**Introduction**

* All the exercises in this worksheet are designed to show different aspects of macro design. Most will go towards building a single, final macro for segmenting cell nuclei.
* If you haven’t already done so, please download the “Example data” folder from the course GitHub repository (<https://tinyurl.com/4zyhbz6a>).
* You’ll also find a folder in the repository called “Worksheet macros” which contains solutions to all the exercises in this worksheet (<https://tinyurl.com/c5fvzz6m>).

**Exercise 1: Using the Macro recorder**

*Background:*

In their most basic form, macros can be used to run a series of steps in ImageJ/Fiji without needing to click through menus. Here, we’re going to automate the nuclei segmentation steps from the ImageJ/Fiji Level 1 course.

*Aims:*

* With the macro recorder running, process the image using several functions (see details below)
* Run macro to process image stack automatically

*Helpful tips:*

* Manual pre-processing of image stack
  + Load “Nuclei\_Stack\_1.tif” from the “Example data” folder
  + Load image into ImageJ/Fiji
* Run Macro Recorder
  + Plugins🡪Macro🡪Record
* Process image stack (to learn more about why these steps were chosen, please refer to our [ImageJ/Fiji Level 1 course](https://github.com/wbif-bristol/ImageJ-Fiji-Level-1-course)):
  + Z-Project
    - Image🡪Stacks🡪 Z Project🡪Max Intensity
  + Rolling-ball background subtraction
    - Process🡪Subtract Background
      * Rolling ball radius = 20px
      * (Everything else unticked)
  + Filter
    - Process🡪Filters🡪Median
      * Radius = 2px
  + Threshold
    - Image🡪Adjust🡪Auto Threshold
      * Method = Otsu
      * (Everything else unticked)
    - At this point you should see black nuclei on white background, if you see the opposite go to Edit🡪Invert
  + Fill holes in the binarised objects (else the next step will over-split the nuclei)
    - Process🡪Binary🡪Fill Holes
  + Separate touching objects
    - Process🡪Binary🡪Watershed
  + Set the measurements to make
    - Analyze🡪Set Measurements
      * Tick “Area”, “Centroid” and “Display label” (this will show the filename next to each measurement)
  + Analyse objects
    - Analyze🡪Analyze Particles
      * Size (micron^2) = 50-500
      * Show = Count Masks
      * Tick Display results
      * (Everything else unticked)
  + Change lookup table
    - Image🡪Lookup Table🡪Glasbey
* Click ‘create’ in macro recorder
  + A new macro script should be created in a new Script Editor window.
  + Save this to file
* Close all images apart from the original z-stack
* Click ‘run’ in Script Editor window
  + Should repeat all your manual processing…
  + You’ll end up with a Results Table showing the ID number for that nucleus, the filename of the image and the area and centroid measurements.

**Exercise 2: Batch processing a macro on multiple images**

*Background:*

We now have a working macro that can be run on a single image stack via the Script Editor window. In this exercise we’ll go further and process a whole folder of images using the same macro in a single run.

*Aims:*

* Run the macro from Exercise 1 on multiple different image stacks (stored in a folder)
* End up with a separate folder with the output images and a Results Table showing the area measurements for all nuclei for all processed images

*Helpful tips:*

* Launch the batch processor tool from Process🡪Batch🡪Macro
  + Set “Input” to the “Batch examples input” folder in the “Example data” folder.
  + Create an empty folder in the “Example data” folder called “Batch examples output” and select this by clicking “Output”.
    - The active image at the end of each macro run will be automatically saved to this folder
  + Click “Open” below the text editor window and select your macro from Exercise 1.
  + Click “Process”
    - Within seconds the Results Table should appear, showing the results from all three images. The “Batch examples output” folder will also contain the count masks image for each input image.

**Exercise 3: Display a phrase in the log window**

*Background:*

Now we can run basic ImageJ/Fiji commands we’ll start looking at how we can add functionality to our macro. This exercise will introduce using functions to perform specific operations.

*Aims:*

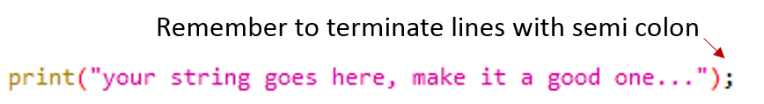
* Here we will create a macro which makes use of two important concepts in programming:

1. A function
2. A string

* The aim of this exercise is to print a simple message to the log window

*Syntax:*

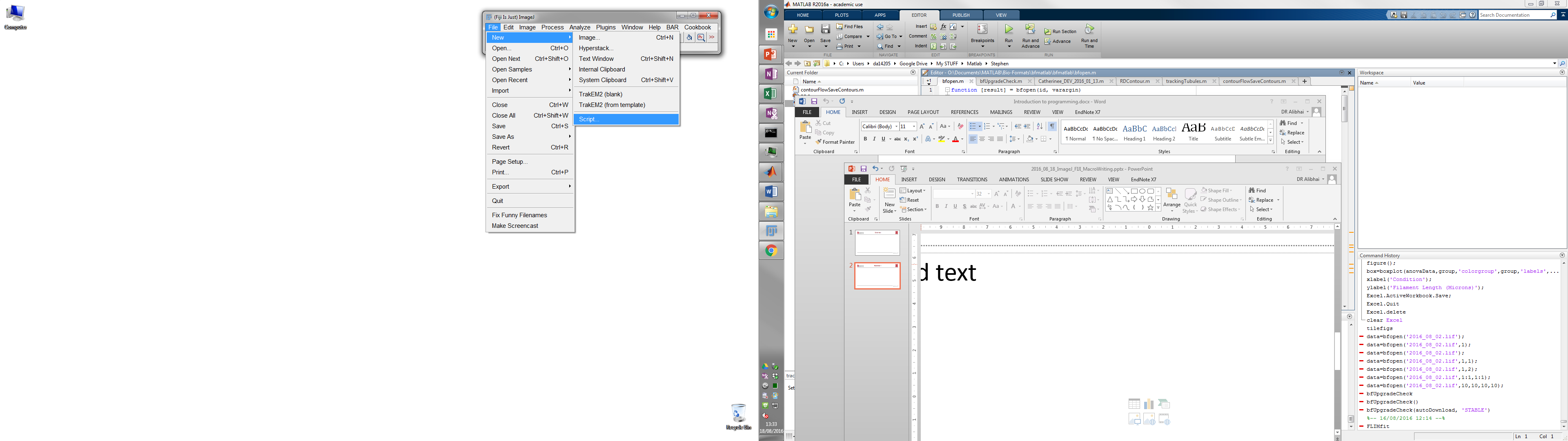
* The print function has the following syntax:



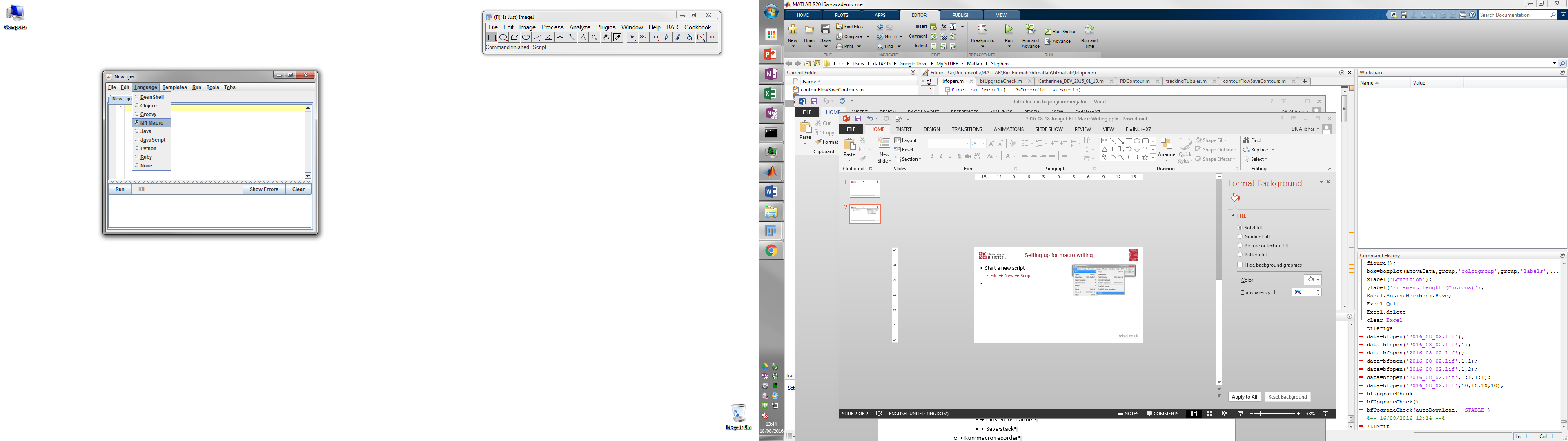
* Strings are identified using speech marks, you will notice the text turns pink once the programme recognises the words as a string (this will only happen upon closing the speech marks):
  + “This is a string”
  + This is not a string

*Helpful tips:*

* Start a new script by going to File🡪New🡪Script



* Change the programming language to ImageJ macro by going to Language🡪IJ1 Macro



* In line 1 of the script window insert a print() function
* Within the brackets of the function insert a string you wish to print to the log window
* Don’t forget to terminate the function using a semicolon!
* Click Run

**Exercise 4: Determine if a number is odd or even**

*Background:*

This exercise introduces the use of more complex programming concepts which will be used in almost all macros:

1. A loop
2. If/else statements
3. Printing different statements depending on the answers to If/else statements

*Aims:*

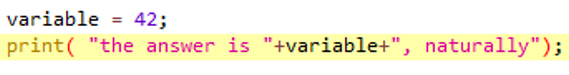
* Your macro should eventually print a list of numbers and determine if they are even or odd
  + First, we’ll create a for loop to list the numbers 1 to 10 in the log window
  + Then we’ll add some conditional statements to determine if the number is odd/even
  + Finally, well print a phrase which will say if the number is odd/even

*Syntax:*

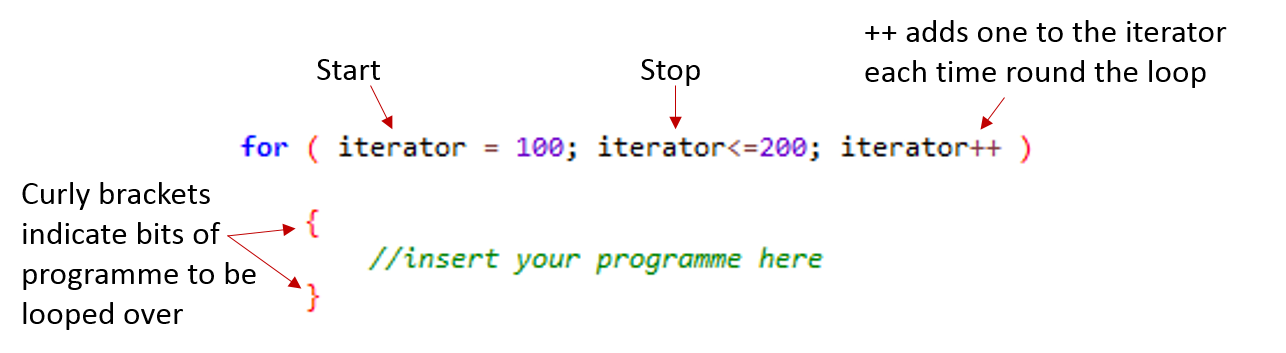
* Linking strings together can be achieved with the ‘+’ function:



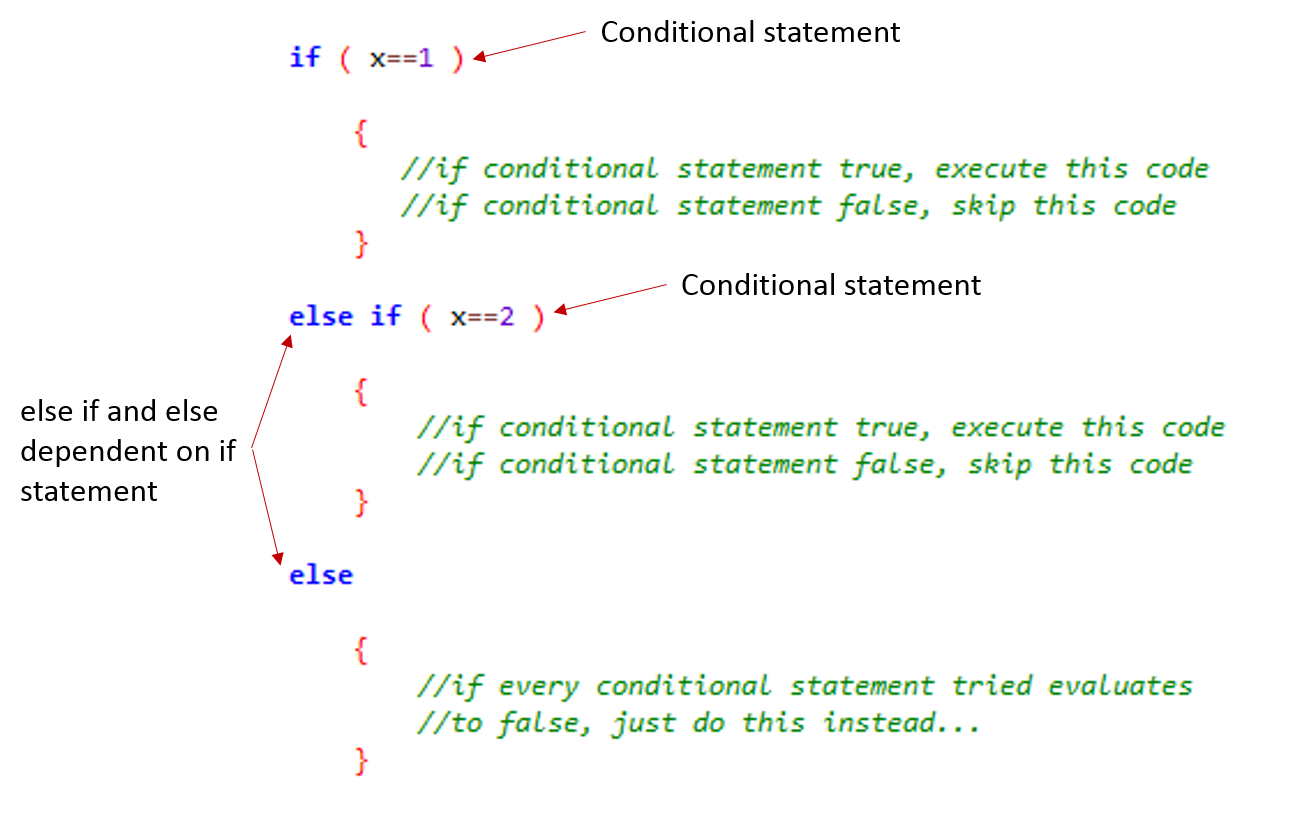
* This also works for linking strings to the contents of variables:



* The operator % can be used to give the *remainder* of a division
  + 2%2 = 0 (2 divided by 2 = 1 i.e. no remainder)
  + 3%2= 1 (3 divided by 2 =1.5 i.e. not a whole division)
* The syntax of the “for-loops” is as follows:



* The syntax of the “conditional statements” is as follows:



*Helpful tips:*

* Start a new script File🡪New
* Start by creating a loop which runs through the numbers 1 to 10:
  + Set up a for loop
  + Pick an iterator
    - This can be anything, typically we use ‘i’, but anything will do!
    - The iterator picked is then a variable assigned the start value and changes by 1 each loop.
    - Add a print function to display the iteration number
* Add in conditional statements (if/else if) to decide if the number is even or odd
  + Remember the % operator – although other methods also work…
* Add in a print statement to display a phrase indicating if the current iteration is even or odd

**Exercise 5: Calculating the distance between two objects**

*Background:*

From Exercise 1 we know we’re able to detect nuclei and measure their properties (such as area and centroid location) and from Exercises 3 and 4 we’ve seen how the macro language can be used to perform calculations. In this exercise we’ll combine these to extract the existing centroid measurements and calculate a new value – the distance between nuclei.

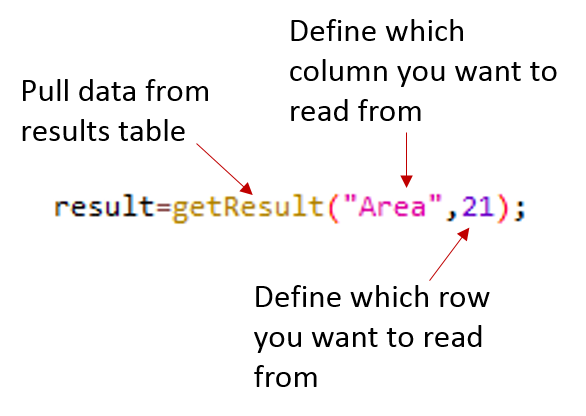
For this first step, we’ll use an existing Results Table containing measurements for just 2 nuclei – this will allow us to test our code in a more controlled way.

*Aims:*

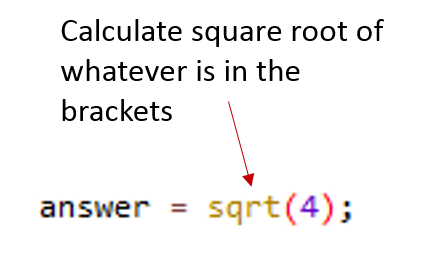
* Read in coordinate information from results table file (“Two nuclei centroids.csv” in “Example data” folder) and calculate straight line distance between two nuclei.
* Print the calculated distance to the log window.

*Syntax:*

* To get a result stored in the results table and store it in a variable:



* To calculate the square root of a number or variable:



*Helpful tips:*

* Drag and drop the “Two nuclei centroids.csv” file from the “Example data” folder into ImageJ/Fiji. You should see a new Results table open with 2 rows (one for each of the previously-detected nuclei) and 2 columns (“X” and “Y”). You’ll need to rename this to “Results” (by going to “File > Rename” in the table menu) for some of the macro functions (e.g. nResults()) to work.
* Open a new scripting window
* Get the x and y coordinates of nuclei 1 (remembering macros use zero based indexing…)
* Get the x and y coordinates of nuclei 2
* Calculate the distance between these two points using:
* The pow() function can be used to square a number (e.g. pow(3,2) is the same as 32)

**Exercise 6: Calculating the distance to the nearest nucleus**

*Background:*

Next, we’re going to perform the same nucleus-nucleus distance measurement, but measuring the distance from one nucleus to many different nuclei. By keeping track of the smallest value we measure, we end up with the “nearest neighbour” distance.

As with Exercise 5, we’ll use some pre-prepared nuclei coordinates for simplicity.

For now, we’re just going to measure the distance from the first nucleus in the list to all other nuclei. In Exercise 7 we’ll extend this further to measure the nearest neighbour distance for all nuclei-nuclei pairs.

*Aims:*

* Create a macro to measure the nearest neighbour distance for the first nucleus in the “Many nuclei centroids.csv” file in the “Example data” folder.
* Print the calculated distance to the log window.

*Helpful tips:*

* You’ll need to use a for-loop to iterate over all the nuclei in the results table (i.e. iterate over each row)
* Use a variable to keep a record of the shortest distance you measure. This is the “nearest neighbour” distance that we’ll print to the log window at the end.
* If you end up with a nearest neighbour distance of “0” you might be measuring the distance of the first nucleus to itself.
* Remember to close any open Results tables before each time you run the macro, else these coordinates will be included in the nearest neighbour calculation.

**Exercise 7: Calculating the nearest neighbour distance for all nuclei**

*Background:*

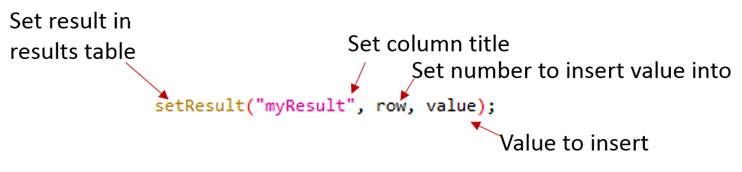
Now we’re going to extend our code from Exercises 5 and 6 to measure the nearest neighbour distance for all nuclei-nuclei pairs. This largely uses the same code from Exercise 6; however, we’ll be using nested loops (one loop inside the other) to allow us to test all combinations. We’ll also be storing the nearest neighbour distance for each nucleus in the results table.

*Aims:*

* Create a macro to measure the nearest neighbour distance for each nucleus in the “Many nuclei centroids.csv” file in the “Example data” folder.
* Update the results table to show the calculated nearest neighbour distance in the results table and save table as .csv file.

*Helpful tips:*

* You’ll need to use a pair of for-loops nested so one is inside the other. So the iterators don’t clash you’ll need to call them different things (e.g. “i" and “j”). By nesting one inside the other the macro will eventually access every combination of nucleus-nucleus pairs.
* Use the following command to add the nearest neighbour distance to the results table:



**Exercise 8: Working with simple dialog boxes**

*Background:*

If passing a macro on to someone else, it can be easier to indicate which parameters can be changed using dialog boxes. These appear when the macro runs and allows the user to enter values which can be used later in the macro.

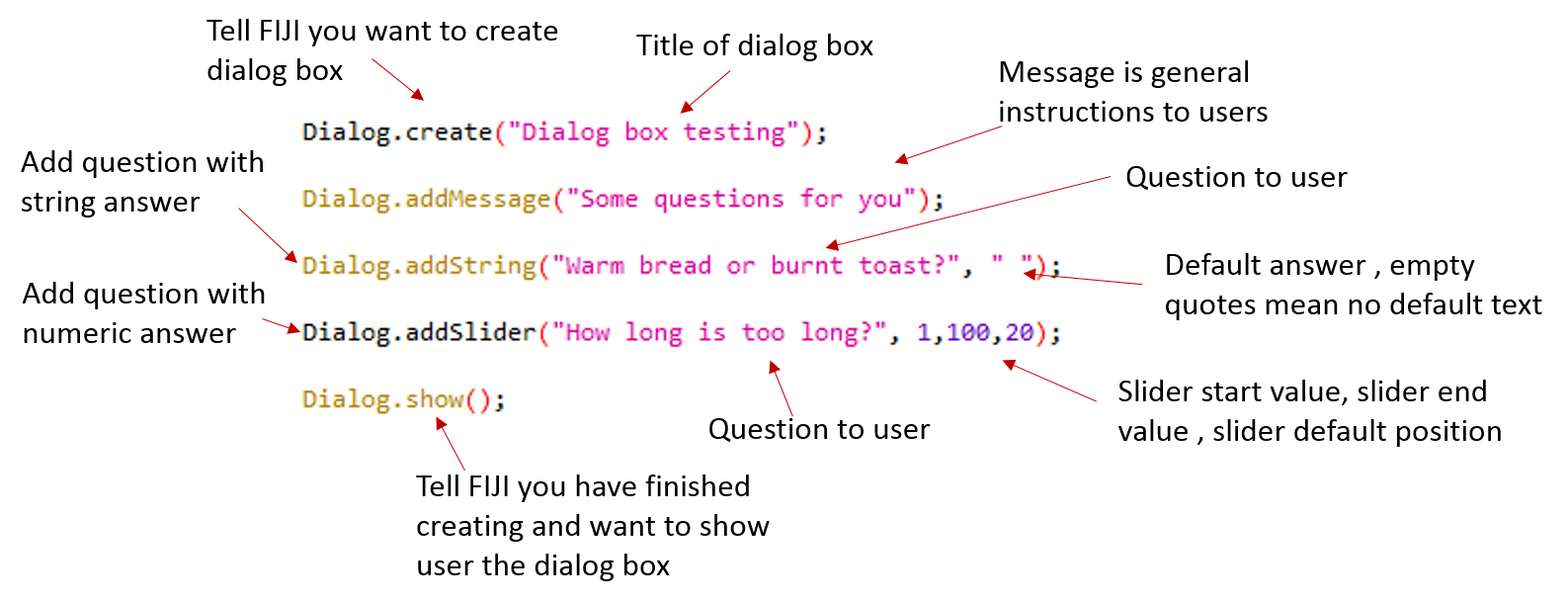
Note: If you’re planning to batch process the macro it’s best to avoid using dialog boxes as they would appear for every image in the folder.

*Aims:*

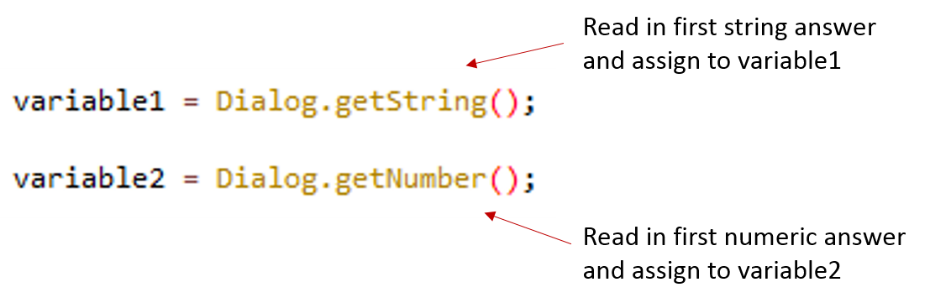
* This exercise introduces the use of dialog boxes enabling the user to input variables to the programme (e.g. threshold value, size sieve etc.).
  + The aim of this exercise is to create a dialog box which will pose several questions to the user and allow them to input data to the programme
  + Create at least two questions (one string answer and one numerical answer) For examples:
    - Question 1: “What day of the week is it?”
    - Question 2: “What date is it?”
    - Question 3: “What month is it?”
  + Read these values in and assign to variables
  + Display a concatenated answer in the log window, for the questions above a concatenated output should read:
    - Wednesday 14th September (although you can add any extra text you like too)

*Syntax:*

* The syntax needed for creating dialog boxes is as follows:

****

* Creating a dialog box using these options will allow the user to input their own values into your programme
* However, we need to add some more code to “read in” these values and assign them to variables.
* The syntax for reading in the user inputted values is as follows:

****

*Helpful tips:*

* Start a new script: File🡪New
* Create a dialog box and give it a title of your choice
* Add in a few Options for user inputs
  + A mixture of strings and numerical values would be good
* Use the Dialog.show function to finish the dialog options.
* Add the necessary lines to read in data from your dialog box
  + Remember these will have to be different depending on if the answers to the questions are numerical or string
    - The order of the read in statements will have to be the same as the order the questions were posed
  + Assign each answer to a variable
* Print the output to the log window
  + e.g. if you’re wanting to display todays date then you should be printing something like:
    - “The date today is Wednesday 14th September”

**Exercise 9: Getting serious…adding GUI elements to your image segmentation macro.**

*Background:*

We’ll now use what we’ve learned about dialog boxes to add some user-controllable options to the start of our macro. Here, we’re going to add a couple of dialog box options to our macro from Exercise 1.

*Aims:*

* Take the code from Exercise 1 and add GUI options
* Add option to change threshold level
* Add option to change filter type

*Syntax:*

* The basic functions for working with dialog boxes are shown below:



*Helpful tips:*

* Start by copying image analysis code from Exercise 1 into a new macro script
* Identify the parts of the script that you want to make variables
  + Do they need to be of type string or numeric?
* Create a dialog box above your image analysis code
  + Insert a useful message telling the user what to do
  + Insert relevant questions to change the threshold and filter type
    - Changing filter type requires the creation of an array of pre-determined filters
  + Remember to insert the Dialog.show() statement when you’ve finished defining your dialog box
* Insert the relevant statements to read in the required information
  + One will need to be able to read in information from your array of choices
    - The necessary function to read in from an array is: Dialog.getChoice()
  + The other will need to be able to read in information in a numeric format.
  + The order these appear will need to match the order you asked the questions.
  + Assign this information to variables
  + Switch in these variable names into your image analysis code in place of the threshold value and filter type.

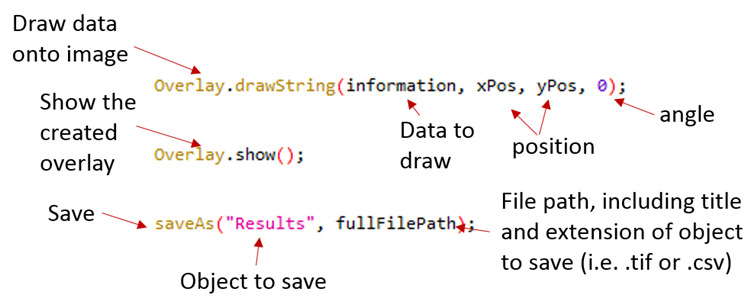
**Exercise 10: Putting everything together**

*Aims:*

* Create a macro to segment and measure the nearest neighbour distance for each nucleus in “Example\_Data.tif”
* Include a GUI for the user to input options (threshold, filter type, filter size etc.)
* Update the results table to show the calculated nearest neighbour distance in the results table and save table as .csv file.
* Display the nearest neighbour distance on the segmented image

*Syntax:*

* Some useful functions for this exercise can be found below:



*Helpful tips:*

* A good start is to take the result of Exercise 9 as a starting point
* Add in the nearest neighbour measurement from Exercise 7, but now the results table should appear straight away after running the Analyze Particles command (rather than being loaded from .csv file).
* Add the nearest neighbour distance to the results table
* Convert the scaled CY nuclei coordinates to pixel values using the toUnscaled(x,y) function.