

# Smith/FCAD IDL Cheat Sheet

Rob Gutermuth

## IDL: Interactive Data Language

IDL is a software package for data analysis and visualization from ITT Visual Information Solutions. See <http://www.itvis.com/ProductServices/IDL.aspx> . It is used heavily in modern astronomy research.

### Starting IDL from a Smith College Macintosh computer with IDL installed

Log into a machine using the **ast337** account. Ensure that **Terminal** is running (found on the Dock or in the **Utilities** folder). In **Terminal**, type *ast337idl* . The machine may sync with Rob's server for new software, requiring you to enter the *ast337* password in the Terminal. Now your Terminal should have an IDL> prompt. One can test it by typing *plot,[0,1]* at the IDL> prompt to make a plot window appear.

### Starting IDL from a Linux or Macintosh computer without IDL installed

Make sure X11 is running on your system (Look for X11 icon in the Dock on a Mac). In any terminal (konsole/terminal/xterm in Linux or Terminal/iTerm/xterm on a Mac), type *ssh -X yourusername@urania.ast.smith.edu* to login to urania. In the same window (now logged into urania), type *ast337idl* . Now your terminal should have an IDL> prompt. One can test it by typing *plot,[0,1]* at the IDL> prompt to make a plot window appear.

## Quitting IDL

Enter the command *exit* at the IDL> prompt and then log out of the machine.

## Organization—IDL is composed of...

...variables; IDL> *mydata = [ 4, 5, 1, 2, 3, 4 ]*  
...functions; IDL> *result = mean( mydata )*  
...procedures; IDL> *print, ' The mean of mydata is: ', result*

## UNIX commands in IDL:

You can use any UNIX command from within IDL (when running on a Mac or Linux machine) by prepending a "\$" to the command, such as: IDL> *\$ ls*

## Getting Information from IDL:

The most important IDL command is the **question mark** ("?"). This command brings up the *IDL Hypertext Help* window, with easily searchable documentation for everything IDL. Prepending a question mark to an IDL command or topic name not only brings up the Help window, but also performs a fast search for that text in the Help documents.

Another very useful command is **help**. Enter this command and IDL will print an informative listing of the data currently in memory, by variable name. You can also get information on just a specific variable by entering: IDL> *help, variable\_name* .

**If you want to see all the value(s) stored in a variable** named *mydata* for example, you can print it to the terminal by entering: IDL> *print, mydata*

**As in UNIX, you can navigate the directory structure in IDL** by entering:

IDL> *cd, '/this/is/a/fake/directory/name'*

## Variables and Arrays

IDL has a variety of data types: bytes, short integers, long integers, floating, double, complex, and strings. Variables should start with a character and may continue either with characters, numbers or '\$' and '\_'. Variables cannot be named as IDL commands (e.g, *if*, *else*, *do*).

IDL provides the following data types, among others:

Type	Bytes	Range	Conversion Routine Name	Constant	Array	Index
Byte	1	0-255	byte	0B	bytarr	bindgen
Integer	2	-32768 -- +32767	fix	0	intarr	indgen
32 Bit Signed Long	4	2x +/- 10 <sup>9</sup>	long	0L	lonarr	lindgen
Float	4	+/- 10 <sup>38</sup>	float	0.	fltarr	findgen
Double Precision	8	+/- 10 <sup>308</sup>	double	0.D	dblarr	dindgen
String	0-32767	Text	string	" or ""	strarr	sindgen

**IDL variables and arrays are defined dynamically** and you do not need to declare them at the beginning. For example the type of a variable or array may change several times along an IDL procedure. In principle this an IDL advantage, but **be careful since it is very easy to make mistakes!**

### Conditional Statements

```
if myvalue gt 1 then print, ' Greater than 1. ' else print, ' Less than or equal to 1. '
```

```
case 1 of
```

```
  myvalue gt 0: begin
```

```
    print, ' Greater than or equal to 0. '
```

```
  end
```

```
  else: print, ' Less than 0. '
```

```
endcase
```

```
result = where(myvector eq 3)
```

### Loop Statements

```
for i=1,10 do begin
```

```
  print, ' This is iteration '+strtrim(i,2)+' of 10 in your for loop!'
```

```
endfor
```

```
while i le 1000 do begin
```

```
  i = i*2
```

```
  print, ' Is i = '+strtrim(i,2)+' less than or equal to 1000? If so, your while loop continues!'
```

```
endwhile
```

## Selected example tasks in IDL

**Read the first four columns of an ASCII (text) table** into four IDL arrays named *column1*, *column2*, *column3*, *column4* with format types string (*a*), integer (*i*), float (*f*), and double (*d*) :

```
IDL> readcol, 'filename', column1, column2, column3, column4, format='a, i, f, d'
```

**Read a FITS file's primary image data and header data** into two IDL arrays, *image* and *header*:

```
IDL> fits_read, 'filename.fits', image, header
```

**Element-by-element arithmetic is fast and easy in IDL**, if the arrays involved are the same size:

```
IDL> result = image - image2 ; Subtract one array from another
```

```
IDL> result = image + image2 ; Add one array to another
```

```
IDL> result = image * image2 ; Multiply one array with another
```

```
IDL> result = image / image2 ; Divide one array by another
```

**Display an image** in an IDL Direct Graphics window:

```
IDL> tvimage, bytscl( image - median(image), min=-20, max=200), /keep_aspect, /erase
```

**Open an image in ATV**, a general purpose image visualizer (similar to ds9):

```
IDL> atv, 'filename.fits'
```

**Open PhotVis**, Rob's Photometry Visualization Tool:

```
IDL> photvis
```

**Plot a graph** of the arrays *xdata* and *ydata* using asterisks:

```
IDL> plot, xdata, ydata, psym=2, xtitle='X data', title='My Plot!', xrange=[min(xdata), max(xdata)]
```

**Plot a histogram** of the array *data* with a bin width of 5:

```
IDL> plothist, data, binsize=5
```

**Plot directly to a PostScript file:**

```
IDL> set_plot, 'ps' ; This sets the "IDL display device" to the PostScript device.
```

```
IDL> device, file='mypostscriptfile.ps', /inches, xsize=6.5, ysize = 6, xoffset=1, yoffset=4
```

```
IDL> plot, xdata, ydata, psym=2 ; An example plot. Replace with anything, including image displays.
```

```
IDL> device, /close_file
```

```
IDL> set_plot, 'x' ; Set the "IDL display device" back to IDL Direct Graphics windows.
```

**Extract the contents of an IDL Direct Graphics window and write it to a web-friendly image file:**

```
IDL> plothist, data, binsize=5 ; Another example plot. Replace with anything, as above.
```

```
IDL> myplot = tvrd(true=1) ; The true keyword ensures that color information is extracted!
```

```
IDL> write_png, myplot, 'myplotimage.png' ; PNG is a pretty common image format that is like GIF.
```

```
IDL> write_jpeg, myplot, 'myplotimage.jpg', true=1, quality=100 ; JPEG is a little more complex.
```

## Basic Program Construction

**To make a batch file** (a simple collection of IDL commands), in your favorite text editor, type:

```
; plot_example - A batch file that demonstrates some plotting features.  
; A name and brief description of the program at the top of the file is always helpful.  
; Comments can be added after a semicolon character on any line and IDL will ignore them.  
  
xdata = findgen(10) & ydata = xdata^2  
; Note that you can put more than one valid IDL command on one line using an ampersand ( & ).  
  
plot, xdata, ydata, $           ; Use a dollar sign ( $ ) to continue a single IDL command on the next line.  
title = textoidl(' Y = X^2')   ; Note that end of line comments are perfect for short code lines.  
  
end ; A batch file generally needs an end statement.
```

... and save the document as anything you like, (I use *plot\_example.pro* here). Then you can run this batch file by simply entering: IDL> .run /where/i/put/plot\_example.pro

**To make a minimal IDL procedure** (a program with no explicit output), in your text editor, type:

```
; hello_world - The first program anyone has to write when learning a new programming language!  
pro hello_world  
  
print, ' Hello World! '  
  
; All IDL procedures must have a 'pro' statement and an 'end' statement  
end
```

... and save the document as *hello\_world.pro* . Then you can run this procedure by simply entering: IDL> *hello\_world*

**To make a minimal IDL function** (a program with explicit output), in your text editor, type:

```
; roll_dice - Return the results from ndice simulated six-sided dice rolls. Use ndice = 5 for Yahtzee!  
function roll_dice, ndice  
  
; randomu() returns random numbers between 0 and 1.  
result = fix( randomu( seed, ndice, /uniform) * 6 + 1 )  
print, result  
  
; All IDL functions must return something, so they need a 'return' statement!  
return, result  
  
; Similar to IDL procedures, all IDL functions must have a 'function' and an 'end' statement.  
end
```

... and save the document as *roll\_dice.pro* . Then you can run this procedure by simply entering: IDL> *result = roll\_dice( 5 )*