An Introduction to IDL

Instructor:
Joe Munchak
Tuesdays 3:00-4:30 PM
ATS 101

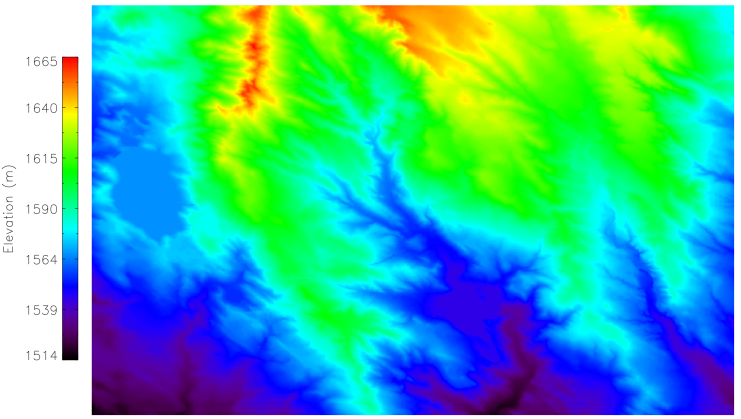
What we'll cover today

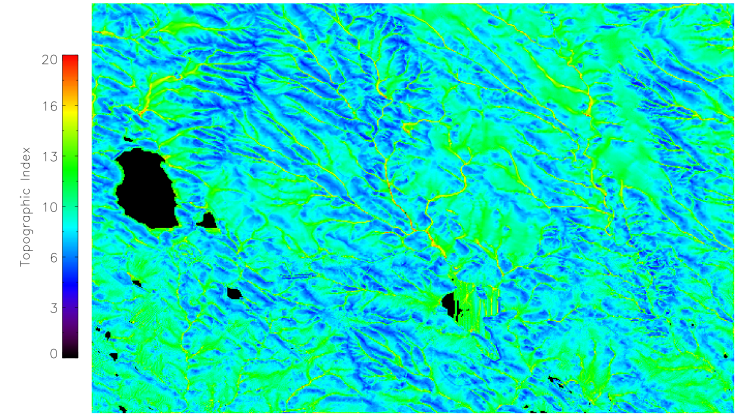
- What is IDL and what it should (and should not) be used for
- How to run IDL from your computer
- Basic syntax (variable types, loops, program control)
- Program structure (procedure and subroutines)
- IDL-specific efficient coding practices
- Common array operations
- String manipulation
- Running other programs from IDL

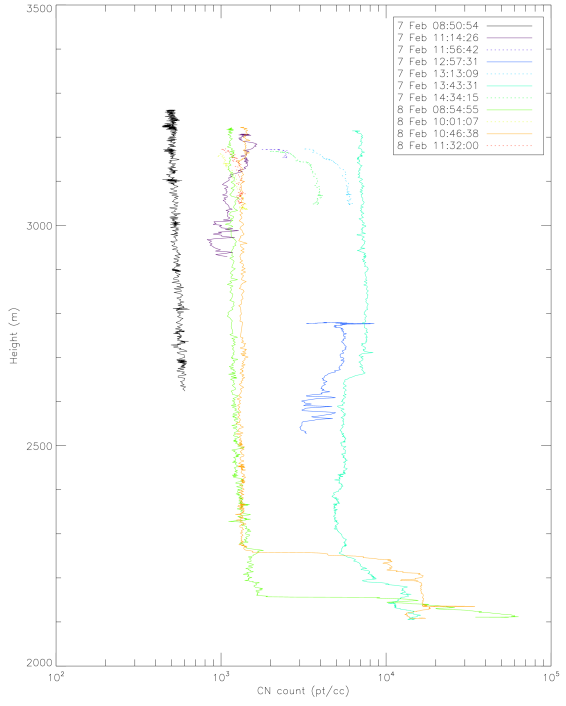
Advantages of IDL

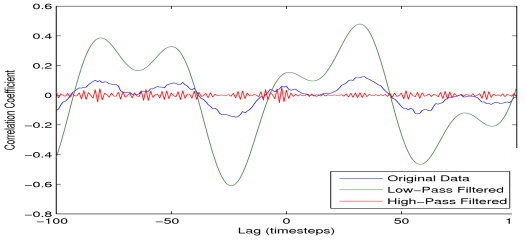
- Optimized for array operations
  - Independent operations on elements of an array will use as many CPUs as you have available...if you write the code correctly!
  - Arrays can be dynamically created and resized
- Enormous library of built-in functionality (so you don't have to re-invent the wheel)
  - Read multiple data formats (e.g., ASCII, binary, HDF)
  - Advanced mathematics and statistics
  - Plotting and visualization
- Much more difficult to seg fault
  - But it can be done

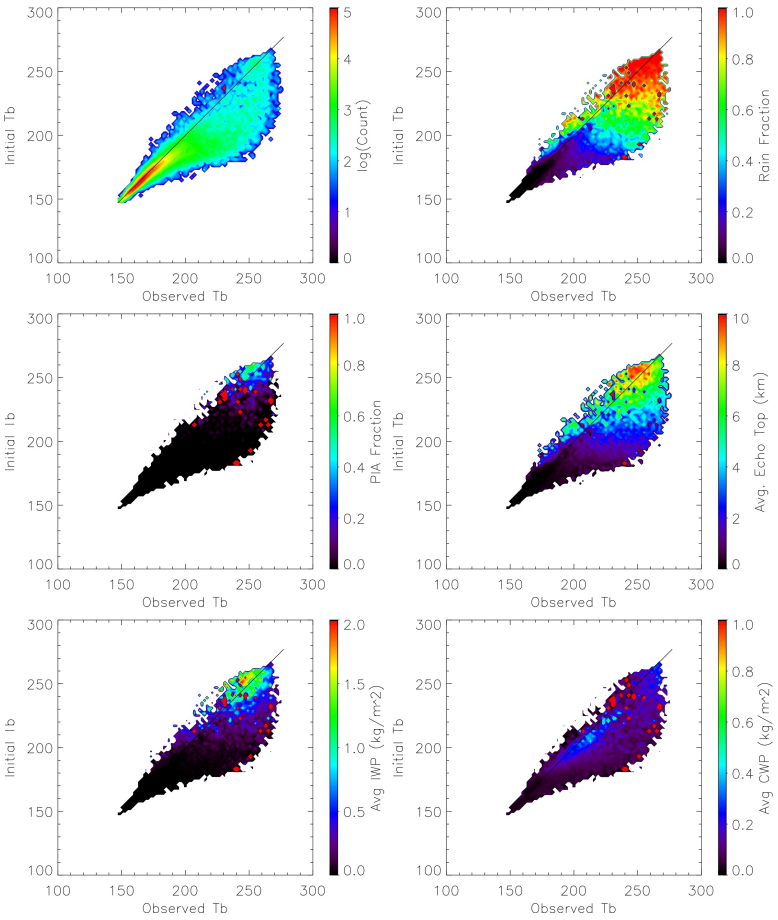
Visualization is limited by your imagination...

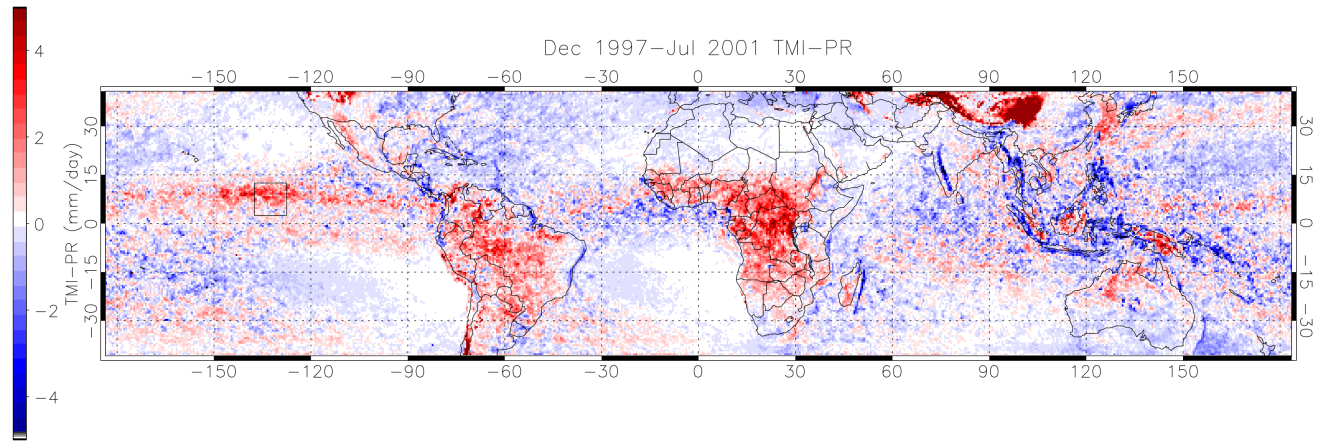


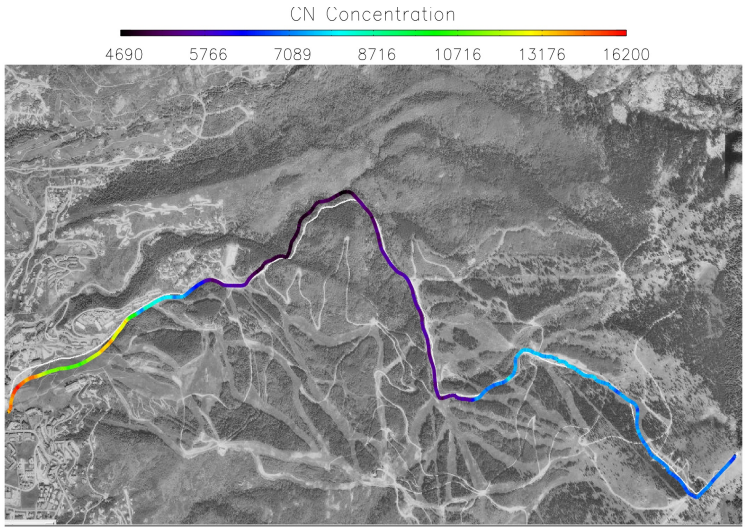






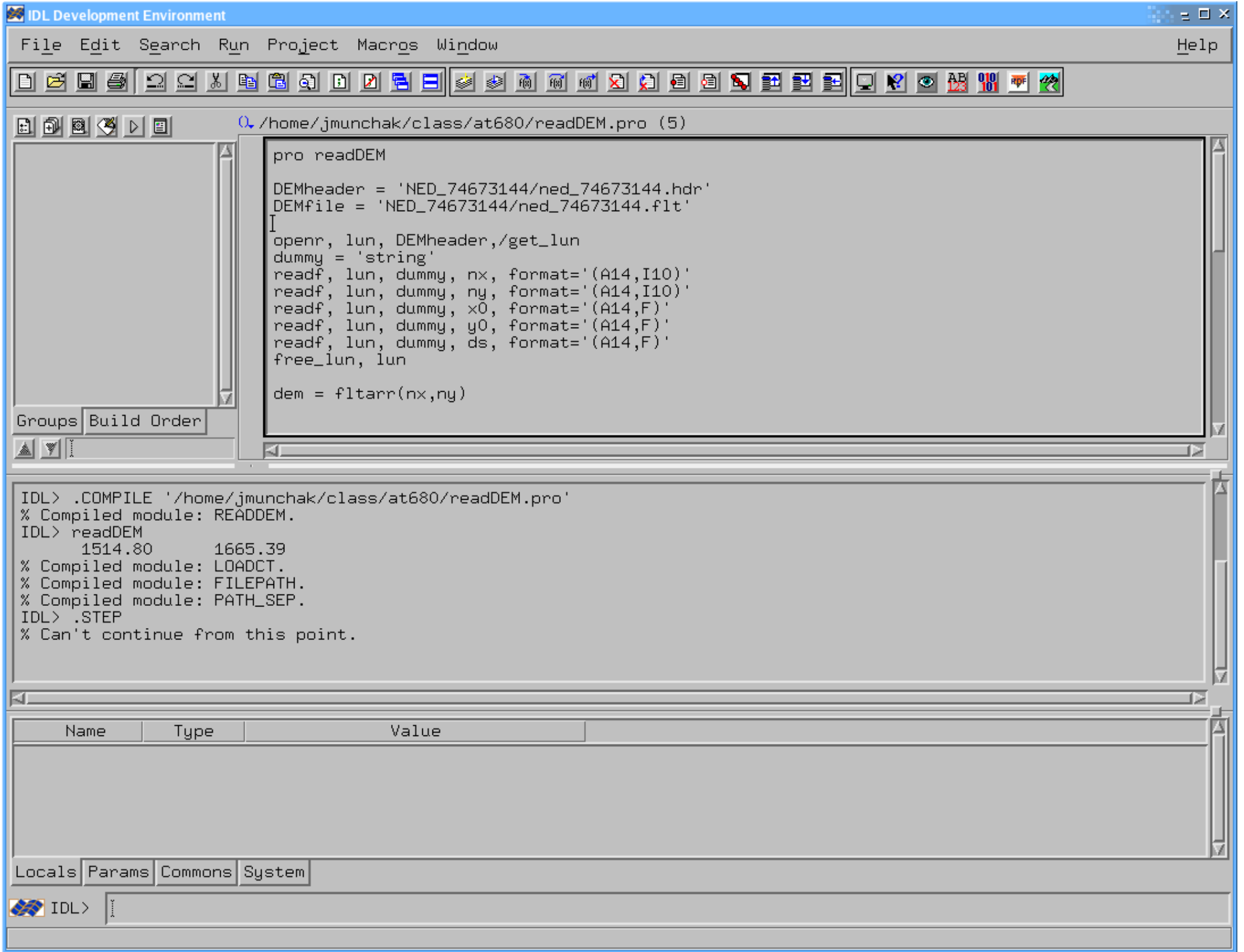


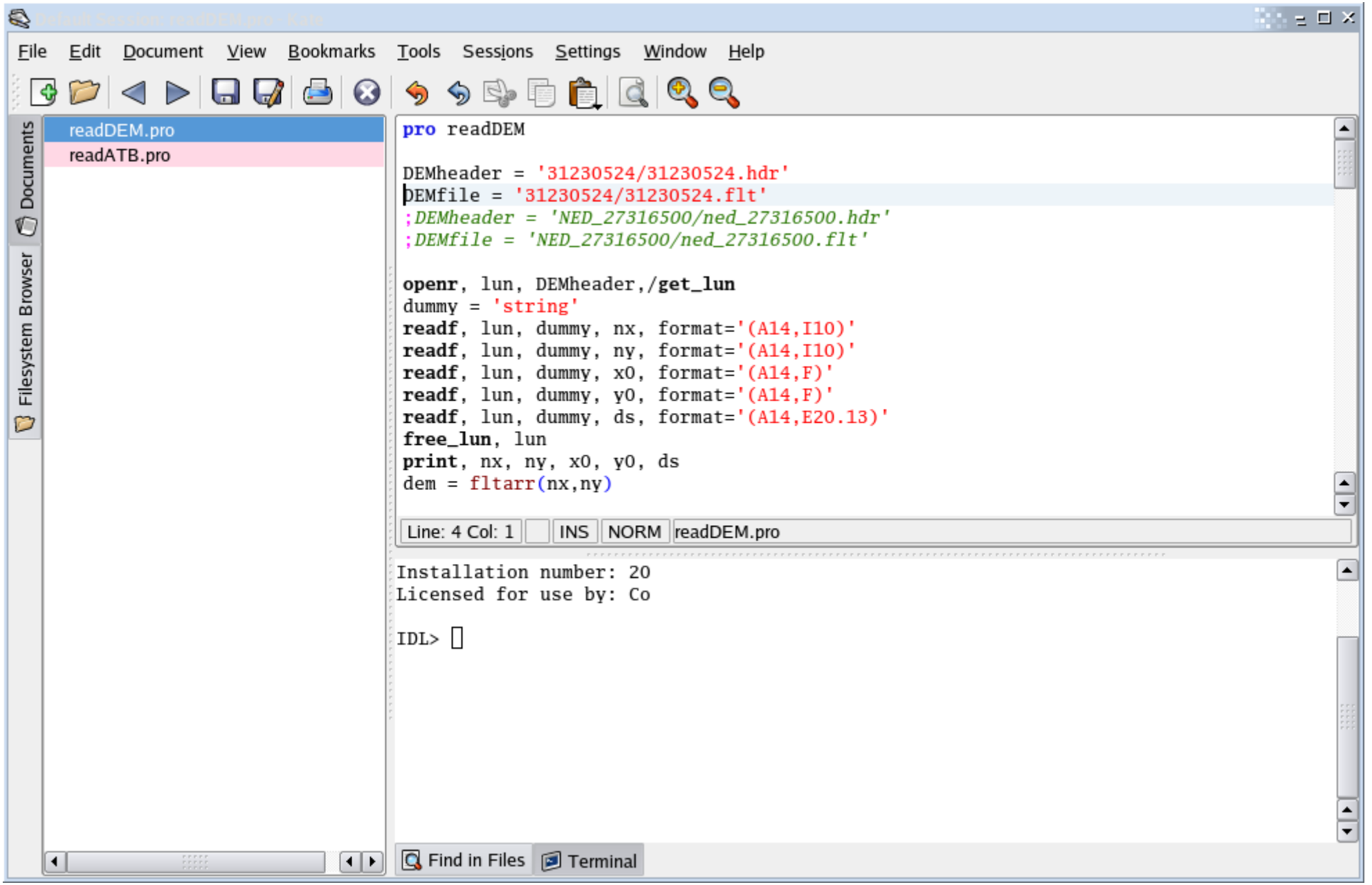




Running IDL

- Make sure that you have it...either on your own machine or a server you can access.
- Two modes available...command-line or IDL Development Environment
  - Mac or Windows...use IDL Development environment (type idlde in xterm)
  - Unix/Linux...both options available (although DE is somewhat clunky)
  - DE offers some debugging tools and support for organizing large projects


IDL Development Environment
File Edit Search Run Project Macros Window Help
/home/jmunchak/class/at680/readDEM.pro (5)
pro readDEM
DEMheader = 'NED_74673144/ned_74673144.hdr'
DEMfile = 'NED_74673144/ned_74673144.flt'
openr, lun, DEMheader,/get_lun
dummy = 'string'
readf, lun, dummy, nx, format='(A14,I10)'
readf, lun, dummy, ny, format='(A14,I10)'
readf, lun, dummy, x0, format='(A14,F)'
readf, lun, dummy, y0, format='(A14,F)'
readf, lun, dummy, ds, format='(A14,F)'
free_lun, lun
dem = fltarr(nx,ny)
Groups Build Order
IDL> .COMPILE '/home/jmunchak/class/at680/readDEM.pro'
% Compiled module: READDEM.
IDL> readDEM
1514.80 1665.39
% Compiled module: LOADCT.
% Compiled module: FILEPATH.
% Compiled module: PATH_SEP.
IDL> .STEP
% Can't continue from this point.
Name Type Value
Locals Params Commons System
IDL>


Default Session: readDEM.pro - Kate
File Edit Document View Bookmarks Tools Sessions Settings Window Help
Documents
Filesystem Browser
readDEM.pro
readATB.pro
pro readDEM
DEMheader = '31230524/31230524.hdr'
DEMfile = '31230524/31230524.flt'
;DEMheader = 'NED_27316500/ned_27316500.hdr'
;DEMfile = 'NED_27316500/ned_27316500.flt'
openr, lun, DEMheader,/get_lun
dummy = 'string'
readf, lun, dummy, nx, format='(A14,I10)'
readf, lun, dummy, ny, format='(A14,I10)'
readf, lun, dummy, x0, format='(A14,F)'
readf, lun, dummy, y0, format='(A14,F)'
readf, lun, dummy, ds, format='(A14,E20.13)'
free_lun, lun
print, nx, ny, x0, y0, ds
dem = fltarr(nx,ny)
Line: 4 Col: 1
INS
NORM
readDEM.pro
Installation number: 20
Licensed for use by: Co
IDL>
Find in Files
Terminal

How to Get Help

- Type '?' at the prompt
- Websites:
  - http://www.dfanning.com/
  - http://astro.berkeley.edu/~jbloom/IDL/
  - http://groups.google.com/group/comp.lang.idl-pvwave/topics?pli=1
  - http://idlastro.gsfc.nasa.gov/idl_html_help/Functional_List_of_IDL_Routines.html
- Books:
  - David Fanning, IDL Programming Techniques, 2nd Edition.
  - Liam E. Gumley, Practical IDL Programming

Interactive vs. compiled mode

- Not to be confused with interface
- Interactive: Type statements one at a time
  - Good for quick calculations and data display
- Compiled: Write programs in a text editor, compile and run all at once
  - Allows for loops and other program control
  - You can always put a **stop** statement in a program to enter interactive mode

IDL executive (dot) commands

.compile(.com) *filename*: compiles a program
.go(.g) *programname*: runs compile program
.run(.r) *filename*: compiles and runs a program
.rnew(.rn) *filename*: clears memory, compiles, and runs a program
.continue: resumes a stopped program
For more, see references. Note that these commands can only be executed from the IDL prompt, not within a program!

Other useful commands:
exit: exits IDL
Ctrl-C: stops the current program running in IDL

Elements of IDL syntax

- Essentially, almost all languages do the same basic things:
  - Store data as variables or arrays
  - Perform calculations on those variables
  - Repeat those calculations (loops)
  - Make logical decisions (if...)
- IDL is no different, however, the syntax is unique (just like every other language)
- IDL is not case-sensitive (except when dealing with file names)
- Comments begin with a semicolon (;)

Variable Types in IDL

Type	Bits	Minimum	Maximum	Precision	Suffix	Conversion	Create
Byte	8	0	255	1	b	byte()	bytarr(), bindgen()
Integer	16	-32768	32767	1		fix()	intarr(), indgen()
Unsigned Integer	16	0	65535	1	u	uint()	uintarr(), uindgen()
Long	32	-2^{31}	$2^{31}-1$	1	l	long()	lonarr(), lindgen()
Unsigned Long	32	0	$2^{32}-1$	1	ul	ulong()	ulonarr(), ulindgen()
64-bit Long	64	-2^{63}	$2^{63}-1$	1	ll	long64()	lon64arr(), l64indgen()
64-bit Unsigned Long	64	0	$2^{64}-1$	1	ull	ulong64()	ulon64arr(), ul64indgen()
Float	32	-10^{38}	10^{38}	7 sig. digits	.	float()	fltarr(), findgen()
Double	64	-10^{308}	10^{308}	14 sig. digits	d	double()	dblarr(), dindgen()
Complex	64	$(-10^{38},-10^{38})$	$(10^{38},10^{38})$	7 sig. digits	$	complex(r,i)	complexarr(), cindgen()
Double Complex	128	$(-10^{308},-10^{308})$	$(10^{308},10^{308})$	14 sig. digits	$	dcomplex(r,i)	dcomplexarr(), dcindgen()
String	8*length				'	string()	strarr(), sindgen()

Variable names:

- Must begin with a letter (but can contain numbers and '_')
- Can be up to 128 characters in length
- Can't be a reserved keyword or built-in command (syntax highlighting is your friend, or type ?<name>)

Example code: variable_types.pro, convert_variables.pro

System Variables

- Always begin with '!'
- Store preferences, information about your system, and useful constants:

```
IDL> print, !PI, format='(F)'
      3.1415927
IDL> print, !DPI, format='(F)'
       3.14159265358979 31
IDL> print, !DTOR, format='(F)'
      0.0174533
```

- We'll learn more as needed

IDL Commands: PRINT and HELP

The **PRINT** command prints the contents (values) of a variable.

Example:

```
IDL> x = intarr(5)
IDL> print, x
       0       0       0       0       0
```

The **HELP** command describes a variable.

Example:

```
IDL> help, x
X               INT       = Array[5]
```

Example code: hello_world.pro

Example: Variable Creation and Conversion

```
IDL> ats_buildings =
  ['Main','ACRC','Chemistry','CIRA','CMMAP','Annex']
IDL> help, ats_buildings
ATS_BUILDINGS   STRING    = Array[6]
IDL> i = 32768
IDL> help, i
I               LONG      =       32768
IDL> j = 32767
IDL> help, j
                J               INT       =    32767
IDL> k = fix(i)
IDL> print, k
 -32768
```

Program Control: *if* statement

IF *expression* **THEN** *statement* **[ELSE** *statement***]**

or

IF *expression* **THEN BEGIN**
statements
ENDIF [ELSE BEGIN
statements
ENDELSE]

Example code: test_if.pro

Logical Expressions

- For integer data types, odd values are evaluated as TRUE and even values FALSE
- For floating point data types, non-zero values are TRUE and zero values are FALSE

Logical Operators/order of operations:

1. **eq, ne, lt, gt, le, ge**
2. **and**
3. **or**

Program Control: *Case* and *Switch*

Useful for executing different statements based on the value of one variable

CASE *variable* **OF**
value1: *statement*
value2: *statement*
value3: **BEGIN**
statements
END
...
ENDCASE

SWITCH: same as **CASE**, but executes all statements below the true expression

Example code: test_case.pro

Program Control: The *FOR* loop

FOR *i=i0*,*imax*[,*increment*] **DO** *statement*
FOR *i=i0*,*imax*[,*increment*] **DO BEGIN**
  statements
  [**BREAK**]
  [**CONTINUE**]
ENDFOR

- Increments can be negative (but not zero)
- Index need not be named i
- Be sure that index variable is a long (or long64) if max iteration exceeds 32767 (**FOR** *i*=0l,*imax*...)
- One-line **FOR** loops can often be replaced by an array operation
- **BREAK** ends the loop
- **CONTINUE** skips to the next iteration

Example code: test_for.pro

Program Control: WHILE **and** REPEAT **loops**

REPEAT *statement* **UNTIL** *expression*
REPEAT BEGIN
  statements
ENDREP UNTIL *expression*

WHILE *expression* **DO** *statement*
WHILE *expression* **DO BEGIN**
  statements
ENDWHILE

Be careful for infinite loops!

Example: Loops and If

```
IDL> for i=0,20 do if(i mod 2 eq 1) then print, i
       1
       3
       5
       7
       9
      11
      13
      15
      17
      19
```

Program Structure

IDL programs consist of **procedures** and **functions**.

Procedure definition:

PRO *name,arg1,arg2,...*
  statements
END

To call a procedure:

name,arg1,arg2,...

Function definition:

FUNCTION *name,arg1,...*
  statements
  RETURN *value*
END

To call a function:

result = name(arg1,...)

Note: Nearly all variables are passed by reference, with a few notable exceptions.

Program Structure: Keywords

In addition to data variables, procedures and functions can take a special type of argument called a **keyword**. Keywords:

- May be listed in any order
- Are always optional (a default value will be assumed if they are not specified)
- Can be set to a single value, vector, or with a slash
- Example:

PLOT,*x*,*y*,***linestyle***=*1*,***xrange***=*[0.5,3.0]*,/***isotropic***

Working with Arrays

- Any type of variable may be put in an array
- Arrays may have up to 8 dimensions
- Arithmetic operations that are independent for each array element may be performed using a compact syntax instead of loops (faster and cleaner code)
- Arrays are initialized to zero
  Example: rdata = fltarr(360,180) creates a 360x180 zero-valued floating point array

Array subscripts

- Array elements are accessed with brackets [], to distinguish from function calls which use parentheses.
- The first element in each dimension is given an index of 0 (not 1)
- To access a range of elements, separate the indices by a colon:
  - Example: print, x[3:6]
- To access all elements in a given dimension, use an asterisk
  - Example: print, x[0,*]

More about arrays

- One of the advantages of IDL over fortran is the ability to dynamically resize arrays. To append an existing array with new data, use the following syntax:
  - Example: x = [x,xnew]
  - Note: This only works if the dimensions are compatible!
- Indexed arrays (starting from 0) can be created using the indgen(), findgen(), or similar commands (see Variable Types slide)
- 2-D array indices are [column,row] – this is important for matrix multiplication and plotting

Useful Array Commands

- **n_elements**() - number of array elements
- **size**() - array size and type info
- **reform**() - reduces number of dimensions without changing the total number of elements
- **reverse**() - reverses the order of one dimension
- **rotate**() - rotates a 1D or 2D array by multiples of 90 degrees
- **transpose**() - reflects array elements about a diagonal
- **sort**() - returns indices of array elements in ascending order

More useful array commands

- **min**(), **max**() - minimum and maximum values (and optionally, index)
- **mean**() - mean value of array
- **variance**() - variance of array values
- **stddev**() - standard deviation of array values
- **moment**() - mean, variance, skew, kurtosis
- **total**() - sum of array values
- **median**() - median array value
- **invert**() - inverts a square (n x n) array
- **round**() - rounds elements to nearest integer
- **ceiling**() - smallest integer > each element
- **floor**() - largest integer < each element

Example – Array Commands

```
IDL> nums = randomn(systime(1),1000)
IDL> print, mean(nums)
% Compiled module: MEAN.
   -0.0539399
IDL> print, stddev(nums)
     0.992279
IDL> print, median(nums)
   -0.0352390
```

Example code: array_character.pro, create_lat_lon.pro

The where command

One of the most useful commands in all of IDL is the **where**() command. **where**() returns the indices of array elements that satisfy a logical expression.

Example: *a* = **where**(*x* gt 0)

Note: For multidimensional arrays, **where**() will still return single-dimensional indices. To convert these to the proper number of dimensions, use the **array_indices**() command.

Example: *indices_2d* = **array_indices**(*array, indices_1d*)

Interactive Example: Where

```
IDL> a = where(nums lt -1 or nums gt 1)
IDL> print, n_elements(a)
         307
IDL> print, n_elements(a)/n_elements(nums)
           0
IDL> print, float(n_elements(a))/n_elements(nums)
     0.307000
IDL> a = where(nums lt -2 or nums gt 2)
IDL> print, float(n_elements(a))/n_elements(nums)
    0.0480000
```

Array Arithmetic

- Standard order of operations applies.
- If any variable in an expression is an array, the result will be an array
- If an expression contains arrays of different sizes, the results will have as many elements as the smallest array in the expression.
- If an expression contains arrays with different numbers of elements and dimensions, the result will have as many elements and as many dimensions as the smallest array
- If an expression contains arrays with the same number of elements but different dimensions, the result array will have as many dimensions as the leftmost array in the expression

Example: Array Arithmentic

```
IDL> a = findgen(3,3)
IDL> print, a
      0.00000      1.00000      2.00000
      3.00000      4.00000      5.00000
      6.00000      7.00000      8.00000
IDL> b = 8.-findgen(3,3)
IDL> print, b
      8.00000      7.00000      6.00000
      5.00000      4.00000      3.00000
      2.00000      1.00000      0.00000
IDL> print, a*b
      0.00000      7.00000      12.0000
      15.0000      16.0000      15.0000
      12.0000      7.00000      0.00000
```

Example code: array_arith.pro

Matrix Multiplication

- IDL offers two matrix multiplication operators:
- $A_{(n,m)}$ # $B_{(m,n)} = C_{(n,n)}$
  - outer dimensions must agree
  - C_{ij} = **total**(A[i,*]*B[*,j])
- $A_{(n,m)}$ ## $B_{(m,n)} = C_{(m,m)}$
  - inner dimensions must agree
  - C_{ij} = **total**(A[*,i]*B[j,*])
  - IDL indices are [column,row] by convention, so you may need to use the **transpose()** function to get the result you want

Example code: array_matrix_multip.pro

Interactive Example: Matrix Multiplication

```
IDL> a = [[2,3],[-0.5,4]]
IDL> a_i = invert(a)
IDL> print, a_i#a
      1.00000  2.98023e-08
      0.00000      1.00000
IDL> print, a_i##a
      1.00000  5.96046e-08
      0.00000      1.00000
IDL> a = findgen(2,3)
IDL> b = findgen(3,2)
IDL> print, a#b
      10.0000      13.0000
      28.0000      40.0000
IDL> print, a##b
      3.00000      4.00000      5.00000
      9.00000      14.0000      19.0000
      15.0000      24.0000      33.0000
```

String processing

- **string**(*variable*,[*format*='(*fmt*)']) - converts numeric variable to string following format code *fmt*
- **strmid**(*s*,*p*,*n*) – result is substring of string *s* beginning at position *p* of length *n*.
- **strpos**(*s*,*u*) – result is position of substring *u* within string *s*
- **strtrim**(*s*) – result removes leading and trailing blanks of string *s*
- **strcompress**(*s*) – result shortens all blank space to length 1
- **file_basename**(*s*) – removes directories from file path *s*
- **file_dirname**(*s*) – removes file name from file path *s*
- *u=s1+s2* concatenates *s1* and *s2* into string *u*

Example code: string_conversions.pro

The spawn command

- There are a number of ways to run other programs from IDL, but the most straightforward is the **spawn** command.
- Syntax: **spawn**, *command*[, *output*[, *error*]]
- *command* is a string
- *output* is an array of strings (one for each line sent to stdout)
- *error* is an array of strings (one for each line sent to stderr)
- If output or error are not specified, they are simply printed to the screen.
- IDL waits for a process to finish unless the command ends with '&' or the */nowait* keyword is specified (in Windows)

Example: Spawn and Strings

```
IDL> spawn, 'ls /cdata1/archive/TRMM_2A25_V6/0001/00010', filelist
IDL> print, n_elements(filelist)
          16
IDL> print, filelist[0]
2A25.000101.12049.6.HDF
IDL> print, strmid(filelist,5,12)
000101.12049 000101.12050 000101.12051 000101.12052 000101.12053
000101.12054 000101.12055 000101.12056 000101.12057 000101.12058
000101.12059 000101.12060 000101.12061 000101.12062 000101.12063
000101.12064
```