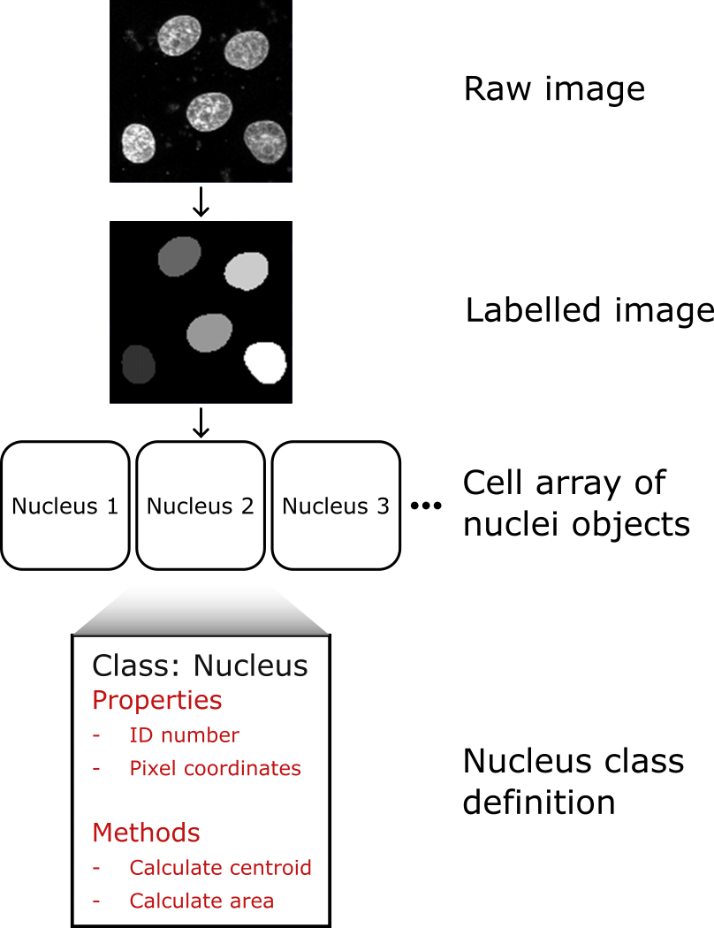
**MATLAB for image processing: Session 4 worksheet**

* Note: There are a pair of pre-prepared functions. The first for taking a pre-loaded pixel array and generating the labelled image; the second for creating a cell array of nuclei objects. [Write the inputs and outputs of these]. Also provided the Nucleus object class we prepared last week.
* Use uigetfile to get the path to the example image.
  + Display this file path in the command window to verify it’s working correctly
* For the selected image
  + Load the image to the workspace
  + Use the two provided functions ([refer to them by name]) to generate a cell array of nuclei objects.
  + Display the number of nuclei detected in that image in the command window
* Create a function which displays the loaded image, then iterate over all nuclei in the cell array, adding centroids to the image to show what was detected.
* Note that there are lots of little fragments that are being detected as nuclei. Note that this should be done immediately after the user has selected the folder – this means all options dialogs will appear one after the other (rather than having a pause).
  + Use inputdlg to get the user to enter a minimum nucleus area in px^2.
  + Display the entered value in the command window to check it’s working correctly.
* Apply the minimum size filter. This can be done many ways, so I won’t provide steps.
  + Create a second cell array which only contains those nuclei with areas larger than the specified threshold.
* Create a histogram showing the distribution of nuclei sizes in the image.
* Create a box and whisker plot showing the distribution of nuclei sizes in the image.

**MATLAB for image processing: Session 4 worksheet**

* Note: There are a pair of pre-prepared functions. The first for taking a pre-loaded pixel array and generating the labelled image; the second for creating a cell array of nuclei objects. [Write the inputs and outputs of these]. Also provided the Nucleus object class we prepared last week.
* Start by checking we can get the functions to work. For the example image ([“im\_001.tif”?])
  + Load the image to the workspace
  + Use the two provided functions ([refer to them by name]) to generate a cell array of nuclei objects.
  + Display the number of nuclei detected in that image in the command window
* Create a function which displays the loaded image, then iterate over all nuclei in the cell array, adding centroids to the image to show what was detected.
* Use uigetfile to get a list of files in a specific folder. [Mention that for this bit we don’t need to use the code from exercises 1 and 2. Exercises 1,2 and 3 will be merged as part of exercise 4 (or 5)].
  + Display this file list in the command window to verify it’s working correctly
* Write a loop to process all images in a folder
  + For each image, use the code from section
  + Recall from Session 3 that cell arrays can be nested (i.e. a cell can itself contain a cell array). Add the nuclei cell array to a cell array, where each cell contains the nuclei cell array for that image. The top-level cell array should also contain the filename.
* Iterate over all nuclei objects (in all images) and create a pair of cell arrays. One which contains the nuclear area measurements (you can use the getArea method of the nuclei objects for this); the other which contains the filename for the image in which that nucleus was detected.
  + Start by initialising the two cell arrays. For this you’ll need to determine how many nuclei there are in total.
* Create a box and whisker plot showing the nuclei areas, grouped by image.

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

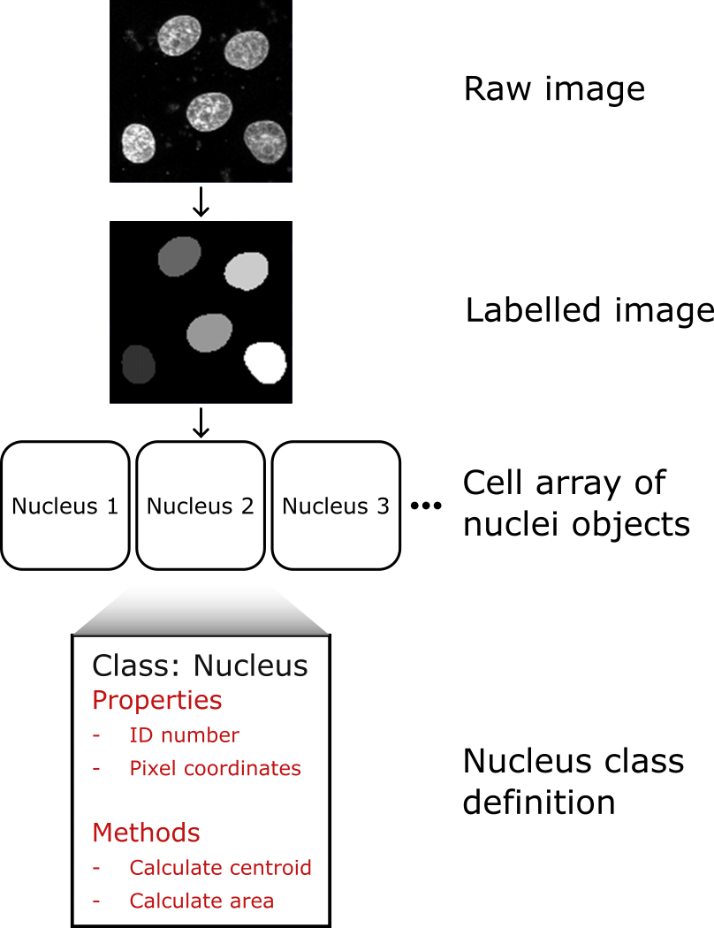
fill\_im = imfill(log\_im,'holes');

% Creating labelled image of nuclei

label\_im = bwlabel(fill\_im);

% Displaying the labelled image

imshow(label\_im,[]);

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,[]);