Introduction to the Remote Sensing and GIS Software Library

(RSGISLib)

Pete Bunting
Aberystwyth University
Earth Observation and Ecosystem Dynamics Group
Department of Geography and Earth Sciences
pfb@aber.ac.uk
This work (including scripts) is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ With the exception of PALSAR images which are copyright © JAXA and Landsat images which are copyright © USGS.
## Contents

1 Introduction ........................................ 6  
   1.1 Software ........................................ 6  
      1.1.1 GDAL ........................................ 7  
      1.1.2 RSGISLib ................................... 7  
      1.1.3 KEA file format ............................. 8  
      1.1.4 TuiView ..................................... 8  
      1.1.5 RIOS ......................................... 8  
      1.1.6 Python ...................................... 9  
      1.1.7 scikit-learn ................................. 9  
   1.2 Computing Environment ........................... 10  
   1.3 Software Installation ............................ 10  
      1.3.1 MacOS and Linux ............................ 10  
      1.3.2 Windows .................................... 13  

2 Getting Help / Documentation ....................... 15  
   2.1 Online Documentation ............................ 15  
   2.2 Blog ............................................ 15  
   2.3 Mailing List .................................... 15  
   2.4 Code Repository ................................. 16  

3 Getting Started .................................... 17  
   3.1 Image Band Maths ............................... 18  
      3.1.1 Tidy up your files ......................... 20  
   3.2 Batch Processing ............................... 20  
      3.2.1 Using multiple cores ...................... 28  
   3.3 Make a command line tool ..................... 29  
   3.4 Tidy up your files ............................. 31  

4 Data Pre-processing ................................ 32  
   4.1 Image Mosaicking ............................... 32  
   4.2 Re-projecting / Re-sampling images ........... 33  
   4.3 Create Valid Data Extent ..................... 34  
   4.4 Raster-to-Vector ............................... 35
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.1 Vector-to-Raster</td>
<td>35</td>
</tr>
<tr>
<td>4.5 Image Spatial Subsetting</td>
<td>36</td>
</tr>
<tr>
<td>4.6 Image Masking</td>
<td>37</td>
</tr>
<tr>
<td>4.7 Image Band Subsetting</td>
<td>38</td>
</tr>
<tr>
<td>4.8 Apply Lee Filter</td>
<td>38</td>
</tr>
<tr>
<td>4.9 Band Maths: Power – dBs</td>
<td>39</td>
</tr>
<tr>
<td>4.10 Band Maths: NDVI &amp; WBI</td>
<td>40</td>
</tr>
<tr>
<td>4.11 Putting it all together</td>
<td>40</td>
</tr>
<tr>
<td>4.12 Tidy up your files</td>
<td>40</td>
</tr>
<tr>
<td>5 Image Classification</td>
<td>42</td>
</tr>
<tr>
<td>5.1 Pixel Based</td>
<td>43</td>
</tr>
<tr>
<td>5.2 Object Based</td>
<td>46</td>
</tr>
<tr>
<td>5.3 Compare the Classifications</td>
<td>50</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>51</td>
</tr>
<tr>
<td>A A brief introduction to the UNIX terminal</td>
<td>53</td>
</tr>
<tr>
<td>A.1 Working with the file system</td>
<td>54</td>
</tr>
<tr>
<td>A.1.1 Listing your current directory</td>
<td>54</td>
</tr>
<tr>
<td>A.1.2 Where am I?</td>
<td>54</td>
</tr>
<tr>
<td>A.1.3 Moving Directories</td>
<td>55</td>
</tr>
<tr>
<td>A.1.4 Creating a new directory</td>
<td>56</td>
</tr>
<tr>
<td>A.1.5 Case Sensitive</td>
<td>57</td>
</tr>
<tr>
<td>A.1.6 Copying Files and Directories</td>
<td>57</td>
</tr>
<tr>
<td>A.1.7 Moving Files and Directories</td>
<td>58</td>
</tr>
<tr>
<td>A.1.8 Removing a Directory</td>
<td>58</td>
</tr>
<tr>
<td>A.1.9 Decompressing files</td>
<td>59</td>
</tr>
<tr>
<td>A.2 Shell Scripts</td>
<td>59</td>
</tr>
<tr>
<td>A.3 Conclusions</td>
<td>60</td>
</tr>
<tr>
<td>B A brief introduction to Python</td>
<td>61</td>
</tr>
<tr>
<td>B.1 Starting Python</td>
<td>61</td>
</tr>
<tr>
<td>B.1.1 Indentation</td>
<td>61</td>
</tr>
<tr>
<td>B.1.2 Keywords</td>
<td>62</td>
</tr>
<tr>
<td>B.1.3 File Naming</td>
<td>62</td>
</tr>
<tr>
<td>B.1.4 Case Sensitivity</td>
<td>62</td>
</tr>
<tr>
<td>B.1.5 File paths in examples</td>
<td>63</td>
</tr>
<tr>
<td>B.1.6 Independent Development of Scripts</td>
<td>63</td>
</tr>
<tr>
<td>B.1.7 Importing Modules</td>
<td>64</td>
</tr>
<tr>
<td>B.1.8 Getting Help</td>
<td>64</td>
</tr>
<tr>
<td>B.2 Hello World Script</td>
<td>65</td>
</tr>
<tr>
<td>B.3 Comments</td>
<td>65</td>
</tr>
<tr>
<td>B.4 Variables</td>
<td>65</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>B.4.1 Numbers</td>
<td>66</td>
</tr>
<tr>
<td>B.4.2 Boolean</td>
<td>66</td>
</tr>
<tr>
<td>B.4.3 Text (Strings)</td>
<td>67</td>
</tr>
<tr>
<td>B.4.4 Example using Variables</td>
<td>67</td>
</tr>
<tr>
<td>B.5 Lists</td>
<td>70</td>
</tr>
<tr>
<td>B.5.1 List Examples</td>
<td>71</td>
</tr>
<tr>
<td>B.5.2 n-dimensional list</td>
<td>72</td>
</tr>
<tr>
<td>B.6 IF-ELSE Statements</td>
<td>72</td>
</tr>
<tr>
<td>B.6.1 Logic Statements</td>
<td>73</td>
</tr>
<tr>
<td>B.7 Looping</td>
<td>73</td>
</tr>
<tr>
<td>B.7.1 while Loop</td>
<td>74</td>
</tr>
<tr>
<td>B.7.2 for Loop</td>
<td>74</td>
</tr>
<tr>
<td>B.8 Conclusions</td>
<td>75</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This course aims to provide an introduction to the Remote Sensing and GIS Library (RSGISLib; Bunting et al., 2014) developed by Pete Bunting (pfb@aber.ac.uk) and Dan Clewley (dac@pml.ac.uk), and the stack of open source software commonly used in combination with it.

1.1 Software

In addition to RSGISLib, this worksheet makes use of the open source Geospatial Data Abstraction Library (GDAL; http://www.gdal.org), the TuiView image viewer (http://www.tuiview.org) the Raster Input/Output Simplification library (RIOS) library (http://www.rioshome.org) and scikit-learn (http://scikit-learn.org) machine learning library. These software are freely available and can be used on a wide variety of computing platforms from desktops to large scale high performance computing (HPC) systems and cloud systems such as Amazon or Google. Together they form a system for performing spatial image analysis, primarily derived from remote sensing imagery (Clewley et al., 2014).

1. Geospatial Data Abstraction Library (GDAL; http://www.gdal.org)
2. The Remote Sensing and GIS software library (RSGISLib; http://www.rsgislib.org)
3. KEA file format (http://www.kealib.org)
4. TuiView (http://www.tuiview.org)
5. Python (http://www.python.org)
6. RIOS (http://www.rioshome.org)
7. scikit-learn (http://scikit-learn.org)
CHAPTER 1. INTRODUCTION

1.1.1 GDAL

At its core the Geospatial Data Abstraction Library (GDAL; [http://www.gdal.org](http://www.gdal.org)) provides software to convert between many image file format but it has been extended to provide a set of utilities for processing image data. The most useful utilities are:

- `gdal_translate` – Copy a raster file, with control of output format.
- `gdalwarp` – Warp an image into a new coordinate system.
- `gdaladdo` – Add overviews to a file.
- `gdalbuildvrt` – Build a Virtual Raster (VRT) from a list of datasets.
- `gdal_contour` – Contours from DEM.
- `gdaldem` – Tools to analyse and visualise DEMs.
- `gdal_merge.py` – Build a quick mosaic from a set of images.
- `gdal_rasterize` – Rasterise vectors into raster file.
- `gdal_proximity.py` – Compute a raster proximity map.
- `gdal_polygonize.py` – Generate polygons from raster.
- `gdal_sieve.py` – Raster Sieve filter.
- `gdal_fillnodata.py` – Interpolate to fill no-data regions.
- `gdalmanage` – Identify, copy, rename and delete raster.
- `gdalcompare.py` – Compare two images and report on differences.

1.1.2 RSGISLib

The Remote Sensing and GIS Software Library (RSGISLib; [Bunting et al. 2014]) is primarily developed by Pete Bunting and Dan Clewley, it was originally designed to just provide the functionality we required for our own research, where it wasn’t available in existing software packages. However, RSGISLib has evolved into a set of Python modules providing a wide range of functionality which is scriptable, creating a flexible system for data analysis and batch processing. The available module are:

- Image Calibration
- Classification
- Elevation
- Image Calculations
CHAPTER 1. INTRODUCTION

- Image Filtering
- Image Morphology
- Image Registration
- Image Utilities
- Histogram Cube
- Raster GIS
- Image Segmentation
- Tools
- Vector Utilities
- Zonal Statistics

1.1.3 KEA file format

The KEA file format developed by Bunting and Gillingham (2013) and named after the New Zealand bird (Kea) is a HDF5 based image file format with a GDAL driver. Therefore, the format can be used in any software using GDAL, provided the KEA library is available. It offers support for large raster attribute tables and uses zlib based compression to provide small file sizes. The development was funded and supported by Landcare Research, New Zealand.

1.1.4 TuiView

TuiView is an open source viewer for remote sensing data, named after the New Zealand bird (Tui). Although primarily for viewing raster data but it will also display vectors. One of the main advantages of TuiView is it has a lot of functionality for viewing and manipulating Raster Attribute Tables (RAT), which the object-based classification within RSGISLib is build on top of. TuiView is cross platform and provides a consistent interface across Windows, Linux and Mac platforms. It is capable of handling very large datasets, providing overviews are generated.

1.1.5 RIOS

The Raster Input and Output (I/O) simplification (RIOS; Gillingham and Flood, 2013, http://www.rioshome.org) library is a set of Python modules which makes it easier to write raster processing code in Python. Built on top of GDAL, it handles the details of opening and closing files, checking alignment of projections and raster grid, stepping through the raster in small blocks, etc., allowing the programmer to concentrate on
implementing the solution to the problem rather than on how to access the raster data
and detail with the spatial header. It also offers functions for reading and writing Raster
Attribute Tables.

1.1.6 Python

Python is a high level scripting language which is interpreted, interactive and object-
oriented. A key attribute of Python is its clear and understandable syntax. Python is
cross-platform with support for Windows, Linux, MacOS and most other UNIX plat-
forms. In addition, many libraries (e.g., purpose built and external C++ libraries)
are available to Python and it has become a very popular language for many applica-
tions.

To fully utilise the Python bindings of RSGISLib, and the other tools described here
knowledge of Python syntax is required. For this course a basic knowledge of Python
will be helpful but is not required, see ‘A brief introduction to Python’ at the end of
this worksheet for some pointers.

For a complete beginners guide to Python the codecademy.com Python tutorials are
recommend, (http://www.codecademy.com/tracks/python). It is recommend that the
following tutorials are covered:

• ‘Python Syntax’
• ‘Strings & Console Output’
• ‘Date and Time’
• ‘Conditionals & Flow Control’
• ‘Functions’
• ‘Python Lists and Dictionaries’
• ‘Lists and Functions’
• ‘Loops’
• ‘File Input / Output’

1.1.7 scikit-learn

Scikit-learn (Pedregosa et al., 2011) is a library of machine learning algorithms acces-
sible from within Python. Scikit-learn provides functionality under the following head-
ings.

• Classification


- Regression
- Clustering
- Dimensionality reduction
- Model selection
- Preprocessing

RSGISLib uses scikit-learn to solve a number of classification and clustering problems.

### 1.2 Computing Environment

The software used for this course are accessed from the command line. Whilst GUI tools are easier to learn, if you are not familiar with the command line, the command line is more powerful for batch processing and/or creating processing chains. Although the software listed is cross-platform (e.g., Windows, Linux, OS X), RSGISLib is currently only tested and extensively used under UNIX-like operating systems (MacOS and Linux). Building tools using the UNIX environment scales well to HPCs, the majority of which run a UNIX-like operating system.

### 1.3 Software Installation

Binary downloads are made available through the conda package management system ([http://conda.pydata.org](http://conda.pydata.org)), which allows multiple ‘environments’ to be created. With conda, you can create, export, list, remove, and update environments that have different versions of Python and/or packages installed in them. Switching or moving between environments is called activating an environment. You can even share an environment file with a coworker. See the Conda documentation: [http://conda.pydata.org/docs/using/envs.html](http://conda.pydata.org/docs/using/envs.html)

#### 1.3.1 MacOS and Linux

**Install Conda**

To install the software you need for this worksheet you need to first install conda on either MacOS or Linux, via using the miniconda installation package ([http://conda.pydata.org/miniconda.html](http://conda.pydata.org/miniconda.html)) – you require the Python 3.5, 64-bit version. You will have downloaded either:

- Miniconda3-latest-MacOSX-x86_64.sh – MacOS
CHAPTER 1. INTRODUCTION

- Miniconda3-latest-Linux-x86_64.sh – Linux

If you already have conda installed then do not install it again but go to section 1.3.1, you can check by running:

```bash
conda --version
```

If a version is printed out to the terminal then it is installed, otherwise an error will be provided stating that the `conda` command is not available.

Once downloaded, open a Terminal window (e.g., Figure 1.1) and navigate to where the downloaded file is stored on your system (e.g., `cd Downloads`). If you are not familiar with a UNIX terminal see Appendix A.

![Figure 1.1: A terminal window.](image)

Once you have navigated to file, you will need to change permissions so it can be executed:

```bash
chmod a+x ./Miniconda3-latest-MacOSX-x86_64.sh
```

and then run is as shown below, by typing the name of the script:

```bash
./Miniconda3-latest-MacOSX-x86_64.sh
```
It will ask a number of questions:

1. Do you want to install Miniconda – **Press Enter**
2. It’ll show you the license, press `spacebar` to scroll through. Type `Yes` to confirm you agree to the license.
3. Confirm the installation location, the default is your home directory and it is suggested that you use this. – **Press Enter**
4. It will ask if it can append the miniconda installation to your `PATH`. The default is Yes, therefore – **Press Enter**

Conda should now be installed, close all your Terminal windows and open a new one, you can test conda is working by running the following command:

```
conda --version
```

### Setup Conda Environment

To install the package you require for this worksheet, you should now run the following commands:

First, setup an environment (`osgeo`) for this worksheet:

```
conda create --name osgeo-env-v1 python=3.5
```

It will install a set of default packages, just agree (press `y`). To activate the environment you run the following command, you will need to do that each time you open a new Terminal.

```
source activate osgeo-env-v1
```

You will be able to tell which environment you are using as to the left of your command prompt it will name it, `(osgeo-env-v1)`, in brackets; for example:

```
(osgeo-env-v1) Petes-MacBook-Pro:Downloads pete$ 
```

Now you have an environment, you are ready to install the software. It is good practice to create a new environment for each project you undertaken, then you can ensure that the appropriate version of each package is maintained throughout the project – different projects can require different versions of packages.

Run the following to install the software you require:
When these are installed they will also download the dependencies those software require, therefore agree to the messages requesting that other packages are also installed.

If there are any problems with your installation (e.g., a missing package) then this can often be resolved by updating your system:

```
conda update -c conda-forge --all
```

To check that the software has been installed, run the following:

```
rgis-config --version
```

You should have at least version 3.4. Also, run tuiview, which will start the viewer (Figure 1.2).

```
tuiview
```

### 1.3.2 Windows

While you can install RSGISLib and the other tools on Windows, RSGISLib is primarily tested on Linux and MacOS therefore we recommend using one these of systems. On Windows 10 you can install Linux alongside Windows, please note this does not provide a graphic interface so TuiView will need to be installed, via Conda, under Windows rather than through the Linux installation. Alternatively, you can install a Linux within Windows using a virtual machine. VirtualBox is a free and open source package which enables this but VMWare is a commercial alternative.
Figure 1.2: The TuiView image viewer.
Chapter 2

Getting Help / Documentation

2.1 Online Documentation

Online documentation for the latest version of RSGISLib is provided on the website (http://www.rsgislib.org) and should be your first point of call. When using RSGISLib you’ll find that this website provides a crucial reference for the available functions.

There are also other tutorials on the website under http://www.rsgislib.org/documentation.html.

2.2 Blog

You will also have the online blog https://spectraldifferences.wordpress.com a useful source of examples of specific problems (e.g., processing PALSAR data https://spectraldifferences.wordpress.com/tag/palsar/).

2.3 Mailing List

We have a mailing list https://groups.google.com/forum/#!forum/rsgislib-support where you can communicate with others using RSGISLib and associated tools if you have specific questions or think there is bug or problem with the software. We do our best to answer emails on this list in a prompt manner. To post (rsgislib-support@googlegroups.com) on this mailing list you first need to register.
2.4 Code Repository

If you would like to see the RSGISLib source code or submit to our issues list this is done through the bitbucket service, [https://bitbucket.org/petebunting/rsgislib](https://bitbucket.org/petebunting/rsgislib).
Chapter 3

Getting Started

To get started we will have a look at the files provided within the Scripts and Datasets directories. The Scripts directory has all the scripts used within the course, with the order they are referred to in this document specified with the preceding number. The datasets directory includes all the data required for this course (Table 3.1).

Table 3.1: Datasets used within this course

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALOS PALSAR-2</td>
<td>JAXA Mosaic data from 2015</td>
</tr>
<tr>
<td>Landsat-8</td>
<td>USGS Landsat-8 imagery, two scenes which have been atmospherically corrected</td>
</tr>
<tr>
<td></td>
<td>to surface reflectance and merged.</td>
</tr>
<tr>
<td>Training data</td>
<td>These were manually drawn with reference to Google Earth imagery using QGIS</td>
</tr>
<tr>
<td>(ESRI Shapefiles)</td>
<td></td>
</tr>
</tbody>
</table>

To run RSGISLib we need to create a python script (a text file with the file extension .py) in which you just list the RSGISLib functions you wish to run. You can think of an RSGISLib function like a dialog box for running a command in a GUI based application (e.g., ENVI / Erdas Imagine / ArcGIS) where you provide all the options required (e.g., input and output files). However, as you can list a series of commands which all run one after another you can streamline your data processing chains ensuring that all files are run with the same options and by looping through a number of input files easily batch process your data.

If you haven’t used Python before I would recommend you look through Appendix B which provides a brief introduction to the syntax. However, for the majority of this course only relatively basic Python syntax will be used.

We will first create a working directory on your system, for example on your Desktop, RSGISLibTraining where we will do our work:
3.1 Image Band Maths

In this example we will calculate a three band composite for a PALSAR-2 scene. The output image will have three image bands:

- HH Power
- HV Power
- HH/HV

Before we start, create a directory for this exercise your working space and copy the file N22E089_15_MOS_F02DAR.tar.gz into that directory.

Extract N22E089_15_MOS_F02DAR.tar.gz using the tar command:

```bash
tar -zxf ./N22E089_15_MOS_F02DAR.tar.gz
```

To completed the processing steps I find it helpful to first write out the steps in coding comments, breaking the problem down. The steps for this problem are the following:

```python
#/usr/bin/env python
# Import python modules
# Calculate Power for HH image
# Calculate Power for HV image
# Calculate HH / HV
# Stack images into a single band
# Calculate overview pyramids and image statistics to make visualisation faster.
```

Producing the solution to those is now a case of ‘filling out the form’. In this case, those steps are filled using the following rsgislib functions:
CHAPTER 3. GETTING STARTED

#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imagecalc
from rsgislib import imageutils

# Calculate Power for HH image
hhImg = 'N22E089_15_sl_HH_F02DAR'
hhImgPow = 'N22E089_15_sl_HH_F02DAR_pow.kea'
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HH', hhImg, 1))
mathExp = 'HH>0?10^(2*log10(HH) - 8.3):0.0'
imagecalc.bandMath(hhImgPow, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Calculate Power for HV image
hvImg = 'N22E089_15_sl_HV_F02DAR'
hvImgPow = 'N22E089_15_sl_HV_F02DAR_pow.kea'
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HV', hvImg, 1))
mathExp = 'HV>0?10^(2*log10(HV) - 8.3):0.0'
imagecalc.bandMath(hvImgPow, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Calculate HH / HV
hhhvImg = 'N22E089_15_sl_HHHV_F02DAR_pow.kea'
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HH', hhImgPow, 1))
bandDefns.append(imagecalc.BandDefn('HV', hvImgPow, 1))
mathExp = 'HV>0?HH/HV:0.0'
imagecalc.bandMath(hhhvImg, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Stack images into a single band
imageList = [hhImgPow, hvImgPow, hhhvImg]
bandNamesList = ['HH','HV', 'HH/HV']
outputImageStack = 'N22E089_15_sl_F02DAR_powstack.kea'
imageutils.stackImageBands(imageList, bandNamesList, outputImageStack, None, 0, 'KEA', rsgislib.TYPE_32FLOAT)

# Calculate overview pyramids and image statistics to make visualisation faster.
imageutils.popImageStats(outputImageStack, usenodataval=True, nodataval=0, calcpyramids=True)

The script file 01_BandMaths.py has this code, copy it into your working directory Exercise1 and run it.

python 01_BandMaths.py

View the end result using the Tuiview image viewer Figure 1.2. You start Tuiview use the Terminal:
You can provide the filename of the file you wish to open on the terminal as well if you wish (this makes it particularly useful for viewing imagery on a remote server over ‘ssh -X’).

tuiview N22E089_15_s1_F02DAR_powstack.kea

3.1.1 Tidy up your files

You can now delete the directory Exercise1 to save disk space.

3.2 Batch Processing

If you want to apply this process to a large number of image datasets we’d need to add some steps to our processing chain:

```
#/usr/bin/env python
# Import python modules
# Extract tar.gz file
# Find the HH and HV images.
# Calculate Power for HH image
# Calculate Power for HV image
# Calculate HH / HV
# Stack images into a single band
# Calculate overview pyramids and image statistics to make visualisation faster.
# Clean up so only the stack remains.
```

To do this it is useful to define a function which does the work which can be called multiple times, once for each input file:

```
#/usr/bin/env python
# Import python modules

def createPALSARStack(inputTAR, outputStackImg, tmpDir):
```
CHAPTER 3. GETTING STARTED

# Extract tar.gz file
# Find the HH and HV images.
# Calculate Power for HH image
# Calculate Power for HV image
# Calculate HH / HV
# Stack images into a single band
# Calculate overview pyramids and image statistics to make visualisation faster.
# Clean up so only the stack remains.

createPALSARStack('N22E088_15_MOS_F02DAR.tar.gz', 'N22E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N22E089_15_MOS_F02DAR.tar.gz', 'N22E089_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E088_15_MOS_F02DAR.tar.gz', 'N23E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E089_15_MOS_F02DAR.tar.gz', 'N23E089_15_MOS_F02DAR_Stack.kea', './tmp')

02A_PALSARStack.py (below) provides the code which will create a temporary working directory, extract the TAR file and find the HH and HV files before deleting the temporary working directory to clean up the file.

```python
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imagecalc
from rsgislib import imageutils
import os
import os.path
import subprocess
import shutil
import glob

def createPALSARStack(inputTAR, outputStackImg, tmpDir):
    # Make sure that the inputs use absolute paths as using a working directory
    inputTAR = os.path.abspath(inputTAR)
    outputStackImg = os.path.abspath(outputStackImg)
    tmpDir = os.path.abspath(tmpDir)

    # Check file input file and tmp directory are present.
    if not os.path.exists(tmpDir):
        raise rsgislib.RSGISPyException('Tmp directory is not present.')

    if not os.path.exists(inputTAR):
        raise rsgislib.RSGISPyException('Input tar file is not present.')

    # Get the rsgis utilities object
```
rsgisUtils = rsgislib.RSGISPyUtils()

# Get a unique id for processing
uidStr = rsgisUtils.uidGenerator()

# Create a working directory
workDIR = os.path.join(tmpDir, uidStr)
if not os.path.exists(workDIR):
    os.makedirs(workDIR)
# Save current working path
cPWD = os.getcwd()
# Move into that working directory.
os.chdir(workDIR)

# Extract tar.gz file - using the terminal commands.
inputTAR = 'input.tar.gz'
cmd = 'tar -zxf ' + inputTAR
print(cmd)
try:
    subprocess.call(cmd, shell=True)
except OSError as e:
    raise rsgislib.RSGISPyException('Could not execute command: ' + cmd)

# Find the HH and HV images.
hhImg = ''
hvImg = ''

hhFiles = glob.glob(os.path.join(workDIR, '*_sl_HH_F02DAR'))
hvFiles = glob.glob(os.path.join(workDIR, '*_sl_HV_F02DAR'))

if len(hhFiles) == 1:
    hhImg = hhFiles[0]
else:
    raise rsgislib.RSGISPyException('Could not find HH file')

if len(hvFiles) == 1:
    hvImg = hvFiles[0]
else:
    raise rsgislib.RSGISPyException('Could not find HV file')

print("HH File: ", hhImg)
print("HV File: ", hvImg)

# Calculate Power for HH image
# Calculate Power for HV image
# Calculate HH / HV
# Stack images into a single band
# Calculate overview pyramids and image statistics to make visualisation faster.

# Clean up by deleting the working directory
os.chdir(cPWD) # Move back to starting directory before delete
CHAPTER 3.  GETTING STARTED

```
shutil.rmtree(workDIR)
createPALSARStack('N22E088_15_MOS_F02DAR.tar.gz', 'N22E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N22E089_15_MOS_F02DAR.tar.gz', 'N22E089_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E088_15_MOS_F02DAR.tar.gz', 'N23E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E089_15_MOS_F02DAR.tar.gz', 'N23E089_15_MOS_F02DAR_Stack.kea', './tmp')
```

Copy the all the tar.gz PALSAR-2 files and the 02A_PALSARStack.py into your working directory under a directory Exercise2 and then run the 02A_PALSARStack.py script.

```
python 02A_PALSARStack.py
```

You should get an output similar to that below:

```
(osgeo) WS0978:Exercise2 pete$ python 02A_PALSARStack.py
tar -zxf /Users/pete/Desktop/RSGISLibTraining/Exercise2/N22E088_15_MOS_F02DAR.tar.gz
HH File: /Users/pete/Desktop/RSGISLibTraining/Exercise2/tmp/e81517/N22E088_15_sl_HH_F02DAR
HV File: /Users/pete/Desktop/RSGISLibTraining/Exercise2/tmp/e81517/N22E088_15_sl_HV_F02DAR
tar -zxf /Users/pete/Desktop/RSGISLibTraining/Exercise2/N22E089_15_MOS_F02DAR.tar.gz
HH File: /Users/pete/Desktop/RSGISLibTraining/Exercise2/tmp/8e9c34/N22E089_15_sl_HH_F02DAR
HV File: /Users/pete/Desktop/RSGISLibTraining/Exercise2/tmp/8e9c34/N22E089_15_sl_HV_F02DAR
tar -zxf /Users/pete/Desktop/RSGISLibTraining/Exercise2/N23E088_15_MOS_F02DAR.tar.gz
HH File: /Users/pete/Desktop/RSGISLibTraining/Exercise2/tmp/5b4716/N23E088_15_sl_HH_F02DAR
HV File: /Users/pete/Desktop/RSGISLibTraining/Exercise2/tmp/5b4716/N23E088_15_sl_HV_F02DAR
tar -zxf /Users/pete/Desktop/RSGISLibTraining/Exercise2/N23E089_15_MOS_F02DAR.tar.gz
HH File: /Users/pete/Desktop/RSGISLibTraining/Exercise2/tmp/01736a/N23E089_15_sl_HH_F02DAR
HV File: /Users/pete/Desktop/RSGISLibTraining/Exercise2/tmp/01736a/N23E089_15_sl_HV_F02DAR
```

Try putting in 'print' statements into the script to understand the processing steps.

We can now combine the two scripts 01_BandMaths.py and 02A_PALSARStack.py to run the whole process creating 02B_PALSARStack.py:

```
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imagecalc
from rsgislib import imageutils

import os
import os.path
import subprocess
import shutil
import glob

def createPALSARStack(inputTAR, outputStackImg, tmpDir):
    # Make sure that the inputs use absolute paths as using a working directory
```
inputTAR = os.path.abspath(inputTAR)
outputStackImg = os.path.abspath(outputStackImg)
tmpDir = os.path.abspath(tmpDir)

# Check file input file and tmp directory are present.
if not os.path.exists(tmpDir):
    raise rsgislib.RSGISPyException('Tmp directory is not present.')
if not os.path.exists(inputTAR):
    raise rsgislib.RSGISPyException('Input tar file is not present.')
# Get the rsgis utilities object
rsgisUtils = rsgislib.RSGISPyUtils()

# Get a unique id for processing
uidStr = rsgisUtils.uidGenerator()

# Create a working directory
workDIR = os.path.join(tmpDir, uidStr)
if not os.path.exists(workDIR):
    os.makedirs(workDIR)
# Save current working path
cPWD = os.getcwd()
# Move into that working directory.
os.chdir(workDIR)

# Extract tar.gz file - using the terminal commands.
cmd = 'tar -zxf ' + inputTAR
print(cmd)
try:
    subprocess.call(cmd, shell=True)
except OSError as e:
    raise rsgislib.RSGISPyException('Could not execute command: ' + cmd)

# Find the HH and HV images.
hhImg = ''
hvImg = ''

hhFiles = glob.glob(os.path.join(workDIR, '*_sl_HH_F02DAR'))
hvFiles = glob.glob(os.path.join(workDIR, '*_sl_HV_F02DAR'))

if len(hhFiles) == 1:
    hhImg = hhFiles[0]
else:
    raise rsgislib.RSGISPyException('Could not find HH file')
if len(hvFiles) == 1:
    hvImg = hvFiles[0]
else:
    raise rsgislib.RSGISPyException('Could not find HV file')

print("HH File: ", hhImg)
print("HV File: ", hvImg)

# Calculate Power for HH image
CHAPTER 3. GETTING STARTED

```python
hhImgPow = os.path.join(workDIR, uidStr+'_HH_Pow.kea')
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HH', hhImg, 1))
mathExp = 'HH>0?10^(2*log10(HH) - 8.3):0.0'
imagecalc.bandMath(hhImgPow, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Calculate Power for HV image
hvImgPow = os.path.join(workDIR, uidStr+'_HV_Pow.kea')
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HV', hvImg, 1))
mathExp = 'HV>0?10^(2*log10(HV) - 8.3):0.0'
imagecalc.bandMath(hvImgPow, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Calculate HH / HV
hhhvImg = os.path.join(workDIR, uidStr+'_HHHV_Pow.kea')
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HH', hhImgPow, 1))
bandDefns.append(imagecalc.BandDefn('HV', hvImgPow, 1))
mathExp = 'HV>0?HH/HV:0.0'
imagecalc.bandMath(hhhvImg, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Stack images into a single band
imageList = [hhImgPow, hvImgPow, hhhvImg]
bandNamesList = ['HH', 'HV', 'HH/HV']
imageutils.stackImageBands(imageList, bandNamesList, outputStackImg, None, 0, 'KEA', rsgislib.TYPE_32FLOAT)

# Calculate overview pyramids and image statistics to make visualisation faster.
imageutils.popImageStats(outputStackImg, usenodataval=True, nodataval=0, calcpyramids=True)

# Clean up by deleting the working directory
os.chdir(cPWD) # Move back to starting directory before delete
shutil.rmtree(workDIR)
```

Copy 02B_PALSARStack.py into your Exercise2 working directory and run it.

```bash
python 02B_PALSARStack.py
```
import os
import os.path
import subprocess
import shutil
import glob

def createPALSARStack(inputTAR, outputStackImg, tmpDir):
    # Make sure that the inputs use absolute paths as using a working directory
    inputTAR = os.path.abspath(inputTAR)
    outputStackImg = os.path.abspath(outputStackImg)
    tmpDir = os.path.abspath(tmpDir)

    # Check if input file and tmp directory are present.
    if not os.path.exists(tmpDir):
        raise rsgislib.RSGISPyException('Tmp directory is not present.')
    if not os.path.exists(inputTAR):
        raise rsgislib.RSGISPyException('Input tar file is not present.')
    
    # Get the rsgis utilities object
    rsgisUtils = rsgislib.RSGISPyUtils()

    # Get a unique id for processing
    uidStr = rsgisUtils.uidGenerator()

    # Create a working directory
    workDIR = os.path.join(tmpDir, uidStr)
    if not os.path.exists(workDIR): os.makedirs(workDIR)

    # Save current working path
    cPWD = os.getcwd()
    # Move into that working directory.
    os.chdir(workDIR)

    # Extract tar.gz file - using the terminal commands.
    cmd = 'tar -zxf ' + inputTAR
    print(cmd)
    try:
        subprocess.call(cmd, shell=True)
    except OSError as e:
        raise rsgislib.RSGISPyException('Could not execute command: ' + cmd)

    # Find the HH and HV images.
    hhImg = ''
    hvImg = ''
    hhFiles = glob.glob(os.path.join(workDIR, '*.sl_HH_F02DAR'))
    hvFiles = glob.glob(os.path.join(workDIR, '*.sl_HV_F02DAR'))
    if len(hhFiles) == 1:
        hhImg = hhFiles[0]
    else:
        raise rsgislib.RSGISPyException('Could not find HH file')
if len(hvFiles) == 1:
    hvImg = hvFiles[0]
else:
    raise rsgislib.RSGISPyException('Could not find HV file')

print("HH File: ", hhImg)
print("HV File: ", hvImg)

# Calculate Power for HH image
hhImgPow = os.path.join(workDIR, uidStr+'\_HH\_Pow.kea')
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HH', hhImg, 1))
mathExp = 'HH>0?10^(2*log10(HH) - 8.3):0.0'
imagecalc.bandMath(hhImgPow, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Calculate Power for HV image
hvImgPow = os.path.join(workDIR, uidStr+'\_HV\_Pow.kea')
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HV', hvImg, 1))
mathExp = 'HV>0?10^(2*log10(HV) - 8.3):0.0'
imagecalc.bandMath(hvImgPow, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Calculate HH / HV
hhhvImg = os.path.join(workDIR, uidStr+'\_HHHV\_Pow.kea')
bandDefns = []
bandDefns.append(imagecalc.BandDefn('HH', hhImgPow, 1))
bandDefns.append(imagecalc.BandDefn('HV', hvImgPow, 1))
mathExp = 'HV>0?HH/HV:0.0'
imagecalc.bandMath(hhhvImg, mathExp, 'KEA', rsgislib.TYPE_32FLOAT, bandDefns)

# Stack images into a single band
imageList = [hhImgPow, hvImgPow, hhhvImg]
bandNamesList = ['HH', 'HV', 'HH/HV']
imageutils.stackImageBands(imageList, bandNamesList, outputStackImg, None, 0, 'KEA', rsgislib.TYPE_32FLOAT)

# Calculate overview pyramids and image statistics to make visualisation faster.
imageutils.popImageStats(outputStackImg, usenodataval=True, nodataval=0, calcpyramids=True)

# Clean up by deleting the working directory
os.chdir(cPWD)  # Move back to starting directory before delete
shutil.rmtree(workDIR)
createPALSARStack('N22E088_15_MOS_F02DAR.tar.gz', 'N22E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N22E089_15_MOS_F02DAR.tar.gz', 'N22E089_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E088_15_MOS_F02DAR.tar.gz', 'N23E088_15_MOS_F02DAR_Stack.kea', './tmp')
createPALSARStack('N23E089_15_MOS_F02DAR.tar.gz', 'N23E089_15_MOS_F02DAR_Stack.kea', './tmp')

You can now load your images into tuiview
3.2.1 Using multiple cores

An extension for more advanced users is to use the Python multiprocessing module ([https://docs.python.org/3.5/library/multiprocessing.html](https://docs.python.org/3.5/library/multiprocessing.html)) to enable multi-core processing. We’ll use the Pool object which requires a function to do ‘the work’, where the function takes just a single parameter (which can be an array). **Please note, I would not recommend you run this script from within a virtual machine (i.e., VirtualBox) unless you have a lot of resources (including multiple cores) available to your virtual machine.**

Therefore, we will edit 02B_PALSARStack.py, creating 02C_PALSARStack.py so the three parameters are passed as an array:

```python
# Old functions
def createPALSARStack(inputTAR, outputStackImg, tmpDir):

# New function interface
def createPALSARStack(params):
    inputTAR = params[0]
    outputStackImg = params[1]
    tmpDir = params[2]

We then define the input parameters as an array of arrays, find the number of processing cores available (set to an integer >0 and < the total number of cores if you don’t want all to be used). A pool is then created and the list of parameters queued and run on the available cores.

```python
inputParams = []
inputParams.append(['N22E088_15_MOS_F02DAR.tar.gz', 'N22E088_15_MOS_F02DAR_Stack.kea', './tmp'])
inputParams.append(['N22E089_15_MOS_F02DAR.tar.gz', 'N22E089_15_MOS_F02DAR_Stack.kea', './tmp'])
inputParams.append(['N23E088_15_MOS_F02DAR.tar.gz', 'N23E088_15_MOS_F02DAR_Stack.kea', './tmp'])
inputParams.append(['N23E089_15_MOS_F02DAR.tar.gz', 'N23E089_15_MOS_F02DAR_Stack.kea', './tmp'])

numCores = multiprocessing.cpu_count() # find the number of cores available on the system.
with multiprocessing.Pool(numCores) as p:
    p.map(createPALSARStack, inputParams)
```

Look at the code in 02C_PALSARStack.py, try to understand what is going, and then copy the file into your Exercise2 removing the previously created stacked files and execute it as you did before the result will be the same as 02B_PALSARStack.py but will execute faster as multiple images will be processing at the same time on different cores.

```bash
python 02C_PALSARStack.py
```
3.3 Make a command line tool

Building on script 02B_PALSARStack.py the next step will be to make this tool parametrisable by the user from the terminal. Python has an easy to use module, *argparse* ([https://docs.python.org/3/library/argparse.html](https://docs.python.org/3/library/argparse.html)), for this purpose.

We will just change the bottom of the script to read the three parameters from the terminal:

```python
# Only run this code if it is called from the terminal (i.e., not from another python script)
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--input", required=True, type=str, help="""Input tar.gz file""")
    parser.add_argument("-o", "--output", required=True, type=str, help="""Output stacked KEA file""")
    parser.add_argument("-t", "--tmpath", required=True, type=str, help="""Temporary path which will be generated and removed during processing."""")

    args = parser.parse_args()

    createPALSARStack(args.input, args.output, args.tmpath)
```

You can now run this script from the terminal (from your Exercise2 directory) using the following command:

```bash
python 02D_PALSARStack.py -i N22E088_15_MOS_F02DAR.tar.gz -o N22E088_Stack.kea -t tmp
```

If you wanted your script to process a whole directory of tar.gz files then you can make the command line parser code as follows (02E_PALSARStack.py):

```python
# Only run this code if it is called from the terminal (i.e., not from another python script)
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--input", required=True, type=str, help="""Input directory containing the tar.gz files""")
    parser.add_argument("-o", "--output", required=True, type=str, help="""Output directory for the KEA files""")
    parser.add_argument("-t", "--tmpath", required=True, type=str, help="""Temporary path which will be generated and removed during processing."""")

    args = parser.parse_args()

    # Get all the tar.gz files.
    inputFiles = glob.glob(os.path.join(args.input, '*.tar.gz'))
```
CHAPTER 3. GETTING STARTED

# Loop through files.
for inFile in inputFiles:
    # Get basename
    basename = os.path.basename(inFile).split('.')[0]
    # Create name of output file.
    outputFile = os.path.join(args.output, basename+'_stack.kea')
    # Call function.
    createPALSARStack(inFile, outputFile, args.tmpath)

You can now run this script from the terminal (from your Exercise2 directory) using the following command:

```
python 02E_PALSARStack.py -i . -o . -t tmp
```

NOTE. '.' signifies the current directory in your terminal. Try running `ls .`.

Finally, we can merge 02E_PALSARStack.py with 02C_PALSARStack.py to allow multiple cores to be used.

```
# Only run this code if it is called from the terminal (i.e., not from another python script)
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--input", required=True, type=str, help="Input directory containing the tar.gz files")
    parser.add_argument("-o", "--output", required=True, type=str, help="Output directory for the KEA files")
    parser.add_argument("-t", "--tmpath", required=True, type=str, help="Temporary path which will be generated and removed during processing.")

    args = parser.parse_args()

    inputFiles = glob.glob(os.path.join(args.input, '*.tar.gz'))
    # Array for the function parameters.
    inputParams = []
    # Loop through files.
    for inFile in inputFiles:
        # Get basename
        basename = os.path.basename(inFile).split('.')[0]
        # Create name of output file.
        outputFile = os.path.join(args.output, basename+'_stack.kea')
        # Add parameters to array.
        inputParams.append([inFile, outputFile, args.tmpath])

    # find the number of cores available on the system.
    numCores = multiprocessing.cpu_count()
    # Create pool and run.
    with multiprocessing.Pool(numCores) as p:
        p.map(createPALSARStack, inputParams)
```
p.map(createPALSARStack, inputParams)

Try executing `02F_PALSARStack.py`, from your Exercise2 directory, and you find all the files processing. This script could be used to processing thousands of files if required using all the cores on your system.

`python 02F_PALSARStack.py -i . -o . -t tmp`

### 3.4 Tidy up your files

To tidy up your files and save space only the final stacked KEA files from your Exercise2 directory are needed for the next exercises, the rest can be deleted.
Chapter 4

Data Pre-processing

Stacking and converting the SAR data to power is part of the pre-processing ahead of doing something ‘useful’ with the image data. However, there are many other processes which can be required or helpful in preparing data for the application, in this case classification.

To start the analysis create a new directory (Exercise3) in your working space and copy the KEA image files outputted from the previous step into that directory.

```bash
cd Desktop/RSGISLibTraining
mkdir Exercise3
cd Exercise3
```

4.1 Image Mosaicking

The first process is to mosaic the four PALSAR-2 images to create a single image file, the code shown below.

```python
#/usr/bin/env python
# Import python modules
import rsgislib
from rsgislib import imageutils

# List of the files to be mosaicked.
inputList = ['N22E088_15_MOS_F02DAR_stack.kea',
            'N22E089_15_MOS_F02DAR_stack.kea',
            'N23E088_15_MOS_F02DAR_stack.kea',
            'N23E089_15_MOS_F02DAR_stack.kea']

outImage = './Sundarbans_15_MOS_F02DAR.kea'

imageutils.createImageMosaic(inputList, outImage, 0.0, 0.0, 1, 0,
```

CHAPTER 4. DATA PRE-PROCESSING

See the documentation [http://www.rsgislib.org/rsgislib_imageutils.html#rsgislib.imageutils.createImageMosaic] for explanation of the different inputs.

Please note, you can also use the glob module to automatically find the input files. For example, see script 03B_MosaicImages.py – remember to delete the previous output (Sundarbans_15_MOS_F02DAR.kea) before running this script.

```
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils
import glob

# Search for all files with the extension 'kea'
inputList = glob.glob('*.kea')
outImage = './Sundarbans_15_MOS_F02DAR.kea'
imageutils.createImageMosaic(inputList, outImage, 0.0, 0.0, 1, 0,
'KEA', rsgislib.TYPE_32FLOAT)
imageutils.popImageStats(outImage, usenodataval=True,
nodataval=0, calcpyramids=True)
```

4.2 Re-projecting / Re-sampling images

In the classification we are going to use PALSAR-2 and Landsat-8 imagery. However, the PALSAR-2 imagery is provided in lat/long WGS84 while the Landsat-8 imagery is provided in WSG84 UTM Zone 45. We are therefore going to re-project and resample the PALSAR-2 imagery to match the Landsat-8 imagery, this will also mean our coordinates will be in metres, which is more convenient.

```
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

inRefImg = 'Sundarbans_201511_Landsat.kea'
inProcessImg = 'Sundarbans_15_MOS_F02DAR.kea'
outImg = 'Sundarbans_15_MOS_F02DAR_utm45n.kea'

imageutils.resampleImage2Match(inRefImg, inProcessImg, outImg,
'KEA', rsgislib.TYPE_32FLOAT)
```
CHAPTER 4. DATA PRE-PROCESSING

imageutils.popImageStats(outImg, usenodataval=True, nodataval=0, calcpyramids=True)

The documentation for this command is available here [http://www.rsgislib.org/rsgislib_imageutils.html#rsgislib.imageutils.resampleImage2Match](http://www.rsgislib.org/rsgislib_imageutils.html#rsgislib.imageutils.resampleImage2Match). The command can be used where ever you need one image to match another in terms of the pixel grid, pixel size, extent and projection. If you want to re-project without providing an image file to match to then you can use rsgislib.imageutils.reprojectImage. Other useful commands for dealing with projection information include:

- rsgislib.imageutils.assignProj – assigned a wkt file to be the projection representation for the input image.
- rsgislib.imageutils.assignSpatialInfo – assign the spatial header information for the input image.
- rsgislib.imageutils.copyProjFromImage – copy the projection representation from one image and assign to another.
- rsgislib.imageutils.copySpatialAndProjFromImage – copy the projection and spatial header from one image and assigned to another.

4.3 Create Valid Data Extent

The following script creates a valid data mask for both the PALSAR-2 and Landsat-8 imagery. The valid data command outputs a binary mask, where 1 means that the pixel is valid data and 0 not. Valid data is defined as a pixel where no band is assigned the no data value (i.e., in this case 0).

```
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils
from rsgislib import rastergis

landsatImg = 'Sundarbans_201511_Landsat.kea'
palsarImg = 'Sundarbans_15_MOS_F02DAR_utm45n.kea'
validMask = 'Sundarbans_validmsk.kea'

imageutils.genValidMask(inimages=[landsatImg, palsarImg], outimage=validMask, gdalformat='KEA', nodata=0.0)
rastergis.populateStats(clumps=validMask, addclrtab=True, calcpyramids=True, ignorezero=True)
```
4.4 Raster-to-Vector

We will now use the raster to vector function to create a polygon shapefile for the area of valid data:

```python
#!/usr/bin/env python

import rsgislib
from rsgislib import vectorutils

inputImg = 'Sundarbans_validmsk.kea'
outShp = 'Sundarbans_validmsk_shp.shp'
vectorutils.polygoniseRaster(inputImg, outShp, imgBandNo=1, maskImg=inputImg, imgMaskBandNo=1)
```

The documentation is here: [http://www.rsgislib.org/rsgislib_vectorutils.html#rsgislib.vectorutils.polygoniseRaster](http://www.rsgislib.org/rsgislib_vectorutils.html#rsgislib.vectorutils.polygoniseRaster)

4.4.1 Vector-to-Raster

Within RSGISLib we commonly use rasters in preference of vectors and therefore rasterising a vector on to same pixel grid as the imagery we intend to use the vector layer with is a common operation. In this case, we demonstrate rasterising the vector we just extracted from the previous step but on to the image grid of the Landsat image (i.e., rather than the combined valid mask image which is a different (common) extent).

```python
#!/usr/bin/env python

import rsgislib
from rsgislib import vectorutils

inputVec = 'Sundarbans_validmsk_shp.shp'
inputImage = 'Sundarbans_201511_Landsat.kea'
outImage = 'Sundarbans_ValidMask_Landsat_tmp.kea'
vectorutils.rasterise2Image(inputVec, inputImage, outImage, gdalformat='KEA', burnVal=1)
```

The documentation for this command is available here [http://www.rsgislib.org/rsgislib_vectorutils.html#rsgislib.vectorutils.polygoniseRaster](http://www.rsgislib.org/rsgislib_vectorutils.html#rsgislib.vectorutils.polygoniseRaster). Please note that this function is using the gdal_rasterize command [http://www.gdal.org/gdal_](http://www.gdal.org/gdal_)
rasterize.html) and therefore the documentation for that might also be a useful reference.

The following commands are also worth being aware of:

- `rsgislib.rastergis.importVecAtts` – allows attribute information from the vector to be copied to a raster file (i.e., raster attribute table). You must be using KEA or HFA file types for this command.
- `rsgislib.vectorutils.copyShapefile2RAT` – This command combines the two command to rasterize a vector file and copy all the attributes to the output raster.

### 4.5 Image Spatial Subsetting

The next command allows you to subset to a region of interest an image file, in this case copy the shapefile generated from the previous step. We could have also used the valid image mask (`Sundarbans_validmsk.kea`) directly using the `rsgislib.imageutils.subset2img` function, but for the sake of the course I wanted to illustrate the vector-to-raster and raster-to-vector functions. You can now run the script `07_Subset2ROI.py`:

```python
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

inputvector = 'Sundarbans_validmsk_shp.shp'
inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub.kea'
imageutils.subset(inputimage, inputvector, outputimage, 'KEA', rsgislib.TYPE_32FLOAT)
imageutils.popImageStats(outputimage, usenodataval=True, nodataval=0, calcpyramids=True)

inputimage = 'Sundarbans_201511_Landsat.kea'
outputimage = 'Sundarbans_201511_Landsat_sub.kea'
imageutils.subset(inputimage, inputvector, outputimage, 'KEA', rsgislib.TYPE_16UINT)
imageutils.popImageStats(outputimage, usenodataval=True, nodataval=0, calcpyramids=True)
```

The documentation for this command is available here [http://www.rsgislib.org/rsgislib_imageutils.html#rsgislib.imageutils.subset](http://www.rsgislib.org/rsgislib_imageutils.html#rsgislib.imageutils.subset) This command subsets to the extend on a shapefile, however there are also commands to subset using other datasets:
• \texttt{rsgislib.imageutils.subset2img} – subsets an image to the same extent as another image.

• \texttt{rsgislib.imageutils.subsetbbox} – subsets an image to the extent defined by the bounding box.

• \texttt{rsgislib.imageutils.subset2polys} – creates multiple output images, matching the number of polygons in the input file. Each output image is subsetted to the extent of one of the polygons in the input shapefile.

• \texttt{rsgislib.imageutils.subsetImgs2CommonExtent} – calculates the union in extent of all the images and subsets them all to that extent.

  – \texttt{rsgislib.imageutils.buildImgSubDict} – can be used to automatically build the input for the \texttt{rsgislib.imageutils.subsetImgs2CommonExtent} command.

\section*{4.6 Image Masking}

Masking an image is a commonly applied function to eliminate parts of the image data we do not want to consider. RSGISLib has a flexible masking command \url{http://www.rsgislib.org/rsgislib_imageutils.html#rsgislib.imageutils.maskImage}, this is used below to mask both the landsat and palar-2 data to the valid region of both.

```python
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

imagemask = 'Sundarbans_validmsk.kea'

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk.kea'

imageutils.maskImage(inputimage, imagemask, outputimage, 'KEA', rsgislib.TYPE_32FLOAT, 0.0, 0.0)

imageutils.popImageStats(outputimage, usenodataval=True, nodataval=0, calcpyramids=True)

inputimage = 'Sundarbans_201511_Landsat_sub.kea'
outputimage = 'Sundarbans_201511_Landsat_sub_msk.kea'

imageutils.maskImage(inputimage, imagemask, outputimage, 'KEA', rsgislib.TYPE_16UINT, 0.0, 0.0)

imageutils.popImageStats(outputimage, usenodataval=True, nodataval=0, calcpyramids=True)
```
Please note when you are masking, the final two input parameters outvalue the value which is going to be given to the pixels which are being ‘masked out’ during the process and maskvalue which is the pixel value(s) in the mask file which correspond to the regions in the input image which will be ‘masked out’ and assigned the outvalue. It is worth noting that maskvalue can also be a list of values (e.g., [1,2,3]), when a list of value is provided all the values are treat the same and replaced with the same outvalue.

4.7 Image Band Subsetting

RSGISLib can also subset in terms of the image bands in the file, in this case for converting to dBs we just want the HH and HV bands (i.e., 1 and 2). The following script performs that operation.

```bash
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea'
imageutils.selectImageBands(inputimage, outputimage, 'KEA',
                          rsgislib.TYPE_32FLOAT, [1,2])

imageutils.popImageStats(outputimage, usenodataval=True,
                          nodataval=0, calcpyramids=True)
```

4.8 Apply Lee Filter

For many applications it can be useful to filter the SAR imagery, RSGISLib has a number of filters available (http://www.rsgislib.org/rsgislib_imagefilter.html) but in this case a Lee filter is applied.

```bash
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imageutils
from rsgislib import imagefilter

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee.kea'
```
imagefilter.applyLeeFilter(inputimage, outputimage, 5, 5, 'KEA', rsgislib.TYPE_32FLOAT)

imageutils.popImageStats(outputimage, usenodataval=True, nodataval=0, calcpyramids=True)

If you have a very large image, filtering can be very slow. Therefore, RSGISLib provides a tiled base filtering option (rsgislib.imagefilter.tiledfilter.performTiledImgMultiFilter) which first tiles the input image, with an appropriate overlap, filters each of the tiles, using multiple processing cores if requested, and then mosaics the results ignoring the overlap resulting in a seamless result but much faster for large images.

### 4.9 Band Maths: Power – dBs

Before classification we also want to convert the HH and HV bands to dBs, in this case we’ll use the rsgislib.imagecalc.imageMath function rather than rsgislib.imagecalc.bandMath that was shown earlier as it outputs a multi-band image where the same expression is applied to all the bands. In the expression the band must be referred to as b1. The following script converts both the filtered and unfiltered image to dBs.

```python
#!/usr/bin/env python
# Import python modules
import rsgislib
from rsgislib import imageutils
from rsgislib import imagecalc

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_dB.kea'
imagecalc.imageMath(inputimage, outputimage, 'b1>0?10*log10(b1):999', 'KEA', rsgislib.TYPE_32FLOAT)

imageutils.popImageStats(outputimage, usenodataval=True, nodataval=999, calcpyramids=True)

inputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_Lee.kea'
outputimage = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_Lee_dB.kea'
imagecalc.imageMath(inputimage, outputimage, 'b1>0?10*log10(b1):999', 'KEA', rsgislib.TYPE_32FLOAT)

imageutils.popImageStats(outputimage, usenodataval=True, nodataval=999, calcpyramids=True)
```
4.10 Band Maths: NDVI & WBI

From optical image the normalised difference vegetation index (NDVI) and water band index (WBI) are useful indices which are commonly applied. These can be calculate as shown below using the `imagecalc.bandmaths` function.

```python
#!/usr/bin/env python

# Import python modules
import rsgislib
from rsgislib import imagecalc
from rsgislib import imageutils

inputimage = 'Sundarbans_201511_Landsat_sub_msk.kea'
outputimage = 'Sundarbans_201511_Landsat_sub_msk_ndvi.kea'
bandDefns = [imagecalc.BandDefn('red', inputimage, 4),
             imagecalc.BandDefn('nir', inputimage, 5)]
imagecalc.bandMath(outputimage, 'nir==0?999:(nir-red)/(nir+red)', 'KEA',
                    rsgislib.TYPE_32FLOAT, bandDefns)
imageutils.popImageStats(outputimage, usenodataval=True,
nodataval=999, calcpyramids=True)

outputimage = 'Sundarbans_201511_Landsat_sub_msk_wbi.kea'
bandDefns = [imagecalc.BandDefn('blue', inputimage, 2),
             imagecalc.BandDefn('nir', inputimage, 5)]
imagecalc.bandMath(outputimage, 'nir==0?999:(blue/nir)', 'KEA',
                    rsgislib.TYPE_32FLOAT, bandDefns)
imageutils.popImageStats(outputimage, usenodataval=True,
nodataval=999, calcpyramids=True)
```

4.11 Putting it all together

It is worth stating that all those scripts could be combined to create a single script rather than multiple separate scripts. For easy of learning/understanding I have presented this course as individual scripts but if I was applying this analysis for myself I would have put together a single script. An overall script is provided in `13_CombinedPreProcessing.py` for illustration.

4.12 Tidy up your files

Following all that analysis you will have a number of intermediate files taking up disk space, clean that up so the only files you have remaining are:
• Sundarbans_15_MOS_F02DAR_utm45n_sub_msk.kea – HH, HV and HH/HV power image.

• Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea – HH and HV power image.

• Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee.kea – Lee filtered HH and HV power image.

• Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_dB.kea – HH and HV dB image.

• Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee_dB.kea – Lee filtered HH and HV dB image.

• Sundarbans_201511_Landsat_sub_msk.kea – Landsat image subsetted and masked.

• Sundarbans_201511_Landsat_sub_msk_wbi.kea – WBI Image.

• Sundarbans_201511_Landsat_sub_msk_ndvi.kea – NDVI image.

• Sundarbans_validmsk.kea – Valid pixel mask.
Chapter 5

Image Classification

Image classification is the process of assigning regions (either pixels or segments) to a thematic class based on the available information (i.e., backscatter, spectral reflectance etc.).

Classifications can be performed using a number of methods, on which there is a large body of literature, and the choice of approach is based on a number of factors. The primary factor is which one produces the best (i.e., most accurate) result. However, in choosing a particular algorithm or approach the assumptions of that approach needs to be met. For example, when using Gaussian Maximum Likelihood you are making the assumption that your training data is normally distributed – or at least close to being. Some algorithms require all the input data to be in the same value range, while others will be able to use categorical data within the classification and others won’t. It is not in the scope of this document to consider those issues but please ensure that if you are not sure that you look up the classifier assumptions applying in your own applications.

In terms of approaches you can group them as follows:

- Rule Base – i.e., manually defined decision trees.
- Unsupervised – i.e., clustering algorithms such as KMeans and ISOData.
- Statistical Supervised
  - Minimum Distance
  - Paralleled Pipe
  - Mahalanobis Distance
  - Gaussian Maximum Likelihood
- Machine Learning – basically more advanced statistical supervised classifiers.
  - K- nearest neighbour

42
CHAPTER 5. IMAGE CLASSIFICATION

- Decision Trees
- Support Vector Machines
- Random Forests
- Neural Networks

RSGISLib and associated tools can provide options for applying most of these classifiers through the range of Python libraries which implement these algorithms using numpy (http://www.numpy.org) as the array data representation.

Another common difference in approaches is what you classify. Most commonly, classification is undertaken on individual pixels however this is also a wide body of literature which proposes that if a good image segmentation can be achieved, providing a good representation of the environment and features which are being classified then a better overall classification can be achieved. RSGISLib therefore provides tools for performing per-pixel and segment (object) based classification. In this course both pixel and object methods will be demonstrated using machine learning classifiers from the scikit-learn library.

The scene you are classifying is the Sundarbans on the Indian and Bangladesh border. The classes of interest are:

- Mangroves
- Water
- Agriculture
- Urban
- Forests/Rural Livings

You have been provided with some training samples as a shapefile, one for each class. *Please note: The training data provided is not a demonstration of best practice but quickly drawn for illustration purposes.*

### 5.1 Pixel Based

Create a new directory **Exercise4** and copy the previous files into that directory including the shapefiles provided to train the classifier.

The steps for the script to perform the classification are:

- Rasterise the vector layers providing the training regions for the classification.
• Sample the pixels within the raster regions for each class (i.e., normalise the number of training samples for each class).

• Define the layers to be used for the classification and for which the training data needs to be extracted.

• Extract the training data from the input images for the sampled training pixels and save as a HDF5 file.

• Define the classification classes with the HDF5 file holding the training data and the colour to be used to visualise the classification.

• Create the scikit-learn classifier – any classifier in the library can be defined with the required parameters.
  
  – Note. the function `classimgutils.findClassifierParametersAndTrain` can be used to find the optimal classifier parameters and train the classifier.

• Train the classifier

• Apply the classifier

This has been implemented in the following script (`14_PixelBasedClass.py`), copy it into your working directory and run it to produce your classification.

```python
# /usr/bin/env python
import rsgislib
from rsgislib import imageutils
from rsgislib import vectorutils
from rsgislib.classification import classimgutils
from sklearn.ensemble import ExtraTreesClassifier

landsatImg = 'Sundarbans_201511_Landsat_sub_msk.kea'
palsar2Img = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee_dB.kea'
landsatNDVI = 'Sundarbans_201511_Landsat_sub_msk_ndvi.kea'
landsatWBI = 'Sundarbans_201511_Landsat_sub_msk_wbi.kea'
validImgMsk = 'Sundarbans_validmsk.kea'
mangroveVec = 'Mangroves.shp'
urbanVec = 'Urban.shp'
agricultureVec = 'Agriculture.shp'
forestRuralVec = 'ForestRuralUrban.shp'
waterVec = 'Water.shp'
mangroveImg = 'MangrovesTrain.kea'
urbanImg = 'UrbanTrain.kea'
agricultureImg = 'AgricultureTrain.kea'
forestRuralImg = 'ForestRuralUrbanTrain.kea'
waterImg = 'WaterTrain.kea'
```
mangroveImageSamp = 'MangrovesTrainSamp.kea'
urbanImageSamp = 'UrbanTrainSamp.kea'
agricultureImageSamp = 'AgricultureTrainSamp.kea'
forestRuralImageSamp = 'ForestRuralUrbanTrainSamp.kea'
waterImageSamp = 'WaterTrainSamp.kea'
mangrovePx1TrainVals = 'MangrovesPx1TrainVals.h5'
urbanPx1TrainVals = 'UrbanPx1TrainVals.h5'
agriculturePx1TrainVals = 'AgriculturePx1TrainVals.h5'
forestRuralPx1TrainVals = 'ForestRuralUrbanPx1TrainVals.h5'
waterPx1TrainVals = 'WaterPx1TrainVals.h5'

# Rasterise the training samples (shapefiles)
vectorutils.rasterise2Image(mangroveVec, landsatImg, mangroveImg,
gdalformat='KEA', burnVal=1)
vectorutils.rasterise2Image(urbanVec, landsatImg, urbanImg,
gdalformat='KEA', burnVal=2)
vectorutils.rasterise2Image(agricultureVec, landsatImg, agricultureImg,
gdalformat='KEA', burnVal=3)
vectorutils.rasterise2Image(forestRuralVec, landsatImg, forestRuralImg,
gdalformat='KEA', burnVal=4)
vectorutils.rasterise2Image(waterVec, landsatImg, waterImg,
gdalformat='KEA', burnVal=5)

# The number of pixels to be sampled from each training class
nSamples = 40000

# Sample the training did to identify the training pixels.
imageutils.performRandomPx1SampleInMaskLowPx1Count(inputImage=mangroveImg,
outputImage=mangroveImageSamp,
gdalformat='KEA', maskvals=[1],
numSamples=nSamples)
imageutils.performRandomPx1SampleInMaskLowPx1Count(inputImage=urbanImg,
outputImage=urbanImageSamp,
gdalformat='KEA', maskvals=[2],
numSamples=nSamples)
imageutils.performRandomPx1SampleInMaskLowPx1Count(inputImage=agricultureImg,
outputImage=agricultureImageSamp,
gdalformat='KEA', maskvals=[3],
numSamples=nSamples)
imageutils.performRandomPx1SampleInMaskLowPx1Count(inputImage=forestRuralImg,
outputImage=forestRuralImageSamp,
gdalformat='KEA', maskvals=[4],
numSamples=nSamples)
imageutils.performRandomPx1SampleInMaskLowPx1Count(inputImage=waterImg,
outputImage=waterImageSamp,
gdalformat='KEA', maskvals=[5],
numSamples=nSamples)

# Define the layers to be used for classification and the bands in those layers.
imgFileInfo = [imageutils.ImageBandInfo(landsatImg, 'landsat', [2,3,4,5,6,7]),
imageutils.ImageBandInfo(palsar2Img, 'palsar2', [1,2]),
imageutils.ImageBandInfo(landsatNDVI, 'ndvi', [1]),
imageutils.ImageBandInfo(landsatWBI, 'wbi', [1])

# Extract the training data to HDF files.
imageutils.extractZoneImageBandValues2HDF(imgFileInfo, mangroveImgSamp, mangrovePxlTrainVals, 1.0)
imageutils.extractZoneImageBandValues2HDF(imgFileInfo, urbanImgSamp, urbanPxlTrainVals, 2.0)
imageutils.extractZoneImageBandValues2HDF(imgFileInfo, agricultureImgSamp, agriculturePxlTrainVals, 3.0)
imageutils.extractZoneImageBandValues2HDF(imgFileInfo, forestRuralImgSamp, forestRuralPxlTrainVals, 4.0)
imageutils.extractZoneImageBandValues2HDF(imgFileInfo, waterImgSamp, waterPxlTrainVals, 5.0)

# Define the classes to be classified with the training data (HDF5 file) and colour
classTrainInfo = dict()
classTrainInfo['Mangroves'] = classimgutils.ClassInfoObj(id=1, fileH5=mangrovePxlTrainVals, red=0, green=153, blue=0)
classTrainInfo['Urban'] = classimgutils.ClassInfoObj(id=2, fileH5=urbanPxlTrainVals, red=192, green=192, blue=192)
classTrainInfo['Agriculture'] = classimgutils.ClassInfoObj(id=3, fileH5=agriculturePxlTrainVals, red=204, green=204, blue=0)
classTrainInfo['ForestRural'] = classimgutils.ClassInfoObj(id=4, fileH5=forestRuralPxlTrainVals, red=0, green=204, blue=204)
classTrainInfo['Water'] = classimgutils.ClassInfoObj(id=5, fileH5=waterPxlTrainVals, red=0, green=0, blue=204)

# Create scikit-learn classifier (can be any in the library)
skClassifier = ExtraTreesClassifier(n_estimators=100)
# Train that classifier
classimgutils.trainClassifier(classTrainInfo, skClassifier)

# Apply the classification.
outImgClass = 'Sundarbans_PxlBaseClass.kea'
classimgutils.applyClassifier(classTrainInfo, skClassifier, validImgMsk, 1, imgFileInfo, outImgClass, 'KEA')

If you need to make some disk space available before going to the next classification stage, remove all the files other than the outputted classification (Sundarbans_PxlBaseClass.kea) as we will compare this classification to the object based result produced in the next step.

## 5.2 Object Based

Object based classification first required that a segmentation is undertaken. The algorithm provided in RSGISLib is executed using the function segutils.runShepherdSegmentation.
The algorithm is initialised using a K-Means clustering and then small objects are eliminated, merging into their ‘nearest’ neighbour (defined using the euclidean distance),
The segmentation image (Sundarbans_Clumps.kea) has been provided to save analysis time but is generated using the following script (15_SegData.py):

```python
#!/usr/bin/env python
import rsgislib
from rsgislib.segmentation import segutils

landsatImg = 'Sundarbans_201511_Landsat_sub_msk.kea'
palsar2Img = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV_lee.kea'

# Select the image bands to be used for segmentation from the landsat image.
landsatImg564 = 'Sundarbans_201511_Landsat_sub_msk_b564.kea'
rggislib.imageutils.selectImageBands(landsatImg, landsatImg564, 'KEA', rsgislib.TYPE_16UINT, [5, 6, 4])

# Stack the selected landsat bands with the PALSAR-2 imagery.
segTmpImg = 'Sundarbans_SegTempImg.kea'
rggislib.imageutils.stackImageBands([landsatImg564, palsar2Img], None, segTmpImg, None, 0, 'KEA', rsgislib.TYPE_32FLOAT)

# Perform the segmentation
clumpsImg = 'Sundarbans_Clumps.kea'
segutils.runShepherdSegmentation(segTmpImg, clumpsImg, tmpath='./segtmp', numClusters=120, minPxls=50, distThres=100, sampling=100, kmMaxIter=200)
```

For very large image datasets there is a tiled version of the segmentation algorithm. At the moment, this is still single threaded but in the future a version of the algorithm will be able to use multiple processing cores on a single machine. The function is `rsgislib.segmentation.tiledsegsingle.performTiledSegmentation`.

For the next stage of processing you will need to create a new working directory (Exercise5), copy the files produced during the pre-processing other than the dB images (we calculate the mean of the power image and then convert to dBs) and the clumps image (Sundarbans_Clumps.kea) into that working directory.

The stages of processing which will be undertaken to perform the classification are as follows (within 16_PerformObjClass.py)

- Populate the segmentation (clumps file) with the variables you wish to use for the classification. Building the raster attribute table. For this study we will populate with:
  - Mean HH and HV backscatter (power)
  - Mean Landsat reflectance
CHAPTER 5. IMAGE CLASSIFICATION

- Mean NDVI
- Mean WBI

- Convert mean HH and HV power values to dBs.
- Define the training data and add it to the RAT.
- Balance the training data so the number of samples are similar/same for each class.
- Define variables to be used for the classification
- Search for the optimal classifier parameters (GridSearch) and train the classifier.
- Define the class output colours
- Applying the classifier
- Collapse the clumps file to just the classification

This is implemented with the following script, copy it into your working directory and run it.

```python
#usr/bin/env python

from rsgislib import rastergis
from rsgislib.rastergis import ratutils
from rsgislib.classification import classratutils
from rsgislib import classification
import osgeo.gdal as gdal
from rios import rat
import numpy
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.grid_search import GridSearchCV

landsatImg = 'Sundarbans_201511_Landsat_sub_msk.kea'
palsar2Img = 'Sundarbans_15_MOS_F02DAR_utm45n_sub_msk_HHHV.kea'
landsatNDVI = 'Sundarbans_201511_Landsat_sub_msk_ndvi.kea'
landsatWBI = 'Sundarbans_201511_Landsat_sub_msk_wbi.kea'
validImgMsk = 'Sundarbans_validmsk.kea'
clumpsImg = 'Sundarbans_Clumps.kea'

# Add PALSAR-2 HH and HV Mean power to each of the segments
bs = [rastergis.BandAttStats(band=1, meanField='HHPow'),
      rastergis.BandAttStats(band=2, meanField='HVPow')]
rastergis.populateRATWithStats(palsar2Img, clumpsImg, bs)

# Add mean landsat reflectance to each of the segments
bs = [rastergis.BandAttStats(band=1, meanField='CoastMean'),
      rastergis.BandAttStats(band=2, meanField='BlueMean'),
      rastergis.BandAttStats(band=3, meanField='GreenMean'),
      rastergis.BandAttStats(band=4, meanField='RedMean')]
rastergis.populateRATWithStats(landsatImg, clumpsImg, bs)
```

rastergis.BandAttStats(band=4, meanField='RedMean'),
rastergis.BandAttStats(band=5, meanField='NIRMean'),
rastergis.BandAttStats(band=6, meanField='SWIR1Mean'),
rastergis.BandAttStats(band=7, meanField='SWIR2Mean')]
rastergis.populateRATWithStats(landsatImg, clumpsImg, bs)

# Add mean NDVI reflectance to each of the segments
bs = [rastergis.BandAttStats(band=1, meanField='NDVI')]
rastergis.populateRATWithStats(landsatNDVI, clumpsImg, bs)

# Add mean NDVI reflectance to each of the segments
bs = [rastergis.BandAttStats(band=1, meanField='WBI')]
rastergis.populateRATWithStats(landsatWBI, clumpsImg, bs)

# Create and Calculate dB columns
ratDataset = gdal.Open(clumpsImg, gdal.GA_Update)
HHPow = rat.readColumn(ratDataset, 'HHPow')
HVPow = rat.readColumn(ratDataset, 'HVPow')
HHdB = numpy.where(HHPow>0, 10*numpy.log10(HHPow), 0)
HVdB = numpy.where(HVPow>0, 10*numpy.log10(HVPow), 0)
rat.writeColumn(ratDataset, "HHdB", HHdB)
rat.writeColumn(ratDataset, "HVdB", HVdB)
ratDataset = None

# Create list of training data and populate the RAT.
classesDict = dict()
classesDict['Mangroves'] = [1, 'Mangroves.shp']
classesDict['Urban'] = [2, 'Urban.shp']
classesDict['Agriculture'] = [3, 'Agriculture.shp']
classesDict['ForestRural'] = [4, 'ForestRuralUrban.shp']
classesDict['Water'] = [5, 'Water.shp']
tmpPath = './tmp'
classesIntColIn = 'ClassInt'
classesNameCol = 'ClassStr'
ratutils.populateClumpsWithClassTraining(clumpsImg, classesDict, tmpPath,
classesIntColIn, classesNameCol)

# Ensure there are a minimum number of training samples and
# balance so there are the same number for each class
classesIntCol = 'ClassIntSamp'
classratutils.balanceSampleTrainingRandom(clumpsImg, classesIntColIn,
classesIntCol, 100, 200)

# RAT variables used for classification
variables = ['HHdB', 'HVdB', 'NIRMean', 'SWIR1Mean', 'RedMean', 'NDVI', 'WBI']

classParameters = {'n_estimators':[10,100,500], 'max_features':[2,3,4]}
gSearch = GridSearchCV(ExtraTreesClassifier(), classParameters)
preProcessing = None
classifier = classratutils.findClassifierParameters(clumpsImg, classesIntCol, variables,
classColours = dict()
classColours['Mangroves'] = [0,153,0]
classColours['Urban'] = [192,192,192]
classColours['Agriculture'] = [204,204,0]
classColours['ForestRural'] = [0,204,204]
classColours['Water'] = [0,0,204]

# Perform classification...
outClassIntCol = 'OutClass'
outClassStrCol = 'OutClassName'
classratutils.classifyWithinRAT(clumpsImg, classesIntCol, classesNameCol, variables,
    classifier=classifier, outColInt=outClassIntCol,
    outColStr=outClassStrCol, classColours=classColours,
    preProcessor=preProcessing)

# Export to a 'classification image' rather than a RAT...
outClassImg = 'Sundarbans_ObjBaseClass.kea'
classification.collapseClasses(clumpsImg, outClassImg, 'KEA', 'OutClassName', 'OutClass')

5.3 Compare the Classifications

You have now produced two classification through different methods but using the same data, classifier and training data. How do they compare?
Chapter 6

Conclusions

I hope following the completion of this course you feel you have a better idea of how you can use RSGISLib for your spatial data processing. This course has covered some of the most commonly used functions within RSGISLib however there are 100s more functions you can explore and try out – see the website. Particular areas we have not covered include:

- **Image Calibration** – tools for calibrating imagery, primarily optical data and used within the ARCSI (www.rsgislib.org/arcsi) software for performing atmospheric correction of optical satellite imagery.
- **Elevation** – tools for doing things with DEMs.
- **Image Morphology** – functions for applying image morphology operations
- **Image Registration** – function for applying an automated image-to-image registration
- **Zonal Statistics** – functions for retrieving information into a vector attribute table from a raster image.

We’ve only scratched the surface of a number of libraries (i.e., rastergis) and I would recommend you have a look though these on the website (http://www.rsgislib.org).

Don’t forget, that any or all of these scripts can be wrapped up as functions and command line tools if required or even attached to a GUI using pyqt or similar library for creating your GUI. If wrapped up as functions then don’t forget that you can use the multiprocessor module to apply batch processing using multiple processing cores.
Bibliography


Appendix A

A brief introduction to the UNIX terminal

Linux is a UNIX like operating system which is widely used, for example the Android operating system running on most mobile phones uses the Linux kernel and most servers forming the internet are running a flavour of Linux/UNIX. In fact, the majority of you are using a UNIX/Linux operating system every day as while Android is Linux MacOS and iOS run an a flavour of UNIX based on the BSD (Berkeley Software Distribution) UNIX. Linux/UNIX is generally a free operating system which is very stable and secure. There are many application which can be run the operating system and it can be used in place of Windows, either as MacOS or various desktop flavours of Linux (e.g., Mint https://www.linuxmint.com, Ubuntu https://www.ubuntu.com, CentOS (https://www.centos.org), of which there are hundreds.

In this tutorial we will be looking at using the UNIX Terminal (Figure 1.1), alternatively called the ‘Command Line’. This is a text based interface to the computer system, which allow manipulation of the files and directories of the file system and actions to be carried out on those files, including in the form of scripts. Scripts an simple a formal, computer readable, method of writing down a series of actions (steps) the computer is to carry out in sequence. You can do everything that you can typically do through a graphical user interface (GUI) using the terminal and when scripted it is more reproducible and can be more easily applied to large datasets through batch processing.

The majority of high performance computing (HPC) environments and cloud services such as Amazon and Google use the Linux operating system and in many cases only provide a terminal driven interface through which to configure and install your applications. Knowledge and ability to use a terminal interface on Linux is therefore tremendously useful.
A.1 Working with the file system

When you open a Terminal window you will almost certainly start in your home directory, this is the directory structure where you are expected to store your files.

A.1.1 Listing your current directory

The first command you need is to list the file in the directory you are currently working in. You might find it useful to have the graphical file browser open alongside the terminal as you go through these steps.

To list the files in your home directory, run the following command:

```
ls
```

Compare what you see in your graphical file browser.

Most commands and options which are accessed via things called ‘switches’, this come straight after the command (with a space after the command) with a dash (‘-’) and then a letter or double-dash (‘--’) and then a word.

A useful switch on the `ls` command is `-l`, which lists the files with information, such as size and date of modification.

```
ls -l
```

Again, compare to your graphical file browser (list view). You can have more than one switch on a command, for example the `-h` switch on the `ls` command is useful when combined with `-l` is it converts the files sizes to a human readable unit. Try:

```
ls -l -h
```

note that you can combine switches, so `-lh` is the same as `-l` `-h`, try:

```
ls -lh
```

If you want to sort files by time then add `-t`

```
ls -lht
```

A.1.2 Where am I?

A common problem is not knowing where you are within the file system, in a graphical interface you can usually click back to home and start navigating again and you have
an address bar on the top of the screen telling you where you are. There is a command for to providing you the same information as is within the address bar:

```bash
pwd
```

Compare what `pwd` has printed to the terminal screen and what is in your graphical browser.

### A.1.3 Moving Directories

Within your graphical interface you will be used to double clicking on a directory icon and the interface updating to show you the contents of that directory. Obviously, you can do the same thing on the terminal, using the `cd` (change directory) command.

Try moving to the directory Desktop:

```bash
cd Desktop
```

Check where you are in the file system using the `pwd` command and then list the files within this directory using the `ls` command. Compare this to the graphical file browser.

Once you have moved into a directory you will want to go back (i.e., as you would it you pressed the back button in your graphical interface). On the terminal this is done using two dots (..), which means move one directory back. Try it and then run `pwd` and `ls` to check you are back in your home directory.

```bash
cd ..
pwd
ls
```

You can provide a whole path to the `cd` command, referring to multiple directories. For example,

```bash
cd Desktop
pwd
cd ../Documents
pwd
```

If you want to get back to your home directory where ever you are within the directory structure you can just type `cd` without specifying a directory path.

```bash
cd
```
It is also useful to know that you can navigate to your home directory using the ~ symbol:

```bash
cd ~
```

and that it can be used to navigate to a path starting from your home directory, for example:

```bash
cd ~/Downloads
```

### A.1.4 Creating a new directory

The command to create a new directory is `mkdir`. To create a new directory run the command providing the name of the new directory as input. For example, run the following starting in your home directory.

```bash
cd Desktop
ls
mkdir Test
ls
cd Test
pwd
```

If you have a whole directory path you want to create then you can use the `-p` switch which will create all the directories list in the path provided which don’t currently exist. For example, run the following.

```bash
cd Desktop
mkdir -p Test/Test1/Test2/Test3/Test4
ls
cd Test
pwd
ls
cd Test1
pwd
ls
cd Test2/Test3/Test4
pwd
```

To navigate back to the Desktop you can run either:

```bash
cd ../../../
```

or
A.1.5 Case Sensitive

You should be aware that file and directory names are case sensitive and therefore Test and test would be two different files or directories on your system. Try:

```
cd test
```

This fails as the directory you created earlier is Test.

A.1.6 Copying Files and Directories

For illustration download an existing file, for example an image e.g., LS8_20130511_lat450lon16985_r91p75_sref.kea. These instructions assume that the file will be downloaded to the 'Downloads' directory. You can list the contents of your Downloads directory using the following command:

```
ls ~/Downloads
```

Check what has been listed using your graphical file browser.

To copy a file or directory you need to use the `cp` command, for example to copy the LS8_20130511_lat450lon16985_r91p75_sref.kea to the Test folder you have created on your Desktop you can use the following command:

```
cp ~/Downloads/LS8_20130511_lat450lon16985_r91p75_sref.kea ~/Desktop/Test
ls ~/Desktop/Test
```

Check in your graphical file browser what has happened with the files.

Let’s copy the Test directory to your documents folder, you can do this using the `-R` switch, which causing the `command` to recurse down a directory structure.

```
cd ~/Desktop
cp -R ./Test ~/Documents
ls ~/Documents
```

```
cd ~/Documents/Test/Test1/Test2/Test3/Test4
pwd
```
Check what you have in your graphical file browser.

### A.1.7 Moving Files and Directories

Moving rather than copying files and directories means that the data is moved rather than a copy made. For example, let's move the copy of `LS8_20130511_lat450lon16985_r91p75_sref.kea` in the directory `Documents\Test` to the `Desktop`, run the following:

```
ls ~/Documents/Test
mv ~/Documents/Test/LS8_20130511_lat450lon16985_r91p75_sref.kea ~/Desktop
```

Let's move the test directory structure to your `Downloads` folder, run the following commands.

```
cd ~/Documents
ls
mv Test ~/Downloads
ls
cd ~/Downloads
ls
cd Test
pwd
```

Again, check in your graphical file browser so you can see what has happened.

### A.1.8 Removing a Directory

Now you have created some directories and moved things about you've got a bit of mess to clear up so let's look at deleting files and directories. Deleting is done using the `rm` (remove) command.

Let's first delete the KEA image file we can left on the Desktop:

```
cd ~/Desktop
ls
rm LS8_20130511_lat450lon16985_r91p75_sref.kea
ls
```

Check in your graphical file browser that the file has been removed.
Similar to the copy command, to delete a directory we need to use the \texttt{-R} command to recurse through the directory structure deleting it all, let’s delete the Test folder on the Desktop:

\begin{verbatim}
cd ~/Desktop
ls
rm -R Test
ls
\end{verbatim}

Check in your graphical file browser that the directory has been removed. Let’s do the same in your Downloads directory.

\begin{verbatim}
cd ~/Downloads
ls
rm -R Test
ls
\end{verbatim}

### A.1.9 Decompressing files

The most common compression format under UNIX is a tar.gz file, where the \texttt{tar} command has been used to create an archive and the \texttt{gzip} command has been used to compress the archive. This can all be accomplished using the \texttt{tar} command. For example, using the \texttt{LC82040242013139LGN01.tar.gz} file downloaded from the USGS Earth Explorer (https://earthexplorer.usgs.gov).

\begin{verbatim}
tar -zxf LC82040242013139LGN01.tar.gz
\end{verbatim}

The switches are: \texttt{z} uncompress using gzip, \texttt{x} extract from archive and \texttt{f} from file.

### A.2 Shell Scripts

A shell script (‘the shell’ is another term for the terminal) is a simple way of listing the command you wish to run in a test file and then running them in a single step. A shell script is just a simple text file, so open a text editor and save an empty file on your Desktop called \texttt{testscript.sh}, we use the file extension \texttt{.sh} to indicate a shell script. Within the script save the following:

\begin{verbatim}
echo "Hello World"
\end{verbatim}

To run the script (ensure it is saved in your text editor) and then type the following on the terminal:
**APPENDIX A. A BRIEF INTRODUCTION TO THE UNIX TERMINAL**

---

```bash
cd ~/Desktop
sh ./testscript.sh
```

Edit the script again so it reads:

```bash
echo "Hello World"
ls
cd ..
pwd
cd Documents
pwd
ls
cd ~/Desktop
mkdir -p Test/Test1/Test2/Test3/Test4
cd Test/Test1/Test2/Test3/Test4
pwd
```

This can be useful for documenting the commands you are running so it is reproducible but it can also be used for batch processing using a command or python script which is doing something to our input data (e.g., images) and we have a lot of processes, for example we could create a shell script similar to the one below and when it runs it’ll run each of the images in turn.

```bash
python ImageProcessingCmd.py inputimg1.kea
python ImageProcessingCmd.py inputimg2.kea
python ImageProcessingCmd.py inputimg3.kea
python ImageProcessingCmd.py inputimg4.kea
python ImageProcessingCmd.py inputimg5.kea
python ImageProcessingCmd.py inputimg6.kea
```

Note. Python scripts have the file extension `py`.

---

**A.3 Conclusions**

You should now be able to move around the Linux/UNIX Terminal moving from one directory to another and manipulating files. There is a lot more that you can do which is out of scope here but there are loads of online resources and materials you can use. However, just spending time using the Terminal and command line tools is the best way to learn, there are few short cuts to perseverance.
Appendix B

A brief introduction to Python

B.1 Starting Python

Python may be started by opening a command (Terminal) window and typing:

```
python
```

This opens python in interactive mode. It is possible to perform some basic maths try:

```
>>> 1 + 1
2
```

To exit type:

```
>>> exit()
```

To perform more complex tasks in python often a large number of commands are required, it is therefore more convenient to create a text file containing the commands, referred to as a ‘script’

B.1.1 Indentation

There are several basic rules and syntax which you need to know to develop scripts within Python. The first of which is code layout. To provide the structure of the script Python uses indentation. Indentation can be in the form of tabs or spaces but which ever is used needs to be consistent throughout the script. The most common and recommend is to use 4 spaces for each indentation. The example given below shows an if-else statement where you can see that after the if part the statement which is executed
APPENDIX B. A BRIEF INTRODUCTION TO PYTHON

if the if-statement is true is indented from rest of the script as with the corresponding else part of the statement. You will see this indentation as you go through the examples and it is important that you follow the indentation shown in the examples or your scripts will not execute.

```python
if x == 1:
    x = x + 1
else:
    x = x - 1
```

B.1.2 Keywords

As with all scripting and programming languages python has a set of keywords, which have special meanings to the compiler or interpreter when the code is executed. As with all python code, these keywords are case sensitive i.e., ‘else’ is a keyword but ‘Else’ is not. A list of pythons keywords is given below:

<table>
<thead>
<tr>
<th>and</th>
<th>as</th>
<th>assert</th>
<th>break</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>continue</td>
<td>def</td>
<td>del</td>
</tr>
<tr>
<td>elif</td>
<td>else</td>
<td>exec</td>
<td>except</td>
</tr>
<tr>
<td>finally</td>
<td>for</td>
<td>from</td>
<td>global</td>
</tr>
<tr>
<td>if</td>
<td>import</td>
<td>in</td>
<td>is</td>
</tr>
<tr>
<td>lambda</td>
<td>not</td>
<td>or</td>
<td>pass</td>
</tr>
<tr>
<td>print</td>
<td>raise</td>
<td>return</td>
<td>try</td>
</tr>
<tr>
<td>while</td>
<td>with</td>
<td>yield</td>
<td></td>
</tr>
</tbody>
</table>

B.1.3 File Naming

It is important that you use sensible and identifiable names for all the files you generate throughout these tutorial worksheets otherwise you will not be able to identify the script at a later date. Additionally, it is highly recommended that you do not included spaces in file names or in the directory path you use to store the files generated during this tutorial.

B.1.4 Case Sensitivity

Something else to remember when using python, is that the language is case sensitivity therefore if a name is in lowercase then it needs to remain in lowercase everywhere it is used.
For example:

VariableName is not the same as variablename

B.1.5 File paths in examples

In the examples provided (in the text) file paths are given as './PythonCourse/TutorialX/File.xxx'. When writing these scripts out for yourself you will need to update these paths to the location on your machine where the files are located (e.g., /home/pete.bunting or C:\). Please note that it is recommended that you do not have any spaces within your file paths. In the example (answer) scripts provided no file path has been written and you will therefore need to either save input and output files in the same directory as the script or provide the path to the file. Please note that under Windows you need to insert a double slash (i.e., \\) within the file path as a single slash is an escape character (e.g., \n for new line) within strings.

B.1.6 Independent Development of Scripts

There is a significant step to be made from working your way through notes and examples, such as those provided in this tutorial, and independently developing your own scripts from scratch. Our recommendation for this, and when undertaking the exercises from this tutorial, is to take it slowly and think through the steps you need to undertake to perform the operation(s) you need.

I would commonly first ‘write’ the script using comments or on paper breaking the process down into the major steps required. For example, if I were asked to write a script to uncompress a directory of files into another directory I might write the following outline, where I use indentation to indicate where a process is part of the parent:

```python
# Get input directory (containing the compressed files)
# Get output directory (where the files, once uncompressed, will be placed).
# Retrieve list of all files (to be uncompressed) in the input directory.
# Iterator through input files, uncompressing each in turn.
    # Get single file from list
    # create command line command for the current file
    # execute command
```

By writing the process out in this form it makes translating this into python much simpler as you only need to think of how to do small individual elements in python and not how to do the whole process in one step.
B.1.7 Importing Modules

There are a number of modules included with Python and external libraries (e.g., RIOS, RSGISLib), these must be imported so they are available within a script using the `import` statement.

```python
# Import RSGISLib module
import rsgislib

# Import the imagecalc module from RSGISLib
from rsgislib import imagecalc
```

Namespaces are used for each module so, in the example above, to access functions from the image calc module they must be prefixed with `imagecalc`.

B.1.8 Getting Help

Python provides a very useful help system through the command line. To get access to the help run python from the terminal

```bash
>> python
```

Then import the library want to get help on

```python
>>> import math
```

and then run the help tool on the whole module

```python
>>> import math
>>> help(math)
```

or on individual classes or functions within the module

```python
>>> import rsgislib.imageutils
>>> help(rsgislib.imageutils.subset)
```

Note, you can us the `as` keyword to shorten long imports, which can also make your code more readable and save typing long module paths a number of times.

```python
>>> import rsgislib.imageutils as imgutil
>>> help(imgutil.subset)
```

To exit the help system just press the ‘q’ key on the keyboard.
B.2 Hello World Script

To create your first python script, create a new text file using your preferred text editor and enter the text below:

```python
#! /usr/bin/env python
#
# A simple Hello World Script
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#
print('Hello World')
```

Save your script to file (e.g., helloworld.py) and then run it using the Terminal (UNIX), using the following command:

```bash
> python helloworld.py
Hello World
```

B.3 Comments

In the above script there is a heading detailing the script function, author, and version. These lines are preceded by a hash (#), this tells the interpreter they are comments and are not part of the code. Any line starting with a hash is a comment. Comments are used to annotate the code, all examples in this tutorial use comments to describe the code. It is recommended you use comments in your own code.

B.4 Variables

The key building blocks within all programming languages are variables. Variables allow data to be stored either temperately for use in a single operation or throughout the whole program (global variables). Within python the variable data type does not need to be specified and will be defined by the first assignment. Therefore, if the first assignment to a variable is an integer (i.e., whole number) then that variable will be an integer for the remained of the program. Examples defining variables are provided below:

```python
name = 'Pete'  # String
age = 25  # Integer
height = 6.2  # Float
```
B.4.1 Numbers

There are three types of numbers within python:

**Integers** are the most basic form of number, contain only whole numbers where calculation are automatically rounded to provide whole number answers.

**Decimal** or floating point numbers provide support for storing all those number which do not form a whole number.

**Complex** provide support for complex numbers and are defined as $a + bj$ where $a$ is the real part and $b$ the imaginary part, e.g., $4.5 + 2.5j$ or $4.5 - 2.5j$ or $-4.5 + 2.5j$.

The syntax for defining variables to store these data types is always the same as python resolves the suitable type for the variable. Python allows a mathematical operations to be applied to numbers, listed in Table reftab:maths

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>x + y</td>
<td>x plus y</td>
</tr>
<tr>
<td>x - y</td>
<td>x minus y</td>
</tr>
<tr>
<td>x * y</td>
<td>x multiplied by y</td>
</tr>
<tr>
<td>x / y</td>
<td>x divided by y</td>
</tr>
<tr>
<td>x ** y</td>
<td>x to the power of y</td>
</tr>
<tr>
<td>int(obj)</td>
<td>convert string to int</td>
</tr>
<tr>
<td>long(obj)</td>
<td>convert string to long</td>
</tr>
<tr>
<td>float(obj)</td>
<td>convert string to float</td>
</tr>
<tr>
<td>complex(obj)</td>
<td>convert string to complex</td>
</tr>
<tr>
<td>complex(real, imag)</td>
<td>create complex from real and imaginary components</td>
</tr>
<tr>
<td>abs(num)</td>
<td>returns absolute value</td>
</tr>
<tr>
<td>pow(num1, num2)</td>
<td>raises num1 to num2 power</td>
</tr>
<tr>
<td>round(float, ndig=0)</td>
<td>rounds float to ndig places</td>
</tr>
</tbody>
</table>

B.4.2 Boolean

The boolean data type is the simplest and just stores a true or false value, an example of the syntax is given below:

```python
moveForwards = True
moveBackwards = False
```
B.4.3 Text (Strings)

To store text the string data type is used. Although not a base data type like a float or int a string can be used in the same way. The difference lies in the functions available to manipulate a string are similar to those of an object. A comprehensive list of functions is available for a string is given in the python documentation [http://docs.python.org/lib/string-methods.html](http://docs.python.org/lib/string-methods.html).

To access these functions the string modules needs to be imported as shown in the example below. Copy this example out and save it as StringExamples.py. When you run this script observe the change in the printed output and using the python documentation to identify what each of the functions lstrip(), rstrip() and strip() do.

```python
#! /usr/bin/env python

# Example with strings
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0

import string

stringVariable = ' Hello World '  
print('"' + stringVariable + '"')

stringVariable_lstrip = stringVariable.lstrip()  
print('lstrip: "' + stringVariable_lstrip + '"')

stringVariable_rstrip = stringVariable.rstrip()  
print('rstrip: "' + stringVariable_rstrip + '"')

stringVariable_strip = stringVariable.strip()  
print('strip: "' + stringVariable_strip + '"')
```

B.4.4 Example using Variables

An example script illustrating the use of variables is provided below. It is recommend you copy this script and execute making sure you understand each line. In addition, try making the following changes to the script:

1. Adding your own questions.
2. Including the persons name within the questions.
3. Remove the negative marking.

```python
#! /usr/bin/env python

#########################################
# A simple script illustrating the use of
# variables.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.1
#
# Added helper function for 2/3 compatibility
#########################################

# Compatibility function for 2/3
import sys
def getInput(question):
    """Python 2/3 helper function
    for getting input.
    """
    if sys.version[0] == '3':
        answer = input(question)
    else:
        answer = raw_input(question)
    return(answer)

score = 0 # A variable to store the ongoing score

# print is used to 'print' the text to the command line
print('#################################################')
print('Sample Python program which asks the user a few ' \
'simple questions.')
print('#################################################

# input is used to retrieve user input from the
# command line
name = getInput('What is your name?
')

print('Hello ' + name + '. You will now be asked a series ' \
'of questions please answer \'y\' for YES and \'n\' for ' \
'NO unless otherwise stated.')

print('Question 1:')
answer = getInput('ALOS PALSAR is a L band spaceborne SAR.\n')
if answer == 'y': # test whether the value returned was equal to y
    print('Well done')
score = score + 1 # Add 1 to the score
else: # if not then the anser must be incorrect
    print('Bad Luck')
score = score - 1 # Remove 1 from the score

print('Question 2:')
```

answer = getInput('CASI provides hyperspectral data in ' \ 'the Blue to NIR part of the spectrum.
')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

print('Question 3:')
answer = getInput('HyMap also only provides data in the ' \ 'Blue to NIR part of the spectrum.
')
if answer == 'y':
    print('Bad Luck')
    score = score - 1
else:
    print('Well done')
    score = score + 1

print('Question 4:')
answer = getInput('Landsat is a spaceborne sensor.
')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

print('Question 5:')
answer = getInput('ADS-40 is a high resolution aerial ' \ 'sensor capturing RGB-NIR wavelengths.
')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

print('Question 6:')
answer = getInput('eCognition is an object oriented ' \ 'image analysis software package.
')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

print('Question 7:')
answer = getInput('Adobe Photoshop provides the same ' \ 'functionality as eCognition.
')
if answer == 'y':
    print('Bad Luck')
score = score - 1
else:
    print('Well done')
    score = score + 1

print('Question 8:')
answer = getInput('Python can be executed within the java virtual machine.
')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

print('Question 9:')
answer = getInput('Python is a scripting language not a programming language.
')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

print('Question 10:')
answer = getInput('Aberystwyth is within Mid Wales.
')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

# Finally print out the users final score.
print(name + ' you got a score of ' + str(score))

Note, due to differences between Python 2 and Python 3 a function is defined called 'getInput', which determines uses the appropriate function depending on the version of python being used with input used for Python 3 and raw_input used for Python 2.

B.5 Lists

Each of the data types outlined above only store a single value at anyone time, to store multiple values in a single variable a sequence data type is required. Python offers the List class, which allows any data type to be stored in a sequence and even supports the storage of objects of different types within one list. The string data type is a sequence data type and therefore the same operations are available.
List are very flexible structures and support a number of ways to create, append and remove content from the list, as shown below. Items in the list are numbered consecutively from 0-n, where n is one less than the length of the list.

Additional functions are available for List data types (e.g., len(aList), aList.sort(), aList.reverse()) and these are described in [http://docs.python.org/lib/typesseq.html](http://docs.python.org/lib/typesseq.html) and [http://docs.python.org/lib/typesseq-mutable.html](http://docs.python.org/lib/typesseq-mutable.html)

### B.5.1 List Examples

```python
# Example with lists
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0

# Create List:
aList = list()
anotherList = [1, 2, 3, 4]
emptyList = []

print(aList)
print(anotherList)
print(emptyList)

# Adding data into a List
aList.append('Pete')
aList.append('Dan')
print(aList)

# Updating data in the List
anotherList[2] = 'three'
anotherList[0] = 'one'
print(anotherList)

# Accessing data in the List
print(aList[0])
print(anotherList[0:2])
print(anotherList[2:3])

# Removing data from the List
del anotherList[1]
print(anotherList)
aList.remove('Pete')
print(aList)
```
B.5.2 n-dimensional list

Additionally, n-dimensional lists can be created by inserting lists into a list, a simple example of a 2-d structure is given below. This type of structure can be used to store images (e.g., the example given below would form a grey scale image) and additional list dimensions could be added for additional image bands.

```python
#!/usr/bin/env python

# Example with n-lists
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0

# Create List:
alist = [
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
]

print(alist)
```

B.6 IF-ELSE Statements

As already illustrated in the earlier quiz example the ability to make a decision is key to any software. The basic construct for decision making in most programming and scripting languages are if-else statements. Python uses the following syntax for if-else statements.

```python
if <logic statement>:
    do this if true
else:
    do this

if <logic statement>:
```
do this if true
elif <logic statement>:
    do this if true
elif <logic statement>:
    do this if true
else
    do this

Logic statements result in a true or false value being returned where if a value of true is returned the contents of the if statement will be executed and remaining parts of the statement will be ignored. If a false value is returned then the if part of the statement will be ignored and the next logic statement will be analysis until either one returns a true value or an else statement is reached.

B.6.1 Logic Statements

Table B.3 outlines the main logic statements used within python in addition to these statements functions which return a boolean value can also be used to for decision making, although these will be described in later worksheets.

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equals</td>
<td>expr1 == expr2</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>expr1 &gt; expr2</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>expr1 &lt; expr2</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than and equal to</td>
<td>expr1 ≥ expr2</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than and equal to</td>
<td>expr1 ≤ expr2</td>
</tr>
<tr>
<td>not</td>
<td>logical not</td>
<td>not expr</td>
</tr>
<tr>
<td>and</td>
<td>logical and</td>
<td>expr1 and expr2</td>
</tr>
<tr>
<td>or</td>
<td>logical or</td>
<td>expr1 or expr2</td>
</tr>
<tr>
<td>is</td>
<td>is the same object</td>
<td>expr1 is expr2</td>
</tr>
</tbody>
</table>

B.7 Looping

In addition to the if-else statements for decision making loops provide another key component to writing any program or script. Python offers two forms of loops, while and for. Each can be used interchangeably given the developers preference and available information. Both types are outlined below.
B.7.1 while Loop

The basic syntax of the while loop is very simple (shown below) where a logic statement is used to terminate the loop, when false is returned.

```python
while <logic statement> :
    statements
```

Therefore, during the loop a variable in the logic statement needs to be altered allowing the loop to terminate. Below provides an example of a while loop to count from 0 to 10.

```python
#! /usr/bin/env python

# A simple example of a while loop
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0

count = 0
while count <= 10:
    print(count)
    count = count + 1
```

B.7.2 for Loop

A for loop provides similar functionality to that of a while loop but it provides the counter for termination. The syntax of the for loop is provided below:

```python
for <iter_variable> in <iterable>:
    statements
```

The common application of a for loop is for the iteration of a list and an example if this is given below:

```python
#! /usr/bin/env python

# A simple example of a for loop
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0

aList = ['Pete', 'Richard', 'Johanna', 'Philippa', 'Sam', 'Dan', 'Alex']
```
for name in aList:
    print('Current name is: ' + name)

A more advance example is given below where two for loops are used to iterate through a list of lists.

```python
#!/usr/bin/env python

# Example with for loop and n-lists
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0

# Create List:
aList = [
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,0,1,1,1,1,0,1,1,1,1,1,1],
    [1,1,0,1,1,1,1,0,1,1,1,1,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,0,1,1,1,1,1,1,1],
    [1,1,1,1,1,1,0,1,1,1,1,1,1,1],
    [1,1,1,1,1,0,0,0,1,1,1,1,1,1],
    [1,0,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,0,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,1,0,0,0,0,0,0,0,0,0,0,1,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1]
]

for cList in aList:
    for number in cList:
        # Print with a space at the end
        # rather than a new line
        print(number,end=" ")
    print()
```

B.8 Conclusions

This is a simple introduction to Python and the basics of its syntax. For more information and more advanced examples you can look at the Python tutorial on using Python for spatial data processing of which this material is the first two chapters at https://bitbucket.org/petebunting/python-tutorial-for-spatial-data-processing/downloads. However, there are also many online resources available which can help on this front. Again, perseverance and putting the time in to build your experience is important if you are to become truly proficient.
Within the RIOS library which is used in this tutorial for working with raster attribute tables, but also supports image processing, the Numpy library is heavily used. Numpy provides a format for representing multi-dimensional arrays and provides a number of functions for operating on them.

For more information on Numpy see the documentation (https://docs.scipy.org/doc/numpy/index.html).

If you are familiar with Matlab, the following cheat-sheet may be of use: http://wiki.scipy.org/NumPy_for_Matlab_Users.