IDL Reference for Beginners

adapted from the corresponding reference card by Wolfgang Dobler distributed under the GNU Free Documentation License.

Introductory remark. IDL is a programming languange very similar to Fortran90. In particular, arithmetic operations can be defined for scalar *and* array variables `a,b,c` without difference, e.g., `a=b+c, a=b*c`.
The most important difference concerns the array-indices *which always start with* `0`!!
All *keywords* can be abbreviated, if unique (`plot,x,y, linestyle=1` is equivalent to `plot,x,y,line=1`).

1. Special characters

&	combine several statements in one line
;	comment character
$	continuation line; shell escape
^	to power: `2^8`

2. Variables + data types

IDL is case-insensitive; `N` and `n` are the same.
integer: 2 byte (from $-2^{15}\dots 2^{15}-1=32767$). `k=15`
long (int): 4 byte (like Fortran). `N1=15L; N2=100000`
float: 4 byte. `ZERO = 0.; c=0.156623; a=1.67e-8`
double precision: 8 byte. `ONE = 1.D0`
complex: `z2 = complex(1.,-1.)/sqrt(2.)`
double complex: `z2 = dcomplex(1.,-1.)/sqrt(2.)`
string: `s1 = 'I''m going'` or `s2 = "I'm going"`

3. Logical operators and min/max

Numerical comparison:
`eq, ne, gt, lt, ge, le`

Minimum/Maximum of scalars
`a=min([b,c])` or `a=b<c`
`d=max([a1,a2])` or `d=a1>a2`
NOTE: `<,>` are minimum/maximum operators in IDL, for array arguments pointwise evaluation.

Minimum/Maximum of arrays
`print,min(a),max(a),min([a,b])`

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
k=i^2
print,k
endfor
```

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

5. Arrays

Note: All array indices start with `0`!

5.1. Array constructors

brackets: `pow2 = [1.,2.,4.,8.]`
indgen: `nn=indgen(10)` (integers 0 to 9)
findgen: `xx=findgen(10)` (floats 0. to 9.)
intarr: `xx=intarr(10)`: 10 elem. array created, all elements set to zero; analogous:
fltarr, dblarr, strarr floating point, double prec. and string array created.

5.2. Array slices

if `f` is an array of shape [20,10], then
`f(*,*)` is f
`f(0:9,*)` has shape [10,10]
`f(*,0)` has shape [20,1]
`f(3,4)` has shape 1 (is a scalar)
Array indices can also be (index) arrays, e.g.

```
x=findgen(10)
b=[5,6]
c=x(b)
```

`c` has shape two with value `[5.,6.]`.

5.3. `where` statement

allows to choose specific elements of an array (very mighty!)

```
x=findgen(10)
b=where(x gt 4. and x lt 9.)
y=x(b)
```

`b` is an *index-array* with elements `[5,6,7,8]`, thus: `y` contains the elements `[5.,6.,7.,8.]`
Remember: indices start with `0`!

5.4. Array inquiries

`n_elements(xx)` return 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] etc.
where type = 2 for short integers, 3 for long integers, 4 for floats, 5 for doubles, 6 for complex, 7 for strings and 9 for double complex.

6. Plotting

6.1. Important plotting routines

1-d data:
`plot,x,y`
`oplot,x,z` overplots 2nd graph
`plot_oi,x,y` x-axis logarithmic
`plot_io,x,y` y-axis logarithmic
`plot_oo,x,y` both axes logarithmic
2-d scalar data:
`surface,f,x,y` 3-d plot `f(x,y)`
`contour,f,x,y` iso-contours of `f` in x-y plane

```
x=findgen(10)
y=findgen(10)
z=fltarr(10,10)
  for i=0,9 do begin
    for k=0,9 do begin
      z(i,k)=x(i)^2+y(k)^2
    endfor
  endfor
surface,z,x,y
window,1
contour,z,x,y
```

6.2. Important plotting keywords

xrange, yrange: plotting range [x$_{\mathrm{min}}$, x$_{\mathrm{max}}$]
title, xtitle, ytitle: top title and axes titles
psym: symbol for data points:
0(default, connect points with line), 1(+), 2(*), 3(.), 4(rhomb), 5(triangle), 6 (square), 7(x), 8(user-defined), 10 (histogram)
linestyle: type of connecting lines:
0(default, bold line), 1(dotted), 2(dashed), 3(dashed-dotted), 4(dashed-dotted-dotted), 5 (long-dashed)

```
plot_io,x,y,xrange=[0,10], $
title= 'Pressure', xtitle='R/Rsun',$
ytitle='log P', line=2
```

6.3. colors

`xloadct` allows to choose various color-tables
`loadct,n` color-table number `n` is chosen
`plot,x,y,color=100` plots graph with color 100 (corresponding to chosen color-table)

6.4. Multiple plots + Window management

Set `!p.multi = [0, Ncol, Nrow]` to combine Ncol*Nrow plots in one window;
(re-)set `!p.multi = 0` for single-plot mode.

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

NOTE: all variables starting with an exclamation mark are system variables (usually *structures*), e.g. `!p, !x, !y, !z, !d` used as default for certain graphics keywords, which can be overwritten by the user

window: create window with a given number:
`window, 1` (new window number `1` is used)
wset: switch to given window: `wset,0`

6.5. Hardcopy

`set_plot,'ps'` switches to postscript device,
default filename *idl.ps*
`device,file='file.ps'` writes to file *file.ps*
`device,file='file.ps',/color` allows for color

when finished with plotting to ps-device
`device, /close` closes file
`set_plot,'x'` switches back to terminal display

7. Reading/writing

7.1. from/to cmd line

`print,a,b,c` prints variables `a,b,c` to cmd line
`read,a,b,c` reads variables `a,b,c` from cmd line

7.2. from/to file

`openr,1,'test.txt'` opens read access to file
test.txt (connected via logical unit `1`)
`openw,1,'test.txt'` opens write access to *test.txt*
`readf,2,a` unformatted read from logical unit `2`
into variable `a`
`printf,3,a` unformatted write of variable `a`
to logical unit `3`
NOTE: formatted write/read in analogy to Fortran.
`close, 4` closes logical unit `4`
`close, /all` closes all open files

8. Procedures

File name should be name of procedure with extension `.pro`. In this case, it can be called from the cmd line or from other procedure(s) **without** prior compilation (`.run`) by a simple call:
`name, argumentlist, keywords.` Example:

```
pro readfile,file,n,x,y
  x=fltarr(n)
  y=x
  openr,1,file
  for i=0,n-1 do begin
    readf,1,a,b
    x(i)=a
    y(i)=b
  endfor
  close,1
end
pro test,file,n
  readfile,file,n,x,y
  plot,x,sin(y)
return
end
```

name of procedure-file: `test.pro`
called (from the command line or another procedure) by
`test,'file',n`
if *file* is the file to be read and `n` the number of lines (x,y-pairs) contained.

NOTE1: order of procedures/functions essential if called without prior `.run`
NOTE2: use of *keywords* described in IDL-documentation or built-in help system.
NOTE3: IDL-`functions` also possible, see docu.

9. Files; running

`@file1` includes the file *file1.pro* at cmd line
or in procedure
`.run file` compiles and runs the file *file.pro*
required only if procedure/file changed while IDL is running
or (sometimes) if more than one procedure/function is present
`.continue` continues the execution of a procedure
after a STOP statement

10. Help

`help` info about *all* variables
`help, var` info about variable `var`
`?` starts built-in help tool
`idlhelp &` starts help tool from OS-shell
all IDL-included procedures and functions described in detail

11. Diverse

`retall` returns to the uppermost layer at
cmd line (IMPORTANT!!)
`spawn` starts new OS-shell (allows, e.g.,
to modify procedure file if erroneous)
`exit` (shell command) returns from OS-shell to IDL
`!pi` returns value of π (`!dpi` double prec.)