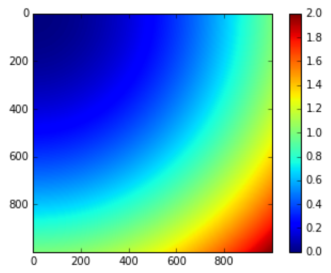
Plotting 2D functions using numpy

We use `numpy.ogrid` to create a 2-dimensional grid as the input set for a 2D function. In numpy, the letter `j` is used to indicate the imaginary unit $i = \sqrt{-1}$. In `numpy.ogrid`, it changes the way that an equispaced list is created. For example, compare the following inputs, and pay special attention to the first and last elements of the arrays.

```
import numpy as np
np.ogrid[0:10]
np.ogrid[0:10:.5]
np.ogrid[0:10:20j]
np.ogrid[0:1:1000j,0:1:1000j]
```

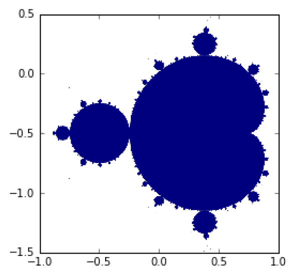
We plot our first 2D function, using the `imshow()` function in `matplotlib.pyplot`. In this case the function takes a 2D array of real numbers, and maps them to a color function. We can ask pyplot to add a colorbar to the image, displaying the mapping function.

```
import matplotlib.pyplot as plt
x,y=np.ogrid[0:1:1000j,0:1:1000j]
z=x^2+y^2
plt.imshow(z)
plt.colorbar()
```



As another example of a 2D funtion, we plot the Mandelbrot set. For an explanation of the definition of this fractal, see `mathworld.wolfram.com/MandelbrotSet.html`, although this is by no means part of the course material.

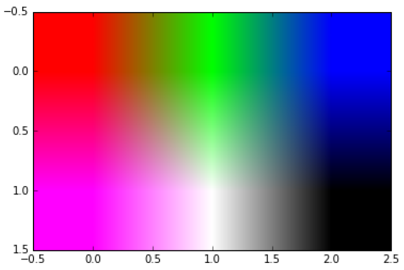
```
a,b=np.ogrid[-1:1:1000j,-1.5:.5:1000j]
c=a*1j+b
z=0
for i in range(1000):
   z=z**2+c
plt.imshow(abs(z))
```



Imshow() and RGB color channels

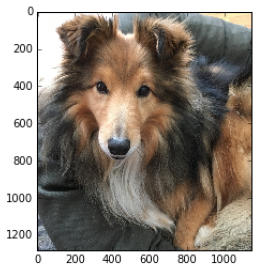
Instead of using a single real number per pixel, for which we then need to find a colormap, we can also specify the color explicitly, using three channels (red, green and blue). If an array is $n \times m \times 3$, then `imshow()` interprets this as an image with RGB colors. The strength of each of the three colors is specified in the range [0,1]. For example, this is how we specify a 2×3 image, with colors red, green, blue, and purple, white, black:

```
img=[[[1,0,0],[0,1,0],[0,0,1]],[[1,0,1],[1,1
plt.imshow(img)
```



We may also import image files, which are translated to arrays of this form, using the `matplotlib.image` module.

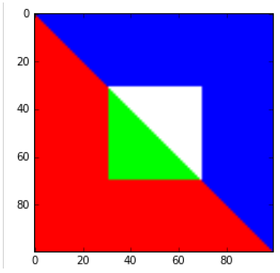
```
import matplotlib.image as mpimg
img=mpimg.imread('susie.png')
plt.imshow(img)
```



Masks

Masks are arrays of Boolean variables (that take values `True` or `False`). If we apply a mask to an image before performing an operation, it will only perform the operation at values where the mask is true. For example, see how we create the following image using masks. The second line converts the array to a numpy array, which allows us to specify the type of the elements. In this case, and almost always in the future, we want to make sure that they are floats, not integers.

```
im=[[[0,0,1]]*100]*100
im=np.array(im,dtype='float')
x,y=np.ogrid[0:100, 0:100]
mask=x>y
im[mask]=[1,0,0]
mask=(x>30)&(x<70)&(30<y)&(y<70)
im[mask]=[1,1,1]
mask=(x>30)&(x<70)&(y>30)&(y<70)&(x>y)
im[mask]=[0,1,0]
plt.imshow(im)
```



Exercises

- Use masks to create images of the flags of Japan, Iceland and Jamaica.
- Use numpy to plot a Newton fractal.
- Import an image of your choice, and add a white border to it.