Python and Slicer

NAMIC 2009 AHM, Salt Lake City

Luca Antiga
Medical Imaging Unit, Mario Negri Institute, Bergamo, Italy
Acknowledgements

- Dan Blezek - major contributor and the man behind Python through Tcl
- Demian Wasserman - Scipy wiz and author of Numpy (embedded) slides
- Steve Piper
- Michael Halle
- ...

Why Python

• More than just yet another scripting language

• Object-oriented, garbage-collected, fully introspective, allows metaprogramming

• Comes with batteries included (lots of modules ready to use, e.g. xmlrpc, http, sqlite) and many good quality external modules available

• Widely adopted by the scientific community: Numpy, Scipy, matplotlib, Nipy, ... PETSc, FeNiCs, ... VTK, ITK, MayaVi, vmtk, ...

• Thanks to Scipy, possible alternative to Matlab
Python features

- **Strongly typed**

  ```python
  >>> a = 1
  >>> b = 'cow'
  >>> c = a + b
  TypeError: unsupported operand type(s) for +: 'int' and 'str'
  ```

- **Dynamically typed**

  ```python
  >>> a = 1
  >>> type(a)
  <type 'int'>
  >>> a = 'cow'
  >>> type(a)
  <type 'str'>
  ```

- **Variables as ‘handles’, type attached to the pan rather than the handle; or to the dog rather than the leash**
Python features

• Variables as ‘leashes’
  ```python
  >>> a = [1, 2, 3]
  >>> b = a
  >>> a[1] = 4
  >>> b = [1, 4, 3]
  ```

• Basic datatypes
  • Literals (int, float, complex, bool, str)
  • Tuples: immutable ordered containers
    ```python
    t = ('a', 2)
    t = ['a', 2]
    ```
  • Lists: mutable ordered containers
    ```python
    t = [‘a’, 2]
    ```
  • Dictionaries: key/value maps
    ```python
    t = {‘a’: 2}; t[‘a’] = 3
    ```
  • Sets: unordered unsubscriptable containers
    ```python
    t = sets.Set(‘a’, 2)
    ```
  • Functions
    ```python
def Add(a,b):
    return a+b
t = Add
```  
  • Classes
  • Modules
Python features

• Classes, instances and inheritance

class Cow(object):
    def __init__(self,color):
        self.Color = color
    def GetColor():
        return self.Color

class BrownCow(Cow):
    def __init__(self):
        Cow.__init__(self,'brown')

>>> a = Cow('brown')
>>> a.GetColor()
'brown'
>>> b = BrownCow()
>>> b.GetColor()
'brown'
Numpy, Scipy

Numpy basic Datatypes

- array datatype
  - Multidimensional array
  - Operations are done in an element by element basis
- matrix datatype
  - Bidimensional array of elements
  - matrix semantics
Numpy, Scipy

From Matlab to Python

<table>
<thead>
<tr>
<th>Matlab</th>
<th>Python / Numpy</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(2:5)</td>
<td>a[1:4]</td>
</tr>
<tr>
<td>a(1:end)</td>
<td>a(0:)</td>
</tr>
<tr>
<td>a'</td>
<td>a.T</td>
</tr>
<tr>
<td>a(a&gt;.5)</td>
<td>a[a&gt;.5]</td>
</tr>
<tr>
<td>[V,D]=eig(a)</td>
<td>V,D=linalg.eig(a)</td>
</tr>
</tbody>
</table>

and there are lot of packages for optimization, image processing, statistics, learning, etc.

http://www.scipy.org/NumPy_for_Matlab_Users
Numpy, Scipy

Numpy: slicing

>>> a[0,3:5]
array([[3, 4]])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2, 22, 52])

>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])

Slicing does not create copies of the array's contents
### Numpy: fancy indexing

#### INDEXING BY POSITION
```python
>>> a = arange(0,80,10)
>>> y = a[[1, 2, -3]]
>>> print y
[10 20 50]
```

# fancy indexing
```python
>>> y = a[mask]
>>> print y
[10, 20, 50]
```

# using take
```python
>>> y = take(a, [1, 2, -3])
>>> print y
[10 20 50]
```

#### INDEXING WITH BOOLEANS
```python
>>> mask = array([0, 1, 1, 0, 0, 1, 0, 0],
                dtype=bool)
>>> y = a[mask]
>>> print y
[10, 20, 50]
```

# fancy indexing
```python
>>> y = a[mask]
>>> print y
[10, 20, 50]
```

# using compress
```python
>>> y = compress(mask, a)
>>> print y
[10, 20, 50]
```
Numpy, Scipy

Numpy: fancier indexing

```python
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([[ 1, 12, 23, 34, 45]])

>>> a[3:, [0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```

Unlike slicing, fancy indexing creates copies instead of views into original arrays.

[Jones, Oliphant]
Numpy, Scipy

Numpy broadcasting

Semantic of binary operations between arrays
Python in Slicer

- Wrapping the VTK way
Python in Slicer

- Wrapping the Slicer way
Python in Slicer

- Wrapping the Slicer way

...more details towards the end of the presentation
The Slicer Python shell
The Slicer module

```
Python 2.5.3c1 (release25-maint:67820, Dec 17 2008, 10:12:52)
[GCC 4.0.1 (Apple Inc. build 5465)]
Copyright (c) 2001-2008 Python Software Foundation.
All Rights Reserved.

Copyright (c) 2000 BeOpen.com.
All Rights Reserved.

Copyright (c) 1995-2001 Corporation for National Research Initiatives.
All Rights Reserved.

Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
All Rights Reserved.

Python Console v1.4 by Ka-Ping Yee <ping@lfw.org>
>>> from Slicer import slicer
>>> image = slicer.vtkImageData()
>>> image.GetDimensions()
[0, 0, 0]
>>> interactor = slicer.ApplicationGUI.GetRenderWindowInteractor()
```
Fetching/creating/editing MRML nodes

Python 2.5.3c1 (release25-maint:67820, Dec 17 2008, 10:12:52)
[GCC 4.0.1 (Apple Inc. build 5465)]
Copyright (c) 2001-2008 Python Software Foundation.
All Rights Reserved.

Copyright (c) 2000 BeOpen.com.
All Rights Reserved.

Copyright (c) 1995-2001 Corporation for National Research Initiatives.
All Rights Reserved.

Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
All Rights Reserved.

Python Console v1.4 by Ka-Ping Yee <ping@lfw.org>
>>> from Slicer import slicer
>>> scene = slicer.MRMLScene
>>> t1MRIVolume = scene.GetNodeByID('vtkMRMLScalarVolumeNode1')
>>> type(t1MRIVolume)
<class 'Slicer.vtkMRMLScalarVolumeNode'>
>>> t1MRIImageData = t1MRIVolume.GetImageData()
>>> type(t1MRIImageData)
<class 'Slicer.vtkImageData'>
>>> t1MRIImageData.GetDimensions()
[256, 256, 136]
>>>
Volumes to Numpy ndarrays and back

```python
>>> from Slicer import slicer
>>> scene = slicer.MRMLScene
>>> node = scene.GetNodeByID('vtkMRMLScalarVolumeNode1')
>>> arr = node.GetImageData().ToArray()
>>> type(arr)
<type 'numpy.ndarray'>
>>> arr.max()
367
>>> arr[arr>200] = 200
>>> node.Modified()

>>> arr2D = arr[:,:,2]
>>> node.GetImageData().FromArray2D(arr2D)
```
Controlling Slicer from Python

```python
>>> from Slicer import slicer
>>> layout = slicer.ApplicationGUI.GetGUILayoutNode()
>>> layout.SetViewArrangement(3)
```
Command-line (XML) modules

- In addition to executables and shared libraries
- Readily available: simply copy the .py file in the Plugins directory (or point Slicer to an external Plugins directory)
- Slicer doesn’t have to be restarted for changes to the code inside Execute to take effect
- Run in the main thread, i.e. they can change the MRML scene.
Command-line (XML) modules

```xml
<?xml version="1.0" encoding="utf-8"?>
<executable>

  <category>Python Modules</category>
  <title>Python Surface ICP Registration</title>
  <description>
  Performs registration of the input surface onto a target surface using
  on the Iterative Closest Point algorithm.
  </description>
  <version>1.0</version>
  <documentation-url></documentation-url>
  <license></license>
  <contributor>Luca Antiga and Daniel Blezek</contributor>

  <parameters>
    <label>Surface ICP Registration Parameters</label>
    <description>Parameters for surface registration</description>

    <string-enumeration>
      <name>landmarkTransformMode</name>
      <longflag>landmarkTransformMode</longflag>
    ...

    def Execute (inputSurface, targetSurface, outputSurface, 
    landmarkTransformMode="RigidBody", meanDistanceMode="RMS", 
    maximumNumberOfIterations=50, maximumNumberOfLandmarks=200, 
    startByMatchingCentroids=False, checkMeanDistance=False, 
    maximumMeanDistance=0.01):
      Slicer = __import__("Slicer")
      slicer = Slicer.slicer
      scene = slicer.MRMLScene
      inputSurface = scene.GetNodeByID(inputSurface)
      targetSurface = scene.GetNodeByID(targetSurface)
      outputSurface = scene.GetNodeByID(outputSurface)

      icpTransform = slicer.vtkIterativeClosestPointTransform()
      icpTransform.SetSource(inputSurface.GetPolyData())
      icpTransform.SetTarget(targetSurface.GetPolyData())
      ...
```
Numpy command-line (XML) modules

See Demian’s slides
Running a plugin from Python

- Command line modules calling any registered command line module (not necessarily another Python module)

```python
>>> import Slicer
>>> from Slicer import slicer
>>> volume1 = slicer.MRMLScene.GetNodeByID("vtkMRMLVolumeNode1")
>>> volume2 = slicer.MRMLScene.GetNodeByID("vtkMRMLVolumeNode2")
>>> plugin = Slicer.Plugin("Subtract Images")
>>> plugin.Execute(volume1.GetID(),volume2.GetID())
```

- This is an easy way of having ITK functionality available
- Alternative way: wrap ITK classes in vtkITK classes (or create VTK classes that contain an ITK pipeline) and instantiate them directly in a Python module
Scripted modules in Python

- Python CLI modules, like CLI modules in general, are not interactive: they only respond upon pushing “Apply” and cannot provide custom interaction, dynamic GUI updates and additional observations in general

- Scripted modules allow to do that (at the price of more coding)
Scripted modules in Python

- Example: PythonGADScriptedModule

```python
from SlicerScriptedModule import ScriptedModuleGUI
from Slicer import slicer

class PythonGADScriptedModuleGUI(ScriptedModuleGUI):
    def __init__(self):
        ...

    def AddGUIObservers(self):
        ...

    def ProcessGUIEvents(self, caller, event):
        ...

    def UpdateGUI(self):
        ...

    def BuildGUI(self):
        ...
```
Scripted modules in Python

- Example: Adding an observer to an existing Slicer object instance (e.g. for placing custom fiducials, ...)

```python
from SlicerScriptedModule import ScriptedModuleGUI
from Slicer import slicer

class PythonGADScriptedModuleGUI(ScriptedModuleGUI):

    def __init__(self):
        ...

    def AddGUIObservers(self):
        interactor = slicer.ApplicationGUI.GetRenderWindowInteractor()

    def TestCallback(self):
        print "I’m the callback!"
```