

# Writing Macros in FIJI

## Ressources (ImageJ website)

Introduction to the macro language: <Help><Documentation> - Macro language  
<https://imagej.nih.gov/ij/developer/macro/macros.html>

Macro Functions: <Help><Macro Functions>  
<https://imagej.nih.gov/ij/developer/macro/functions.html>

Macro Examples: <Help><Macros>  
<https://imagej.nih.gov/ij/macros/>

Interactive help in the macro writing window

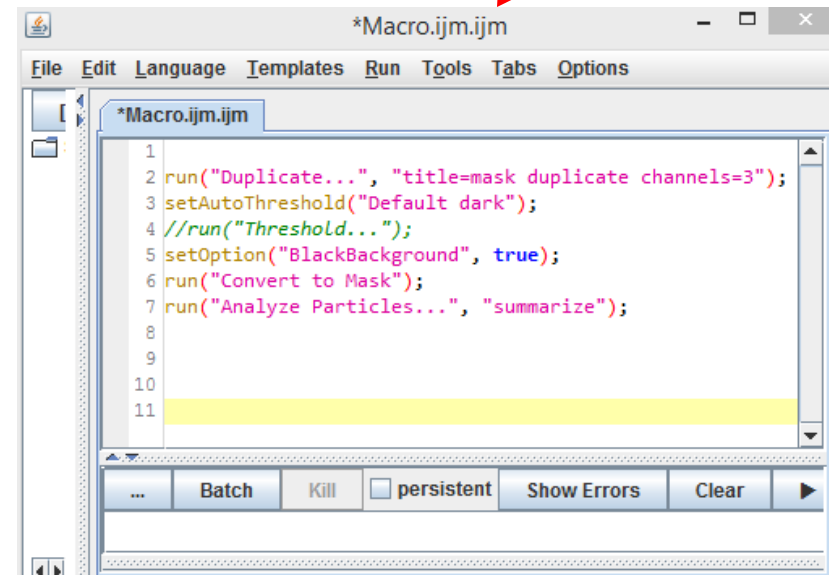
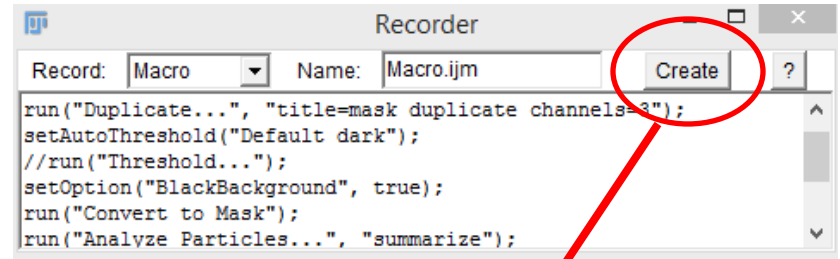
## Macro Creation

Do either:

- <Plugins><Macros><Record>
- <Plugins><New><Macro>
- Drag an existing macro onto the FIJI menu bar

# Basic Method Using Record

1. Work out/test the process
2. Record the process using the record option
3. Create the macro from the record window
4. Edit the code to remove unwanted items
5. Further code can be copied/pasted from the record window as necessary
6. Make the code generic
7. Add further code for:
  - Conditional statements
  - Loops
  - ROI Manager
  - Add dialogue windows
  - Output (results tables and plots)
  - File operations (opening and saving)
  - Functions



# Some Useful Methods/Code

Using the // to annotate the code in the macro

```
// this is the start of the main loop  
for (i=0;i<10;i++){
```

Using the // to comment out commands that you want to keep but deactivate temporarily

```
//run("Gaussian Blur...", "sigma=2");
```

Sometimes removing the parameters from recorded code will bring up the command dialogue. This is useful when testing. Final values can be edited later

```
run("Gaussian Blur...");  
run("Threshold...");
```

Using the “Duplicate” command can produce tidier code rather than using the “Split Channels” command – it can create a named copy of the specific channel/slice/frame without destroying the original.

```
run("Duplicate...", "title=mask duplicate");
```

Using the **exit ();** command to end the macro at a certain point or selecting parts of the code and using the <run> <run selected code> option on the menu bar of the macro window are both useful in testing

**NB. Syntax: Statements and functions must end in a semicolon “;”**

# Variables

## Assigning Information to Variables

- Assigning information to variables allows that information to be used in further coding
- Variables do not need to be declared and do not have explicit data types.
- A variable can contain a number, a boolean, a string or an array.
- They are automatically initialized when used in an assignment statement.
- Variable names are case-sensitive

Example:

**fn=getTitle ();** // gets the current image title and assigns it to the variable fn so for example, when you need to select that image again you can use the **selectWindow (fn);** command. You could also use it in the naming of a results table or plot window

**NB – when creating variable names, make them descriptive of the value they represent. Example: use Cell\_area rather than just A. It will make the code easier to read and to find bugs.**

# Constants and Arrays

## Using Variables as Constants

- Assigning fixed values to variable allows that information to be used in further code
- Useful for testing code with different settings.

Example:

```
1 Iterations=10;
2 for (it=1;it<Iterations;it++){
3     print ("iteration "+it+" of "+Iterations);
4 }
```

## Arrays

- An array is a variable containing a list of elements
- Usually needs creation before use: **name=newArray (elements)**  
(elements can be the number of elements or the actual values)  
**name=newArray (5);** or **name=newArray ("2","Green",25.45");**
- The elements can be referenced using square brackets [], the numbering starting at 0.  
eg Name[0]=2, name[1]=Green,
- Some macro functions will generate arrays **getFileList (directory)**  
returns an array containing the names of the files in the specified directory
- Only 1 dimensional arrays are allowed
- Array functions eg **Array.getStatistics(array, min, max, mean, stdDev)**

**More functions for arrays can be found on the ImageJ Macro Functions Website**

# Conditional Statements

```
if (condition){  
    code; //runs code if this condition is true  
}
```

-----  
-----

```
if (condition){  
    code; //runs code if this condition is true  
}  
else {  
    code; //runs code if condition is not true  
}
```

-----  
-----

```
if (condition){  
    code; //runs code if this condition is true  
}  
else if (another condition){  
    code; //runs code if this condition is true  
}  
else if (another condition){  
    code; //runs code if this condition is true  
}  
else {  
    code; //runs code if none of the above is  
true  
}
```

**if/elseif** - If the condition is true then the code between the braces {} will be run otherwise it is ignored

**else** – code between braces {} is only run if the above conditional statements are not true otherwise it is ignored

Conditional operators:

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

Joining conditions:

&&	boolean AND
	boolean OR

Useful tool for error checking

# Operators

Operator	Precedence	Description
++	1	pre or post increment
--	1	pre or post decrement
-	1	unary minus
!	1	boolean complement
~	1	bitwise complement
*	2	multiplication
/	2	division
%	2	remainder
&	2	bitwise AND
	2	bitwise OR
^	2	bitwise XOR
<,>	2	left shift, right shift
+	3	addition or string concatenation
-	3	subtraction
<, <=	4	less than, less than or equal
>, >=	4	greater than, greater than or equal
==, !=	4	equal, not equal
&&	5	boolean AND
	5	boolean OR
=	6	assignment
+=, -=, *=, /=	6	assignment with operation

**Parentheses can be used to define the order of operations, e.g.  $a*(b+c)$ .**

# Loops

## (for, while and do...while)

**NB. Syntax: The code in loops is delineated by Braces {}**

```
//for loop
for (i=0; i<10; i++) {
    code;
}
```

### For Loops

Format: for (start, end, step size) {  
Code in braces  
}

start, end, & step size consist of a variable name that is assigned a start number(=), an end point (<, >, ==, <=, >=) and the step size method (++ increment by 1, -- decrement by 1, += number or -= number for different step size. eg +=2 will increment by steps of 2)

```
//while - conditional loop
while (i<=90) {
    code;
}
```

### While Loops

Will run the code in the braces while the variable meets the condition

If the condition is not met, the code is not run

```
//do while - conditional loop
do {
    code;
} while (i<=90);
```

### Do While Loops

Will run the code in the braces at least once and will continue running the code while the variable meets the condition



# Making the Code Generic (and getting Image Information)

## Image Identifiers

**fn=getTitle ();** // gets the current image title and assigns it to a variable (eg fn)

**fn2=getInfo ("image.filename");** //gets the current image filename and assigns it to a variable

**Rename ("Channel 2 temp");** // useful to help make processing generic

## Image Information

**getDimensions (width, height, channels, slices, frames);** // gets the width and height in pixels, and the number of channels, z-slices and time frames in the image and assigns them to meaningful variables  
eg **getDimensions (ImageWidth, ImageHeight, ImageChannels, ImageSlices, ImageFrames);**

**getVoxelSize(x, y, z, unit);** // get the calibration of the pixels in X,Y and Z and the Units and assigns them to variables eg: **getVoxelSize(px, py, pz, unit);**

**Bit=bitDepth();** //( 8, 16 ,24 = (RGB) ,32 ) returns the bit-depth of the image as a number and assigns it to a variable (eg Bit)

**Stack.getUnits(Xunit, Yunit, Zunit, Timeunit);** // gets the units in a stack and assigns them to variables

**frameInterval=Stack.getFrameInterval();** //gets the time interval and assigns it to a variable

# User Interaction

## “Get” Commands

(displays an interactive dialog box and the result is assigned to a variable)

- **ans=getBoolean ("message")** Displays a dialog box containing the specified message and "Yes", "No" and "Cancel" . The result is assigned to a variable (eg ans)
- **num=getNumber ("prompt", defaultValue)** Displays a dialog box and returns the number entered by the user to a variable. (eg. num)
- **name=getString ("prompt", "default")** Displays a dialog box and returns the string entered by the user to a variable. (eg. name)
- **output=getDirectory (“window title”)** Displays a dialog box that allows the selection of a folder and returns the folder name to a variable. (eg. output)

## Wait for user

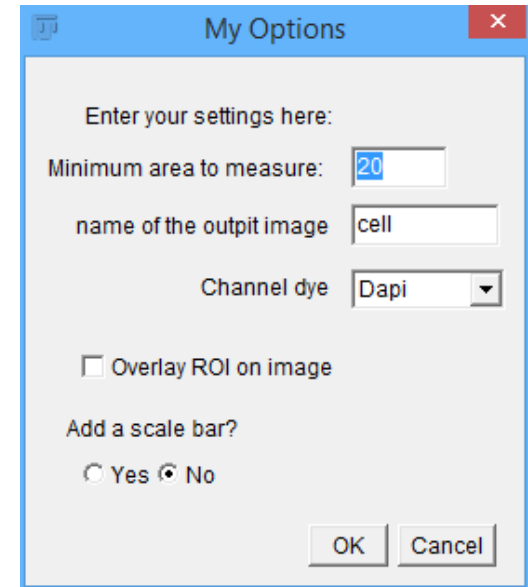
(pause the macro)

- Using **waitForUser (string)** halts the macro and displays message *string* in a dialog box. The macro proceeds when the user clicks "OK".
- Or **waitForUser (title, message)** as above - the *title* is the dialog box title and *message* is the text displayed in the dialog.

# Dialogues

## Pauses the macro to input multiple options

```
2
3 yesno=newArray("Yes", "No");
4 Dye = newArray("Brightfield", "Dapi", "488", "546", "568");
5
6 Dialog.create("My Options");
7     Dialog.addMessage("Enter your settings here:");
8     Dialog.addNumber("Minimum area to measure: ", 20);
9     Dialog.addString("name of the output image", "cell ");
10    Dialog.addChoice("Channel dye", Dye, Dye[1]);
11    Dialog.addCheckbox("Overlay ROI on image", false);
12    Dialog.addRadioButtonGroup("Add a scale bar?", yesno, 1, 2, "No");
13 Dialog.show();
14
15 area_limit=Dialog.getNumber();
16 Image_name=Dialog.getString();
17 Used_Dye=Dialog.getChoice();
18 Overlay=Dialog.getCheckbox();
19 Add_sb=Dialog.getRadioButton();
20
```



### Description:

- Lines 3 & 4 create two arrays that will be used in the dialogue
- The dialogue must start with **Dialogue.create (window title)** and finish with **Dialogue.show ()**;
- Messages can be added **Dialogue.addMessage ()**, numbers and strings entered, and checkboxes added eg. **Dialogue.addNumber ()**, **Dialogue.addString ()** etc. Also drop-down choices, and radio buttons can be added using the arrays for selection options.

After the **Dialogue.show ()**; command, the responses are assigned (**in the order they appear**) to variables, for use in the rest of the macro

**More dialogue commands can be found on the ImageJ Macro Functions Website**

# The ROI Manager

A key use of the macro is to manipulate ROI's (regions of interest) automatically particularly in loops

ROI's are stored as an array and can be referenced in the same way ie the first ROI is number 0

```
roiManager ("Reset");  
roiManager ("Add");  
n=roiManager ("Count");
```

```
roiManager ("Select",0);  
roiManager ("Rename", "");  
roiManager ("Set Color", "red");  
roiManager ("Set Line Width", 1);  
roiManager ("Remove Channel Info");  
roiManager ("Remove Slice Info");  
roiManager ("Remove Frame Info");  
  
roiManager ("Show None");  
roiManager ("Show All");
```

```
run ("Enlarge...", "enlarge=5");  
run("Interpolate", "interval=1");  
run("Rotate...", " angle=15");
```

```
roiManager ("Delete");  
  
roiManager ("Select", newArray(0,n));  
roiManager ("XOR");  
roiManager ("AND");  
roiManager ("OR");  
  
roiManager ("Measure");  
roiManager("Multi Measure");
```

**Most commands can be recorded or are available on the ImageJ macro Functions Website**

# Output

Data can be displayed in several ways

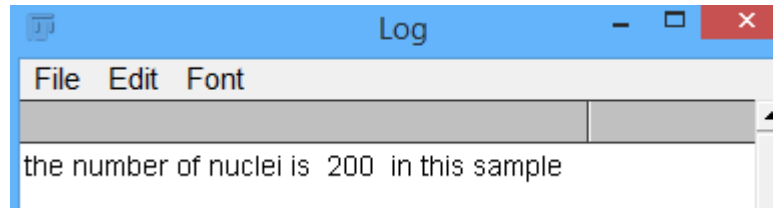
## The Print Command

- Using **print ("message" or values);** - Prints the contents of the brackets to the log window or a text window `print("[My Window]", "Hello, world")`.. Multiple objects can be joined using commas or the + symbol

Eg. `nuc_count=200;`

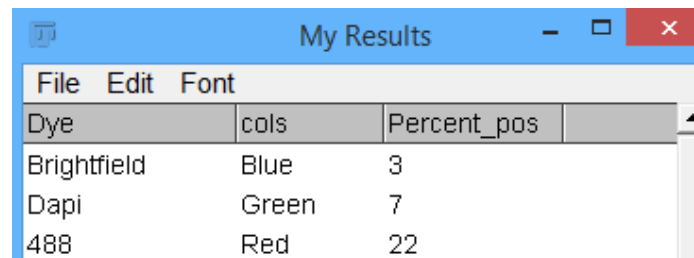
`Print ("the number of nuclei is ", nuc_count , " in this sample");`

Will display



## The Array show Command

- Data collected into an array can be displayed as a results table using **Array.show ("title", array1, array2, ...)** - Displays one or more arrays in a results window with *title* (optional)



A screenshot of a window titled "My Results" with a menu bar containing "File", "Edit", and "Font". The main content area displays a table with three columns: "Dye", "cols", and "Percent\_pos".

Dye	cols	Percent_pos
Brightfield	Blue	3
Dapi	Green	7
488	Red	22

# Results and Tables

## The Result Table from Analysis

A results/summary table - generated by <Analyse> <Measure> **run ("Measure")**; or <Analyse> <Analyse Particles> **run ("Analyze Particles...")**;

## Renaming Results Tables

Use either:

**IJ.renameResults (oldName,newName)**;

or in versions of FIJI later 1.52a

**Table.rename (oldName, newName)**;

## Creating tables using the Table commands

**Table.create (name)**;

**Table.set (columnName, rowIndex, value)**;

**Table.set (Column (columnName, array))**; - Assigns an array to the specified column.

## Extracting Data from Tables

Variable= **getResult ("Column", row)**;

Variable= **getResultString ("Column", row)**

Variable= **Table.get ("Column", row)**;

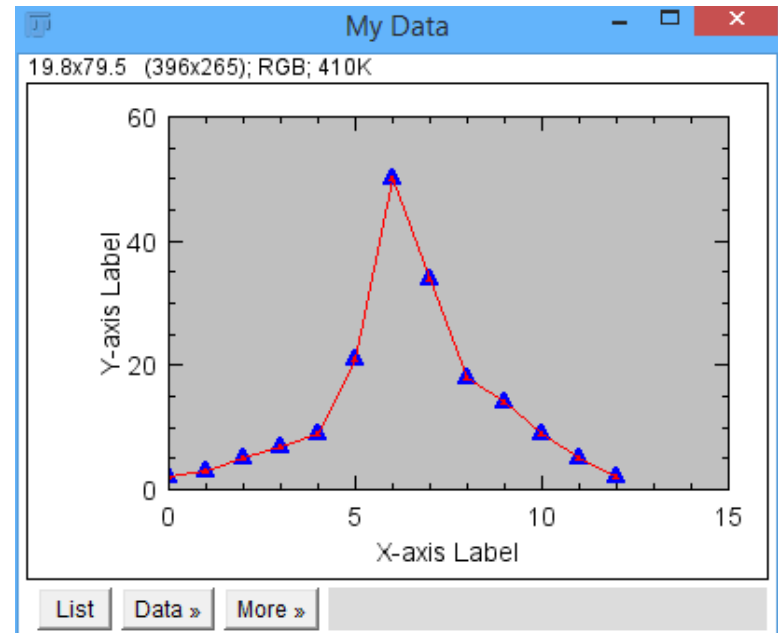
Variable= **Table.getString ("Column", row)**;

Variable array= **Table.getColumn (columnName)**; Returns the specified column as an array.

# Plots

## Functions to Generate and Display Plots.

```
data=newArray(2,3,5,7,9,21,50,34,18,14,9,5,2);  
Plot.create ("My Data", "X-axis Label", "Y-axis Label");  
Plot.setLimits (0, 15, 0, 60);  
Plot.setFrameSize (300, 200);  
Plot.setBackgroundColor ("lightgray");  
Plot.setColor ("blue", "red");  
Plot.setLineWidth (2);  
Plot.add ("triangle", data);  
Plot.setColor ("red", "red");  
Plot.setLineWidth (1);  
Plot.add ("line", data)  
Plot.show ();
```



**more commands can be found on the [ImageJ Macro Functions Website](#)**

**Curve fitting functions available using the separate “Fit” functions**

# File Operations

`open()`; //opens a folder window to select file to open (tiff type images)

`run("Bio-Formats Importer", "open= autoscale color_mode=Composite view=Hyperstack stack_order=XYCZT");` //opens a folder window to select file to open (Raw data type images)

(the second set of parameters list the bio-formats options: open= with no name will open the file dialogue, autoscale sets the autoscale to ticked, color\_mode= allows setting of the way the image will appear once loaded, etc)

## Getting the Directory or Folder

Variable = `getDrectory("Window title text");`

Examples: `input = getDirectory("Input folder where images are stored");` //opens a dialogue to select directory of images to open and assigns it to a variable (eg input)

`output = getDirectory("Output folder for results");` //opens a dialogue to select location where images/results are to be stored and assigns it to a variable (eg output)

`list = getFileList(input);` //gets list of files in the folder and assigns it to a variable array.  
`Array.show(list);` would display the list. Note array numbering starts at 0 so the first image would be referenced by `list[0]`.



# Saving

**saveAs (format, path) – eg** saveAs("Tiff","Image\_name");

*The path name must end in ".tif", ".jpg", ".gif", ".raw", ".avi", ".png", for images or ".txt", ".xls", ".csv" for results tables or ".zip" for rois*

The *format* argument must be "tiff", "jpeg", "gif", "zip", "raw", "avi", "bmp", "fits", "png", "pgm", "text image", "lut", "selection", "results", "xy Coordinates" or "text".

Use **saveAs ("format")** to have a "Save As" dialog displayed.

The saving can be combined with the output folder as previously assigned to a variable

```
//save image
```

```
Image_out = output + "Image name";
```

```
saveAs ("Tiff", Image_out);
```

```
//save results file
```

```
Res_out = output + "measurements_name.xls";
```

```
saveAs ("results", Res_out);
```

```
//save rois
```

```
Roi_out = output + "ROIs_name.zip";
```

```
roiManager ("Save", roi_out);
```

# Batch Processing (user code)

```
input = getDirectory("Input folder for images"); //get input folder

output = getDirectory("Output folder for results"); //get output folder

list = getFileList(input); //get list of files

for (image=0; image<list.length; image++){ //start of loop for opening images

full = input + list[image]; //reads the images sequentially from the list array
open(full); //open image (or use bio-formats command)

fn=getTitle(); //get file name
//code
//code
//code

out = output + fn + " Cell overlay"; – or use array name from list instead of fn: list[image]
saveAs("Tiff",out); //save drawing or image
close(); // closes the image

out = output + fn + " measurements.xls";
saveAs("Results",out); //save results file
selectWindow("Results"); //select the results window
run("Close"); //closes the results window
} //end of loop
```

# User-defined Functions

A user-defined block of code that can be passed values and can return a value, useful for replacing repetitive code and breaking larger macros into blocks of routines. Can be placed at the end of the main macro code.

## format

```
function myFunction(arg1, arg2. arg3) {
```

```
Code
```

```
Code
```

```
return (optional return value statement )
```

```
}
```

And can be called by: **myFunction (arg1, arg2. arg3);**

(arguments arg1,arg2,etc are optional but can be used to give values to the function, and the number of arguments given when calling a function must correspond to the number of arguments in the function definition).

Functions can also return a value to a variable

```
value= myFunction(arg1, arg2. arg3);
```

# User-defined Functions – example

This function can be used to generate a random colour for use in an ROI and plot display

```
//random colour generator
function ColourGen(){
Red=floor(random*255);
RedHex=toHex(Red);
Green=floor(random*255);
GreenHex=toHex(Green);
Blue=floor(random*255);
BlueHex=toHex(Blue);
RoiCol="#" + RedHex + GreenHex + BlueHex;
return RoiCol;
}
```

And is called by; **Col=ColourGen();**

And the return value assigned to Col is used to colour the roi

```
roiManager ("Set Color", Col);
```

Which is then used later to colour the plot

```
PlotCol=Roi.getStrokeColor;
```

```
Plot.setColor (PlotCol);
```

# Useful Code Snippets

```
//initialise
```

```
roiManager("Reset"); // clears the roi manager  
run("Clear Results"); //empties the results table  
run("Select None"); //makes sure there is nothing selected
```

```
//Image information
```

```
fn=getTitle(); //gets image title  
getDimensions(ImageWidth, ImageHeight, ImageChannels, ImageSlices, ImageFrames);  
// gets the image dimnesion properties  
getVoxelSize(px, py, pz, unit); // gets the pixel dimension  
setVoxelSize(px, py, pz, unit); // sets the pixel dimension  
Bit=bitDepth() ; // gets the bit depth ( 8, 16 ,24 (RGB) ,32 )  
selectWindow ("image title"/or variable);
```

```
//ROI control
```

```
total_roi=roiManager("Count"); //gets the number of rois  
  
for (roi=0; roi<total_roi; roi++) { // loop through the rois  
    roiManager("Select", roi);  
    roiManager("Remove Channel Info");  
    roiManager("Rename", " ROI "+roi+1);  
    roiManager("Measure");  
}
```

# Useful Code Snippets (continued)

```
// changing the default colours and other settings
```

```
run("Colors...", "foreground=white background=black selection=white");
```

```
run("Properties...", " stroke=red width=5");
```

```
setTool("rectangular");
```

```
setLineWidth(width);
```

```
// using specify to create an ROI
```

```
run("Specify...", "width=4 height=4 x=a y=b slice=1 oval centered scaled");
```

```
run("Add Selection...");
```

```
// using create selection to create a multiple object ROI
```

```
run("Create Selection");
```

```
roiManager("Add");
```

```
// multi-dimensional commands
```

```
Stack.setChannel(Chan);
```

```
Stack.setSlice(Slice);
```

```
Stack setFrame(Frame);
```

```
Stack.getPosition(channel, slice, frame);
```

```
Stack.setPosition(channel, slice, frame);
```

```
Stack.getStatistics(voxelCount, mean, min, max, stdDev);
```

# Useful Code Snippets (continued)

```
//save results file if they exist  
if (isOpen("Results")==true){ //error checking  
out = output + fn + " measurements.xls";  
saveAs("Results",out);  
selectWindow("Results");  
run("Close");  
}
```