IDL Reference Card

Commands marked with * are local extensions.

1 Special characters

& combine several statements in one line

; comment character

$ continuation line; shell escape

^ recall commands from history

2 Variables + data types

IDL is case-insensitive: `N` and `n` are the same.

byte: `b=15B` (decimal); `c="17B` (octal); `d='0F'XB` (hexadecimal)

integer: 2 byte. `k=15`; `l="17`; `m='0F'X`

long (int): 4 byte (like Fortran). `N1=15L`; `N2=100000`

float: 4 byte. `ZERO=0.`; `c=0.577215`; `G=6.67e-11`

double precision: 8 byte. `ONE=1.D0`

complex: `z=complex(1,1)/sqrt(2)`

double complex: `z2=dcomplex(1,-1)/sqrt(2)`

string: `s1="T'was brillig"`; `s2='T''was'`

3 Logical operators and min/max

Numerical comparison:
`gt` ($>$), `lt` ($<$), `ge` ($\geq$), `le` ($\leq$).

(Bitwise) logical operators:
`and` ($\wedge$), `or` ($\vee$) `not` ($\neg$), `xor` (exclusive or).

Not logical operators at all:
`a>b` is maximum of `a` and `b` (pointwise for arrays); `a<b` the minumum.

Minumum/maximum value in an array:

```
print, min(a), max(a)
print, minmax*(a)
```

4 Statements + blocks

4.1 if–then–else

simple statement:
```
if (x lt 0) then y=-1
```
simple statement with else branch:
```
if (x lt 0) then y=-1 else y=1
```
if block:
```
if (x lt 0) then begin
  y=-1
endif
```
if block with else branch:
```
if (x lt 0) then begin
  y=-1
endif else begin
  y=1
endelse
```

4.2 for loop

simple statement:
```
for i=0,10 do print, i
```
block form:
```
for i=0,10 do begin
  print, i
endfor
```

Beware of
```
for i=0,100000
```
which will never finish; you need
```
for i=0L,100000
```

4.3 case (multi-comparison) statement

```
case N of
  0: ; do nothing
  1: print, 'One'
  2: begin
    print, 'TWO'
  end
  else: begin
    print, 'No simple number'
  endelse
endcase
```

4.4 while–do / repeat–until

while--do:
```
while (i lt 20) do begin
  i=i+2
endwhile
```

repeat--until:
```
repeat begin
  i=i+2
endrep until (i ge 20)
```

5 Arrays

5.1 Array constructors

brackets: `pows2 = [1.,2,4,8];`
`mat=[[0.,1.],[2.,-3.]]`

indgen: `nn=indgen(10)` (integers 0 to 9)

findgen: `xx=findgen(10)` (floats 0. to 9.)

linspace*: `x = linspace*(-2,2,Nx)`
(`Nx` equidistant points from -2. to 2.)

spread*: `y=linspace*(0,5,Ny)`
`xx=spread*(x,1,Ny) & yy=spread*(y,0,Nx)`
(replicate data in given direction(s))

make_array: `zz=make_array(Nx,Ny)`

rebin: change size, but not rank (dimensionality)

reform: change rank, not size; `reform(x)` removes any degenerate dimensions

5.2 Array slices

If `f` is an array of shape $[20, 17, 9]$, then

- `f[*,*,*]` is `f`
- `f[*,0:5,6:*]` has shape $[20, 6, 3]$
- `f[*,0:5,6]` has shape $[20, 6]$
- `f[*,0,6:*]` has shape $[20, 1, 3]$
- `reform(f[*,0,6:*])` has shape $[20, 3]$
- `f[*,0,6]` has shape $[20]$
- `f[19,0,6]` has shape 1 (is a scalar)

Round brackets: `f(1,2,3)` is the same as `f[1,2,3]`.

Array syntax is much faster than explicit loops.

5.3 Array inquiries

`n_elements(xx)` returns total number of elements (or 0 if xx is undefined).

`size(xx)` returns detailed info:

scalar: [0, *type*, 1]

1d array: [1, N_x, *type*, *n_elements*]

2d array: [2, N_x, N_y, *type*, *n_elements*]

3d array: [3, N_x, N_y, N_z, *type*, *n_elements*]

where *type* is 2 for short integers, 3 for long integers, 4 for floats, 5 for doubles, 6 for complex and 7 for strings.

6 Plotting

6.1 Plotting routines

1-d data: `plot, x, f;`
`oplot, x, g` to plot second curve onto this graph

2-d scalar data: `surface, f, x, y;`
`shade_surf, f, x, y;`
`contour, f, x, y;`
`contourfill*, f, x, y`

2-d vector data: `velovect, vx, vy, x, y`
or (considerable improved)
`wdvelovect*, vx, vy, x, y;`
`vel, vx, vy, x, y`

2-d 3-vector data:
`plot_3d_vect*, vx, vy, vz, x, y`
or simply `plot_3d_vect*, v, x, y`

6.2 Plotting keywords

XRANGE,YRANGE: plotting range $[x_{\min}, x_{\max}]$

XSTYLE,YSTYLE: type of axis (1: strict; 3: add 2%)

TITLE,XTITLE,YTITLE: top title and axes titles

XLOG,YLOG: flag (0/1) for (semi-)logarithmic plotting

PSYM: symbol for data points: 0 (none – connect points with line), 1 (+), 2 (∗), 3 (·), 4 (◇), 5 (△), 6 (□), 7 (×), 8 (user defined), 10 (histogram), With `PSYM=-`*`SYM`*, points are plotted with symbol *`SYM`* and connected by line

LINESTYLE: type of line connecting points: 0 (——), 1 (·····), 2 (– – – –), 3 (–·–·–·), 4 (–···–···–), 5 (— —)

To set a keyword to 1, use it with `/KEYWORD`. Example:

```
plot, x, f, /YLOG, YRANGE=[0.1,10], $
  TITLE='Pressure', PSYM=4, LINE=2
```

6.3 Multiple plots + Window management

Set `!p.multi = [0,N_cols,N_rows]` to combine $N_{\text{cols}} \times N_{\text{rows}}$ plots in one window; set `!p.multi = 0` for single-plot mode.

```
!p.multi=[0,2,3]
for i=0,5 do plot, x, f[i,*]
```

window: create window with a given number:
`window,2`

wset: switch to given window: `wset,0`

6.4 Hardcopy

Switch to *PostScript* device and back with

```
psa* & plot, [0,1] & pse*
```

Keywords:

```
psa*, FILE='1.ps', /LANDSCAPE, THICK=3
or
psa*, /FULLPAGE, /NOPSFONTS
```

7 System variables

Most graphics keywords have corresponding system variables to set default values. E.g.:

```
!p.title = 'Temperature' & !x.range = [0,2]
plot, x, f
```

Use `help, /struct` on `!p`, `!x`, `!y`, `!z` and `!d`. The search path for files is given by `!path`.

7.1 Save/restore graphics state

`save_state*`, `restore_state*` allow to temporarily modify the graphics state:

```
save_state*
  !p.multi=[0,2,2] & !x.range=[0,2]
  for i=0,3 do plot, x, f[i,*]
restore_state*
```

`restore_state*, /full` reverts fully back.

8 Files; running

`@file1` includes file *file1.pro* at cmd line or in script.

.run run a file: `.run file2` runs *file2.pro*

.rnew like `.run`, but clears all variables first

.continue continue after `STOP` or Ctl-c

file1 must have no final `end`; *file2* needs one.

9 Help

9.1 *help* command

`help` – info about all variables

`help, `*`var`* – variable *var*

`help, NAME='x*'` – variables '*x*', '*xx*', '*x1*', etc.

`help, /STRUCT` – structure variables

`help, /RECALL` – command line history

`help, /KEYS` – keyboard settings

`help, /DEVICE` – graphics device

`help, /FUNCTIONS` – compiled functions

`help, /PROCEDURES` – compiled procedures

`help, /SOURCE` – file names for compiled procedures/functions

9.2 Built-in help tool

Available with '`?`' (from within IDL) or '*idlhelp*' (from the Shell). Indispensable, but strictly suboptimal.

9.3 Online docs

Starting point for PDF manuals: `$IDL/docs/onlguide.pdf`, where `$IDL` is something like `/usr/local/rsi/idl` or `/opt/idl/idl_actual`.

9.4 WWW

`http://www.dfanning.com/` (very useful)
`news:comp.lang.idl-pvwave` (newsgroup)

10 Diverse

10.1 Reading formatted data from a file

```
data = input_table*('list.dat')
col1 = data[0,*] & col2=data[1,*]
```

10.2 Adding a legend

```
esrg_legend*, ['Curve 1','Curve 2'], $
  LINE=[0,1], SPOS='tl', /BOX
```

`SPOS` specifies the position (*t*op *l*eft in the example).

10.3 Setting default values

To initialise a variable only if it is undefined, do

```
default*, N, 20
if (n_elements(M) le 0) then M = 20
```