

Image analysis 2010

session 12

ImageJ macro programming

Morten Larsen

Department of Basic Sciences and Environment
Mathematics and computer science group
ml@life.ku.dk

Session 12 — Slide 1/87

Recording macros

The first step in creating a macro is usually to *record* some actions.

Activate with: PLUGINS, MACROS, RECORD...

Every action you perform from the menu becomes a step in the macro.

Typical recorded step:

```
run("Duplicate...", "title=newimage.tif");
```

A macro editing window may then be created with the CREATE button.

The macro can be saved in a file with extension “.ijm” or “.txt”.

Image analysis 2010 — Session 12 — Slide 3/87

Part I

Macro basics

- ① ImageJ Macros
- ② Image titles and IDs
- ③ Invoking macros and installing macro sets
- ④ Text output
- ⑤ Generalizing macros

Image analysis 2010 — Session 12 — Slide 2/87

The ImageJ macro editor

As editors go, the ImageJ macro editor is rather poor...

However, from the editor you can:

- Run the macro (CTRL-R).
- Run a single line (CTRL-Y).
- Run a selection (select and CTRL-Y/CTRL-R).

All very useful for debugging...

Recommendation: For larger macros, use your favourite editor, save the file and load it into the macro editor for debugging.

Image analysis 2010 — Session 12 — Slide 4/87

When doing exercises...

Create a macro for each exercise (FILE, NEW, TEXT WINDOW).

Copy-paste the relevant lines from the macro recorder.

Edit the macro text as needed.

Verify that you have the correct lines by running the macro!

Add comment lines stating which exercise this is (and other comments as needed).

Anything on a line after // is a comment:

```
// Exercise 2-15
// part (a)
run("Duplicate...", "title=newimage.tif"); // run on duplicate image
```

Macros: Other recordable functions

Some recordable operations are not invoked through run but rather have their own functions:

```
open("myimage.tif");
imageCalculator("Add create", "myimage1.tif", "myimage2.tif");
selectWindow("myimage.tif");
close();
```

Experiment with the recording feature to “discover” these functions and their parameters...

...or look them up in the macro functions reference.

Macros: the run function

The most common type of recordable macro step is invocation of some menu item with the run function:

```
run("Duplicate...", "title=newimage.tif");
run("Smooth");
run("Convolve...", "text1=[0 0 0 1 0 0 0 0] normalize");
run("Add...", "value=25");
```

A run call takes one or two arguments:

- The *verbatim menu item name* (required).
- Dialog *parameters* in the form *name=value*. If omitted, the associated parameter dialog (if any) will be displayed; if present, no dialog will be displayed.

The current image

Most functions work on the *current image*.

Whenever a new image is opened or created it becomes the current image.

You make a named image the current image with:

```
selectWindow("myimage.tif");
```

or with:

```
selectImage("myimage.tif");
```

selectWindow is recordable and works even for non-image windows.

selectImage works even for “hidden” images (in *batch mode*).

You can obtain the image title of the current image with the call getTitle().

Example with getTitle()

```
// Show difference between raw and median filtered current image
curtit = getTitle();
run("Duplicate...", "title=newimage.tif");
run("Median...", "radius=2");
imageCalculator("Difference create", curtitt, "newimage.tif");
selectImage("newimage.tif");
close();
// End by making same image current again
selectImage(curtitt);
```

Here `curtit` is a *variable* we *assign* to the title of the current image. The image title is unknown when we write the macro.

We can use `curtit` to refer to the stored title.

The temporary image "newimage.tif" has its title written into the macro (no variable needed).

Image IDs (II)

Image IDs work not only with `selectImage` but also with other functions normally taking image titles as arguments, e.g. `imageCalculator`.

It is always safer to use the image ID than other forms of identification, because:

- Image title may change, or there may be multiple windows with the same title...
- When a window is closed, the window menu changes...

Image IDs (I)

Other ways to identify images to `selectImage`:

- `selectImage(n)` for positive number `n` activates the `n`'th image in the WINDOW menu.
- `selectImage(id)` for a negative number `id` activates the image with unique ID `id`.

Image ID is a number uniquely identifying an image.

You can obtain the unique ID of the current image with `getImageID()`.

Using getImageID()

```
// Show difference between raw and median filtered current image
curID = getImageID();
run("Duplicate...", "title=newimage.tif");
newID = getImageID();
run("Median...", "radius=2");
imageCalculator("Difference create", curID, newID);
selectImage(newID);
close();
// End by making same image current again
selectImage(curID);
```

We store the IDs of the two images in the variables `curID` (original image) and `newID` (temporary image).

It is safer to get and use `newID` than to rely on the image title "newimage.tif" (why?).

Invoking macros

- *As a “plugin”*: Saving a macro “.ijm” file in the ImageJ **plugins** folder makes it appear in the PLUGINS menu. The same for a “.txt” file with at least one underscore in its name.
- *Directly from file*: With PLUGINS, MACROS, RUN...
- *From another macro*:

```
runMacro("filename", "arguments");
```

(full path or relative to the ImageJ **macros** folder).
- *Installed*: “Install” with PLUGINS, MACROS, INSTALL...; macros appear in the PLUGINS, MACROS menu.
Note: Only one macro file can be installed at any time.
Note: May specify keyboard shortcuts / tool buttons.
- *Installed at startup*: The file StartupMacros.txt.

Persistent variables

Installed macros may have global, persistent variables, which retain their values between macro invocations, as long as the macro set is installed.

```
var counter = 0;

macro "Increment [+]" {
    counter = counter + 1;
    print(counter);
}

macro "Decrement [-]" {
    counter = counter - 1;
    print(counter);
}
```

Note the var keyword in the declaration of counter.

print() outputs its argument(s) in the *Log window*.

Macro sets

A macro file to be installed may contain more than one macro:

```
macro "Translate right [r]" {
    run("Convolve...", "text1=[1 0 0\n]");
}

macro "Translate left [l]" {
    run("Convolve...", "text1=[0 0 1\n]");
}
```

These two macros have the keyboard shortcuts “r” and “l”.

A macro need not have a keyboard shortcut – simply omit the [...] after the macro name.

Output to log with print

The function print prints its argument(s) in the ImageJ log window.

```
print(2+2);
print("The value is: ", 6*7);
```

Extremely useful for debugging!

The log may be cleared by this special format print:

```
print("\\Clear");
```

The log may be saved in a text file:

```
selectWindow("Log");
run("Text...", "save=myfilename.txt");
```

Output to text window with print

print can output to a named text window instead of the log:

```
print("MyWindow.txt","Some text\n");
```

In this case there must be only two arguments and the first must be the window name in square brackets.

The \n means “new line” (not automatically added when not printing to log).

To open a new text window with a specific name:

```
run("New... ", "name=[MyWindow.txt] type=Text");
```

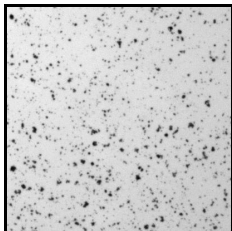
– or activate PLUGINS, NEW, TEXT WINDOW... from the menu.

Background subtraction macro v. 0

```
run("Duplicate...", "title=background");
run("Maximum...", "radius=15");
run("Minimum...", "radius=15");
run("Gaussian Blur...", "sigma=15");
run("Subtract...", "value=207");
imageCalculator("Subtract",
               "Cell_Colony.tif", "background");
selectWindow("background");
close();
```

To generalize macro: Parameterize, i.e. replace **constants** with variables.

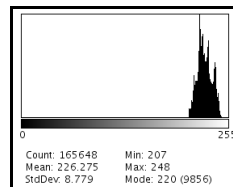
Example: Background subtraction



Cell_Colony.tif



background



Histogram of
background

```
run("Duplicate...", "title=background");
run("Maximum...", "radius=15");
run("Minimum...", "radius=15");
run("Gaussian Blur...", "sigma=15");
run("Histogram");
close();
run("Subtract...", "value=207");
imageCalculator("Subtract", "Cell_Colony.tif", "background");
selectWindow("background");
close();
```

Background subtraction macro v. 1

```
im = getImageID();
run("Duplicate...", "title=background");
bgim = getImageID();
run("Maximum...", "radius=15");
run("Minimum...", "radius=15");
run("Gaussian Blur...", "sigma=15");
run("Subtract...", "value=207");
imageCalculator("Subtract", im, bgim);
selectImage(bgim);
close();
selectImage(im);
```

We know how to make a macro robust with image IDs...

But some **constants** remain, embedded in strings.

String concatenation

Strings may be concatenated with the + operator:

"ab" + "cd" gives the string "abcd".

"radius=" + 7 gives the string "radius=7".

This is useful for run argument strings:

```
rad = 15;
:
run("Maximum...", "radius="+rad);
run("Minimum...", "radius="+rad);
run("Gaussian Blur...", "sigma="+rad);
```

Here rad is a variable.

Getting image statistics

The histogram and some information derived from it can be obtained in a macro as follows:

```
getRawStatistics(nPixels, mean, min, max, std,
                histogram);
```

On the places nPixels, mean, *et cetera* must be the names of variables *to receive* the information.

getRawStatistics will change the variable values to the statistics of the current image.

(Not very nice when this looks like a normal function call — functions do not change their arguments...)

We need only specify as many variables as we need, e.g.:

```
getRawStatistics(nPixels, mean, min);
```

Background subtraction macro v. 2

```
rad = getNumber("Enter radius", 15);
im = getImageID();
run("Duplicate...", "title=background");
bgim = getImageID();
run("Maximum...", "radius="+rad);
run("Minimum...", "radius="+rad);
run("Gaussian Blur...", "sigma="+rad);
run("Subtract...", "value=207");
imageCalculator("Subtract", im, bgim);
selectImage(bgim);
close();
selectImage(im);
```

The function `getNumber(prompt, default value)` asks the user for a number, in a pop-up window.

Background subtraction macro v. 3

```
rad = getNumber("Enter radius", 15);
im = getImageID();
run("Duplicate...", "title=background");
bgim = getImageID();
run("Maximum...", "radius="+rad);
run("Minimum...", "radius="+rad);
run("Gaussian Blur...", "sigma="+rad);
getRawStatistics(nPixels, mean, min);
run("Subtract...", "value="+min);
imageCalculator("Subtract", im, bgim);
selectImage(bgim);
close();
selectImage(im);
```

No more constants; the macro is fully general!

Part II

Some macro language elements

- ⑥ Variables, values and expressions
- ⑦ Strings
- ⑧ Arrays

Macro language arithmetic

The macro language supports all “usual” arithmetic operators: +, −, *, /, % (remainder).

Example: $y = x + 2;$

Parentheses may be used to change evaluation order.

Many common mathematical functions are defined: sin, cos, exp, log, pow, ...

Example: $(x^5) \ y = \text{pow}(x, 5);$

The constant π is written PI.

The *Process, Math, Macro...* filter allows you to use a macro expression to compute pixel values.

Macro variables

Variable names consist of letters, digits and/or underscore characters (the first character cannot be a digit).

Variables in ImageJ macros are *untyped* (may contain any kind of value) and are created when used (so need not be declared).

So e.g. the following sequence of assignments is legal:

```
x = 7;
x = "a string";
x = newArray(7, 9, 13);
```

The single equals sign “=” assigns a value to a variable.

The example also shows the three types of values in the macro language: *numbers* (floating point), *strings* and *arrays*.

Bit pattern arithmetic

Integers can also be manipulated as bit patterns.

Bitwise logical operators:

~ (complement), & (and), | (or), ^ (exclusive or)

Example: $y = \sim(x \& \text{mask});$

If e.g. x is 101_2 and mask is 110_2 then $(x \& \text{mask})$ is 100_2 and y becomes 011_2 .

There are operators for shifting bit patterns left or right:

<< and >>

Example: $y = x \ll 4;$

If e.g. x is 3 (11_2) then y will become 48 (110000_2).

Example: Global Sauvola & Pietaksinen

Sauvola & Pietaksinen threshold: $T = \bar{v} \left(1 + k \left(\frac{\sigma_p}{R} - 1\right)\right)$.

An ImageJ macro thresholding the current image with one global S&P threshold:

```
k = getNumber("Please enter k", 0.5);
R = getNumber("Please enter R", 128);
getRawStatistics(nP, mean, min, max, std);
T = mean*(1+k*(std/R-1));
setThreshold(0, T);
run("Convert to Mask");
```

setThreshold(low, high) sets the thresholds.

run("Convert to Mask") binarizes the image according to the thresholds set.

More string operations

To convert a string *s* to an integer: `i = parseInt(s);`

To convert a string *s* to a floating point number:

`f = parseFloat(s);`

or simply:

`f = 0+s;`

Extracting substrings:

`substring("abcde", 2, 4)` gives "cd".

(Indexing is zero based, so the first character is number 0.)

Other useful functions:

```
indexOf(haystack, needle)
startsWith(haystack, needle)
endsWith(haystack, needle)
lengthOf(string)
replace(string, pattern, substitution)
```

Strings

Strings are concatenated with the + operator:

`"ab" + "cd" + "ef"` gives the string "abcdef".

`"a" + 38` gives the string "a38".

That means a number can easily be converted to a string:

`("" + x)` gives *x* as a string.

Strings may be compared with the usual comparison operators.

Note: String comparisons are case insensitive, so e.g. `"yes"=="Yes"` is true!

Arrays

ImageJ supports 1D array variables.

An *array* is a variable containing multiple values, *indexed* by a number.

Indexing is zero based; the first element is number 0, the next is number 1, *et cetera*.

Square brackets are used around the index, e.g. `x[3]` for element number 3 of the array *x*.

Array elements may be assigned and referenced like variables.

Arrays of arrays are not allowed.

Array operations

Arrays may be created and used as follows:

```
x = newArray(7, 9, 13);
y = newArray("some", "string", "values");
z = newArray(10); // array of length 10 containing all zeros
print(x[1]);      // prints 9
print(y[2]);      // prints "values"
print(z[8]);      // prints 0
x[1] = 17;
print(x[1]);      // now prints 17
print(x.length);  // prints 3 (number of elements in x)
x[1] = "a";       // converts x to an array of strings...
```

Arrays have fixed length once created.

If both strings and numbers are stored in the same array, the numbers will be converted to strings.

Arrays cannot be compared directly with e.g. ==.

Some useful array functions

`Array.copy(A)` returns a copy of array A.

Example: `B = Array.copy(A);`

`Array.sort(A)` sorts the elements of A. The array must contain *either* numbers *or* strings.

`Array.getStatistics(A, min, max, mean, std)` stores statistics of A in the other variables.

The histogram is an array

The function `getRawStatistics` can give you the histogram of the current image, in an array:

```
getRawStatistics(nPix, mean, min, max, std, hist);
print(hist[0]); // Number of pixels with value 0
print(hist[255]); // Number of pixels with value 255
```

You can also get it with the function `getHistogram`:

```
getHistogram(0, hist, 256); // for 8-bit image
print(hist[0]); // Number of pixels with value 0
```

For a 16-bit or 32-bit image, e.g.:

```
getHistogram(vals, hist, 1000); // Use 1000 bins
print(vals[0]); // Start (lowest value) of bin 0
print(hist[0]); // Number of pixels in bin 0
```

Part III

Control structures

- 9 Conditional execution, if statements
- 10 for loops
- 11 while and do...while loops

Macro language boolean logic

There are the following comparison operators:

== equal to
 != not equal to
 < less than
 > greater than
 <= less than or equal to
 >= greater than or equal to

Example: `if (x != 0) ...`

Logical operators: `!` (not), `&&` (and), `||` (or).

Example: `if (x > 0 && y > 0) ...`

In ImageJ, `true` is 1 and `false` is 0 (so there is no real boolean type).

Example `if ... else`

```
x = getNumber("Enter value to add", 5);
if (x==0)
    showMessage("Adding zero would have no effect!");
else {
    print("Adding " + x + " to all pixel values.");
    run("Add...", "value=" + x);
}
```

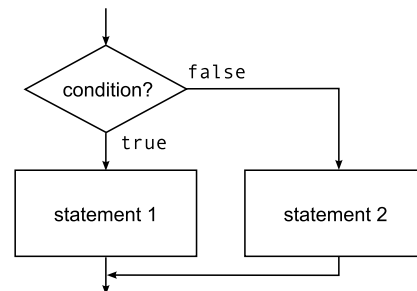
The function `showMessage` shows the message in a pop-up window.

Note the use of `{ ... }` to group together multiple statements as one.

Conditional execution: `if ... else`

Choosing between two alternatives:

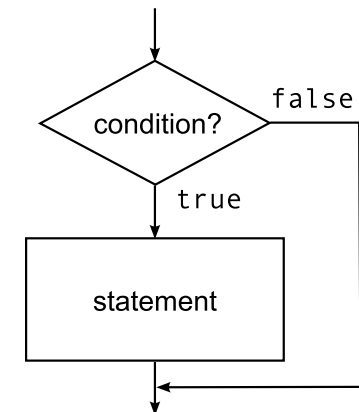
```
if ( condition )
    statement 1
else
    statement 2
```



Conditional execution: `if`

You may omit the `else` branch:

```
if ( condition )
    statement
```



Example if

```
if( bitDepth() != 32 ) {
    convert = getBoolean("Do you wish to convert to 32 bit?");
    if(convert) run("32-bit");
}
```

The code above checks if the current image is 32 bit and, if not, asks the user whether to convert it.

The `bitDepth` function returns the bit depth of the current image: 8, 16, 24 (RGB) or 32.

The `getBoolean` function pops up a window with “Yes” and “No” buttons; returns `true` for “yes” and `false` for “no”.

Example for (I)

```
for ( initialisation ; condition ; increment )
    body
```

The *body* is the statement or block { ... } to repeat.

Example: Printing the numbers from 1 to 100 in the log window.

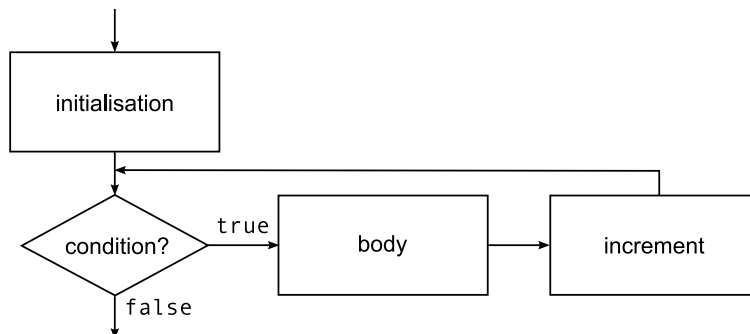
```
for(i=1; i<=100; i=i+1)
    print(i);
```

The loop above executes its body 100 times, with `i=1, 2, ..., 100`.

for loops

A `for` can repeat something a fixed number of times:

```
for ( initialisation ; condition ; increment )
    body
```



Example for (II)

```
for ( initialisation ; condition ; increment )
    body
```

Example: Repeating an operation a number of times.

```
iter = getNumber("Iterations", 1);
for(i=1; i<=iter; i++)
    run("Median...", "radius=1.5");
```

The loop above runs the radius 1.5 median filter `iter` times.

Note: `i++` is shorthand for `i=i+1`, i.e. “increment `i`”.

Example for (III)

```
for ( initialisation ; condition ; increment )
    body
```

Example: Finding the largest and smallest elements in an array A:

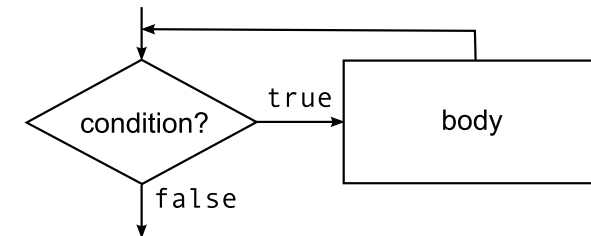
```
minval = A[0];
maxval = A[0];
for(i=1; i<A.length; i=i+1) {
    if ( A[i] < minval ) minval=A[i]; // A new minimum
    if ( A[i] > maxval ) maxval=A[i]; // A new maximum
}
```

After the loop, `minval` and `maxval` contain the minimum and maximum elements from A.

Loops: while

A while loops while a specified condition is true:

```
while ( condition )
    body
```



The *body* can be a statement or block { ... }.

Example for (IV)

```
for ( initialisation ; condition ; increment )
    body
```

Example: Finding the mode of the histogram.

```
getHistogram(0, hist, 256);
modei = 0;
for(i=1; i<256; i++)
    if ( hist[i] > hist[modei] ) modei = i;
```

After the loop, `modei` contains the number of the bin with the highest count, i.e. the most common pixel value.

Example while

```
while ( condition )
    body
```

```
while (nImages()>0) {
    selectImage(nImages());
    close();
}
```

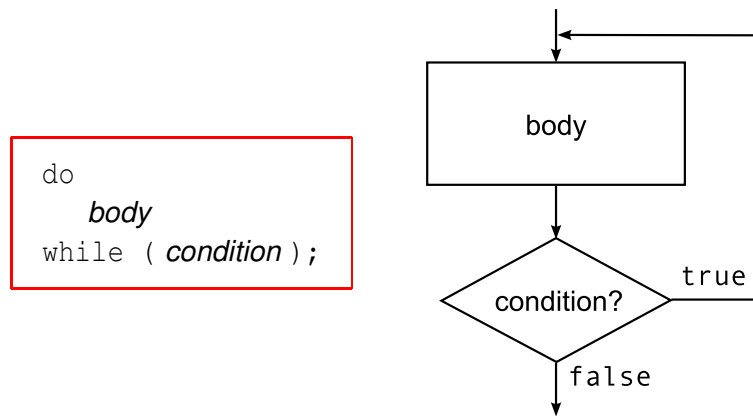
The loop above closes all displayed images.

The function `nImages` returns the number of open images.

The function `close` closes the current image.

Loops: do ...while

A do ...while executes the loop body at least once:



Part IV

Accessing pixels, results, and more

- 12 Pixels
- 13 The results table
- 14 Selections and ROIs
- 15 Accessing the filesystem
- 16 Batch Mode

Example do ...while

```
do
  body
while ( condition );
```

```
getHistogram(0, hist, 256);
run("Options...", "iterations=1 count=5");
do {
  numzero = hist[0];
  run("Erode");
  getHistogram(0, hist, 256);
} while( numzero != hist[0] );
```

This erodes a binary image as long as the number of zero valued pixels changes.

Pixel access functions

<code>getPixel(x,y)</code>	Get pixel value at coordinates (x,y).
<code>setPixel(x,y,value)</code>	Sets pixel at coordinates (x,y) to value.
<code>updateDisplay()</code>	Redraws the active image (otherwise pixel changes are only shown when macro exits).
<code>getWidth()</code>	Gets the width in pixels of the current image.
<code>getHeight()</code>	Gets the height in pixels of the current image.

Pixel manipulation example

Example: Invert a 5×5 block of pixels at the centre of the image:

```
cx = getWidth()/2;
cy = getHeight()/2;
for (x=cx-2; x<=cx+2; x++)
    for (y=cy-2; y<=cy+2; y++) {
        v = getPixel(x, y);
        setPixel(x, y, 255-v);
    }
```

Note: Extensive pixel manipulation (e.g. filtering) is *much* more efficient with a plug-in.

RGB pixel manipulation example

Example: Double the intensity of green in all pixels:

```
h = getHeight();
w = getWidth();
for (x=0; x<w; x++)
    for (y=0; y<h; y++) {
        v = getPixel(x, y);
        r = v>>16;
        g = (v>>8)&255;
        b = v&255;
        g = g*2;
        if (g>255) g = 255;
        setPixel(x, y, (r<<16)+(g<<8)+b);
    }
```

RGB pixel values

In ImageJ, RGB pixel values are coded as 24-bit numbers:

Pixel value: 1243791
 Binary: 000100101111101010001111
 Decimal: 18 250 143

Encoding the color above:

```
v = (18<<16) + (250<<8) + 143;
setPixel(x, y, v);
```

Decoding into red, green and blue components:

```
v = getPixel(x, y);
r = v>>16;
g = (v>>8)&255;
b = v&255;
```

(Recall: & is binary AND, << and >> shift bit patterns.)

The results table

nResults()	Number of results (rows).
getResult(colLabel,row)	One field value.
setResult(colLabel,row,value)	Set result field. Adds column if colLabel is new. Adds row if row==nResults().
updateResults()	Show results after change.
print("[Results]", text)	Append a line to results. Use \t to separate columns.
String.copyResults()	Copy results to clipboard.
run("Clear Results")	Clear results table.

Columns are identified by name, e.g. "Circ.".

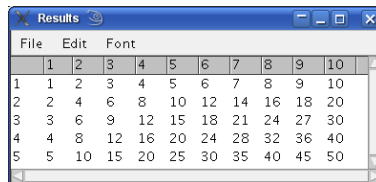
Rows are identified by number, starting at 0.

Results table example (I)

Create a 5×10 multiplication table:

```
for (row=1; row<=5; row++)
  for (col=1; col<=10; col++)
    setResult(col, row-1, row*col);
updateResults();
```

(Result row numbers start at zero.)



	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50

Results table example (III)

Compute statistics for circularity ("Circ." column):

```
circ = newArray( nResults() );
for(i = 0; i < nResults(); i++)
  circ[i] = getResult("Circ.",i);
Array.getStatistics(circ, min, max, mean, std);
print("Circularity range: "+min+" to "+max);
print("Mean circularity: "+mean);
```

Results table example (II)

Compute 2-axial symmetry, $\frac{\lambda_-}{\lambda_+} = \left(\frac{b}{a}\right)^2$, with a and b the half axis lengths of the best fit ellipse:

```
for(i = 0; i < nResults(); i++) {
  a = getResult("Major",i)/2;
  b = getResult("Minor",i)/2;
  sym = pow(b/a, 2);
  setResult("Symmetry",i,sym);
}
updateResults();
```

Selections (I)

In ImageJ, the user or a macro can *select* pixels for processing or measuring.

Three basic types: *area selection* (rectangle, ellipse, polygon, freehand, multiple), *line selection* (straight, polyline, freehand) and *point selection* (single or multiple).

(There is also an *angle selection*...)

An area selection is also called a ROI (region of interest).

Most operations changing pixels (filters) respect the ROI.

You can "Measure" a selection; it will generate a row of measurements in the results table.

Selections (I)

A binary mask image with the selected pixels set to 255 can be created from a selection.

Vice versa, a selection can be created from a binary image.

Selections can be *drawn* or *filled* with the current foreground colour.

(A fifth type of selection, *text selection*, is intended for writing text onto images...).

Selections in macros (II)

<code>getSelectionBounds(x,y,w,h)</code>	Stores, in <code>x</code> , <code>y</code> , <code>w</code> and <code>h</code> , the current selection's bounding box (coordinates and size).
<code>getSelectionCoordinates(x,y)</code>	Stores in <code>x</code> and <code>y</code> arrays of the coordinates that define the current selection.
<code>selectionType()</code>	Returns an integer "code" for the current selection type; -1 if no selection.
<code>run("Make Inverse")</code>	Inverts current selection.
<code>run("Create Mask")</code>	Creates binary mask image from selection.
<code>run("Create Selection")</code>	Creates selection from binary mask image.

Selections in macros (I)

<code>makeRectangle(x,y,w,h)</code>	Make rectangular selection with given upper left corner, width and height.
<code>makeOval(x,y,w,h)</code>	Make ellipse selection given bounding rectangle of ellipse.
<code>makePolygon(x1,y1,x2,y2,...)</code>	Make polygonal selection (3–200 points).
<code>makeLine(x1,y1,x2,y2)</code>	Make a line selection.
<code>makePoint(x,y,x,y)</code>	Make point selection.
<code>makeSelection(type,x,y)</code>	Make the different types of multi-point selections from <i>arrays</i> of <i>x</i> and <i>y</i> coordinates; see macro reference manual...

Selections in macros (III)

<code>run("Select All")</code>	Selects the entire image.
<code>run("Select None")</code>	Set "no current selection".
<code>setForegroundColor(r,g,b)</code>	Sets the current foreground colour, where <i>r</i> , <i>g</i> and <i>b</i> must be 0–255.
<code>run("Draw")</code>	Draws outline of current selection in current foreground colour.
<code>run("Fill")</code>	Fills current selection with current foreground colour.

The ROI manager

Selections may be stored in (and recalled from) the “ROI manager”.

Type `Ctrl-T` to add the current selection to the ROI manager and show the ROI manager window.

Selections may be saved to or restored to files in the ROI manager.

Selections can be drawn or filled from the ROI manager.

(Many other operations can be performed as well).

More ROI manager commands

More possible `roiManager(cmd)` calls:

<i>cmd</i>	Meaning
"Draw"	Draws the ROIs which are currently selected in the ROI manager. Draws all ROIs if none are selected.
"Fill"	Fills the ROIs which are currently selected in the ROI manager. Fills all ROIs if none are selected.
"Count"	<code>roiManager("Count")</code> returns the number of ROIs in the ROI manager.
"Index"	<code>roiManager("Index")</code> returns the index number of the currently selected ROI in the manager; -1 if none or more than one.

The ROI manager in macros

In macros, the function call `roiManager(cmd)` will perform ROI manager operations specified by the string *cmd*:

<i>cmd</i>	Meaning
"Add"	Adds current selection as new ROI.
"Add & Draw"	Add current selection and draw its outline.
"Delete"	Deletes the currently selected ROIs from the manager.
"Reset"	Remove all selections from the manager.
"Select"	<code>roiManager("Select",i)</code> marks ROI number <i>i</i> as selected (first ROI is number 0).
"Deselect"	Makes no ROIs selected in the manager.

Getting a list of files

The function `getFileList` gives you a list of files in a directory, as an array.

The functions `File.directory` and `File.name` give you the directory and name of the file last opened or saved.

Example: Print a list of (files named as) JPEG files in the directory of the last file opened:

```
dir = File.directory();
list = getFileList(dir);
for(i=0; i<list.length; i++)
    if(endsWith(list[i], ".jpg")) print(list[i]);
```

Note: The file names returned by `getFileList` do not include the path.

Some file-related functions

<code>name = File.getName(path)</code>	Get the last component (file name) of a path.
<code>dir = File.getDirectory(path)</code>	Get the directory component of a path.
<code>File.exists(path)</code>	True if the path exists, false otherwise.
<code>File.isDirectory(path)</code>	True if path is a directory.
<code>path = File.openDialog(title)</code>	A file chooser dialog.
<code>dir = getDirectory(title)</code>	A directory chooser dialog.
<code>s = File.openAsString(path)</code>	Return the contents of a text file as a string.
<code>s = File.openAsRawString(path, count)</code>	Return the first count bytes of the given file, as a string.

Entering and leaving batch mode

Batch mode is entered with:

```
setBatchMode(true);
```

Batch mode can be exited with:

```
setBatchMode(false);
```

To exit batch mode and display created images in new windows:

```
setBatchMode("exit and display");
```

Batch mode

Macros may run in *batch mode*, which may be (much) faster than normal mode.

While in batch mode,

- windows are not updated and
- created (new) images are not shown.

Thus batch mode is especially fast if many new images are created (and subsequently closed).

When batch mode is ended, updates are displayed.

Part V

User dialogs

- 17 Macro dialog windows
- 18 Special dialogs

User input in a dialog window

You may get parameters from the user from a *dialog*.

The steps are:

- ❶ Create an empty dialog window.
- ❷ Fill it with labels and fields.
- ❸ Show it and have the user interact with it.
- ❹ Save the entered field values in variables.

Macro user dialog example (2)

Add labels and fields:

```
Dialog.create("Translate Right");
Dialog.addNumber("X translation", 1, 0, 3, "pixels");
Dialog.addCheckbox("New image", false);
Dialog.show();
xoff=Dialog.getNumber();
asNew=Dialog.getCheckbox();
```

- A numeric field labelled "X Translation" with default value 1, 0 decimal places (integer), width 3 characters and unit label "pixels".
- A checkbox labelled "New image" and default value false (unchecked).

Macro user dialog example (1)

Create an empty dialog window:

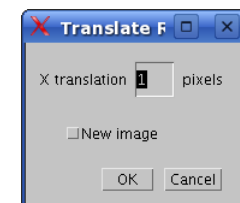
```
Dialog.create("Translate Right");
Dialog.addNumber("X translation", 1, 0, 3, "pixels");
Dialog.addCheckbox("New image", false);
Dialog.show();
xoff=Dialog.getNumber();
asNew=Dialog.getCheckbox();
```

Macro user dialog example (3)

Display the dialog and have the user interact with it:

```
Dialog.create("Translate Right");
Dialog.addNumber("X translation", 1, 0, 3, "pixels");
Dialog.addCheckbox("New image", false);
Dialog.show();
xoff=Dialog.getNumber();
asNew=Dialog.getCheckbox();
```

If the user presses CANCEL, the macro will be aborted.



Macro user dialog example (4)

Save the entered field values in variables:

```
Dialog.create("Translate Right");
Dialog.addNumber("X translation", 1, 0, 3, "pixels");
Dialog.addCheckbox("New image",false);
Dialog.show();
xoff=Dialog.getNumber();
asNew=Dialog.getCheckbox();
```

- Get the value of the (next) numeric field and save it in variable xoff.
- Get the value of the (next) checkbox field and save it in variable asNew.

Note: same order as fields were added to dialog.

Macro user dialog functions

Dialog.addMessage(text)	Add some text.
Dialog.addString(label, default) Dialog.getString()	Add/get a string field.
Dialog.addNumber(label, default) Dialog.addNumber(label, default, decimalPlaces, columns, units) Dialog.getNumber()	Add/get a numeric field.
Dialog.addCheckbox(label, default) Dialog.getCheckbox()	Add/get a checkbox field.
Dialog.addChoice(label, items) Dialog.addChoice(label, items, default) Dialog.getChoice()	Add/get a pull-down menu with items from a string ar- ray.

Macro user dialog example (5)

Completing the macro:

```
Dialog.create("Translate Right");
Dialog.addNumber("X translation", 1, 0, 3, "pixels");
Dialog.addCheckbox("New image",false);
Dialog.show();
xoff=Dialog.getNumber();
asNew=Dialog.getCheckbox();

if (asNew) run("Duplicate...", "title=Translated");
for (i=1 ; i<=xoff ; i=i+1)
    run("Convolve...", "text1=[1 0 0\n]");
```

This uses convolution in a for-loop to translate the image right by xoff pixels.

Example: Choose direction

```
// Possible directions:
dirs = newArray("left","right","up","down");

// Get the user's choice of direction:
Dialog.create("My translation");
Dialog.addChoice("Direction",dirs);
Dialog.show();
dir = Dialog.getChoice();

// Depending on direction, choose one of the translations...
if ( dir == "left" )
    run("Convolve...", "text1=[0 0 1\n]"); // left
if ( dir == "right" )
    run("Convolve...", "text1=[1 0 0\n]"); // right
if ( dir == "up" )
    run("Convolve...", "text1=[0\n0\n1\n]"); // up
if ( dir == "down" )
    run("Convolve...", "text1=[1\n0\n0\n]"); // down
```

Specialized dialogs

<code>showMessage(message)</code>	Show message in pop-up window.
<code>showMessageWithCancel(message)</code>	...with "Cancel" button.
<code>showStatus(message)</code>	Shows message in the status bar.
<code>x = getNumber(prompt, default)</code>	Dialog for a single number.
<code>s = getString(prompt, default)</code>	Dialog for a single string.
<code>b = getBoolean(prompt)</code>	Dialog for getting a yes or no answer.
<code>path = File.openDialog(title)</code>	A file chooser dialog.
<code>dir = getDirectory(title)</code>	A directory chooser dialog.

Pressing the "Cancel" button in any dialog will exit the macro.

Image analysis 2010 — Session 12 — Slide 81/87

Defining and using a function

Functions allow re-use of code (within a macro file):

```
function repeatTranslate(num, kernel) {
    for(i=1; i<=num; i++)
        run("Convolve...", "text1=["+kernel+"]");
}

Dialog.create("Image translation");
Dialog.addNumber("X translation", 1, 0, 3, "pixels");
Dialog.addNumber("Y translation", 0, 0, 3, "pixels");
Dialog.show();
xoff=Dialog.getNumber();
yoff=Dialog.getNumber();

repeatTranslate(-yoff, "0\n0\n1\n"); // up
repeatTranslate(yoff, "1\n0\n0\n"); // down
repeatTranslate(-xoff, "0 0 1\n"); // left
repeatTranslate(xoff, "1 0 0\n"); // right
```

Image analysis 2010 — Session 12 — Slide 83/87

Part VI

Re-using macro code

19 Defining and using functions

Image analysis 2010 — Session 12 — Slide 82/87

Return value

A function may return a value:

```
function maxOf3(n1, n2, n3) {
    return maxOf(maxOf(n1, n2), n3);
}

function minOf3(n1, n2, n3) {
    return minOf(minOf(n1, n2), n3);
}

print( maxOf3(7, 9, 13) ); // Will print 13
print( minOf3(7, 9, 13) ); // Will print 7
```

The built-in functions `maxOf` and `minOf` can only handle two arguments...

Image analysis 2010 — Session 12 — Slide 84/87

Local and global variables

Normally, all variables in functions are local to the function:

```
function setY(x) { y = x; }

y = 1;
setY(2);
print(y); // Will print 1
```

This can be changed by declaring the variable global (*outside* the function):

```
function setY(x) { y = x; }

var y = 1;
setY(2);
print(y); // Will print 2
```

Arrays and functions

Arrays are passed to functions *by reference* – that means you can change their contents:

```
function setVals(x, a) {
    x = 7;
    a[0] = 9;
}

y = 13;
b = newArray(1, 2, 3);
setVals(y, b);
print(y, b[0]); // Will print 13 9
```

An array may also be used to return more than one value from a function.

Local and global variables (2)

You must also use `var` to declare global variables to be accessed inside functions:

```
function area() { return sidelen*sidelen; }

sidelen = 7;
print( area() ); // Will fail...
```

This can be changed by declaring the variable global (*outside* the function):

```
function area() { return sidelen*sidelen; }

var sidelen = 7;
print( area() ); // Will print 49
```