IDL Lab #6: The IDL Command Line

Name: ____________________________

IDL Lab #6: A Sub-component of FOR 504 Advanced Topics in Remote Sensing

The objectives of this laboratory exercise are to introduce the student to using IDL via the command line to:

- Enter variables
- Create and manipulate arrays
- View an image

The tasks provided within this lab are designed to help the student better understand the practical details of programming in IDL and will help you prepare for the class assessment.

If you have problems: ASK!

Location: RS/GIS Lab

Login: XXXX
Password: XXXX

Before you start:

Double click the ENVI icon on the desktop:

This starts both ENVI (The Environment for Visualizing Images) and the IDL (Integrated Development Language) programming interface

Ignore the ENVI toolbar but don’t close it as this closes IDL as well.

1. **Interactive and Compiled Mode**

In IDL, you have two choices on how you analyze and interpret data. You can either do as we have in the first five labs and write programs or you can type your data and commands directly into the IDL command line. Writing programs in IDL is described as using it in compiled mode, which allows you to perform complicated analyzing tasks. Using the command line in IDL, is using it in interactive mode, which provides you with quickly visualize and analyze small data sets.

I personally prefer to use IDL in compiled mode as once you have the program compiled and running successfully, you can save the file and use it again another day; or modify the code for a similar task.

However, part of using IDL, is to use the interactive mode via the command line. Therefore, this lab exercise will introduce you to a number of useful interactive commands.

If during these exercises, IDL stops working or you find that in the future, an ENVI commands like say open a new file doesn’t appear to work; then at the IDL/ENVI command prompt type:

```idl
retall
```

This tells IDL to return all memory back to the computers and allow applications to run normally. It is one of those useful commands you come across that you will use all the time.

In this lab I will refer to the IDL command prompt as ENVI> as when you run IDL alongside with ENVI this is what you get at the command line. Running IDL on its own gives the IDL command prompt.

2. **Entering Variables in Interactive Mode**

A useful property of IDL over other high-level programming languages is that you can enter a new variable into IDL at any time. Furthermore, you can create a variable directly without the need to define what type of variable it is (i.e. float, string, etc). For instance, at the IDL/ENVI command prompt type:
ENVI> variable = 1.0
ENVI> help, variable

The first line sets a variable of name ‘variable’, which has a value of 1.0; and as such must be a float. The second line allows you to ‘see’ what the type and value of a variable is.

Another example is in the use of variables made up of a series of letters. These are called ‘strings’: At the IDL/ENVI command prompt type:

ENVI> words = ‘today’
ENVI> help, words

As you can see from the output, you have defined a STRING with value today.

**Task #1**  **Try entering integer values (i.e. 1879, -874, etc) or long integers (>65535) to check that IDL gets it right**

In IDL, you can convert one variable type to another (WARNING if this is from a float to an integer then the information after the decimal point will be lost). For example at the command prompt type:

ENVI> pi = 3.14159
ENVI> help, pi

<table>
<thead>
<tr>
<th>PI</th>
<th>FLOAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.14159</td>
</tr>
</tbody>
</table>

ENVI> pi = long(pi)
ENVI> help, pi

<table>
<thead>
<tr>
<th>PI</th>
<th>LONG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

**Task #2**  **Convert the following:**

- 259 to Bytes
- 3.14159 to Integers  [Hint: the command in not int() but fix()]
- 289 to floats

**Type:**

ENVI> print, float (‘3.2’), float(‘123ABC’) and describe what you get and why.
3. Entering Arrays in Interactive Mode

Using IDL you can enter up to eight dimensional arrays (although I have no idea why you would want an eight dimensional array!).

Creating Arrays

To create an array at the IDL command prompt you need only type:

ENVI> my_array = [1, 2, 3, 4]

This sets up a one-dimensional array with x values 1, 2, 3, 4. Typing: ENVI> help, my_array will show you that this is a one-dimensional integer array of size 4.

There are many other ways to create arrays, For instance, to create an integer array full of one value you could write one of the following commands:

ENVI> zeros = intarr(6)
ENVI> print, zeros

0 0 0 0 0 0

OR

ENVI> ones = MAKE_ARRAY(6, /Integer, value=1)
ENVI> Print, ones

1 1 1 1 1 1

The Make_Array command creates a new array in which several of the array options can be fixed. See the IDL help file for more details.

OR

ENVI> twos = replicate (2.0, 3,3)
ENVI> print, twos

2.00000 2.00000 2.00000
2.00000 2.00000 2.00000
2.00000 2.00000 2.00000

The replicate command creates an array by replicating a scalar (int, float, etc) value in each element of the array size that you specify in the command (3x3 in this example).

To create an array of numbers increasing from 0 upwards to the array size you would use the index generation command:

ENVI> index_arr = indgen(6)
ENVI> print, index_arr

0 1 2 3 4 5

This might prove useful when needing a counter for an array in a program. The index command for other arrays can be found on Table 2.5; page 27 of Gumley (2002).
Converting Array Types

You can also convert array value types using the same commands as in the last task. For instance at the IDL command prompt, write:

```
ENVI> my_array = [1.11, 2.22, 3.33, 4.44, 5.55]
ENVI> help, my_array
```

```
MY_ARRAY        FLOAT     = Array[5]
```

```
ENVI> print, fix(my_array)
```

```
1       2       3       4       5
```

Task #3 Print the following:

Type:

```
ENVI> my_array = [1.99, 2.77, 3.55, 4.33, 5.11]
```

my_array to a long array

my_array with the values rounded to integers [Hint: try round(my_array)]

my_array with the values rounded down to integers [Hint: try floor(my_array)]

my_array with the values rounded up integers [Hint: try ceil(my_array)]

There are a number of words that cannot be used for variable or array names in IDL. These reserved words are typically used for special functions, of which ‘print’ is an obvious example. Others include: `int, intarr, float, case, common, endwhile, pro, repeat, goto, until, xor, end, and, begin, function, etc`. Any command use in IDL cannot be used as a variable or array name.

To create a multi-dimensional array in IDL you use nested square brackets. For instance type:

```
ENVI> multi_arr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
ENVI> help, multi_arr
```

```
MULTI_ARR       INT       = Array[3, 3]
```

```
ENVI> print, multi_arr
```

```
1       2       3
4       5       6
7       8       9
```

Array Indexing

As you can see, we have created a 3x3 integer matrix. Note that the 3rd element of the 1st sub array is at the top right. This corresponds to array element 2,0 where 2 denotes the x value (i.e. the 3rd row) and 0 denotes the y value (i.e. the first column starts at zero). The array elements for this 3x3 array can be written as:

```
A_{0,0}   A_{1,0}   A_{2,0}
A_{0,1}   A_{1,1}   A_{2,1}
A_{0,2}   A_{1,2}   A_{2,2}
```

These subscripts are commonly referred to as array indexing.
It is very important to remember that in IDL the array subscripts refer to the columns and then the rows; and that the numbers always start at zero.

Say you wanted to only print the values along the center diagonal, you then would type:

```
ENVI> print, multi_arr(0,0), multi_arr(1,1), multi_arr(2,2)
```

```
1 5 9
```

To select an index value from an array you can type:

```
ENVI> next_arr = [0, 2, 4, 6, 8, 10]
ENVI> the_one = 5
ENVI> print, next_arr[the_one]
```

```
10
```

Alternatively, you can type:

```
ENVI> print, (next_arr*4)[the_one]
```

```
40
```

A neat trick with arrays is that you can add new values onto an existing array. To do an example of this type:

```
ENVI> growing_arr = [1, 2, 3, 4, 5]
ENVI> print, growing_arr
```

```
1 2 3 4 5
```

```
ENVI> growing_arr = [growing_arr, 5, 6, 7, 8]
ENVI> print, growing_arr
```

```
1 2 3 4 5 5 6 7 8
```

This is particularly nice function, which can be used to great effect in programs if you want to keep adding data to an existing array in a loop.

**Task #4**  For the Following Arrays print both the second and second last index values (twice if the same number). Print these as both integers and floats.

```
first_arr = [23, 4, 56, 2, 98, 1]
second_arr = [1.25, 5.43, 2.98, 7.98, 4.56, 2.79, 3.59, 79.2, 19.0, 47.9, 12.02, 84.29, 49.23]
third_arr = [1.23AC, 3.567, 8787.97, -123.LKUI]
fourth_arr = 2.8* [[1, 4, 6], [5, 8, 9], [3, 8, 2], [9, 5, 2], [9, 4, 8]]
```

For the fourth array above, type:

```
ENVI> print, fourth_arr [0,1]* fourth_arr [0,2]
ENVI> x=1
ENVI> print, fourth_arr [0,1+x]* fourth_arr [0,2+x]
```
The last print statement was an example of how to increments the array values at the command line.

**Task #5** Using your experiences from Lab 3 calculate at the IDL command line the MIN, MAX, and TOTAL of each of the arrays in task 4. Then using the MEAN and STTDEV, commands calculate the mean and standard deviation of each of the arrays in task 4.

4. **The Simple Image Viewer**

Type the following: Remember to replace the pathname with where you have the file from the first lab session saved.

```
ENVI> OPENR, 1, ’C:\ATRS\image1’
ENVI> image = FLTARR(400,400)
ENVI> READU, 1, image
ENVI> tv, image
```

As you can see, the viewing with image one isn’t very helpful as it highlights the bright features so well, you can’t make out main areas of the darker image. To view this image as you might expect to see it type:

```
ENVI> image_equal = HIST_EQUAL(image)
ENVI> tv, image_equal
```

What is wrong with the image you have produced? Can you explain why this is the case? [Hint: Think of how IDL and ENVI subscript arrays – do they do the same thing?]
Now type:

ENVI> slide_image, image

As you can see this produces a useful utility, in which large images can be analyzed more effectively:

Next week, we will return to more IDL programming and start on structures and pointers.