Introduction to Interactive Data Language (IDL)

Jules Kouatchou and Bruce Van Aartsen

Jules.Kouatchou@nasa.gov and bruce.vanaartsen@nasa.gov

Goddard Space Flight Center
ASTG - Code 606

June 2017

Training Objectives

We want to introduce:

- Basic concepts of IDL programming
- Conditional statements and loops
- Functions and procedures
- Array manipulations
- Reading and writing files
- Visualization

What is IDL?

What is IDL?

- Interactive Data Language
- Used both interactively and to create sophisticated functions, procedures, and applications
- Operators and functions work on entire arrays
- Can do rapid 2D plotting, multi-dimensional plotting, volume visualization, image display
- Immediate compilation and execution
- Contains many numerical and statistical analysis routines
- Existing FORTRAN and C routines can be dynamically linked into IDL

Launching IDL

IDL can be started in two different modes:

- IDL Development Environment (type - `idlde`).
- IDL Command Line (type `-idl`) only available on unix/linux machine.

Interactive vs Compiled Modes

Interactive mode: Commands are typed at the IDL prompt and executed when you press the enter key.

- Good for prototyping and interactive analysis
- Provides immediate feedback in numerical and visual form

Compiled mode: Programs consisting of sequences of IDL commands are created and executed.

- Two types of program units: procedures and functions
- Can be reused
- Portable

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

Other useful commands:

exit: exit IDL

Ctrl-C: stops the current program running in IDL

Basic Syntax and Variable Types

The print Command

- *print* is used to print information to the screen
- *print* is good for debugging, writing error statements, and prompting the user

```
print, [Expression1, Expression2 etc]
```

Simple Command

```
print,3*5                ; semicolon = comment, IDL skips the re
a=3*5                    ; no variable declaration needed
a = 3 * 5                ; add spaces as you like
help,a                   ; show nature and value of this variabl
help,A                   ; IDL is case-insensitive, shows variab
whatever_name_you_like$like_this_perhaps = a        ; _ an
print,whatever_name_you_like$like_this_perhaps      ; no s
spectrum_AR10910=1           ; variable names must start with a

? [search term] ; IDL help: inspect some IDL routines an
```

Integer Datatypes

IDL stores integers in

- 1 byte (The Byte Datatype): values from 0 to 255
- 2 bytes (Integers and Unsigned Integers): values from 0 to 256^2 but typically from -32768 to 32768. You generate and array using `uindgen` and `indgen`.
- 4 bytes (Long Integers and Unsigned Long Integers): values from 0 to 256^4. You generate and array using `ulindgen` and `lindgen`.
- 8 bytes (64-bit Long Integers and Unsigned 64-bit Long Integers): values from 0 to 256^8. You generate and array using `ul64indgen` and `l64indgen`.

Examples of Integers - 1

```
d=32767                ; "short" integers run from -32768 to
print,d+1              ; did you predict this value?
print,d+1.
print,2^15
print,2.^15            ; why is the integer word length not 1
? integer              ; check the other number formats
print,3276700l         ; long integer, sign+31 bits
print,3276700ul        ; unsigned long integer, 32 bits
print,3276700ull       ; unsigned long long integer, 64 bits
print,3/5
```

Examples of Integers - 2

```
 n = 10
 print, uindgen(n)
 print, lindgen(n)

a = 165
print, format='(i8)', a
print, format='(i08)', a
```

Floating Point Datatypes

There are two floating point datatypes:

- Floats (4 bytes): You generate and array using `findgen`
- Double Precision (8 bytes): You generate and array using `dindgen`.

Examples of Floating Point Numbers -1

```
print,3/5.  ; with one float makes the result a float
print,2^15.
a=[1,2,3,4,5,6] ; variables can be 1-8 dimension arrays
a=[0,a,7]        ; add values to  1D "vector"
print,a,1E6*a   ; float: 6 significant digits, < 10^38
print,a,1D6*a   ; double precision: 16 significant digits
print,a,1/a     ; divide by 0 gives error message without
print,a,1./a
print,a,a^2
print,a,alog10(10^a)   ; NaN = Not a Number
print,a,alog10(10^float(a))
```

Examples of Floating Point Numbers -2

```
n = 10
print, findgen(n)
print, dindgen(n)

a= 1.23456789
print, format='(f20.10)', a

a= 1.23456789d0
print, format='(f20.10)', a
```

Strings

- Store characters: letters, symbols, and numbers
- Enclosed by single or double quotes

String Processing

- **string**(variable,[format='(fmt)']): converts numeric variable to string following format code fmt
- **strlen**(s): length of string s
- **strmid**(s,p,n): substring of string s beginning at position p of length n
- **strpos**(s,u): position of substring u within string s
- **strtrim**(s): removes leading and trailing blanks of string s
- **strcompress**(s): 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

Examples of Strings

```
str1 = 'This is a string.'
str2 = '10.03456'
str3 = strmid(str1,3,8)
print, str3
print, strpos(str1,str3)
print, strlen(str1)

x = 10.05
print, string(x)
print, string(x,format='(I0)')
print, string(x,format='(F5.2)')

fileName = '/discover/nobackup/myfile.pro'
print, file_basename(fileName)
print, file_dirname(fileName)
```

Numeric Data Types

Types	Bit	Range	Suffix	Conversion	Create
Byte	8	0 to 255	b	byte()	bytarr(), bindgen()
Integer	16	-32,768 to 32,768		fix()	intarr(), indgen()
Unassigned Integer	16	0 to 65,535	u	unint()	uintarr(), uindgen()
Long	21	-2^{31} to $2^{31}-1$	l	long()	lonarr(), lindgen()
Unsigned Long	32	0 to $2^{32}-1$	ul	ulong()	ulonarr(), ulindgen()
64-bit Long	64	-2^{63} to $2^{63}-1$	ll	long64()	lon64arr(), l64indgen()
64-bit Unsigned Long	64	0 to $2^{64}-1$	ull	ulong64()	ulon64arr(), ul64indgen()
Float	32	-10^{38} to $10^{38}-1$		float()	fltarr(), findgen()
Double	64	-10^{308} to $10^{308}-1$	d	double()	dblarr(), dindgen()
Complex	64		$	complex(r,i)	complexarr(), cindgen()
Double Complex	128		$	dcomplex(r,i)	dcomplexarr(), dcindgen()

Nonnumeric Data Types

Data Type	Explanation
string	Character string (0-32,767 characters)
struct	Container for one or more variables
pointer	Reference to a dynamically allocated variable
objref	Reference to an object structure

Mathematical Operators

Types	Basic Operations
Integer	-, - -, +, ++, *, /,^, MOD
Floating point	-, - -, +, ++, *, /,^, MOD
String	+
Logical	eq, ne, lt, gt, le, ge and, or

Mathematical Functions

ABS		
SQRT		
EXP	ALOG	ALOG10
COS	ACOS	COSH
SIN	ASIN	SINH
TAN	ATAN	TANH
FIX	ROUND	
MAX	MIN	
MEAN	TOTAL	
FACTORIAL		
FFT		
FLOAT		
!PI		
PRIMES		
RANDOMU		

Reserved Words in IDL

eq	ne	lt	gt
le	ge	and	or
xor	not	mod	of
if	then	endif	
else	endelse	do	
for	endfor	begin	
pro	function	end	
case	endcase	common	
endrep	goto	on_ioerror	
repeat	until	while	endwhile

Conditional Statements and Loops

Syntax for Conditional Statements

```
IF expression THEN statement [ ELSE statement ]
or
IF expression THEN BEGIN
   statements
ENDIF [ ELSE BEGIN
   statements
ENDELSE ]
```

Examples of Conditional Statements

```
A = 2
B = 4

IF (A EQ 2) AND (B EQ 3) THEN BEGIN
    PRINT, 'A = ', A
    PRINT, 'B = ', B
ENDIF ELSE BEGIN
    IF A NE 2 THEN PRINT, 'A != 2' ELSE PRINT, 'B != 3'
ENDELSE
```

Syntax for FOR Loop

```
FOR variable = init, limit [, Increment] DO statement
or
FOR variable = init, limit [, Increment] DO BEGIN
    statements
    [BREAK]
    [CONTINUE]
ENDFOR
```

- Increments can be negative (but not zero)
- Be sure that index variable is a long (or long64) if max iteration exceeds 32767 (**FOR** $i = 0l, imax ...$)
- **BREAK** ends the loop
- **CONTINUE** skips to the next iteration

Examples of FOR Loop

```
FOR I = 1, 4 DO PRINT, I, I^2

f = 1
for k=1,6 do begin
     f = k * f
    print, f
endfor
```

Syntax for WHILE Loop

```
WHILE expression DO statement
or
WHILE expression DO BEGIN
   statements
ENDWHILE
```

Examples of WHILE Loop

```
i = 10
WHILE (i GT 0) DO PRINT, i--

array = [2, 3, 5, 6, 10]
i = 0 ;Initialize index
n = N_ELEMENTS(array)

; Increment i until a point larger than 5 is found
; or the end of the array is reached:

WHILE (array[i] LT 5) && (i LT n) DO i++

PRINT, 'The first element >= 5 is element ', i
```

Syntax for REPEAT Loop

```
REPEAT statement UNTIL expression
or
REPEAT BEGIN
   statements
ENDREP UNTIL expression
```

Examples of REPEAT Loop

```
A = 1
B = 10
REPEAT A = A * 2 UNTIL A GT B
;
A = 1
B = 10
REPEAT BEGIN
   A = A * 2
ENDREP UNTIL A GT B
```

Syntax for CASE and SWITCH

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

```
CASE variable OF value1: statement value2: statement val
statements
END
...
ENDCASE
```

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

Examples of CASE and SWITCH

```
x=2

CASE x OF
    1: PRINT, 'one'
    2: PRINT, 'two'
    3: PRINT, 'three'
    4: PRINT, 'four'
    ELSE: PRINT, 'Not one through four'
ENDCASE
```

Arrays

General Principles on 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 360×180 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,*]**

Examples of One Dimensional Arrays

```
a=intarr(100)                  ; define a as integer array a
a=dblarr(100)                  ; double-precision float arra
a=a+1                          ; now they are all 1.0000000
for i=0,19 do a[i]=i           ; remember that IDL starts co
a=indgen(20)                   ; same thing: a=[0,1,....,19]
print,a[0],a[19],a[10:19]
print,a[*]                     ; same as print,a and as prin
b=sqrt(a)                      ; check that b is a float arr
print,a+b
c=b                            ; define float array the same
for i=0,19 do if (b[i] gt 3) then $
     c[i] = a[i] + b[i] else c[i] = a[i]
d = a+b*(b gt 3) ; the same, processes faster, needs no
```

Examples of Two Dimensional Arrays

```
ar = [[1,2,3],[4,5,6]] ; integer [3,2] array
print,ar[0],ar[0,0]     ; mind the virtual finger
print,ar[0,*]           ; * = all values of this index
print,n_elements(ar)    ; predict all these
print,total(ar)         ; for large arrays set /double
print,shift(ar,-1)
print,transpose(ar)
print,reverse(ar)
```

Examples of Three Dimensional Arrays

```
ar=indgen(3,4,5)+1              ; let s say 3x4 px frames i
print,ar                        ; successive indices run slow
ar3=ar(*,*,2)                   ; third movie frame
print,total(ar)                 ; sum all elements
print,total(ar,1)               ; (4,5) row sums = sum over o
print,total(ar,2)               ; (3,5) column sums
print,total(ar,3)               ; (3,4) frame sums
sizear=size(ar)
print,sizear                    ; nr dims, dim1, dim2, dim3,
mean=total(ar,3)/sizear(3)      ; temporal mean of this movie
xslice=ar[*,0,*]                ; distill (x,t) timeslice at
xslice=reform(xslice)           ; reform removes degenerate d
```

Operations on Arrays - 1

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

Operations on Arrays - 2

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, kurtosiss

total() - sum of array values

median() - median array value

invert() - inverts a square ($n \times n$) array

round() - rounds elements to nearest integer

ceiling() - smallest integer ¿ each element

floor() - largest integer ¡ each element

Examples of Array Commands

```
nums = randomn(systime(1),1000)
print, mean(nums)
print, stddev(nums)
print, median(nums)
print, total(nums)
```

The WHERE Command

WHERE() returns the indices of array elements that satisfy a logical expression.
Example: **a = WHERE(x gt 0)**
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)**

Examples with WHERE()

```
nums = randomn(systime(1),1000)
a = WHERE(nums lt -1 or nums gt 1)
print, n_elements(a)
print, n_elements(a)/n_elements(nums)
print, float(n_elements(a))/n_elements(nums)
a = where(nums lt -2 or nums gt 2)
print, float(n_elements(a))/n_elements(nums)
```

Array Arithmetic

```
a = findgen(3,3)
print, a
b = 8.-findgen(3,3)
print, b
print, a*b
```

Matrix Multiplication

- $A_{(n,m)} \# B_{(m,n)} = C_{(n,n)}$
  - 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

Examples of Matrix Multiplication

```
a = [[2,3],[-0.5,4]]
a_i = invert(a)
print, a_i#a

a = findgen(2,3)
b = findgen(3,2)
print, a#b
print, a##b
```

Structure Data Type

Definition of Structures

- Structures are a special data type that allows variables of different types and sizes to be packaged into one entity.
- There are two kinds of structures:

  anonymous structure: a package of arbitrary variables

  named structure: a package of variables that conform to a template created by the user.

Structures are used when it makes sense to collect and store a group of related items (e.g., the name, identification number, and grade for each student in a class).

Anonymous Structure

Created by enclosing variable name/value pairs within curly brackets .

```
image = {name:'Test Image', valid_range: [0.0, 100.0], $
          data:dist(256)}
```

```
IDL>help, image
IDL>help, image, /structure
IDL>print, image.name
IDL>help, image.(1)
```

Nested Structure in Anonymous Structure

```
sample = {date:'19-Mar-1999', time:'20:02:05', $
            image: image}
```

```
IDL>help, sample, /structure
IDL>print, n_tags(sample)
IDL>help, sample.image, /structure
IDL>print, sample.date
IDL>print, sample.image.name
```

Arrays of Structures

Anonymous structure arrays can be created by calling the *replicate* function:

```
sequence = replicate(image, 10)
```

```
IDL> sequence[0].name = 'First Test Image'
IDL> sequence[0].data[0:100] = 1.0
; This is not allowed
IDL> a = {name:'Sean', age:32}
IDL> b = {name:'Kate', age:25}
IDL> c = [a, b]
```

Named Structures

- Conform to a template that is created by the user.
- Once a template is created, it cannot be modified for the remainder of the IDL session.
- To create a new named structure, the template name is enclosed in curly brackets, followed by variable name/value pairs.

Example of Named Structure

```
rec = {nav_record, time: 0.0, lat:0.0, lon: 0.0, $
       heading: 0.0}
```

```
IDL>help, rec
IDL>help, rec, /structure
IDL>print , rec.lat, rec.lon
IDL>print, n_tags(rec)
```

Named Structure Duplication

```
xrec = {nav_record}
xrec.time-- 12.25
yrec = rec
yrec.time = 13.50

data = replicate({nav_record}, 100)

a = {nav_record}
b - {nav_record}
c = [a, b]
```

Functions Associated with Structures

Function	Description
n_tags()	Return the number of variables (tags) within a structure
tag_name()	Return the name of each variable (tag) within a structure
create_struct()	Create a structure, or append variables to a structure

Program Structure

Principles

- IDL program files are assumed to have the extension `.pro`.
- A main program unit consists of a sequence of IDL commands that ends in an End statement. There can be only one main program unit active in IDL at any time
- When IDL searches for a user-defined procedure or function, it searches for files consisting of the name of the procedure or function, followed by the `.pro`.

Procedures

```
PRO procedure_name,arg1,arg2,...
    statements
END

procedure_name,arg1,arg2
```

Functions

```
FUNCTION fuction_name,arg1,...
   statements
   RETURN value
END

 result = function_name(arg1,...)
```

Arguments of Procedures and Functions

- **Nearly all variables are passed by reference, with a few notable exceptions.**
- We can also have **keyword** arguments:
  - May be listed in any order
  - Are always optional
  - Can be set to a single value, vector, or with a slash

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

Arguments: Reference versus Value

Passed by Reference	Passed by Value
Scalars	Constants
Arrays	Indexed subarrays
Structures	Structure elements
Undefined variables	System variables
	Expressions

Hello World Procedure

```
PRO hello_world
    PRINT, 'Hello World'
END
```

Routines for Checking Arguments

Name	Purpose
N_PARAMS()	Returns number of Parameters passed (not including keywords)
N_ELEMENTS()	Returns number of elements in a variable (zero means variable is undefined)
SIZE()	Returns size and type information about a variable
ARG_PRESENT()	Returns true if an argument was present and was passed by reference
MESSAGE	Prints a message and halts execution

Hello World Procedure with Keywords

```
PRO hello_world2, name, INCLUDE_NAME = include
   IF (KEYWORD_SET(include) && (N_ELEMENTS(name) NE 0))
            THEN BEGIN
      PRINT, 'Hello World From '+ name
   ENDIF ELSE PRINT, 'Hello World'
END
```

```
IDL>.run hello_world2
IDL> hello_world2, name, /INCLUDE_NAME

IDL>name = "Horton"
IDL>hello_world2, name, /INCLUDE_NAME
```

Is the Argument Changed?

```
PRO change_arg, arg
    arg = 3 * arg
    print, 'Value within change_arg is ', arg
END
```

```
IDL>.run change_arg
IDL> arr = [0, 1, 2, 3, 4]
IDL>change_arg, arr
IDL>print, arr

IDL>arr = [0, 1, 2, 3, 4]
IDL>change_arg, arr[0:3]
IDL>print, arr
```

Example of Procedure without using all Arguments

```
PRO plaw, X, A, F, pder
   F = A[0]*X^A[1]
   IF N_PARAMS() GE 4 then pder=[[X^A[1]] ,[A[0]*X^A[1]*
END
```

```
IDL>.run plaw
IDL> A = [2.0, 3.0]
IDL> X = [1.0, 2.0]
IDL>plaw, X, A, F
IDL> print, X
IDL>plaw, X, A, F, P
IDL>print, P
```

Example of Recursive Procedure

```
PRO factor2,X
   if x MOD 2 eq 1 then return
   x=x/2
   factor2,x
end
```

```
IDL>.run factor2
IDL> x = 22
IDL>factor, x
IDL>print, x
IIDL> x = 24
IDL>factor, x
IDL>print, x
```

Example of Function 1

```
function space_function, x
     s='
'
     return,strmid(s,0,x)
end
```

```
IDL>.run space_function
IDL> w = space_function(4)
IDL>print, w
```

Example of Function 2

```
function step_function, x, a
  n = n_elements(x)
  ; make a result array the same size as x
  result = replicate(1., n)
  ; identify by array index those
  ; elements where x is less than a
  idx = where(x LT a, count)
  ; If there is at least 1 element that is < a,
  ; set the result to 0
  if count GT 0 then result[idx] = 0
  return, result
end
```

```
IDL>.run step_function
IDL>print, step_function(1, -3.)
IDL>print, step_function([-4, -1.0, 2, 6], -3.)
```

Include Files

An include file:

- Contains a sequence of IDL statements that are inserted in a procedure or function
- May contain any statement that is legal in a procedure or function, including multiline statement blocks. statement block

To include the file *my_include_file.inc* in your function or procedure, preface it with the 'at' character (@):

```
@my_include_file.inc
```

Example of Include File

```
; Include file: fundamental_constants.inc
planck_constant = 6.6260755d-34     ; Joule second
light_speed = 2.9979246d+8          ; meters per second
boltzmann_constant = 1.380658d-23   ; Joules per Kelvin
rad_cl = 2.0d0 * planck_constant * light_speed^2
rad_c2 = (planck_constant * light_speed) / boltzmann_constant
```

```
FUNCTION PLANCK, V, T
@fundamental_constants.inc
   vs = 1.0D2 * v
   return, vs^3 * ((rad_cl * 1.0D5) / $
                   (exp(rad_c2 * (vs / t)) - 1.0D0))
END
```

Reading and Writing Text Files

General Concept

There are three steps in reading and writing files:

1. Open a file for read, write, or update.
2. Read, write, or update the content of a file
3. Close the file

Procedures for Opening Files

There are three commands for opening a file:

- **OPENR** (OPEN Read) opens an existing file for input only.
- **OPENW** (OPEN Write) opens a new file for input and output. If the file exists, it is truncated and its old contents are destroyed.
- **OPENU** (OPEN Update) opens an existing file for input and output.

Calling Sequence for Opening Files

```
OPENR, Unit, File [, Record_Length]
OPENW, Unit, File [, Record_Length]
OPENU, Unit, File [, Record_Length]
```

Unit: The unit number to be associated with the opened file.
File: A string containing the name of the file to be opened.

Closing Files

```
CLOSE[,Unit1, ..., Unitn]
```

Uniti: The IDL file units to close.

```
CLOSE, 1, 3, 7, 15
```

File Unit Numbers

- 0 : The standard input stream
- -1 : The standard output stream
- -2 : The standard error stream
- 1-99 : You can use directly, but you will have to keep track of them yourself
- 100-128 : You can use through the `Get_LUN` and `Free_LUN` procedures.

```
Get_LUN, lun
 OpenR, lun, NameOfFile
 ... do something...
Free_LUN, lun
```

Commonly Used Format Codes

Code	Output
iN.M	Integer value with up to N characters (.M is optional; however, if .M is used, any blank positions in the rightmost M characters are filled with zeroes)
fN.M	Single-precision value with up to N characters, and M digits after the decimal point
dN.M	Double-precision value with up to R characters, and M digits after the decimal point
eN.M	Floating-point value in exponential format with up to N characters, and M digits after the decimal point
aN	String with up to N characters (if N is omitted, all characters in the input string are printed)
N**x**	Skip N character positions
/	Start a new line
$	Suppress new line (on output only)
:	Terminate output if no more arguments are available

Reading Format Data

We are dealing here with a free formatted file, i.e., a file that uses either commas or whitespace (tabs and spaces) to distinguish each element in the file.

- **Read** : Reads free format input from standard input, usually the keyboard.
- **ReadF** : Reads free format input from a file.
- **ReadS** : Reads free format input from a string variable.

Interactive Reading from Standard Input

```
PRO interactive_read
    text = ''
    count = 0
    PRINT , ' Enter text (done to quit)'
    REPEAT BEGIN
        READ, text
        count++
    ENDREP UNTIL  (text EQ 'done')
    PRINT, 'Number of lines entered: ', count--
END
```

Rules for Reading Format Data

1. Input data must be separated by commas or whitespace.
2. Input is performed on scalar variables. Arrays and structures are treated as collections of scalar variables.
3. If the current input line is empty, and there are still variables left requiring input, read another line.
4. If the current input line is not empty, but there are no variables left requiring input, ignore the remainder of the line.
5. Convert data into the data type expected by the variable.
6. If reading into a string variable, all characters remaining on the current line are read into the variable.

Simple Reading Syntax

To read free format data:

```
array = IntArr(5)
Read, array
```

To read explicit format data:

```
ReadF, lun, var1, var2, Format='(3(8(F6.2,X)), 10I5)'
```

Example, Reading Salary Data

The file *employee_salary.asc* contains employee data records. Each employee has a name (String, 32 columns) and the number of years they have been employed (Integer, 3 columns) on the first line. The next two lines contain each employee's monthly salary for the last twelve months.

```
Bullwinkle                      10
1000.0    9000.97   1100.0                          2000.0
5000.0    3000.0    1000.12   3500.0    6000.0       900.0
Boris                           11
400.0     500.0    1300.10    350.0     745.0     3000.0
200.0     100.0     100.0      50.0      60.0        0.25
Natasha                         10
950.0    1050.0    1350.0     410.0     797.0      200.36
2600.0    2000.0    1500.0    2000.0    1000.0      400.0
Rocky                           11
1000.0    9000.0    1100.0       0.0       0.0     2000.37
5000.0    3000.0    1000.01   3500.0    6000.0      900.12
```

We want to write and IDL program that reads the file and prints the name of each employee together with their number of year of employment and the annual salary.

IDL Program for Reading the Salary Data

```
PRO employee_data
  OPENR, lun, 'employee_salary.asc', /Get_Lun
  ;Create variables to hold the name, number of years, a
  name = '' & years = 0 & salary = FLTARR(12)
  ;Output a heading for the summary.
  PRINT, FORMAT='("Name", 28X, "Years", 4X, "Yearly Sala
  PRINT, '========================================='

  ;Loop over each employee.
  WHILE (~ EOF(lun)) DO BEGIN
       ;Read the data on the next employee.
       READF, lun, FORMAT = '(A32,I3,2(/,6F10.2))', name
       ;Output the employee information. Use TOTAL to su
       ;salaries to get the yearly salary.
       PRINT, FORMAT='(A32,I5,5X,F10.2)', name, years, T
  ENDWHILE
  Free_Lun, lun
END
```

Reading Exercise

- The file *colDiagStationList.asc* contains a list of stations together with their latitudes, longitudes and locations
- You need to write a IDL program that:
  1. Reads the file
  2. Stores the station names in array *station_names*
  3. Stores the latitudes in array *lats*
  4. Stores the longitudes in array *lons*
  5. Stores the station locations in array *station_locations*
  6. Verifies at the end that all the arrays have the same number of elements.

Procedures for Writing Files

There are two commands in IDL for writing formatted data:

Print : Writes formatted output to the standout

PrintF : Writes formatted output to a file

Examples of Writing Data

```
data = [[1.00000, 2.00000, 3.00000], [4.00000, 5.00000,
Print, data
OPENW, lun, 'DataOutput.dat', /Get_Lun
PrintF, lun, data ; write the data into the file
Free_Lun, lun
```

To read explicit format data:

```
OPENW, lun, 'datafile.dat', /Get_Lun
PrintF, lun, var1, var2, format='(3(8(f6.2,x)),10I5))
Free_Lun, lun
```

Writing Exercise

- Read the file *colDiagStationList.asc* to create a new one (with the same information) that
  1. Contains only stations which latitudes (in absolute value) are greater than 40.0.

Obtaining Information about Files

```
file = FILEPATH ('employee_salary.asc' , $
                 subdir='File_IO')
OPENR, lun, file , /Get_Lun
info = FSTAT(lun)
help, info.size , info.read , info.name
Free_Lun, lun
```

Plotting

Syntax for Plotting

```
graphic = PLOT(Y, [Format] [, Keywords=value] $
                   [, Properties=value])
graphic = PLOT(X, Y, [Format] [, Keywords=value] $
                   [, Properties=value])
graphic = PLOT(Equation, [Format] [, Keywords=value] $
                       [, Properties=value])
```

Line Graph: Sine Plot

```
x=findgen(101)*(0.01 * 2.0 * !pi)
y=sin(x)
plot,x,y
```

Plotting Procedures

Name	Purpose
plot	Plot a line graph
oplot	Overplot a line graph on axes created by *plot*
plots	Plot a line graph in one of three coordinate systems
axis	Create a new axis
contour	Plot contours
surface	Plot a mesh surface
shade_surf	Plot a shaded surface

Overplotting

```
x=findgen(101)*(0.01 * 2.0 * !pi)
plot,x,sin(x)
oplot, x, sin(-x)
oplot, x, sin(x)*cos(x)
```

Plot Keywords

Keyword	Purpose
/polar	Create a polar plot (default: create a cartesian plot)
/ynozero	y-axis minimum is not zero when all points are positive
	(default: y-axis minimum is zero when all points are positive)
min_value	Minimum coordinate to plot (default: data minimum)
max_value	Maximum coordinate to plot (default: data maximum)
nsum	Number of values to average when plotting (default: no averaging)

Scatter Plot

```
n = 100
x=findgen(n)
y = x + (20.0 * randomu(-1L, n))
plot, x, y, psym=1
```

- The *psym* value selects one of nine predefined symbols.
- User-defined symbols may be created with the *usersym* procedure.
- A positive symbol code causes only the symbol to be plotted at each data point.
- A negative symbol code causes a symbol to be plotted at each point along with a line connecting all the points:

Symbol Codes

Code	Symbol
0	None
1	Plus
2	Asterisk
3	Dot
4	Diamond
5	Triangle
6	Square
7	Cross
8	User defined

Polar Plot

```
n = 100
r=findgen(n)*0.01
t = 4.0 * !pi * r
plot, r, t, /polar
```

Plot Related System Variables

Variable	Purpose
!p	General plot properties
!x	x-axis properties
!y	y-axis properties
!z	z-axis properties
!d	Graphics window/device properties (read-only)

```
print, !p.linestyle
!p.linestyle = 2
plot, indgen(10)
```

Specify a Plot Position

The keyword textitposition can be used to specify the location for a plot.

```
window, /free, xsize=640, ysize=512
x = findgen(200) * 0.1
plot, x,  sin(x), position=[0.10, 0.10, 0.45, 0.90]
plot, x, cos(x), position=[0.55, 0.10, 0.90, 0.90],  $
      /noerase
```

```
window, /free
x = findgen(200) * 0.1
pos = getpos(1.0)
plot, x,  sin(x), position=pos
```

Positioning Multiple Plots

```
!p.multi = [0, 2, 2, 0, 0]
x = findgen(200) * 0.1
plot, x, sin(x)
plot, x, sin(x) * x^2
plot, x, randomu(1, 200) * x, psym=1
plot, x, 4.0 * !pi * x * 0.1, /polar
```

!p.multi is:

- A *long* vector with five elements
- The second element specifies the number of plot columns
- The thitd element specifies the number of plot rows

To disable multiple plots, simply reset all elements of *!p.multi* to zero.

Plot Customization

```
x = findgen(200) * 0.1
plot, x, sin(x),                                $
    title      = 'SIN(X) vs. X',                $
    subtitle   = 'A sample IDL plot' , $
    charsize   = 1.25, $
    font       = 1,    $
    linestyle  = 3,    $
    thick      = 2.0,  $
    psym       = -1
```

Common Plot Customization Keywords

Keyword	Purpose
position	Position vector (default: automatic)
title	Title string (default: no title)
subtitle	Subtitle string (default: no subtitle)
charsize	Character size (default: 1.0)
charthick	Character thickness (default: 1)
font	Character font index (-1, 0, or 1; default:-1)
color	Color index (default: highest color table index)
linestyle	Linestyle (0-5; default: 0)
thick	Line thickness (default: 1.0)
ticklen	Tick length (default: 0.02)
psym	Symbol code (0-8, default: 0)
symsize	Symbol size (default: 1.0)
/xlog	Create logarithmic x-axis (default: create linear x-axis)
/ylog	Create logarithmic y-axis (default: create linear y-axis)
/noerase	Don't erase display (default: erase display before plotting)
/nodata	Create axes only (default: create axes and plot data)
/noclip	Don't clip data (default: clip the data to axis range)

Axis Customization Example 1

```
x = findgen(200) * 0.1
y = sin(x)
plot, x, sin(x),  xrange=[0,13.5]
plot, x, y, xrange=[0,13.5], xstyle=1
plot, x, y, xrange=[0,13.5], xstyle=1, $
               yrange=[-2.5,2.5],ystyle=1
```

Common Axis Customization Keywords

Keyword	Purpose
[*xyz*]range	Axis range (default: automatic)
[*xyz*]title	Title string (default: no title)
[*xyz*]charsize	Character size (default: 1.0)
[*xyz*]style	Axis style (0-31, default 0)
[*xyz*]thick	Axis and tick mark thickness (default 1.0)
[*xyz*]ticklen	Tick length (default: 0.02)
[*xyz*]margin	Margin at axis edges (default x: [10, 3],y: [4, 2])
[*xyz*]minor	Number of minor tick intervals (default: automatic)
[*xyz*]gridstyle	Grid style (0-5, default 0)
[*xyz*]tickformat	Tick label format code (default: automatic)
[*xyz*]ticks	Number of major tick intervals (default: automatic)
[*xyz*]tickv	Array of tick values (default: automatic)
[*xyz*]tickname	Array of tick labels (default: automatic)
[*xyz*]tick_get	Return an array of tick values

Axis Customization Example 2

```
t = findgen(11)      ; time
a = 9.8              ; acceleration due to gravity
v = a * t            ; velocity
x = 0.5 * a * t^2    ; distance

plot, t, x, /nodata, ystyle = 4, $
      xmargin = [10, 10] , xtitle = 'Time (sec) '
axis, yaxis = 0, yrange = [0, 100], /save, $
      ytitle = 'Velocity (meters/sec, solid line)'
oplot, t, v, linestyle = 0
axis, yaxis = 1, yrange = [0, 500], /save, $
      ytitle = 'Distance (meters, dashed line)'
oplot, t, x, linestyle = 2
```

Logarithmic Axes

```
x = findgen(200) * 0.1 + 1.0

plot, x, x^3, /ylog
```

Titles and Symbols

```
x = findgen(100) * 0.1 - 5.0
y = 1.0 - exp(-(x^2))
title  = '!3CO!D2!N Spectral Absorption Feature!X'
xtitle = '!3Wavenumber (cm!U-1!N)!X'
ytitle = '!3Transmittance!X'
plot, x + 805.0, y, title = title, xtitle = xtitle, $
      ytitle=ytitle
```

Common Font-Formatting Commands

Command	Meaning
!3	Select Simplex Roman font (default)
!4	Select Simplex Greek font
!6	Select Complex Roman font
!X	Revert to entry font
!C	Begin new line
!D	Select subscript level and character size
!U	Select superscript level and character size
!N	Select normal level and character size

Error Plot

```
n = 10
x = findgen(n)
y = randomu(-1L, n) + 10
plot, x, y, yrange=[9.5, 11.5 ]
err = 0.1
err_plot, x, y - err , y+err
```

Bar Plot

```
sites = [20, 55, 102, 235, 350]
years = ['1995' , '1996' , '1997' , '1998' , '1999']
xtitle = 'Year'
ytitle = 'Number of Sites'
title = 'IDL Web Sites Worldwide'
loadcolors
bar_plot, sites, barnames=years, $
    colors=[1, 2, 3, 4, 5], $
    title=title, xtitle=xtitle, ytitle=ytitle, /outline
```

Contour Plot

```
n = 50
z = randomu(-100L, n, n)
for i = 0, 4 do z = smooth(z, 15, /edge)
z = (z - min(z)) * 15000.0 + 100.00  ; total ozone
x = findgen(n) - 100.0               ; longitude
y = findgen(n) + 10.0                ; latitude
levels = [150, 200, 250, 300, 350, 400, 450, 500]
c_labels = [0, 1, 0, 1, 0, 1, 0, 1]
contour, z, x, y, levels = levels , c_labels=c_labels
```

Filled Contour Plot

```
levels = [150, 200, 250, 300, 350, 400, 450, 500]
nlevels = n_elements(levels)
ncolors = nlevels + 1
bottom = 1
c_levels = [min(z), levels, max(z)]
c_labels = [0, replicate(1, nlevels), 0]
c_colors = indgen(ncolors) + bottom
loadct, 33, ncolors=ncolors, bottom=bottom
contour, z, x, y, $
  levels=c_levels, c_colors=c_colors, /fill, $
  xstyle=1, ystyle=1, title='Simulated Total Ozone', $
  xtitle='longitude', ytitle='Latitude'
contour, z, x, y, $
    evels = c_levels, c_labels=c_labels, /overplot
```

Surface Plot

```
v = findgen(41) * 0.5 - 10.0
x = rebin(v, 41, 41, /sample)
y = rebin(reform(v, 1, 41), 41, 41, /sample)
r = sqrt(x^2 + y^2) + 1.0e-6
z = sin(r) / r
surface, z, x, y, charsize=1.5
```

Shaded Surface Plot

```
v = findgen(41) * 0.5 - 10.0
x = rebin(v, 41, 41, /sample)
y = rebin(reform(v, 1, 41), 41, 41, /sample)
r = sqrt(x^2 + y^2) + 1.0e-6
z = sin(r) / r
shade_surf, z, x, y
```

Saving a Plot in a png File

```
SET_PLOT, 'png'
DEVICE, /DECOMPOSED, /COLOR

v = findgen(41) * 0.5 - 10.0
x = rebin(v, 41, 41, /sample)
y = rebin(reform(v, 1, 41), 41, 41, /sample)
r = sqrt(x^2 + y^2) + 1.0e-6
z = sin(r) / r
shade_surf, z, x, y

WRITE_PNG, 'output.png', TVRD(TRUE = 1)
DEVICE, /CLOSE
```

Saving a Plot in a ps File

```
SET_PLOT, 'ps'
DEVICE, filename='output.ps'

v = findgen(41) * 0.5 - 10.0
x = rebin(v, 41, 41, /sample)
y = rebin(reform(v, 1, 41), 41, 41, /sample)
r = sqrt(x^2 + y^2) + 1.0e-6
z = sin(r) / r
shade_surf, z, x, y

; Close the postscript
DEVICE, /CLOSE

; Set idl back to one plot per page
!p.multi=0

; Set idl to draw to the screen
set_plot,'x'
```

References I

Kenneth P. Bowman.
An Introduction to Programming with IDL: Interactive Data Language.
Elsevier Inc, 2005.

David W. Fanning.
Traditional IDL Graphics.
Coyote Book Publishing, 2011.

Lilian Gumley.
Practical IDL Programming.
Morgan Kaufmann, 2001.