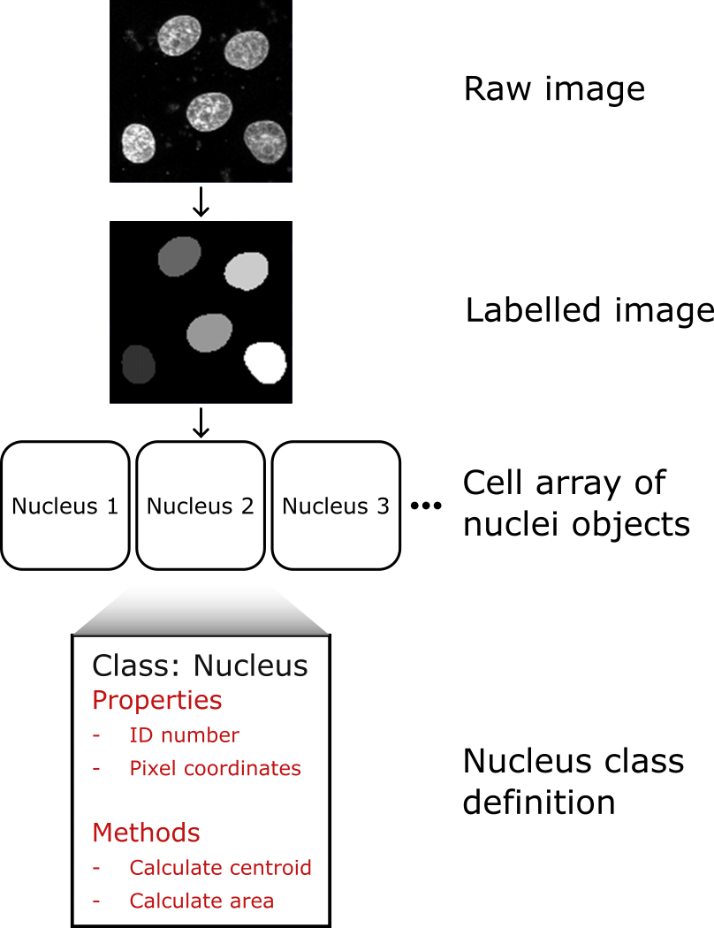
**MATLAB for image processing: Session 3 worksheet**

This session’s worksheet is a continuation of the workflow for segmenting nuclei we worked on in Session 2. At the end of the previous worksheet we had a labelled image, where the all the pixels corresponding to a specific nucleus had the same value (effectively that nucleus’ ID number). This is a functional way to store the nuclei, but any time we want to make measurements for a nucleus, we must first identify which pixels correspond to it. It would be more efficient if we used a data structure that allowed us to keep a numeric array of pixel coordinates corresponding to each nucleus. Any of the mixed data-type structures covered in the slides (structure arrays, cell arrays or tables) can do this, but we can go one better by using an object-oriented programming (OOP) approach. Using an OOP approach allows us to also assign useful functions (e.g. measuring the area or centroid location) to the coordinate stores. In this worksheet, we will build one such OOP model. We will then iterate over each nucleus in the labelled image, create a Nucleus object instance and add the object to a 1D cell array.

1. **Getting the labelled nuclei image**

This first exercise is a repeat of the final main exercise from the Session 2 worksheet. It will take us from a raw image of nuclei to a labelled image. To ensure everyone’s starting with the same materials, the code to do these steps is provided.

Note: For the exercises in this worksheet we’ll use a smaller image, which only contains 5 nuclei. This will make checking results easier.

1. If you haven’t already done so, download the “NucleiImage\_small.tif” image from the Session 3 GitHub repository (<https://github.com/SJCross/MATLAB-course>).
2. Create a new script file and save this to a location accessible to MATLAB.
3. Paste the following code into the new script file

% Clearing the workspace

clear

% Loading the nuclei image

nuc\_im = imread('NucleiImage\_small.tif');

% Applying a 2D median filter

filt\_im = medfilt2(nuc\_im, [5, 5]);

% Calculating and applying threshold

thresh = graythresh(filt\_im)\*255;

log\_im = filt\_im > thresh;

% Filling holes in the binarisation

fill\_im = imfill(log\_im,'holes');

% Creating labelled image of nuclei

label\_im = bwlabel(fill\_im);

% Displaying the labelled image

imshow(label\_im,[]);

1. Run the script from the command window to ensure everything is working correctly. You should see the labelled image as it appears in the figure at the top of this worksheet.
2. **Getting pixel coordinates for a single nucleus**

Here, we will establish a method to create a 2D numeric array containing the coordinates of all pixels corresponding to a specific nucleus. In subsequent exercises, this array will be the coordinate store for the Nucleus class objects. The array should have size Nx2, where N is the number of pixels in that nucleus. The first column of the array should be for the pixel row coordinates and the second for the pixel column coordinates. One way to achieve this is to use the *find* function, which returns the row and column coordinates corresponding to a logical check on an array.

Hint: If you get stuck implementing the *find* function, an example is hidden between the following quotation marks. When pasted into MATLAB the code should appear.

Code: “[rows, cols] = find(label\_im == 2)”

1. For the nucleus number 4 (i.e. pixel values all equal to 4) use the *find* function to get the row and column indices of all pixels. The output from this function should be two Nx1 arrays, where N is the number of pixels in that nucleus.
2. Use array concatenation (covered in Session 2) to combine these 1D arrays to the final Nx2 array.
3. Check the size of the array. It should be 645x2.
4. **Defining an object class**

We will now start setting up our nucleus class definition. This definition specifies what properties objects of that class will have as well as how any methods should operate. For this exercise, we’ll only include the constructor method (others will be added in subsequent exercises).

1. Create a new script file and save it to a location accessible to MATLAB with the name “nucleus.m”. This will be our class file.
2. Set up the class file to include properties for the nucleus ID and the 2D array of pixel coordinates. It doesn’t matter what you call these, but it’s probably sensible to use simple names like “ID” and “pixels”.
3. Add a constructor method which allows the user to provide the ID and the 2D array of pixel coordinates. The constructor method needs to return “obj” (see the example in the Session 3 slides).
4. Using the 2D pixel array you obtained for nucleus 4, initialise a new instance of the nucleus object. If you omit the semicolon at the end of the line, you should get the message:

nucleus with properties:

ID: 4

pixels: [645x2 double]

1. **Adding methods to class definitions (nucleus area)**

Now we have our base class definition, we can start adding more functionality in the form of methods. The first method we’ll add is to calculate the area of each nucleus.

1. In the class definition file, add a new method called “calculateArea”. This only needs to take “obj” as an argument (as do all non-constructor methods). It should return a single value which is the count of pixels in the object.
2. Run this new method for your existing nucleus 4 object. It should return 645.
3. **Adding methods to class definitions (nucleus centroid)**

As in exercise 4, we’re going to add more functionality to our nucleus class definition. This time, a method which calculates the centroid of the nucleus. The “centroid” we’ll use is the mean row and column for the pixel coordinates. This should be returned from the method as a 2x1 numeric array (i.e. [mean\_row, mean\_col]).

Hint: The *mean* function was covered as an exercise during Session 3 when discussing tables.

1. In the class definition file, add a new method called “calculateCentroid”. As before, this only needs “obj” as an argument. It should return a vector in the form [mean\_row, mean\_col].
2. Run this new method for your existing nucleus 4 object. It should return [35.4930, 96.7860].
3. **Adding objects to a cell array**

Our class definition is now complete, so the next thing to do is to process all nuclei in the test image. For each nucleus, we want to generate a nucleus object instance and add it to a cell array. Storing the objects in a cell array allows us to keep them all together and means we don’t need to assign distinct references to each one (e.g. obj1, obj2, obj3, etc.).

1. Use the unique function to determine how many different nuclei there are in our image. You can use the example solution to Exercise 9 from Session 2 as an example of this.
2. Initialise a cell array with enough cells to hold all the nuclei objects once they’re complete. This wasn’t covered in the slides, so you’ll need to research how to do it.
3. Loop over all nuclei IDs in the labelled image. For each one, obtain the pixel coordinates as a 2D array (code from exercise 2) and use this to create a nucleus object instance.
4. Store the nucleus object in the cell array.
5. **Using objects**

At this point we should have a 5-element cell array, where each cell of that array contains a nucleus object. To check our data is correct, we will loop over it, running the two methods (getArea and getCentroid).

1. Use a loop to access each nucleus stored in the cell array.
2. For each nucleus, run the getArea() and getCentroid() methods. Convert the output of each of these methods to text, so you can create a message that reads something like “Object (ID = 4), area = 645, centroid = (35.493, 96.786)”.
3. When you run the code, you will hopefully get the following lines appear in the command window.

Object (ID = 1), area = 519, centroid = (101.1349, 20.6435)

Object (ID = 2), area = 649, centroid = (25.1926, 49.2897)

Object (ID = 3), area = 680, centroid = (78.6868, 70.4515)

Object (ID = 4), area = 645, centroid = (35.493, 96.786)

Object (ID = 5), area = 669, centroid = (99.275, 109.9088)