

Scripting Digital Micrograph™

Dave Mitchell

www.dmscripting.com

adminnospam@dmscripting.com (remove the nospam)

Workshop Aims

- To provide users with a basic understanding of :
 - What scripts are.
 - How they work.
 - What scripts can do.
 - How to write scripts.
 - What scripting resources are available.
- Enlarge the pool of scripting talent.

Why Script?

- No software does everything you want.
- Customisable software enables:
 - Simplified and efficient experimental work flows.
 - Everything to be done within DM.
 - Creation of extended and new EM capabilities.
 - Use DM for 'other things' eg optical microscopy.
 - Enables sharing of experimental methods via the common DM platform.

Anatomy of a Script

- Filename - 'Script Name.s'
- Scripts (extension= .s).
- Simple text files.
- Can be created in DM or any text editor.
- White space is ignored and no line numbers.
- Commands are not case sensitive.
- Usually a single command per line followed by a carriage return.
- Scripts are interpreted not compiled - SLOW!!!!
- Whole image operations are supported - FAST!!!!

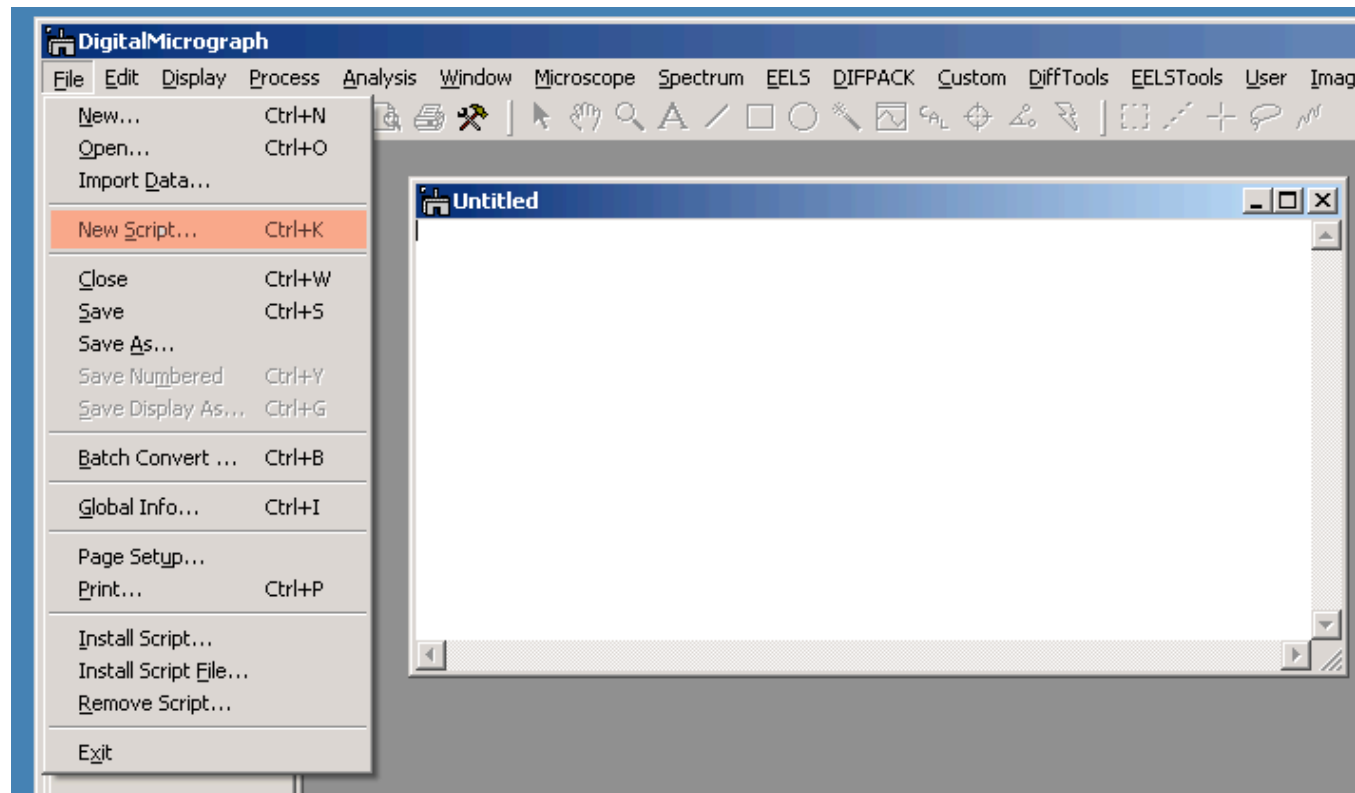


1. Script Basics

- Learning objectives:
 - How to create a script.
 - How to edit a script.
 - How to save a script.
 - How to execute a script.
 - How to halt a script.
 - How to install a script.

1. Script Creation

- From the *File* Menu select *New Script* or enter **CONTROL + K**



Exercise 1a. Editing, Saving & Execution

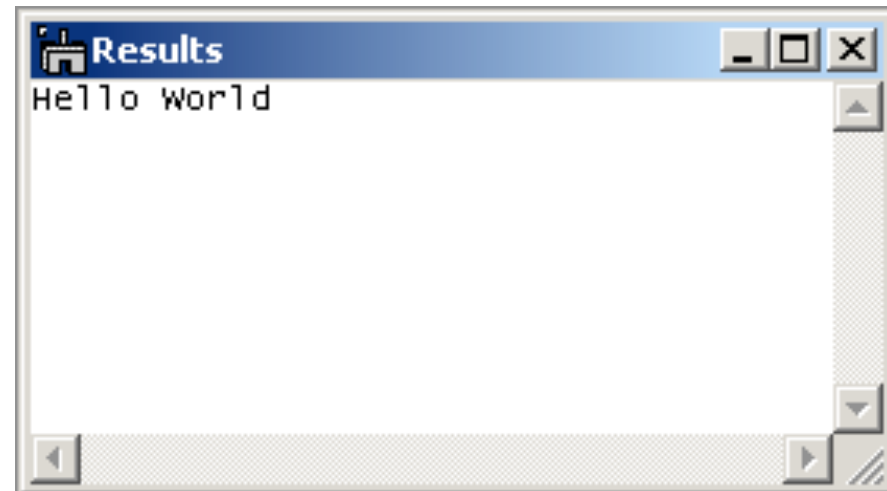
- Simply type in the commands into the script window.
- Save the script using **File/Save As** with the option Script Files(*.s).
- Execute the script by having it foremost and hitting **CONTROL + ENTER**
- Run script 1a.

Commands used in Script 1a:

// (Comments)

void Result(String)

(outputs to Results Window)



1. Playing with the Script

- Run the script 3 times.
- Edit the 'result' command in the script to read each of the following then run it:

```
result("Hello World\n")
```

```
result      ("Hello World\n")
```

```
RESULT("Hello World\n")
```

```
result("Hello\t World\n")
```

```
result(Hello World\n)
```

- Note \n and \t add carriage returns and tabs to strings respectively - use these for formatting your outputs.

1. Halting a Script

- Scripts can get into infinite loops.
- Careful coding can help trap for these.
- If an infinite loop occurs halt the script with **CONTROL + NUM LOCK** (on the numeric keypad - it may vary with keyboard mapping).

1. Installing a script

- Have the script foremost in DigitalMicrograph (title bar is blue).
- From the 'File' menu select 'Install Script'.
- The default installation menu is the 'Custom' menu and you can create a sub-directory within that menu. Alternatively, create a new menu of your own, also with sub-directories if required.
- Select the name for the menu function - it takes the name of the script by default, but this may be longer than ideal - truncate if necessary.
- Click install - the script becomes available in the chosen menu.
- Installed scripts stored inside the 'Preferences.dm8' file, inside the /Programs/Gatan/DigitalMicrograph/Preferences folder.
- Script installations can be copied between installations of DigitalMicrograph by simply copying the .dm8 preference file.
- Scripts defining functions (see later) can be installed as Library files which other scripts can call.

1. Review

- Learning objectives:
 - How to create a script.
 - How to edit a script.
 - How to save a script.
 - How to execute a script.
 - How to halt a script.
 - How to install a script.

2. Understanding and using Variables

- Learning Objectives:
 - What the different types of variables are.
 - How to declare variables.
 - How to carry out simple operations with variables.
 - Some special image variables and sub-area operations.

2. What are Variables?

- Variables are containers which hold information.
- Information may be a number, a string of text, an image etc.
- In DM scripting variables must be declared so that the computer knows what the variable is called, and what type of information it will hold, and possibly what its value is.
- Declaring a variable will initialise it (sets its value to zero).
- Variables remain in scope until the script ends.
- Variables have a name, a type and a value.
- Variables can be Global or Local in scope (more later).

2. Variable Names

- Variables can be called virtually anything eg a, z, potato, alphabet123
- They can not be called, or start with, a number eg variable names of 1, 123alphabet are illegal, and spaces are not allowed.
- It is good practice to use intuitive variable names ie use 'temperature' instead of 't'.
- Capitalisation of longer variable names helps with readability eg use 'CoreTemperature' rather than 'coretemperature'.
- Note 'CoreTemperature' will be treated the same as 'coretemperature' - there is no case sensitivity.

2. Variable Types

- The main types of variable are:
- number - shorthand for realnumber:
 - Number variables hold numerical values eg
`CoreTemperature=542.723, BoilingPoint=100`
 - Number variables can store integer or real values.
- string:
 - String variables hold text values eg
`MyName="Dave"`
`MyAge="21"`
 - Note the variable MyAge is a string not a number - so the expression
`number Realage=2 x MyAge` will not work.
- image:
 - Image variables hold images - of which there are several types.

2. Declaring Variables

- All variables used must be declared.
- Declaration of variables is usually at the start of the script.
- Variables declared in this way are said to be Global - ie any part of the script has access to and can change a Global variable.
- For simple scripts making all your variables Global is fine.
- Complex scripts which use functions, class methods etc should use predominantly local variables (more later).

Exercise 2a. Declaring Variables (number and string)

- Run script 2a to see how the following variables appear.

number temperature, time, gravity

number acceleration=9.8

number ZeroPoint=273.15, delay=10

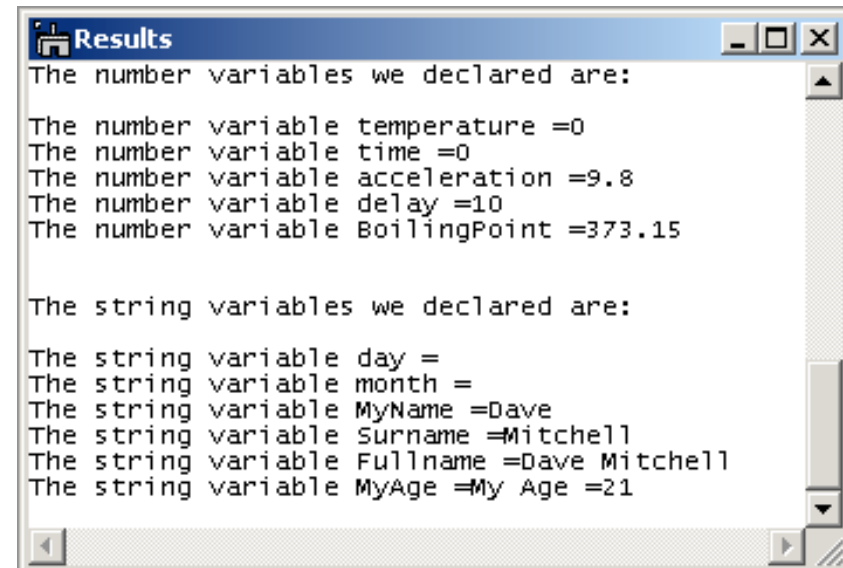
number BoilingPoint=ZeroPoint+100

string day, month

string MyName="Dave", Surname="Mitchell"

string FullName=Myname+" "+Surname

string MyAge="My Age ="+"21



```
Results
The number variables we declared are:
The number variable temperature =0
The number variable time =0
The number variable acceleration =9.8
The number variable delay =10
The number variable BoilingPoint =373.15

The string variables we declared are:
The string variable day =
The string variable month =
The string variable MyName =Dave
The string variable Surname =Mitchell
The string variable FullName =Dave Mitchell
The string variable MyAge =My Age =21
```

2. Test your Variable Skills

- Write a short script in which you declare variables to represent:

- Day
- Month
- Year
- Date

Useful Script Commands:

number NumberVariable (declares a number)

string StringVariable (declares a string)

Result(string) (outputs string to Results Window)

// (Comments)

/n (in strings adds a carriage return to a string)

/t (in strings adds a tab to a string)

- Then outputs these to the Results Window in the form :
‘Today is : Sunday, 10 February, 2008’
- NB the answer to this is in the Answer Scripts folder as:
‘2 Answer to test your variable skills’

2. Image Types

- Before discussing declaration of image variables it is important to understand the types of images which DM supports:

Kind	Size	Signed	Range	Smallest Value
Binary	1		0 or 1	1
Integer	1	Signed	-128 to 127	1
	1	Unsigned	0 to 255	1
	2	Signed	-32,768 to 32,767	1
	2	Unsigned	0 to 65,535	1
	4	Signed	-2,147,483,648 to 2,147,483,647	1
	4	Unsigned	0 to 4,294,967,295	1
Real	4		$\pm 3.4E+38$	$1.5E-45$
	8		$\pm 1.7E+308$	$5.0E-324$
Complex	8		$\pm 3.4E+38$	$1.5E-45$
	16		$\pm 1.7E+308$	$5.0E-324$
RGB	4		R/G/B/A 0 to 255	1

2. Declaring Image Variables

- Image variables are declared like any other variable:
`image FrontImg, AveragelImage`
- The foremost image in DM can be referenced as follows:
`image FrontImg:=getfrontimage()`
- New (empty) integer images can be created as follows:
`image NewImg:=integerimage("",bytes,sign, x,y)`
Where "" is a null string, bytes = no. of bytes in the image (1, 2, or 4),
sign = (0=unsigned, 1=signed), and x & y = pixel dimensions.
- New (empty) real images can be created as follows:
`image NewImg:=realimage("",bytes, x,y)`
There is no sign value to a real image, as by definition it can handle negative numbers, bytes can be 4 or 8.
- Maths on image variables is similar to any other variable:
`image FlatImg:=getfrontimage()`
`image RootImg=sqrt(FlatImg)`
`image SquaredImg=FlatImg*FlatImg`

2. Explicit vs Implicit Image Creation

- The previous example of creating an image by calling an image creation function is referred to as Explicit Creation eg
`image myimg:=binaryimage("",xsize, ysize)`
- Explicitly creates an image called myimg, which is a binary image of dimensions xsize and ysize.
- Fortunately DM also creates images implicitly by context. This avoids the need to define image types and sizes every time a new image is created.
- Implicit creation will automatically create an image of the appropriate type and size, based on the context of the image operation eg
`image newimg= oldimg+5`
- Creates an image variable called newimg, which is the same size as oldimg and is of type real.
- There are three default image types for implicit creation:
- Real, Complex and RGB (binary & integer operations default to real)

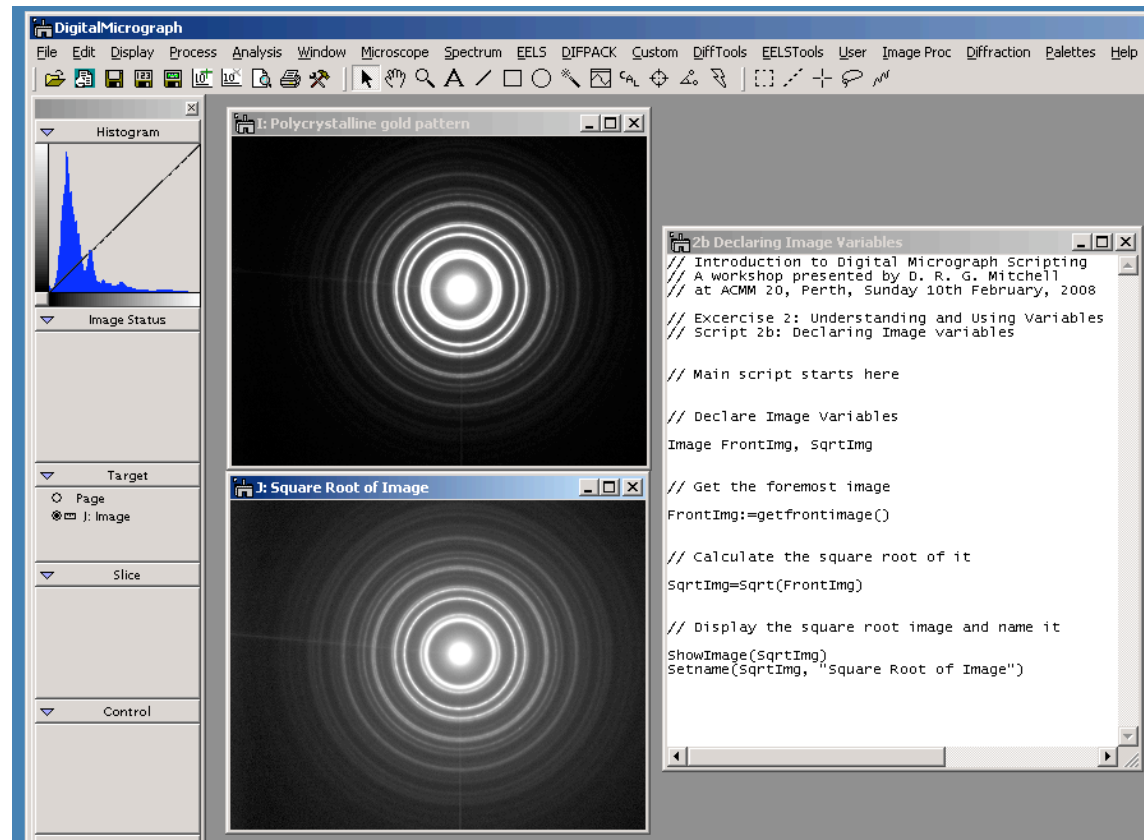
Exercise 2b. Declaring Image Variables

- Open the image 'Polycrystalline gold pattern'.
- Open script 2b and run it.

Declare some image variables
Carry out simple image maths

```
image FrontImg, SqrtImg  
FrontImg:=getfrontimage()  
SqrtImg=sqrt(FrontImg)
```

```
showimage(SqrtImg)  
setname(SqrtImg, "Square Root of Image")
```



2. Test your Image Variable Skills

Write a script to:

Calculate the min, max & mean values in the original gold pattern.

Display these values in the Results Window.

Useful Script Commands:

`image ImageVar` (declares an image variable called ImageVar)

`ImageVar:=getfrontimage()` (gets foremost image as imagevar)

`minmax(ImageVar, min, max)` (gets the min/max (numbers) of an image)

`number meanval=mean(ImageVar)` (gets the mean of an image)

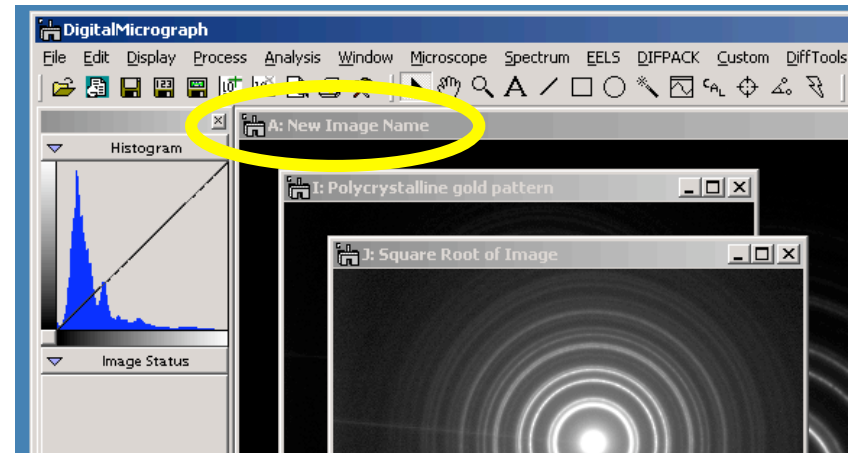
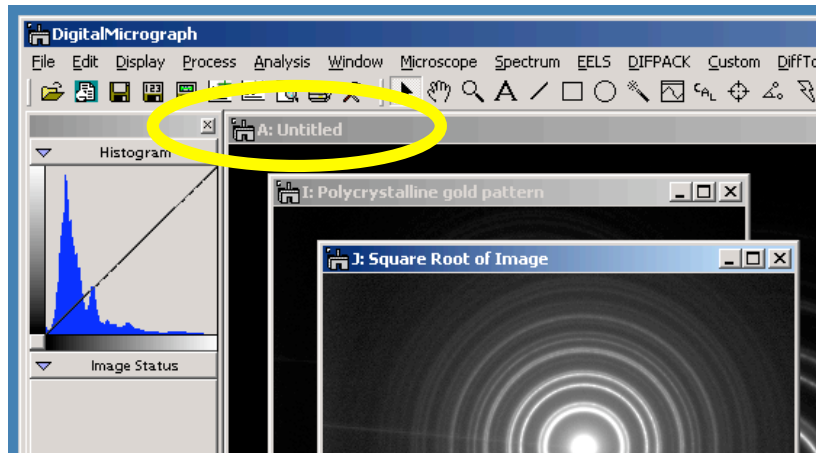
- NB the answer to this is in the Answer Scripts folder as:
‘2 Answer to test your image variable skills’

2. Special Image Variables : ID

- Every image in DM has an identifying reference letter .
- These can be used in scripts to refer to images.
- For example the script:

Setname(a, "New Image Name")

- Changes the name of image a (not case sensitive) from 'untitled' to 'New Image Name'.
- Scripts can use ID variables without declaring them or 'getting' them - this is very handy for 'on-the-fly' scripting to manipulate images directly.



Exercise 2c. Special Image Variables : Intrinsic

- Intrinsic variables are 'intrinsic' to images.
- icol - the position of a column in an image.
- irow - the position of a row in an image.
- iradius - the distance from the geometric centre of an image.
- Others: iheight, ipoints, itheta, iwidth, iplane

Note image origin in
DM is top left

- Open and run script 2c
- Relate the resulting images with
the code which creates them

Image

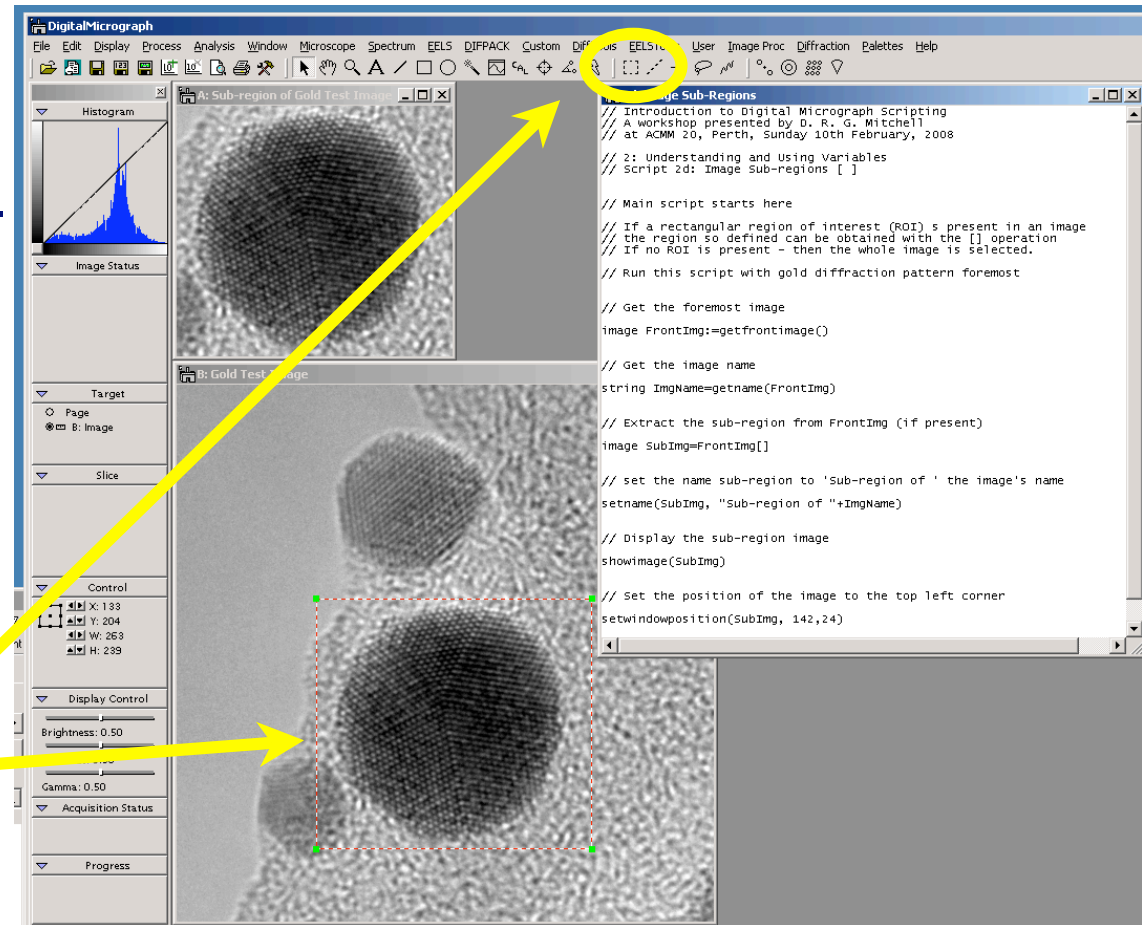
irow=0					
1					
2					
3					
	icol=0	1	2	3	4

2. Image Sub-regions []

- Sub-regions of images can be defined with [].
- The script :
 image FrontImg:=getfrontimage()
 image SubImg=FrontImg[]
 - Assigns a sub-region in FrontImg (defined by a rectangular region of interest (ROI)) to SubImg. If no ROI is present, SubImg is set to FrontImg
- The script :
 Image FrontImg:=getfrontimage()
 Image SubImg=FrontImg[top, left, bottom, right]
 - Assigns a sub-region of FrontImg to SubImg. The region is defined by four numbers : top, left, bottom right - in pixels.
 - The sub-region must lie within the bounds of the image.
 - The origin in images is the top left corner.

Exercise 2d Image Sub-regions []

- Open 'Gold Test Image'.
- Open and run script 2d.
- Get 'Gold Test Image' foremost.
- Add an ROI to the image.
- Run script 2d.



2. Review

- Learning Objectives:
 - What the different types of variables are.
 - How to declare variables.
 - How to carry out simple operations with variables.
 - Some special image variables and sub-area operations.

3. Operators

- Learning Objectives:
 - Understand the available operators.
 - Use operators in scripts.
 - Understand the difference between assignment (=) and assignment by reference(:=).

3. Operators

- Arithmetic:
 - + addition : `expr+expr`
 - - subtraction : `expr-expr`
 - * multiplication : `expr*expr`
 - / division : `expr/expr`
 - ** exponentiation : `expr**expr`
- Equality (return `true=1` or `false=0`):
 - == Equality `expr==expr` (works on strings too)
 - != Inequality `expr!=expr` (works on strings too)
- Relational (return `True=1` or `False=0`):
 - < less than, <= less than or equal : `expr<=expr`
 - > greater than, >= greater than or equal : `expr>-expr`
- Logical (return `True=1` or `False=0`):
 - ! Logical NOT : `!expr`
 - || Logical OR : `expr || expr`
 - && Logical AND : `expr && expr`

Exercise 3a. Assignment vs Reference

`image test=getfrontimage()` (= works on all variable types).

- Means create a new image called test and make its contents equal (=) to those of the frontmost image.
- The new image (test) has no title, no scale bar and is of type real, regardless of the type of whether the original image is binary, integer or real (implicit creation)

`image test:=getfrontimage()` (:=works on images only).

- Means create a new image variable (test) and assign it by reference (:=) so that it becomes a pointer to the frontmost image.
 - The 'new' image is the frontmost image (no new window) and all the original image details are present (name, scale bar etc).
- Open the image 'Gold Test Image'.
 - Run script 3a - then change the = to := and rerun the script.

3. Review

- Learning Objectives:
 - Understand the available operators.
 - Use of operators in scripts.
 - Understand the difference between assignment (=) and assignment by reference(:=).

4. If Statements

- Learning Objectives:
 - Understand the use of various forms of If() statements.
 - Understand the use of !, != and == operators in if() statements.

Exercise 4a. If() Statements with various operators

- If(expr) statements evaluate an expression (expr).
- They have the following forms:
 - If(expr is true) execute this command eg.
`if(3>2) okdialog("Yes")`
will display "Yes" on the screen because 3 is greater than 2.
 - If(!true) execute this command (! Is the logical NOT).
`if(!(2>3)) okdialog("Yes")`
will display "Yes" because 2>3 is Not True.
`if(!(3>2)) okdialog("Yes")`
will display nothing because 3>2 is True.
- Run script 4a - work through the list of if() statements and predict whether they will display 'Yes' on the screen. Remove the comments (//) from the statement and run the script - were you correct?
- Replace the comments and move on to the next if() statement.

4. If() continued

- Other forms of If() statements.

- If(expr)

```
{  
    if expr is true execute all the code between these { }  
}
```

If(expr)

```
{  
    if expr is true execute all the code between these { }  
}
```

Else

```
{  
    otherwise execute all the code between these { }  
}
```

4. Test Your if() Skills

- Write a script which :
 - Creates two number variables : First, Second
 - Assigns values to the two variables.
 - Evaluates these variables reporting on three possible conditions:
 1. First has a value greater than Second.
 2. First has the same value as Second.
 3. First has a value less than Second.
 - Reports on the condition by showing the status on screen.
 - Test your script with values of First and Second of:
 - 5 and 10; 10 and 10; and 15 and 10 respectively.

Useful Script Commands:

`okdialog(string prompt)`

`If(expr) command` (If expr is true then command is executed)

- NB the answer to this is in the Answer Scripts folder as:
‘4 Answer to test your if skills’

4. Review

- Learning Objectives
 - Understand the use of various forms of If() statements
 - Understand the use of !, != and == operators in if() statements

5. Inputs and Outputs

- Learning Objectives:
 - How to source strings/numbers/images.
 - How to display answers.
 - How to save an image.

Exercises 5a,b Getting Strings & Numbers and Displaying Answers.

- Prompt for strings with the `getString()` command:
 `number getString(string prompt, string default, string value)`
 - number takes values of 1 (True) if OK is pressed, 0(False) if Cancel is pressed in the `getString()` dialog.
- Prompt for numbers with the `getnumber()` command:
 `number getnumber(string prompt, number default, number value)`
- Display Answers with:
 `showalert(string, number alerttype)` (alerttype has values 0-3)
 `okdialog(String prompt)`
- Run script 5a and identify the problems with the data input method.
- Run script 5b - see how the data input problems have been solved.

5. Sourcing Displayed Images

- The foremost image is sourced with:
`Image front:=getfrontimage()`
- Select a single image with a prompt:
`getoneimagewithprompt(string prompt, string "", image select1)`
 - There are 3 variants of this command :
`gettwoimages . . ; getthreeimages. . ; getfourimages . .`
With the appropriate number of image variables added :
`. . . image select2; . . image select3; . . image select4).`
 - These functions return 1 when OK is pressed and 0 when Cancel is pressed.

5. Opening and Saving Images

- Image files can be opened by putting up a dialog to identify the file, then opening the file as follows:

```
string path
if(!opendialog(path)) exit(0)
image HDImage:=openimage(path)
showimage(HDImage)
```

- Note use := openimage() rather than =openimage() unless you want to make a copy of the image file.
- The foremost image can be saved in Gatan format as follows:

```
image front:=getfrontimage()
string prompt, default, savepath
if(!saveasdialog(prompt, default, savepath)) exit(0)
saveasgatan(front, savepath)
```

- There are related commands for other image formats such as
saveastiff(front, savepath)

5. Image Manipulation and Saving

- Write a script to:
 - Get the gold diffraction pattern out of several images displayed on screen.
 - Prompt the user for a number (divisor value) (range 0.1-5).
 - Select the top left quarter of the image.
 - Divide this region by the divisor value supplied and display the image
 - Save the resulting image to the desktop.

Useful Script Commands:

`getoneimagewithprompt(prompt, default, imgvar)` (Select a displayed image)
`getnumber(prompt, defaultno, enteredno)` (Prompt for a number)
`img[top, left, bottom, right]= expr` (set the sub area of img to expr)
`showimage(imgvar)` (shows imgvar - brings it to front)
`showalert(alert, type)` (shows a string as an alert string of type 0-3)
`If(!saveasdialog(prompt, "", path)) exit(0)` (get a save path - exits on Cancel)
`saveasgatan(image, path)` (saves image to path)
`getsize(image, xsize, ysize)` (gets the size of image in x and y)

- NB the answer to this is in the Answer Scripts folder as:
‘5 Answer to image manipulation and save

5. Review

- Learning Objectives:
 - How to source strings/numbers/images.
 - How to display answers.
 - How to save an image.

6. Tert(), Loops and More Variables

- Learning Objectives:
 - Understand the power of tert().
 - Implementation of loops : for() and while().
 - Global vs Local variables.

6. My Favorite Command : Tert()

- Tert () is an extremely useful command for manipulating images. It allows the equivalent of an if(expr) condition to be applied to each pixel in an image - in parallel ie it is fast.
- It has the general form:
number Tert(expr, number trueval, number falseval)
- It can take numbers or images as its arguments and return.
- Example:
`imgvar=tert(imgvar<0, 0, imgvar)`
- If the levels in imgvar are <0 (True) set those values to 0. If this condition is False, then the values are left unchanged - this is really useful for stripping out negative numbers - such as occur in EFTEMS.
- Write a one line script using the image ID variable with tert() to remove negative numbers from the gold diffraction pattern.

Exercises 6a,b. Using Tert()

- Tert() statements are not limited to changing the image being evaluated.
- Thresholding/Measurement with tert() can be done using binary images to store grey scale levels.
- This script excerpt shows how to measure the number of pixels in an image (front) between the grey scale levels 80 and 120 inclusive. The regions where this condition is True are set to True(1) in a binary image (binimg).

```
image binimg:=binaryimage("", xsize, ysize)
```

```
binimg=tert(front>=80 && front<=120, 1, 0)
```

- Open the 'Gold Test Image' and experiment with different threshold values in script 6a.
- Open the 'Gold Diffraction Pattern' and experiment with script 6b.

6. For() and While() Loops

- Increment or decrement processes can be achieved with for() loops.

Number i

For(i=0; i<var; i++)

{

Code here is repeatedly executed while i is less than var - break terminates

}

Once i is no longer less than i script execution continues here

- i++ is a shorthand meaning $i=i+1$, i-- means $i=i-1$
- break will cause the loop to terminate and the script to continue
- while(expr)
 - {
 - creates a loop which continues while ever expr is True - break terminates
 - }
- Write a script using for() which writes the numbers 0-10 incl., and the sum of their squares (0^2), (0^2+1^2), ($0^2+1^2+2^2$) etc to the Results window.
- Edit the script so that it terminates if the sum of the squares >99.
- NB Answer is in Answer Scripts folder : '6 Answer to for and while loops'

Exercises 6c,d. Global vs Local Variables

- In DM variables can be Global or Local:
 - Global variables can be accessed and changed (be within scope) from anywhere in the script.
 - Local variables can only be accessed and changed (be within scope) from within the section of code they are declared in.
 - It is permissible to have a global and local variable of the same name.
- Controlling access to Global variables is easy in short scripts.
- In complex scripts using Global variables extensively can create runtime errors which are difficult to trace. Local variables help.
- Variables are made local when declared within a section of code bounded by { } ie within if(), while(), for() statements.
- Functions (see later) contain only local variables and their extensive use can avoid the issues described here.
- Run scripts 6c and 6d.

6. Review

- Learning Objectives:
 - Understand the power of tert().
 - Implementation of loops : for() and while().
 - Global vs Local variables.

7. Functions

- Learning Objectives:
 - Understand the advantages of using functions.
 - Function return types and arguments.
 - Getting more than one return from a function using arguments passed in by reference.

7. Functions

- Writing scripts as sequential instructions is fine for short scripts of perhaps a dozen lines or less.
- It is a very poor way of writing longer scripts because:
 - Logical flow is difficult to follow.
 - Debugging is difficult.
 - Testing is laborious - dependency on other (buggy) code.
 - Sections of code may need to be duplicated - code bloat.
 - Maintaining the code is challenging.
 - Re-using code fragments in other scripts is nearly impossible.
 - Management of variables is a difficult.
- All these problems can be resolved by writing code in the form of functions.
- When writing longer scripts you **MUST** use functions!!!!

7. Advantages of Functions

- Functions are standalone pieces of code either embedded in a script or saved as a library.
- A function is a black box which typically requires a very limited range of inputs (arguments) and produce a very limited range of outputs (returns).
- Once written, what happens inside a function becomes unimportant to the script. The only thing which matters are the inputs and the outputs.
- Variables in a function are local and are unaffected by the rest of the script.
- Functions can simply be copied from script to script.
- Functions are called from the main script, leading to very simple scripts.
- Functions can be developed in isolation to the main script simplifying debugging.

7. The Forms of Functions

- The simplest function - no arguments, no returns.
- All functions must be declared with a return type: void, image, string, number etc.
- void is the return type when the function returns nothing.
- There should be no external dependencies eg on Global Variables.

```
void MyFunction()  
{  
    Function code goes here  
}
```

```
// Main script  
Myfunction()
```

Exercise 7a. A Simple Function

```
void SayGDay()  
{  
    okdialog("G'Day Scripters")  
    return  
}
```

// Main program

SayGDay()

- Important: The function must be declared before it is called, so functions are usually placed at the start of a script.
- At declaration functions must specify a return type. If they return nothing, the type is void.
- Open script 7a - run it.
- Remove the function call (SayGDay()) from the script then install it as a library.
- Write a one line script to access it.

Exercise 7b. Returns from Functions

- Functions can return a single variable only - which can be of any type - string, number, image etc.

```
number CalculateSquare(number EnteredValue)
{
    number SquaredValue=EnteredValue*EnteredValue
    return SquaredValue
}
```

```
// Main program
number Anumber=5, SquaredValue
SquaredValue=CalculateSquare(Anumber)
```

- Run script 7b

Exercise 7c. Multi-argument Functions

- Functions can take any number of arguments of any type.
- Too many arguments will make the function cumbersome to use.
- Try and keep functions simple - write 2 simple functions rather than one complex one.

```
number CalculateArea(number length, number breadth)
{
    number area=length * breadth)
    return area
}
```

// Main program

```
number mylength=5, mybreadth=10, myarea
myarea=CalculateArea(mylength,mybreadth)
```

- Run script 7c

Exercise 7d. Functions - returning more than one value

- Only one variable can be returned from a function.
- This limitation can be overcome by passing in arguments to a function by reference with the use of &.
- Here the function changes the value of the reference arguments.

```
void RootSqr(number myvalue, number &squareval, number &rootval)
{
    squareval=myvalue*myvalue
    rootval=sqrt(myvalue)
    return
}

// main program
number testval=25, testsqrd, testroot
RootSqr(testval, testsqrd, testroot)
okdialog("My Number="+testval+" Square="+testsqrd+" Root="+testroot)
```

- Run script 7d.

7. The Ideal DM Script

- The vast majority of the script functionality is declared in functions.
- The main script is predominantly a handful of function calls.

// Functions begin here

```
type Function1()
```

```
{  
}
```

```
type Function2()
```

```
{  
}
```

etc

// Main program

```
Function1()
```

```
Function2()
```

etc

- NB type is void, image, number, string etc.

7 Review

- Learning Objectives:
 - Understand the advantages of using functions.
 - Function return types and arguments.
 - Getting more than one return from a function using arguments passed in by reference.

8. Debugging and Good Scripting Practice

- Learning Objectives:
 - Understand the limitations of the DM software development environment.
 - Good habits for creating robust and maintainable code.
 - Debugging techniques.

8. Debugging and Good Scripting Practice

- Learning Objectives:
 - Understand the limitations of the DM software development environment.
 - Good habits for creating robust and maintainable code.
 - Debugging techniques.

8. Good Scripting Habits

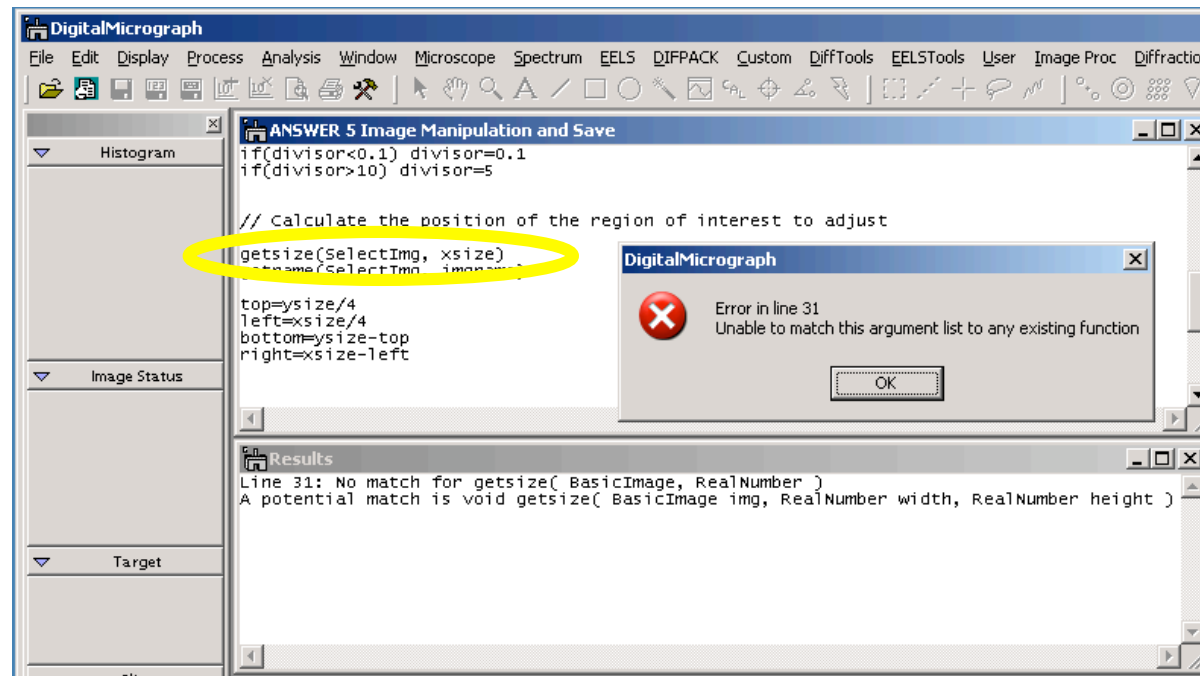
- Add a title, version, date and detailed description at the start of your script.
- Annotate scripts with comments and put spaces before and after.
- Use simple functions wherever possible with no external dependencies.
- Limit the use of global variables.
- Use intuitive variable names.
- Indent loops and if statements especially where nested.

```
for(i=0; i<20; i++)  
    {  
        ExecuteMyFunction(i)  
    }
```

- Break up scripts into blocks of 4 lines with spaces between.
- Save scripts under different version names (080210 1a script name) on completion of each development stage - every 10mins or so.

8. Debugging - Interpretation Errors

- Script errors are reported in a dialog and potential matches are suggested in the Results Window.
- The insertion point moves to where the error occurred.
- The offending command is highlighted .



Tip: Execute commands as single line scripts with no arguments or returns (eg `getsize()`) to:

- a) get the syntax;
- b) see if command exists.

8. Debugging - Run Time Errors

- Track variable values by inserting `okdialog()`, `result()` or `showimage()` commands which output them.
- Stop the script by inserting `exit(0)` at appropriate points.
- Isolate single commands by commenting them out with `//`.
- Isolate blocks of commands by enclosing them between
`/*`
`Commands here are ignored`
`*/`
- Major sources of error : variables not declared or misspelled, failure to close loops correctly, confusing `(= , ==)` and `(= , :=)`, confusing local and global variables, wrong command syntax.

Exercise 8a,b Find the Bugs

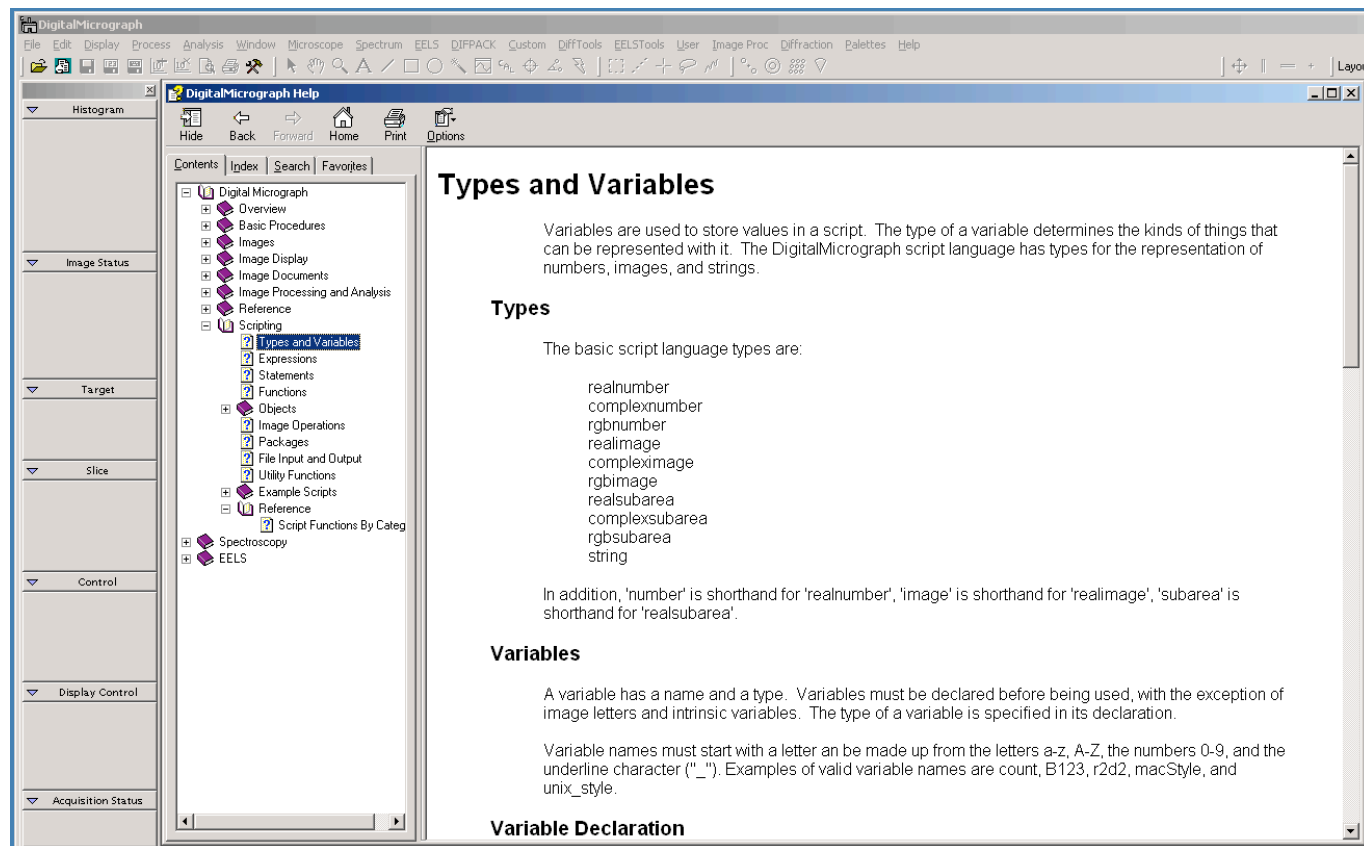
- Open script 8a. This script will not run, it contains a number of interpretation errors.
 - The script simply carries out some maths on the supplied constants
 - Debug the script and get it to run.
 - It should put up a dialog with the answer.
-
- Open script 8b. This script will not run. It contains a number of interpretation errors as well as runtime errors.
 - The script should print the 0 - 5 multiplication table to the Results Window.
 - Can you find the bugs?
 - NB Debugged scripts are in Answers folder as '8a Debugged' etc.

8. Review

- Learning Objectives:
 - Understand the limitations of the DM software development environment.
 - Good habits for creating robust and maintainable code.
 - Debugging techniques.

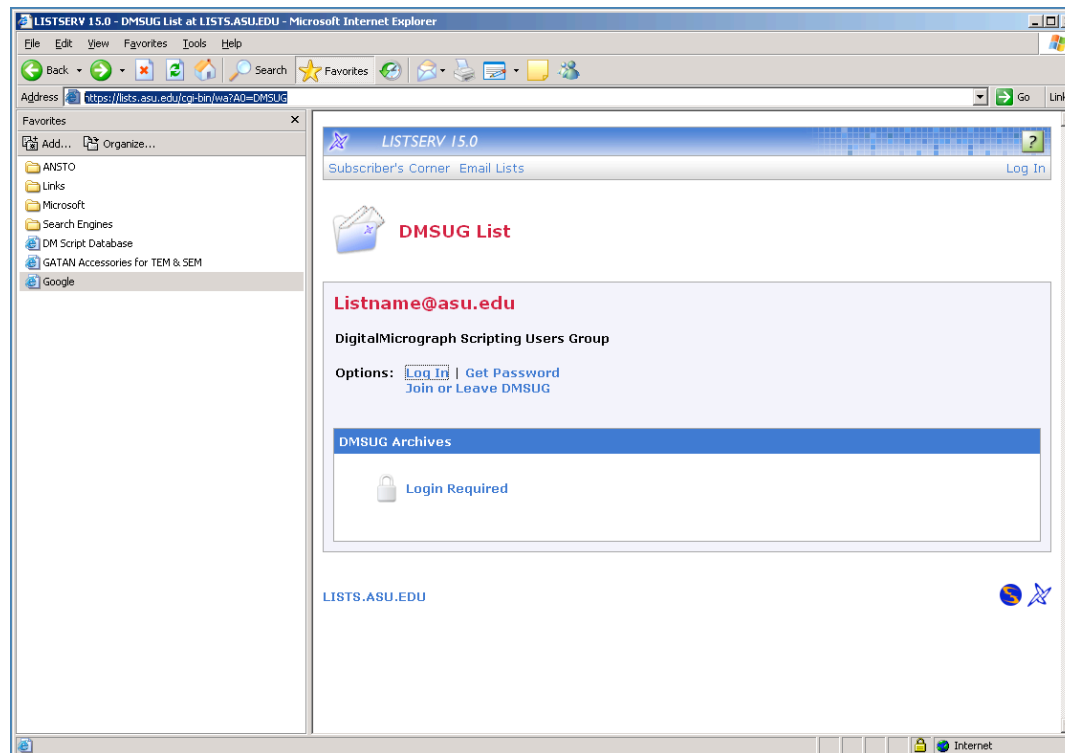
9. Scripting Resources - Online Help

- Online help contains good example code and explanations. Specific commands can be hard to find quickly.



9. Scripting Resources - DMSUG

- The Digital Micrograph Scripting Users Group (DMSUG).
- ≈80 subscribers including Gatan staff.
- <https://lists.asu.edu/cgi-bin/wa?A0=DMSUG>



9 Scripting Resources - Script Command

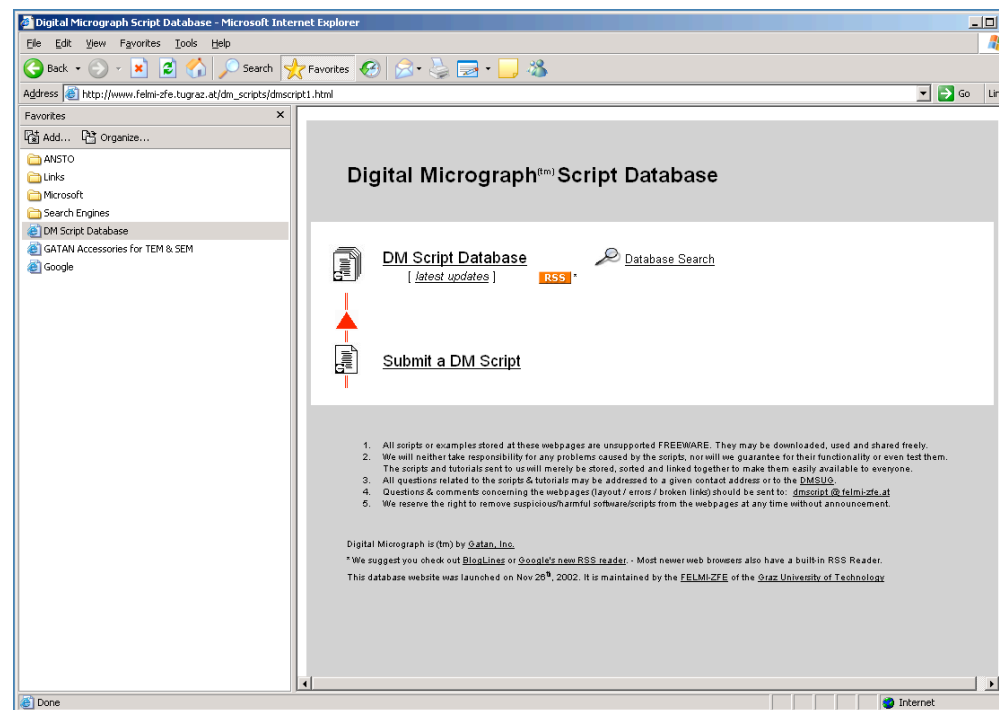
Web Archive (on CD and DM Script Database)

- Originally Gatan posted a full listing of scripting commands on the web - it is now incorporated into the online help.
- 'DM Script and Dialog' functions is a web archive of the original web-based listing.
- Use it to find commands quickly by keyword searching (ALT +f).
- Look for candidate commands for the functionality you require.
- Much quicker than looking in the online help.

```
'  
String GetLabel( ImageReference )  
    Return the image label of the image.  
→†  
void GetLimits( ImageReference, NumberVariable lowPtr, NumberVariable highPtr )  
    Stores display limits into the lowPtr and highPtr variables.  
→†  
void GetMaximalDocumentWindowRect( Number options, NumberVariable top, NumberVariable left, NumberVariable bottom, NumberVariable right )  
    Gets the bounds of the content region of the largest document window.  
→†  
String GetName( ImageReference )  
    Return the name of the image's image document.  
→†  
ImageReference GetNamedImage( String name )  
    Return the image with the image document name.  
→†  
Boolean GetNamedImage( ImageVariable, String name )  
    Store the image with the image document name into the image variable. Return 1 if one is found; return 0 otherwise.  
→†  
Number GetNextImageID( Number id )  
    Return the id of the image window following the image with the given id.  
→†
```

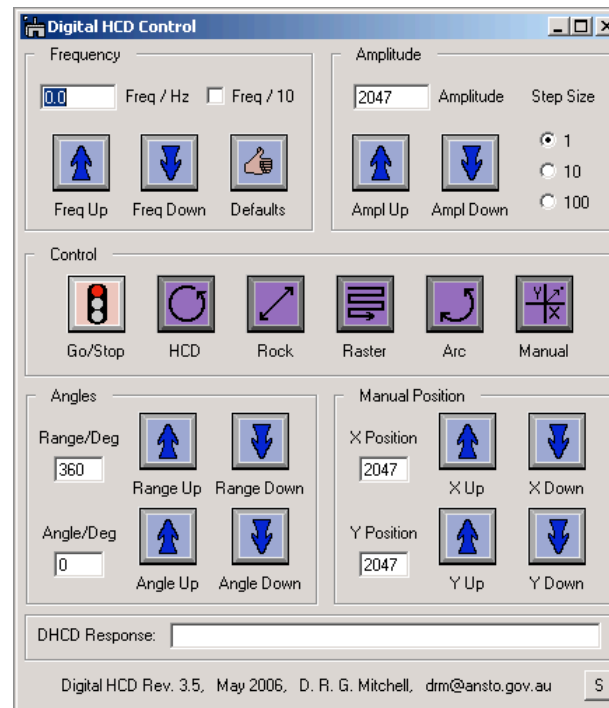
9. Digital Micrograph Script Database

- Repository for dozens of donated scripts, tutorials and example scripts.
- Please consider donating your scripts.
- http://www.felmi-zfe.tugraz.at/dm_scripts/dmscript1.html



10. Scripting Potential

- Scripting is very powerful, enabling DM to be tailored to meeting the experimentalist's needs.
- Scripts range from a simple image manipulation through to sophisticated dialogs for hardware control.
- Experiment!



Script-based control
Interface for a
Digitally Controlled
Hollow Cone Device

Workshop Aims : Review

- To provide users with a basic understanding of :
 - What scripts are.
 - How they work.
 - What scripts can do.
 - How to write scripts.
 - What scripting resources are available.
- Enlarge the pool of scripting talent.

Scripting Digital Micrograph™

Dave Mitchell

www.dmscripting.com

adminnospam@dmscripting.com (remove the nospam)