The following are sample Seg3D python code snippets that can be run in the python console:

```
# Threshold tool doesn't need to be open in GUI to run
# thresholding as a python function.
# Assumes a data layer with ID layer_0 is already open.
result = threshold(layerid='layer_0', lower_threshold=56360,
upper_threshold=24248)

# Open the Threshold tool.
id = opentool(tooltype='thresholdtool')

# Close the Threshold tool.
closetool(toolid=id)

# Run the Neighborhood Connected filter (does not have to be
# open in the GUI).
# Assumes a data layer with ID layer_0 is already open.
# Need at least one seed.
result = neighborhoodconnectedfilter(layerid='layer_0', seeds=
[[-0.24,20.3,57.2],[35.3,18.1,57.3],[18.7,25.4,57.3]])
print(result)
```

Seg3D can be manipulated through state variables using the **get** and **set** functions. Look for the variable names in the Controller Window under the State Variables tab.

```
# Switch the first view window (there are 6 view windows in total) to
the Sagittal slice plane.
# Returns bool value.
result = set(stateid='viewer0::view_mode', value='Sagittal')
```

Tool IDs can be obtained either from using the **opentool** method as shown above, or by looking them up in the Controller Window under the State Variables tab. Tool variables can be used to batch process slices in a given layer.

```
# Get the list of vertices making up a polyline in the Polyline or
Speedline tools.
current_vertices=get(stateid='speedlinetool_0::vertices')

# Use polyline list
polyline(target='layer_2', slice_type=0,
slice_number=93,vertices=current_vertices,erase=False)
polyline(target='layer_2', slice_type=0,
slice_number=94,vertices=current_vertices,erase=False)
```

Python scripts can be run in the console using the **open** and **exec** commands, for example:

```
exec(open('/home/user/mypythonscript.py').read())
```

A more complex example involves waiting for a filter to finish before exporting the results, in this case, mask layers. It is necessary to check if data is available in the layer using the ID returned by the filter (a layer has 4 states: creating, processing, in use and available).

In this case, a condition variable blocks until the predicate (lambda function that checks the layer state) evaluates to True.

```
import os
import threading

class MyThread(threading.Thread):
  def __init__(self, layerID, timeout=2.0, max_iterations=1000):
    self.layerID = layerID
    self.outputDir = outputDir
    self.TIMEOUT = timeout
    self.MAX_ITERATIONS = max_iterations
    self.condition = threading.Condition()
    threading.Thread.__init__(self)


  def run(self):
    stateIDData = self.layerID + "::data"
    layerStatus = get(stateid=stateIDData)
    print(self.layerID, " ", layerStatus)

    with self.condition:
      self.condition.wait_for(lambda: "available" == get
(stateid=stateIDData), timeout=self.TIMEOUT)

    print("Layer {} done processing".format(self.layerID))


def transformExportNRRD(filesnames, outputDir):
  if not os.path.exists(outputDir):
    raise ValueError("Path %s does not exist." % outputDir)

  layers = []
  for f in files:
    if not os.path.exists(f):
      raise ValueError("Path {} does not exist.".format(f))
    importedFile = importlayer(filename=f, importer='[Matlab
Importer]', mode='label_mask')
    if len(importedFile) < 1:
      raise Exception("importlayer failed")
    layers.append(importedFile[0])
  transformedLayers = transform(layerids=layers, origin=[-128.5,
-109.5, -108.5], spacing=[1, 1, 1])
  print(transformedLayers)

  threads = []

  # make sure data is available in layers
  for l in transformedLayers:
    thread = MyThread(l, 0.5, 4)
    thread.start()
    threads.append(thread)

  for t in threads:
    t.join()

  exportsegmentation(layers=transformedLayers, file_path=outputDir,
exporter='[NRRD Exporter]', extension='.nrrd', mode='single_mask')
```