

An Introduction to IDL

Josh Walawender

walawend@starburst.colorado.edu

1. Starting IDL

IDL is a command line interface (just like we use UNIX), therefore we start it up from a terminal by using:

```
cosmos>idl
IDL Version 5.5, Solaris (sunos sparc). (c) 2001, Research Systems, Inc.
Installation number: 100.
Licensed for use by: cosmos.Colorado.EDU
IDL>
```

Now we have an IDL prompt to let us know it will be IDL interpreting what we type instead of UNIX.

Note that IDL is *case insensitive* (unlike UNIX). Therefore, when typing in IDL, the case of the command or variable name doesn't matter.

2. Variables and Data Types

Information in IDL is usually stored in variables. For example, to define the variable `x` as 4.3, we'd type:

```
IDL> x=4.3
```

Then we could print the value of `x` by:

```
IDL> print, x
      4.30000
```

Variables in IDL are classified as one of several types (see Table 2).

Type	Range	Example	Comments
Byte	0 – 255	<code>x=5B</code>	Used for viewing certain images
Integer	-32,768 – +32,767	<code>x=6</code>	
Long Integer	$\pm 2 \times 10^9$	<code>x=15L</code>	Used for large integers
Floating Point	$\pm 10^{38}$	<code>x=12. , x=1.5e7</code>	6 decimal places of significance
Double Precision	$\pm 10^{308}$	<code>x=18D , x=3.3d5</code>	14 decimal places of significance
String	text	<code>x='abcde'</code>	

To determine the variable type of a variable, use:

```
IDL> help, x
X          FLOAT      =      4.30000
```

Which means that x is a floating point variable type (it contains six decimal places of information).

Be sure to keep in mind which variable types you are using. When you do an operation in integers, the result will be an integer. For example, note the difference between the following operations and results:

```
IDL> print, 4/3
      1
IDL> print, 4.0/3.0
      1.33333
```

In the first example, 4 and 3 are integers, so the result is an integer. In the second example, I specified 4.0 and 3.0 making them floating point numbers rather than integers, so that the result would be floating point. You can specify a value as floating point by adding a . or .0 after the number. To specify a floating point number using scientific notation, use `IDL> x=1.25e6`. To specify a value as double precision, use `IDL> x=4.0d` or `IDL> x=1.25d6`. See the examples in Table 2 for examples of how to specify other variable types.

Beware of the upper and lower limits to these numbers, with some variable types you get an error message when the number overflows, sometimes not. For example:

```
IDL> print, 10000+30000
      -25536
IDL> print, 1e56
      Inf
% Program caused arithmetic error: Floating overflow
```

You can change or define a variable or value's data type by using the functions of that variable type: `float`, `double`, etc. For example:

```
IDL> x = 1
IDL> help, x
X          INT        =      1
IDL> y = float(1)
IDL> help, y
Y          FLOAT      =      1.00000
IDL> z = double(1)
IDL> help, z
Z          DOUBLE     =      1.0000000
```

3. Procedures and Functions

IDL has an extensive set of built in commands which are formatted as either procedures or functions. Procedures and functions are called a in different format and have different ways of passing information

into the program (I'll use program as a generic term for either a procedure or a function) and returning any results. The calling format for each is:

```
IDL> procedurename, parameter1, parameter2,..., keyword1=value, keyword2=value, ...
IDL> result = functionname(parameter1, parameter2,..., keyword1=value, keyword2=val
```

We've already seen a procedure: `print` is a simple procedure, with the value you want to print as the input parameter. An example of a function would be the trigonometric sine function:

```
IDL> x = sin(1.1)
IDL> print, x
      0.891207
```

The output of the function is sent to the variable `x`. We can also combine procedures and functions, using the output of a function as the input to a procedure:

```
IDL> print, sin(1.1)
      0.891207
```

Inputs to a procedure or function are either parameters or keywords. Parameters are identified by the program because they are entered in a particular order. Keywords are usually a set of optional inputs which are set by a `keyword=value` format. For example, in the `print` statement, we enter the value to be printed as the first parameter (we can also enter additional values to be printed). Then one of the keywords available for use is the `format` keyword in which we'd use the specific syntax of the procedure to describe the format we wanted printed. For Example,

```
IDL> print, 4.0/3.0, format='("Result = ", F5.3)'
Result = 1.333
```

To get help on the inputs and outputs to a procedure or function, use the on-line help in IDL which is quite extensive. To get help on the `print` procedure, use `IDL> ? print`, then select the "PRINT procedure" option on the menu. IDL's on-line help doesn't require you to know the name of the program you want, you can get help on various topics as well. For example, if you wanted to find out how to plot data, you could use `IDL> ? plot` and see that there is not only a procedure called `plot`, but that there are various other procedures and functions which are related to plotting data.

Some keywords in IDL are used as boolean variables (they're either true or false). In IDL this is done by setting the keyword equal to 0 for false, and any other value for true. The shortcut in IDL is that if you call the keyword with a slash in front of it then it will be set to one. For example, in the `plot` function (which we'll use later) the `isotropic` keyword forces the x and y plot axis to be the same scale, we'd set that by calling `IDL> plot, x, y, /isotropic`. We can also shortcut keywords, by typing just enough of them to make them unique – as long as IDL can tell what you are talking about, so in our previous example, `IDL> plot, x, y, /iso` would also work.

4. Arrays

One of the things which makes IDL so powerful is its built in ability to handle arrays of data. An array is simply a single variable which contains many fields, each of which containing a value. For example, a 3-dimensional position can be an array – it has 3 values: x, y, and z.

Vectors can be defined similar to the variables we've used before, just separate the fields of the array by commas and enclose the array in square brackets. For example:

```
IDL> position=[1.0, 2.0, 3.0]
IDL> print, position
      1.00000      2.00000      3.00000
```

The `position` array we just defined is a one dimensional array. IDL can also do multidimensional arrays, with each row contained in its own set of square brackets:

```
IDL> matrix=[[1.0, 2.0, 3.0],[4.0, 5.0, 6.0],[7.0, 8.0, 9.0]]
IDL> print, matrix
      1.00000      2.00000      3.00000
      4.00000      5.00000      6.00000
      7.00000      8.00000      9.00000
```

Each array can only be of a single variable type (i.e. integer, float, etc.). For example, you cannot define an array which has some elements as floats and some as strings.

To use the value of only one of the array elements, we can index the array with the column and row we want. For example, to only print out the first (element of the `matrix` array we just defined we'd index the array as `matrix[0,0]`. Notice that IDL is a *zero based numbering system* – the first element of an array is the zeroth, not the first.

```
IDL> print, matrix[0,0]
      1.00000
IDL> print, matrix[0,1]
      4.00000
IDL> print, matrix[1,2]
      8.00000
IDL> print, matrix[2,2]
      9.00000
```

We could use a wildcard to print all of the elements of a given row or column, by using `*` as the index. For example:

```
IDL> print, matrix[0,*]
      1.00000
      4.00000
```

```
7.00000
IDL> print, matrix[*,1]
4.00000      5.00000      6.00000
```

You can also generate arrays without typing in each element by using some of IDL's built in functions. If we wanted to generate a two-dimensional, floating point array of all zeros which was 5 columns and 8 rows, we could use:

```
IDL> array = fltarr(5,8)
IDL> help, array
ARRAY          FLOAT      = Array[5, 8]
IDL> print, array
0.00000      0.00000      0.00000      0.00000      0.00000
0.00000      0.00000      0.00000      0.00000      0.00000
0.00000      0.00000      0.00000      0.00000      0.00000
0.00000      0.00000      0.00000      0.00000      0.00000
0.00000      0.00000      0.00000      0.00000      0.00000
0.00000      0.00000      0.00000      0.00000      0.00000
0.00000      0.00000      0.00000      0.00000      0.00000
0.00000      0.00000      0.00000      0.00000      0.00000
```

You can also create arrays with the elements numbered sequentially:

```
IDL> foo = findgen(4,7)
IDL> help, foo
FOO           FLOAT      = Array[4, 7]
IDL> print, foo
0.00000      1.00000      2.00000      3.00000
4.00000      5.00000      6.00000      7.00000
8.00000      9.00000      10.0000     11.0000
12.0000     13.0000     14.0000     15.0000
16.0000     17.0000     18.0000     19.0000
20.0000     21.0000     22.0000     23.0000
24.0000     25.0000     26.0000     27.0000
```

There are similar functions to `fltarr` and `findgen` for the other datatypes (`dblarr`, `dindgen`, etc.).

IDL can also perform array math. The standard operations `+`, `-`, `*`, `/` are all performed element by element. For example:

```
IDL> print, [1.0, 2.0, 3.0] + [2.0, 4.0, 6.0]
3.00000      6.00000      9.00000
```

```
IDL> print, [1.0, 2.0, 3.0] * [2.0, 4.0, 6.0]
      2.00000      8.00000      18.00000
```

To do matrix operations, we use a different set of operators. For example, the ## operator is traditional mathematical matrix multiplication:

```
IDL> foo = findgen(3,3)
IDL> print, foo
      0.00000      1.00000      2.00000
      3.00000      4.00000      5.00000
      6.00000      7.00000      8.00000
IDL> bar = [11.0, 12.0, 13.0]
IDL> print, bar
      11.0000      12.0000      13.0000
IDL> print, foo##bar
      38.0000
      146.000
      254.000
IDL> print, bar##foo
      114.000      150.000      186.000
```

5. Plotting Examples

5.1. Example 1: A Simple Sine Wave

To start, let's make a simple plot of the sine function. First we need to make an array representing the angle, to do this we'll use `findgen`. Let's make an array of 101 elements (with values 0 through 100 in steps of 1). Then we'll divide by 100 to make the array go from 0 to 1 in steps of 0.01. Then we multiply by 2π to make it go from 0 to 6.28 (IDL assumes angles are in radians).

```
IDL> x = findgen(101)/100.0 * 2.0 * 3.14
```

Now we take the sine of this array:

```
IDL> y = sin(x)
IDL> help, x, y
X          FLOAT      = Array[101]
Y          FLOAT      = Array[101]
```

Finally to make a plot, we use the `plot` command and input the parameters `x` and `y`:

```
IDL> plot, x, y
```

This should open a plotting window with a nice sine curve plotted.

5.2. Example 2: A Parametric Plot

As our second example, we will plot a circle by making a parametric plot. Let's call our parametric variable t (for angle theta perhaps).

```
IDL> t = FINDGEN(360)/360.0 * 2.0 * 3.14
```

Now, let's make our x and y positional arrays:

```
IDL> x = sin(t)
```

```
IDL> y = cos(t)
```

Now let's try plotting:

```
IDL> plot, x, y, /isotropic
```