# ENGG1003 - Lab Week 10

Sarah Johnson

The purpose of this lab is to demonstrate some image processing steps which will be required for The second assignment.

## Task 1: Load an RGB image

1. Download the image of Grace Hopper from

   `https://www.womenshistory.org/sites/default/files/images/2018-07/Hopper-3%20square.jpg`

   and save it in the same directory as your python project. This image uses the RGB format we saw in the lecture in week 7.
2. Import `numpy, matplotlib.pyplot` and `matplotlib.image`
3. Load this Grace Hooper into a 3D numpy array using the `matplotlib.image` function `imread` (as per lecture in Thursday week 7)
4. What datatype is the numpy array? Use the console or the built in `type()` function to check this.

## Task 2: Change the datatype of an RGB image

Image data is typically stored as RGB values where the value each colour channel can take is scaled from 0 to 255. This range is chosen as it is the range of values for which an 8-bit unsigned integer (`np.uint8`) can take, resulting in efficient use of memory. This is a common standard as it represents a good compromise between storage size and output quality. Storage as other datatypes (eg: 16-bit integers or even 32-bit floats) can be done but filesizes are much larger for an imperceivable improvement in quality. Image editing, however, typically requires a higher level of precision. As such, editing is often performed on 16-bit data (so-called np.uint16 allowing for 65536 levels of grey. This datatype is chosen because professional grade DSLR cameras shooting "raw" photographs typically record 12 to 14-bit data. 32 and 64 bit datatypes can also be used.

1. Convert the numpy array to a new array with datatype 8-bit unsigned integer (0 to 255) with `np.uint8()`. Observe the values in this third array. The `unit8` function does truncating (i.e. 9.8 is converted to 9) so you could first use `np.round()` followed by `np.unit8()` for improved accuracy of the conversion.
2. Save the image as a jpg using the `matplotlib.pyplot` function `imwrite` (as per lecture in Thursday week 7). Note that `imwrite` also requires the image in RGB format to consist of integers between 0 and 255 or floats between 0 and 1.
3. What datatype is the numpy array? Use the console or the built in `type()` function to check this.

## Task 3: Clipping and Scaling

Using the 64bit array (let's call it a):

1. Find the maximum and minimum values in the image array using `a.max()` and `a.min` (where 'a' is the array name). You can also use `np.max(a)` or `np.min(a)`.
2. Make every pixel in the image redder by adding 50 to the R value in every pixel. (you could do this in a loop or using array slicing and += 50). Now find the minimum and maximum values again. The maximum value should be above 255.
3. View the new image by repeating `imshow` and remembering to convert to a uint8 format first. What do you notice?
4. One way to deal with the pixels that had red values greater than 255 after step 9 would be to

clip all the red pixels greater than this maximum. You can do this using loops or use array indexing such as:

```
a[a>255] = 255
```

Clip the array ‘a’ in this way before converting to uint8 and viewing using `imshow`. Is the image any better?

5. Another way to deal with pixels that had red values greater than 255 would be to scale all pixels equally so that the minimum value is 0 maximum pixel value is 255. You will need to find the minimum value in the array and subtract that from all entries. Then find the maximum value in the array, and divide all entries by this value and multiply by 255. Is there a difference in how the image appears?
6. Repeat the steps above using toy image so that you can see how the values in the array are different when clipping as compared to scaling. e.g.

```
img = np.array( [ [ [255, 255, 255], [0, 0, 0] ], [ [255, 0, 0],
[255,192,203] ] ] )
```

## Task 4: Convert the image from RGB to HSL and vice versa using the python colorsys module

The HSL colourspace specifies a colour via a hue, saturation, and luminance. The hue is a ”base” colour from the rainbow, with the extension that variations of magenta exist between blue and red. Hue is typically specified as an angle in degrees around an imaginary “colour wheel”. For this project hue will be a number from 0 to 360. The saturation figure specifies how “colourful” the colour is. A minimum saturation is a shade of grey while a maximum saturation specifies a pure colour (ie: a single wavelength of light). For this project saturation will be a number between 0 and 1. Lastly, the luminance figure is a brightness: minimum luminance is black, maximum luminance is white. Note that if the luminance is minimum or maximum the hue and saturation figures become irrelevant. For this project luminance will be a number between 0 and 1. You can experiment with how the RGB and HSL colourspaces work by playing with this colour picker:

`https://www.w3schools.com/colors/colors_picker.asp?colorhex=E8B5CE`

We will use the colorsys module. The functions you will need from the colorsys model are

```
colorsys.hls_to_rgb(h, l, s)
```

```
colorsys.rgb_to_hls(r, g, b)
```

where r g b are the red, green and blue intensity values of the pixel and h, l, s are the hue, luminance and saturation of the pixel. Note that the colorsys module is written to accept scalars as inputs not numpy arrays so you will need to call it in a loop when converting an image. The output of `rgb_to_hls` is a list of 3 numbers not a numpy array.

1. Import the colorsys module.
2. Use the colorsys module to convert the RGB white (255, 255, 255), black (0, 0, 0), pink (255, 192, 203) and blue (0, 0, 255) to HLS.
3. Using loops write python code to convert a 3D RGB array into a 3D HLS array.
4. Test this conversion on a toy 4 by 4 by 3 RGB array you have created.