Reading / HW

- Chapters 4, 5 (short!), 6 (also short)
- Today covers most of ch 4
- Next lecture will cover most of 5,6
- Current HW (assignment 2) is mostly on Ch 3: numerical derivatives & integrals

Reading Question

```
IDL> x=findgen(5)
IDL> y=max(x,z)
```

What is z?

A) 5

B) 4

C) 0

D) 2

E) None of the above / I don’t know

Directory Structure

- / is the “root directory”
  - it is the “highest level”
  - Your user directories are in /home/astr/ugrad/username/
  - ~ is a shortcut for /home/astr/ugrad/username/

Directory Structure

/home/shared/astr2600

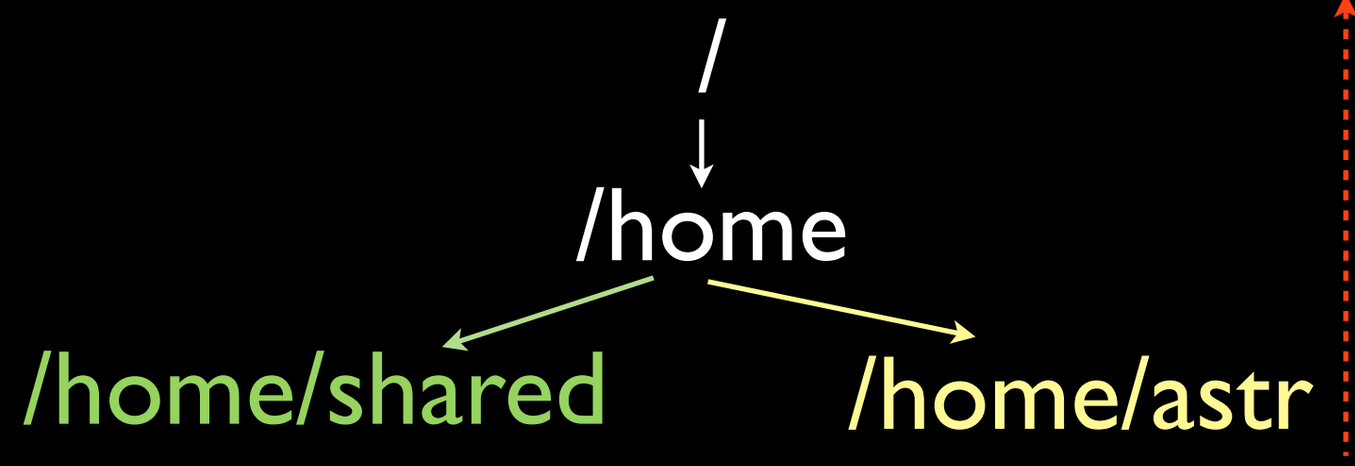
/home/shared/astr2600/data

/home/astr/ugrad

/home/astr/ugrad/username

/home/astr/ugrad/username/ASTR2600

Directory Structure



"cd .." to move up

/home/shared/astr2600

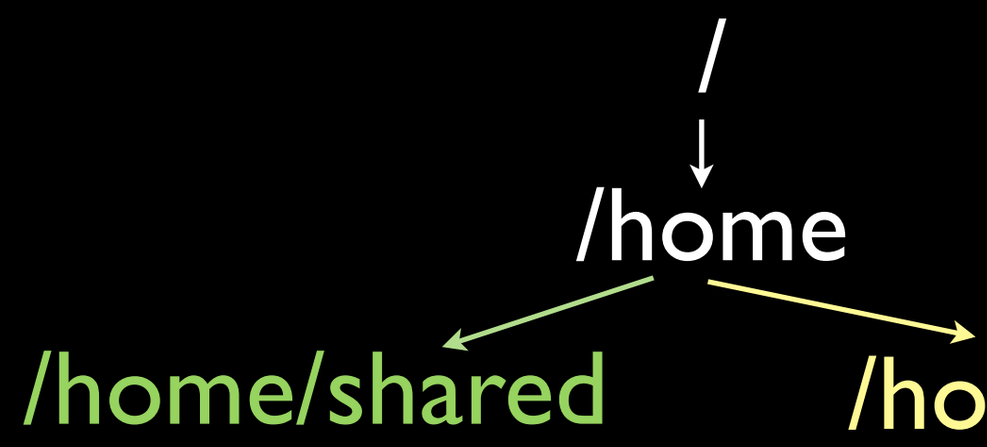
/home/shared/astr2600/data

/home/astr/ugrad

/home/astr/ugrad/username

/home/astr/ugrad/username/ASTR2600

Directory Structure



Just "`cd`"
or "`cd ~`"
to jump home

/home/shared/astr2600

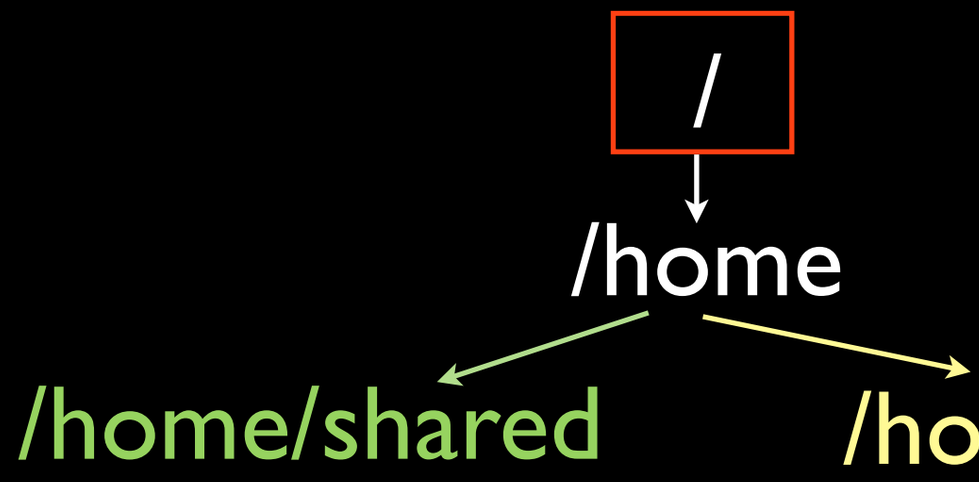
/home/shared/astr2600/data

/home/astr/ugrad

/home/astr/ugrad/username

/home/astr/ugrad/username/ASTR2600

Directory Structure



"`cd /`"
to hop to the top

/home/shared/astr2600

/home/shared/astr2600/data

/home/astr/ugrad

/home/astr/ugrad/username

/home/astr/ugrad/username/ASTR2600

Move command

- `mv [source] [destination]`
  - destination can be a *filename* or a *directory*
    - if it's a filename, the source file will be renamed
    - if it's a directory, the source file will be moved

Move Command Examples

```
cosmos ~/test$ mkdir test_directory
cosmos ~/test$ mv test_file.pro fest_tile.pro # rename the file
cosmos ~/test$ mv fest_tile.pro test_directory # move the file
cosmos ~/test$ ls
test_directory
cosmos ~/test$ ls test_directory/
fest_tile.pro
cosmos ~/test$ mv test_directory/fest_tile.pro test_file.pro
cosmos ~/test$ mv test_file.pro test_directory/
               # move, but it's more clear
```

```
cosmos ~$ mkdir oops
cosmos ~$ cd oops/
cosmos ~/oops$ ls
cosmos ~/oops$ echo "Test Text" > test
cosmos ~/oops$ ls
test
cosmos ~/oops$ mv test subdirectory
```

Will this work? If so, what will I see if I do "`ls -l`" now?

A) Yes:
```
-rw-r--r-- 1 ginsbura grad 10 Sep 11 13:00 subdirectory
```

B) Yes:
```
drwxr-xr-x 2 ginsbura grad 4096 Sep 11 13:00 subdirectory
```

C) No:
```
mv: cannot move `test' to `subdirectory/': Not a directory
```

D) Yes:
```
drwxr-xr-x 2 ginsbura grad 4096 Sep 11 13:00 subdirectory
-rw-r--r-- 1 ginsbura grad   10 Sep 11 13:00 test
```

E) None of the above / I don't know

```
cosmos ~$ mkdir oops
cosmos ~$ cd oops/
cosmos ~/oops$ ls
cosmos ~/oops$ echo "Test Text" > test
cosmos ~/oops$ ls
test
cosmos ~/oops$ mv test subdirectory/
```

Will this work? If so, what will I see if I do "`ls -l`" now?

A) Yes:
```
-rw-r--r-- 1 ginsbura grad 10 Sep 11 13:00 subdirectory
```

B) Yes:
```
drwxr-xr-x 2 ginsbura grad 4096 Sep 11 13:00 subdirectory
```

C) No:
```
mv: cannot move `test' to `subdirectory/': Not a directory
```

D) Yes:
```
drwxr-xr-x 2 ginsbura grad 4096 Sep 11 13:00 subdirectory
-rw-r--r-- 1 ginsbura grad   10 Sep 11 13:00 test
```

E) None of the above / I don't know

Calling Functions and Procedures

Why functions?

- Easier to do things: your work becomes re-useable
- Clearer: you can use action words to describe the function's purpose
- Represent mathematical functions

Example: Planck Function

$$B_\nu(T) = \frac{2h\nu^3}{c^2} \frac{1}{e^{h\nu/(k_\mathrm{B}T)} - 1}$$

- You could put in this IDL command each time:

```
IDL> B = 2*h*nu^3 / c^2 * (exp(h*nu/(kb*T))-1)^(-1)
```

- But that means you have to define h, nu, c, kb, and T each time and keep track of them

Example: Planck Function

$$B_\nu(T) = \frac{2h\nu^3}{c^2} \frac{1}{e^{h\nu/(k_\mathrm{B}T)} - 1}$$

- Makes more sense if you have a function that looks like the math function

```
IDL> B = planck(nu,T)
```

(it's also considerably shorter)

```
IDL> B = 2*h*nu^3 / c^2 * (exp(h*nu/(kb*T))-1)^(-1)
```

Why functions?

- Technically, you could compute sin(x) via a Taylor approximation
  - `sin(x) ~ x-x^3/3!+x^5/5!....`
    - (computers actually use CORDIC....)
- sin(x) is easier to read AND write

What are functions? Procedures?

- Grouped under a general heading of ‘routines’
- In many languages, there are *only* functions
- functions *return* something

Functions & Procedures

- Functions give back a value

```
IDL> x = sin(5)
```

- Procedures do not

```
IDL> plot,x
IDL> print,x
```

IDL v Python

- Python: all routines are functions
  - `lines = plot(x,y)`
- There are also “statements”:
  - `print “this”, that`
  - `def f(x):`
  - `if x:`

Functions and Procedures

- Are not interchangeable!

```
IDL> sin(5)
  ^
% Syntax error.
```

```
IDL> y = plot,x
             ^
% Syntax error.
```

Procedures

- Order always matters

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

- These are not the same (usually)

Procedures

- Can accept a variable number of arguments....
- Both of these are acceptable

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

- `plot,y` is equivalent to `plot,findgen(n_elements(y)),y`

Functions...

- Return things! but just one thing....
  - `x = sin(x)`
  - `data,error = my_function(x)`
    is not a valid expression

What do you do if you want multiple outputs?

- Procedures can modify their inputs
  - technically functions can too but they generally shouldn't
  - Example: a "derivative" procedure could look like:
    - `derivative,y,dy,d2y,d3y,d4y`
    - "y" is an input vector you want the derivative of, while dy,d2y,d3y.... are output vectors

Exceptions to “Rule”

- I said functions shouldn’t change inputs, but, well, that’s not really always true...
  - `x=max(a, wheremax)`
  - `mask=where(y gt 0, howmany)`
- These are called “side effects”
  - In IDL, they are a feature, but they can easily lead to bugs if you don’t document them well

Side Effects cont'd

- `x = max(x,wheremax)` is OK
- `x = max(x,y[1])` is NOT OK - you can't assign a value to an element of an array
- `x = max(x,5)` is also NOT OK - you can't assign anything to a literal number

Using procedures and functions

Which of the following could be a valid expression?

A) `IDL> plot,x`

B) `IDL> y=plot,x`

C) `IDL> y=max(x,5)`

D) `IDL> sin(x)`

E) None of the above / I don’t know

Parameters

- Parameters are the things passed to routines
  - e.g. `plot,x,y` : `x,y` are parameters
- For many routines, parameters must be of a specific type
  - e.g. `x,y` must both be *arrays* for the plot procedure

Comparison to Other Languages

- Not all languages distinguish "procedures" and "functions"
- C++ has a "void" function that returns nothing, but all routines are functions
- python routines are all functions, but they can return "`None`" (a special name for nothing)

Keyword Parameters

- A way to pass extra information to a procedure or function
  - e.g., `plot,x,y,linestyle=2`
- Unfortunately, you can't specify everything as keyword
  - `plot,x=x,y=y,linestyle=2` is not valid

Keyword Parameters

- Common practice:
  - `linestyle = 2`

```
plot,x,y,linestyle=linestyle
oplot,x,y^2,linestyle=linestyle
```

- Nice because you can re-use it, then only change the style once in your code

Keyword Parameters

- Nicer way to do the same:
  - `dashed = 2`
    `plot,x,y,linestyle=dashed`

Keyword Parameters

- These are all acceptable - keyword order does NOT matter
  - `plot,x,y,linestyle=2,color=5`
  - `plot,x,y,color=5,linestyle=2`
  - `plot,color=5,linestyle=2,x,y`

Keyword Parameters

- WARNING: You can use abbreviations for keyword parameters, but this is discouraged.
- `plot,x,y,li=2` is valid, but it is quite unclear what it does
- (it is actually shorthand for `plot,x,y,linestyle=2`)

Keyword Flags

- It is common to use keyword parameters that have values either just “yes” or “no” (1 or 0)

- IDL has a useful shorthand for this:

  - `x = reform(x,2,8,/overwrite)`
    `x = reform(x,2,8,overwrite=1)`

  - are exactly equivalent

Using Keywords

Which of the following is NOT a valid expression?

A) `IDL> plot,linestyle=5,color=6,x`

B) `IDL> plot,x,co=5,l=6`

C) `IDL> plot,x=x,linestyle=6,color=5`

D) `IDL> plot,x,linestyle=6,color=5`

E) None of the above / I don’t know

Non-optional keywords

- Sometimes keywords are required

Keyword Modification

- `regress` solves `y=mx+b`
- `m = regress(x,y,const=b)`
- `b` is an output in this context (it doesn't matter what it is when you call the function, it will be the 'b' value from the equation above afterwards)

SURVEY: How long did Assignment 1 exercises & WDIDs take?

A) < 1 hour

B) ~ 1 hour

C) ~ 2 hours

D) > 2 hours

E) I didn’t do the exercises & WDIDs

SURVEY: How long did Assignment 1 homework take?

A) < 1 hour

B) ~ 1 hour

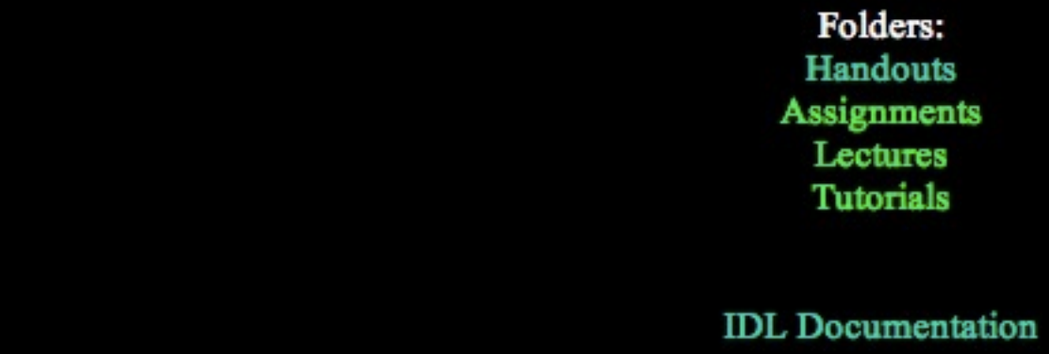
C) ~ 2 hours

D) > 2 hours

E) I didn’t do / haven’t finished the homework (but if you know how long it will take you, answer one of the others)

Documentation

- Linked from the class webpage



- or you can get to a help page from the command line:

```
?print
```

EXELIS
Visual Information Solutions

Welcome to the Exelis VIS product documentation center! Here you will find reference guides, help documents, and product libraries. Discover the products including ENVI, IDL, and E3De, and solutions developed by Exelis VIS at www.exelisvis.com.

Docs Center | IDL Reference | Using IDL | ENVI API | Libraries | Resources

exelisvis.com / Docs Center

Search docs...

Documentation Center

What's New: IDL 8.2.1

Explore: IDL Routines | ENVI API | User Libraries

Learn: Graphics | Image analysis | IDL Concepts | Creating a GUI

Distribute your App: Creating a Distribution | For Free | On Removable Media

Interesting: Creating Video | Map Projections | Detect Edges in Images

Most visited: Using Device Fonts | TEXT | IDLnetURL::GetFtpDirList

Code Library Pick

DLM Builder
This widget application provides a means for writing Dynamically Loadable Module definition .dlm files, along with the appropriate C source code for the corresponding IDL_Load() function.

HELP

Syntax | Arguments | Keywords | Examples | Version History

The HELP procedure gives the user information on many aspects of the current IDL session. The specific area for which help is desired is selected by specifying the appropriate keyword. If no arguments or keywords are specified, the default is to show the current nesting of procedures and functions, all current variables at the current program level, and open files. Only one keyword can be specified at a time.

Syntax

HELP, *Expression*$_1$, ..., *Expression*$_n$ [, /BREAKPOINTS] [, /BRIEF] [, /DEVICE] [, /DLM] [, /FILES] [, /FULL] [, /FUNCTIONS] [, /HEAP_VARIABLES] [, /KEYS] [, /LAST_MESSAGE] [, LEVEL=*value*] [, /MEMORY] [, /MESSAGES] [, NAMES=*string_of_variable_names*] [, /OBJECTS] [, OUTPUT=*variable*] [, /PATH_CACHE] [, /PREFERENCES] [, /PROCEDURES] [, /RECALL_COMMANDS] [, /ROUTINES] [, /SHARED_MEMORY] [, /SOURCE_FILES] [, /STRUCTURES] [, /SYSTEM_VARIABLES] [, /TRACEBACK]

Arguments

Expression(s)

The arguments are interpreted differently depending on the keyword selected. If no keyword is selected, HELP displays basic information for its parameters. For example, to see the type and structure of the variable A, enter:

```
HELP, A
```

Note: Normally, providing an object reference as an argument to HELP prints information about the object heap variable referred to by the object reference. If the argument is an instance of an object class that overloads the IDL_Object::_overloadHelp method, the value returned by that method will be printed.

Keywords

Note that the use of some of the following keywords causes any arguments to HELP to be ignored and HELP provides other types of information instead. If the description of the keyword does not explicitly mention the arguments, the

MAX

Syntax | Return Value | Arguments | Keywords | Examples | Version History | See Also

The MAX function returns the value of the largest element of *Array*.

Syntax

Result = MAX(*Array* [, *Max_Subscript*] [, /ABSOLUTE] [, DIMENSION=*value*] [, MIN=*variable*] [, /NAN] [, SUBSCRIPT_MIN=*variable*])

Return Value

Return the largest array element value. The type of the result is the same as the type of *Array*.

Arguments

Array

The array to be searched.

Max_Subscript

A named variable that, if supplied, is converted to a long integer containing the one-dimensional subscript of the maximum element. Otherwise, the system variable !C is set to the one-dimensional subscript of the maximum element.

What about python?

- Python works much like IDL in terms of keyword arguments
  - ALL python arguments can be specified as keywords
  - instead of `/flag`, python uses `flag=True` (booleans)
  - must use the full keyword name

Lab Tips

- You can load the assignments & tutorials from the web page in lab
- Use tab completion!
- Use the up arrows to recall commands
  - e.g., “mv ^”
- Select = copy, middle-click = paste

Multiple Terminals

- If you ever need to do something at the terminal when you're in IDL, open a new one
  - You can also open multiple tabs in the same terminal
- Better to do this than close IDL, since you'll have to re-start the journal if you do the latter

ctrl-z, bg, fg

- The other way to do something at the terminal is ctrl-z to send IDL to the background, then “fg” to re-start it
- At the IDL prompt, you can run single commands too:
- `IDL> $echo “hi”`

HW, Ex

- Reminder: Follow instructions in the book! The homework is pretty straightforward if you do.
- Don’t turn in incomplete homeworks. It will cost you.