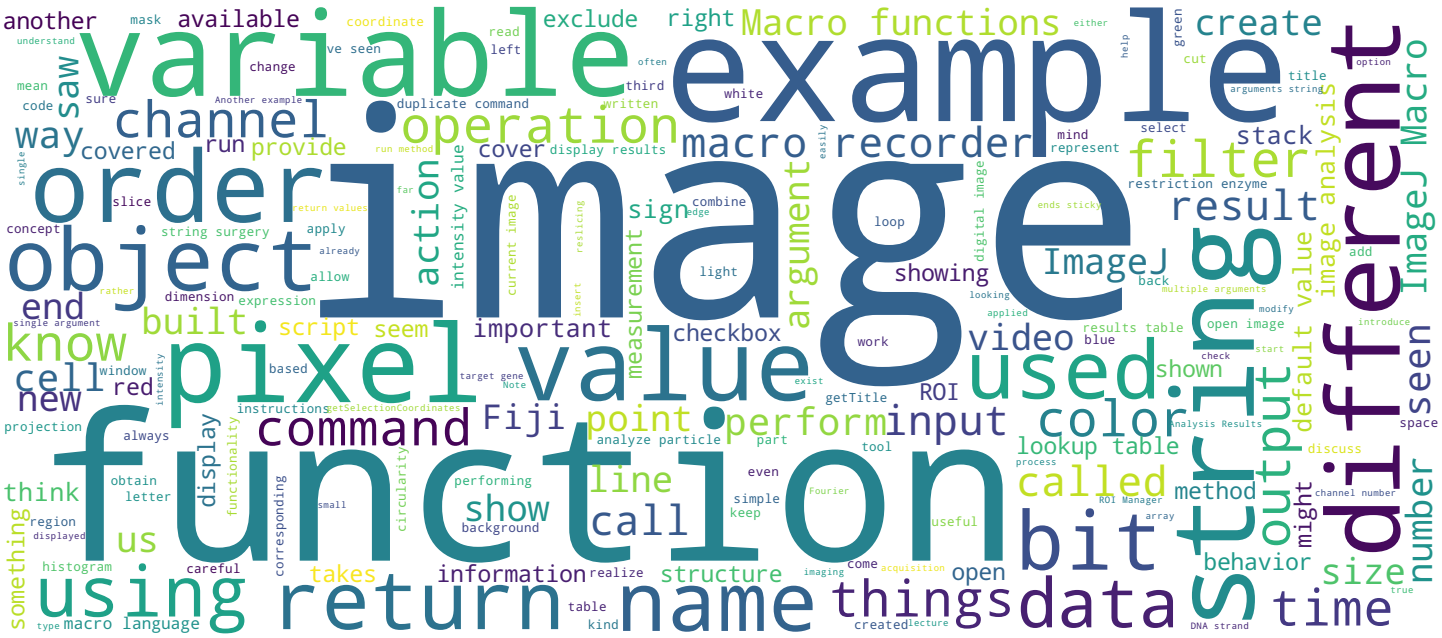
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

ImageJ Macro Functions and Custom Functions

Image Processing & Analysis for Life Scientists

Olivier Burri, Romain Guiet & Arne Seitz



Outlook

- Macro Functions in Fiji
- Types of Functions
- `run()` String Surgery

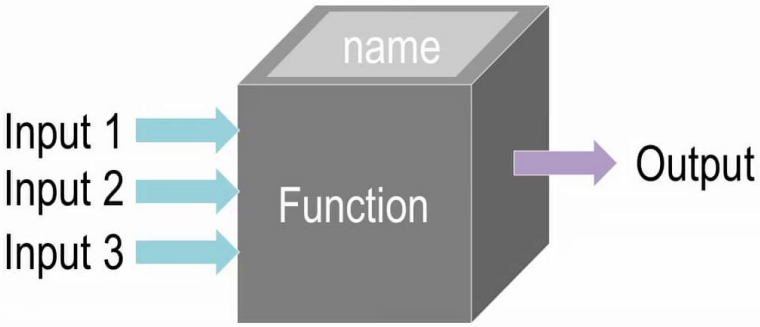
Welcome to this video where we will cover functions in the ImageJ Macro Language. We will see how to write your own functions, and we'll study the different function types available, how to use them, and we'll discuss how to add our own variables into imageJ's run command.

Notes

Summary

0m 05s

Function Fundamentals



So, what's a function anyway? We can think of it as something that can take inputs, and performs an action with them. Which would result in an output, which could be a value or an action.

Notes

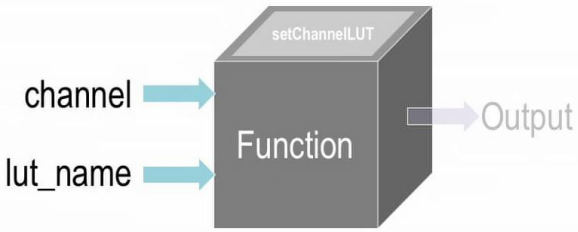
Summary

0m 21s

Custom Functions

```
function myFunction(arguments) {
    // Your code goes here
}
```

```
function setChannelLUT(channel, lut_name) {
    Stack.setChannel(channel);
    run(lut);
}
```



We can write our own functions, and the structure in the ImageJ Langage looks like this. You start off by typing the special keyword with the special word "Function", followed by the name you want to use. Note that by convention, the first letter should be lowercase and the first letter of each word should be uppercase. Then, in parentheses, you will provide the arguments, or what we would call inputs, of your function. The actual macro code goes between the two curly brackets that encapsulate the beginning and the end of the function. As an example, we created a function we chose to call set channel lookup table which takes the channel number and a lookup table name. The function will set the channel of the current image to whatever we wish, and it will turn it into the given lookup table. This is a summary of what our function looks like. It takes two inputs and provides no output other than the current image having a new lookup on the chosen channel.

Notes

Summary

0m 34s

Functions

- Functions are subroutines
  - A sequence of instructions that performs a specific task, packaged as a unit.
  - Is reusable

```
// returns the title of the current image
image_title = getTitle();

// returns all the xy coordinates of the
// current ROI
getSelectionCoordinates(xpoints, ypoints);
```

This is how we would use our new function. Suppose we have a 3 Color image open. And we'd like to set up the first channel as cyan, the second as green and the third as magenta. And we could just type it like this, In this script editor. Fundamentally a function is a sequence of instructions that is packaged as a unit. When you call it, it will perform some action. And, what is very useful about it, is that you can reuse it as many times as you need. Here, I'm showing you two examples of typical functions already available in ImageJ: getTitle, for example, returns the title of the currently open image In the background, this is performing a series of instructions such as checking if an image is open what the open image currently is, and querying that image for its name, checking if the name is valid, and, finally, if it's valid, sending it back to us as a string. But you don't need to know all the functionality that that function is performing. All you need to know is its name and what the output is. Another example is getSelectionCoordinates, that checks if there is a ROI, and then picks up the points in x and y and returns them as two separate arrays, one for the x coordinates, and one for the y coordinates.

Notes

Summary

1m 39s

ImageJ Macro Functions

With ImageJ 1.48h or later, you can use 'show' and 'hide' arguments to individually show or hide images. By not displaying and updating images, batch mode macros run up to 20 times faster. Examples: BatchModeTest, BatchMeasure, BatchSetScale and ReplaceRedWithMagenta.

setBatchMode("exit and display")
Exits batch mode and displays all hidden images.

setBatchMode("show")
Displays the active hidden image, while batch mode remains in same state.

setBatchMode("hide")
Enters (or remains in) batch mode and hides the active image

setColor(r, g, b)
Sets the drawing color, where *r*, *g*, and *b* are >= 0 and <= 255. With 16 and 32 bit images, sets the color to 0 if r=g=b=0. With 16 and 32 bit images, use *setColor(1,0,0)* to make the drawing color the same is the minimum displayed pixel value. SetColor() is faster than setForegroundColor(), and it does not change the system wide foreground color or repaint the color picker tool icon, but it cannot be used to specify the color used by commands called from macros, for example *run("Fill")*.

setColor(value)
Sets the drawing color. For 8 bit images, 0<=*value*<=255. For 16 bit images, 0<=*value*<=65535. For RGB images, use hex constants (e.g., 0xff0000 for red). This function does not change the foreground color used by *run("Draw")* and *run("Fill")*.

All available macro functions are in the built in macro functions page from the ImageJ website.

Notes

Summary

2m 59s

Types of ImageJ Macro Functions

- 2 ½ 'types'
  - **Methods**: Do something

```
// Activates the window called "Analysis Re
// for further processing
selectWindow("Analysis Results");
```

  - **Functions**: Return values

```
// returns the title of the current image
image_title = getTitle();
```

  - **System Variables:** Access internal ImageJ state

```
// Gives the number of rows currently in the
// Results Table
nResults;
```

You can reach the page by going to Help, Macro functions We can roughly classify ImageJ Macro functions into two categories and a third one that is a little special. The first category, we will call methods. They return nothing, but they affect the behavior of imageJ. for example here, selectWindow will activate the ImageJ window with the name "Analysis Results" in this example. You see we don’t get anything back. It returns nothing, but it affected the behavior of ImageJ. The "Analysis Results" Window is now currently active And we can do things with it. What we call functions are things that will return values, such as getTitle, like we've seen before. Finally, we have this last type, that we will call system variables. There are not really functions per say but they give us information about the current state of ImageJ, like nResults here.

Notes

Summary

3m 05s

Types of ImageJ Macro Functions

```
// returns the title of the current image
image_title = getTitle();
```

Store the return value of the function

- **Functions**: Return values

Provide variables that will be filled by the function

```
// returns all the xy coordinates of the
// current ROI
getSelectionCoordinates(xpoints, ypoints);
```

All it does is return the number of rows currently in the Results table Going back to functions, you might have noticed that there seem to be two kinds of behaviors. Some, like getTitle, will return a value. Note that functions like these can only return a single value, unlike languages like matlab, for example. But there is a way for them to return more than one value, which is counter-intuitive from our first definition. So if we look at getSelectionCoordinates, you'll see that it takes two inputs. But the function will modify, xpoints, ypoints in this case. and fill these variables with the appropriate values Please note that this is rather unusual, as in most languages, return values or variables would always appear on the left of an equal sign. But do keep this in mind so as not to be confused between return variables and, in this case, the input arguments. And, feel free to read the ImageJ documentation for each function, so that you can make sure to understand how to use it best.

Notes

Summary

4m 04s

Macro Function Structure

```
keyWord(argument1, argument2);

roiManager("select", index);
```

Now, most ImageJ Macro functions would have a structure like this. There's a keyword, that's built in, and you need to provide arguments, or inputs, that the function needs. In this example, we realize that selectWindow needs a string with the name of the window to be selected, So it has a single argument. The ROI Manager command, on the other hand, can take two arguments. Here the first is a string with the action to perform, “Select”, and a second argument, with the index of the ROI to select.

Notes

Summary

5m 17s

Run Method

```
run("Command", "Arguments String");
```

```
run("Duplicate…", "title=[Image Copy]");
```

Now, we need to cover one function you will see a lot, especially when using the recorder. And that is the "run" method This is a bit of a do-it all method. The first argument is actually the name of the imageJ command, plugin or button you use, and the second argument is… the multiple arguments to that command. So, notice how both arguments are actually strings, and that inside the second string, we can find multiple arguments separated by spaces. So, to try and make it more clear, let's look at an example: If we look at the duplicate command here, which I have applied to a simple 2D image, simply needs the title of the new image. Which is given in the following way: you see that after the comma there is the keyword "title", followed by an equal sign and the name of that duplicated image that we want to create. So, in what we will now call the arguments string, there is a single argument, title, and its value is Image Copy.

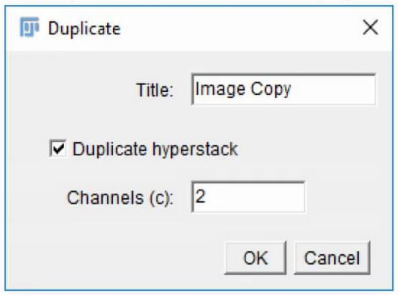
Notes

Summary

5m 52s

Run Method

```
run("Command", "Arguments String");
```




```
run("Duplicate...", "duplicate title=[Image Copy] channels=2");
```

Let's look at a slightly more complex example with the same duplicate command but, this time, with a multichannel image. We see that we need to provide the channel number we want to duplicate. So what will that look like? If you run the command, and have the macro recorder on, this is something that it will return. you see that it still runs duplicate, but in the arguments string, we see that there are a few more things now.

Notes

Summary

6m 58s

Run Method

```
run("Command", "Arguments String");

run("Duplicate...", "duplicate title=[Image Copy] channels=2");
```

Argument 1 Argument 2 Argument 3

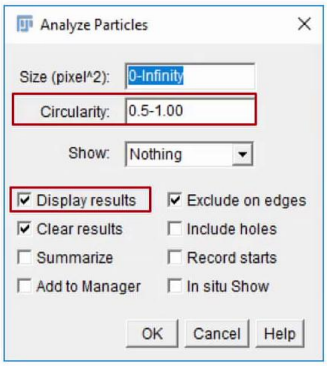
There are three arguments inside the string. duplicate, the title, as we've seen before, and the channels that were chosen.

Notes

Summary

7m 23s

Macro Function Structure




```
run("Analyze Particles...", "circularity=0.5-1.00 display exclude clear");
```

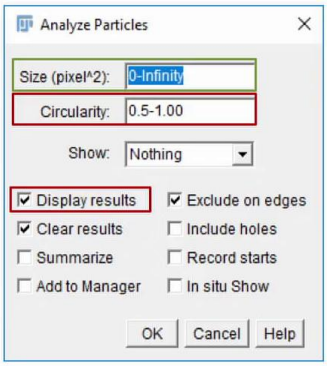
Now, I just want to tell you that you do not need to know these commands by heart! The only point here is for you to get an understanding on how and why they are built, so that you are able to interpret what you see when you're looking at the output of the macro recorder, and it doesn't seem incomprehensible. But let's look at another example. Here we are running the very common analyze particles command on an image, with a non-default value in the circularity. Which is 0.5 And we've checked display results, clear results and exclude on edges. So, what will that look like after we run the macro recorder? All right, we've seen that it's rather complex. But let's break it apart. We first see that the circularity value was entered here. And that it shows up in the macro recorder. Then there is a space and display results which seem to have been shortened to just display. And now, maybe you've noticed the pattern. For the name of the argument, it is always the first word of the dialog, like circularity or the 'exclude' in the exclude on edges. Arguments either have an equal sign followed by their value that they will have, and, if it's a checkbox, the checkbox is considered 'checked' if the first word describing the checkbox is present.

Notes

Summary

7m 32s

Macro Function Structure



Default values are usually not recorded!

```
run("Analyze Particles...", "circularity=0.5-1.00 display exclude clear");
```

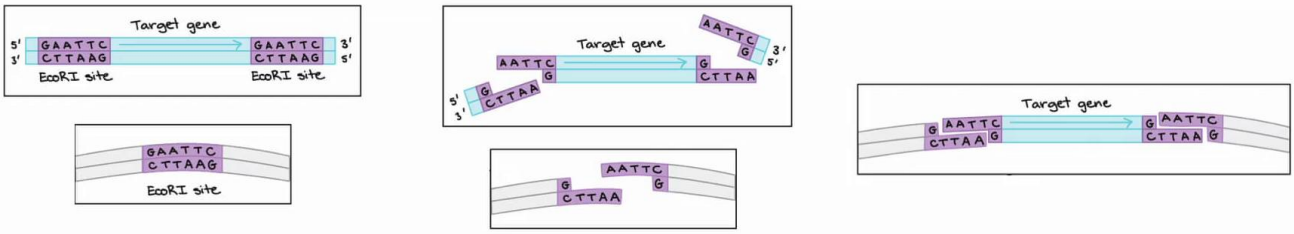
Notice however that default values are not recorded. So be careful when using the recorder, and try to change the default values if you realize that the macro recorder isn't showing you all the options.

Notes

Summary

8m 47s

String Surgery



So we've seen how the run method needs two string arguments to work. But how do we actually blend that with the variables we will have created, to use it in our scripts? For that, we'll need to to a bit of string surgery. And to show you this, I will first show you something completely different, but perhaps more familiar, before showing you how to do it. In classical biology, when you need to combine two bits of DNA, a classical method is to use restriction enzymes. In the example here, we would like to place the target gene above into the other DNA strand below. Because they both have EcoRI sites, we can use the EcoRI enzyme to cut the target gene, and open the DNA strand below. Because these have what are called 'sticky ends', then there will be a high chance that they will recombine and form what we were expecting. So, you know...

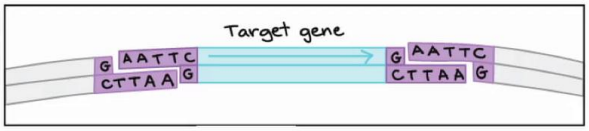
Notes

Summary

8m 59s

Adding Variables to Strings

Variable [redacted]



String "[redacted]"+ [redacted] +"[redacted]"

```
my_title = "This title is in a variable";
run("Duplicate...", "title=["+my_title+"]");
```

Classic, beautiful biology. Now, what does this have to do with us and our current problem? Let’s look at an example command. Here, we're seeing the duplicate command again to which we would like to modify the title using a variable. Now, as you see on the top right, we are in a very similar situation as for the restriction enzyme example. We want to combine, or concatenate, a variable into a string. Basically, we need to do the job of the restriction enzyme and cut the string where we want to insert our variable; and remove the part of the string we do not want. Now we need to make the ends ‘sticky’ somehow for the variable actually to concatenate inside the string, and for this to be valid code. Seeing as what we've basically is split the string into two, the first thing we need to do is make sure that both strings are enclosed in the double quotes. Then we finish making the ends sticky by adding two plus signs, which represent this concatenation operation, when working on strings. Now we can finally insert our variable inside the string and it should look like this. There you go! Now you know how to do string surgery.

Notes

Summary

9m 54s

```
        image_name = getTitle();
        blur_sigma = 2.0;
min_particle_size = 100;
        morph_size = 3;

pre_processed_image_name = preProcessImage(image_name, blur_sigma);

    segmented_image_name = segmentImage(pre_processed_image);

         clean_image_name = applyMorphology(segmented_image_name, morph_size);

analyzeSegmentedImage(clean_image_name, min_particle_size);
```

From our analyze particles example, if we take the minimum circularity and put it inside a variable, this is what it would look like. Now, hopefully you see the potential of functions. as a way to simplify the logic bits of your code. The things that actually do things are in functions and we can write functions one after the another, in a linear way, which tends to be more intuitive to read. Consider this example. We've set variables that we're going to use for analysis workflow at the top. So they're all bundled together. And we do not need to know the specifics of preProcessImage, segmentImage, applyMorphology and analyzeSegmentedImage in order to undetstand what each function is supposed to do. The code that actually does things will be written at the bottom of the script.

Notes

Summary

11m 06s

Conclusion

- Functions are logical units that perform a task
- Can create custom functions
- Many functions in ImageJ macro language
- Careful when recording
- String Surgery

Before finishing, let's recap. We've seen how we should think of functions as logical blocks that perform a specific operation. They are reusable and are the core of every programming language. We saw how we can create our own custom functions in order to increase the functionality of our code We covered where you can find all the built-in functions of the imageJ macro language. And, we went through how they are written and should be used. We noted that when using the macro recorder, some default values tend to be missing, and that we should be careful about that, when recording your macros. Then, we saw how to perform string surgery, and append variables inside strings for the run function. Thank you for your time and good-bye!

Notes

Summary

11m 57s