Fix Me! **This page is not fully translated, yet. Please help completing the translation.**
(remove this paragraph once the translation is finished)

GDL (IDL) for beginners

The programming language GDL – a free variant of IDL – uses library functions that were specially designed for the analysis of astrophysical data and includes routines to read and write FITS files.

The interpreter is started in the console with:

```
gdl
```

Now enter the GDL program code line by line and the interpreter executes it right away.

However, it's more conveniant to write the program as a file. Open a text editor of your choice (kate, emacs, nedit, vi,...) and write/edit the program file (its file extension should be .pro), then save it and compile it in GDL (in contrast to other languages like Fortran or C this will not create an executable file, because IDL/GDL works differently) using the command

```
.compile myprog.pro
```

Depending on the name given to the program (inside the file as first line: *pro*), e.g.

```
pro prog1
```

the compiling will make the program “prog1” available for use. Execute it by calling

```
prog1
```

in GDL.

Syntax

Attention, in contrast to the regular Linux console capitalization does not matter for commands in GDL, i.e.

```
print, "Hello World!"
PRINT, "Hello World!"
```

and even

```
PrInT, "Hello World!"
```

return the same output. However, for strings and filenames it's different, as usual.

Basics

Calling the interpreter:	gdl
Starting GDL help:	?
Anzeigen aller Variablen:	help
Kompilieren eines eigenen Scripts:	.compile meinprogramm.pro
Ausführen des Scripts in gdl:	meinprogramm
Beenden von GDL:	exit
Erste Zeile:	pro meinprogramm
Letzte Zeile (Ende des Programms):	end

FITS-Dateien

Einlesen eines Fits-Files in eine Variable:

```
bild1 = readfits('filename.fits')
```

Liegt das Bild nicht im aktuellen Verzeichnis muss man hier den ganzen (relativen) Pfad vor dem Dateinamen angeben.

Je nach interner Rechnerarchitektur werden Daten, die mehr als ein Byte umfassen entweder in High- oder Lowbyteordnung abgespeichert. In seltenen Faellen kann es passieren, dass die Standardordnung zwischen Quellsystem und Auswertesystem nicht identisch ist, was sich in (scheinbar) unsinnigen Werten ausdrückt und sich leicht mit einem Blick auf das Bild der FITS-Datei ermitteln laesst. In solchen Faellen muss die Byteordnung konvertiert werden mittels:

```
byteorder, bild1
```

Abspeichern eines Arrays (hier *bild2*) als FITS-Datei:

```
writefits, 'neuerfilename.fits', bild2
```

Beachte: Alle FITS-Befehle benoetigen die (frei verfügbare) Astrolib fuer GDL/IDL.

Ein- und Ausgabe

Schreiben auf die Konsole

```
print, "Hallo ", namevar
```

Mittels Komma getrennt koennen beliebig viele weitere Werte oder Variablen angehaengt werden. Es erfolgt an Ende ein Zeilenumbruch.

Schreiben in eine Datei:

```
openw, handle, 'dateiname.dat'
printf,handle,i,variable1,"Beispieltext",variable2
```

```
close, handle
```

handle muss dabei ein beliebiger, aber eindeutiger Integerwert sein.

Variablen und Felder

Indizierte Variablen beginnen mit dem Index 0. Ein Array mit den Eintraegen 0, 1,... bis MMAX hat somit MMAX + 1 Eintraege

Dimensionierung eines zweidimensionalen Arrays fuer Integer-Zahlen durch:

```
variablenname = intarr(MMAX+1,NMAX+1)
```

In machen Faellen werden groessere Zahlenwerte benoetigt, als der Datentyp Integer hergibt, in diesem Fall kann

```
variablenname = lonarr(MMAX+1,NMAX+1)
```

verwendet werden.

Fuer Fliesskommazahlen (Float) lautet der Befehl

```
variablenname = fltarr(MMAX+1,NMAX+1)
```

Die Anzahl der Eintraege eines Arrays (hier *arrayvar*) kann ueber den Befehl

```
size(arrayvar,/dimensions)
```

ermittelt werden. Allerdings liefert `size` wiederum ein Array mit Informationen, von denen nur ein Wert die Anzahl der Eintraege ist. Um diesen direkt zu ermitteln, sollte der Befehl ergaenzt werden zu

```
anz = (size(arrayvar,/dimensions))(0)
```

Kontrollstrukturen

Schleifen-Syntax:

```
for j=n1,n2 do begin
  ...
endfor
```