8 Old-fashioned photographs from the nineteenth century are not quite black and white and not quite color, but seem to have shades of gray, brown, and blue. This effect is known as **sepia**. Write and test a function named **`sepia`** that converts a color image to sepia. This function should first call **`grayscale`** to convert the color image to grayscale. A code segment for transforming the grayscale values to achieve a sepia effect follows. Note that the value for green does not change.

```
if red < 63:
    red = int(red * 1.1);
    blue = int(blue * 0.9)
elif red < 192:
    red = int(red * 1.15);
    blue = int(blue * 0.85);
else:
    red = min(int(red * 1.08), 255);
    blue = int(blue * 0.93);
```

9 Darkening an image requires adjusting all of its pixels toward black as a limit, whereas lightening an image requires adjusting them toward white as a limit. Because black is RGB (0, 0, 0) and white is RGB (255, 255, 255), adjusting the three RGB values of each pixel by the same amount in either direction will have the desired effect. Of course, the algorithms have to avoid exceeding either limit during the adjustments.

Lightening and darkening are actually special cases of a process known as **color filtering**. A color filter is any RGB triple applied to an entire image. The filtering algorithm adjusts each pixel by the amounts specified in the triple. For example, you can increase the amount of red in an image by applying a color filter with a positive red value and green and blue values of 0. The filter (20, 0, 0) would make an image's overall color slightly redder. Alternatively, you can reduce the amount of red by applying a color filter with a negative red value. Once again, the algorithms have to avoid exceeding the limits on the RGB values.

Develop three algorithms for lightening, darkening, and color filtering as three related Python functions, **`lighten`**, **`darken`**, and **`colorFilter`**. The first two functions should expect an image and a positive integer as arguments. The third function should expect an image and a tuple of integers (the RGB values) as arguments. The following session shows how these functions can be used with the images **`image1`**, **`image2`**, and **`image3`**, which are initially white:

```
>>> darken(image1, 128)                # Converts to gray
>>> darken(image2, 64)                 # Converts to dark gray
>>> colorFilter(image3, (255, 0, 0))   # Converts to red
```

Note that the function **`colorFilter`** should do most of the work.

8 Old-fashioned photographs from the nineteenth century are not quite black and white and not quite color, but seem to have shades of gray, brown, and blue. This effect is known as **sepia.** Write and test a function named **sepia** that converts a color image to sepia. This function should first call **grayscale** to convert the color image to grayscale. A code segment for transforming the grayscale values to achieve a sepia effect follows. Note that the value for green does not change.

```
if red < 63:
    red = int(red * 1.1);
    blue = int(blue * 0.9)
elif red < 192:
    red = int(red * 1.15);
    blue = int(blue * 0.85);
else:
    red = min(int(red * 1.08), 255);
    blue = int(blue * 0.93);
```

9 Darkening an image requires adjusting all of its pixels toward black as a limit, whereas lightening an image requires adjusting them toward white as a limit. Because black is RGB (0, 0, 0) and white is RGB (255, 255, 255), adjusting the three RGB values of each pixel by the same amount in either direction will have the desired effect. Of course, the algorithms have to avoid exceeding either limit during the adjustments.

Lightening and darkening are actually special cases of a process known as **color filtering.** A color filter is any RGB triple applied to an entire image. The filtering algorithm adjusts each pixel by the amounts specified in the triple. For example, you can increase the amount of red in an image by applying a color filter with a positive red value and green and blue values of 0. The filter (20, 0, 0) would make an image's overall color slightly redder. Alternatively, you can reduce the amount of red by applying a color filter with a negative red value. Once again, the algorithms have to avoid exceeding the limits on the RGB values.

Develop three algorithms for lightening, darkening, and color filtering as three related Python functions, **lighten**, **darken**, and **colorFilter**. The first two functions should expect an image and a positive integer as arguments. The third function should expect an image and a tuple of integers (the RGB values) as arguments. The following session shows how these functions can be used with the images **image1**, **image2**, and **image3**, which are initially white:

```
>>> darken(image1, 128)               # Converts to gray
>>> darken(image2, 64)                # Converts to dark gray
>>> colorFilter(image3, (255, 0, 0))  # Converts to red
```

Note that the function **colorFilter** should do most of the work.