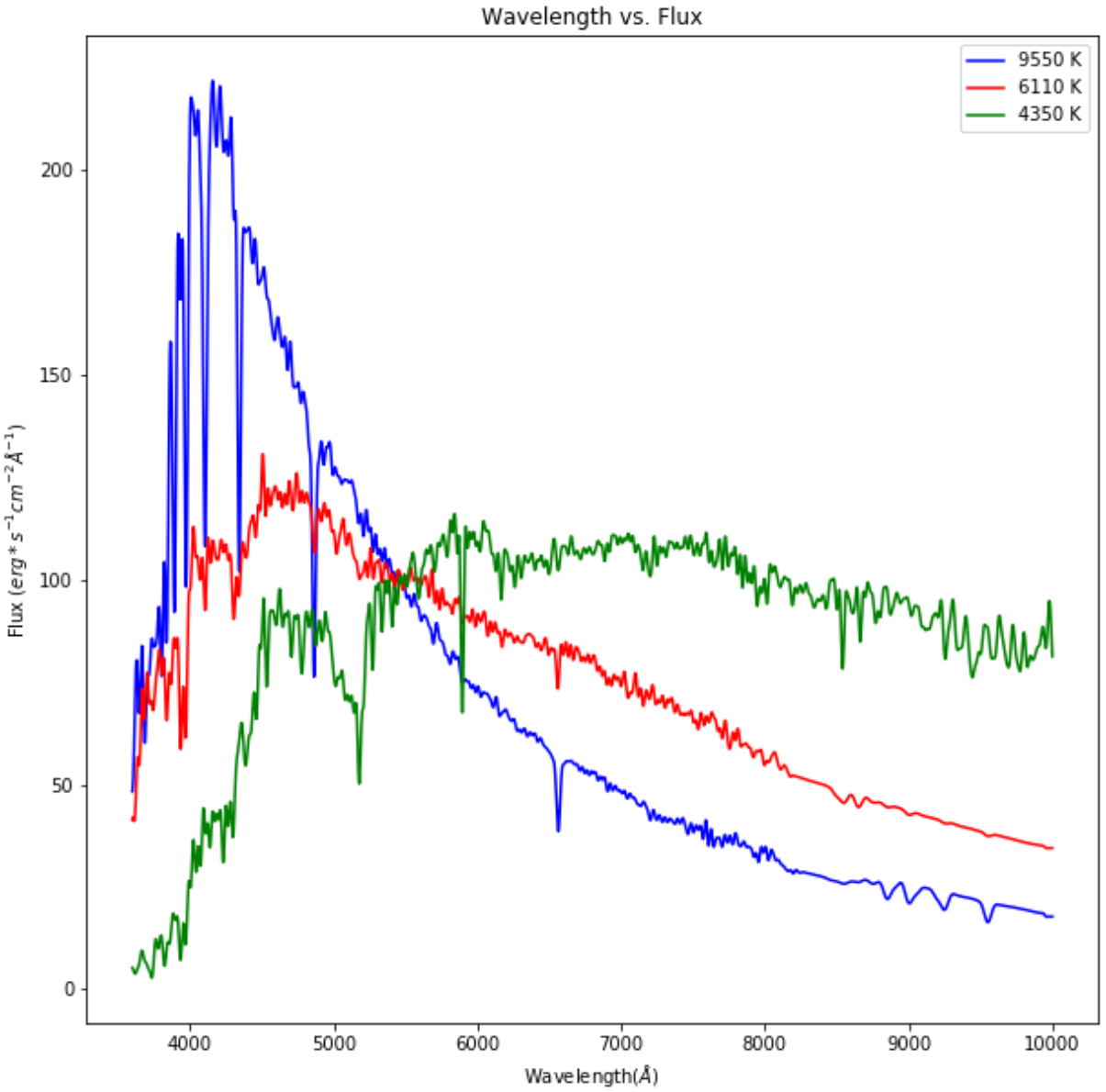
In [1]:
```
#Importing necessary modules and libraries
import os
import numpy as np
from astropy.io import fits
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import cm
```

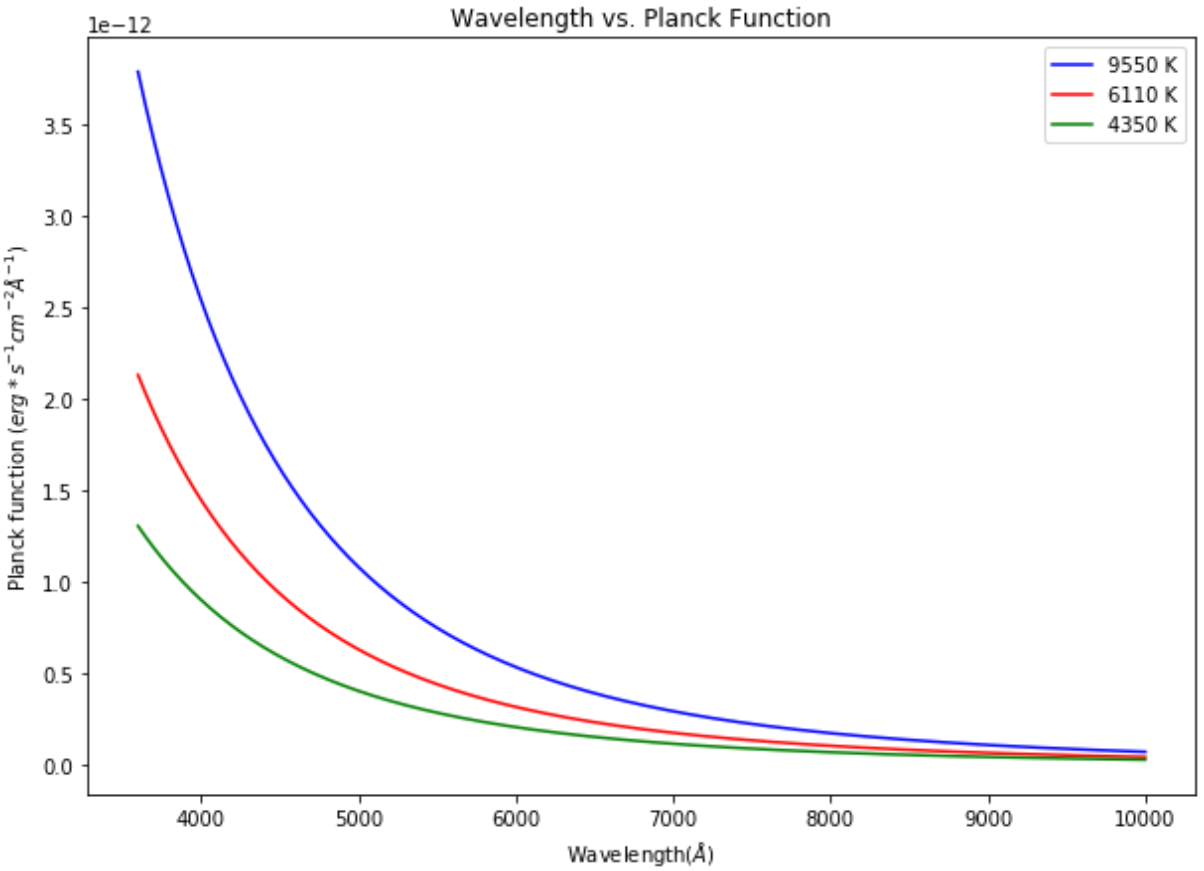
In [2]:
```
#Assigning variables to the .dat files
spec1 = np.loadtxt('f05.dat')
spec2 = np.loadtxt('f10.dat')
spec3 = np.loadtxt('f15.dat')
```

In [3]:
```
#Indexing and assigning variables to columns in the .dat files, then ploting ther
lam1 = spec1[:,0]
lam2 = spec2[:,0]
lam3 = spec3[:,0]

flux1 = spec1[:,1]
flux2 = spec2[:,1]
flux3 = spec3[:,1]

plt.figure(figsize = (10,10))
plt.xlabel(r'Wavelength($\AA $)')
plt.ylabel(r'Flux ($erg * s^{-1}cm^{-2}\AA ^{-1}$)')
plt.title('Wavelength vs. Flux')
plt.plot(lam1, flux1, '-b', label='9550 K')
plt.plot(lam2, flux2, '-r', label='6110 K')
plt.plot(lam3, flux3, '-g', label='4350 K')
plt.legend()
```

Out[3]: <matplotlib.legend.Legend at 0x1fa6ad75400>



In [4]:
```
#Creating the function for Planck's function, then plotting the 3 wavelengths vs
def planck(T, Lambda):
    c = 3*10**10
    h = 6.6261*10**(-23)
    k = 1.3807*10**(-19)
    num = 2*h*c**2
    x = h*c/(Lambda*k*T)
    den=Lambda**5*(np.exp(x)-1)
    B = num/den
    return B

plt.figure(figsize = (10,7))
plt.xlabel(r'Wavelength($\AA $)')
plt.ylabel(r'Planck function ($erg * s^{-1}cm^{-2}\AA ^{-1} $)')
plt.title('Wavelength vs. Planck Function')

plt.plot(lam1, 10**7*planck(9550, lam1), '-b', label = '9550 K')
plt.plot(lam2, 10**7*planck(6110, lam2), '-r', label = '6110 K')
plt.plot(lam3, 10**7*planck(4350, lam3), '-g', label = '4350 K')
plt.legend()

plt.show()
```



In [5]:
```
a = 0
b = 1
N = 1000

x = np.linspace(a, b, N+1)
print(x[0])
print(x[N])
#checking to see if x[0] = a, x[N] = b
```

```
0.0
1.0
```

In [6]:
```
#Creating a numerical integration function
def integral(f, a, b, N):
    x = np.linspace(a, b, N+1)
    f = f(x)
    area = np.sum(f)*(b-a)/N
    return area

integral(np.sin, 0, 1, 2000)
```

Out[6]:
```
0.45990805230102694
```

In [7]:
```
#Integrating sine with the function
integral(np.sin, 0, 1, 500)
```

Out[7]:
```
0.4605390118840933
```

In [8]:
```
#the real integrated value is = 0.459697694131860
```

In [9]:
```
#defining wavelength arrays
from scipy import interpolate
```

In [10]:
```
Flam1 = interpolate.interp1d(lam1, flux1, kind = 'cubic', bounds_error = False, 
Flam2 = interpolate.interp1d(lam2, flux2, kind = 'cubic', bounds_error = False, 
Flam3 = interpolate.interp1d(lam3, flux3, kind = 'cubic', bounds_error = False, 
```

In [11]:
```
#calling transfer function for Bfilt and Vfilt

Bfilt = np.loadtxt('Bfilt.dat')
Vfilt = np.loadtxt('Vfilt.dat')

lamB = Bfilt[:, 0]
lamV = Vfilt[:, 0]

TransB = Bfilt[:, 1]
TransV = Vfilt[:, 1]

TlamB = interpolate.interp1d(lamB, TransB, kind = 'cubic', bounds_error = False,
TlamV = interpolate.interp1d(lamV, TransV, kind = 'cubic', bounds_error = False,
```

In [12]:
```
#Multiplying the Transfer and Flux Density functions so I can integrate it later
def F1B(x): return TlamB(x) * Flam1(x)
def F2B(x): return TlamB(x) * Flam2(x)
def F3B(x): return TlamB(x) * Flam3(x)
def F1V(x): return TlamV(x) * Flam1(x)
def F2V(x): return TlamV(x) * Flam2(x)
def F3V(x): return TlamV(x) * Flam3(x)
```

In [13]:
```
#Finding the magnitude of each spectral data while using integration function to
M1B = -2.5 * np.log10(integral(F1B, 3600, 5600, 2000))
M2B = -2.5 * np.log10(integral(F2B, 3600, 5600, 2000))
M3B = -2.5 * np.log10(integral(F3B, 3600, 5600, 2000))
M1V = -2.5 * np.log10(integral(F1V, 4700, 7000, 2000))
M2V = -2.5 * np.log10(integral(F2V, 4700, 7000, 2000))
M3V = -2.5 * np.log10(integral(F3V, 4700, 7000, 2000))
```

In [14]:
```
print(M1B, M2B, M3B, M1V, M2V, M3V)
```

```
-13.0032509660578 -12.524132189614647 -11.86717592143442 -12.361035804529422 -1
2.377182953561052 -12.312774790531737
```

In [15]:
```
#Finding B-V color for each file:
color1 = M1B - M1V
color2 = M2B - M2V
color3 = M3B - M3V

print(color1, color2, color3)
```

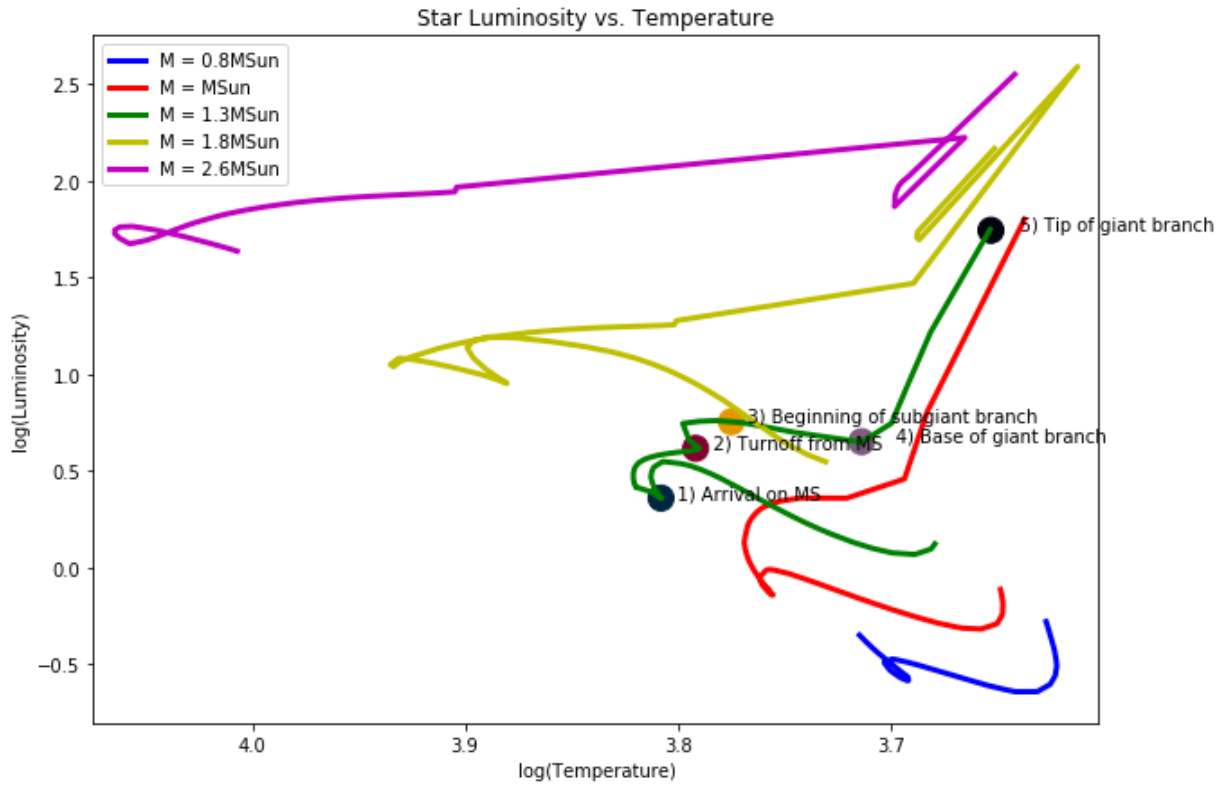
```
-0.6422151615283784 -0.14694923605359556 0.44559886909731716
```

In [16]:
```
#Reading in and indexing the 5 stellar evolutionary files
m08 = np.loadtxt('evol_M0.8.dat')
m10 = np.loadtxt('evol_M1.0.dat')
m13 = np.loadtxt('evol_M1.3.dat')
m18 = np.loadtxt('evol_M1.8.dat')
m26 = np.loadtxt('evol_M2.6.dat')

T08 = m08[:, 0]
T10 = m10[:, 0]
T13 = m13[:, 0]
T18 = m18[:, 0]
T26 = m26[:, 0]
L08 = m08[:, 1]
L10 = m10[:, 1]
L13 = m13[:, 1]
L18 = m18[:, 1]
L26 = m26[:, 1]
```

In [17]:
```
print(min(T08), min(T10), min(T13), min(T18), min(T26))
print(max(T08), max(T10), max(T13), max(T18), max(T26))
```

```
3.6222 3.6373 3.6535 3.6124 3.6419
3.7148 3.7689 3.821 3.9351 4.0645
```

In [18]:
```
#plotting each stellar evolutionary track, and the 5 stages in the 1.0 MSun track

%matplotlib inline


plt.figure(figsize = (10,7))
plt.xlabel(r'log(Temperature)')
plt.ylabel(r'log(Luminosity)')
plt.title('Star Luminosity vs. Temperature')


plt.xlim(max(T26)+0.01, min(T18)-0.01)
plt.plot(T08, L08, '-b', label = 'M = 0.8MSun', lw = 3)
plt.plot(T10, L10, '-r', label = 'M = MSun', lw = 3)
plt.plot(T13, L13, '-g', label = 'M = 1.3MSun', lw = 3)
plt.plot(T18, L18, '-y', label = 'M = 1.8MSun', lw = 3)
plt.plot(T26, L26, '-m', label = 'M = 2.6MSun', lw = 3)

plt.legend()

xMSun = [3.808, 3.79175, 3.77496, 3.71381, 3.6531]
yMSun = [0.3599, 0.617418, 0.754789, 0.651762, 1.745]
plt.scatter(xMSun, yMSun, c = ['#002642', '#840032', '#E59500', '#7B5E7B', '#020
label1 = ['   1) Arrival on MS', '   2) Turnoff from MS', '   3) Beginning of sul

for i, txt in enumerate(label1):
    plt.annotate(txt, (xMSun[i], yMSun[i]))

plt.show()
```



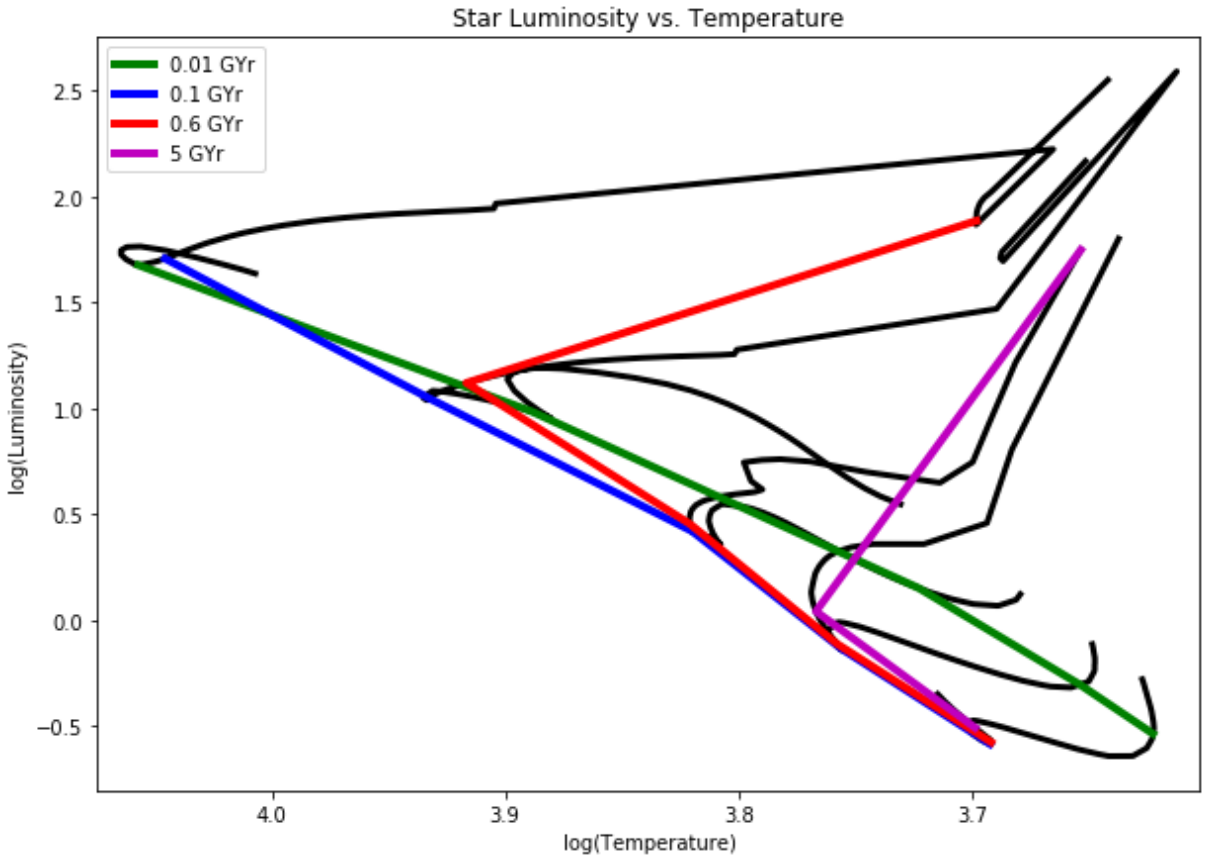
In [19]:
```
#Finding where age of each star = 0.01GYr
age08 = m08[:, 2]
age10 = m10[:, 2]
age13 = m13[:, 2]
age18 = m18[:, 2]
age26 = m26[:, 2]

#My method for finding the coords for each specified age in the 5 .dat files

#for n, age in enumerate(age08):
    #if age == 0.01:
        #print(T08[n], L08[n])
    #if age == 0.1:
        #print(T08[n], L08[n])
    #if age == 0.60256:
        #print(T08[n], L08[n])
    #if age == 5.013:
        #print(T08[n], L08[n])

#for n, age in enumerate(age10):
    #if age == 0.01:
        #print(T10[n], L10[n])
    #if age == 0.1:
        #print(T10[n], L10[n])
    #if age == 0.60256:
        #print(T10[n], L10[n])
    #if age == 5.013:
        #print(T10[n], L10[n])

#for n, age in enumerate(age13):
    #if age == 0.01:
        #print(T13[n], L13[n])
    #if age == 0.1:
        #print(T13[n], L13[n])
    #if age == 0.60256:
        #print(T13[n], L13[n])
    #if age == 4.4679:
        #print(T13[n], L13[n])

#for n, age in enumerate(age18):
    #if age == 0.01:
        #print(T18[n], L18[n])
    #if age == 0.1:
        #print(T18[n], L18[n])
    #if age == 0.60256:
        #print(T18[n], L18[n])
    #if age == 4.4679:
        #print(T18[n], L18[n])

#for n, age in enumerate(age26):
    #if age == 0.01:
        #print(T26[n], L26[n])
    #if age == 0.1:
        #print(T26[n], L26[n])
    #if age == 0.60256:
        #print(T26[n], L26[n])
```

```
    #if age == 4.4679:
        #print(T26[n], L26[n])
```

In [20]:
```
#The coordinates for each point on the respective isochrones (0.01, 0.1, 0.6, and
x_10mil = [3.6226, 3.6539, 3.7228, 3.8883, 4.0571]
y_10mil = [-0.5359, -0.3036, 0.1514, 0.9845, 1.6775]

x_100mil = [3.6924, 3.7563, 3.8199, 3.9328, 4.0456]
y_100mil = [-0.5839, -0.132, 0.4206, 1.0497, 1.7079]

x_600mil = [3.6919, 3.7578, 3.8209, 3.9166, 3.6982]
y_600mil = [-0.5756, -0.1118, 0.4551, 1.1208, 1.8864]

x_5bil = [3.6988, 3.7669, 3.6535]
y_5bil = [-0.5069, 0.042, 1.749]
```

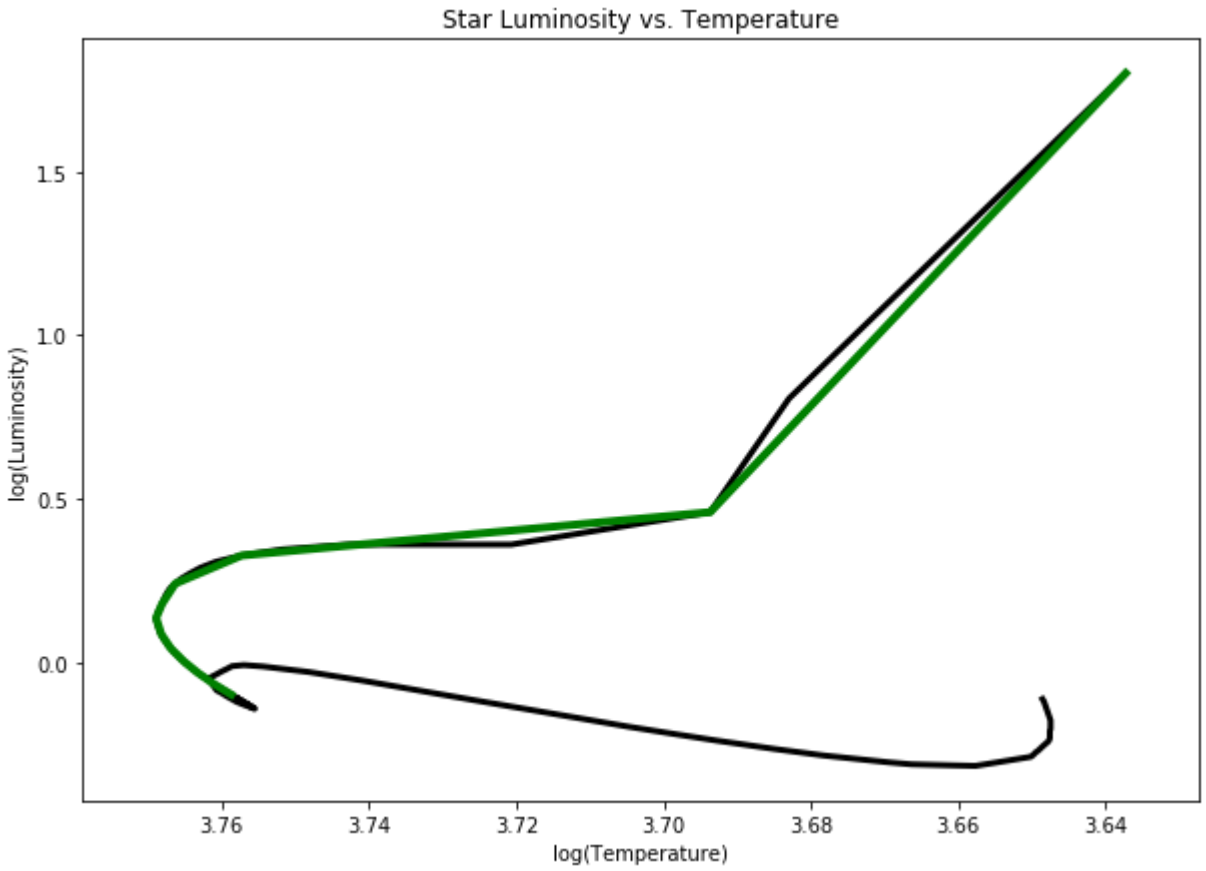
In [21]:
```
#plotting each isochrone on the all-black plot

plt.figure(figsize = (10,7))
plt.xlabel(r'log(Temperature)')
plt.ylabel(r'log(Luminosity)')
plt.title('Star Luminosity vs. Temperature')

plt.xlim(max(T26)+0.01, min(T18)-0.01)
plt.plot(T08, L08, '-k', lw = 3)
plt.plot(T10, L10, '-k', lw = 3)
plt.plot(T13, L13, '-k', lw = 3)
plt.plot(T18, L18, '-k', lw = 3)
plt.plot(T26, L26, '-k', lw = 3)

plt.plot(x_10mil, y_10mil, '-', lw = 4, c = 'g', label = '0.01 GYr')
plt.plot(x_100mil, y_100mil, '-', lw = 4, c = 'b', label = '0.1 GYr')
plt.plot(x_600mil, y_600mil, '-', lw = 4, c = 'r', label = '0.6 GYr')
plt.plot(x_5bil, y_5bil, '-', lw = 4, c = 'm', label = '5 GYr')

plt.legend()
```

Out[21]: <matplotlib.legend.Legend at 0x1fa6ba71c50>



In [22]:
```
#Plotting integer (in GYr) ages of the 1 MSun star
plt.figure(figsize = (10,7))
plt.xlabel(r'log(Temperature)')
plt.ylabel(r'log(Luminosity)')
plt.title('Star Luminosity vs. Temperature')

plt.xlim(max(T10)+0.01, min(T10)-0.01)
plt.plot(T10, L10, '-k', lw = 3)
plt.plot([3.7587, 3.761, 3.7632, 3.7651, 3.7669, 3.7682, 3.7689, 3.7681, 3.7663,
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x1fa6bac1320>]
```



In [23]:
```
import glob
```

In [24]:
```
#Finding the exposure times across the flat frames
flatExptime = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4
flatExp = []
for img in flatExptime:
    flatExp.append(fits.open(img)[0].header['EXPTIME'])
print(flatExp)
```

```
[3.0, 3.0, 3.0, 3.0, 3.0, 3.0]
```

In [25]:
```
#Finding the exposure times across the dark frames
darkExptime = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4
darkExp = []
for img in darkExptime:
    darkExp.append(fits.open(img)[0].header['EXPTIME'])
print(darkExp)
```

```
[300.0, 300.0, 300.0, 300.0, 300.0, 300.0, 300.0, 300.0, 300.0]
```

In [26]:
```
#Now processing the 3 color images from 4/28/11 of M51

biasimages = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4\
bias = []
for img in biasimages:
    bias.append(fits.open(img)[0].data)
bias = np.median(np.array(bias), axis = 0)

darkimages = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4\
dark = []
for img in darkimages:
    dark.append(fits.open(img)[0].data - bias)
dark = (np.median(np.array(dark), axis = 0))/300
```

In [27]:
```
#Finding the exposure times across the dark frames in the 4/28/11 data
darkExptime = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4
darkExp = []
for img in darkExptime:
    darkExp.append(fits.open(img)[0].header['EXPTIME'])
print(darkExp)
```

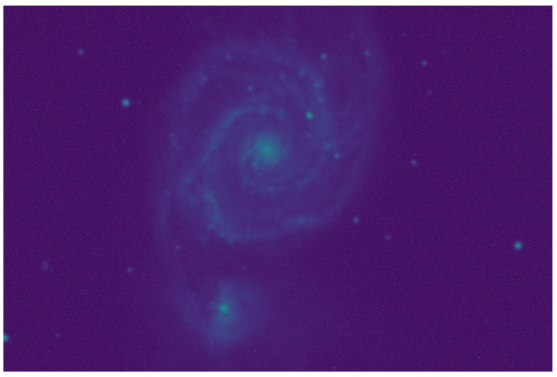
```
[300.0, 300.0, 300.0, 300.0, 300.0, 300.0, 300.0, 300.0, 300.0]
```

In [28]:
```
#Grouping the flat field images by filter first:
Bflatimages = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4
Bflat = []
for img in Bflatimages:
    Bflat.append(fits.open(img)[0].data - bias - dark)
    Bflat[-1] /= np.median(Bflat[-1])
Bflat = np.median(np.array(Bflat), axis = 0)
Bflat /= np.median(Bflat)

Vflatimages = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4
Vflat = []
for img in Vflatimages:
    Vflat.append(fits.open(img)[0].data - bias - dark)
    Vflat[-1] /= np.median(Vflat[-1])
Vflat = np.median(np.array(Vflat), axis = 0)
Vflat /= np.median(Vflat)

iflatimages = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4
iflat = []
for img in iflatimages:
    iflat.append(fits.open(img)[0].data - bias - dark)
    iflat[-1] /= np.median(iflat[-1])
iflat = np.median(np.array(iflat), axis = 0)
iflat /= np.median(iflat)
```

In [29]:
```
#Doing the B Filter first of M51
Bsciimages = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4\
Bscistack = []
for img in Bsciimages:
    hdu = fits.open(img)[0]
    sci = (hdu.data - bias - (dark)/Bflat)
    hdu.data = sci
    outfile = os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4' ,'os.pat
    hdu.writeto(outfile.replace('.fits', '_cal.fits'), overwrite=True)
    Bscistack.append(sci)
Bscistack = np.median(np.array(Bscistack), axis=0)
hdulist = fits.HDUList(fits.PrimaryHDU(Bscistack))
outfile = os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4' , 'median_c
hdulist.writeto(outfile, overwrite = True)
```

In [30]:
```
#Comparing the uncalibrated and calibrated images of M51 B filter

import matplotlib as mpl
from matplotlib.colors import LogNorm
from matplotlib import cm

uncal_B_M51 = os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4\20110428_
cal_B_M51 = os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4\median_comb

img1 = fits.open(uncal_B_M51)[0].data
img2 = fits.open(cal_B_M51)[0].data

fig, axs = plt.subplots(1,2, figsize = (50,50))

cmap = 'viridis'
gamma = (30)
norm = mpl.colors.LogNorm(gamma)

axs[0].imshow(img1, origin = 'lower', cmap = cmap, norm = norm)
axs[1].imshow(img2, origin = 'lower', cmap = cmap, norm = norm)
axs[0].axis('off')
axs[1].axis('off')
```

Out[30]:
```
(-0.5, 1535.5, -0.5, 1023.5)
```

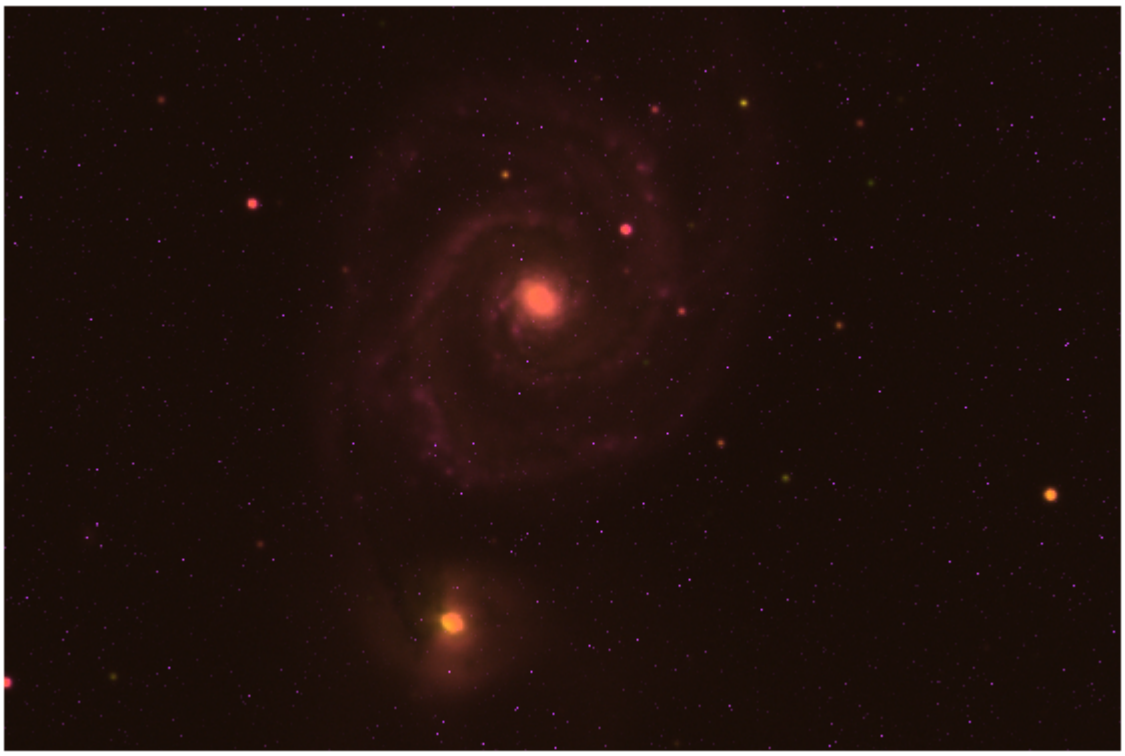


In [31]:
```
Vsciimages = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4\
Vscistack = []
for img in Vsciimages:
    hdu = fits.open(img)[0]
    sci = (hdu.data - bias - (dark)/Vflat)
    hdu.data = sci
    outfile = os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4' ,'os.pat
    hdu.writeto(outfile.replace('.fits', '_cal.fits'), overwrite=True)
    Vscistack.append(sci)
Vscistack = np.median(np.array(Vscistack), axis=0)
hdulist = fits.HDUList(fits.PrimaryHDU(Vscistack))
outfile = os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4' , 'median_co
hdulist.writeto(outfile, overwrite = True)
```

In [32]:
```
isciimages = glob.glob(os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4'
iscistack = []
for img in isciimages:
    hdu = fits.open(img)[0]
    sci = (hdu.data - bias - (dark)/iflat)
    hdu.data = sci
    outfile = os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4' ,'os.pa
    hdu.writeto(outfile.replace('.fits', '_cal.fits'), overwrite=True)
    iscistack.append(sci)
iscistack = np.median(np.array(iscistack), axis=0)
hdulist = fits.HDUList(fits.PrimaryHDU(iscistack))
outfile = os.path.join(os.sep, r'C:\Users\Tim\Documents\Python\lab4' , 'median_c
hdulist.writeto(outfile, overwrite = True)
```

In [34]:
```
#Creating RGB Image of stacked calibrated M51 files
from astropy.visualization import make_lupton_rgb

b_name = ('median_combined_image_B.fits')
r_name = ('median_combined_image_V.fits')
i_name = ('median_combined_image_i.fits')
b = fits.open(b_name)[0].data
r = fits.open(r_name)[0].data
i = fits.open(i_name)[0].data

rgb = make_lupton_rgb( r/np.max(r)*3, i/np.max(i)*0.9, b/np.max(b)*3.85, filenam
plt.figure(figsize = (10,10))
plt.imshow(rgb, origin='lower')
plt.axis('off')
```

Out[34]:
```
(-0.5, 1535.5, -0.5, 1023.5)
```



In [ ]:

Physics 134L, Fall 2021
Professor Max Millar-Blanchaer

Lab 4

Names: Tim, Takashi, Emma

Due date: Monday, Nov 15, by 5:00 pm, though Gauchospace.

Many factors influence the appearance of a star's emitted energy spectrum, but the most important is the stellar temperature. You already saw this at the end of Lab 2: the three stars have almost the same chemical composition, but their spectra look very different. Their surface temperatures range from around 4000 K to around 10000 K: Spectrum 1 is from a ~5000 K star, Spectrum 2 is from a ~9500 K star, and Spectrum 3 is from a ~6200 K star.

The cluster images that you have been looking at, however, are not spectra. The telescope takes the light from the stars, focuses it, and uses a filter to let only some colors through. Only photons within a specified wavelength range actually make it onto the detector. Here we will investigate how the visible-light spectrum changes with temperature, and how these changes are manifested in the stars' photometric colors.

The Lab 4 data directory contains three filter files and three additional stellar spectra. The stars are at three different temperatures:

`f05.dat` = 9550 K

`f10.dat` = 6110 K

`f15.dat` = 4350 K

These new files contain the energy flux F_λ (proportional to erg s^{-1} cm^{-2} Å^{-1}), where wavelength λ is measured in Angstroms (Å), and 1 Angstrom = 0.1 nm. The flux values have been arbitrarily normalized so they have a value of 100 units near 5500 Å. Remember from last class that the normalization of the stellar spectrum told you the angular diameter of the star on the sky. The shape of the spectrum tells you about the physical conditions of the star, mainly its temperature and chemical composition.

Take the spectra of the three stars `f05.dat`, `f10.dat`, and `f15.dat`, and plot them together on the same graph, using different colors. Also plot the Planck functions at these three temperatures on the same axes. You will notice that the hottest star is a particularly poor match to the blackbody; this is due to the hydrogen in the star's atmosphere.

Using a CCD detector with colored filters, we measure the energy flux from stars integrated over the bandpass of each filter:

$$F = \int_0^\infty T[\lambda] F_\lambda d\lambda,$$

where $T[\lambda]$ is called the *transfer function* and describes how efficiently we record energy from photons of a given wavelength. $T[\lambda] = 1$ means that our instrument records every photon at a given wavelength, while $T[\lambda] = 0$ means that it never records a single photon. **We will do the integrals for the filters B and V.** It's worth spending a bit of time on how to do an integral numerically, as this is something many of you will have to do again and again.

The integral is defined as the limit of the Riemann sum,

$$\int_a^b f(x)dx = \lim_{N\to\infty} \sum_{i=0}^{N-1} f[x_i]\Delta x_i,$$

where x_i is the spacing between values of x_i, $x_0 = a$, and $x_N = b$. Let's see how we can write this in python. First, we'll define x to run from a to b with $N+1$ points in total. For now, define $a = 0$, $b = 1$, and $N = 1000$ (we'll change these later when we actually integrate over the band passes).

Angle 2

`x = np.linspace(a, b, N + 1)`

Now check: use the print function to verify that `x[0]` = a and `x[N]` = b. In your case, the spacing is uniform, so Δx_i is the same for all points i.

Now use your value for `Delta_x`, the numpy function sum, and your values of x to numerically evaluate the integral

$$\int_0^1 \sin x dx$$

and compare to the actual value of the integral. Remember that your (left) Riemann sum should stop at $N-1$, which you can write using index notation as `[:N]`. Show that your answer is closer to the truth if $N = 2000$, but is farther from the truth if $N = 500$.

Now we can do an integral numerically, but how do we integrate

$$F = \int_0^\infty T[\lambda] F_\lambda d\lambda$$

We need to define a wavelength array, and then be able to call $T[\lambda]$ and $F[\lambda]$ just as we did above with sine. We can do this by first reading in the data files and then *interpolating* between the data points. We will use the package `scipy.interpolate`, and we will use cubic polynomials (called *splines*) to interpolate:

```
from scipy import interpolate
```

From this point, python actually makes our lives pretty easy. First, let's read in one of the spectral files (first column is wavelength, second column is flux density F_λ):

```
spec_data = np.loadtxt('f05.dat')
```

and then

```
Flam = interpolate.interp1d(spec_data[:, 0], spec_data[:, 1], kind='cubic')
```

Now `Flam` is a function that you can call just like `np.sin`. Calling this function will return an error if you ask it for the value outside the range of x-values you supplied to `interpolate.interp1d`. **For the transfer function, we may assume that it is zero outside of the wavelength range given. Read the documentation page (check the keywords `bounds_error` and `fill_value`) to make the function behave in this way.**

Now we are ready to compute magnitudes. **Use the tools you have developed above to integrate the three spectra over the transfer functions given by `Bfilt.dat` and `Vfilt.dat`.** The definition of magnitude includes a zero point:

$$M = -2.5 \log_{10} F - Z.$$

For now, ignore the zero points Z (let $Z = 0$). What are the magnitudes of the three stars in the two bands? Please fill in the following table. Also, please compute the difference between the B and V band magnitudes (the $B-V$ color). This is the ratio between the flux in the B band and the flux in the V band.

	f05.dat	f10.dat	f15.dat
B magnitude	-13.0032	-12.5241	-11.8672
V magnitude	-12.361	-12.3272	-12.3127
$B-V$ color	-0.6422	-0.147	0.4457

Does a high value of $B-V$ indicate a star that is very blue (puts out lots of blue light relative to yellow/green light), or a star that is very red?

High value (more negative B-V value) means more blue. So f05.dat is more blue than f10 and f15

What is the qualitative relation between temperature and $B-V$ color?

The relationship is inverse log

So colors provide a way to estimate stellar temperatures. Suppose we know the temperature then what else do we know? Stellar evolution theory provides relations between the characteristics of stars at the time they are born (mass and composition), and their properties (including temperature and luminosity) at any time in their later lives. For a quick introduction to (or refresher on) many of the basic notions, see

`http://www.tim-thompson.com/hr.html`

For a more quantitative picture of stellar evolution, consider the files `evol_M0.8.dat`, `evol_M1.0.dat`, `evol_M1.3.dat`, `evol_M1.8.dat`, and `evol_M2.6.dat`. These contain evolutionary tracks for stars with compositions similar to the Sun's and with masses of 0.8, 1.0, 1.3, 1.8, and 2.5 times the mass of the Sun. The columns of the tables contain $\log_{10}$(Temperature (K)), $\log_{10}$(Luminosity (solar units)), and Age (in Gyr). Notice (by opening one of the files in a text editor) that the range of ages in the various tables is fairly wide massive stars live for much shorter times than low-mass ones, though they are much more luminous. They burn brightly but burn out quickly.

Make a plot that shows the evolutionary tracks for all 5 stars overplotted on one graph. Use different colors to distinguish the different masses, and (for once) connect the points with straight lines, using no symbols. Make the x-axis log(Temperature), but make Temperature increase from right to left. (Because that's the way astronomers do it. Don't ask.) Make the y-axis log(Luminosity), increasing from bottom to top. On the track for the 1-solar-mass star, mark the following points with text integers 1-5 the following locations: (1) Arrival on the main sequence. (2) Turnoff from the main sequence. (3) Beginning of the subgiant branch. (4) Base of the giant branch. (5) Tip of the giant branch.

Now make the same plot again, only make all of the curves black, and do not provide any annotating integers. In each of the five data files, identify the point at which the age of the star is 10 million years old (0.01 Gyr). In the text editor, create a table that you can read into python, giving log(Temperature) and log(Luminosity) for each of the stars at this age. Overplot this (jagged, because there are not very many points on it) curve, using a heavy green line. This is the 10-million-year isochrone (line of constant age). If you start with a group of stars (the members of a star cluster, say) that have the same initial composition and are formed at nearly the same time, then 10 million years later, you will see the stars of different masses strung out along this isochrone. In the same way, plot isochrones for ages of 0.1 Gyr, 0.6 Gyr, and 5.0 Gyr.

1) $x = 3.802$ $y = 0.3599$

2) $x = 3.79175$ $y = 0.613419$

3) $x = 3.79496$ $y = 0.754719$

4) $x = 3.71381$ $y = 0.651762$

5) $x = 3.6531$ $y = 1.745$

What does it mean if the oldest age in a table is smaller than one of these isochrone ages?

It means that the star turned into a non-stellar object (e.g. supernova) and its mass got too large

Do a similar plot, but this time use only the data for the 1-solar-mass star (evol_M1.0.dat). Use a text editor and create a new text file that contains only the lines that lie as close as you can come to ages that are integer multiples of 1 Gyr, up to 12 Gyr. Overplot this table onto the original curve, using suitable symbols, but no lines connecting the points.

About what fraction of this star's life is spent on the main sequence? What fraction on the giant branch?

About 5/6 of its life. It will spend about 1/12 of it on the giant branch.

Calibration of Astronomical Images

Now we are going to learn how to process raw astronomical images into science quality images. Read Chromey 9.3.

You will use the code written in the rest of the lab as a model for your own jupyter notebook. Your notebook (which you will turn in) will contain the code that will actually reduce the data.

There are four main operations done to reduce the data. The first is to make a bias frame and subtract it off. Here is some code to make the bias frame:

```
biasimages = glob.glob(os.path.join(os.sep, 'Z:', '20110427_raw', 'c_b_*.fits'))
bias = []
for img in biasimages:
    bias.append(fits.open(img)[0].data)
bias = np.median(np.array(bias), axis=0)
```

The first line uses something called `glob` to create a list of files that match the path. The path shown can have any number of characters between `c_b_` and `.fits`. If you prefer, you may explicitly give a list of files, for example

```
rawdatadir = os.path.join(os.sep, 'Z:', 20110427_raw')
biasimages = [os.path.join(rawdatadir, 'c_b_20110427-001_2.fits'),
              os.path.join(rawdatadir, 'c_b_20110427-002_2.fits'),
              os.path.join(rawdatadir, 'c_b_20110427-003_2.fits')]
```

et cetera. The `glob` module is very powerful. If you use it, just make sure you are matching all of the files you want, and only the files you want.

Explain how the bias subtraction code works. Why does this code use a median rather than a mean to combine the data?

The code creates an empty list that will be filled with .fits files that are bias frames. These files are retrieved with the glob.glob and os.path.join modules, which go into the raw file directory and pulls out the biases. It takes the median instead of mean since the mean doesn't completely remove unwanted cosmic ray data, and the median does.

What is the purpose of bias subtraction?

Bias frames are frames that include purely device electronic background noise that interfere with the data. Subtracting them helps remove data bias, like radioactivity and device electronic interference.

The next step is to create a dark frame. In your explanation, please justify why the code subtracts a bias frame. What is the purpose of a dark frame, and how is this distinct from a bias?

The code subtracts a bias frame for every dark frame so that, when all the frames are utilized, there are no unwanted data that biases the final product. This includes having as accurate of a dark frame as possible. Unlike a bias frame, dark frames are data errors over a non-zero integration time (longer exposure). This data comes primarily from heat from the device, instead of ultra-fast cosmic rays or radiation.

The code to create a dark frame is below. You will have to write your own code to select all of the dark images (they are the ones whose filenames start with c_d_); the code assumes that you have made such a list and called it darkimages.

```
dark = []
for img in darkimages:
    dark.append(fits.open(img)[0].data - bias)
dark = np.median(np.array(dark), axis=0)
```

Explain what the code is doing to create a dark.

The code creates an empty list, then fills it with all the dark frames files with append. (The dark frames are (_d_*.fits). It subtracts the bias frames because the dark frames can't have any of the bias frame data. It then combines the data with a median of the dark frames,

Dark frames need to be scaled with exposure time. Why?

Because dark frames' responses are caused over a non-zero integration time. Dark frames are obtained with a long exposure with the shutter closed, and each hot pixel differ in visibility, so it's important to scale it with the exposure time when combining the frames.

After we subtract off bias and dark frames we need to flat field the data. What is the purpose of flat fielding?

Flat fielding takes into account of how sensitive the device's response is to a uniform target (e.g. a pitch-black sky). This is because there could be different pixel responses of an image, which could produce different pixel outputs in the image.

The code that makes a flat is below this box. Again, this assumes that you have created a list of flatfielding images called `flatimages`. Flat-field images have filenames that start with `c_f_`. Please note that you should use the flat-field images with the same filter as your science images (the filter is the last character before `.fits`. Why should you use flats taken with the same filter?

Because a flat is dependent on how the device captures image data, so if a filter is used, the the flats must also use the filter to accurately detect and equalize each response. Filters also influence the focal-plane image, so using a different/no filter will provide inaccurate flat fielding.

```
flat = []
for img in flatimages:
    flat.append(fits.open(img)[0].data - bias - dark)
    flat[-1] /= np.median(flat[-1])
flat = np.median(np.array(flat), axis=0)
flat /= np.median(flat)
```

Note that `flat[-1]` refers to the flatfield image that we just added to the end of the list of flats: the second line within the `for` loop only modifies this most recently appended image.

Explain how the flatfielding code works. In your explanation, state why we divide flatfields by their median values.

In the code, there is an empty list created called flat = []. This is then used to fill in with the flatfield images pulled out of the raw directory. Also, these images have the bias and dark frames subtracted from them. We divide by the median value after combining the data with a median so that the median pixel has a value of 1 ADU.

Modify the code in your notebook to account for the different exposure times between the flat-field and dark exposures. You may look up the exposure times in the fits headers, or access them using something like

```
exptime = fits.open(imagefile)[0].header['EXPTIME']
```

After creating bias, darks, and a flat we can finally process our science images using the code below. Similarly to before, sciimages should be a list of the science images that you would like to calibrate and combine.

```
scistack = []
for img in sciimages:
    hdu = fits.open(img)[0]
    sci = (hdu.data - bias - dark)/flat
    hdu.data = sci
    outfile = os.path.join(os.sep, 'Z:', 'yourdir', os.path.basename(img))
    hdu.writeto(outfile.replace('.fits', '_cal.fits'), overwrite=True)
    scistack.append(sci)
scistack = np.median(np.array(scistack), axis=0)
hdulist = fits.HDUList(fits.PrimaryHDU(scistack))
outfile = os.path.join(os.sep, 'Z:', 'yourdir', 'median_combined_image.fits')
hdulist.writeto(outfile, overwrite=True)
```

Please make sure to replace 'yourdir' with your directory (using your names). And don't forget to scale the dark by the appropriate exposure time!

Explain how the reduction and stacking code works.

The code first uses glob to pull the science images into an empty list called scistack. The variable sci integrates the formula for properly subtracting off dark, bias, & flat field frames. This is the reduction stage. The first outfile exports the data to a custom directory. The "scistack" after the for-loop then is used to find the median of the data, which is the exported in the 2nd "outfile" as a fits file. This is the stacking stage.

Now process the three color images of M51 taken on 4/28/11. The first step is change the filenames in reduce to match the names of the calibration files we took on 4/28. Lets do the *B* band images first. Once you have gotten all of the file names and/or calls to glob, run your jupyter notebook to produce reduced images. The data taken on 4/28 do not have the appropriate flatfields, so please use the flatfield images from 4/27.

Open up one of the calibrated output files with ds9. Compare it to the uncalibrated image. Describe the differences, and include a side-by-side comparison of a small part of the images in your jupyter notebook (use matplotlib to make the image plots as you have done in several past labs) to demonstrate these differences.

In the uncalibrated image, it's difficult to scale properly, and the image is not as accurate in its pixel data (i.e. it's not clear compared to calibrated image). In calibrated image, it's easier to scale and more visible when plotted in Jupyter.

Problem 2

Replace all of the calls to `np.median` with `np.mean`. What effect does this have on your calibrated data?

Replacing np.median with np.mean gives an image that makes the object more difficult to see (less white pixels, other noise, etc.)

Repeat the reduction process from the previous page on the V and i band images of M51 (use `np.median()`). Please make sure to give each stacked file a unique filename.

Now that we have calibrated images in three colors, lets combine them into a full color image! You can make full-color RGB images directly in matplotlib. As easy way to do this it to use Astropy, as shown in `https://docs.astropy.org/en/stable/visualization/rgb.html`. You will likely need to normalize the RGB images you pass to the `make_lupton_rgb()` function and change the Q and stretch parameters. Something like this should be a place to start:

```
image=make_lupton_rgb(image_r/np.max(image_r)*3,image_g/np.max(image_g)*0.9,
image_b/np.max(image_b)*0.85, Q=1, stretch=.1 )
```

Regardless of how you proceed, please make as pretty a color image as you can from this data, and describe the procedure you used below.

I manually uploaded the 3 different filter images onto jupyter, assigned variables to them, opened them, then used the make_lupton_rgb function code given above. I set Q = 0 and stretch = 0.05; my r multiplier = 3, i multi = 0.9, and b multiplier = 3.85.

Now download the isochrones (read up on isochrones!) from
http://www.astronomy.ohio-state.edu/iso/pl.html
Plot the solar metallicity isochrone from
http://www.astronomy.ohio-state.edu/iso/lj98m01757.txt
that best fits your data, and use this to estimate the age and distance of the cluster. Please make sure to include this plot in your notebook.

How close is your estimate to the published value of age and distance (look this up)? What do you think are the largest sources of error in your estimate?

In this box put the name of your partner, and describe how you worked together to complete this lab.

Takashi, Emma. We worked on it in the library outside of class

Upload this worksheet and the pdfs of your jupyter notebook(s) to the Gradescope assignment link on the "Lab 4" tab of the Gauchospace page. Each group member should upload their own version of this page, but it is ok if your code submissions are the same if you worked in partners the whole time. Clearly label the different parts of the lab using the Markdown features of Jupyter notebook and using code comments. You want to make it easy for the TA to find your work.