BEGINNER IDL TUTORIAL

BY XUAN LIU & AMY COLON
UNDER SUPERVISION OF PROFESSOR SHEILA KANAPPAN

UNIVERSITY OF NORTH CAROLINA - CHAPEL HILL
September 29, 2008

ADDITIONAL SOURCES:


*THIS TUTORIAL ASSUMES YOU ALREADY HAVE EXPERIENCE WITH LINUX AND EMACS*

***WARNING: IDL INDEXES START AT 0 NOT 1***

***WARNING: ERROR CAN RESULT IF OPERATION ON VARIABLES EXCEED INDIVIDUAL VARIABLE’S DATA RANGE***

***WARNING: WHEN COMPARING ARRAY TO SINGLE VALUE, > AND < COMPAR IS EACH INDIVIDUAL ELEMENT OF ARRAY TO SINGLE VALUE***

***WARNING: MATRIX MULTIPLICATION IS NOT EQUIVALENT TO ARRAY MULTIPLICATION***

***WARNING: "NOT" OFTEN DOES NOT BEHAVE WAY YOU WANT IT TO, USE 1-(EXPRESSION) INSTEAD***

***WARNING: WHEN DEALING WITH MULTI-DIMENSIONAL ARRAYS USING INTERPOLATE, MAKE SURE YOU SEPARATE DIMENSIONAL INDICES INTO THEIR OWN ARRAYS (I.E. X-COORDINATE INDICES GO INTO ONE ARRAY, Y-COORDINATES INDICES GO INTO ANOTHER ARRAY)***

***WARNING: FAILURE TO SPECIFY DATA TYPES IS OFTEN THE SOURCE OF ERRORS IN PROGRAMS. RECTIFY THIS BY SPECIFYING DATA TYPE OF VARIABLE AS YOU ARE DEFINING ITY***
***WARNING: CONVERT_COORD BEHAVES DIFFERENTLY FOR EVERY NEW PLOT WINDOW

IDL>a=6
IDL>b=9
IDL>print, (a+b)/2
   7
IDL>a=float(a)
IDL>b=float(b)
IDL>print, (a+b)/2
IDL>print, (a+b)/2
   7.50000

TABLE OF CONTENTS

1.00 Fundamentals of IDL
   1.10 Opening IDL
   1.20 Two modes of IDL
   1.30 Variables
   1.40 Data Types
2.00 The Arrays
   2.10 General Facts
   2.20 Appending Arrays
   2.30 Zero and Index Arrays
2.40 Vector Indexing
2.50 Multi-dimensional Array Indexing

3.00 Operators
  3.10 Precedence
  3.20 Operating on Arrays
  3.30 Important Operators

4.00 Managing Data
  4.10 Where Function
  4.20 Array Manipulation

5.00 Plotting Data
  5.10 Plot Procedure
  5.20 Oplot Procedure
  5.30 Plot Labeling
  5.40 IDL Coordinate Systems
  5.50 Histograms
  5.60 Bar Graphs
  5.70 Contour Plots

6.00 Programming in IDL
  6.10 PROCEDURES AND FUNCTIONS
  6.20 KEYWORDS AND PARAMETERS
  6.30 COMPILING
  6.40 CONTROL STATEMENTS
  6.50 PRACTICE PROGRAMS
1.00 FUNDAMENTALS OF IDL

1.10 OPENING IDL

*Type "idl" on terminal for terminal mode

*Type "idlde" on terminal for developer’s edition

1.20 TWO MODES OF IDL

*Interactive mode designed for simple calculations and commands

------

IDL>a=6
IDL>b=3
IDL>print, a/b

2

------
*Compiled mode designed for programming

=> Process "whe.pro" written with emacs:

------

PRO whe, A1, A2
On_Error, 1

CASE A1+A2 OF
1: print, "wow"
2: print, "hehehehe"
ELSE: print, "you haven’t found the right digits yet"

ENDCASE
END

------

=> whe.pro compiled an run in IDL

------
IDL> .compile whe.pro

% Compiled module: WHE.

IDL> whe,1,2
you haven’t found the right digits yet

IDL> whe,0,1
wow

IDL> whe,0,2
hehehehe

IDL>

-------

1.30 VARIABLES

*SYNTAX: (VARIABLE)=(EXPRESSION)

*Use "help" and "print" to find information

-------

IDL> a=6

IDL> print,a

6
IDL> help, a
A       INT  =  6
IDL>

1.40 DATA TYPES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>RANGE</th>
<th>CONVERSION ROUTINE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>0-255</td>
<td>byte</td>
</tr>
<tr>
<td>String</td>
<td>Text</td>
<td>string</td>
</tr>
<tr>
<td>Integer</td>
<td>-32768 to 32768</td>
<td>fix</td>
</tr>
<tr>
<td>Long</td>
<td>-2^32 to 2^31-1</td>
<td>long</td>
</tr>
<tr>
<td>Positive long</td>
<td>0 to 2^32-1</td>
<td>ulong</td>
</tr>
<tr>
<td>Long</td>
<td>-2^63 to 2^63-1</td>
<td>long64</td>
</tr>
<tr>
<td>Positive long</td>
<td>0 to 2^64-1</td>
<td>ulong64</td>
</tr>
<tr>
<td>Float</td>
<td>-10^38 to 10^38</td>
<td>float</td>
</tr>
<tr>
<td>Double Prec.</td>
<td>-10^308 to 10^308</td>
<td>double</td>
</tr>
<tr>
<td>Complex</td>
<td>(Real-imaginary Pair)</td>
<td>complex</td>
</tr>
</tbody>
</table>
| 2X Prec. Complex (Real-imaginary Prec. Pair) | dcomplex |}

"help" can give you a the data type of a variable
IDL> a=1234567890
IDL> help,a
A   LONG     =  1234567890
IDL> b=123.456
IDL> help,b
B   FLOAT    =  123.456
IDL> c=123456789123456789
IDL> help,c
C   LONG64   =  123456789123456789
IDL>

***WARNING: ERROR CAN RESULT IF OPERATION ON VARIABLES EXCEED INDIVIDUAL VARIABLE’S DATA RANGE

IDL> a=32767
IDL> b=576
IDL> help,a
A    INT    =    32767
IDL> help,b
B    INT    =    576
IDL> print,a+b
-32193
IDL> help,a+b
<Expression>    INT    =    -32193
IDL>

---

**WARNING: FAILURE TO SPECIFY DATA TYPES IS OFTEN THE SOURCE OF ERRORS IN PROGRAMS. RECTIFY THIS BY SPECIFYING DATA TYPE AS YOU ARE DEFINING A VARIABLE

=> Incorrect Method

---

IDL> a=6
IDL> b=9
IDL> print, (a+b)/2

7

=> Correct Method

------

IDL> a=float(6)
IDL> b=float(9)
IDL> print, (a+b)/2

7.50000

IDL>

------

2.00 THE ARRAYS
2.10 GENERAL FACTS

*SYNTAX: VARIABLE=[ARRAY]

*Vector is a one dimensional array (i.e. x-values)

------

IDL> a=[5,6,3]
IDL> print,a
      5  6  3
IDL>

------

*Up to total of 8 dimensions can be created through nested brackets
  (but rarely use any array past 3 dimensions)

=> Two dimensional (i.e. x- corresponds to y-values)

------
IDL> a=[[1,2,3],[6,7,3]]
IDL> print,a
    1    2    3
    6    7    3
IDL>

_______

2.20 APPENDING ARRAYS

*Insert one array into another array as a element

_______

IDL> a=[1,2,3,4,5]
IDL> b=[a,6,7,8,9,10]
IDL> print,b
    1    2    3    4    5    6    7    8    9    10
IDL>

_______
### 2.30 ZERO AND INDEX ARRAYS

<table>
<thead>
<tr>
<th>TYPE</th>
<th>INITIALIZE TO ZERO</th>
<th>INITIALIZE TO INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>bytarr</td>
<td>bindgen</td>
</tr>
<tr>
<td>Integer</td>
<td>intarr</td>
<td>indgen</td>
</tr>
<tr>
<td>Long integer</td>
<td>longarr</td>
<td>lindgen</td>
</tr>
<tr>
<td>Float</td>
<td>fltarr</td>
<td>findgen</td>
</tr>
<tr>
<td>Double Precision Float</td>
<td>dblarr</td>
<td>dindgen</td>
</tr>
<tr>
<td>nComplex</td>
<td>complexarr</td>
<td>cindgen</td>
</tr>
<tr>
<td>Double Prec. Compex</td>
<td>dcomplexarr</td>
<td>dcindgen</td>
</tr>
<tr>
<td>String</td>
<td>strarr</td>
<td>sindgen</td>
</tr>
</tbody>
</table>

*IDL recognizes arrays in COLUMN BY ROW syntax*
IDL> a=bytarr(2,4)
IDL> print,a

0 0
0 0
0 0
0 0

IDL>

------

*Change type of array data with variable type routine names

------

IDL> a=indgen(3,5)
IDL> print,a

0 1 2
3 4 5
6 7 8
9 10 11
12 13 14

IDL> a=float(a)
IDL> print,a

12 13 14
2.40 VECTOR INDEXING

*SYNTAX: ARRAY[INDEX NUMBER]

ARRAY[INDEX RANGE]

IDL> a=indgen(3)*8
IDL> print,a

    0   8  16
IDL> print,a[0]

    0
IDL> print,a[1]

    8
IDL> print, a[0:2]
    0  8  16

IDL> print, a[*]
    0  8  16

IDL> print, a[1:*]
    8  16

IDL> print, a[2:*]
    16

IDL> b=[0, 1, 2]

IDL> print, a[b]
    0  8  16

IDL> c=1

IDL> print, a[c-1:c+1]
    0  8  16

IDL>

-------

***WARNING: IDL INDEXES START AT 0 NOT 1

-------

IDL> a=indgen(4)
IDL> print,a
        0  1  2  3

IDL> print,a[1]
        1

IDL> print,a[0]
        0

IDL>

-------

2.50 MULTI-DIMENSIONAL ARRAY INDEXING

*SYNTAX: ARRAY[COLUMN NUMBER,ROW NUMBER] (TO IDENTIFY BY COLUMN
AND ROW)

-------

IDL> a=indgen(3,3)*2

IDL> print,a
   0   2   4
   6   8  10
  12  14  16

IDL> print,a[0,0]
IDL> print, a[1,1]
   8
IDL> print, a[1,2]
   14
IDL> print, a[0:1, 0:2]
   0  2
   6  8
   12 14
IDL> print, a[0, *]
   0
   6
   12
IDL> print, a[1, *]
   2
   8
   14
IDL> print, a[0:2, 0]
   0  2  4
IDL> print, a[*, 0]
   0  2  4
IDL>
-------
**SYNTAX:** ARRAY[ELEMENT INDEX NUMBER] (TO IDENTIFY BY INDEX NUMBER)

-----

IDL> a=indgen(3,3)

IDL> print,a

```
   0  1  2
   3  4  5
   6  7  8
```

IDL> print,a[0]

0

IDL> print,a[8]

8

IDL> print,a[6]

6

IDL> b=indgen(4,4)

IDL> print,b

```
   0  1  2  3
   4  5  6  7
   8  9 10 11
  12 13 14 15
```
IDL> print, b[a]
          0   1   2
          3   4   5
          6   7   8
IDL> print, a[0:5]
          0   1   2   3   4   5
IDL>

-------

3.00 OPERATORS

3.10 PRECEDENCE

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>MEANING</th>
<th>PRECEDENCE LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Parentheses</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>Pointer dereference</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>Scalar multiplication</td>
<td>3</td>
</tr>
<tr>
<td>##</td>
<td>Matrix multiplication</td>
<td>3</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>3</td>
</tr>
<tr>
<td>mod</td>
<td>Modulus</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>4</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction/negation</td>
<td>4</td>
</tr>
</tbody>
</table>
<  Minimum  4
>  Maximum  4
not  Boolean negation  4
eq  Equal to  5
ne  Not equal to  5
le  Less than or equal to  5
lt  Less than  5
ge  Greater than/equal to  5
gt  Greater than  5
and  Boolean AND  6
or  Boolean OR  6
xor  Boolean exclusive OR  6
?:  Ternary operator  7
=  Assignment  8

*IDL evaluates lower precedence level operators before higher ones

-------

IDL> a=19
IDL> b=19
IDL> print,a+b/2
3.20 OPERATING ON ARRAYS

*Operation on an array by a single element – Change each element of array by given element

```
IDL> a=indgen(4)
IDL> print,a
0   1   2   3
IDL> b=indgen(4)*4
IDL> print,b
0   4   8  12
IDL> c=indgen(4)+5
IDL> print,c
5   6   7   8
```
*Operation on two vectors of the same size - Change each element of array by corresponding element of other array

------

IDL> a=[1,2,3,4,5,6]
IDL> b=[2,3,4,5,6,7]
IDL> print,a+b
   3  5  7  9  11  13
IDL> print,a*b
   2  6 12 20 30 42
IDL>

------

*Operation on two vectors of different sizes - New vector same size as smaller vector
IDL> a=[1,2,3,4,5,6]
IDL> print,a
      1  2  3  4  5  6
IDL> c=[1,2]
IDL> print,c
      1  2
IDL> print,a*c
      1  4

*Resetting elements within an array

IDL> a=[1,2,3,4,5,6]
IDL> a[0:3]=3
IDL> print,a
      3  3  3  3  5  6
IDL>
*Operation on two arrays with same number of elements but different dimensions - New array same dimensions as the first array in the operation

```
IDL> a=indgen(6)
IDL> print,a
     0 1 2 3 4 5
IDL> b=[[4,5,8],[1,4,2]]
IDL> print,b
   4 5 8
   1 4 2
IDL> print,a*b
     0 5 16 3 16 10
IDL> print,b*a
     0 5 16
     3 16 10
IDL> print,a+b
    4 6 10 4 8 7
```
IDL> print,b+a
     4   6   10
     4   8   7

IDL>

------

*Operation on two arrays with different number of elements and dimensions - New vector same dimensions and elements as smallest vector

------

IDL> a=indgen(5)
IDL> b=[[3,4,2],[4,5,2]]
IDL> print,a*b
     0   4   4  12  20

IDL> a=[[0,0],[0,0]]
IDL> print,a
     0   0
     0   0

IDL> print,a+b
     3   4
3.30 IMPORTANT OPERATORS

*Minimum Operator (<) - Returns smaller value between two corresponding elements of two arrays

IDL> a=[5,3,5,3,5]
IDL> b=[5,4,65]
IDL> print,a<b
      5  3  5
IDL>

*Maximum Operator (>) - Returns larger value between two corresponding elements of two arrays
---

IDL> a=[5,3,5,3,51]
IDL> b=[5,4,65]
IDL> print,a>b
    5  4  65
IDL>

---

***WARNING: WHEN COMPARING ARRAY TO SINGLE VALUE, > AND <
COMPARATE EACH INDIVIDUAL ELEMENT OF ARRAY TO SINGLE VALUE

---

IDL> a=[5,4,5,4,51]
IDL> b=[5,4,65]
IDL> z=9
IDL> print,z<a
    5  4  5  4  9
IDL> print,z>a
    9  9  9  9  51
IDL>
*Modulo Operator (mod) - Returns remainder between two elements

IDL> a = indgen(3,2)*2
IDL> b = indgen(2,3)

*Matrix Multiplier Operator (##) - Multiplies two arrays by arithmetic matrix multiplications

IDL> a = indgen(3,2)*2
IDL> b = indgen(2,3)
IDL> print,a
  0  2  4
  6  8 10

IDL> print,b
  0  1
  2  3
  4  5

IDL> print,a##b
  20  26
  56  80

*****

***WARNING: MATRIX MULTIPLICATION IS NOT EQUIVALENT TO ARRAY
MULTIPLICATION

*****

IDL> a = indgen(3,2)*2
IDL> b=indgen(2,3)

IDL> print,a
  0  2  4
  6  8 10
IDL> print, b
    0   1
    2   3
    4   5

IDL> print, a##b
    20  26
    56  80

IDL> print, a*b
    0   2   8
    18  32  50

-------

*Relational Operators - Returns value of 1 if statement is true
and 0 if false

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq</td>
<td>Equal to</td>
</tr>
<tr>
<td>ne</td>
<td>Not equal to</td>
</tr>
<tr>
<td>le</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>lt</td>
<td>Less than</td>
</tr>
<tr>
<td>ge</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>


gt  
Greater than  

------

IDL> print, 5 gt 2
   
1

IDL> print, 10 ge 11
   
0

IDL> print, 5 eq 5
   
1

IDL> a=[4,5,2,3,7,3,1,4]

IDL> b=[1,2,3,4,5,6,7,8]

IDL> print, a gt b
   
1 1 0 0 1 0 0 0

IDL> print, b gt a
   
0 0 1 1 0 1 1 1

IDL>

------

*Boolean Operators - Returns value 1 if given conditions satisfied and 0 if not

OPERATOR     RETURNS 1 IF...
and
  BOTH (A) AND (B) are true
or
  EITHER (A) OR (B) are true
xor
  AND ONLY IF (A) or (B) is true (only one is true)
not
  Statement given is false

_______

IDL> a=(6 gt 5)
IDL> b=(9 gt 8)
IDL> c=(1 gt 5)
IDL> d=(2 gt 9)
IDL> print, a AND b
   1
IDL> print, a AND c
   0
IDL> print, a OR b
   1
IDL> print, a OR c
   1
IDL> print, c OR d
   0
IDL> print, a XOR d
   0
```idl
1
IDL> print, a XOR b
 0
IDL> print, c XOR d
 0
IDL> print, not a
 254
IDL> print, not c
 255
IDL> a=float(a)
IDL> c=float(c)
IDL> print, not a
 0.00000
IDL> print, not c
 1.00000
IDL>

-------

***WARNING: OPERATOR "NOT" OFTEN DOES NOT BEHAVE THE WAY YOU WANT IT TO, USE
1-(EXPRESSION) INSTEAD

-------
```
IDL> a=(6 gt 5)
IDL> b=(1 gt 5)
IDL> print, not a
   254
IDL> print, 1-a
   0
IDL> print, not b
   255
IDL> print, 1-b
   1
IDL>

-------

4.00 MANAGING DATA

4.10 WHERE FUNCTION

*Where - Searches through array for elements matching given
criteria, returning the indices of elements

- SYNTAX: WHERE(CRITERIA)
IDL> a=indgen(3,2)*1.5

IDL> print,a

0.00000  1.50000  3.00000
4.50000  6.00000  7.50000

IDL> print,where(a gt 4)

           3  4  5

IDL> print,a[where(a gt 4)]

4.50000  6.00000  7.50000


*"where" very useful in changing data within an array based on given criteria


IDL> a=indgen(30)

IDL> b=indgen(30)*3-8

IDL> a[where(a mod 2 eq 0)]=0

IDL> print,a
IDL> a = indgen(30)  
IDL> b = indgen(30)*3-8  

0 1 0 3 0 5 0  
7 0 9 0 11 0 13  
0 15 0 17 0 19 0  
21 0 23 0 25 0 27  
0 29  

IDL> a[where(a mod 2 eq 0)] = b[where(b mod 2 eq 0)]  

IDL> print,a  
-8  1  -2  3  4  5  10  
7  16  9  22  11  28  13  
34  15  40  17  46  19  52  
21  58  23  64  25  70  27  
76  29  

IDL>  

-----  

*More efficient alternative to "where" is to use inherent array manipulation characteristic of IDL  

-----  

IDL> a = indgen(30)  
IDL> b = indgen(30)*3-8
IDL> c=b[where(a mod 2 ne 0)] + a[where(a mod 2 eq 0)]

IDL> print,c
    -5   3  11  19  27  35  43
    51   59  67  75  83  91  99
    107

IDL>

-------

4.20 ARRAY MANIPULATION

*Rotate - Rotates arrays in 90 degree increments by the amount suggested

- SYNTAX: ROTATE(ARRAY NAME, NUMBER OF INCREMENTS)

-------

IDL> a=indgen(3,3)

IDL> print,a
    0   1   2
    3   4   5
    6   7   8

IDL> print,rotate(a,1)
    6   3   0
7  4  1
8  5  2
IDL>

------

*Transpose - Reverses index dimensions of each individual element in array (array[a,b] becomes array[b,a])

- SYNTAX: TRANSPOSE(ARRAY NAME)

------

IDL> a=indgen(3,2)
IDL> print,a
IDL> print,a
    0  1  2
    3  4  5
IDL> print,transpose(a)
IDL> print,transpose(a)
    0  3
    1  4
    2  5
IDL>

------
*Shift - Shifts elements or dimensions in array by specified amount of times

- SYNTAX: SHIFT(ARRAY NAME, NUMBER OF ELEMENT SHIFTS)
  SHIFT(ARRAY NAME, NUMBER OF COLUMN SHIFTS,
  NUMBER OF ROW SHIFTS)

------

IDL> a=indgen(3,4)
IDL> print,a

IDL> print,a
  0   1   2
  3   4   5
  6   7   8
  9  10  11

IDL> print,shift(a,1)

IDL> print,shift(a,0,2)

IDL> print,shift(a,0,2)
IDL> print, shift(a, 1, 2)
   8   6   7
  11   9  10
   2   0   1
   5   3   4

IDL>

------

*Sort - Sorts array and returns indices of elements in sorted array
- SYNTAX: SORT(ARRAY)

------

IDL> a=[4, 7, 3, 8, 4, 3, 7, 5, 7]
IDL> print, sort(a)
   2   5   0   4   7
   1   6   8   3
IDL> print, a(sort(a))
   3   3   4   4   5   7   7
*Unique - Returns indices of unique elements in a SORTED array
- SYNTAX: UNIQ(SORTED ARRAY)

IDL> a=[5,3,3,1,3,4,5,1]
IDL> b=a(sort(a))
IDL> print,b
    1  1  3  3  3  4  5
      5
IDL> print,uniq(b)
% Compiled module: UNIQ.
    1  4  5  7
IDL> print,b(uniq(b))
    1  3  4  5

IDL>
*Rebin - Returns new array according to dimensions that are integer multiples of original array dimensions using interpolation
- SYNTAX: REBIN(ARRAY NAME, NEW DIMENSIONS)
- Add keyword "/sample" if nearest-neighbor sampling to be used

------

IDL> a=indgen(3)*10
IDL> print,a
 0   10   20
IDL> print, rebin(a, 6)
 0   5   10   15   20   20
IDL> print, rebin(a, 6, /sample)
 0   0   10   10   20   20
IDL> print, rebin(a, 3, 8, /sample)
 0   10   20
 0   10   20
 0   10   20
 0   10   20
 0   10   20
IDL> y=reform(a,2,3)
IDL> print,y
 0   10   20
 0   10   20
 0   10   20

IDL> print, rebin(a,5)
% REBIN: Result dimensions must be integer factor of 
   original dimensions
% Execution halted at: $MAIN$

IDL>

------

*Reform - Returns new array with same elements organized under 
   different dimensions
   - SYNTAX: REFORM(ARRAY NAME, NEW DIMENSIONS)

------

IDL> a=indgen(6)
IDL> print,a
IDL> print,a
  0    1    2    3    4    5
IDL> y=reform(a,2,3)
IDL> print,y
  0    1
*Interpolate - Returns new array whose elements have been specified
- SYNTAX: INTERPOLATE(ARRAY NAME, INDEX LOCATION)

IDL> a=indgen(6)*1.0
IDL> print,a
        0.00000   1.00000   2.00000   3.00000
        4.00000   5.00000
IDL> b=[0.5,1.5,2.5,3.5]
IDL> print,interpolate(a,b)
        0.500000   1.50000   2.50000   3.50000
IDL> a=indgen(3,2)*10
IDL> print,a
        0   10   20
        30   40   50
IDL> print,interpolate(a,1,1)
        40
IDL> print,interpolate(a,2,1)
        50
IDL> print,interpolate(a,0,0)
         0
IDL> print,interpolate(a,0.5,0.5)
         20

***WARNING: WHEN DEALING WITH MULTI-DIMENSIONAL ARRAYS USING
INTERPOLATE, MAKE SURE YOU SEPARATE DIMENSIONAL INDICES INTO
THEIR OWN ARRAYS (I.E. X-COORDINATE INDICES GO INTO ONE
ARRAY, Y-COORDINATES INDICES GO INTO ANOTHER ARRAY)

      ------

IDL> a=indgen(4,4)*10
IDL> print,a
        0    10    20    30
        40    50    60    70
5.00 PLOTTING DATA

5.10 PLOT PROCEDURE

*If two arrays are defined PLOT plots values in one array with respective value in second array

*If only one array is defined PLOT plots values of array against index number of values

*SYNTAX: PLOT, ARRAY1, ARRAY2, KEYWORD(S)
IDL>x=findgen(101)*(.01*2*!pi)
IDL>y=sin(x)
IDL>plot,x,y
IDL>plot,x

*POSITION Keyword - Sets size of graph on window in normal coordinates
  - x-values range from X1 to X2; y-values range from Y1 to Y2
  - SYNTAX: POSITION=[X1,Y1,X2,Y2]

-----

IDL>a=indgen(10)
IDL>plot, a, position=[0,0,0.5,1]

*PSYM Keyword - Change visual appearance of graph
  - Values range from -1 to 8 with negative values
being counterparts to positive values but
connected with lines

-----

IDL>x=findgen(100)*(2*pi/100)
IDL>y=sin(x)
IDL>plot,x,y,psym=5,position=[0,0,0.5,1]
IDL>plot,x,y,psym=4,position=[0.5,0,1,1], /noerase

-----

*Other useful keywords

<table>
<thead>
<tr>
<th>KEYWORD</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Title string</td>
</tr>
<tr>
<td>/xlog</td>
<td>Create logarithmic x-axis</td>
</tr>
<tr>
<td>/ylog</td>
<td>Create logarithmic y-axis</td>
</tr>
<tr>
<td>/noerase</td>
<td>Don’t erase window before plotting</td>
</tr>
<tr>
<td>/nodata</td>
<td>Create axes only</td>
</tr>
<tr>
<td>[xyz]title</td>
<td>Axis title string</td>
</tr>
<tr>
<td>[xyz]ticks</td>
<td>Number of major tick intervals</td>
</tr>
</tbody>
</table>
---

IDL>x=findgen(100)*(2*!pi/100)
IDL>y=sin(x)
IDL>plot,x,y,title="Sin Graph",xtitle="X-Values", ytitle="Sin(X)"
IDL>plot,x,cos(x),psym=5

---

5.20 OPlot Procedure

*Overplots additional graphs on same axis constructed by "plot"

*Keywords nearly same as "plot"

*Syntax: OPlot, ARRAY1, ARRAY2

---

IDL>x=findgen(101)*(.01*2*!pi)
IDL>y=sin(x)
IDL>plot,x,y
IDL>oplot,x

5.30 PLOT LABELING

*SYNTAX: XYOUTS, X POSITION IN NORMAL COORDINATES,
          Y POSITION IN NORMAL COORDINATE, LABEL

IDL>x=findgen(200)*(0.1)
IDL>y=sin(x)
IDL>plot, x, y
IDL>xyouts,0,0, systime()
*Normal Coordinates - Coordinates dictating position in the plot
   window
   - x and y range from 0 to 1; corners
     (0,0),(0,1),(1,1),(1,0)

*Device Coordinates - Coordinates dictating the pixel size of
   the window; primarily used for image
   manipulation

*CONVERT_COORD - Convert coordinate form one coordinate system
to another
   - SYNTAX: CONVERT_COORD(X,Y,/TO_[DATA,NORMAL,DEVICE])

-------

IDL> x=1
IDL> y=1
IDL> print,x,y
     1   1
IDL> convert=convert_coord(x,y,/to_normal)
IDL> print,convert
***WARNING: CONVERTCOORD BEHAVES DIFFERENTLY FOR EVERY NEW PLOT WINDOW

*PLOTS - Plot a line graph in one of three coordinate systems
    - SYNTAX: PLOTS, ARRAY1, ARRAY2, /DEVICE,NORMAL,DATA

5.50 HISTOGRAMS

*By default, histogram has bin size of 1.0 with first bin
starting at minimum data value and last bin straddling maximum data

*SYNTAX: HISTOGRAM(ARRAY)

<table>
<thead>
<tr>
<th>KEYWORD</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>binsize=</td>
<td>Sets the size of the histogram’s bins</td>
</tr>
<tr>
<td>normalize=</td>
<td>Normalizes the histogram</td>
</tr>
<tr>
<td>fill=</td>
<td>Fills the plotted histogram</td>
</tr>
<tr>
<td>_extra(keyword)</td>
<td>Permits use of &quot;plot&quot; keywords</td>
</tr>
</tbody>
</table>
IDL> a=[2,4,6,8,9,3]
IDL> print,histogram(a)

    1   1   1   0   1   0
    1   1

IDL>

5.60 BAR PLOTS

*SYNTAX: BAR_PLOT, ARRAY1, BARNAMES=ARRAY2

<table>
<thead>
<tr>
<th>KEYWORD</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>title=</td>
<td>Gives title to bar graph</td>
</tr>
<tr>
<td>xtitle=</td>
<td>Gives title to x-axis</td>
</tr>
<tr>
<td>ytitle=</td>
<td>Gives title to y-axis</td>
</tr>
<tr>
<td>colors=</td>
<td>Select a color for each bar</td>
</tr>
<tr>
<td>outline</td>
<td>Creates outline around each bar</td>
</tr>
</tbody>
</table>
IDL> b=[4,2,6,4,8,4,7,3]
IDL> a=indgen(5)
IDL> bar_plot,a,barnames=b
% Compiled module: BAR_PLOT.
IDL>

5.70 CONTOUR PLOTS

"contour" plots elements in a two-dimensional array against their indices in each dimension
Optional parameters X-AXIS and Y-AXIS to change default index axis
SYNTAX: CONTOUR, 2 DIMENSIONAL ARRAY, X-AXIS, Y-AXIS

IDL> x=findgen(100)-54.0
IDL> y=findgen(100)+45.0
IDL> z=dist(100)
% Compiled module: DIST.
IDL> contour,z
IDL> contour,z,x,y
IDL>

-------

6.00 PROGRAMMING IN IDL

6.10 PROCEDURES AND FUNCTIONS

*Procedures begins with PRO statement and ends with END

PRO PROCEDURE NAME, PARAMETERS, KEYWORDS
(Statements...)
END

*Functions begins with FUNCTION statement, ends with END
statement, and must contain a RETURN value to give back to
caller

FUNCTION FUNCTION NAME, PARAMETER, KEYWORDS
(Statements...)
RETURN (Statement)
END

*Main programs is a list of IDL commands ending with END; does not require inputs

(Statements...)
END

6.20 KEYWORDS AND PARAMETERS

*Parameters are the inputs to a procedure or function

*EXAMPLE: Procedure that graphs sin(X) has parameters X and Y

PRO GRAPH_SINX, X, Y

*Keywords are optional inputs to a procedure or function;
replace argument for which default is otherwise defined

*EXAMPLE: Procedure that graphs sin(X) has parameters X and Y
with optional maximum possible value, MAXVAL

PRO GRAPH_SINX, X, Y, MAXVAL=MAXVAL
6.30 COMPILING

"Self-constructed IDL source files generally end with ".pro"
".pro" files evoked on IDL by ".compile" executive command

------

IDL> .compile vgfield.pro
 % Compiled module: VGFIELD.
IDL>

------

".run" is another compiling command that additionally runs the program if it is a main program

6.40 CONTROL STATEMENTS

*IF Statement - If "Condition" is true, execute "Statement 1"
 - Optional: Otherwise execute "Statement 2"

IF (Condition) THEN (Statement 1) ELSE (Statement 2)
IF (Condition) THEN BEGIN (Statement 1) ENDIF ELSE BEGIN (Statement 2) ENDELSE

*IF Statement Alternative - More efficient

-------

IDL> logicalVariable=2
IDL> x=5
IDL> if logicalVariable eq 1 then print, x
IDL> if logicalVariable eq 2 then print, 8
     8
IDL> y=x*(logicalVariable eq 1)+8*(logicalVariable eq 2)
IDL> print,y
     8
IDL>

-------

*CASE Statement - Continues going through (Expression) until it
is equal to (Condition) and executes
corresponding statements
- Continues going through conditions until (Cond) true (eq 1) and executes corresponding statements
- Required: execute ELSE if never true

CASE (Condition) OF
    (Expression1): (Statements)
    (Expression2): (Statements)
    ...
    (ExpressionN): BEGIN
        (Statements)
    END
ELSE: (Statements)
ENDCASE

CASE 1 OF
    (Cond1): (Statements)
    (Cond2): (Statements)
    ...
    (Cond3): BEGIN
        (Statements)
    END
ELSE: (Statements)
ENDCASE

*FOR Statement - Continues to add positive (increment) to (Value 1) until it is less than or equal to (Value 2), then executes (Statement)
- Continues to subtract (increment) from (Value 1) until it is greater than or equal to (Value 2), then executes (statement)

FOR i=(Value 1), (Value 2), (increment), DO (Statement)

FOR i=(Value 1), (Value 2), (increment), DO BEGIN
(Statement)
ENDFOR

*WHILE Statement - Executes (Condition) holds true, executes (Statement)

WHILE (Condition) DO (Statement)

WHILE (Condition) DO BEGIN
(Statement)
ENDWHILE

*REPEAT Statement - Executes (Statement) until (Condition) is true

REPEAT (Statement) UNTIL (Condition)

REPEAT BEGIN
   (Statement)
ENDREP UNTIL (Condition)

*RETURN Statement - Immediate exit from program, returning control to caller
   - If in a function, returns (Result)

RETURN, (Result)

RETURN

*GOTO Statement - Jumps to specific location (Label)

GOTO, (Label)
*SWITCH Statement - Continues going through (Expression) until it is equal to (Condition) and executes corresponding statements
- Continues going through conditions until (Cond) true (eq 1) and executes corresponding statements
- ELSE IS OPTIONAL NOT REQUIRED

SWITCH (Condition) OF

(Expression1): (Statements)
(Expression2): (Statements)
...
(ExpressionN): BEGIN

(Statements)
END
ELSE: (Statements)
ENDSWITCH

SWITCH 1 OF

(Cond1): (Statements)
(Cond2): (Statements)
...
(Cond3): BEGIN
END
ELSE: (Statements)
ENDSWITCH

*BREAK Statement - Immediate exit form FOR, WHILE, SWITCH, CASE, REPEAT

BREAK

*CONTINUE Statement - Executes next iteration of FOR, WHILE, REPEAT

CONTINUE

6.50 PRACTICE PROGRAMS

*PROGRAM #1: CREATE PROCEDURE CALLED MAXIMUM THAT WOULD PRINT SENSIBLE ANSWER WHEN FED TWO ARRAY INPUTS

- WHY DO THIS?

Due to array nature of IDL max function, might not do exactly what is desired
IDL> a=5
IDL> b=6
IDL> x=[2,3]
IDL> y=[5,6]
IDL> z=[3,2]
IDL> print,max
% PRINT: Variable is undefined: MAX.
% Execution halted at: $MAIN$
IDL> print,max(x)
   3
IDL> print,max(y)
   6
IDL> print,max(x,y)
   3
IDL> print,max(5,6)
% Attempt to store into an expression: <INT (       6)>.
% Execution halted at: $MAIN$
IDL> print,max([5,6])
   6
IDL> print,max(a,b)
   5
IDL> print, max(z, b)
    3

IDL>

-------

- Finished program should behave in the following manner

-------

IDL> maximum, x, y
      5    6
IDL> maximum, x, z
      3    3
IDL> maximum, x, 2.5
      2.5    3
IDL> maximum, a, b
      6
IDL> maximum, z, b
      6    6

-------

- HINT: Consider IDL function n_elements and > < operators
*PROGRAM #2: CREATE PROCEDURE CALLED HISTOGRAM AS ONE WOULD EXPECT

A HISTOGRAM PROGRAM TO BEHAVE

- WHY DO THIS?

Default IDL histogram function just returns an array of occurrences when fed an array

------

IDL> a=[5,3,9,4,6,4,2,1,5,8,9]
IDL> print,histogram(a)

    1  1  1  2  2  1
       0  1  2

IDL>

------

- Finished program should:

  (1) Sensibly create histogram with occurrences on y-axis and corresponding values on x-axis

  (2) Automatically create plot of sensible histogram without the need to manually evoke plot in IDL
(3) Have the optional keyword binsize and work if non-default binsize is specified

(4) Ensure that plot is centered accordingly
   - If binsize=1 and there are occurrences at 1,
     IDL should show bar from 0.5 to 1.5 and not from default 1.0 to 2.0