improving 3D
submission will be carefully read.
See more at http://goo.gl/6Bvc
About
The Main Window
Loading and Saving
Display
Mouse & Keyboard
Documentation & Tutorials
Acknowledgment
Data Probe
Show Zoomed Slice
L
F
B
3. Load the dataset spgr.nrrd located in the directory HelloPython/data

Add Volume


Add data into the scene
?
Choose Directory to Add
Choose File(s) to Add
Show Options
File
Description
...HelloPython_Nightly/helloPython/data/spgr.nrrd
Volume
4. Verify in **Description** column that file will be loaded as **Volume**
Reset
OK
Cancel

Access to MRML and Arrays



Run the following code in the Python console

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

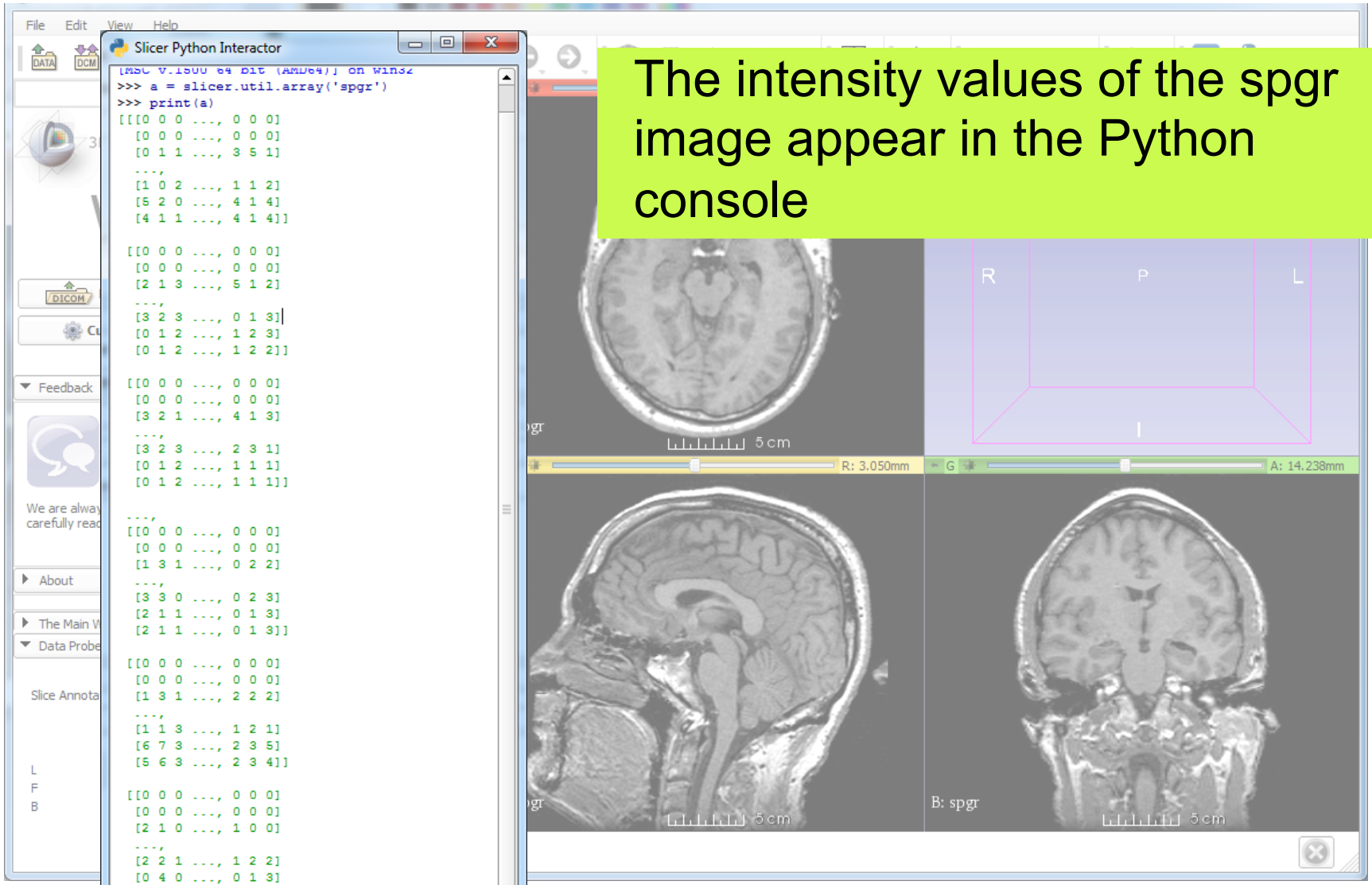
→Uses the slicer.util package to return a numpy array of the image

→The variable 'a' is a numpy *ndarray* of the volume data we just loaded

```
print(a)
```

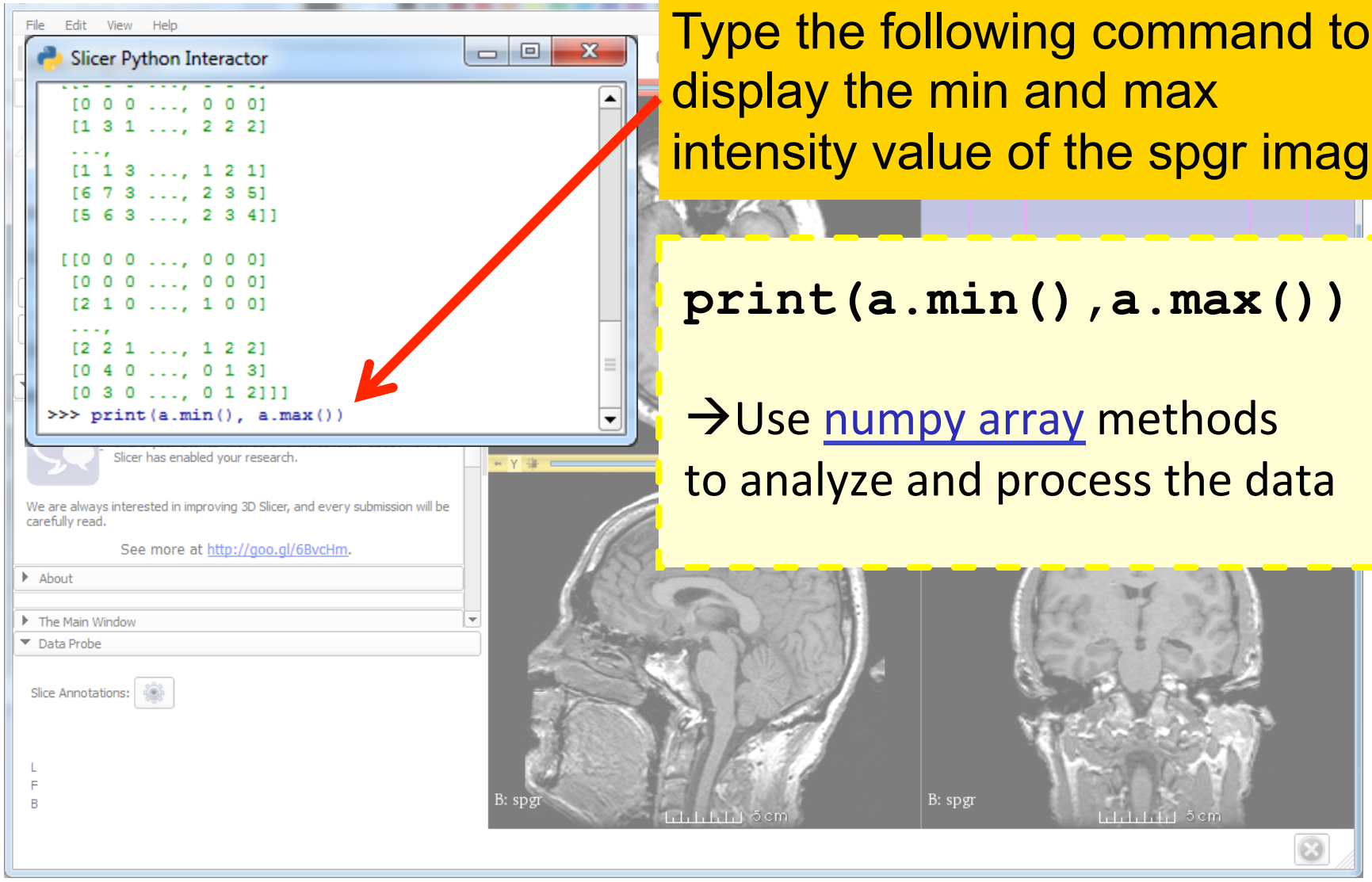
→Shows a shortened view of the array

Access to MRML and Arrays



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

Access to MRML and Arrays

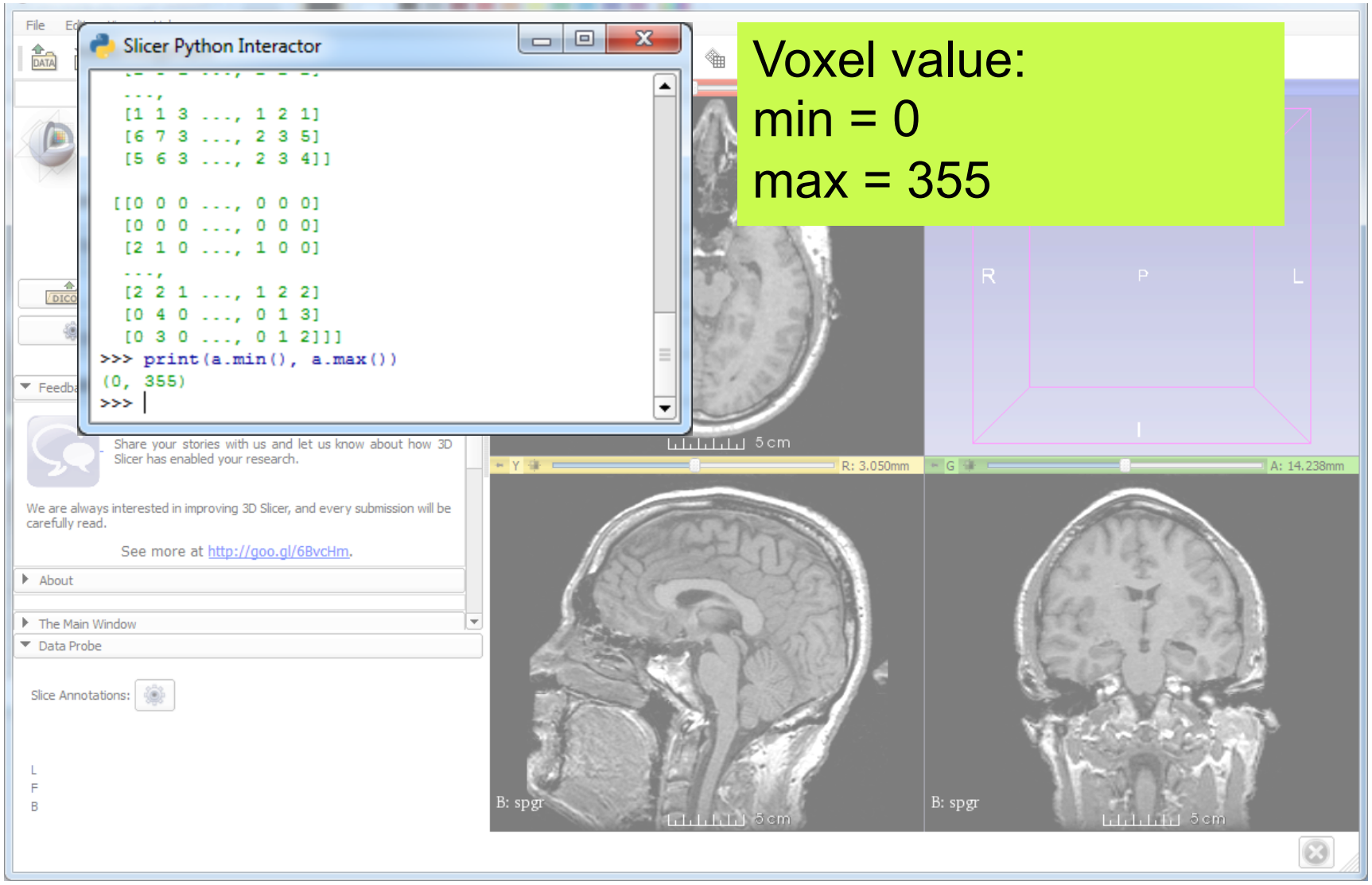


Type the following command to display the min and max intensity value of the spgr image

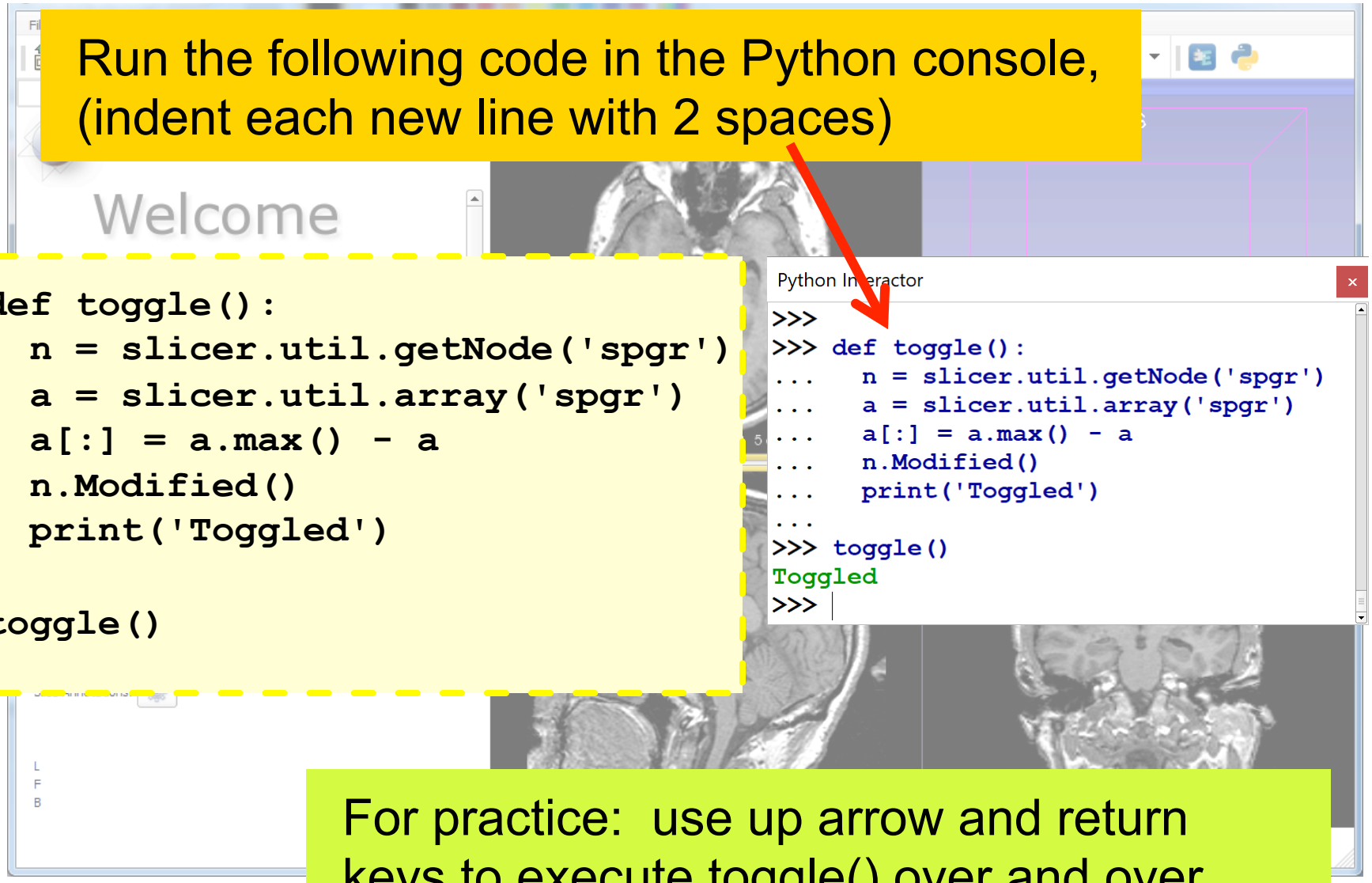
```
print(a.min(),a.max())
```

→Use numpy array methods to analyze and process the data

Access to MRML and Arrays



Manipulating Arrays

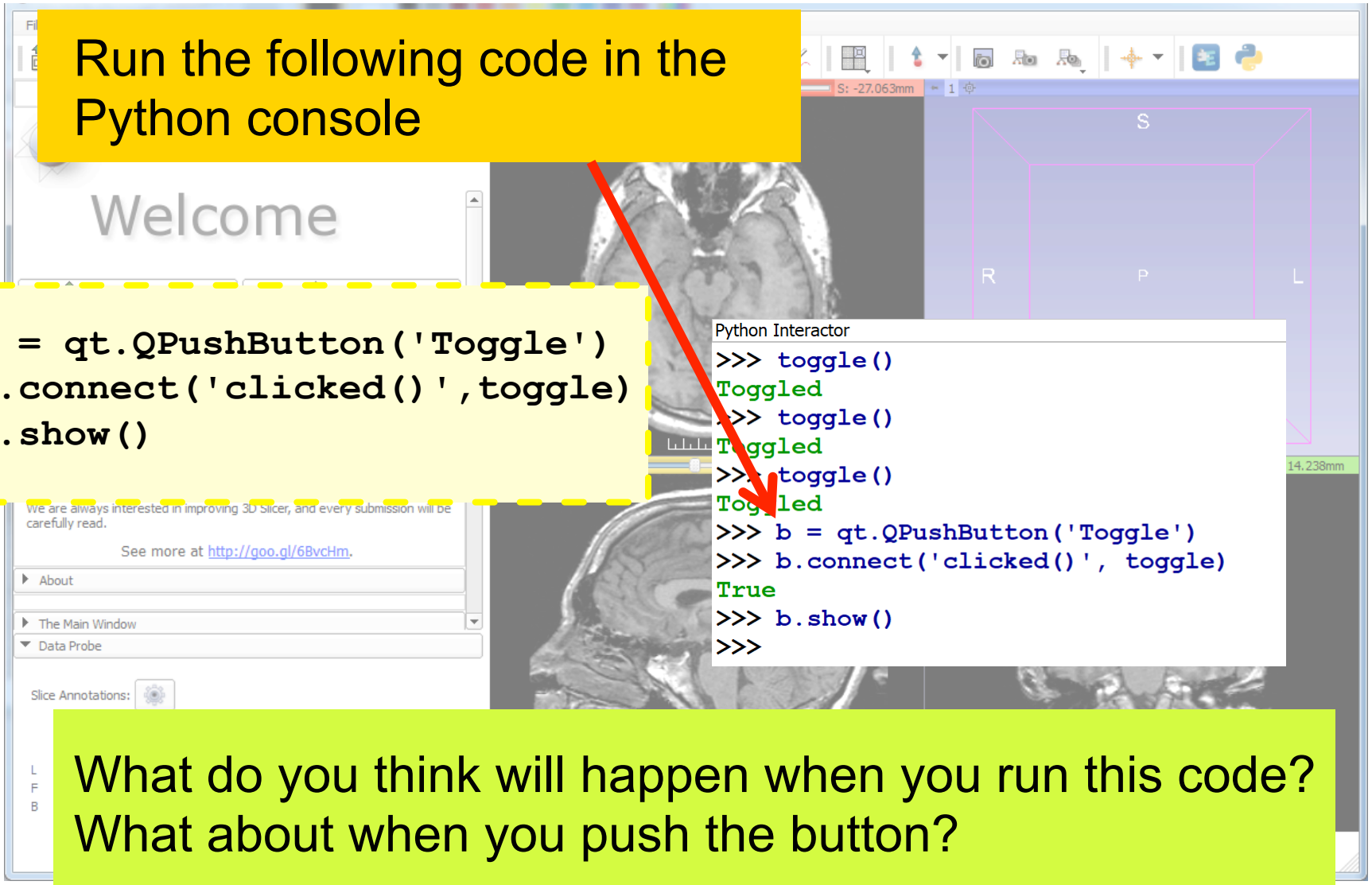


The toggle function in more detail

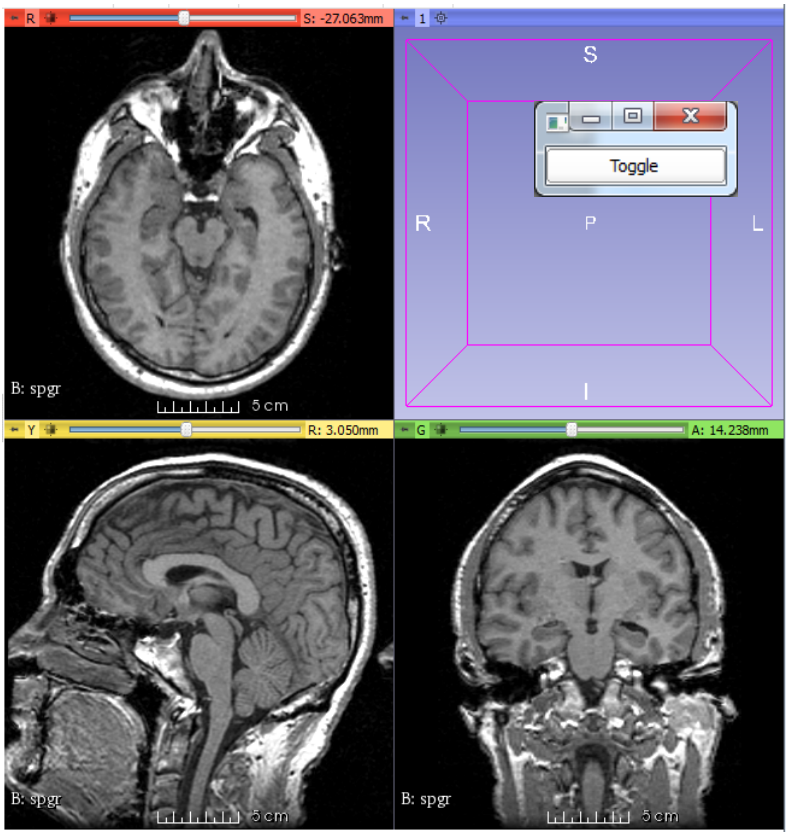
- **def toggle():**
  - Defines a python function
  - Body of function performs element-wise math on entire volume
  - Easy mix of scalar and volume math

- Telling Slicer that the image data for node 'n' has been modified causes the slice view windows to refresh

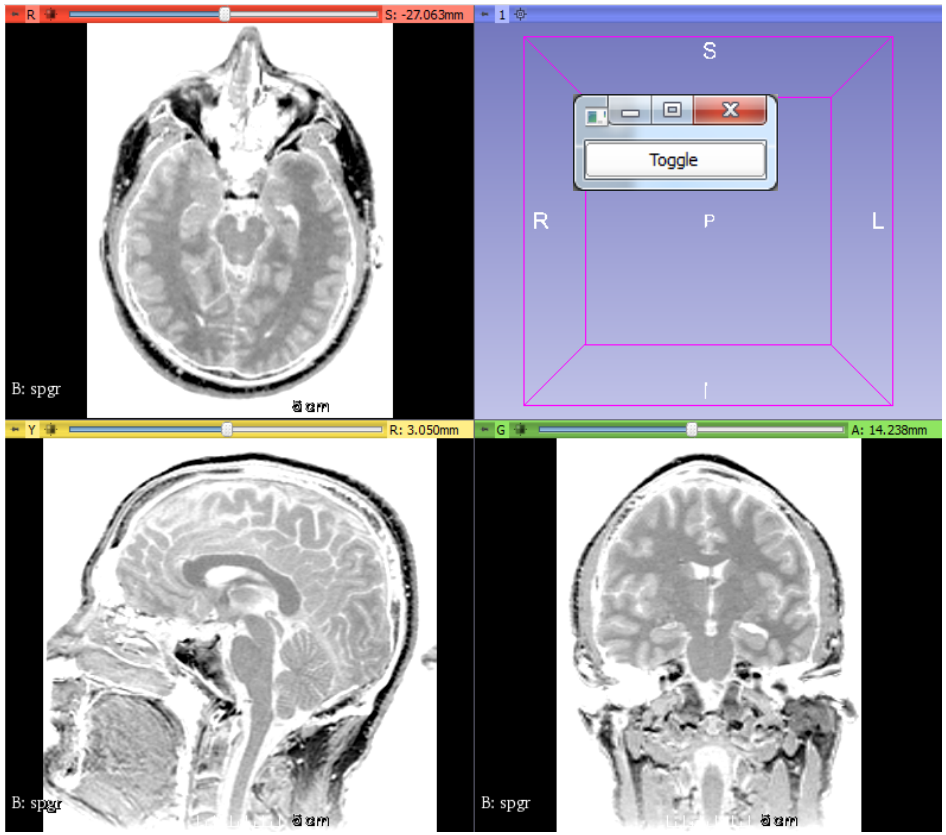
Qt GUI in Python



Result with button toggling



→



Original

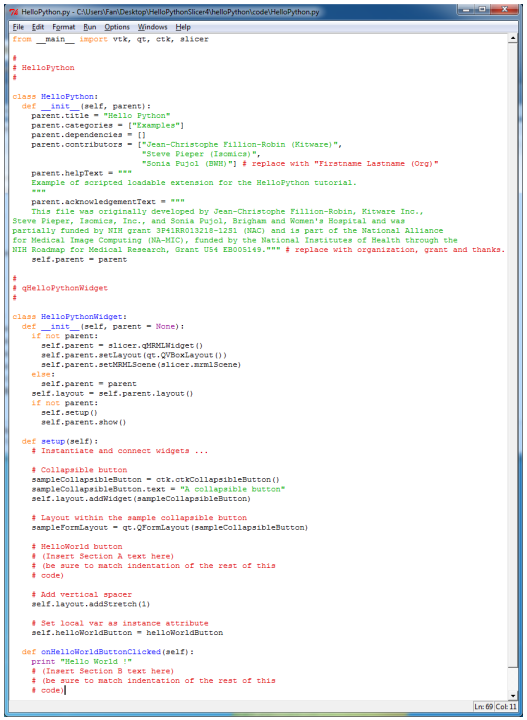
Result

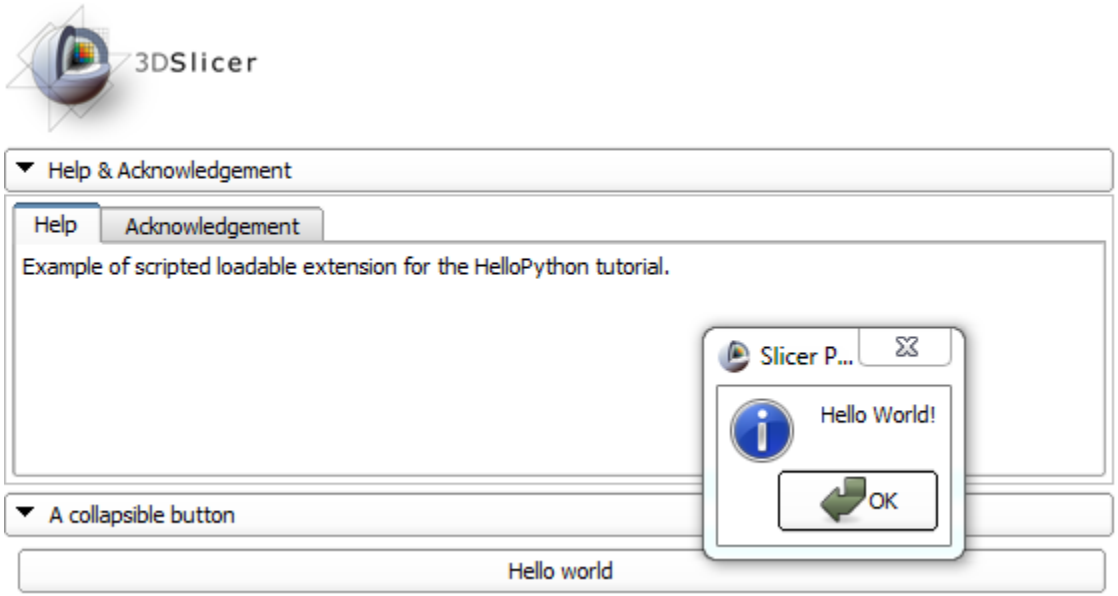
Put the slicer view windows front to see the changes

In More Detail

- Slicer uses **PythonQt** to expose the Qt library in Python (http://pythonqt.sourceforge.net)

- Sophisticated interactive modules can be written entirely with Python code calling C++ code that is wrapped in Python
  - e.g. Endoscopy, SegmentEditor, SampleData, ChangeTracker, and other slicer modules in the Slicer source code



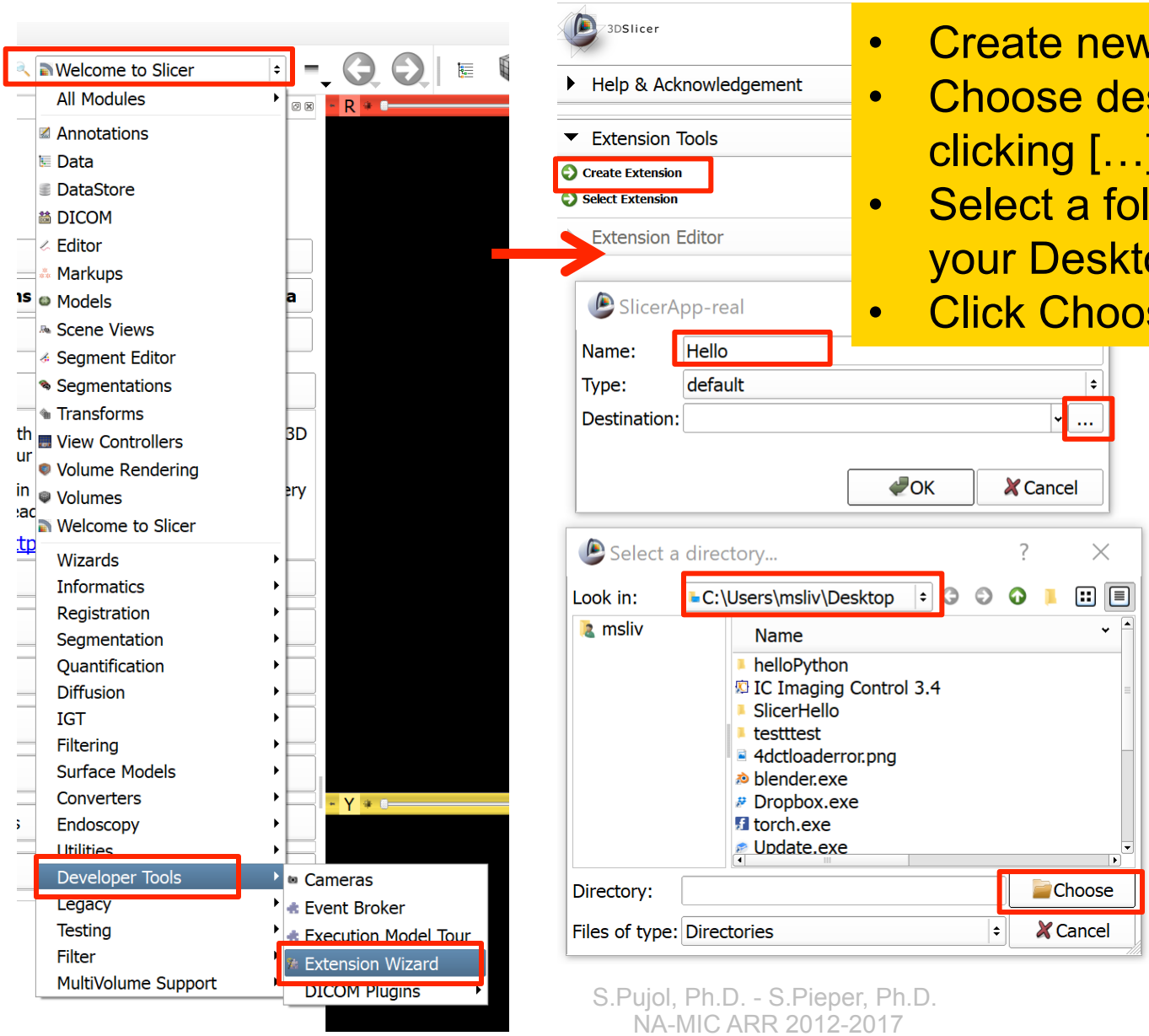


PART B: INTEGRATION OF THE HELLOPYTHON TUTORIAL TO SLICER4

Modules and Extensions

- **Module:**
  - Defines user interface, processing, self-tests
  - Various types: Scripted (Python), Command Line Interface (CLI), Loadable (C++)
- **Extension:**
  - Collection of modules
  - Packaged in a single zip file and distributed through the Extension Manager (Slicer app store)

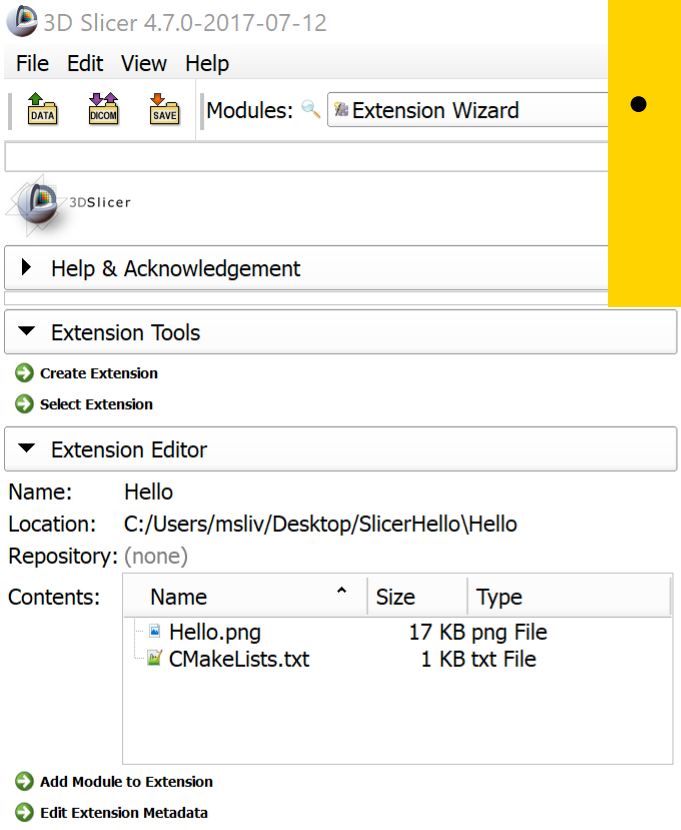
Create new extension

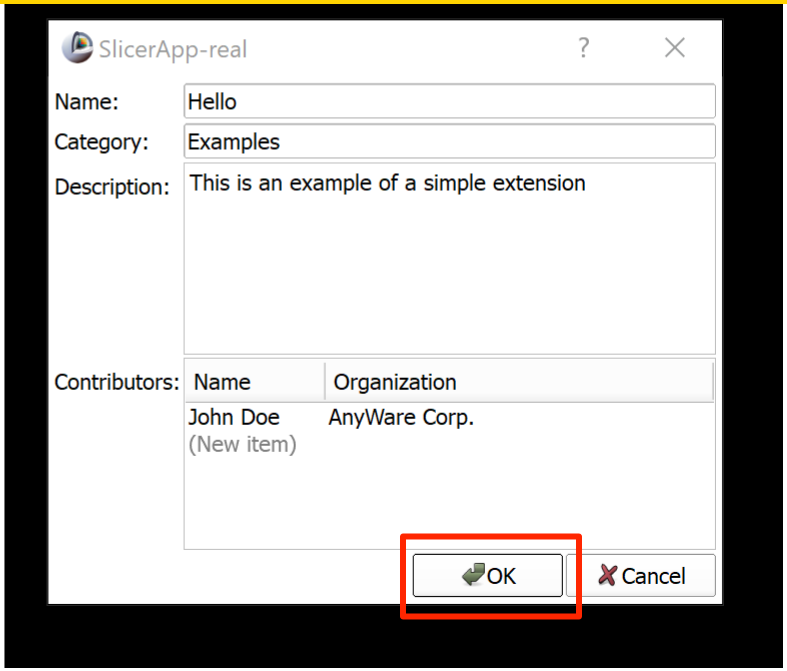


- Create new extension: “Hello”
- Choose destination folder by clicking […] button
- Select a folder (for example, your Desktop)
- Click Choose and OK

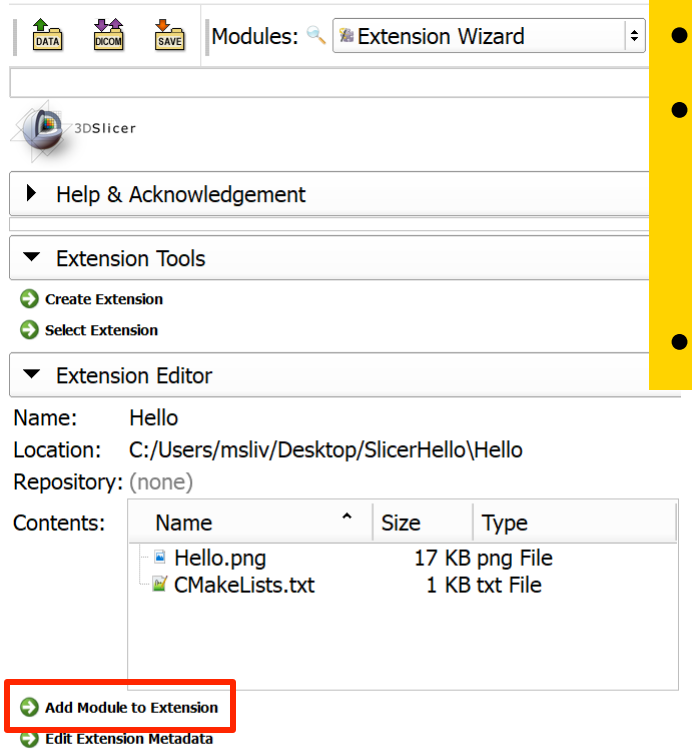
Set extension metadata

- No need for setting metadata now, just click OK.
- Metadata can be edited later by clicking “Edit Extension Metadata”

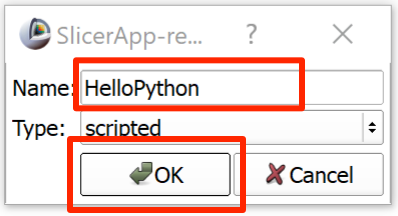




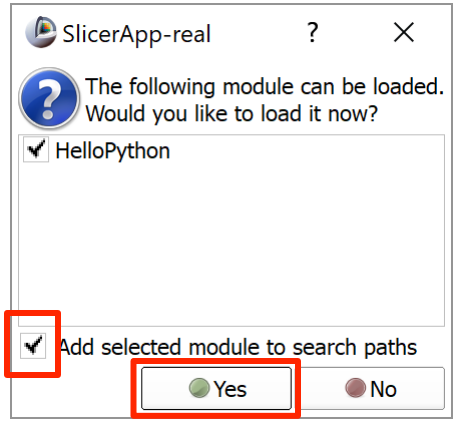
Add module



- Click “Add module to Extension”
- Enter “HelloPython” as module name (must not contain spaces or special characters)
- Click “OK”

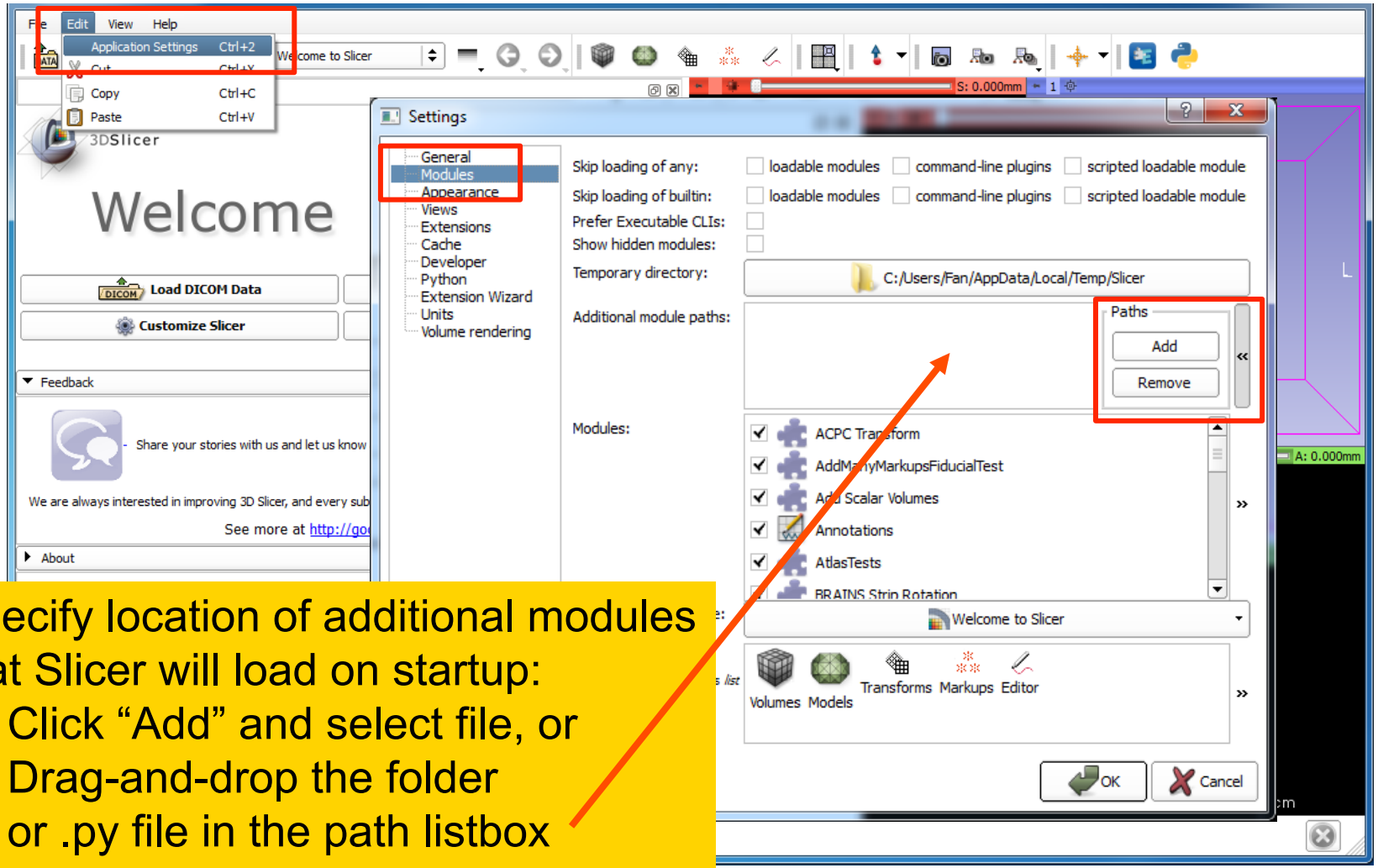


Check “Add selected module…” to load the module automatically when Slicer is restarted



Adding modules to be loaded

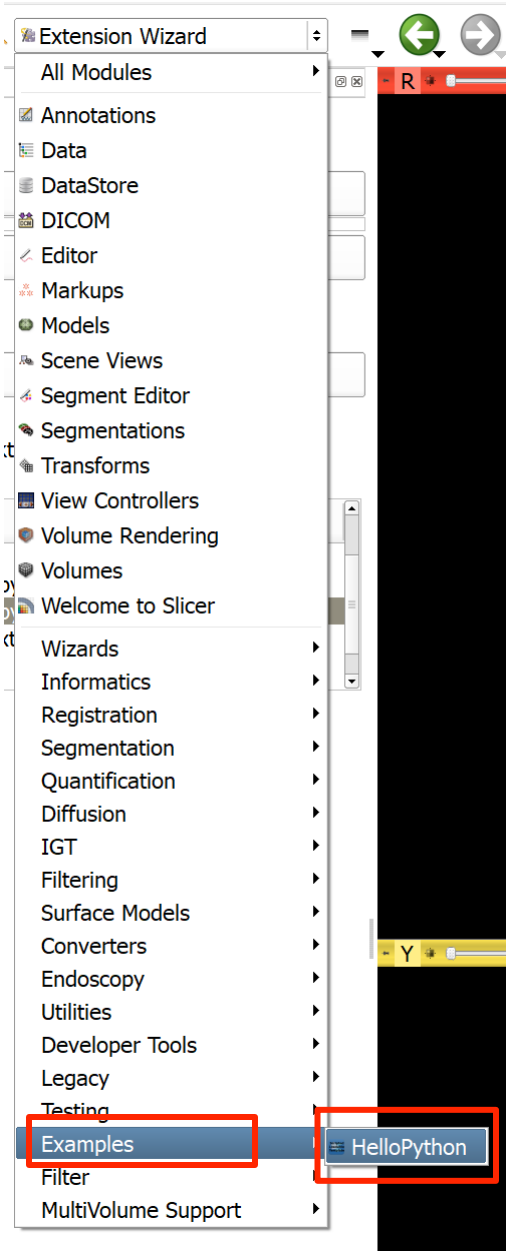
For future reference only - no need to add the module manually if “Add selected module to search paths” box (see previous slide) was checked.

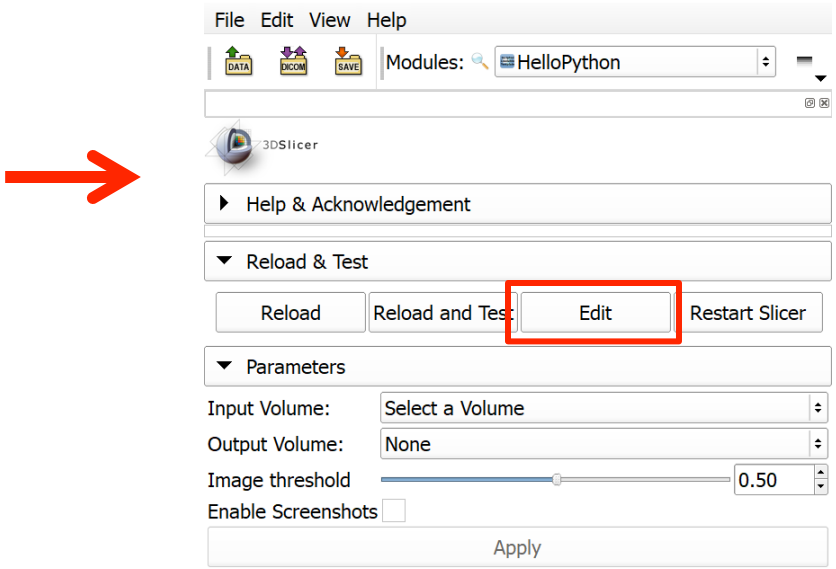


Specify location of additional modules that Slicer will load on startup:

- Click “Add” and select file, or
- Drag-and-drop the folder or .py file in the path listbox

Edit module





- Click “Edit” to open source code of the module in default application associated with .py files
- If the source code is not opened automatically then start an editor and load the source code file: …*\Desktop\Hello \HelloPython\HelloPython.py*

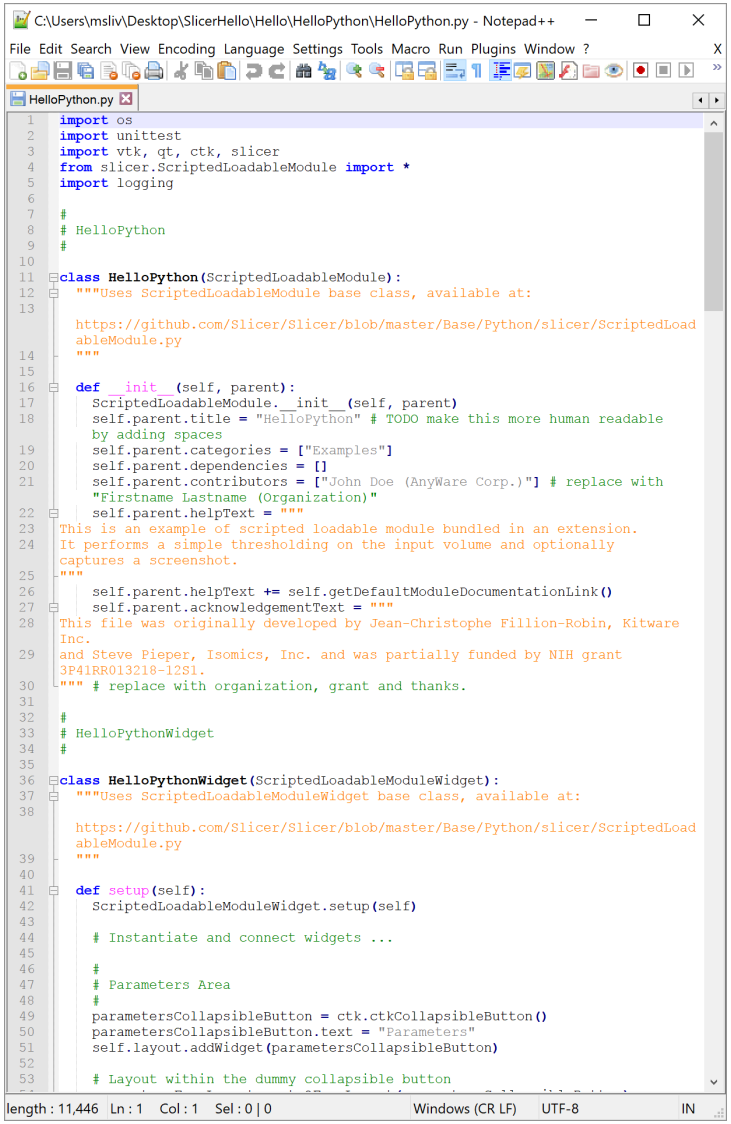
HelloPython.py

Module Description

Module Widget (GUI)

Module Logic (processing, no user interface)

Module Test (automatic self-tests)



Update Module Description

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

  def __init__(self, parent):
    ScriptedLoadableModule.__init__(self, parent)
    self.parent.title = "HelloPython" # TODO make this more human readable by adding spaces
    self.parent.categories = ["Examples"]
    self.parent.dependencies = []
    self.parent.contributors = ["John Doe (AnyWare Corp.)"]
    self.parent.helpText = """
This is an example of scripted loadable module bundled in an extension.
It performs a simple thresholding on the input volume and optionally captures a screenshot.
"""
    self.parent.helpText += self.getDefaultModuleDocumentationLink()
    self.parent.acknowledgementText = """
This file was originally developed by Jean-Christophe Fillion-Robin, Kitware Inc.
and Steve Pieper, Isomics, Inc. and was partially funded by NIH grant 3P41RR013218-12S1.
""" # replace with organization, grant and thanks.
```

constructor

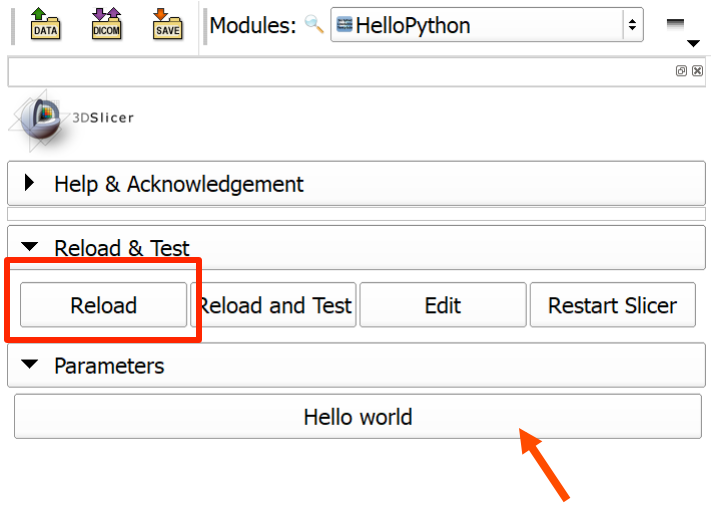
Implement Module GUI

```
class HelloPythonWidget(ScriptedLoadableModuleWidget):
 …
  def setup(self):
    ScriptedLoadableModuleWidget.setup(self)
    …
    parametersCollapsibleButton = ctk.ctkCollapsibleButton()
    parametersCollapsibleButton.text = "Parameters"
    self.layout.addWidget(parametersCollapsibleButton)

    # Layout within the dummy collapsible button
    parametersFormLayout = qt.QFormLayout(parametersCollapsibleButton)

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

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

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

def onHelloWorldButtonClicked(self):
    logic = HelloPythonLogic()
    result = logic.process()
    qt.QMessageBox.information(slicer.util.mainWindow(),
      'Slicer Python', result)
```

Update widget to have only a single button

Reload module

- Save HelloPython.py file
- Click “Reload” button to update the module based on changes in the source code
- If the module GUI is not updated or disappears then check the Python console for error messages

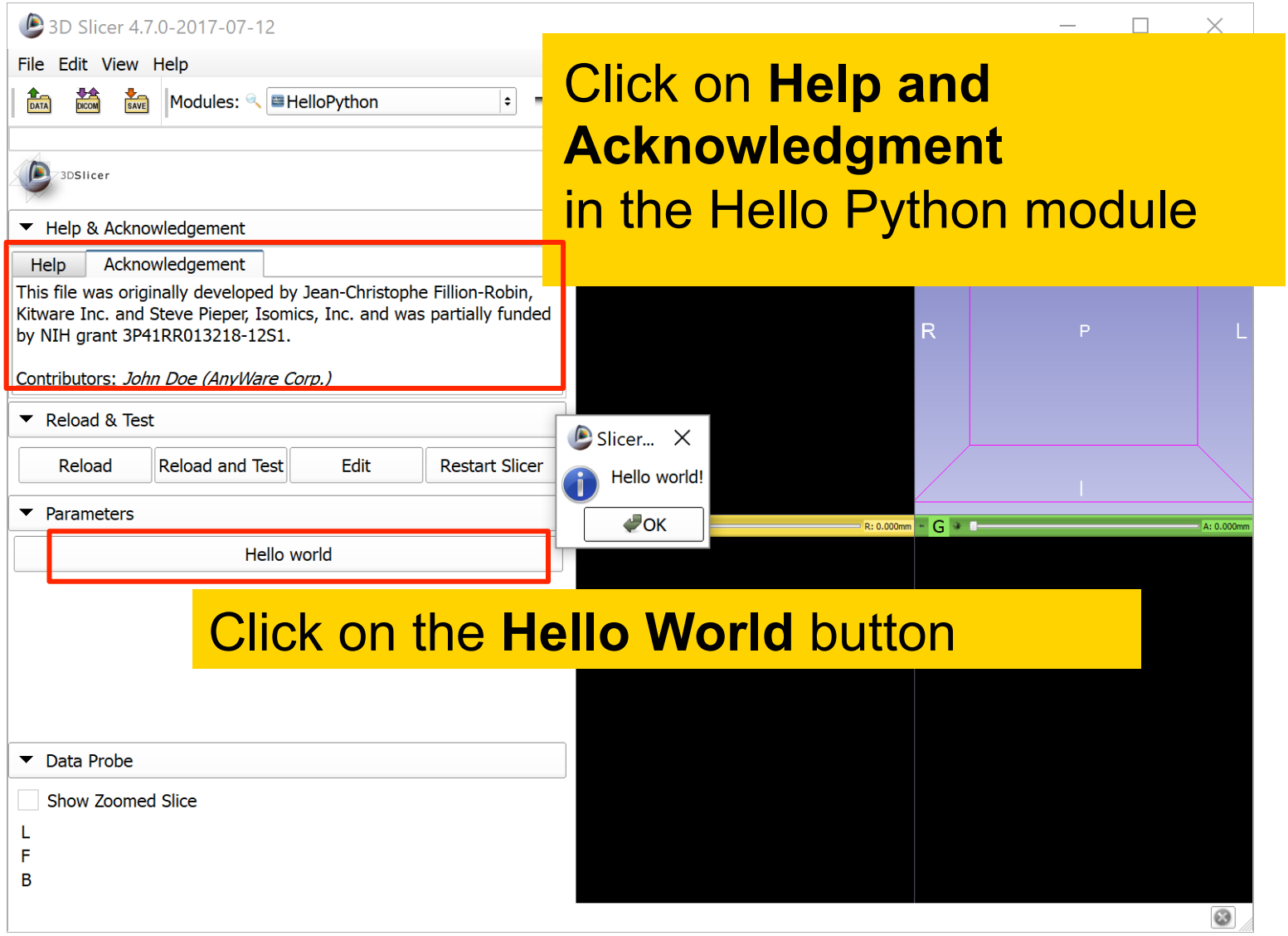


Implement Module Logic

Replace content of logic class with a single method.

```
class HelloPythonLogic(ScriptedLoadableModuleLogic):
  """This class should implement all the actual
  computation done by your module.  The interface
  should be such that other python code can import
  this class and make use of the functionality without
  requiring an instance of the Widget.
  Uses ScriptedLoadableModuleLogic base class, available at:
  """

  def process(self):
    return "Hello world!"
```

Run the module



Implement self-test

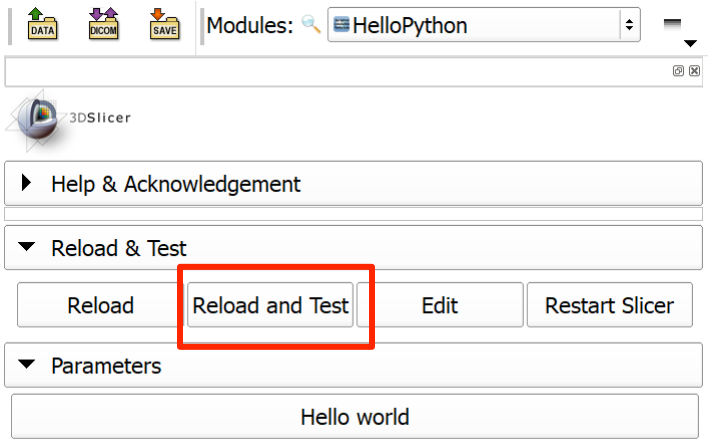
Replace test_HelloPython1 with a simple test.
Use *asserts*, such as `assertIsNone(…)`, `assertTrue(…)` to test that the module works correctly.

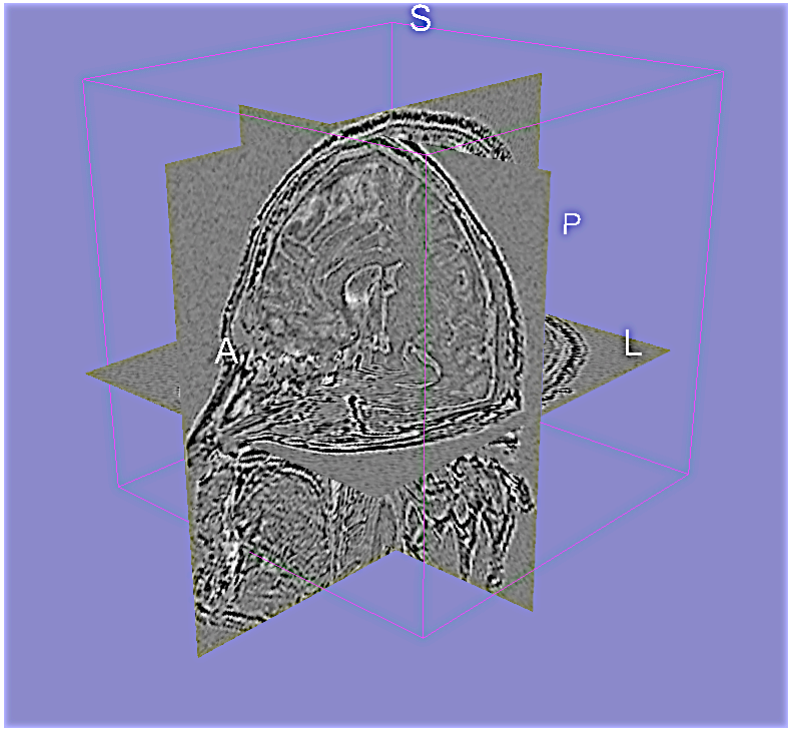
```
class HelloPythonTest(ScriptedLoadableModuleTest):

  def setUp(self):
    slicer.mrmlScene.Clear(0)

  def runTest(self):
    self.setUp()
    self.test_HelloPython1()

  def test_HelloPython1(self):

    self.delayDisplay("Starting the test")
    logic = HelloPythonLogic()
    result = logic.process()
    self.assertIsNotNone(result)
    self.delayDisplay('Test passed!')
```

If an extension is submitted to the Extension Manager (Slicer App Store) then all tests of all of its modules are executed for every release and results are reported on the dashboard: http://slicer.cdash.org/index.php?project=Slicer4

Test the module

- Save HelloPython.py file
- Click “Reload and Test” button to update the module based on changes in the source code and run all the tests





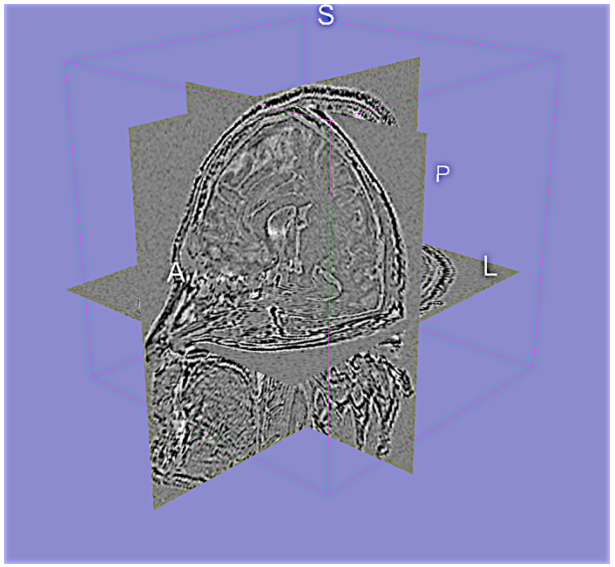
Part C:

Implementing the Laplace* Operator

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

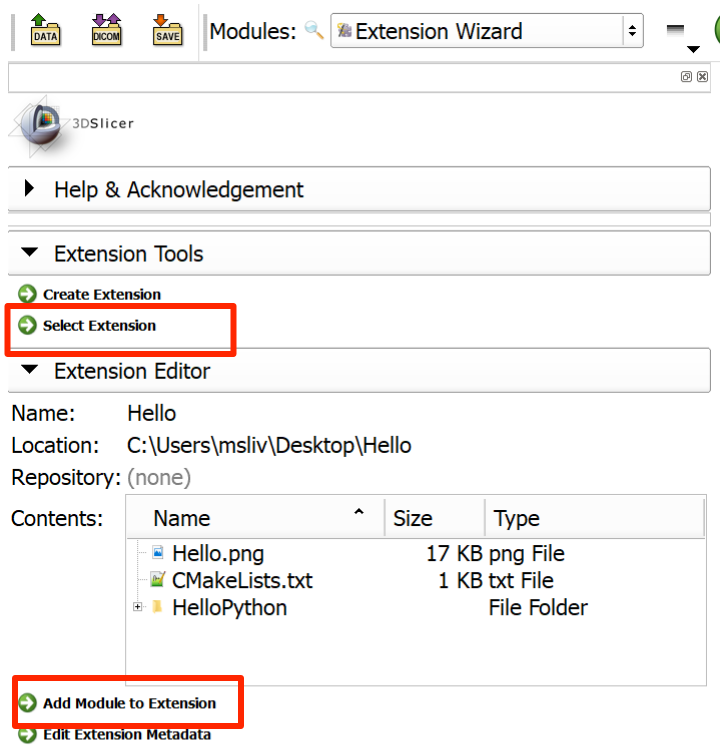
Overview

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

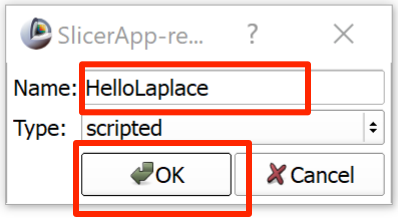


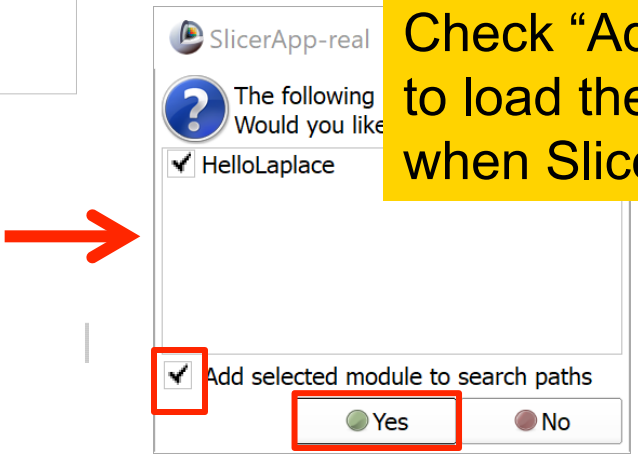
- Use qMRML widgets: widgets that automatically track the state of the Slicer MRML scene
- Use VTK filters to manipulate volume data

Add module to existing extension



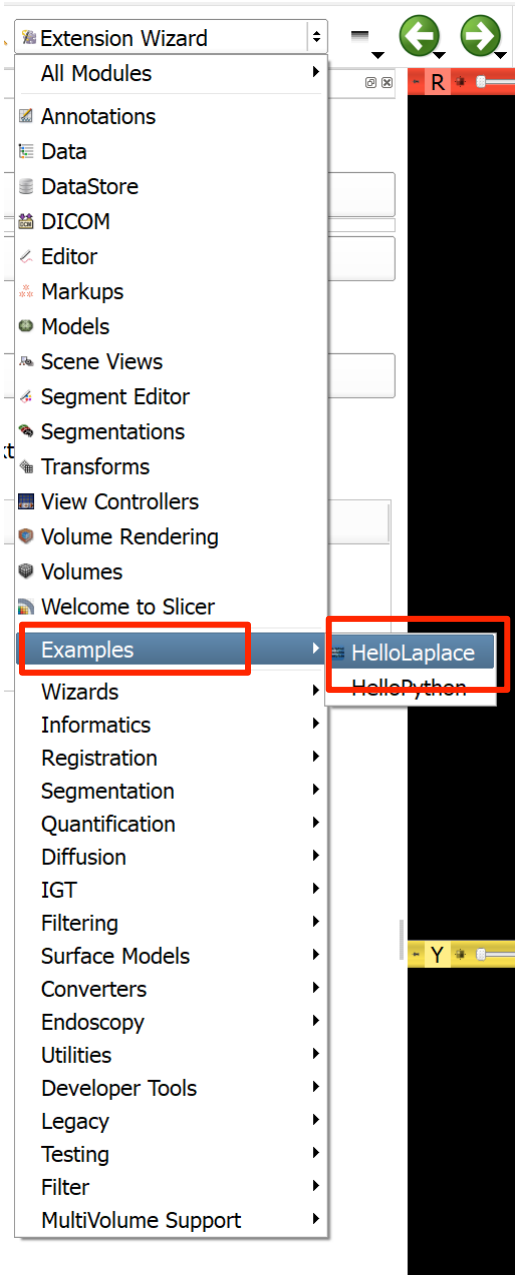
- Open Extension Wizard module (press Ctrl+F shortcut, type `exte`, and select Extension Wizard from list)
- Click “Select Extension”, select your extension folder (…Desktop/Hello)
- Click Add module to Extension”
- Enter “HelloLaplace” as module name
- Click “OK”

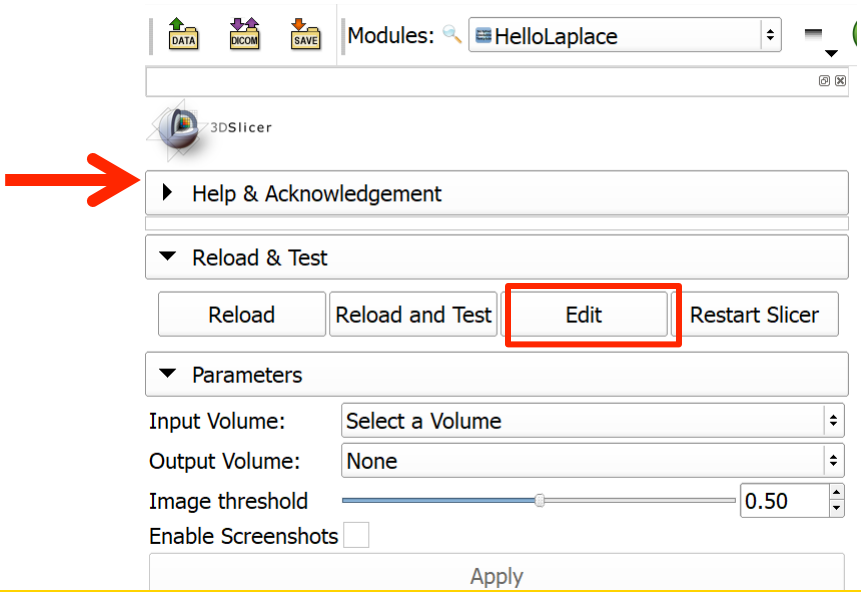




Check “Add selected module…” to load the module automatically when Slicer is restarted

Edit module





- Click “Edit” to open source code of the **HelloLaplace** module in default application associated with .py files
- If the source code is not opened automatically then start an editor and load the source code file: …*\Desktop\Hello\HelloPython\HelloLaplace.py*

Implement Module GUI (Part 1)

We only need input and output volume selectors, so **remove threshold value** and **check box** sections from the HelloLaplace Widget

```
class HelloLaplaceWidget(ScriptedLoadableModuleWidget):
…

    self.outputSelector.setToolTip( "Pick the output to the algorithm." )
    parametersFormLayout.addRow("Output Volume: ", self.outputSelector)

    #
    # threshold value
    #
    self.imageThresholdSliderWidget = ctk.ctkSliderWidget()
    self.imageThresholdSliderWidget.singleStep = 0.1
    self.imageThresholdSliderWidget.minimum = -100
    self.imageThresholdSliderWidget.maximum = 100
    self.imageThresholdSliderWidget.value = 0.5
    self.imageThresholdSliderWidget.setToolTip("Set threshold value for computing the
output image. Voxels that have intensities lower than this value will set to zero.")
    parametersFormLayout.addRow("Image threshold", self.imageThresholdSliderWidget)

    #
    # check box to trigger taking screen shots for later use in tutorials
    #
    self.enableScreenshotsFlagCheckBox = qt.QCheckBox()
    self.enableScreenshotsFlagCheckBox.checked = 0
    self.enableScreenshotsFlagCheckBox.setToolTip("If checked, take screen shots for
tutorials. Use Save Data to write them to disk.")
    parametersFormLayout.addRow("Enable Screenshots",
self.enableScreenshotsFlagCheckBox)

    #
    # Apply Button
    #
    self.applyButton = qt.QPushButton("Apply")
…
```

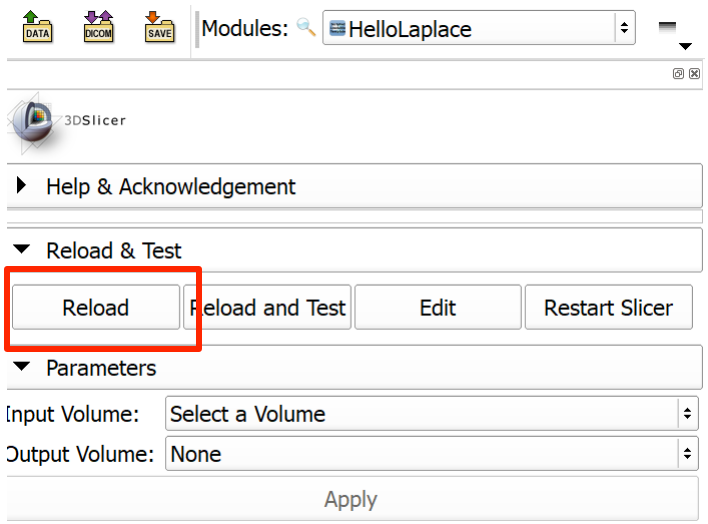
Implement Module GUI (Part 2)

Update Apply button event handler (onApplyButton) to only use input and output volumes

```
class HelloLaplaceWidget(ScriptedLoadableModuleWidget):
…

  def onSelect(self):
    self.applyButton.enabled =
      self.inputSelector.currentNode() and
      self.outputSelector.currentNode()

 def onApplyButton(self):
    logic = HelloLaplaceLogic()
    logic.run(self.inputSelector.currentNode(),
      self.outputSelector.currentNode())
    # make the output volume appear in all the slice views
    slicer.util.setSliceViewerLayers(
      background = self.outputSelector.currentNode())

#
# HelloLaplaceLogic
#
```

Reload module

- Save HelloLaplace.py file
- Click “Reload” button to update the module based on changes in the source code
- “Image threshold” slider and “Enable screenshots” checkbox should disappear
- If the module GUI is not updated or disappears then check the Python console for error messages



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 (we used *vtkMRMLScalarVolumeNode*). Documented here: http://apidocs.slicer.org/master/

Implement Module Logic

Update processing method to compute output by a VTK filter.

```
class HelloLaplaceLogic(ScriptedLoadableModuleLogic):
…
  def run(self, inputVolume, outputVolume):

    if not self.isValidInputOutputData(inputVolume, outputVolume):
      return False

    logging.info('Processing started')

    laplacian = vtk.vtkImageLaplacian()
    laplacian.SetInputData (inputVolume.GetImageData())
    laplacian.SetDimensionality(3)
    laplacian.Update()

    # Copy image origin, spacing, directions from input
    ijkToRAS = vtk.vtkMatrix4x4()
    inputVolume.GetIJKToRASMatrix(ijkToRAS)
    outputVolume.SetIJKToRASMatrix(ijkToRAS)
    outputVolume.SetAndObserveImageData(laplacian.GetOutput())

    logging.info('Processing completed')

    return True
```

In More Detail

- **vtkImageLaplacian** is a vtkImageAlgorithm operates on vtkImageData (see http://vtk.org)

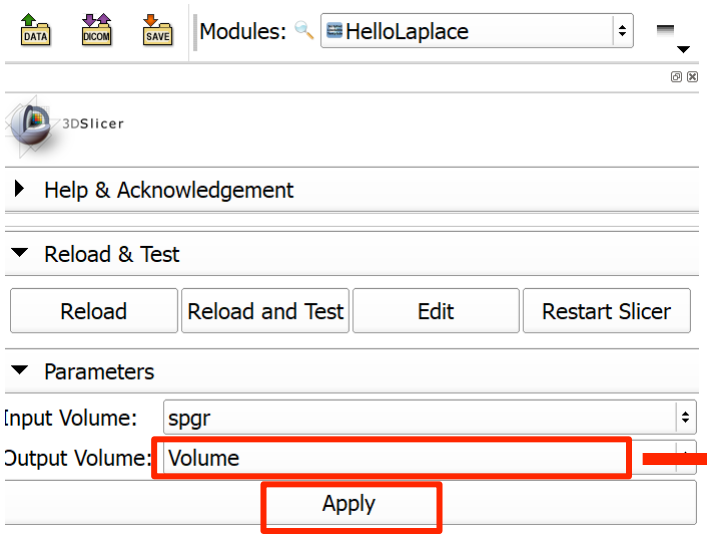
- **vtkMRMLScalarVolumeNode** is a Slicer MRML class that contains vtkImageData, plus orientation information ijkToRAS matrix (see http://www.slicer.org/slicerWiki/index.php/Coordinate_systems)

In More Detail (Continued)

- Global **slicer** package gives python access to:
  - Convenience functions: **slicer.util**
    http://slicer.readthedocs.io/en/latest/developer_guide/slicer.html#module-slicer.util
  - Data: **slicer.mrmlScene**
    http://apidocs.slicer.org/master/classvtkMRMLScene.html
  - CLI modules: **slicer.cli**
  - Scripted modules: by importing Python classes
  - Loadable modules: **slicer.modules.(modulename).logic()**
    http://apidocs.slicer.org/master/classvtkSlicerModuleLogic.html
  - Application GUI: **slicer.app**
    http://apidocs.slicer.org/master/classqSlicerApplication.html

Running the module

1. Note that Input Volume combobox auto-selected new volume

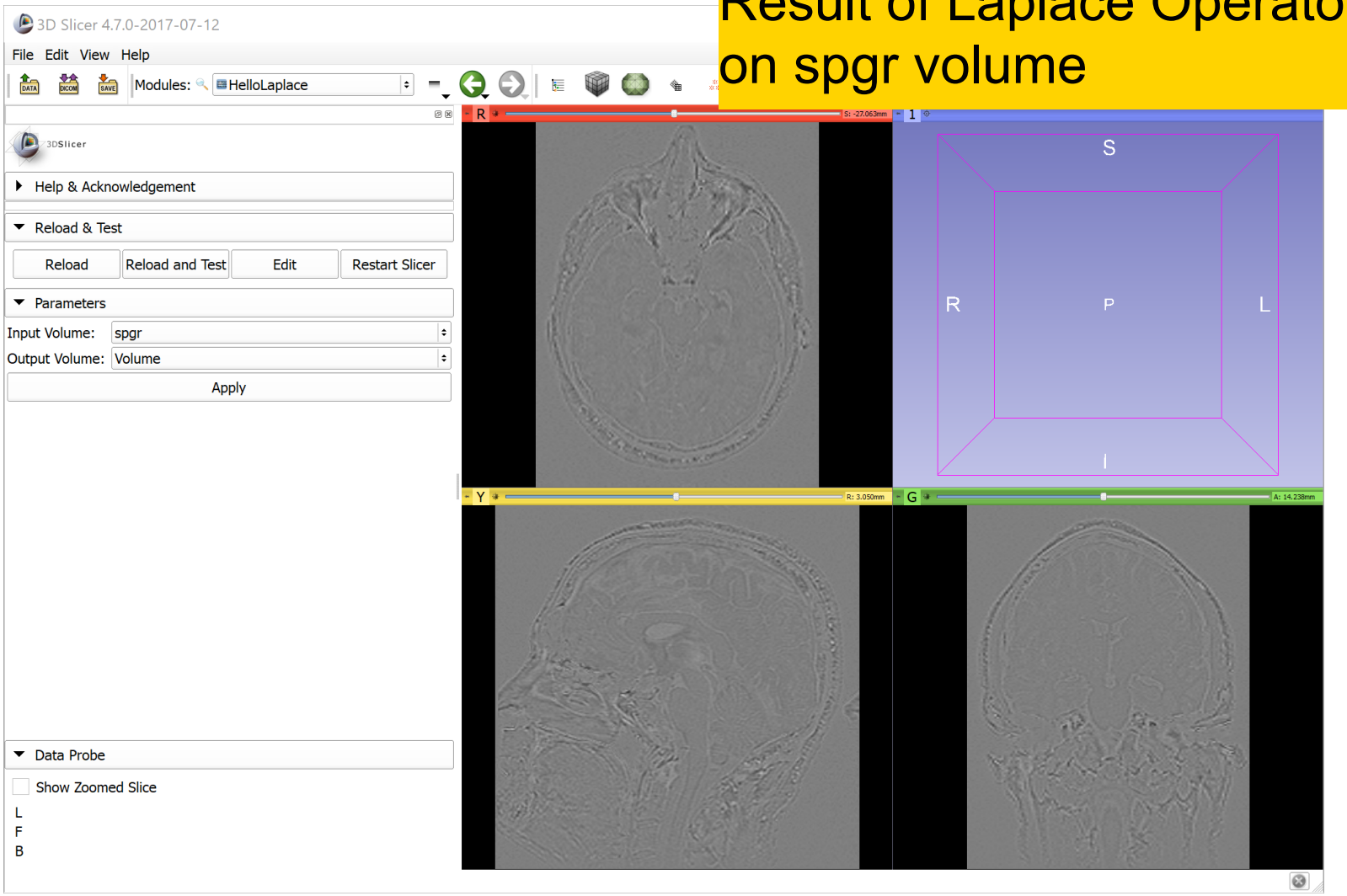


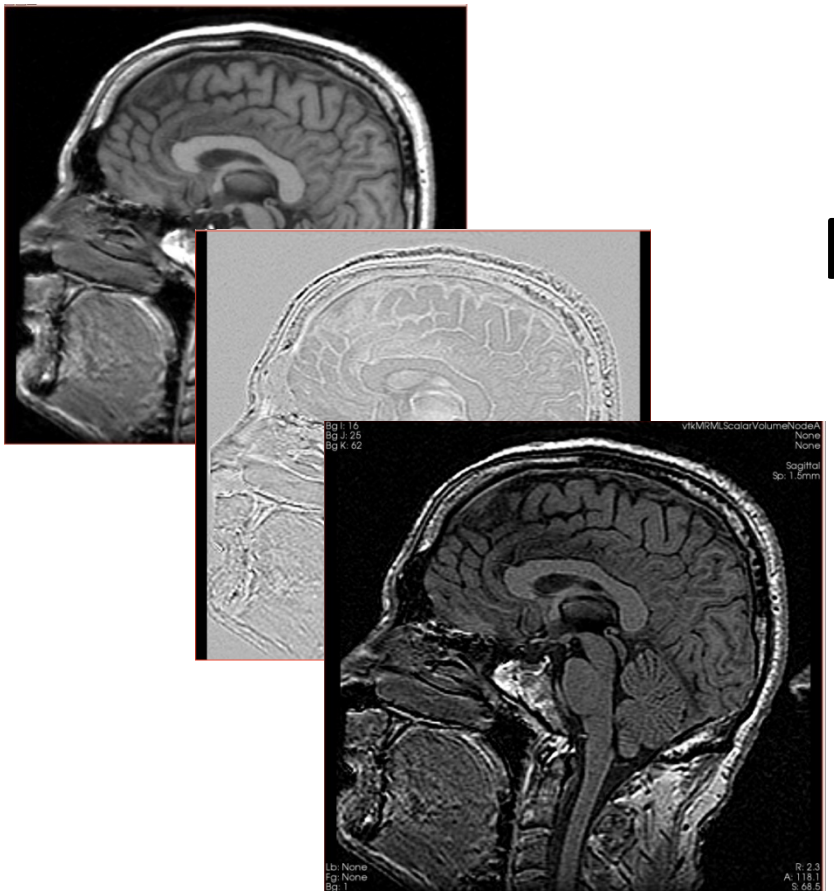
2. Create new volume for output

3. Run the module

Output of HelloLaplace

Result of Laplace Operator on spgr volume



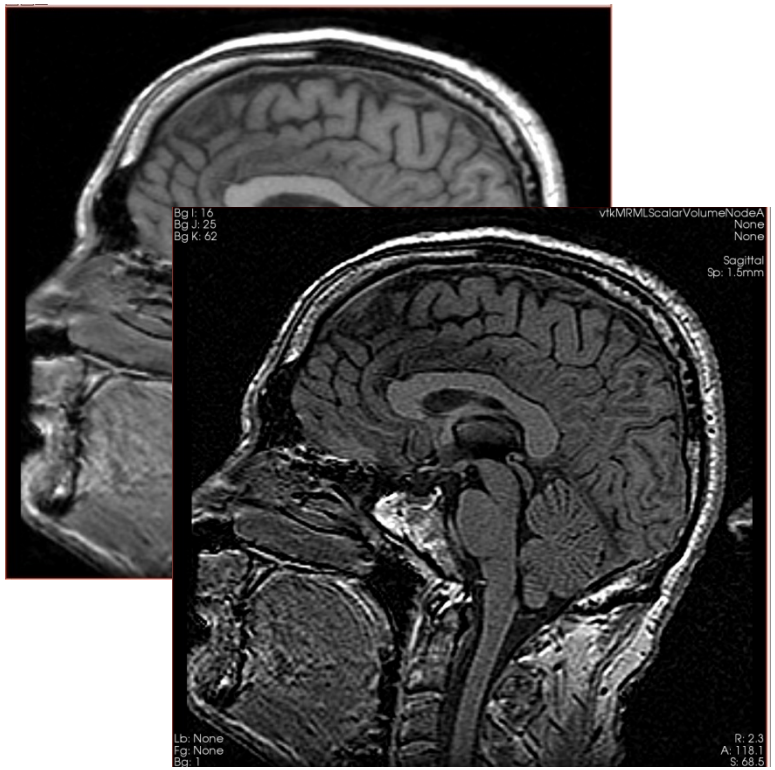


Part D:

Image Sharpening with Laplace Operator and subtraction

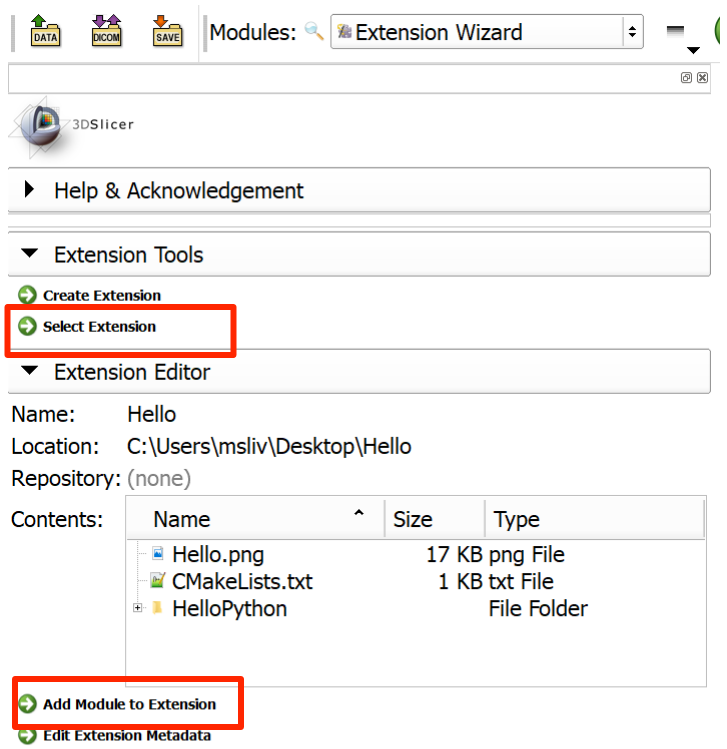
Overview

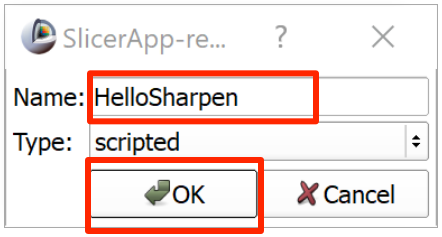
The goal of this section is to add a processing option for image sharpening.



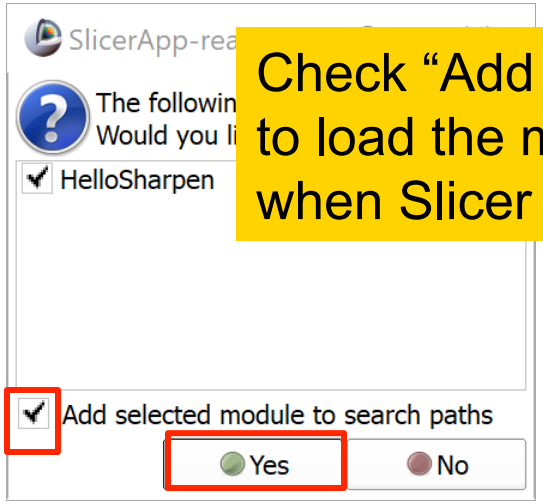
- We'll implement this operation using our HelloLaplace module and an existing Slicer Command Line Module 'Subtract Scalar Volumes'

Add module to existing extension



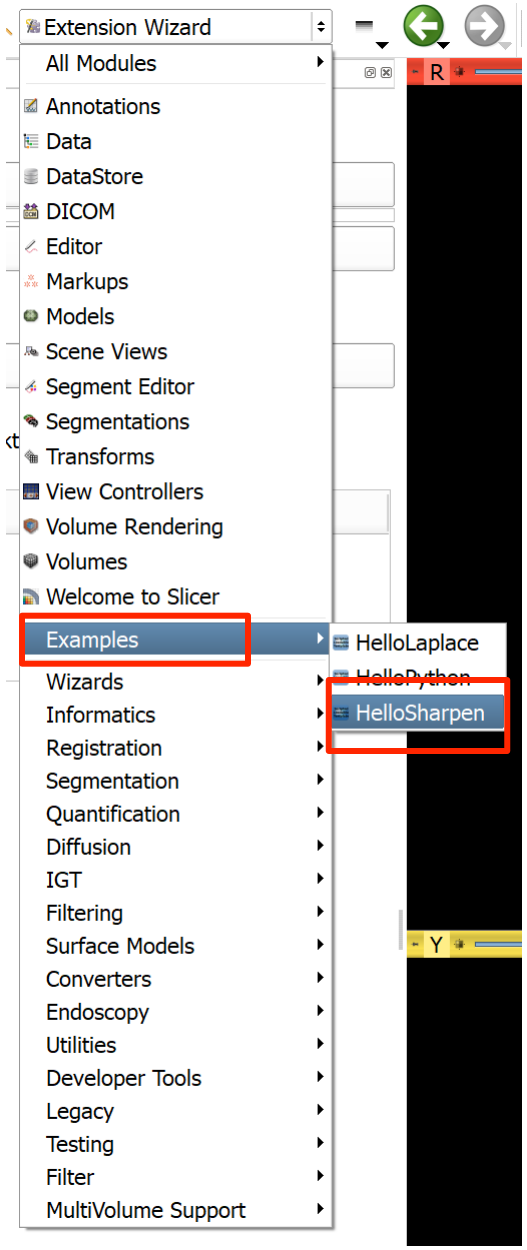


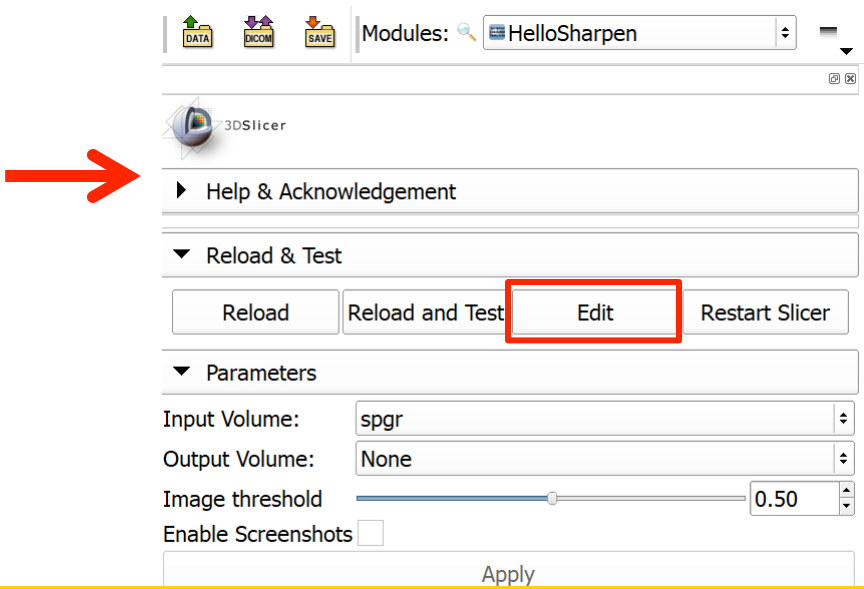
- Open Extension Wizard module (press Ctrl+F shortcut, type `exte`, and select Extension Wizard from list)
- Click “Select Extension”, select your extension folder (…Desktop/Hello)
- Click Add module to Extension”
- Enter “HelloSharpen” as module name
- Click “OK”



Check “Add selected module…” to load the module automatically when Slicer is restarted

Edit module





- Click “Edit” to open source code of the **HelloSharpen** module in default application associated with .py files
- If the source code is not opened automatically then start an editor and load the source code file: …*\Desktop\Hello\HelloPython\HelloSharpen.py*

Implement Module GUI (Part 1)

Remove unneeded slider and checkbox (same as for HelloLaplace module)

```
class HelloSharpenWidget(ScriptedLoadableModuleWidget):
…

    self.outputSelector.setToolTip( "Pick the output to the algorithm." )
    parametersFormLayout.addRow("Output Volume: ", self.outputSelector)

    #
    # threshold value
    #
    self.imageThresholdSliderWidget = ctk.ctkSliderWidget()
    self.imageThresholdSliderWidget.singleStep = 0.1
    self.imageThresholdSliderWidget.minimum = -100
    self.imageThresholdSliderWidget.maximum = 100
    self.imageThresholdSliderWidget.value = 0.5
    self.imageThresholdSliderWidget.setToolTip("Set threshold value for computing the
output image. Voxels that have intensities lower than this value will set to zero.")
    parametersFormLayout.addRow("Image threshold", self.imageThresholdSliderWidget)

    #
    # check box to trigger taking screen shots for later use in tutorials
    #
    self.enableScreenshotsFlagCheckBox = qt.QCheckBox()
    self.enableScreenshotsFlagCheckBox.checked = 0
    self.enableScreenshotsFlagCheckBox.setToolTip("If checked, take screen shots for
tutorials. Use Save Data to write them to disk.")
    parametersFormLayout.addRow("Enable Screenshots",
self.enableScreenshotsFlagCheckBox)

    #
    # Apply Button
    #
    self.applyButton = qt.QPushButton("Apply")
…
```

Implement Module GUI (Part 2)

Update Apply button event handler (onApplyButton) to only use input and output volumes

```
class HelloSharpenWidget(ScriptedLoadableModuleWidget):
…

  def onSelect(self):
    self.applyButton.enabled =
      self.inputSelector.currentNode() and
      self.outputSelector.currentNode()

 def onApplyButton(self):
    logic = HelloSharpenLogic()
    logic.run(self.inputSelector.currentNode(),
      self.outputSelector.currentNode())
    # make the output volume appear in all the slice views
    slicer.util.setSliceViewerLayers(
      background = self.outputSelector.currentNode())

#
# HelloSharpenLogic
#
```

Implement Module Logic

Update processing method to compute output by using HelloLaplace module and SubtractVolumes CLI module.

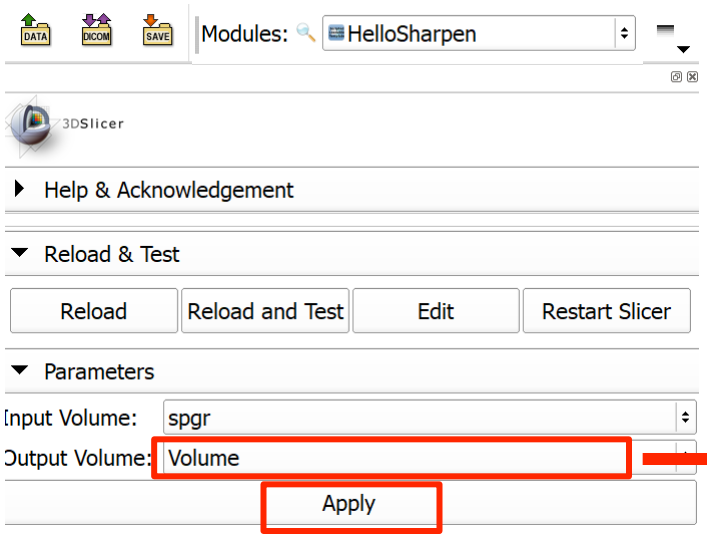
```
class HelloSharpenLogic(ScriptedLoadableModuleLogic):
…

 def run(self, inputVolume, outputVolume):

    if not self.isValidInputOutputData(inputVolume, outputVolume):
      return False

    logging.info('Processing started')

    # Compute Laplacian by scripted module
    import HelloLaplace
    laplaceLogic = HelloLaplace.HelloLaplaceLogic()
    success = laplaceLogic.run(inputVolume, outputVolume)
    if not success:
      return False

    # Subtract Laplacian from input volume using CLI module
    parameters = {}
    parameters['inputVolume1'] = inputVolume.GetID()
    parameters['inputVolume2'] = outputVolume.GetID()
    parameters['outputVolume'] = outputVolume.GetID()
    slicer.cli.runSync(slicer.modules.subtractscalarvolumes, None,
      parameters)

    logging.info('Processing completed')

    return True
```

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
- Note: if only ITK filters needed, they can be accessed directly from Python, using SimpleITK

Run module

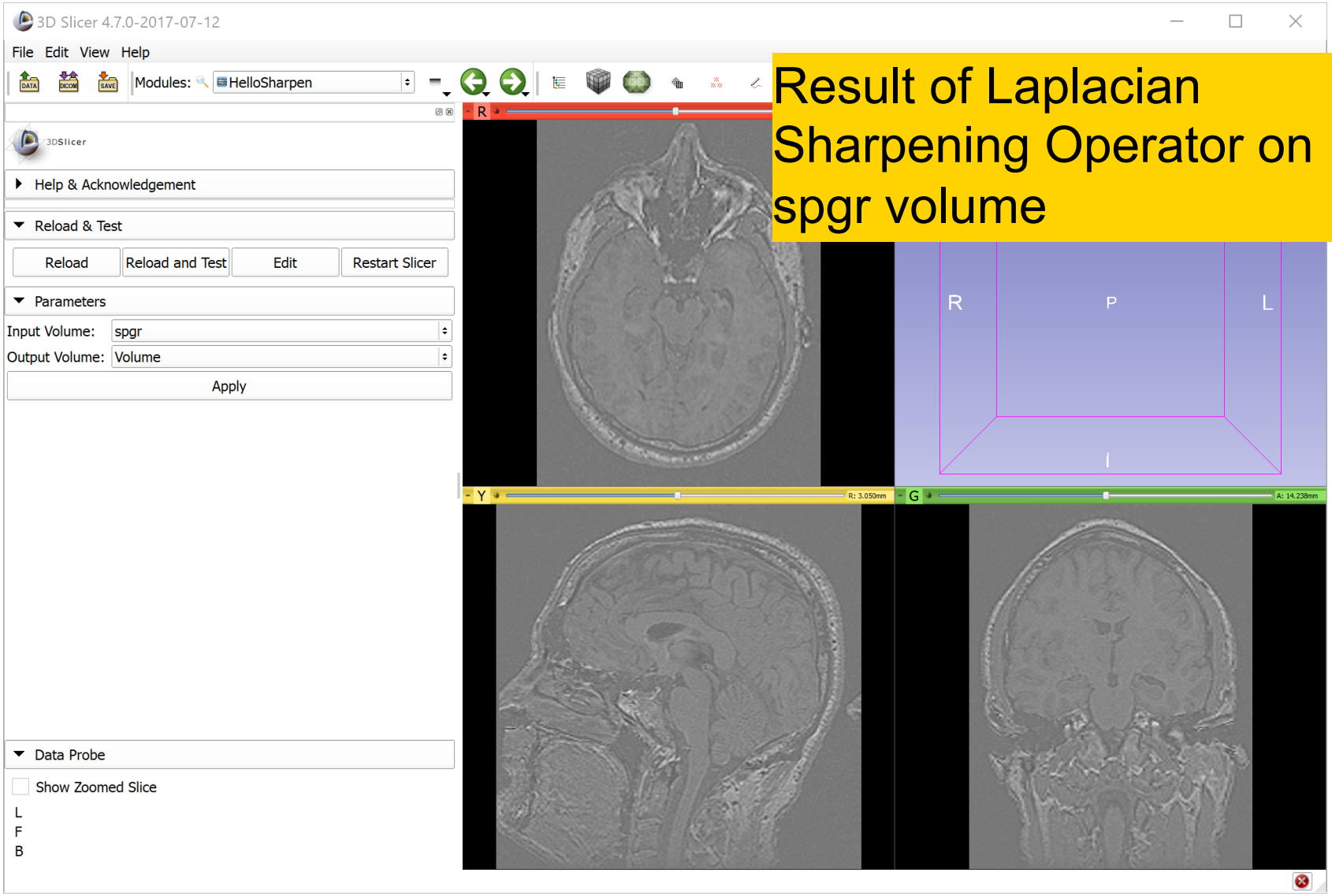
1. Note that Input Volume combobox autoselected new volume



2. Create new volume for output

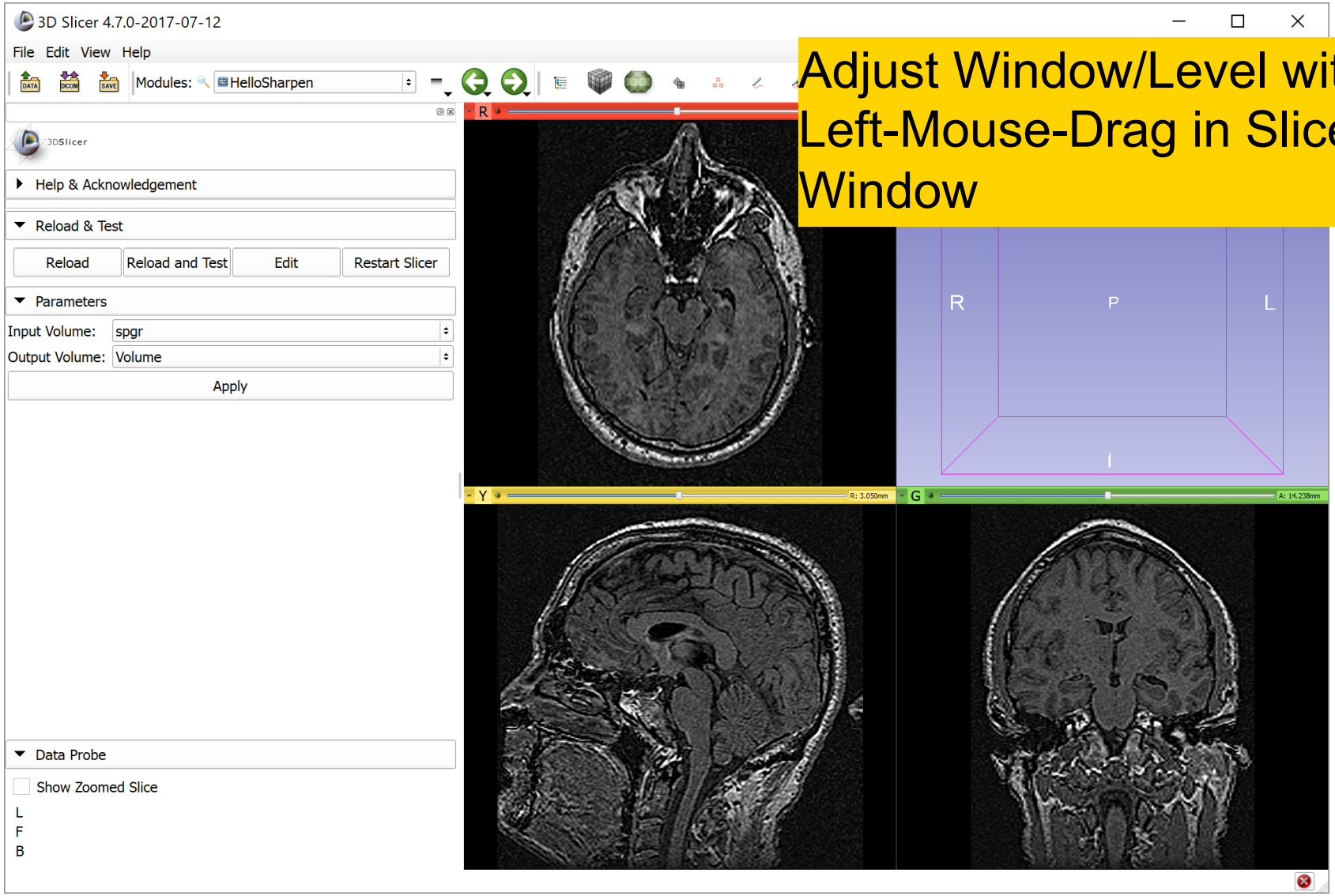
3. Run the module

Output of HelloSharpen



Result of Laplacian Sharpening Operator on spgr volume

Adjusting output display



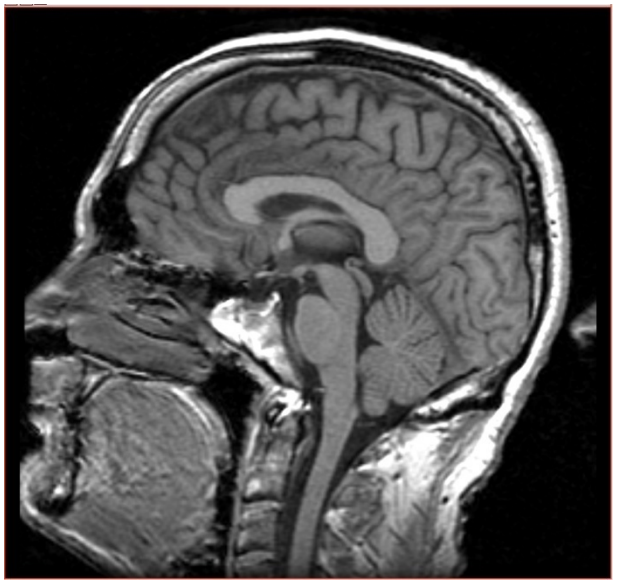
Improve Module Logic

Set output volume's window/level automatically, by copying the input volume's window/level.

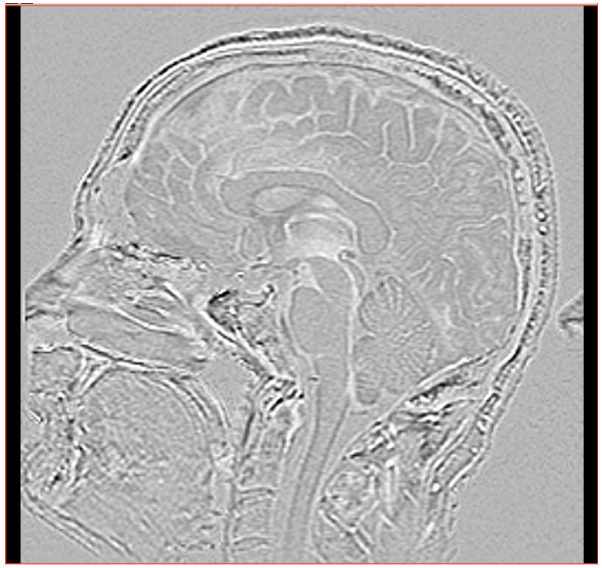
```
class HelloSharpenLogic(ScriptedLoadableModuleLogic):
…
  def run(self, inputVolume, outputVolume):
  …

    slicer.cli.runSync(slicer.modules.subtractscalarvolumes, None,
      parameters)

    # Copy window/level (brightness/contrast) setting to output
    inputDisplay = inputVolume.GetDisplayNode()
    outputDisplay = outputVolume.GetDisplayNode()
    outputDisplay.SetAutoWindowLevel(False)
    outputDisplay.SetWindow(inputDisplay.GetWindow())
    outputDisplay.SetLevel(inputDisplay.GetLevel())

    logging.info('Processing completed')

    return True
```

Image Sharpening Results

original



Laplacian



Sharpen

Going Further

- Review scripted modules in Slicer for more examples
  https://github.com/Slicer/Slicer/tree/master/Modules/Scripted
- Explore numpy for numerical array manipulation
  http://docs.scipy.org/doc/numpy/reference
- Explore SimpleITK for image processing using ITK
  https://itk.org/SimpleITKDoxygen/html/classes.html

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

Andras Lasso, PerkLab, Queen's University