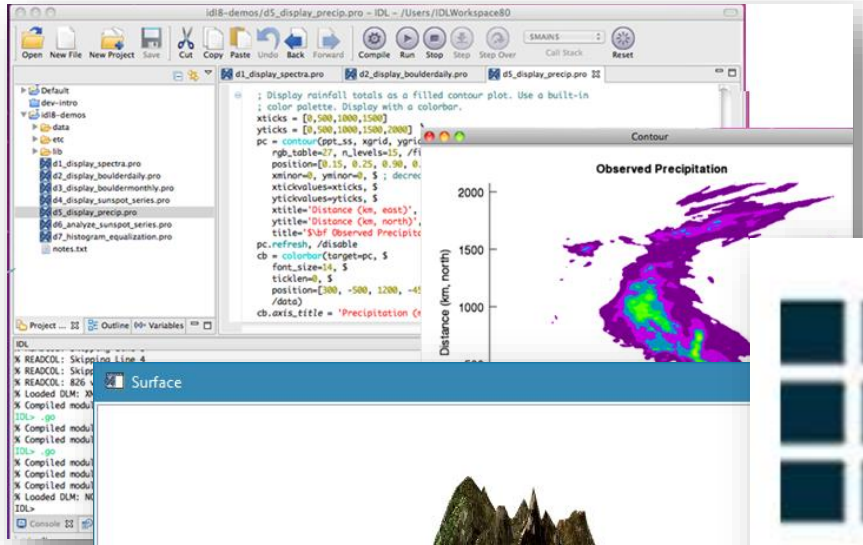
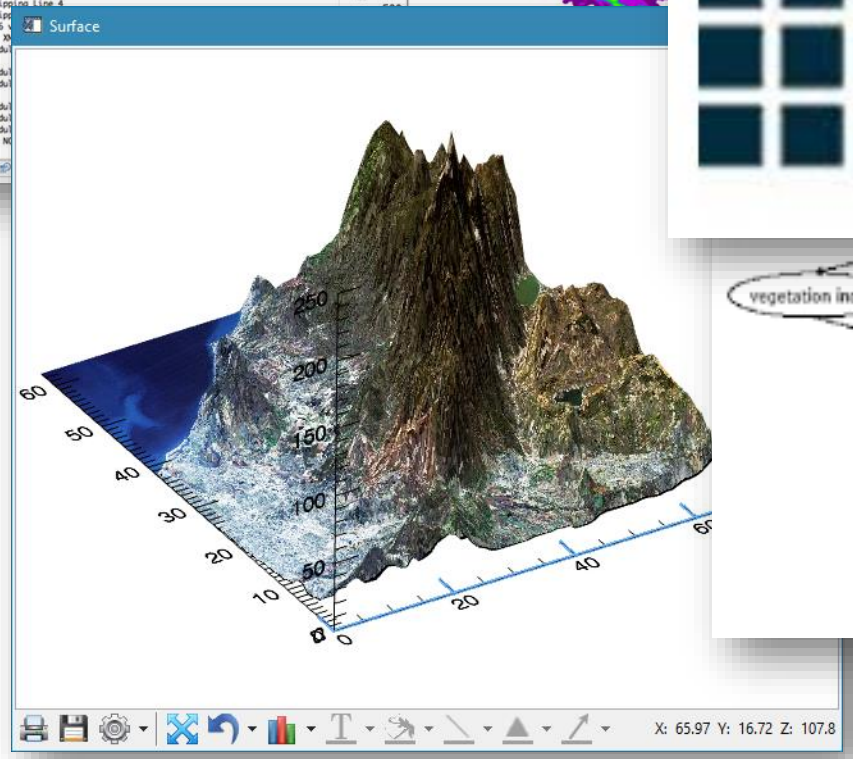
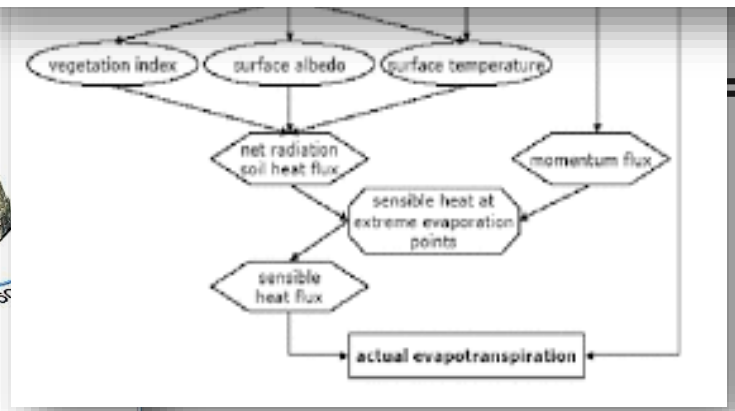
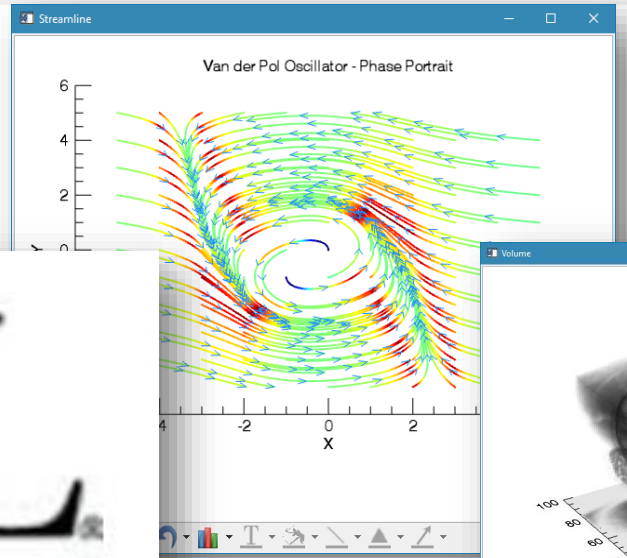
IDL Programming in ENVI

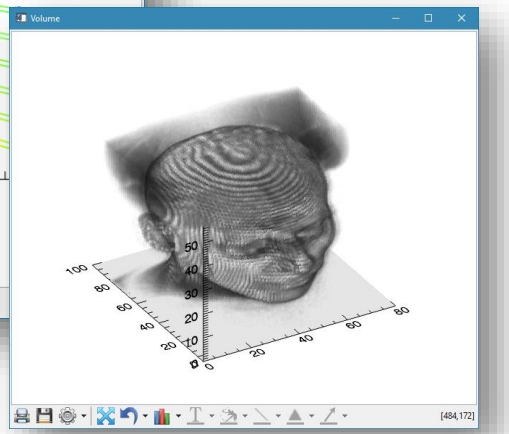




IDL





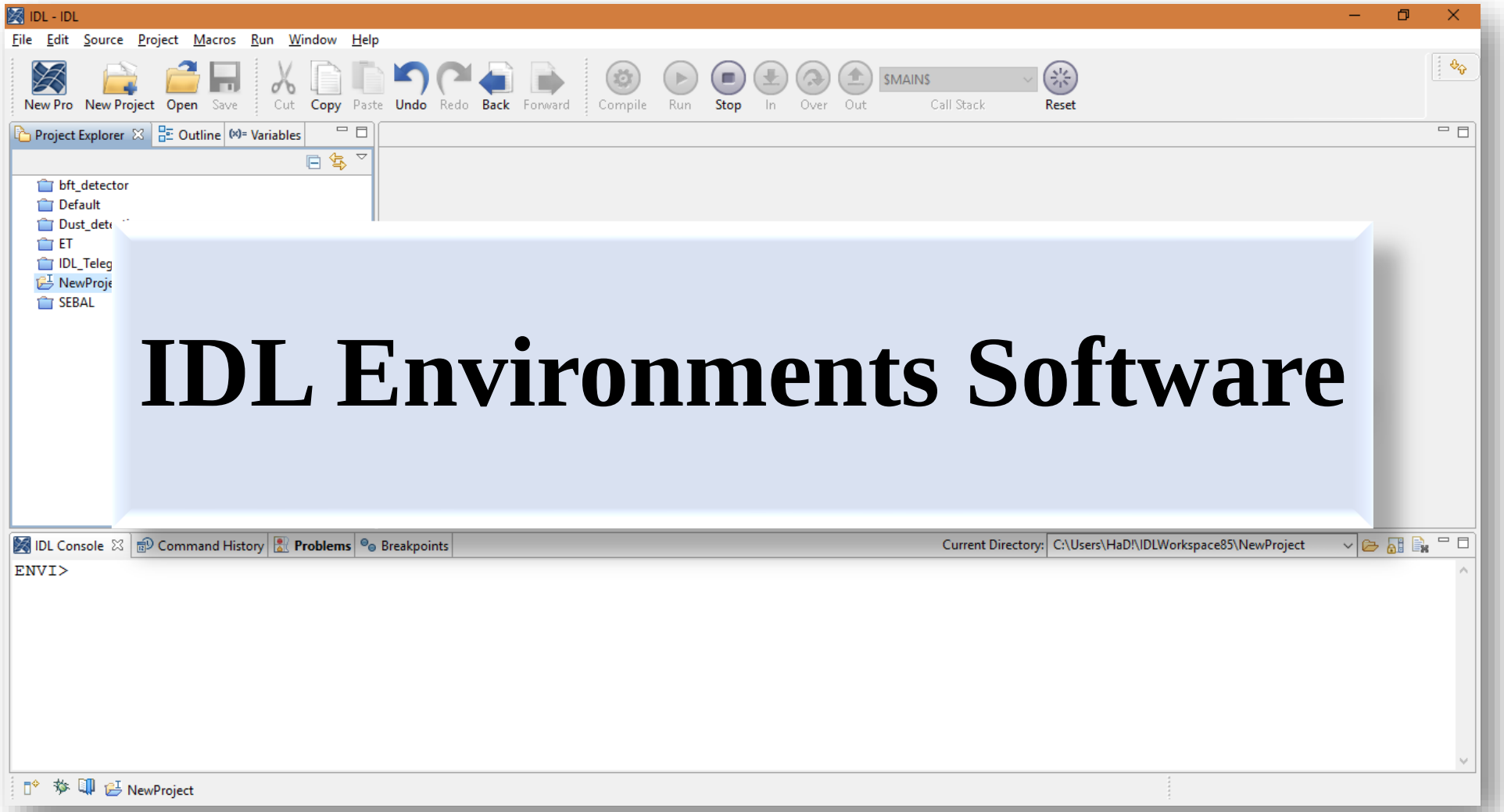


About Us

Instructor: HaDi Taji

MSc. of Water Resource Engineering from TMU

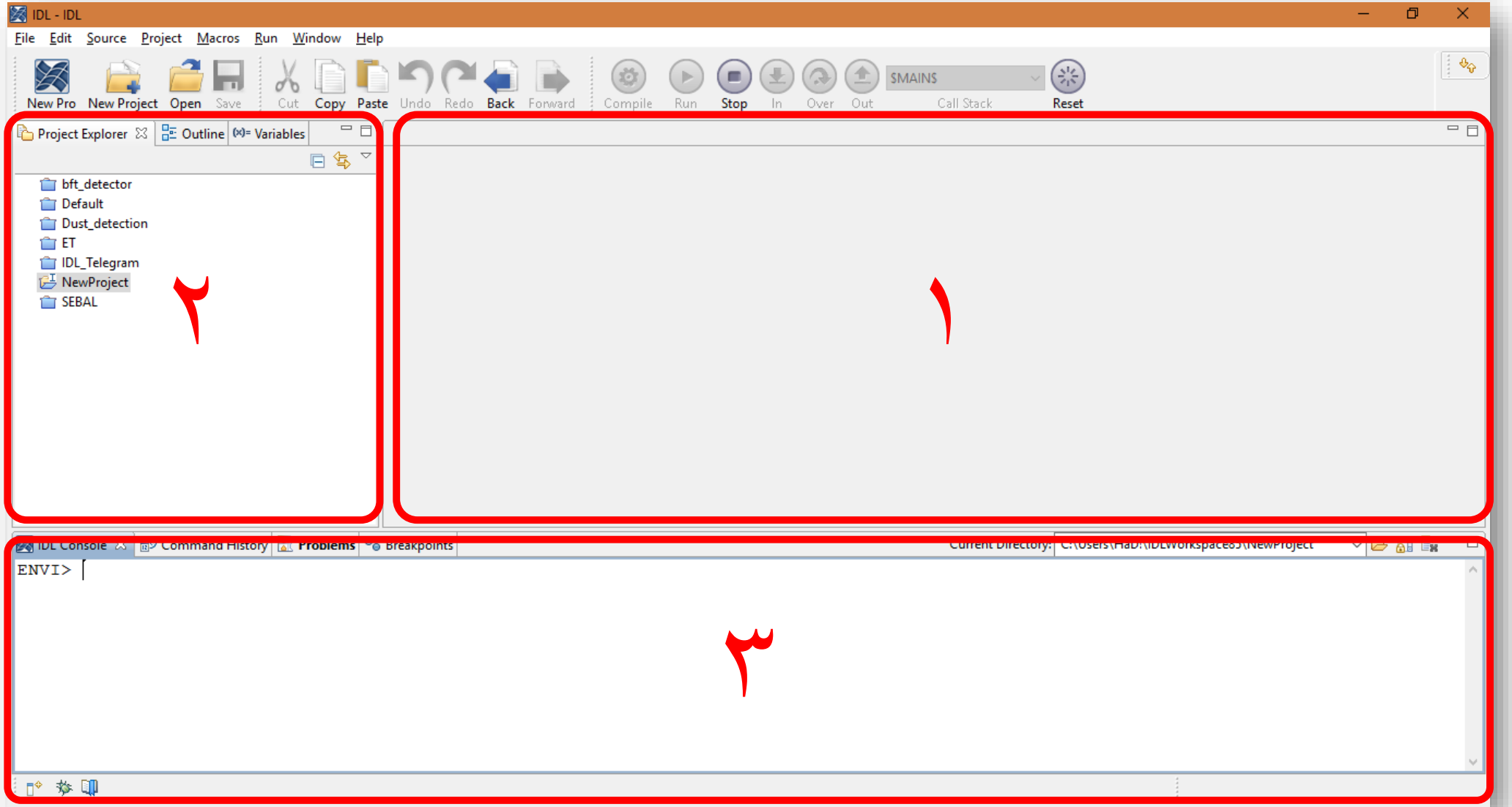
Chapter One


IDL - IDL
File Edit Source Project Macros Run Window Help
New Pro New Project Open Save Cut Copy Paste Undo Redo Back Forward Compile Run Stop In Over Out $MAINS Call Stack Reset
Project Explorer Outline Variables
bft_detector
Default
Dust_dete
ET
IDL_Teleg
NewProje
SEBAL
IDL Environments Software
IDL Console Command History Problems Breakpoints Current Directory: C:\Users\HaD!\IDLWorkspace85\NewProject
ENVI>
NewProject

Introducing the Windows

معرفی پنجره ها و منوها

محیط IDL بطور کلی شامل ۸ پنجره اصلی است که بصورت پیش فرض در سه حالت بشکل زیر نمایش داده می شود.



معرفی پنجره ها و منوها

محیط IDL بطور کلی شامل ۸ پنجره اصلی است که بصورت پیش فرض در سه حالت بشکل زیر نمایش داده می شود.

۱. **پنجره Pro_File :** که محل نمایش کدهای نوشته شده می باشد.

۲. **سه پنجره ترکیبی:**

۲-۱. پنجره Project Explorer : محل نمایش کدهای نوشته شده می باشد.

۲-۲. پنجره Outline : محل نمایش برنامه های دردسترس و نوشته شده در یک ماژول است.

۲-۳. پنجره Variables : محل نمایش متغیرهای ذخیره شده در حین اجرای برنامه است.

۳. **چهار پنجره ترکیبی:**

۳-۱. پنجره IDL Console : جزو مهمترین پنجره ها و محل تایپ و ارائه همزمان و بلافاصله نتایج اجرای کد می باشد.

۳-۲. پنجره Command History : محل ذخیره سازی و نمایش لیستی از دستورات از قبل اجرا شده در IDL می باشد.

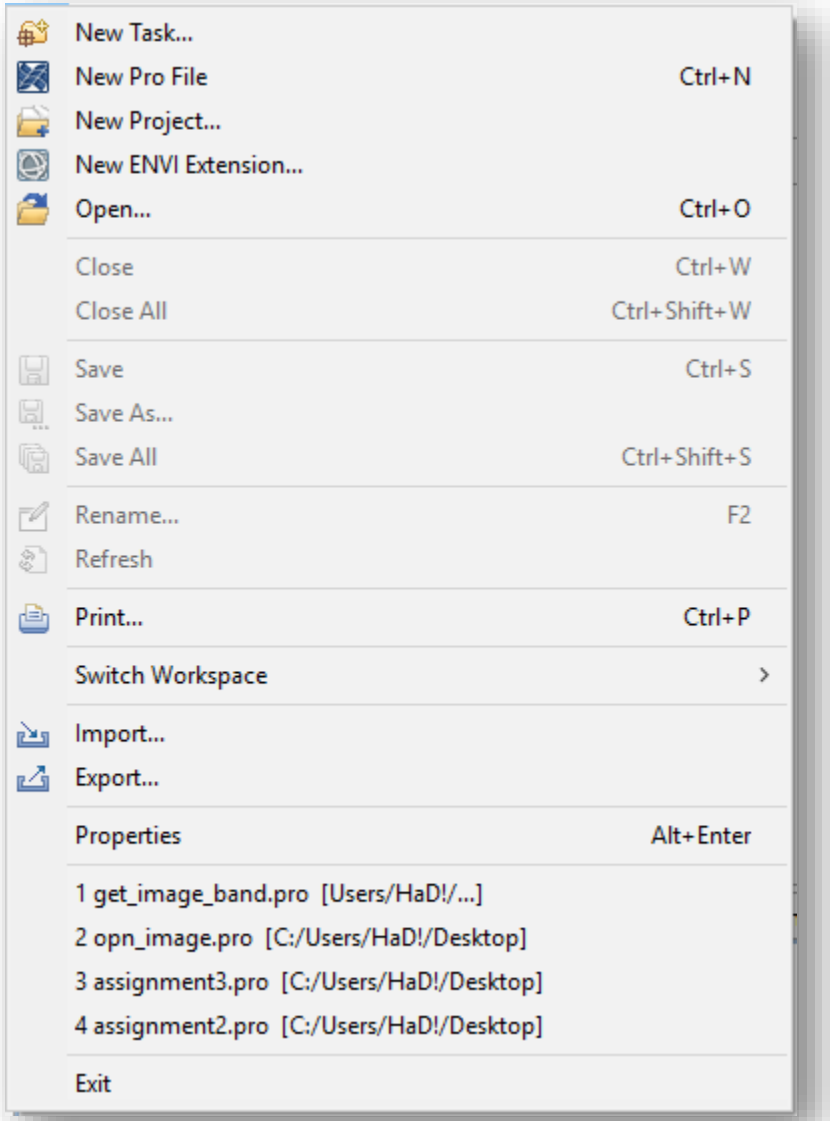
۳-۳. پنجره Problems : محل نمایش خطاها و هشدارهای حاصل از اجرای یک برنامه با قابلیت مکانیابی می باشد.

۳-۴. پنجره Breakpoints : محل نمایش و مکان یابی نقاطی شکست نقطه ای روند اجرای برنامه می باشد که در خطایابی ها و اجرای گام بگام یک برنامه بسیار کارآمد است..

معرفی پنجره ها و منوها

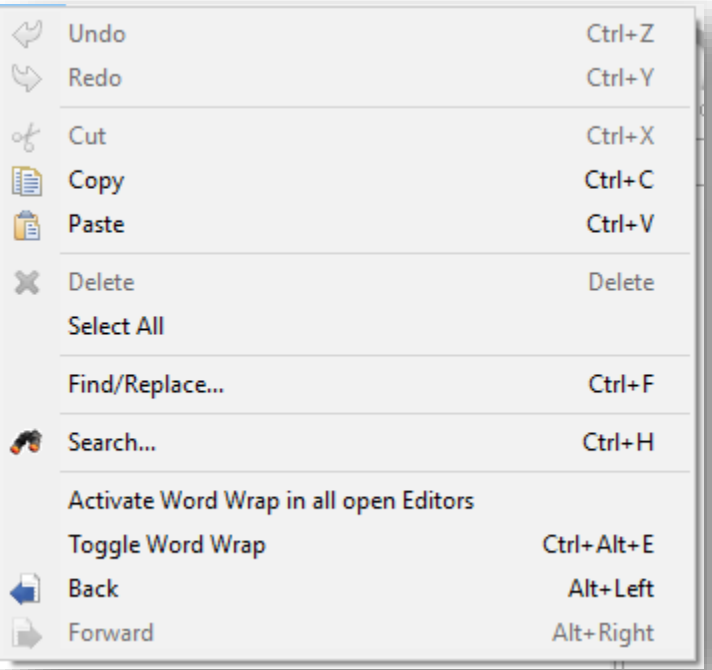
این محیط بطور کلی شامل ۸ نوار منو به شرح مختصر زیر می باشد:

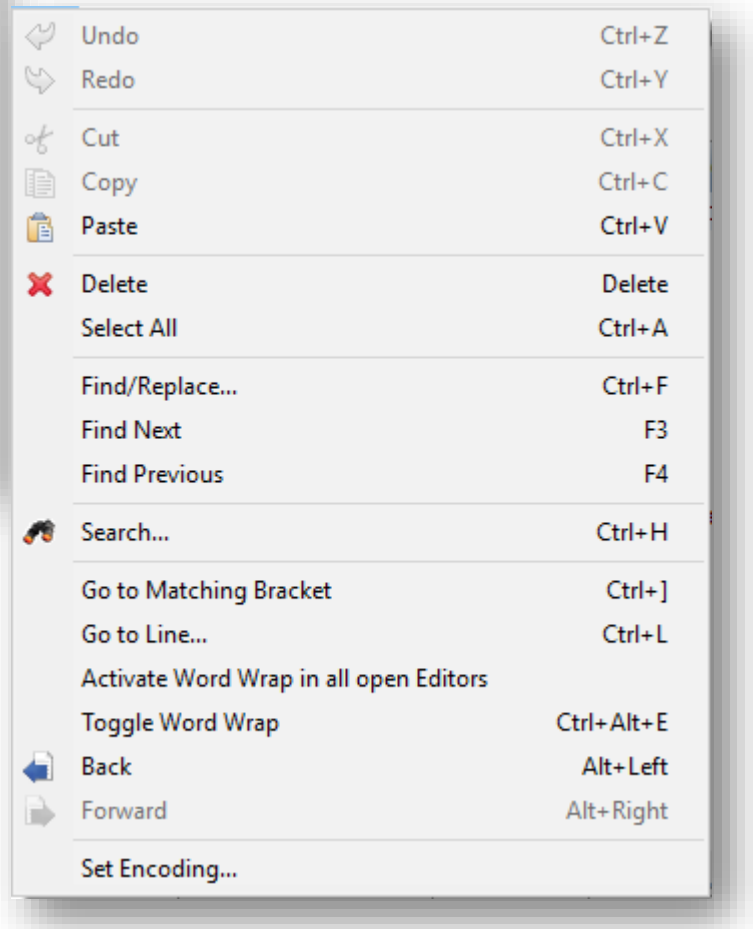
۱- **File :** شامل گزینه هایی برای ساختن، باز کردن و یا بستن تعدادی برنامه ها و یا پروژه از قبل نوشته شده و همچنین تنظیمات کلی پروژه منتخب می باشد.



معرفی پنجره ها و منوها

این محیط بطور کلی شامل ۸ نوار منو به شرح مختصر زیر می باشد:

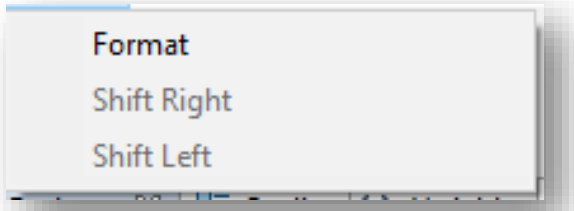


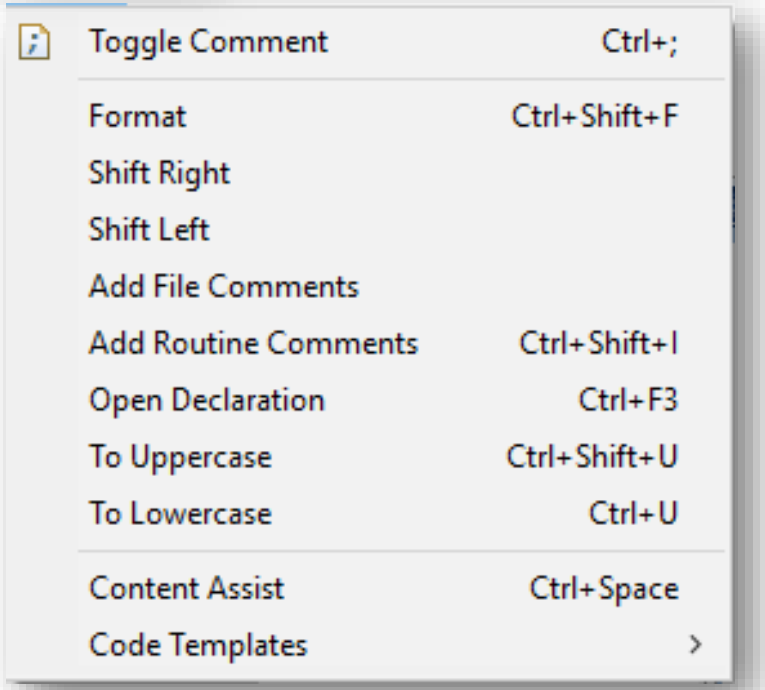


۲- **Edit :** شامل ابزاری برای جستجو و جایگزینی و انجام عملیات برش و یا کپی و چسباندن متن کدها و برنامه های نوشته شده است. این گزینه در در حالت باز بودن یک برنامه دارای گزینه هایی بیشتری خواهد بود.

معرفی پنجره ها و منوها

این محیط بطور کلی شامل ۸ نوار منو به شرح مختصر زیر می باشد:

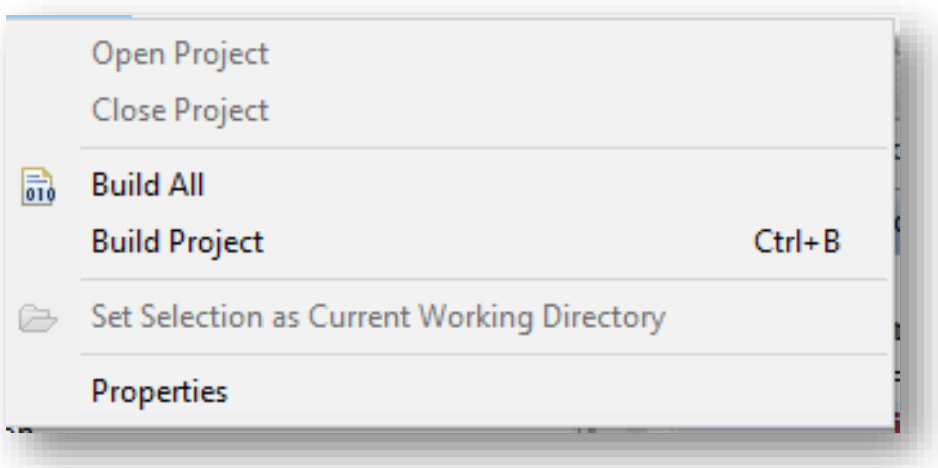




۳- **Source :** شامل گزینه هایی برای ویرایش متن کدها و برنامه های نوشته شده است. این گزینه در در حالت باز بودن یک برنامه دارای گزینه هایی بیشتری همچون غیر اجرایی نمودن خطوطی منتخب از کد، تغییر محل تعدادی از خطوط انتخاب شده از برنامه (TAB Key) و مرتب سازی کتابخانه ای آنها برای درک بهتر خواننده، اضافه نمودن توضیحاتی برای مشخص سازی نام نوسینده کد، تبدیل متن منتخب به حروف بزرگ و یا کوچک و ارائه شمایلی از ساختار برخی کدها مثل دستورات شرطی و حلقه های تکرار خواهد بود.

معرفی پنجره ها و منوها

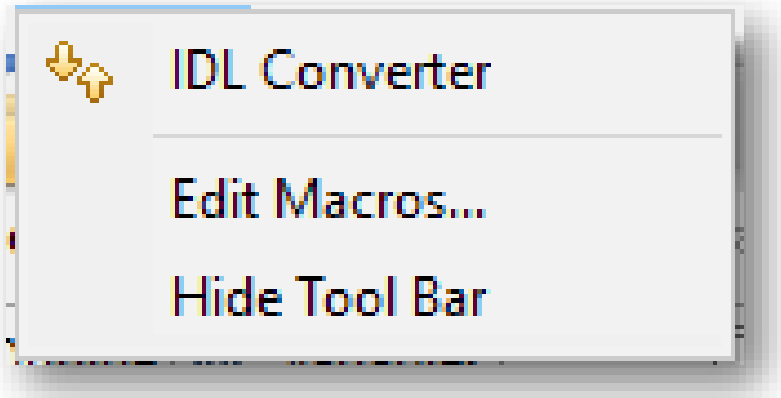
این محیط بطور کلی شامل ۸ نوار منو به شرح مختصر زیر می باشد:



۴- **Project :** شامل گزینه هایی برای باز ویا بسته نمودن یک پروژه، ساخت یک پروژه منبع بسته و غیرقابل ویرایش بشکل sav. از یک پروژه، تعیین محل پروژه بعنوان پوشه جاری IDL و نهایتا انجام تنظیمات کلی پروژه می باشد.

معرفی پنجره ها و منوها

این محیط بطور کلی شامل ۸ نوار منو به شرح مختصر زیر می باشد:

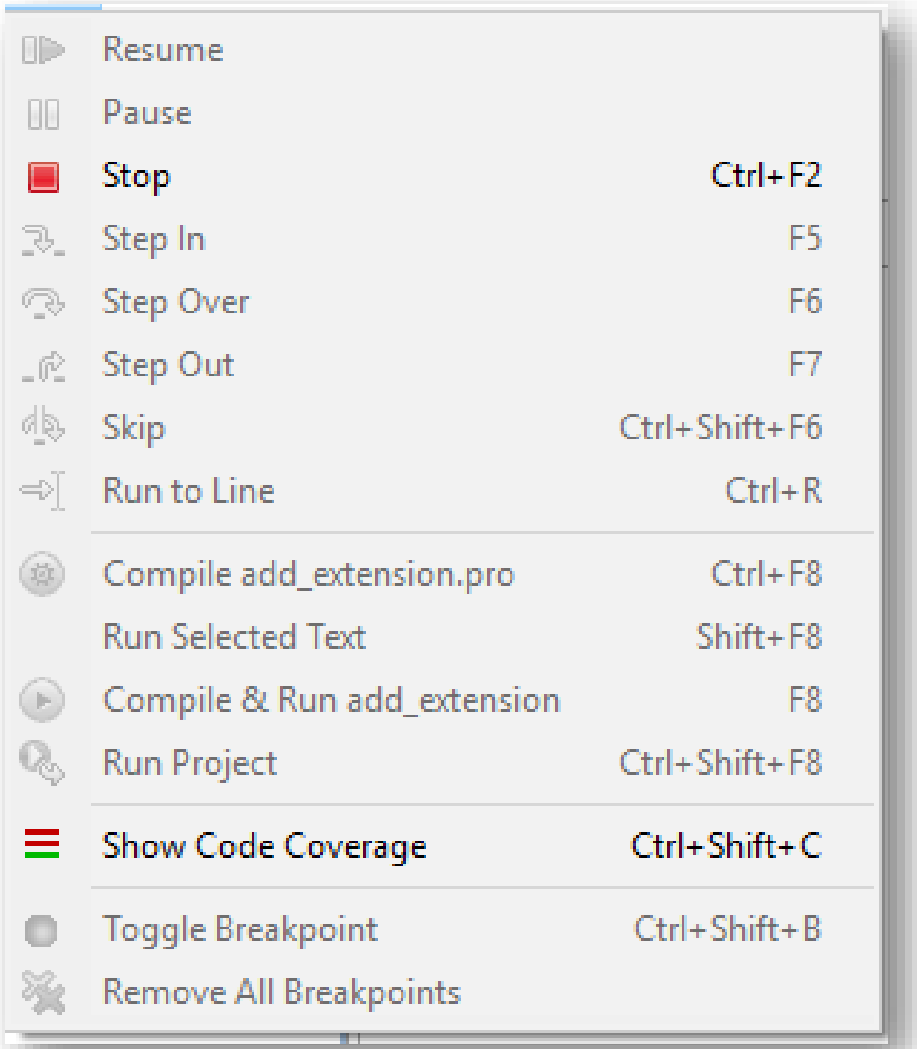


۵- **Macros :** شامل گزینه هایی همچون ماشین حساب IDL ای برای تبدیل واحد کمیت های مختلف، ایجاد و اختصاص یک دکمه به برنامه در قسمتی از نوار ابزار محیط IDL برای کاربری راحت تر کاربر در استفاده و اجرای کد نوشته شده خود و نمایش و پنهان سازی دکمه های اضافه شده می باشد.

بنظر می رسد این نوار در آینده می تواند جای پیشرفت بیشتری داشته باشد.

معرفی پنجره ها و منوها

این محیط بطور کلی شامل ۸ نوار منو به شرح مختصر زیر می باشد:

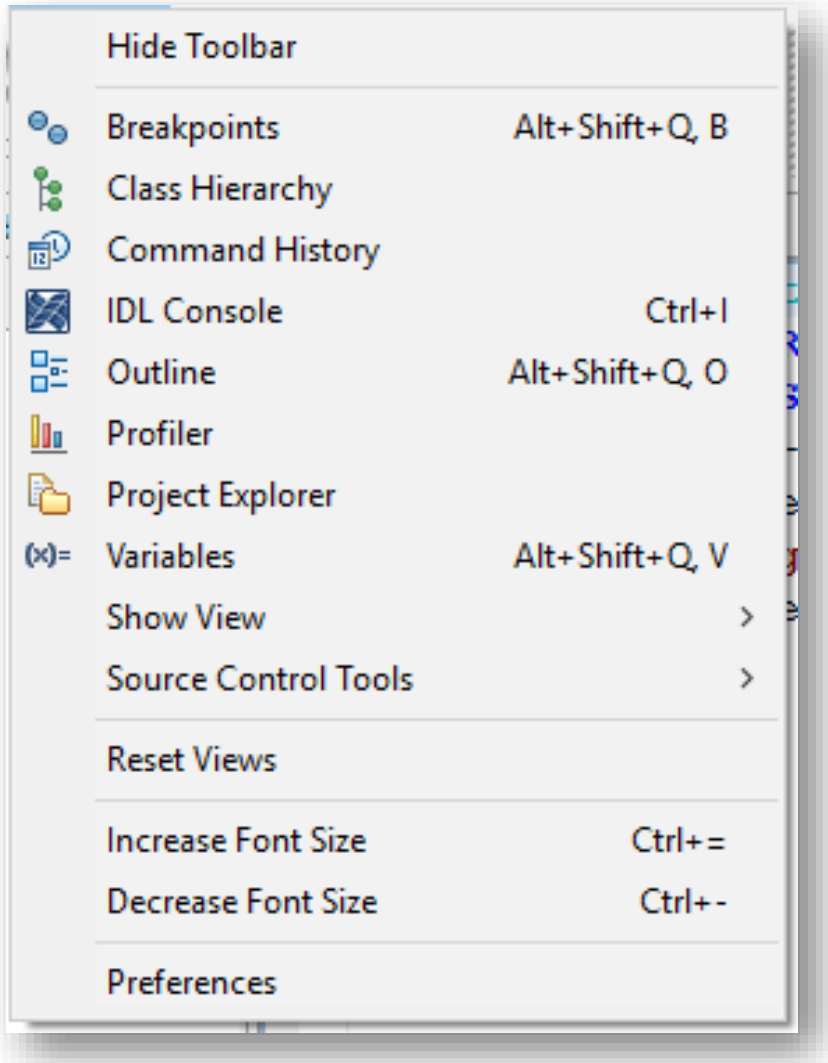


۶- **Run :** شامل گزینه هایی برای اجرا، مکث و ادامه و یا توقف یک برنامه، اجرای گام بگام برنامه، اجرای قسمتی منتخب از یک برنامه، فعال سازی و غیرفعال سازی حالت گرافیکی پوشش اجرای کد (با اختصاص رنگ سبز بمنظور اجرای خطوط خوانده شده و رنگ قرمز بمنظور خطوط خوانده و اجرا نشده از کد) و همچنین ایجاد و حذف نقاطی شکست در روند اجرای برنامه است.

لازم به ذکر است که گزینه Show Code Coverage در مراحل ابتدایی کدنویسی و یادگیری این زبان و همچنین خطایابی می تواند بسیار مفید باشد.

معرفی پنجره ها و منوها

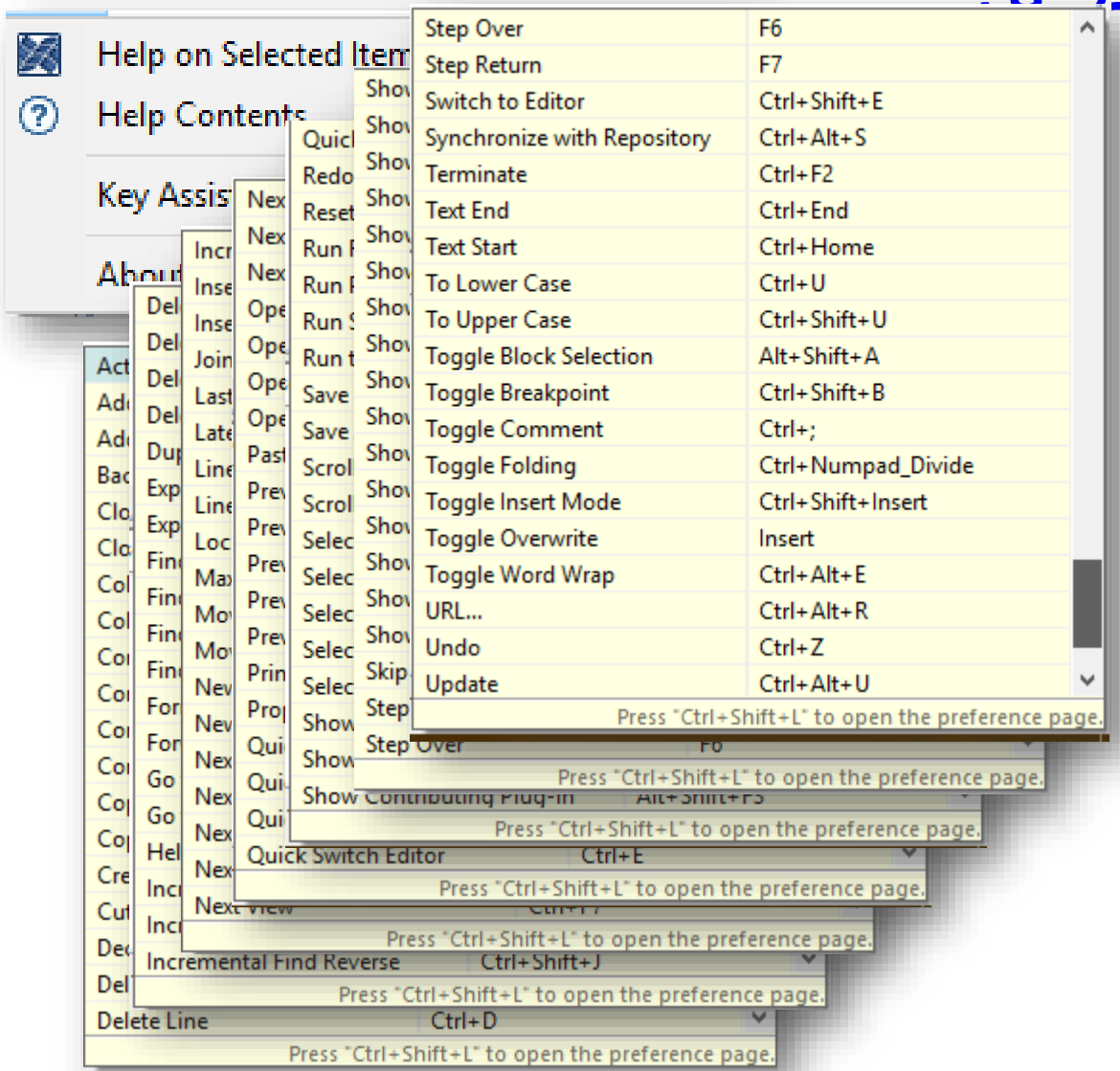
این محیط بطور کلی شامل ۸ نوار منو به شرح مختصر زیر می باشد:



۷- **Window :** شامل گزینه هایی همچون نمایش و عدم نمایش نوار ابزار، پنجره های محیط IDL، برگداندن به حالت پیش فرض مرتب سازی پنجره ها، افزایش و یا کاهش فونت نمایش خطوط متن از کدهای نوشته شده و همچنین تنظیمات کلی محیط IDL می باشد.

معرفی پنجره ها و منوها

این محیط بطور کلی شامل ۸ نوار منو به شرح مختصر زیر می باشد:



۸- Help : شامل گزینه هایی همچون جستجوی راهنمای برخی از دستورات منتخب در متن کد نوشته شده، باز کردن صفحه راهنمای کلی IDL و نمایش کلیدهای ترکیبی متفاوت و کارآمد برای دسترسی سریعتر کاربر به برخی ابزار و فعالیت ها می باشد.

گزینه Key Assistants دارای تعداد زیادی کلید ترکیبی قابل نمایش است:

مهمترین کلیدهای ترکیبی در IDL

از جمله مهترین کلیدهایی که لازم است کاربر در ابتدای کدنویسی بداند می توان به موارد زیر اشاره داشت:

۱. کلید ترکیبی Ctrl + Space (کلید یاری رسان دستورات): به شما در هر قسمت از کدنویسی، راهنما و لیستی از گزینه های انتخابی و اجباری را نشان می دهد و شما را در استفاده از یک دستور بدون نقص و ایراد یاری می رساند.

۲. کلید ترکیبی Ctrl + D (کلید پاک کننده خطوط برنامه): به شما اجازه می دهد در هر خطی از برنامه که نشانگر موس در آن فعال باشد، آن خط را بطور کامل پاک نمایید.

۳. کلید ترکیبی Ctrl + Shift + W (کلید بستن تمامی پنجره های برنامه): در هنگام بستن تعداد زیادی از پنجره های برنامه های باز شده بصورت یکباره کمک می کند.

۴. کلید ترکیبی ↓ or ↑ + Alt (کلید جابجاکننده خطوط برنامه): به شما کمک میکند هر خطی از برنامه که نانشگر موس در آن فعال باشد را بدون برش بین خطوط دیگر جابجا کنید.

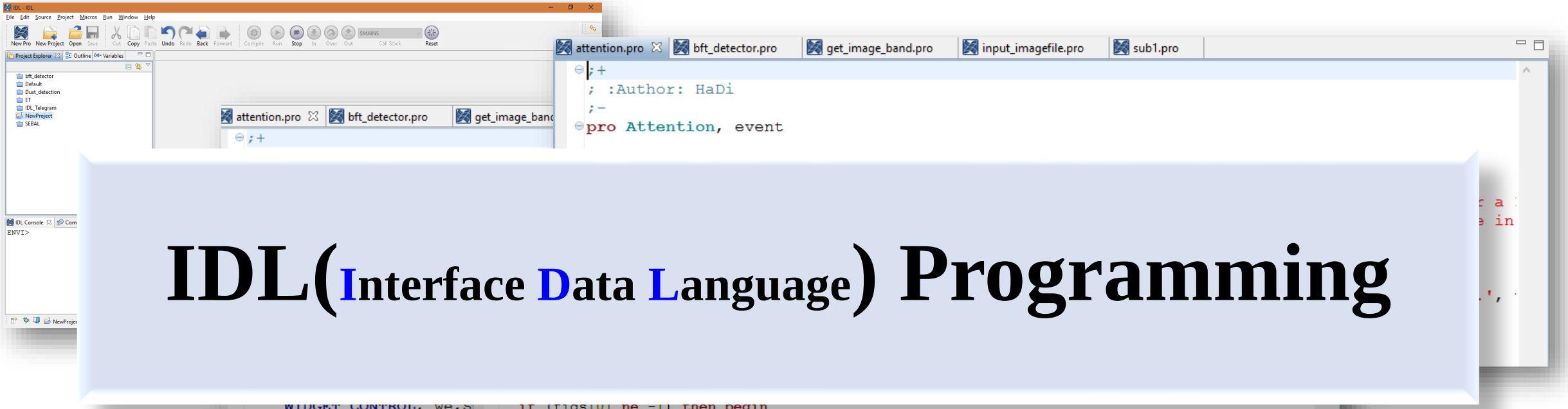
۵. کلید ترکیبی F8 (کلید ران برنامه): برای اجرای یک برنامه باز شده بکار می رود.

۶. کلید ترکیبی - or = + Ctrl (کلید تغییر اندازه فونت نوشتاری): برای تغییر اندازه نوشتاری کدها بکار می رود.

۷. کلید های ترکیبی ویرایش متن Ctrl + C or X or V (کلیدهای کپی، برش و جایگزینی): برای ویرایش قسمتی منتخب از متن برنامه است.

۸. کلید های Up or Down (دستورات قبلی): در محیط Console IDL قادر هستید تا دستورات قدیمی تر را ظاهر و از آنها استفاده نمایید.

Chapter Two



IDL(Interface Data Language) Programming

Beginning Step by Step


IDL
Variables


IDL
Scalers


IDL
Vectors


IDL
Array or Matrices

طبقه بندی متغیرها در IDL

توجه حداکثر تعداد کاراکتر برای اختصاص به نام متغیرها در IDL دارای محدودیت است.

۱- متغیرها از منظر محتوا :

۱-۱- متغیرهای عددی: عدد صحیح، اعشاری، مختلط

۱-۲- باینری (Boolean): تنها دارای مقدار صفر و یا یک

۱-۳- بایتی (Byte): تنها دارای مقدار بین ۰ تا ۲۵۵ و یا تناوبی از آن

۱-۴- رشته ای (string): برای نوشتن و استفاده از کارکترهای نوشتاری و حروفی

۲- متغیرها از منظر گنجایش:

۲-۱- اسکالر (صفر بعدی و تنها دارای یک جایگاه)

۲-۲- وکتوری (سطری/ ستونی) و دارای جایگاهی خطی شکل و یک بعدی

۲-۳- آرایه ای / ماتریسی و دارای جایگاهی دو بعدی یا بیشتر

۲-۴- ساختاری که شکلی از آرایه است با این تفاوت که دارای یک عنوان یا title است.

Variables

- variable is used in IDL to symbolically represent a value. The value of a variable is subject to change (as its name suggests). Variables come in different types. They must be declared as a certain type, and then assigned a value consistent with the type declared.

- The types of variables that IDL supports are: Boolean, Integer, Long Integer, Floating Point Decimal, Double Precision Floating Point Decimal, Complex, Characters, and Strings.

Variables

- An **Integer** variables can be any whole number. Ranging from -32,768 to +32,767. An Integer variable is 16 bits long.
- A **Long Integer** variable is the same as an Integer variable, except that it takes up 32 bits of memory and therefore spans the range of approximately -2, 000,000,000 to +2, 000,000,000.
- A **Floating Point** variable is a 32-bit, single-precision, floating-point number in the range of ±10^38, with approximately six or seven decimal places of significance.
- A **Double Precision Floating Point** variable is a 64-bit, double-precision, floating-point number in the range of ±10^38 on VMS systems, and ±10^308 on machines supporting the IEEE standard, with approximately 14 decimal places of significance.
- A **Complex** variable is a real-imaginary pair of single-precision, floating-point numbers. Complex numbers are useful for signal processing and frequency domain filtering. It also come in a double precision variety.
- **Boolean** variables can only be 0 or 1. They are used in binary logic.
- A **Byte** variables can normally only be between 0 to 255. They are used in ASCII codes.
- A **String** is a sequence of characters, from 0 to 32,767 characters in length, which is interpreted as text. A character is 8 bits long. It is therefore a number between 0 and 255 which is interpreted as an alphanumeric symbol.

Variables

Variables Type will be shown with a long scalers code leads to a certain type in IDL like bellow.

Attend that this table can be used in result of Size() function.

TYPE

The type code to set the type of the result.

Type Code	Type Name	Data Type
0	UNDEFINED	Undefined
1	BYTE	Byte
2	INT	Integer
3	LONG	Longword integer
4	FLOAT	Floating point
5	DOUBLE	Double-precision floating
6	COMPLEX	Complex floating
7	STRING	String
8	STRUCT	Structure
9	DCOMPLEX	Double-precision complex
10	POINTER	Pointer
11	OBJREF	Object reference
12	UINT	Unsigned Integer
13	ULONG	Unsigned Longword Integer
14	LONG64	64-bit Integer
15	ULONG64	Unsigned 64-bit Integer

Variables

Lets assume that we have some variable named ‘a’. We wish to declare it with a value of 5. Here are some various ways to do so.

- Integer: a = 5
- Long Integer: a = 5L (lower case l is acceptable but looks like a 1)
- Floating Point: a = 5.0
- Double Precision: a = 5D
- Byte: a = 5B
- Complex: a = complex(5,0)
- String a = ’5’

Variables

- Names:
  - Can be any string of upper and lower case letters along with numbers and underscores but it must begin with a letter.
  - Reserved names are IF, WHILE, ELSE, END, SUM, etc.
  - Names are Non_case sensitive In point to the letters of a variable names.
- Value: This is the data is associated to the variable
- Re-assignment: There are no warnings if you overwrite a variable with something of a different type.

توجه : برای پاک کردن یک یا چند متغیر می بایست از دستور **Delvar** استفاده نمایید. اگر چنانچه قصد پاک کردن تمامی متغیرهای موجود در پنجره Variables را دارید، باید در نوار ابزار محیط IDL دکمه Reset را فشار دهید و یا از دستور ***.RESET_SESSION*** استفاده نمایید. نکته حائز اهمیت در این خصوص این است که استفاده از این دستور (دکمه) ارتباط گرافیکی IDL را با نرم افزار ENVI از بین برده و در اینصورت شما دیگر قادر به استفاده از دستورات گرافیکی ENVI قابل لود در IDL نیستید و درصورت استفاده با پیغام خطا مواجه خواهید شد.

Scalers

Special Values= Constants

- pi: π value up to 15 significant digits
- i, j: sqrt(-1)
- Inf: infinity (such as division by 0)
- NaN: Not-a-Number (division of zero by zero)
- eps: $\in$ = 2.2204e-016, smallest amount by which 2 numbers can differ
- ans: stores the result of an expression

MATLAB	IDL	Purpose
pi	!PI, !DPI	π
	!RADEG	$180/\pi \approx 57.2958$
	!VALUES.F_INFINITY	∞
	!VALUES.F_NAN	Not a number
inf	!VALUES.D_INFINITY	
NaN	!VALUES.D_NAN	
A = [1,2,3]	A = [1,2,3]	Forming complex arrays
B = [4,5,6]	B = [4,5,6]	
C=A+i*B	C = COMPLEX(A, B)	
i *or* j	COMPLEX(0, 1.)	$\sqrt{-1}$
clear all	.RESET_SESSION	Clear all variables
clear A	DELVAR, A	Clear variable

Create a Matrix

```
A1= INDGEN(5)

      ;0        1        2        3        4
      ;A1                INT          = Array[5]
```

```
A1= LINDGEN(5,START=3)
A1= FINDGEN(5,START=3)  ──────►  A1= INDGEN(5,START=3, /FLOAT)

      ; 3.00000      4.00000      5.00000      6.00000      7.00000
      ; A1               FLOAT      = Array[5]
```

```
A1= CINDGEN(3)      ; Z=X+iY  ──►  A1= DCINDGEN(3)  ;Double COMPLEX=Array[3]
A1= BINDGEN(3,3)    ; Matrix 3*3 of byte
A1= DINDGEN(4,INCREMENT=0.5, START=2)

      ;  2.0000000      2.5000000      3.0000000      3.5000000
      ;  A1                DOUBLE     = Array[4]
```

Create a Matrix

```
A2= MAKE_ARRAY(3,4,/INTEGER)                ; A 3*4 Matrix with element=0
A2= MAKE_ARRAY(3,4,/INTEGER, VALUE = 5)     ; A 3*4 Matrix with element=5

A2= INTARR(3,4)                             ; A 3*4 Matrix with Integer element=0.00
A2= LONARR(3,4)                             ; A 3*4 Matrix with Long element=0.00

A2= FLTARR(3,4)                             ; A 3*4 Matrix with Floating element=0.00
A2= FLTARR(3,4, /NOZERO)                    ; A 3*4 Matrix with NON-ZERO Floating elements

A2= DBLARR(3,4)                             ; A 3*4 Matrix with double element=0.00000
A2= DBLARR(3,4, /NOZERO)                    ; A 3*4 Matrix with NON-ZERO double elements

A2= STRARR(3,4)                             ; A 3*4 Matrix with element= Nothing
A3= RANDOMU(Seed,3,4,/NORMAL)
A3= RANDOMN(Seed,3,4)

;    -1.17849      0.617972     -0.876464
;    -1.32892     -0.0852161     2.03960
;    0.797845      0.175403      0.427904
;    0.0531700    -0.608210     -1.49965

;     A3              FLOAT     = Array[3, 4]
```

Variables- Examples

```
A1= make_array (3,4, value=5)*!PI
A2=indgen (3,4)*!PI
A3= [[1,2,3],[4,5,6],[7,8,9],[10, 11, 12]]
A4= indgen (5)
A5= indgen (2,3,4)
```

تغییر فرمت

برای تغییر فرمت سه روش عمده موجود است:

۱. ترکیب متغیر مغلوب با متغیرهای غالب
۲. استفاده از دستورات تغییر تعداد ارقام اعشار و گرد کردن و ...
۳. استفاده از توابع و دستورات تغییر فرمت (Dot Typeها)

تغییر فرمت

1. Variables in IDL are ‘dynamically cast’, meaning that operations on those variables can change the type of the variable. For instance, say I give the following instructions to IDL:

```
a = 5
print, a
b = a / 2
print, b
b = a / 2.0
print, b
```

We see that the first time that we divide a by 2 we get 2 because b is assumed to be integer. But if we divide a by 2.0 then b will be automatically changed to a floating variable.

```
A= 1024*768
print, A
help, A
```

```
A= 1024 * 768L
print, A
help, A
```

```
A= 1024D * 768.0
print, A
help, A
```

```
A= long(1024 * 768)
print, A
help, A
```

تغییر فرمت

2. You can also explicitly change the type of a variable using the commands fix, round, long, float, and others. A caution is warranted when forcing the type of a mathematical statement. IDL will always evaluate the statement first and then attempt to change the type of the result. This often leads to undesirable results.

دستور	توضیح	مقدار	فرمت
!PI	عدد ثابت پی	3.1415927	FLOAT
Fix(!PI)	رند کردن به سمت عدد صفر	3	INTEGER
Round(!PI)	رند کردن به سمت نزدیک ترین عدد صحیح	3	LONG
Floor(!PI)	صحیح کردن به سمت منفی بینهایت (کف)	3	LONG
Ceil(!PI)	صحیح کردن به سمت مثبت بینهایت (سقف)	4	LONG

تغییر فرمت

2. You can also explicitly change the type of a variable using the commands fix, round, long, float, and others. A caution is warranted when forcing the type of a mathematical statement. IDL will always evaluate the statement first and then attempt to change the type of the result. This often leads to undesirable results.

دستور	توضیح	مقدار	فرمت
A=121	عدد ثابت یک	121	INT
Long(A)	صحیح کردن به سمت منفی بینهایت (کف) و حذف اعشار	121	LONG
Float(A)	عدد اعشاری کوتاه رقم	121.000	FLOAT
Double(A)	عدد اعشاری بلند رقم	121.00000	DOUBLE
String(A)	تغییر به یک عبارت نوشتاری (رشته ای)	121	STRING
byte(A)	تبدیل به حالت بایتی سیستمی	4	BYTE

تغییر فرمت

3. You can also explicitly change the type of a variable using the commands **DotTypes**.

```
A=121
B=A.ToDouble()
C=A.ToBits()
D=A.ToBinary()
E=A.ToInteger()
F=A.ToString()
G=A.Convert(/LONG [/INT] [/FLOAT] [/DOUBLE] [/STRING] [/BYTE])
```

Scalers

Vectors

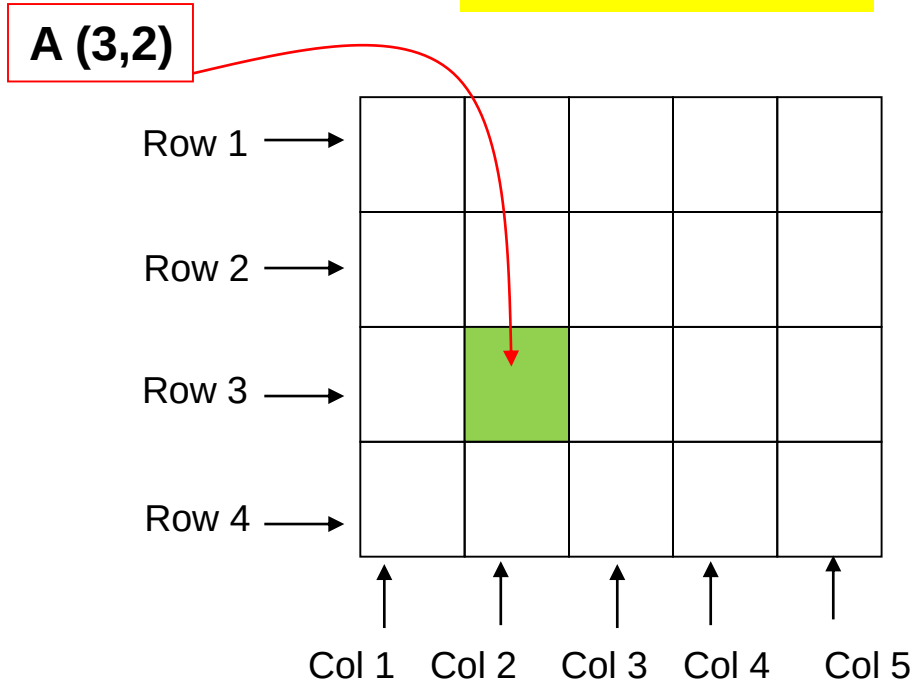
Matrices

Arrays & Assigning an element

- **Array:** A collection of data values organized into rows and columns, and known by a single name.

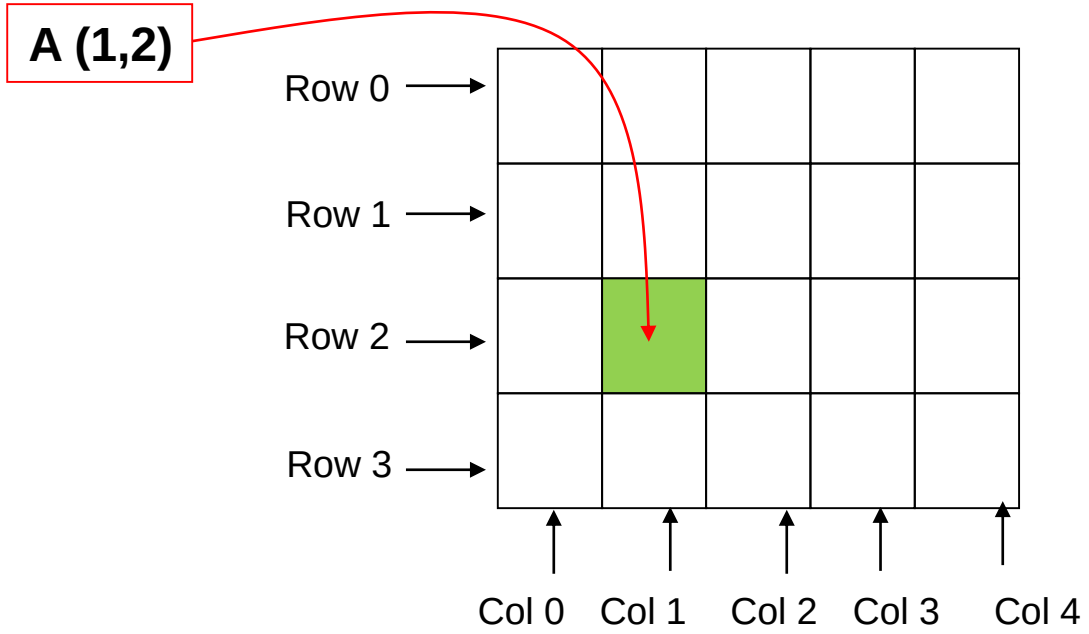
In MATLAB

A= Matrix (4*5)



In IDL

A= Matrix (5*4)



Arrays & Assigning an element

- **Array:** A collection of data values organized into rows and columns, and known by a single name.

In MATLAB

A= Matrix (4*5)

A (7)

1	5	9	13	17
2	6	10	14	18
3	7	11	15	19
4	8	12	16	20

In IDL

A= Matrix (4*3)

A (11)

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

Create a 1D Matrix

```
A1= INDGEN (5,START=3)   >> 16 Bite
A2= LINDGEN(5,START=3)   >> 32 Bite
A2= FINDGEN(5,START=3)   >> 32 Bite
A2= DINDGEN(5,START=3)   >> 64 Bite
A2= BINDGEN(5,START=3)
A2= CINDGEN(5,START=3) A2=DCINDGEN(5,START=3)
```

Create a 2D Matrix

```
A2= MAKE_ARRAY(3,4,/INTEGER)                ;A 3*4 Matrix with element=0 and integer type
A2= MAKE_ARRAY(3,4,/double, VALUE = 5)      ;A 3*4 Matrix with element=5.0000 and double type
```

```
A2= indgen  (3,4) * !PI
```

```
;       0.000000      3.14159      6.28319
;       9.42478       12.5664      15.7080
;       18.8496       21.9911      25.1327
;       28.2743       31.4159      34.5575
;
;       A2     Double          = Array[3, 4]
```

```
A2= [ [1,2,3] , [4,5,6] , [7,8,9] , [10, 11, 12] ]  ⟶  A2= MAKE_ARRAY(3,4)+1
```

```
;          1         2         3
;          4         5         6
;          7         8         9
;          10        11        12

;         A2     INT     = Array[3, 4]
```

Create a 3D Matrix

```
A2= MAKE_ARRAY (2,3,3,  VALUE = 5)          ; A 3*4*5 Matrix with element=5

print, A2
help,  A2
```

```
       5       5
       5       5
       5       5

       5       5
       5       5
       5       5

       5       5
       5       5
       5       5
A2              INT       = Array[2, 3, 3]
```

Create a 3D Matrix

```
A2= DBLARR(2,3,4)                    ; A 1*2*3 Matrix with Double element=0
```

```
print, A2
help,  A2
```

```
      0.00000000     0.00000000
      0.00000000     0.00000000
      0.00000000     0.00000000

      0.00000000     0.00000000
      0.00000000     0.00000000
      0.00000000     0.00000000

      0.00000000     0.00000000
      0.00000000     0.00000000
      0.00000000     0.00000000

      0.00000000     0.00000000
      0.00000000     0.00000000
      0.00000000     0.00000000

A2              INT       = Array[2, 3, 4]
```

Vector Addressing

Vector Addressing : with an integer index enclosed in parentheses.

```
>> A2= indgen(5)
>> AA=size(A2)
>> AAA=A2[3,0]          ; Also You can use brackets instead of parentheses like A2(3,0).
```

```
>> print, A2, AA, AAA
>> help,  A2, AA
```

```
A2 =      0       1       2       3       4
AA=       1       5       2       5
AAA=      3

A2 :      INT     = Array[5]
AA :      LONG    = Array[4]
```

Vector Addressing

Vector Addressing : *with an integer index enclosed in brakets.*

```
>> A2= indgen(3,4)
>> AA=size(A2)
>> AAA=A2[5]
>> A3=A2[*,1:2]
>> A4=A2[0,*]
```

```
>> print, A2, AAA, A3, A4
>> help,  A2, AAA, A3, A4
```

```
A2 = 0    1    2
     3    4    5
     6    7    8
     9   10   11
AAA=  5
A3=  3    4    5
     6    7    8
A4=  0    3    6    9

     AA :      INT     = Array[3, 4]
     AAA:      INT     =       5
     A3:       INT     = Array[3,2]
     A4:       INT     = Array[4]
```

Matrix: Size

```
>> A2= indgen (2,3,4)
>> AA=size(A2)
>> A3=A2[*,0,*]
>> A4=reform(A3,4,2)

>> print, A2, AA
>> help,  A2, AA
```

```
A2=

0       1
2       3
4       5

6       7
8       9
10      11

12      13
14      15
16      17

18      19
20      21
22      23
```

```
AA=

3       2       3       4       2       24
```

```
A3=

0       1

6       7

12      13

18      19
```

```
A4=

0       1       6       7
12      13      18      19
```

```
A2:     INT     = Array[2, 3, 4]
AA:     INT     = Array[6]
A3:     INT     = Array[2, 1, 4]
A4:     INT     = Array[4, 2]
```

Matrix: Size

```
>> A2= indgen (2,3,4)
>> AA=size(A2)
>> A3=A2[*,0,*]
>> A4=reform(A3,2,4)

>> print, A2, AA
>> help,  A2, AA
```

```
AA=size(A2)    ;   INT        = Array[6]

AA=   3    2    3    4    2    24
```

The number of dimensions: zero if Expression is scalar

Dimensions

- Size of Y dimension (Column): (none if Expression is scalar or undefined).
- Size of X dimension (Row): (none if Expression is scalar or undefined).
- Size of Z dimension (Layers): (none if Expression is scalar or undefined).

Type code (zero if undefined) in refer to slide 10.

Number of elements in Expression

Matrix: Size

Examples

Print the size information for a 10 by 20 floating-point array by entering:

```
PRINT, SIZE(FINDGEN(10, 20))
```

IDL prints:

```
    2    10    20     4    200
```

This IDL output indicates the array has 2 dimensions, equal to 10 and 20, a type code of 4, and 200 elements total.
Similarly, to print only the number of dimensions of the same array:

```
PRINT, SIZE(FINDGEN(10, 20), /N_DIMENSIONS)
```

IDL prints:

```
    2
```

Logical Operators

- Logical operators are often used in command loops and also mathematical statements. These operators return either a value of 1 if the statement they evaluate is true or 0 if false.
- There are two kinds of logical operators: Boolean operators and Relational operators.

Boolean Operators:

- There are four Boolean operators in IDL. They take arguments of 1 (true) or 0 (false) and return either a 1 or a 0. The operators are: **AND, OR, XOR, NOT**.
- These are all straightforward with the possible exception of XOR which stands for exclusive OR, meaning that one and only one of the arguments may be true for XOR to be true. Consider the following examples:

1 and 1 = 1	1 and 0 = 0	0 and 1 = 0	0 and 0 = 0
1 or 1 = 1	1 or 0 = 1	0 or 1 = 1	0 or 0 = 0
1 xor 1 = 0	1 xor 0 = 1	0 xor 1 = 1	0 xor 0 = 0
not 1 = 0	not 0 = 1		

Logical Operators

- Logical operators are often used in command loops and also mathematical statements. These operators return either a value of 1 if the statement they evaluate is true or 0 if false.
- There are two kinds of logical operators: Boolean operators and Relational operators.

Relational Operators

- There are six relational operators in IDL. They compare numbers which can be integer or decimal. These return 1 for true and 0 for false. The relational operators are: **EQ, NE, LE, LT, GE, GT**.

1.	**EQ** is 'equals'	5 eq 5 = 1	5 eq 4 = 0	
2.	**NE** is 'not equal'	5 ne 5 = 0	5 ne 4 = 1	
3.	**LE** is 'less than or equal'	5 le 5 = 1	5 le 4 = 0	5 le 6 = 1
4.	**LT** is 'less than'	5 lt 5 = 0	5 lt 4 = 0	5 lt 6 = 1
5.	**GE** is 'greater than or equal'	5 ge 5 = 1	5 ge 4 = 1	5 ge 6 = 0
6.	**GT** is 'greater than'	5 gt 5 = 0	5 gt 4 = 1	5 gt 6 = 0

- These operators can be used in conjunction with each other. (10 le 50) and (20 ge 25) = 0

Mathematical Operators

IDL has all of the familiar mathematical operators (^*/+-) and a few additional ones for matrix mathematics. These operators in combination with the logical operators detailed above follow a system of algebraic precedence. Precedence is what determines which order that operators act within an statement. Operations of the same order of precedence are generally commutative so that ambiguity in their order of execution is usually irrelevant. It is recommended that for complex statements where is order of operation is unclear that parenthesis be used even if not explicitly necessary. If the arguments of +-*/ operators are both integer, the result will be integer as well. If one or two of the operators are decimal, the result will also be a floating decimal. The operators will be listed and briefly explained in the order of their precedence.

1. Parentheses ( )
2. Exponentiation ^
3. Multiplication * or Division / or Modulo (mod)
4. Addition + or Subtraction – or Logical
5. Relational Operators (eq,ne,ge,gt,le,lt)
6. Boolean Operators (And, Or, Not, Xor)

Elementry Function

Trigonometric Functions

SIN Sine (argument in radians)	sin(1.57080)=1
COS Cosine (argument in radians)	cos(1.57080)=0
TAN Tangent (argument in radians)	tan(0.785398)=1
ACOS Inverse Cosine (in radians)	acos(0)= 1.57080
ASIN Inverse Sine (in radians)	asin(1)= 1.57080
ATAN Inverse Tangent (in radians)	atan(1)= 0.785398
SINH Hyperbolic Sine	sinh(1.5) = 2.12928
COSH Hyperbolic Cosine	cosh(.5) = 1.12763
TANH Hyperbolic Tangent	tanh(1.5)= 0.905148

Elementry Function

Complex functions

ABS Absolute value abs(-6)=6

COMPLEX(Real [, Imaginary] [, /DOUBLE]) A = [1,2,3],B = [4,5,6], C = COMPLEX(A, B) (1.00000, 4.00000)(2.00000, 5.00000) (3.00000, 6.00000)

DCOMPLEX Creates a double complex array.

REAL_PART Is the real parts of the complex array REAL_PART(C) 1.00000 2.00000 3.00000

IMAGINARY Is the imaginary parts of the complex IMAGINARY(C) 4.00000 5.00000 6.00000

CONJ Conjugate of the complex

COMPLEXARR Creates an empty, X-element by Y-element by …, complex array COMPLEXARR (5,5,3)

DCOMPLEXARR Creates an empty, X-element by Y-element by …, double complex array DCOMPLEXARR (5,5,3)

COMPLEXROUND Rounding the real and imaginary components of the input array

Elementry Function

Exponential functions

SQRT Square Root	sqrt(16) = 4
EXP Natural Exponent (e)	exp(1)= 2.71828
ALOG Natural Log (ln(x))	alog(2.71828) = 1
ALOG2 Log Base 2	alog(16) = 4
ALOG10 Log Base 10	alog(100) = 2

Elementry Function

Array Or Matrix functions

In the following examples, ‘a’ is a row vector [1,3,5,2,4]

MAX Maximum Value of an Array — max(a) = 5

MAX(Array [, /ABSOLUTE] [, DIMENSION=value] [, MIN=variable])

MIN Minimum Value of an Array — min(a) = 1

MEAN The Mean of an Array — mean(a) = 3

TOTAL Sum of Matrix Operations — TOTAL(Array [, Dimension] [, /CUMULATIVE])

N_ELEMENTS Returns the number of elements contained in an expression or variable.

MEDIAN Computes the median value. — Median(a) = 5

MEDIAN (Array [, Width] [, /DOUBLE] [, DIMENSION=value])

SKEWNESS Returns the floating point or double precision statistical skewness.

SKEWNESS([indgen(5),20]) = 1.25279

Elementry Function

Array Or Matrix functions

In the following examples, ‘a’ is a row vector [1,3,5,2,4]

REVERSE Reverses the order of one dimension of an array. The default dimension is row. reverse(a,1) = reverse(a)

STDDEV Computes the standard deviation of a vector. Stddev(a) = 1.58114 انحراف معیار

VARIANCE Computes the variance of a vector. VARIANCE(a) = 2.50000 واریانس

SORT Returns a vector of integer type with the same number of elements as Array after ascending order sorting.

sort(a) = [0 3 1 4 2]

in ascending order: a[sort(a)] = [1 2 3 4 5]

in descending order: a[reverse(sort(a))] = [5 4 3 2 1]

MEANABSDEV Compute average deviation from the mean. MEANABSDEV(a) = 1.20000

SIZE Return the size of an Array in several Items that describe in slide 31.

PRIMES Computes First n Prime #’s primes(5)= 2,3,5,7,11

Elementry Function

Array Or Matrix functions

In the following examples, ‘a’ is a row vector [1,3,5,2,4]

FACTORIAL Factorial (!)

factorial(4) = 24

INVERT Uses the Gaussian elimination method to compute the inverse of a square array. In working with complex inputs, use the LA_INVERT function instead.

```
invert(findgen(3,3,START=5)) =
[1048574.6     -2097149.8      1048574.8
-2097155.0      4194306.5     -2097152.0
 1048579.9     -2097156.0      1048577.0]
```

DETERM Compute the determinant of an *n* by *n* array. In working with complex inputs, use the LA_DETERM procedure instead.

PRODUCT Returns the product of elements within an array over a given dimension or in row by default.

```
product(a) = 120.00000
```

MATRIX_MULTIPLY Calculates the IDL # operator of two (possibly transposed) arrays.

Elementry Function

Array Or Matrix functions

The # Operator vs. MATRIX_MULTIPLY

The following table illustrates how various operations are performed using the # operator versus the MATRIX_MULTIPLY function:

# Operator	Function
A # B	MATRIX_MULTIPLY(*A*, *B*)
transpose(A) # B	MATRIX_MULTIPLY(*A*, *B*, /ATRANSPOSE)
A # transpose(B)	MATRIX_MULTIPLY(*A*, *B*, /BTRANSPOSE)
transpose(A) # transpose(B)	MATRIX_MULTIPLY(*A*, *B*, /ATRANSPOSE, /BTRANSPOSE)

MATRIX_MULTIPLY can also be used in place of the ## operator. For example, A ## B is equivalent to MATRIX_MULTIPLY(B, A), and A ## TRANSPOSE(B) is equivalent to MATRIX_MULTIPLY(B, A, /BTRANSPOSE).

Syntax:

Result = MATRIX_MULTIPLY(*A*, *B* [, /ATRANSPOSE] [, /BTRANSPOSE])

Elementry Function

Array Or Matrix functions

In the following examples, ‘a’ is a row vector [1,3,5,2,4]

MATRIX_POWER Computes the product of an square matrix with itself. For example, the fifth power of array *A* is *A* # *A* # *A* # *A* # *A*. A power of zero returns the identity matrix and a power of one returns the itself.

```
MATRIX_POWER([[1,2],[3,4]],4) =
      [199.00000      290.00000
      435.00000      634.00000]
```

ROTATE Rotates an array counterclockwise in multiples of 90 degrees. To rotate by amounts other than multiples of 90 degrees, use the ROT function.

ROT Rotates an image or array by an arbitrary amount. At the same time, it can magnify, demagnify, and/or translate an it. If you want to rotate an array by a multiple of 90 degrees, you should use the ROTATE function for faster results.

Elementry Function

Array Or Matrix functions

In the following examples, ‘a’ is a row vector [1,3,5,2,4]

TRACE Computes the trace of an n by n array. (Multiple of an array diag_elements) (مجموع درایه های روی قطر اصلی)

```
A =       [[ 2.0,1.0, 1.0, 1.5], $
          [ 4.0, -6.0, 0.0, 0.0], $
          [-2.0, 7.0, 2.0, 2.5], $
          [ 1.0, 0.5, 0.0, 5.0]]
TRACE(A) =  3.00000
```

TRANSPOSE Returns the transpose of Array.

```
TRANSPOSE(INDGEN(3,3))
```

IDENTITY Returns an identity array (an array with ones along the main diagonal and zeros elsewhere) of the specified dimensions.

DIAG_MATRIX Constructs a diagonal matrix from an input vector, or if given a matrix, then DIAG_MATRIX will extract a diagonal vector.

Elementry Function

Strings functions

STRING The STRING function returns its arguments converted to string type.

```
HELP, STRING(INDGEN(5))
PRINT, STRING([72B, 101B, 108B, 108B, 111B])
IDL> HELLO
```

STRLEN Returns the length of its string-type argument. If the argument is not a string, it is first converted to string type.

```
PRINT, STRLEN('IDL is fun')
```

STRJOIN Returns the merged strings.

STRSPLIT Splits its input *String* argument into separate substrings, according to the specified delimiter or regular expression.

By default, an array of the position of the substrings is returned.

```
Str = 'STRSPLIT chops up strings.'
STRSPLIT(Str,/EXTRACT)

str = 'Out, damned spot! Out I say!'
print, (STRJOIN(STRSPLIT(str, /EXTRACT), ':'))
Out,:damned:spot!:Out:I:say!
```

Elementry Function

Strings functions

```
Str = 'STRSPLIT,chops,up,strings.'
print, STRJOIN(STRSPLIT(Str,',', /EXTRACT), '-')
IDL> STRSPLIT-chops-up-strings.
```

STRPOS Finds the first occurrence of a substring within an object string.

Syntax: -> STRPOS(*Expression*, *Search_String* , /REVERSE_SEARCH)

```
PRINT, STRPOS('IDL is fun', 'fun')
```

STREGEX → Works like *strops* but with a large domain words searching!

STRMID Extracts one or more substrings from a string expression. Each extracted string is the result of removing characters from the input string. Use "**/REVERSE_OFFSET**" Specifies that *First_Character* should be counted from the end of the string rather than the beginning. *Length* then proceeds from this position to the right, toward the end of the string. This allows simple extraction of strings from the end.

```
myString = "IDL is fun"
subString = STRMID(myString, 4, 2)
IDL> is
```

Elementry Function

Strings functions

```
myQuote = "It is not in the stars to hold our destiny but in ourselves."
extractedStr = STRMID(myQuote, 29, 12, /REVERSE_OFFSET)
PRINT, extractedStr
IDL> our destiny
```

STRTRIM removes leading and/or trailing blank from the input *String*. Flag is a value that controls the action of STRTRIM.

If *Flag* is zero or not present, trailing blanks are removed. Leading blanks are removed if it is equal to 1. Both are removed if it is equal to 2.

Syntax: -> STRTRIM(*String* [, *Flag*])

```
STRTRIM(STRING(56), 0)
STRTRIM(STRING(56), 1)
STRTRIM(STRING(56), 2)
```

STRUPCASE Returns a copy of *String* converted to upper case. Only lowercase characters are modified—uppercase and non-alphabetic characters are copied without change.

```
PRINT, STRUPCASE('IDL is fun')
IDL> IDL IS FUN
```

Elementry Function

Strings functions

STRLOWCASE Returns a copy of *String* converted to lowercase characters too.

STRPUT Inserts the contents of one string into another. The source string, *Source*, is inserted into the destination string, *Destination*, starting at the given position, *Position*. Characters in *Destination* before the starting position and after the starting position plus the length of *Source* remain unchanged. The length of the destination string is not changed. If the insertion extends past the end of the destination, it is clipped at the end.

```
A='IBM is FUN'
STRPUT, A, 'IDL', 0
STRPUT, A, 'FUNNY', 7
IDL> IBL is FUN
```

STRMATCH Compares its search string, which can contain wildcard characters, against the input string expression.

```
str = ['foot', 'Feet', 'fate', 'FAST', 'ferret', 'fort']
PRINT, str[WHERE(STRMATCH(str, 'f??t', /FOLD_CASE) EQ 1)]
IDL> foot Feet FAST fort

PRINT, str[WHERE(STRMATCH(str, 'f*t', /FOLD_CASE) EQ 1)]
IDL> foot Feet FAST ferret fort
```

Elementry Function

Strings functions

Wildcard Character	Description
*	Matches any string, including empty strings.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by “-” matches any character lexically between the pair, inclusive. If the first character following the opening [is a !, any character not enclosed is matched.
To prevent one of these characters from acting as a wildcard, precede it with a backslash character (e.g. "*" matches the asterisk character). Quoting any other (non-wildcard) character (including \ itself) is equivalent to the character (e.g. "\a" is the same as "a").	

STRCMP Performs string comparisons between its two String arguments, returning True (1) for those that match and False (0) for those that do not. Normally *String1* and *String2* are compared in their entirety. If *N* is specified, the comparison is made on at most the first *N* characters of each string. String comparison is normally a case sensitive operation. Set FOLD_CASE to perform case insensitive comparisons instead.

Syntax: -> *Result* = STRCMP(*String1*, *String2* [, *N*], /FOLD_CASE)

```
PRINT, STRCMP('Moose', 'moo', 3, /FOLD_CASE)
```

Elementry Function

Strings functions

STRCOMPRESS Returns a copy of *String* with all whitespace (blanks and tabs) compressed to a single space or completely removed.

```
; Create a string variable S:
S = 'This is a string with spaces in it.'
; Print S with all of the whitespace removed:
PRINT, STRCOMPRESS(S, /REMOVE_ALL)
IDL> Thisisastringwithspacesinit.
```

Elementry Function

Directory functions

CD The CD procedure is used to set and/or change the current working directory.

Syntax: -> CD [, *Directory*] [, CURRENT=*variable*]

To specify a full path:

```
CD, '/home/data/'
```

To change to the january:yrotcerid tnerruc eht fo yrotceridbus

```
CD, 'january'
```

To change drives:metsys swodniW a no

```
CD, 'C:'
```

To go back up a directory, use pu og dluoc uoy ,yraunaj/atad/emoh/ si yrotcerid tnerruc eht fi ,elpmaxe roF ."‥" :dnammoc gniwollof eht htiw yrotcerid /atad/emoh/ eht ot

```
CD, '..'
```

If the current directory is htiw yrotcerid yraurbef/atad/emoh/ eht ot egnahc dluoc uoy ,yraunaj/atad/emoh/ :dnammoc gniwollof eht

```
CD, '../february'
```

To mimic the UNIX pwd:dnammoc

```
CD, CURRENT=c & PRINT, c
```

Elementry Function

Time functions

SYSTIME Returns the specified time.

```
SYSTIME()
SYSTIME(/SECONDS)
```

WAIT Suspends execution of an IDL program for a specified period. Its very useful in order to reading or writing a file or running an other program before executing next lines code.

```
Wait,5.5
```

TIC … TOC These routines works together to allow you to check the running time of your IDL programs.

EXTRAC Function

The EXTRAC function returns a defined portion of an array or vector.

The main advantage to EXTRAC is that, when parts of the specified subsection lie outside the bounds of the array, zeros are returned for these outlying elements. It is usually more efficient to use the array subscript ranges (the ":" operator) to perform such operations.

Examples

Extracting elements from a vector:

```
; Create a 1000 element floating-point vector with each element set
; to the value of its subscript:
A = FINDGEN(1000)
; Extract 300 points starting at A[200] and extending to A[499]:
B = EXTRAC(A, 200, 300)
```

EXTRAC Function

The following commands illustrate the use of EXTRAC with multi-dimensional arrays:

```
A = indgen (4,5)          ; Make a 4  by   5 array:

B = EXTRAC (A, 1,2,2,2)   ; Extract a 2 by 2 portion starting at A(1,2). You can Use the array $
                          subscript operator instead of EXTRAC:

B = A (1:2,2:3)
```

```
A=              0       1       2       3
                4       5       6       7
                8       9       10      11
                12      13      14      15
                16      17      18      19

 A:           INT       = Array[4, 5]
               9       10
               13      14

 EXRCT_A :    INT       = Array[2, 2]
```

```
Sum_A= total(A)
Diag_A= DIAG_MATRIX(A)
Trnspos_A= transpose(A)
exrct_A= EXTRAC(A,1,2,2,2)
```

REFORM Functions

The REFORM function changes the dimensions of an array without changing the total number of elements.

Examples

REFORM can be used to remove “degenerate” leading dimensions of size one. Such dimensions can appear when a subarray is extracted from an array with more dimensions. For example a is a 3-dimensional array:

```
a = INTARR(10,10,10)
b = a[5,*,*]                    ; Extract a "slice" from a:
HELP, b, REFORM(b)              ; Use HELP to show what REFORM does:
b = REFORM(a,200,5)      ;They create a new array, b, with dimensions of (200, 5), from a.:
b = REFORM(a,[200,5])
```

```
Executing the above statements produces the output:

        B               INT = Array[1, 10, 10]
   <Expression>         INT = Array[10, 10]
```

Where Function

- The ***WHERE*** function returns a vector that contains the one-dimensional subscripts of nonzero elements of Array_Expression.
- The length of the resulting vector is equal to the number of nonzero elements in Array_Expression.
- Frequently the result of WHERE is used as a vector subscript to select elements of an array using given criteria.

Example:

```
; Create a 10-element integer array where each element is  set to the value of its subscript:
    array = INDGEN(10)
    PRINT, 'array = ', array

; Find the subscripts of all the elements in the array that have  a value greater than 5:
    B = WHERE(array GT 5, count, COMPLEMENT=B_C, NCOMPLEMENT=count_c)

; Print how many and which elements met the search criteria:

    PRINT, 'Number of elements > 5: ', count
    PRINT, 'Subscripts of elements > 5: ', B
    PRINT, 'Number of elements <= 5: ', count_c
    PRINT, 'Subscripts of elements <= 5: ', B_C
```

MATLAB Vs IDL

MATLAB	IDL	Purpose
	`info=ROUTINE_INFO(sin)` & PRINT, info.path	Get info on a function or procedure
`pause(3)`	`WAIT(3)`	Perform a 3 second pause
`cd`	`CD`	Change directory
`cd ..`	`CD, '::'`	Change to upper directory (MacOS)
`ls`	`PRINT, FINDFILE('*')`	List files
`pwd`	`CD, C=c & PRINT, c`	Path of current directory

MATLAB	IDL
`>`	`GT`
`<`	`LT`
`==`	`EQ`
`>=`	`GE`
`<=`	`LE`
`~=`	`NE`

MATLAB Vs IDL

MATLAB	IDL	Purpose
x’;	TRANSPOSE(x)	Transpose
x(3:5);	x[3:5]	Vector portion
A*B;	A##B	Matrix multiplication
A.*B;	A*B	Multiplication
A.^2;	A^2	Exponentiation
A^2;	A##A	Exponentiation
%	;	Comment
...	$	Line continuation
A=[1,2,3;4,5,6];	A=[[1,2,3],[4,5,6]]	Matrix formation
B=A(:,2:3);	B=A[1:2,*]	Submatrix extraction
x=0:9;	x=INDGEN(10)	Integer vector
x=0.0:1.0:9.0;	x=FINDGEN(10)	Float vector
x=byte(0:255);	x=BINDGEN(256)	Byte vector
x=0.0:1.0:9.0;	x=CINDGEN(10)	Complex vector
sum(x);	TOTAL(x)	Sum of vector
sum(sum(A));	TOTAL(A)	Sum over all elements in matrix
x	PRINT, x	Print vector x
z=x;y=2*x;	z=x & y=2*x	Line continuation
size(mat)	SIZE(mat,/DIMENSIONS)	Size of a matrix
length(vec)	N_ELEMENTS(vec)	Length of a vector
linspace(0,pi,100)		Linearly spaced vector
fliplr(A)	REVERSE(A,1)	Flip columns
flipud(A)	REVERSE(A,2)	Flip lines
rot90(A)	ROTATE(A,1)	Linearly spaced vector
repmat	????	Replicate a matrix
10*ones(N)	res=REPLICATE(10,N,N)	Replicate a scalar

Some Commands Overview

1. ABS Absolute Value abs(-6)=6
2. ACOS Inverse Cosine (in radians) acos(0)= 1.57080
3. ALOG Natural Log (ln(x)) alog(2.71828) = 1
4. ALOG10 Log Base 10 alog(100)=2
5. ASIN Inverse Sine (in radians) asin(1)= 1.57080
6. ATAN Inverse Tangent (in radians) atan(1)= 0.785398
7. COS Cosine (argument in radians) cos(1.57080)=0
8. COSH Hyperbolic Cosine cosh(.5) = 1.12763
9. EXP Natural Exponent (e) exp(1)= 2.71828
10. FACTORIAL Factorial (!) factorial(4) = 24

1. FFT Fast Fourier Transform see IDL help on FFT
2. FIX Truncates Decimal to Integer fix(3.8) = 3
3. FLOAT Changes Integer to Decimal float(3) = 3.000
4. MAX Maximum Value of an Array max(a) = 5
5. MEAN The Mean of an Array mean(a) = 3
6. MIN Minimum Value of an Array min(a) = 1
7. !PI System Constant for pi !PI = 3.141593
8. PRIMES Computes First n Prime #’s primes(5)= 2,3,5,7,11
9. RANDOMU Generates a Uniformly Distributed
10. Random # Between 0 and 1 randomu(1) = .04234

1. Random # Between 0 and 1 randomu(1) = .04234
2. ROUND Rounds a Number round(2.7) = 3
3. SIN Sine (argument in radians) sin(1.57080)=1
4. SINH Hyperbolic Sine sinh(1.5) = 2.12928
5. SQRT Square Root sqrt(16) = 4
6. TAN Tangent (argument in radians) tan(0.785398)=1
7. TANH Hyperbolic Tangent tanh(1.5)= 0.905148
8. TOTAL Sum of Matrix Operations see IDL help on TOTAL

Example

```
>> A= indgen (3,3)*!DPI
>> A_Invrs= invert (A)
>> A_Invrs= IMSL_INV(A)

>> Dimension = 2
>> Sum_A= total (A, Dimension, /CUMULATIVE)
>> Trnspos_A= transpose(A)

>> print, A
>> help,  A

>> print,  A_Invrs
 >> help,   A_Invrs
```

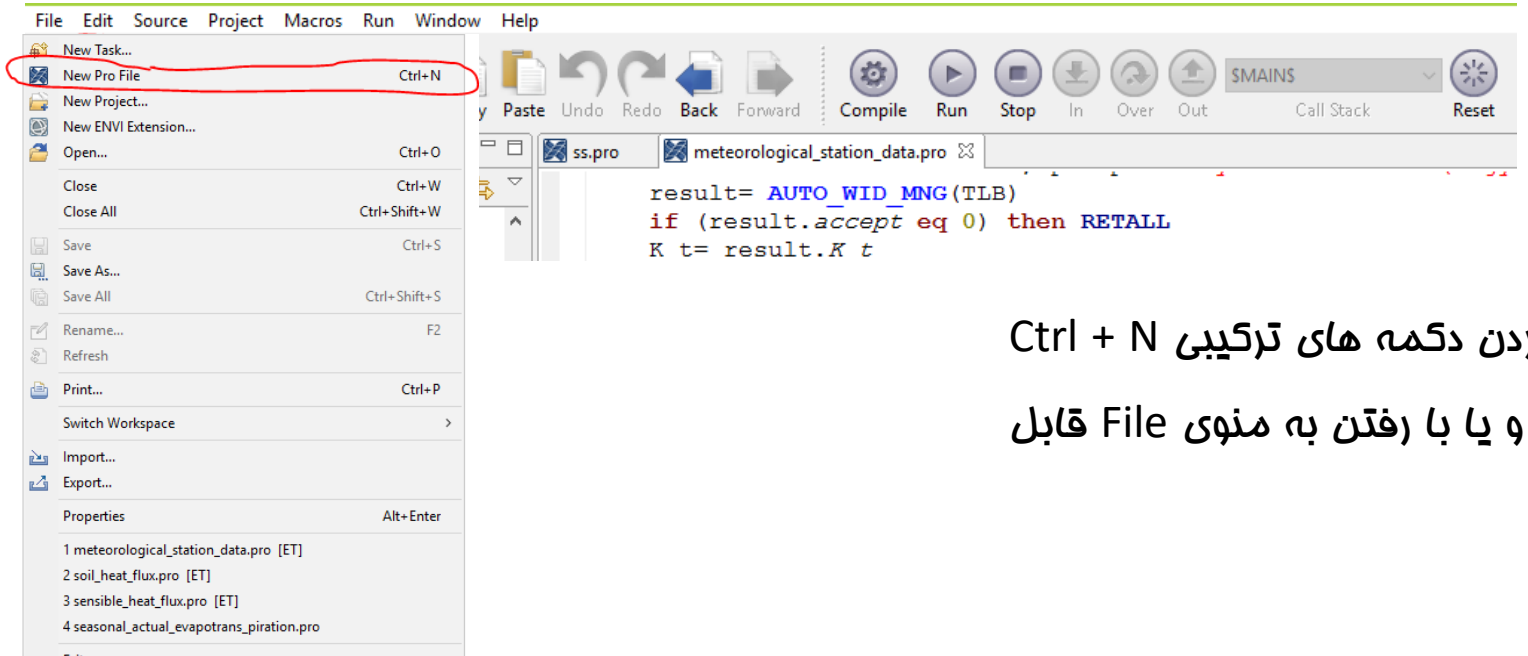
```
A=

0.000000      3.14159       6.28319
9.42478       12.5664       15.7080
18.8496       21.9911       25.1327

A:                DOUBLE      = Array[3, 3]

A_INVRS=

-524288.          1.04858e+006  -524288.
 1.04858e+006  -2.09715e+006    1.04858e+006
-524288.          1.04858e+006  -524288.

A_INVRS:          DOUBLE      = Array[3, 3]
```

Built in a Program Procedure

A project & New Pro_file

Built in a Program Procedure

از آنجایی که کدنویسی در محیط IDL Console بصورت موقتی بوده و با بستن برنامه IDL تمامی نتایج و کدهای نوشته شده پاک شده و دیگر قابل استفاده نیستند (یکبار مصرف هستند) بنابراین هر محیط کدنویسی ای دارای بخشی مخصوص است که در آنجا می توان کدها را بصورت پی در پی نوشته و سپس برای اجراهای آتی ذخیره نمود. این محیط ها با اسم N_File، M_File، محیط اسکریپت نویسی و ... شناخته می شوند.



در IDL به این محیط N_File می گویند که با فشردن دکمه های ترکیبی Ctrl + N و یا دکمه New Pro 

از نوار ابزار بالای صفحه و یا با رفتن به منوی File قابل دسترسی خواهد بود.

Built in a Program Procedure

پس از فشردن این دکمه صفحه سفیدی برای شما گشوده می گردد که شما در آن میتوانید تمامی دستورات مدنظر خود را بصورت خط به خط و طریقی منظم تایپ نمایید و آنها را ذخیره و یا برای اجرا آماده نمایید.

شروع هر برنامه نویسی در IDL با کلمه Pro و انتخاب یک نام برای برنامه بوده و با END خاتمه می یابد. فرمت کلی آن بشکل زیر است:

```
Pro Program_File_Name

  Statements…

END
```

در انتهای کار با فشردن دکمه های ترکیبی Ctrl+S و یا رفتن به منوی فایل گزینه Save یا Save as… و یا فشردن دکمه ذخیره 

در نوار ابزار می توانید برنامه خود را در پوشه جاری و یا هر دایرکتوری دیگری ذخیره نمایید.

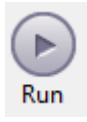
Built in a Program Procedure

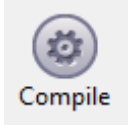
نکاتی که باید همواره بخاطر بسپارید:

۱. همواره نام انتخابی برای ذخیره فایل همان نام پیشنهادی باشد. بعبارتی همواره باید نام فایلی که ذخیره می شود با نام برنامه ای که در مقابل Pro در ابتدای برنامه نوشته اید یکسان باشد. در غیر اینصورت اجرای برنامه با خطا مواجه خواهد شد.

۲. هیچگاه از یک اسم برای دو برنامه متفاوت استفاده نکنید تا باعث بروز اشتباه و اجرای یکی بجای دیگری نشوید.

۳. انتخاب نام برنامه، تابع قوانین انتخاب نام برای متغیرها است. (با عدد شروع نمی شود و ...)

۴. اجرای یک برنامه با دکمه Run 

و یا کامپایل 

نمودن آن قبل از ذخیره، ابتدا پنجره ذخیره سازی را برای شما نمایان می نماید و مادامی که این عمل انجام نشود برنامه قابل اجرا نخواهد بود.

Help:

IDL Programming > Tasks > Using Procedures, Functions, SAVE Files > Defining a Procedure

A sequence of one or more IDL statements can be given a name, compiled, and saved for future use with the **procedure** definition statement. Once a **procedure** has been successfully compiled, it can be executed using a **procedure** call statement interactively from the terminal, from a main program, or from another **procedure** or function.
The general format for the definition of a **procedure** is as follows:

```
PRO  Name, Parameter1, ..., ParameterN
  ; Statements defining procedure.
  Statement1
  Statement2
  ...
; End of procedure definition.
END
```

The PRO statement must be the first line in a user-written IDL **procedure** and the END statement in the final line too.

Built in a Program Procedure

برنامه و زیربرنامه

در برنامه نویسی حرفه ای و زمانیکه تعداد خطوط کد نوشته شده در صفحه N_File زیاد می شود، معمولا از روش برنامه و زیر برنامه استفاده می شود.

در این روش قسمتهایی از برنامه بصورت زیربرنامه هایی جداگانه نوشته شده و ذخیره می شوند. در انتها با نوشتن یک برنامه کلی و اصلی و با استفاده مرحله به مرحله از اسم زیربرنامه ها، برنامه اصلی تکمیل و خروجی های مورد نظر ذخیره می شود.

این روش در خطایابی و فهم دقیقتر قسمتهای مختلف برنامه بصورت کتابخانه ای بسیار مفید می باشد و هر زیربرنامه با گرفتن دکمه Ctrl و کلیک چپ روی آن بصورت هایپرلینک باز شده و قابل مشاهده و ویرایش خواهد بود. توجه کنید که زیربرنامه های نوشته شده در محل برنامه اصلی ذخیره شوند.

Built in a Program Procedure

برنامه و زیربرنامه

شکل کلی و عمومی آن بشرح ذیل است:

```
Pro Main_Program

  Sub_Program1
  Sub_Program2
  Sub_Program3
  Sub_Program4
  …
  Print, Result

End
```

```
Pro   Sub_Program1
  S
  P
End
```

```
Pro   Sub_Program2
  S
  P
End
```

```
Pro   Sub_Program3
  S
  P
End
```

```
Pro   Sub_Program4

  Statements…
  Print, Result

End
```

Saving Variables

Saving Variables

ذخیره متغیرها بمنظور فراخوانی در قسمتی از یک زیربرنامه متفاوت و یا بمنظور استفاده مجدد از برنامه نوشته شده در زمانی دیگر (مشاهده گزینه ورودی منتخب کاربر از اجرای گذشته و ادامه روند اجرای برنامه مشابه اجرای گذشته) می تواند بسیار کارآمد باشد. بدین منظور دو شیوه متفاوت ذخیره سازی متغیرها بشکل زیر ارائه می گردد:

الف) استفاده از دستور Save و Restore

ب) استفاده از دستور Common در زیربرنامه های دیگر

Saving Variables- Save & Restore

برای ذخیره یکسری متغیر دارای مقدار پس از اجرای برنامه، معمولا در انتهای برنامه از دستور Save و نام متغیرها و اسمی برای ایجاد فایل خروجی در آدرس دایرکتوری جاری استفاده می شود. همچنین برای فراخوانی متغیرهایی که از اجرای قبلی ذخیره شده اند و استفاده مجدد آنها در اجرای برنامه جاری از دستور Restore بشکل زیر استفاده می شود.

```
A=Findgen(5,2)+2.0
B= transpose(A) ^ 2.0
C=B(2)- A(0)/2
save, A, B, C, filename = 'File_Name.sav'
…
Restore, A, B, C
…
```

Saving Variables- Save & Restore

حالت کلی تر استفاده از این دستور در اکثر برنامه ها بشکل زیر است:

```
A=''
B= 2.0
if file_test('File_Name.sav') then begin
    restore, 'File_Name.sav'
Endif
Statements …
save, A, B, filename= 'File_Name.sav'
```

با اولین اجرا متغیرها ذخیره می شود و در اجرای دیگر با کمک دستور File_Test('file_Name') وجود فایل ذخیره شده در فولدر جاری (Current Directory) با نام مشخص را بررسی نموده و در صورت وجود، متغیرهای ذخیره شده را فراخوانی و میزان آن را در اسم متغیر مربوط جایگزین می نماید.

Saving Variables- Common

برنامه نویسان برای استفاده از یک متغیر معرفی شده در یک زیربرنامه در زیربرنامه ای دیگر از شیوه ارجاع Common استفاده می کنند. برای این منظور رعایت سه نکته بسیار ضروری می باشد.

۱. استفاده از کلمه مرتبط Event در پیوند دو زیربرنامه در شروع هردو زیربرنامه
۲. اختصاص نام متغیر ارجاعی با پیشوند "S_" و معرفی متغیر حاوی مقدار در زیربرنامه اول
۳. استفاده از نام متغیر ارجاعی در زیربرنامه ای که میخواهیم متغیرهای ذخیره شده فراخوانی و در آنجا مورد استفاده قرار بگیرد.

The COMMON statement creates or references a common block. Common blocks are useful when there are variables that need to be accessed by several IDL procedures or when the value of a variable within a procedure must be preserved across calls. Once a common block has been defined, any program unit referencing that common block can access variables in the block as though they were local variables. Variables in a common statement have a global scope within procedures defining the same common block. Unlike local variables, variables in common blocks are not destroyed when a procedure is exited.

Saving Variables- Common

1

```
pro meteorological_station_data, event

  common S_Latitud,Latitud
  common S_Del,Del
  common S_Omega,Omega
```

2

```
    TGMT= 7.50
    Julian_Day= 121.0
    DA= (Julian_Day-1.0)*2.0*!PI/365.0
    Et_Minate= 0.000075+0.001868*COS(DA)-0.032077*SIN(DA)- 0.014615*COS(DA)- $
     0.04089*SIN(DA)
    Omega= !PI * ((TGMT + Longitud /15.0+Et_Minate/60.0) - 12.0) / 12.0
    Del= 0.006918-0.399912*COS(DA)+ 0.070257*SIN(DA)- 0.006758*COS(2.0*DA)+ $
     0.000907*SIN(2.0*DA)-0.002697*COS(3.0*DA) +0.00148*SIN(3.0*DA)

End
```

Saving Variables- Common

1

```
pro Next_Sub_Program, event

  common S_Latitud
  common S_Del                    3


   Final= (Del * 2.0 - 1.0) / Longitud
   Dialog_Message('The result will be: ',Final)

End
```

در این حالت دیگر در این زیربرنامه نیازی به معرفی دو متغیر Latitude و Del نیست و مقدار آن خودبخود شناسایی می شود.

نکته مهم: حتما باید قبل از اجرای این زیربرنامه، زیربرنامه قبلی زودتر و حداقل یکبار اجرا شده باشد و این مقادیر قابل شناسایی باشد.

Built in Functions

Built in Functions

نوشتن یک تابع در قالب Function معمولا می تواند از تکرار نوشتن و طولانی شدن خطوط یک کد جلوگیری کند. این عمل خصوصا در مواقعی که از یکسری دستورات تکراری در نیل به یک خروجی مشخص، در تعدادی برنامه استفاده می کند، متوانید بسیار کارآمد باشد. فرم کلی آن بصورت زیر است:

```
function function_Name, input1, input2, Type=type

If keyword_set(Type) then begin
  ;Statements in defining the optional input(Type variable)
Endif
Result= input1 + input2 ;An Statement with input1 & input2
return, Result

end
```

Built in Functions

در این دستور ابتدا با تعریف یک نام برای تابع و معرفی متغیرهای نمادین ورودی به این تابع، یک سری عملیات محاسباتی و دستوری انجام گرفته و خروجی نتیجه می گردد. نتیجه محاسبات با کمک دستور Return و نام مقدار خروجی نمادین، به کاربر ارائه می گردد.
مقادیر ورودی شامل مقادیر اجباری و اختیاری نمادین است. مقادیر اختیاری برخلاف مقادیر اجباری در جلوی نام خود دارای یک مساوی است که آن نام دوباره عینا تکرار شده است. اختیاری بودن بدین معنی ست که کاربر می تواند بدون ورود مقداری برای آن، نتیجه ای برای محاسبات خود دریافت کند و یا با اختصاص مقداری به متغیر اختیاری، روند محاسبات را بشکلی دیگر (طبق دستور تابع) تغییر دهد و ارائه نماید.

توجه کنید که متغیر اختیاری به دو شکل اسلش و نام متغیر و یا نام متغیر عبارت مساوی و یک مقدار اختصاصی ورودی می تواند معرفی شود.
نمادین بودن در اینجا بدین معنی ست که برنامه با استفاده از این تابع و یکسری متغیر و یا مقدار بعنوان ورودی، عینا از آنها در جایگاه های دستور تابع نوشته شده استفاده و نتیجه را محاسبه و ارائه می کند.

Built in Functions

```
Function Varianc, a, Type = type
  Summ = 0
  S = Size(a)
  N = n_elements(S)
  If keyword_set(Type) then begin
    if S[N-2] eq 2 then a = a.todouble()
    ; or a = double(a)
  endif
  for i= 0, S[N-1]-1 do begin
    Summ = Summ + a[i]
  endfor
  Ave = Summ / S[N-1]
  MD = 0
  for i= 0, S[N-1]-1 do begin
    MD = abs((a[i] - Ave)^2) + MD
  endfor
  Var = MD / (S[N-1]-1)
  StDe = sqrt(Var)
  result=[Ave,StDe]
  Return, result
End
```

تابع روبرو متغیر A را از کاربر گرفته و سپس یک وکتور دو درایه ای خطی را به کاربر خروجی می دهد. درایه اول آن مقدار میانگین و درایه دوم آن میزان انحراف معیار را نشان می دهد. متغیر اختیاری Type فرمت متغیر ورودی را به اعشاری تبدیل می کند تا محاسبات برای ورودی های مقدار صحیح اشکالی ایجاد نکند.

این برنامه با دستورات STDEV و Mean هم قابل انجام بود.

Command Statements

IF, For Loops, While Loops, Repeat Loops, Switch & Case

IF ... THEN ... ELSE

The IF...THEN...ELSE statement conditionally executes a statement or block of statements.

Note: Another way to write an ***IF...THEN...ELSE*** statement is with a conditional expression using the ?: operator. For more information, see Working with Conditional Expressions.

Tip: Programs with vector and array expressions run faster than programs with scalars, loops, and IF statements.

IF ... THEN ... ELSE

Syntax:

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

or

IF expression THEN BEGIN

statements...

ENDIF [ELSE BEGIN]

[statements...]

[ENDELSE]

IF ... THEN ... ELSE

Example :

The following example illustrates the use of the IF statement using the ELSE clause. Notice that the IF statement is ended with ENDIF, and the ELSE statement is ended with ENDELSE. Also notice that the IF statement can be used with or without the BEGIN...END block:

```
>> 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
```

IDL Prints:

B <> 3

IF ... THEN ... ELSE

MATLAB	IDL

`if I == J`
`  A(I,J) = 2;`
`elseif abs(I-J) == 1`
`  A(I,J) = -1;`
`else`
`  A(I,J) = 0;`
`end`
	`IF (I EQ J) THEN BEGIN`
	`  A[I,J] = 2`
	`ENDIF ELSE BEGIN`
	`  A[I,J] = -1`
	`ENDELSE`

For Loops

Syntax:

```
FOR  variable = init, limit [, Increment]  DO  statement
```

or

```
FOR  variable = init, limit [, Increment]  DO  BEGIN
        statements…
ENDFOR
```

For Loops

Example :

The following example iterates over the elements of an array, printing the value of each element:

```
>> array = ['one', 'two', 'three', 'Four']

>> n = N_ELEMENTS(array)

>> FOR  I = 0, n-1 DO BEGIN
       PRINT, array[i]
   ENDFOR
```

Vector Addressing

MATLAB	IDL
`for k=1:10, disp(k), end`	`FOR k=1,10 DO PRINT, k`

`for k=1:N`
`  x(k) = k;`
`end`
	`FOR k=1,10 DO BEGIN`
	`  x[k]=k`
	`ENDFOR`

While Loops

Syntax:

```
WHILE condition_expression DO   BEGIN
        statements…
ENDWhile
```

Repeat Loops

Syntax:

```
REPEAT BEGIN
        statements...
ENDREP UNTIL condition_expression
```

توضیح اینکه: تفاوت این دستور با while در ***تقدم و تاخر شرط*** گذاشته شده است، بیدن صورت که در While ابتدا شرط را چک می کند و سپس اگر برقرار باشد وارد حلقه می شود، در حالی که در Repeat ابتدا وارد حلقه شده و یکبار دستوارت داخل حلقه را اجرا و اعمال می کند و سپس شرط را برای اجرای دوم حلقه بررسی می کند و اگر برقرار باشد دوباره آن را اجرا می کند.

Switch & Cases

Syntax:

```
SWITCH Variable_Name OF
        Value1: begin
                statements…
                [Break]
        End
        Value2: begin
                statements…
                [Break]
        End
        …
ENDSWITCH
```

```
CASE Variable_Name OF
        Value1: begin
                statements…
        End
        Value2: begin
                statements…
        End
        …
        [ELSE: statement(s)]
ENDCASE
```

Switch & Cases

توضیح اینکه: تفاوت دستور Switch با دستور Case در این است که در صورت برقراری مقدار در Switch عبارات مربوط به آن قسمت اجرا و اعمال می شود و دوباره مقدار را در قسمت های دیگر هم بررسی می کند و اگر باز هم برقرار باشد عبارات آن قسمت را هم اجرا و اعمال می کند و الی آخر تا انتهای دستور EndSwitch، در حالی که اگر مقداری در دستور Case صدق کند عبارات آن قسمت را اجرا و اعمال می کند و پس از آن ادامه بررسی ها متوقف شده و به دستور EndCase می رود.

بعبارتی دیگر دستور Switch با دستور Case زمانی دارای خروجی یکسان است که در دستور Switch از دستور Break استفاده شود.

The BREAK statement provides a convenient way to immediately exit from a loop (FOR, FOREACH, WHILE, REPEAT), CASE, or SWITCH statement without resorting to GOTO statements.

Switch & Cases

Example :

This example illustrates how the CASE statement, unlike SWITCH, executes only the one statement that matches the case expression:

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

Switch & Cases

Example :

More than one statement can be executed when a match for the case selector expression is encountered. To execute multiple statements, surround the statements with BEGIN...END statements:

```
X= 5
CASE x OF
    1: PRINT, 'one'
    2: PRINT, 'two'
    3: PRINT, 'three'
    4: PRINT, 'four'
    ELSE: BEGIN
        PRINT, 'You entered: ', x
        PRINT, 'Please enter a value between
1 and 4'
        END
ENDCASE
```

Switch & Cases

Example :

Another example shows the CASE statement with the number 1 as the selector expression of the CASE. One is equivalent to true and is matched against each of the conditionals.

```
CASE 1 OF
    (X GT 0) AND (X LE 50): Y = 12 * X + 5
    (X GT 50) AND (X LE 100): Y = 13 * X + 4
    (X LE 200): BEGIN
      Y = 14 * X - 5
      Z = X + Y
    END
ELSE: PRINT, 'X has an illegal value.'
ENDCASE
```

Switch & Cases

MATLAB	IDL

switch lower(METHOD)
  case {'linear','bilinear'},...
  disp('linear')
  case 'cubic', disp('cubic')
  case 'nearest', disp('nearest')
  otherwise, disp('Unknown')
end
	CASE name OF
	  'Linda': PRINT, 'sister'
	
	  'John': PRINT, 'brother'
	  'Harry': PRINT, 'step-brother'
	  ELSE: PRINT, 'Not a sibling.'
	ENDCASE

GoTo & *Lable*

Syntax:

```
 statements…
Lable1:
    statements…
GoTo, lable1
 statements…
```

توجه بخاطر بسپارید که استفاده از دستور GoTo بدلیل ایجاد سردرگمی کاربر در هنگام خواندن و ترجمه یک کد و ایجاد پرش در خطوط اجرایی کد توصیه نمی شود و معمولا برنامه نویسان حرفه ای برای مشخص بودن روند اجرای برنامه نوشته شده خود از این دستور استفاده نمی کنند.

The GOTO statement transfers program control to point specified by a label. The GOTO statement is generally considered to be a poor programming practice that leads to unwieldy programs. Its use should be avoided.

GoTo & Lable

Example :

```
pro GOTO_Lable_Exp

Num = 1D
lable1: begin
   print,string(fix(Num)) + " Hi"
end
Num = Num + 1
if Num LT 11D then begin
  goto,lable1
endif
print,"Bye after 10 hi"

end
```

این مثال پس از ۱۰ بار سلام پیام خداحافظی را چاپ می کند.

GoTo & Lable

Example :

```
pro GOTO_Lable_Exp1

Num = 1D
goto,lable2
lable1: begin
   print,string(fix(Num)) + " Hi"
   return
end
lable2:
Num = Num + 1
if Num LT 11D then begin
  goto,lable1
endif
print,"Bye after 1 hi"

end
```

برخلاف مثال قبلی این دستور با چاپ سلام۲ از شما خداحافظی می کند.

All Command Statements

Statement	END Identifier	Example

ELSE BEGIN	**ENDELSE**	`IF (0) THEN A=1 ELSE BEGIN`
		`  A=2`
		`ENDELSE`

FOR *variable=init*, *limit* DO BEGIN	**ENDFOR**	`FOR i=1,5 DO BEGIN`
		`  PRINT, array[i]`
		`ENDFOR`

FOREACH *element*, *variable* [, *key*] DO BEGIN	**ENDFOREACH**	`arr = [1, 3, 5, 7, 9]`
		`FOREACH element, arr DO BEGIN`
		`  PRINT, element`
		`ENDFOREACH`

IF *expression* THEN BEGIN	**ENDIF**	`IF (0) THEN BEGIN`
		`  A=1`
		`ENDIF`

REPEAT BEGIN	**ENDREP**	`REPEAT BEGIN`
		`  A = A * 2`
		`ENDREP UNTIL A GT B`

WHILE *expression* DO BEGIN	**ENDWHILE**	`WHILE ~ EOF(1) DO BEGIN`
		`  READF, 1, A, B, C`
		`ENDWHILE`

LABEL: BEGIN	**END**	`LABEL1: BEGIN`
		`  PRINT, A`
		`END`

case_expression: BEGIN	**END**	`CASE name OF`
		`  'Moe': BEGIN`
		`  PRINT, 'Stooge'`
		`  END`
		`ENDCASE`

switch_expression: BEGIN	**END**	`SWITCH name OF`
		`  'Moe': BEGIN`
		`  PRINT, 'Stooge'`
		`  END`
		`ENDSWITCH`

Stopping Commands

Exit از این دستور برای بستن تمامی پنجره های IDL و ENVI استفاده می شود.

Return از این دستور برای خروج از زیربرنامه نوشته شده و یا ارائه خروجی یک تابع استفاده می شود.

Retall از این دستور برای خروج از تمامی برنامه ها و زیربرنامه های موجود استفاده می شود.

Break از این دستور برای خروج از حلقه استفاده می شود.

Continue از این دستور برای پرش از حلقه جاری استفاده می شود و وارد حلقه بعدی می شود.

Exit, Retrun, Retall, Break & Continue

The **CONTINUE** statement provides a convenient way to immediately start the next iteration of the enclosing FOR, FOREACH, WHILE, or REPEAT loop.

Example :

This example presents one way (not necessarily the best) to print the even numbers between 1 and 10.

```
FOR I = 1,10 DO BEGIN
   ; If odd, start next iteration:
   IF (I AND 1) THEN CONTINUE
   PRINT, I
ENDFOR
```

Exit, Retrun, Retall, Break & Continue

The **BREAK** statement provides a convenient way to immediately exit from a loop (FOR, FOREACH, WHILE, REPEAT), CASE, or SWITCH statement without resorting to GOTO statements.

Example :

This example exits the enclosing WHILE loop when the value of i hits 5.

```
I = 0
PRINT, 'Initial value: ', i
WHILE (1) DO BEGIN
   i = i + 1
   IF (i eq 5) THEN BREAK
   PRINT, 'Loop value: ', i
ENDWHILE
PRINT, 'END VALUE: ', I
```

Exit, Retrun, Retall, Break & Continue

در حالت عمومی دو دستور **Retall** و **Retrun** تفاوتی ندارند و زمانی تفاوت آنها مشخص می شود که شما از برنامه و زیربرنامه استفاده نمایید. در این صورت زمانی که در یکی از زیربرنامه ها از دستور **Retall** استفاده کنید، کل برنامه خاتمه می یابد و اگر در آن از **Retrun** استفاده کنید، اجرای دستورات موجود در آن زیربرنامه متوقف شده و به برنامه اصلی و ادامه اجرای برنامه اصلی برگشت داده می شود.

The RETALL command returns control to the main program level. The effect is the same as entering the RETURN command at the interactive command prompt until the main level is reached.

The RETURN command causes the program context to revert to the next-higher program level.

Strings

در متون String نمایش همچون تایتل پنجره ها و غیره، میتوان با بهره گیری از دو کاکتر _ و ^ و قرار دادن متن داخل دو علامت دلار (‘$String$’) از حروف یونانی از طریق استفاده از علام بک اسلش (\) بهره جست.

‘$Hello_\Alpha^\Beta$’

حـــــروف یـــــونـــــانـــــی		
آلفا Aα	اپسیلون Eε	یوتا Iι
نو Nv	رو Pρ	فی Φφ
بتا Bβ	زتا Zζ	کاپا Kκ
کسی Ξξ	سیگما Σσς	خی Χχ
گاما Γγ	اتا Hη	لامبدا Λλ
امیکرون Oo	تاو Tτ	پسی Ψψ
دلتا Δδ	تتا Θθ	مو Mμ
پی Ππ	اوپسیلون Yυ	اومگا Ωω

A α	alpha	N ν	nu
B β	beta	Ξ ξ	ksi
Γ γ	gamma	O o	omicron
Δ δ	delta	Π π	pi
E ε	epsilon	P ρ	rho
Z ζ	zeta	Σ σς	sigma
H η	eta	T τ	tau
Θ θ	theta	Y υ	upsilon
I ι	iota	Φ φ	phi
K κ	kappa	X χ	chi
Λ λ	lambda	Ψ ψ	psi
Mμ	mu	Ω ω	omega

Greek alphabet chart © by de Traci Regula; licensed to About.com

Widget Statements

Widgets

این صفحه در دست طراحی است!!!!!

Built in Projects

Built in Projects

این صفحه در دست طراحی است!!!!! 😂

برای اینکار باید اسم پروژه با اسم فایل اصلی یکسان باشد و ضمن اینکه بهتر است قبل از ران Built، با رفتن به Option پروژه تیک مربوط به reset Session before builtting project. Execute را در تب مربوط به IDL Built Properties برداشت.

IDL Programming: Examples

Example No. 1

```
pro assignment1, event

  a=[[1,2,3],[4,5,6],[7,8,0]]
  b=transpose(a)

  ; Call IMSL_INV to perform the inversion.
  ainv=IMSL_INV(a) ; Or you can use:->  ainv=invert(a)
  A_ainv=a##ainv
  SinA_ainv = SIN(A_ainv)

;  method 1
;==========
;  Sum_a=0
;  FOR i= 0, n_elements(a)-1 do begin
;    if i EQ 0 or i eq 4 or i eq 8 then begin
;      Sum_a= Sum_a + SinA_ainv(i)
;    endif
;  ENDFOR
```

۱- با کمک تعریف یک ماتریس ۳*۳ و ضرب ماتریسی آن در معکوس خودش ماتریس جدیدی بدست آورید و سپس از آن Sin گرفته و نهایتا مقدار مجموع درایه های روی قطر اصلی ماتریس حاصل را با ابزار dialog_Message() به کاربر نمایش دهید.

Example No. 1

```
;  method 2
;==========
;Dim=size(a,/DIMENSIONS)
;  Sum_a=0
;  FOR i= 0, Dim(1)-1 do begin
;    FOR j= 0, Dim(0)-1 do begin
;      if i EQ j then begin
;        Sum_a= Sum_a + SinA_ainv(j,i)
;      endif
;    ENDFOR
;  ENDFOR

;  method 3 (The best)
;==========
  eye=identity(3) ; UNIT Matrix 3*3 or eye Matrix in
matlab
  B=SinA_ainv*eye
  Sum_a=total(B)
  Show_Massage= dialog_Message(string(Sum_a))
End
```

۱- با کمک تعریف یک ماتریس ۳*۳ و ضرب ماتریسی آن در معکوس خودش ماتریس جدیدی بدست آورید و سپس از آن Sin گرفته و نهایتا مقدار مجموع درایه های روی قطر اصلی ماتریس حاصل را با ابزار dialog_Message() به کاربر نمایش دهید.

Example No. 2

```
pro assignment2, event

a=make_array(3,5,value=1D)
for i=1,n_elements(a)-1 do begin
  a[i]=(a[i-1]+a[i])
endfor

;method 1
========
j=0.0
k=0.0
for i=0.0,n_elements(a)-1.0 do begin
  if (a[i]/2d - floor(a[i]/2d)) eq 0.0 then begin
    if j eq 0.0 then begin
      pos_even=i
      j=1.0
    endif else begin
```

۲- ابتدا با دستور Make_Array() یک ماتریس واحد ۳*۵ بسازید سپس ماتریس را بگونه ای تغییر دهید که درایه های دوم به بعد را بصورت تجمعی از درایه قبلی محاسبه نماید و نهایتا شماره درایه هایی که دارای مقادیر زوج و یا فرد هستند را با کمک محاسبات ریاضیاتی به کاربر نمایش و ماتریس جدید دیگری با همان ابعاد ایجاد و مقادیر درایه های زوج را به ۰ و مقادیر درایه های فرد را به ۱ تغییر دهد.

Example No. 2

```
      pos_even=[pos_even,i]
    endelse
  endif else begin
    if k eq 0.0 then begin
      pos_odd=i
      k=1.0
    endif else begin
      pos_odd=[pos_odd,i]
    endelse
  endelse
endfor

;method 2
;========
;pos_even=where((a/2d - floor(a/2d)) eq 0.0,wt)
;pos_odd=where((a/2d - floor(a/2d)) ne 0.0,wt)
```

۲- ابتدا با دستور Make_Array() یک ماتریس واحد ۳*۵ بسازید سپس ماتریس را بگونه ای تغییر دهید که درایه های دوم به بعد را بصورت تجمعی از درایه قبلی محاسبه نماید و نهایتا شماره درایه هایی که دارای مقادیر زوج و یا فرد هستند را با کمک محاسبات ریاضیاتی به کاربر نمایش و ماتریس جدید دیگری با همان ابعاد ایجاد و مقادیر درایه های زوج را به ۰ و مقادیر درایه های فرد را به ۱ تغییر دهد.

Example No. 2

```
print,'pos_even= ',pos_even
print,'pos_odd= ',pos_odd
if pos_even[0] ne -1.0 then begin
  a[pos_even]= 0.0
endif
if pos_odd[0] ne -1.0 then begin
  a[pos_odd]= 1.0
endif

print,a
end
```

۲- ابتدا با دستور Make_Array() یک ماتریس واحد ۳*۵ بسازید سپس ماتریس را بگونه ای تغییر دهید که درایه های دوم به بعد را بصورت تجمعی از درایه قبلی محاسبه نماید و نهایتا شماره درایه هایی که دارای مقادیر زوج و یا فرد هستند را با کمک محاسبات ریاضیاتی به کاربر نمایش و ماتریس جدید دیگری با همان ابعاد ایجاد و مقادیر درایه های زوج را به ۰ و مقادیر درایه های فرد را به ۱ تغییر دهد.

Example No. 3

۳- ابتدا یک ماتریس ۲*۳ با مقادیر ثابت ۵ تعریف نموده و سپس آن را Transpose و با یک ماتریس رندوم با توزیع یکنواخت صفر تا یک هم بعد با ماتریس Transpose جمع نمایید. سپس با کمک ابزارهای گرافیکی، سه مقدار عددی متفاوت (A,B,C) را از کاربر گرفته و نهایتا عدد اول (A) را با ستون اول ماتریس ساختگی جمع و عدد دوم (B) را در درایه هایی که دارای جایگاه درایه ای زوج هستند ضرب نموده و سپس مقادیری از ماتریس حاصل را که دارای مقادیری بزرگتر از مقدار عدد سوم (C) هست به کاربر نمایش دهد.

Example No. 3

```
pro assignment3, event
a=make_array(3,2,value=5D)
b=transpose(a)+randomu(seed,2,3)

base = widget_auto_base(title='Data Input')
base1=widget_base(base,/row)
we1 = widget_param(base1,prompt='First value (0.0 to 20.0)= ', dt=5, field=2, floor=0, ceil=20, $
  default=10., uvalue='param1',xsize=10,/inches,/auto)
we2 = widget_param(base1,prompt='second value (0.0 to 1.0)= ', dt=5, field=2, floor=0, ceil=1.0, $
  default=0.5, uvalue='param2',xsize=10,/percent,/auto)
we3 = widget_param(widget_base(base,/row),prompt='thierd value(0.0 to 10.0)= ', dt=5, field=2, floor=0,ceil=10,$
  default=5.0, uvalue='param3',xsize=10,/cm,/auto)

result = auto_wid_mng(base)
if (result.accept eq 0) then Retall ; Return
c1=result.param1
c2=result.param2
c3=result.param3

b[0,*]=b[0,*]+c1
index=0
for i=0,n_elements(b)-1,2 do begin
  b[i]=b[i]*c2
endfor
```

Example No. 3

```
;pos_even=where((b[1,*]/2.0 - floor(b[1,*]/2.0)) eq 0.0,wt1)
;if wt1 gt 0 then begin
;  (b[1,*])[pos_even]=(b[1,*])[pos_even]*c2
;endif
pos=where(b gt c3,wt)
if wt gt 0 then begin
  values=string(b[pos])
  val=' '
  for i=0,values.LENGTH -1 do begin
    val=+values[i] + ' ' + val
  endfor
  res=dialog_message('The values of final_Matrix that greater than ' + $
     strtrim(string(c3.tointeger()),2) + ' are:' + string(10B) + val,title='Output',/INFORMATION)
endif

end
```

Example No. 4

```
function assignment4, matrix,num,str,second=second
matrix=matrix.ToDouble()
if matrix.LENGTH ge 3 then begin
  a1=num * matrix[2]
endif else begin
  a1=num + matrix[0]
endelse
a2= str + string(10B) + strtrim(string((a1^(num + $
   1.0)).tointeger()),2)
if keyword_set(second) then begin
  return,a2
endif else begin
  return,a1
endelse
end
```

۴- یک Functionبنویسید که ابتدا سه ورودی (یک ماتریس، یک عدد و یک عبارت String) گرفته و دو خروجی بدهد. خروجی اول نمایش حاصلضرب درایه سوم ماتریسی که دارای سه درایه ویا بیشتر است در مقدار ورودی دوم یا نمایش حاصلجمع درایه اول یک ماتریسی که دارای ۲ درایه یا کمتر است با ورودی دوم. خروجی دوم هم نمایش عبارت ورودی سوم و مقدار خروجی اول به توان (ورودی دوم بعلاوه یک)

Example No. 5

۵- یک برنامه بنویسید که در آن از تمرین ۴ با مقادیر ورودی (آخرین ماتریس خروجی از تمرین دوم، عدد خروجی از تمرین اول، یک عبارت String دلخواه گرفته شده از کاربر با ابزار گرافیکی) استفاده شود و خروجی تمرین چهارم به کاربر نمایش داده شود.

Final Examples

۶- لطفا فلوچارت الحاقی در پیوست را در محیط IDL کدنویسی و برای اجرای آن یک دکمه به برنامه ENVI اضافه نمایید.

برای خواندن تصاویر لطفا از Function پیوست بصورت زیر استفاده نمایید.

```
A=get_Image_Band('File',0)
```

که در آن A همان ماتریس تصویر است که در برنامه استفاده می شود و File یک استرینگ شامل آدرس دایرکتوری و اسم فایل هست و عدد صفر هم یعنی باند اول تصویر layer Stack شده که شامل چندین باند است، توضیح و توجه اینکه چون خط اول داخل این Function بصورت زیر است

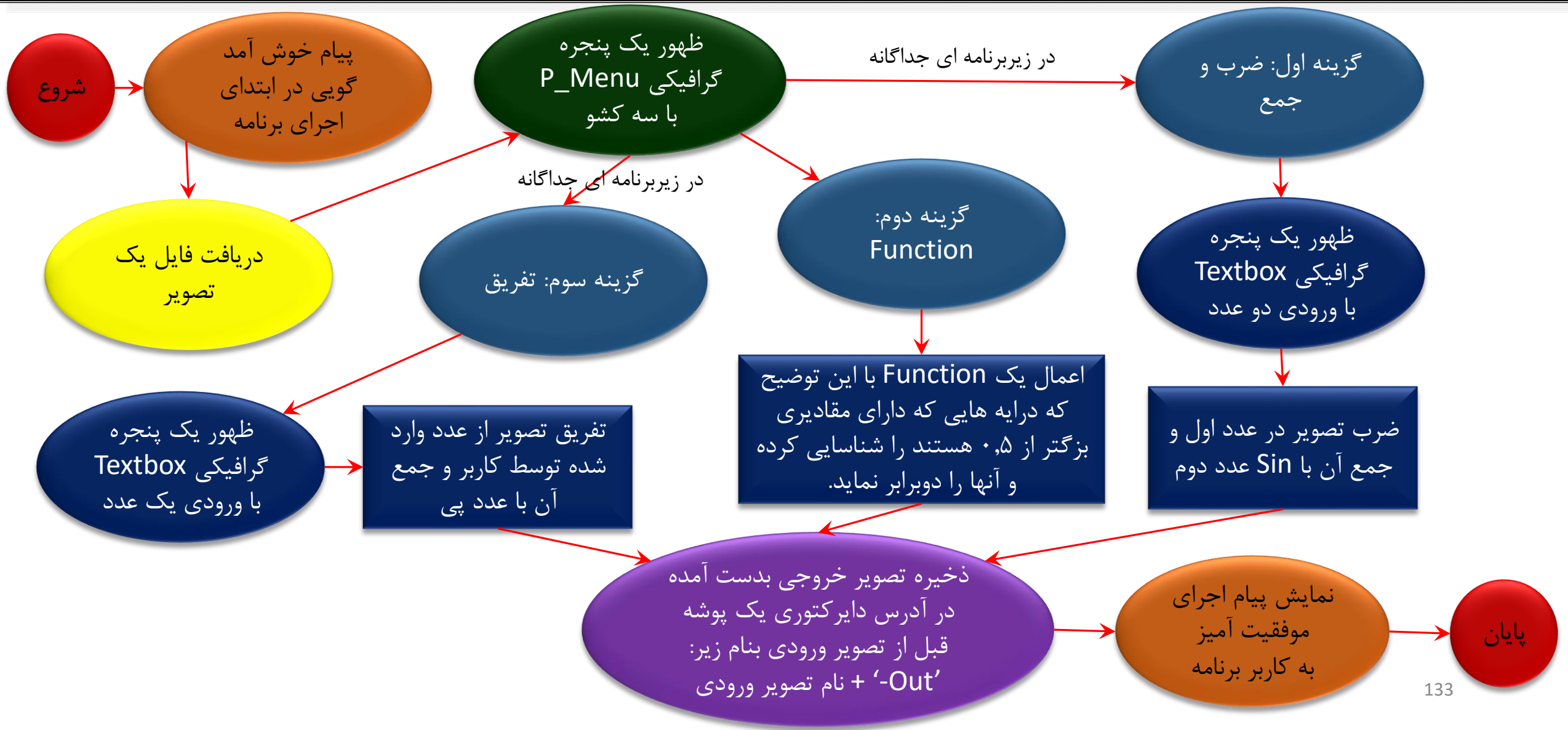
```
common S_DIM
```

حتما باید در برنامه ای که از این فانکشن استفاده می شود، قبل از استفاده باید مقدار DIM مشخص و با دستور زیر از قبل ذخیره شده باشد:

```
common S_DIM,DIM_X,DIM_Y
```

که در آن DIM_Y تعداد ستون ماتریس (تصویر) و DIM_X تعداد سطر ماتریس (تصویر) است.

Final Examples



Final Examples

```
+;
; :Author: HaDi
-;
function get_image_band, filename, band_id

common S_DIM

ENVI_OPEN_FILE, filename, r_fid=fid
ENVI_FILE_QUERY, fid, dims=dims
data = double(ENVI_GET_DATA(fid=fid, dims=dims, pos=band_id))

envi_file_mng, id=fid, /remove
return, data

end
```

Final Examples

۷- برنامه ای بنویسید که مراحل زیر را بترتیب اجرا کند:

- ظاهر شدن پنجره ای گرافیکی که تعداد باند و سپس نوع باندهای یک تصویر را فراخوانی کند.
- اگر تعداد کمتر از دو باند از تصویر لیر استک شده را فراخوانی کرد مقدار (الف) و یا (ب) را بعنوان متغیر D محاسبه نماید.

  الف) ماتریس Dبرابر حاصلضرب باند اول در ماتریس رندوم هم بعد با B0

  ب)ماتریس Dبرابر با حاصلضرب باند اول در transpose باند دوم از تصویر
- درایه هایی از ماتریس نهایی را که دارای مقادیری کمتر از باند اول است را برابر با صفر قرار دهد.
- آدرس یک دایرکتوری را از کاربر پرسیده و با اسم D_File در آنجا ذخیره نماید.

Final Examples

```
pro   opn_image

max_num=10

; NUM_BAND =========
base=widget_auto_base(title='how many bands do you want?     ')
text_box= widget_param(base, prompt='Enter Band Number: ',uvalue='value' , DT=2,floor=1, ceil=max_num,/auto)
result= auto_wid_mng(base)
if result.accept eq 0 then return
num_s=result.value

; wch_BAND =========
base_1= widget_auto_base(title='how many bands do you want?     ')
ulist= 'B_' + strtrim(string(lindgen(num_s)+1),2)
text_box=lindgen(num_s)+1

for i=0,num_s-1 do begin
text_box[i]=widget_param(base_1, prompt='Enter ' + STRtrim(string(i),2) + 'th Band Number: ',uvalue=ulist[i],
DT=2,floor=1, ceil=max_num,/auto )
endfor
  result_1= auto_wid_mng(base_1)
if result_1.accept eq 0 then return
data=make_array(num_s,/INTEGER)
```

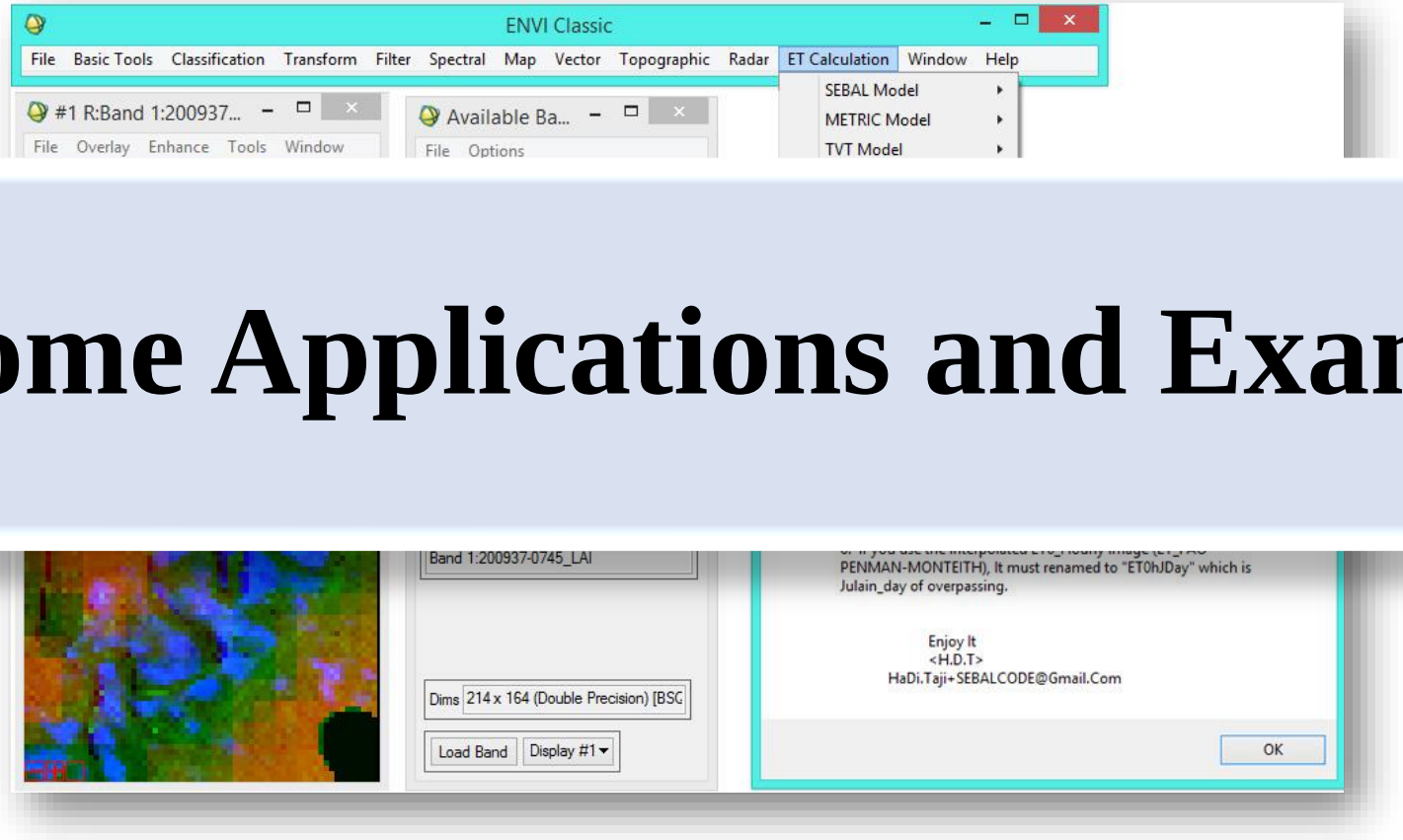
Final Examples

```
file_name= ENVI_pickfile(title='plz entr the img file', default='E:\tasavir', filter='*.dat')
ENVI_open_file,file_name, R_FID=FILE_1,/NO_REALIZE,/ Invisible;/NO_INTERACTIVE_QUERY]
envi_file_query, FILE_1, ns=ns, nl=nl, nb=nb   , dims=dimensions_file1
map_info=envi_get_map_info(fid=FILE_1)
slctd_bnd=make_array(ns,nl,num_s,/DOUBLE)

for i=0,num_s-1   do begin
    data[i]=result_1.(uList[i])
    slctd_bnd[*,*,i]= double(envi_Get_Data(FID=FILE_1, dims=dimensions_file1, pos=data[i]))
endfor

if (size(slctd_bnd))[3] GT 2 then begin
D= slctd_bnd[*,*,0]## transpose(slctd_bnd[*,*,1])
endif else begin
    D= slctd_bnd[*,*,0] + randomu(seed,ns,nl)
endelse

Index=where(D LT slctd_bnd[*,*,0],wt)
if wt GT 0 then D[Index]=0d

base=widget_auto_base(title='Output File Name' )
text_box= widget_string(base, prompt='Enter File Name: ',uvalue='value_1', DEFAULT='D_file',/ auto)
result= auto_wid_mng(base)
if result.accept eq 0 then return
num_s=result.value_1
```

Final Examples

```
des=strpos(file_name, '\',/REVERSE_SEARCH)
out_var= strmid(file_name,0,des+1) + num_s ;'D_file'
envi_write_envi_file,D,MAP_INFO=map_info,OUT_NAME=out_var
end
```

Chapter Three



Some Applications and Examples

Some Applications

Evapotranspiration Models

S E B A L

Surface Energy Balance Algorithms for Land

Some Applications

Dust Models

Some Applications

Landuse Extractions