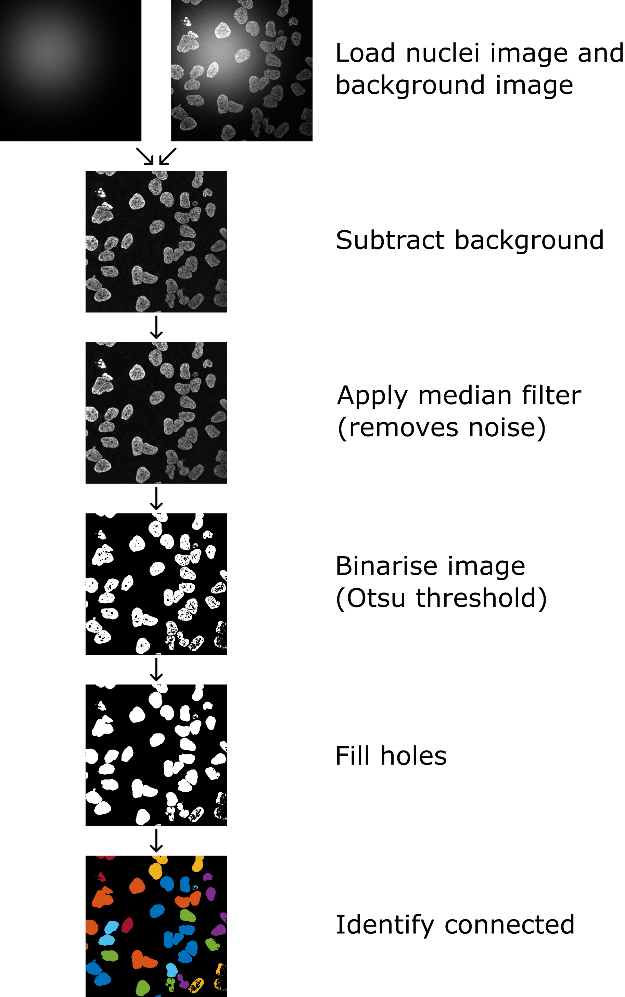
**MATLAB for image processing: Session 2 worksheet**

****In this worksheet we’re going to use what you’ve learnt about arrays, matrices and images to implement a basic image segmentation and analysis workflow. This workflow involves loading an image of cell nuclei, subtracting a background intensity profile (provided from a reference image), then binarizing the image and identifying connected regions. The image to the right shows an outline of the steps you’ll be taking.

* Load nuclei image. Load background image.
* Subtract background from nuclei image
* Apply 2D median filter (2px radius) to remove noise
* Use graythresh function to calculate Otsu threshold for the image. Is the calculated threshold in the correct range for the image? i.e. what is the data type for the image, what range do the values for this type lie in, is this the same range as the threshold value?
* Use loops to apply threshold to the image to create a binary image (i.e. a logical array). Do this with and without initialisation of the logical array and use tic toc to compare the time taken to do this operation.
* Re-run the binarisation, but doing it in using array operations (i.e. apply threshold to all pixels at once). Use tic toc to measure the processing time. Note: In normal usage you could use the im2bw or imbinarise functions; however, here, we’re trying to learn array operations, so you’re not allowed to use im2bw or imbinarise!
* Fill holes (find function to do this).
* [Optional] Watershed. Note: This is a trickier exercise and isn’t pivotal to completing the remaining exercises. If you get stuck here, don’t worry, you can skip this step.
* Use bwlabel to apply connected components labelling to the image. [describe connected components]. Store the labelled image as a new output and display as a figure (i.e. using imshow). Before displaying the image, you may want to play around with label2rgb to shuffle the colours of the labelled image.
* Save the labelled image to an image file.
* [Extension exercise 1] Measure the number of pixels in each labelled object. Hint: For this, you may want to (1) identify the number of unique labels (pixel intensities) in the image, (2) for each unique label, use a logical array operation to identify the pixels with that value, sum the number of true (1) pixels.

In this worksheet we’ll implement the concepts covered in the slides to prepare a simple script, which generates random numbers (double-precision values in the range 0-100), round them to integers, then display a message on whether they’re odd or even. Don’t worry if this problem seems a bit pointless; the key goal here is to familiarise yourself with the basics of MATLAB and practise searching for and implementing solutions. It’s worth remembering that there are often many completely valid ways to solve a problem in MATLAB, so if you try one and get stuck maybe go back to Google and see if you can find an alternative.

1. **Creating a new script**

First, we want to create the script file we’ll work from. For each exercise it’s probably a good idea to make a copy of this script.

1. In the “Home” tab along the top select “New script”.
2. Save this script by going to the “Editor” tab and selecting “Save”.
3. **Running the script file**

We now want to make sure we can run this script. In order to do this, we’ll make the script display a simple message, then call the script from MATLAB’s command window.

1. Add the following code to the script. You can of course change the message if you like!

*disp('Hello from an M-file!');*

1. In the command window, type the name of your scriptand press “Enter”.

The message you typed should appear on the next line. If it doesn’t, check that the script is saved in a location accessible by MATLAB: either the active directory (path shown in the navigation bar), or one of the locations shown when you type *path*.

1. **Generating random numbers**

In this exercise you’ll need to do a bit of research, as I won’t be telling you the functions you need to use. The aim here is to generate a random number in the range 0-100 and assign it a variable reference. You should then update your *disp* command from the previous exercise to display the random number. Google (or equivalent) is typically a good place to start when finding a function for a job.

1. Find a function to generate a random number and assign the returned value to a reference. This will allow us to access this number later on in the script.
2. It may be that the function you’ve used doesn’t have the option to output in a specific range (for example, it may output in the range 0-1). If this is the case, modify the output so it does lie in the range 0-100.
3. Use the *disp* function (or another way if you like) to show the random number when the script is run.
4. Run the script a few times to check you’re getting different random numbers in the range 0-100.
5. **Converting between data types**

At the moment, your random numbers will likely be double precision, but as we’re going to be determining if they’re odd or even we need them to be integers. As before, you’ll need to do some research to determine the best functions for the job.

1. Assuming you’ve run your script at least once, the variable you assigned should be in the workspace (if it is, you’ll see it listed in the “Workspace” panel). Use the *whos* command to see what type your variable is. Chances are it’ll be double, but if it’s already an integer, congratulations, you can skip to the next exercise!
2. In the script add a line to convert the double-precision random number to an integer and store this integer value as a new variable.

Note: While you can just cast the double to an integer, and this appears to round correctly we can’t be certain it works 100%. Instead, it’s good practise to first use an explicit rounding function.

1. Add another command to display the integer value in the command window when the script is run. This should be in addition to the previous *disp*, so we can see the original double value and the new integer value to check it’s working correctly.
2. **Handling text**

So far, we’ve just been displaying numbers to the command window on separate lines; however, this isn’t particularly neat. Instead, it would be better if we could have a single line with a message such as “Double value = 43.1, integer value = 43”. In this exercise you’ll need to combine text and numbers into a single piece of text.

1. Add the relevant code to display a message to the command window, which states the double and integer value numbers. This message should be a combination of explicitly-typed text and the generated random numbers.
2. Run the script a few times to check this is working correctly.
3. **Creating function files**

The aim here is to create a function file which performs the simple operation of determining if our random integer is odd or even. This function will take a single argument (the random number) and output a logical (1 if odd and 0 if even). Yes, we could have just added this code to the script, but we need to practise writing functions too!

1. In the editor, create a new script file.
2. Define a function which accepts a single number as an argument and returns a single value (the logical of if the number is odd). You can give the function any name you want, but it’s best to make it sensible (e.g. “isodd”).
3. Add the necessary code to the function
4. Save the function to a file to a location accessible by MATLAB. The name must be exactly the same as the name you gave the function (so in my example I’d get a file call “isodd.m”).
5. Implement the function in your script and add the output (1 or 0) to your message. As such, you message will now be something like “Double value = 43.1, integer value = 43, is odd = 1”.
6. Run the script a few times to check it works correctly.
7. **Conditional statements**

The current message we get is functional, but the last bit is a bit clunky. It would be better to have it clearly state if the value is odd or even. To do this, we’ll use conditional statements. If the number is even, the conditional statement will print one message, while if it’s odd it’ll print a different message. An example message would be “Double value = 43.1, integer value = 43, odd number”.

1. Use a conditional statement to add text to the message from the previous exercise saying if the number is odd or even.
2. Run the script from the command window a few times to check it’s working correctly.
3. **While loops**

Now we will make the script generate a series of random numbers until we get an even integer. For this, we can use while loops, which continue to execute the code until an even number is generated.

Note: The next exercise will replace this while loop with a for loop, so it’s a good idea to make a copy of the script at this point.

1. Surround the code (or as much as is necessary) with a while loop, so the code continually executes until an even number is generated. On each iteration, display the message from the previous exercise.
2. Run the script from the command window a few times to check it’s working correctly. Since you may get an even number immediately it’s important to run it multiple times.
3. For loops

Finally, we will update the script to replace the while loop with a for-loop. Unlike the while loop, a for loop will execute a specific number of times. We will run the random number generator and odd/even test 10 times.

* Starting with the code from exercise 7 (i.e. prior to the while loop) surround the code in a for-loop structure. Set the for loop to run 10 times.
* Update the message, so it includes the current iteration number (for example, “Iteration 4, double value = 43.1, integer value = 43, odd number”).
* Run the script from the command window to check it’s working correctly.