OOP

June 1, 2016

```
In [1]: # Just to know last time this was run:
        import time
        print time.ctime()
```

```
Wed Jun  1 17:12:28 2016
```

1 G Object Oriented Programation. Objects, classes, etc...

This is part of the Python lecture given by Christophe Morisset at IA-UNAM. More informations at: http://python-astro.blogspot.mx/

```
In [2]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        import os # We will need this latter
        from scipy.integrate import simps
        import urllib2
```

```
/Users/christophemorisset/anaconda/lib/python2.7/site-packages/matplotlib/font_mana
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take
```

We want here to make some plots of atmosphere models that will be downloaded from internet. We need:

- Download the file
- read it
- plot it

This can all be done in functions, and also in object.

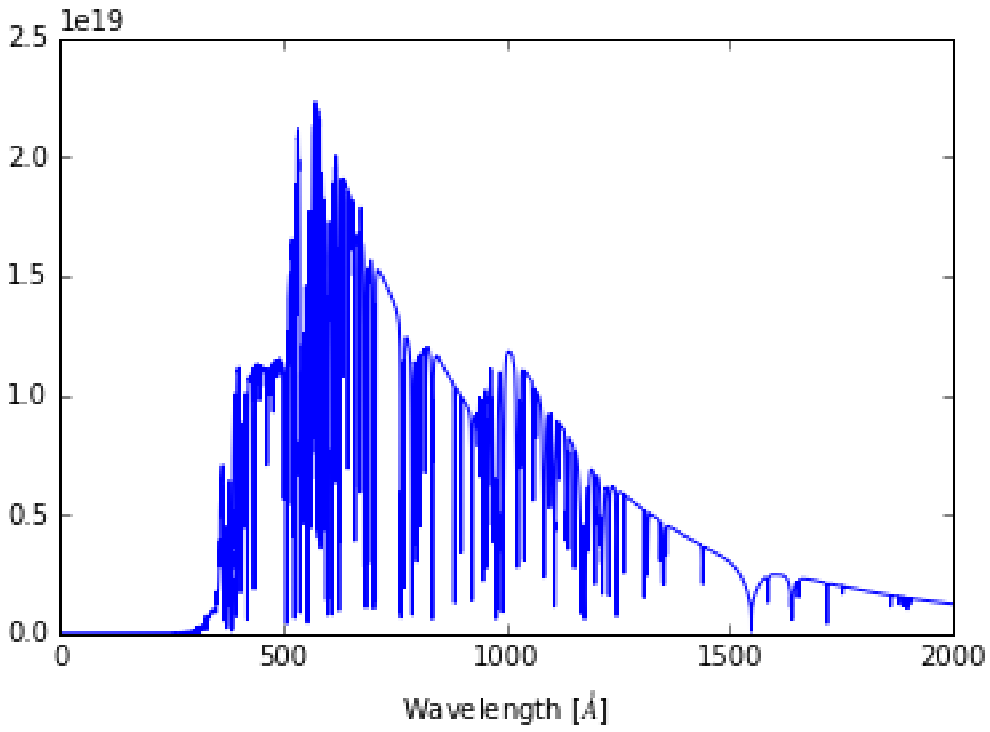
1.0.1 Functions

Let's first see the way we can do it with functions:

The files are located there: http://astro.uni-tuebingen.de/~rauch/TMAF/flux_He+C+N+O.html

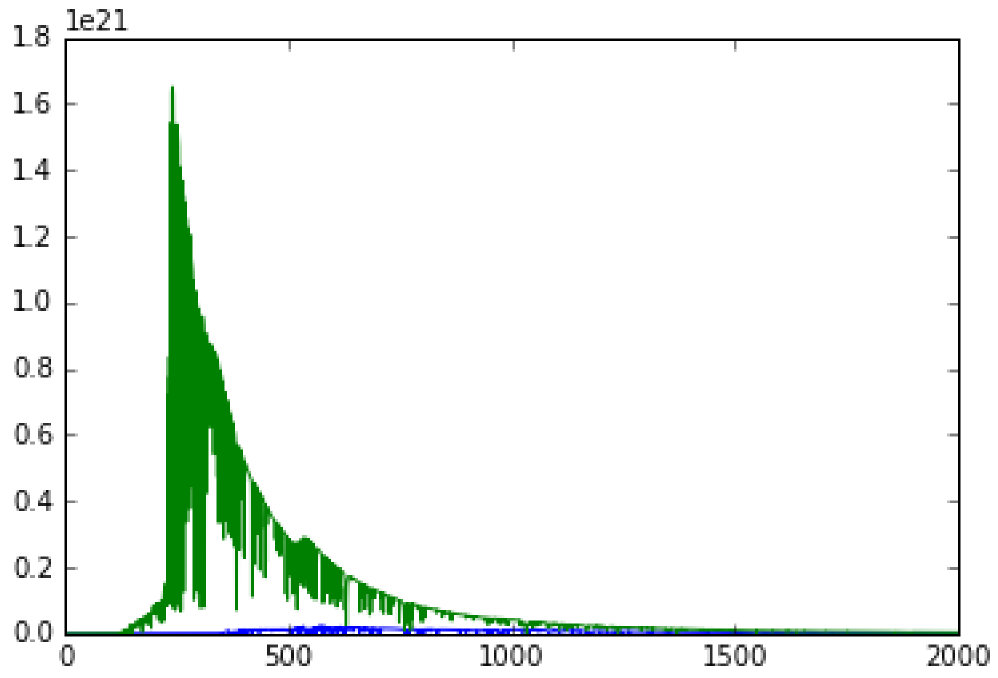
For exemple, a file is: http://astro.uni-tuebingen.de/~rauch/TMAF/NLTE/He+C+N+O/0050000_7.00_33_5
We can download it using urllib2, putting this into a function:

```
In [3]: def dlfile(filename):
            stel_file = urllib2.urlopen('http://astro.uni-tuebingen.de/~rauch/TMAF/
            output = open(filename,'wb') #the file where we will put the data. b st
            output.write(stel_file.read()) # the reading of the distant file is rec
            output.close()

In [4]: filename = '0050000_7.00_33_50_02_15.bin_0.1.gz'
        dlfile(filename)

In [5]: ! ls -l *gz

-rw-------  1 christophemorisset  staff    89353 Oct 26  2015 0040000_6.00_33_50_02
-rw-------  1 christophemorisset  staff    86018 Jun  1 17:12 0050000_7.00_33_50_02
-rw-------  1 christophemorisset  staff    89971 May 31 15:31 0090000_6.00_33_50_02
-rw-------  1 christophemorisset  staff    90544 Oct 26  2015 0100000_6.00_33_50_02
-rw-------  1 christophemorisset  staff    89170 Oct 26  2015 0140000_6.00_33_50_02
-rw-------  1 christophemorisset  staff    88591 Oct 26  2015 0190000_6.00_33_50_02
-rw-------  1 christophemorisset  staff  4229587 Sep  9  2015 CALIFA_ah7.dat.gz
-rw-------  1 christophemorisset  staff  1270918 Sep 21  2015 MySQL.pdf.gz


In [6]: data = np.genfromtxt(filename, comments='*', names='wl, fl') # genfromtxt c

In [7]: data

Out[7]: array([(5.0, 4.596e-20), (5.1, 3.524e-19), (5.2, 2.475e-18), ...,
               (1999.8, 1.242e+18), (1999.9, 1.242e+18), (2000.0, 1.241e+18)],
              dtype=[('wl', '<f8'), ('fl', '<f8')])

In [8]: plt.plot(data['wl'], data['fl']) # let's have a look at the data
        plt.xlabel(r'Wavelength [$\AA$]');
```

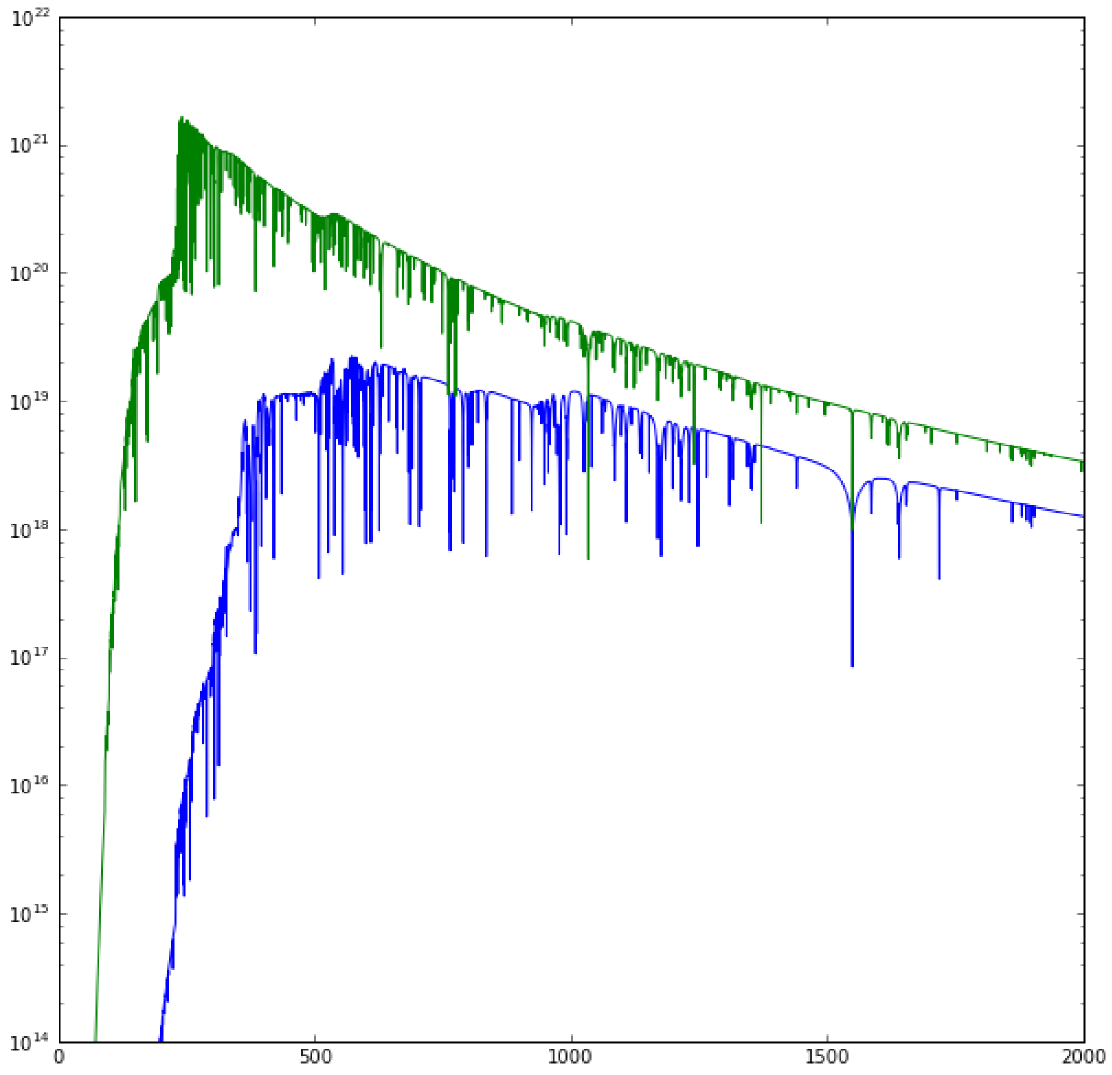


If we want to overplot another file, we only have to download it and follow the same process:

```
In [9]: filename2 = '0110000_7.00_33_50_02_15.bin_0.1.gz'
        dlfile(filename2)

In [10]: data2 = np.genfromtxt(filename2, comments='*', names='wl, fl') # data and

In [11]: plt.plot(data['wl'], data['fl'])
         plt.plot(data2['wl'], data2['fl']);
```

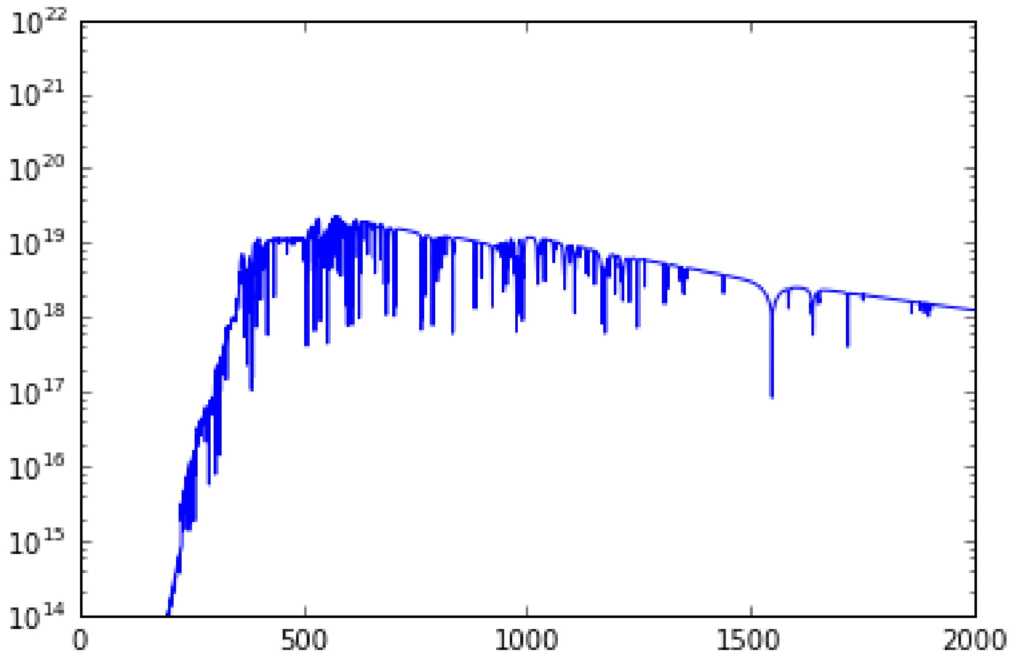


```
In [12]: fig, ax = plt.subplots(figsize=(10,10))
         ax.plot(data['wl'], data['fl'])
         ax.plot(data2['wl'], data2['fl'])
         ax.set_yscale('log')
         ax.set_ylim(1e14, 1e22);
```



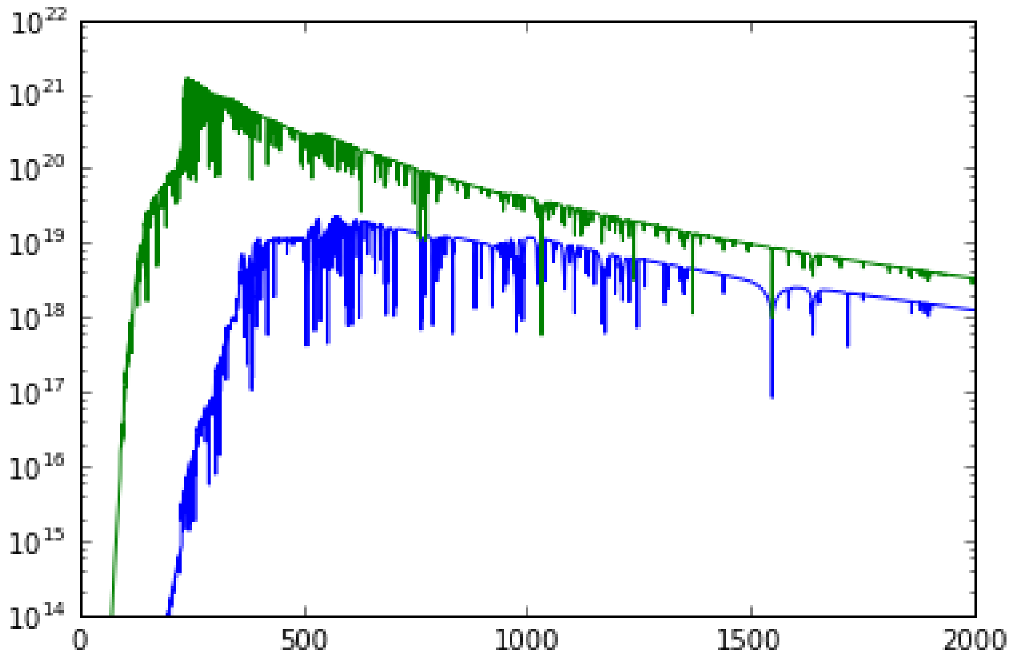
Great, but it would be better if everything were in the same place. Making a function more complete that deal with everything:

```
In [13]: def plot_spr(filename):
             dlfile(filename) # download the file
             data = np.genfromtxt(filename, comments='*', names='wl, fl') # read it
             fig, ax = plt.subplots()
             ax.plot(data['wl'], data['fl']) # plot it
             ax.set_yscale('log') # use log axes
             ax.set_ylim(1e14, 1e22)
         plot_spr(filename)
```

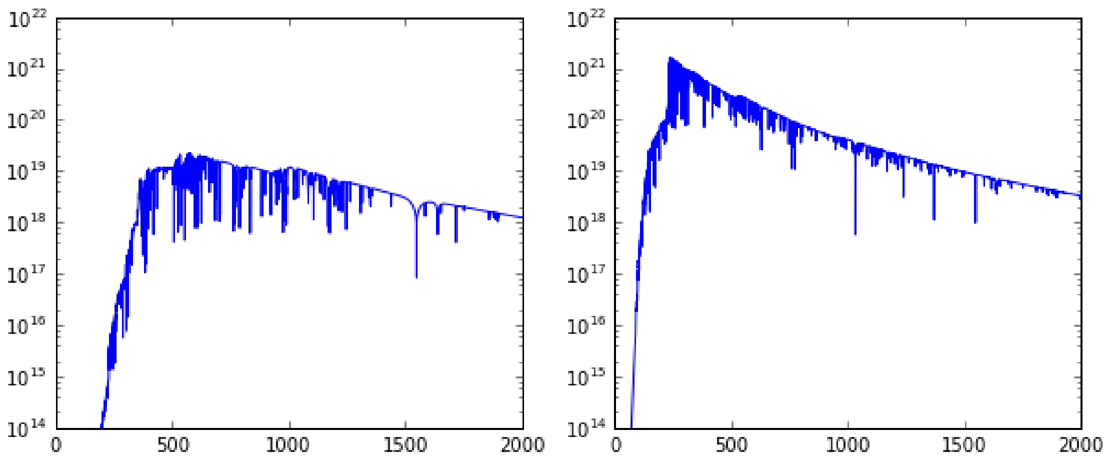


The main problem here is to superimpose the 2 plots. We can define the axis object outside and send it to the function:

```
In [14]: def plot_spr(filename, ax=None): # default is no axis sent to the functio
             dlfile(filename)
             data = np.genfromtxt(filename, comments='*', names='wl, fl')
             if ax is None: # make a figue if no axis is passed to the function
                 fig, ax = plt.subplots()
             else:
                 fig = plt.gcf()
             ax.plot(data['wl'], data['fl'])
             ax.set_yscale('log')
             ax.set_ylim(1e14, 1e22)

         fig, ax = plt.subplots() # the figure and axis is buildt before calling th
         plot_spr(filename, ax=ax) # sending axis let the plots appear on the same
         plot_spr(filename2, ax=ax)
```



```
In [15]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4)) # the figure and axi
         plot_spr(filename, ax=ax1) # sending axis let the plots appear on the same
         plot_spr(filename2, ax=ax2)
```



But now that everything is compact, we don't have access to the data, they are INSIDE the function. . .

1.0.2 Classes and Objects

It's time to make a class and to instantiate objects. Classes are intelligent containers. The can hold variables and functions (called methods). The following terminology is from http://www.tutorialspoint.com/python/python_classes_objects.htm:

- *Class*: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- *Class variable or attribute*: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables aren't used as frequently as instance variables are.
- *Data member*: A class variable or instance variable that holds data associated with a class and its objects.
- *Function overloading*: The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects (arguments) involved.
- *Instance variable or attribute*: A variable that is defined inside a method and belongs only to the current instance of a class.
- *Inheritance* : The transfer of the characteristics of a class to other classes that are derived from it.
- *Instance*: An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- *Instantiation* : The creation of an instance of a class.
- *Method* : A special kind of function that is defined in a class definition.
- *Object* : A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- *Operator overloading*: The assignment of more than one function to a particular operator.

```
In [16]: class Stel_Spectrum(object):
             """
             This object downloads a file from http://astro.uni-tuebingen.de/~rauch
             and is able to make some plots.
             """
             def __init__(self, filename): # This function will be called at the in
                 self.filename = filename # we put the file name value into an inst
                 self.dlfile() # calling a method (defined below). No need for argu
                 self.data = np.genfromtxt(self.filename, comments='*', names='wl,

             def dlfile(self): # method.
                 if not os.path.exists(self.filename): # only donwload if not yet d
                     stel_file = urllib2.urlopen('http://astro.uni-tuebingen.de/~ra
                                                 self.filename)
                     output = open(self.filename,'wb')
                     output.write(stel_file.read())
                     output.close()

             def plot_spr(self, ax=None): # another method. Used to plot
                 if ax is None:
```

```
                fig, ax = plt.subplots()
            else:
                fig = plt.gcf()
            ax.plot(self.data['wl'], self.data['fl'])
            ax.set_yscale('log')
            ax.set_ylim(1e14, 1e22)
```

```
In [17]: sp1 = Stel_Spectrum(filename) # instantiation of an object from the Stel_S
         sp2 = Stel_Spectrum(filename2) # another object. They have the same struct
```

```
In [18]: print sp1.filename # access the instace variable
         print sp2.filename
```

```
0050000_7.00_33_50_02_15.bin_0.1.gz
0110000_7.00_33_50_02_15.bin_0.1.gz
```

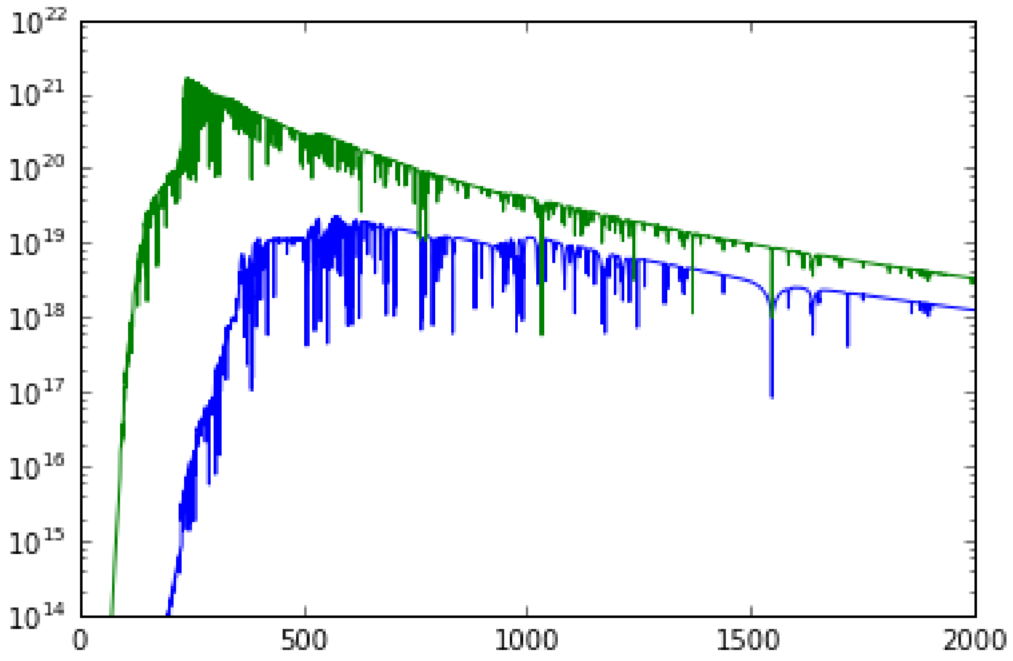
```
In [19]: #sp1.TAB
```

```
          File "<ipython-input-19-4b88285095dd>", line 1
        sp1.
            ^
    SyntaxError: invalid syntax
```

```
In [20]: sp2.data # the data are available.
```

```
Out[20]: array([(5.0, 1028.0), (5.1, 2393.0), (5.2, 5362.0), ...,
                (1999.8, 3.328e+18), (1999.9, 3.327e+18), (2000.0, 3.326e+18)],
               dtype=[('wl', '<f8'), ('fl', '<f8')])
```

```
In [21]: fig, ax = plt.subplots()
         sp1.plot_spr(ax=ax) # calling the metod
         sp2.plot_spr(ax=ax)
```



```
In [22]: len(sp1.data['wl']) # the data from the object are like any other data.

Out[22]: 19951
```

We can add comments and a method that gives information about the object itself.

```
In [23]: class Stel_Spectrum(object):
             """
             This object downloads a file from http://astro.uni-tuebingen.de/~rauch
             and is able to make some plots.
             """
             def __init__(self, filename):
                 """
                 Initialisation of the Stel_Spectrum object.
                 Parameter:
                     - filename e.g. 0050000_7.00_33_50_02_15.bin_0.1.gz
                 """
                 self.filename = filename
                 self.dlfile()
                 self.data = np.genfromtxt(filename, comments='*', names='wl, fl')

             def dlfile(self):
                 """
                 Downloading file if not already here
                 """
```

```
                if not os.path.exists(filename):
                    print('Downloading {}'.format(self.filename))
                    stel_file = urllib2.urlopen('http://astro.uni-tuebingen.de/~ra
                    output = open(filename,'wb')
                    output.write(stel_file.read())
                    output.close()

            def plot_spr(self, ax=None):
                """
                Plot the spectrum.
                Parameter:
                    - ax: an axis (optionnal). If None or absent, axis is created
                """
                if ax is None:
                    fig, ax = plt.subplots()
                ax.plot(self.data['wl'], self.data['fl'])
                ax.set_yscale('log')
                ax.set_ylim(1e14, 1e22)

            def print_info(self):
                """
                Print out the filename and the number of points
                """
                print('Filename: {0}, number of points: {1}'.format(self.filename,
```

```
In [24]: sp1 = Stel_Spectrum(filename) # we have to instatiate again to take the ch
         sp2 = Stel_Spectrum(filename2)
         sp1.print_info()
```

```
Filename: 0050000_7.00_33_50_02_15.bin_0.1.gz, number of points: 19951
```

```
In [25]: help(sp1) # the comments are easily accessible
```

```
Help on Stel_Spectrum in module __main__ object:

class Stel_Spectrum(__builtin__.object)
 |  This object downloads a file from http://astro.uni-tuebingen.de/~rauch/TMAF/NLT
 |  and is able to make some plots.
 |
 |  Methods defined here:
 |
 |  __init__(self, filename)
 |      Initialisation of the Stel_Spectrum object.
 |      Parameter:
 |          - filename e.g. 0050000_7.00_33_50_02_15.bin_0.1.gz
 |
 |  dlfile(self)
 |      Downloading file if not already here
 |
 |  plot_spr(self, ax=None)
 |      Plot the spectrum.
 |      Parameter:
 |          - ax: an axis (optionnal). If None or absent, axis is created
 |
 |  print_info(self)
 |      Print out the filename and the number of points
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

```
In [26]: help(sp1.plot_spr)
```

```
Help on method plot_spr in module __main__:

plot_spr(self, ax=None) method of __main__.Stel_Spectrum instance
    Plot the spectrum.
    Parameter:
        - ax: an axis (optionnal). If None or absent, axis is created
```

```
In [27]: print sp1
```

```
<__main__.Stel_Spectrum object at 0x114033f10>
```

Adding more method and changing the name of the data to wl and fl. We can accept T and logg to define the filename and download it. Some error catching process are implemented. We laso add a method to compute the integrale of the flux over the wavelengths.

```
In [28]: class Stel_Spectrum(object):
             """
             This object downloads a file from http://astro.uni-tuebingen.de/~rauch
             and is able to make some plots.
             """
             def __init__(self, filename=None, T=None, logg=None, verbose=False):
                 """
                 Initialisation of the Stel_Spectrum object.
                 Parameter:
```

```
            - filename
            - T: temperature in K, e.g. 150000
            - logg: e.g. 7.5
        The wl attribute is an array of wavelengths in Angstrom.
        The fl attribute is the flux in erg/s/cm2/A
        """
        self.verbose = verbose
        if filename is None:
            if T is not None and logg is not None:
                self.T = T
                self.logg = logg
                self.filename = '0{0:06.0f}_{1:.2f}_33_50_02_15.bin_0.1.gz
            else:
                raise TypeError("T and logg must be given")
        else:
            self.filename = filename
            self.T = float(filename.split('_')[0])
            self.logg = float(filename.split('_')[1])
        self.dlfile()
        if self.file_found:
            data = np.genfromtxt(self.filename, comments='*', names='wl, f
            self.fl = data['fl']
            self.wl = data['wl'] # in A
            self.fl /= 1e8 # F LAMBDA  GIVEN IN ERG/CM**2/SEC/CM -> erg/s/
            if self.verbose:
                print('Data from {} read.'.format(self.filename))
        else:
            self.wl = None
            self.fl = None

    def dlfile(self):
        """
        Downloading file if not already here. Put it in the curremt direct
        """
        if not os.path.exists(self.filename):
            print('Downloading {}'.format(self.filename))
            try:
                stel_file = urllib2.urlopen('http://astro.uni-tuebingen.de
                                            self.filename)
                output = open(self.filename,'wb')
                output.write(stel_file.read())
                output.close()
                self.file_found=True
            except:
                print('file {} not found'.format(self.filename))
                self.file_found=False
        else:
            self.file_found=True
    def plot_spr(self, ax=None):
        """
        Plot the spectrum.
        Parameter:
            - ax: an axis (optionnal). If Noe or absent, axis is created
        """
        if self.wl is None:
            print('No data to plot')
            return
        if ax is None:
            fig, ax = plt.subplots()
        ax.plot(self.wl, self.fl, label='T3={0:.0f}, logg={1}'.format(self
        ax.set_yscale('log')
        ax.set_ylim(1e6, 1e14)
        ax.set_xlabel('Wavelength (A)')

    def print_info(self):
        """
        Print out the filename and the number of points
        """
        print self.__repr__()

    def __repr__(self):
        """
        This is what is used when calling "print <obj>" or <obj> ENTER
        """
        if self.wl is None:
            return'Filename: {0}, No data'.format(self.filename)
        else:
            return'Filename: {0}, number of points: {1}'.format(self.filen

    def get_integ(self):
        """
        Return the integral of Flambda over lambda, in erg/s/cm2
        """
        if self.wl is None:
            print('No data')
            return None
        return simps(self.fl, self.wl) # perform the integral
```

```
In [29]: sp1 = Stel_Spectrum(T=130000, logg=6)
```

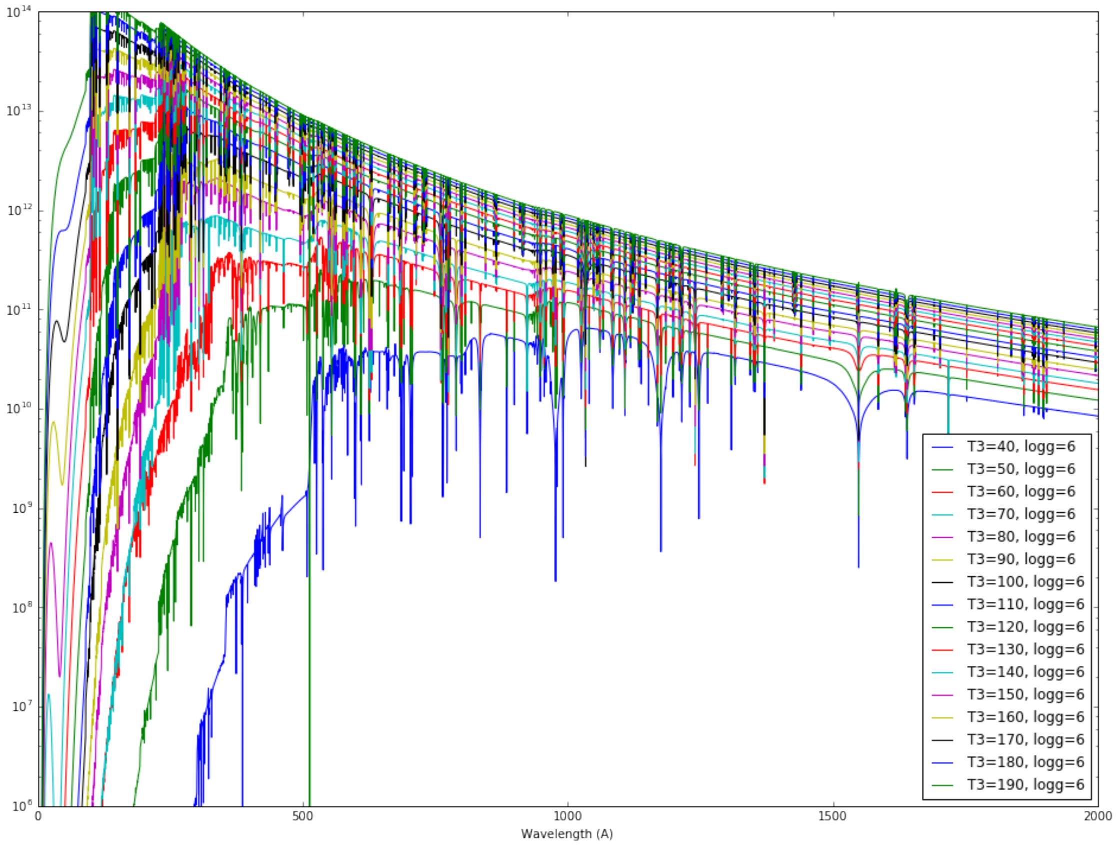
```
Downloading 0130000_6.00_33_50_02_15.bin_0.1.gz
```

```
In [30]: spectra = [] # we create an empty list
         for T in np.linspace(40000, 190000, 16): # this is the list of available t
```

```
            spectra.append(Stel_Spectrum(T=T, logg=6, verbose=True)) # we fill the

Data from 0040000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0050000_6.00_33_50_02_15.bin_0.1.gz
Data from 0050000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0060000_6.00_33_50_02_15.bin_0.1.gz
Data from 0060000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0070000_6.00_33_50_02_15.bin_0.1.gz
Data from 0070000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0080000_6.00_33_50_02_15.bin_0.1.gz
Data from 0080000_6.00_33_50_02_15.bin_0.1.gz read.
Data from 0090000_6.00_33_50_02_15.bin_0.1.gz read.
Data from 0100000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0110000_6.00_33_50_02_15.bin_0.1.gz
Data from 0110000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0120000_6.00_33_50_02_15.bin_0.1.gz
Data from 0120000_6.00_33_50_02_15.bin_0.1.gz read.
Data from 0130000_6.00_33_50_02_15.bin_0.1.gz read.
Data from 0140000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0150000_6.00_33_50_02_15.bin_0.1.gz
Data from 0150000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0160000_6.00_33_50_02_15.bin_0.1.gz
Data from 0160000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0170000_6.00_33_50_02_15.bin_0.1.gz
Data from 0170000_6.00_33_50_02_15.bin_0.1.gz read.
Downloading 0180000_6.00_33_50_02_15.bin_0.1.gz
Data from 0180000_6.00_33_50_02_15.bin_0.1.gz read.
Data from 0190000_6.00_33_50_02_15.bin_0.1.gz read.
```

```
In [31]: spectra # the list hold 16 objects, each one with its own data and methods

Out[31]: [Filename: 0040000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0050000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0060000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0070000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0080000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0090000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0100000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0110000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0120000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0130000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0140000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0150000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0160000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0170000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0180000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951,
          Filename: 0190000_6.00_33_50_02_15.bin_0.1.gz, number of points: 19951]
```

```
In [32]: fig, ax = plt.subplots(figsize=(16,12))
         for sp in spectra: # easy to loop on the objects
             sp.plot_spr(ax=ax)
         ax.legend(loc=4);
```

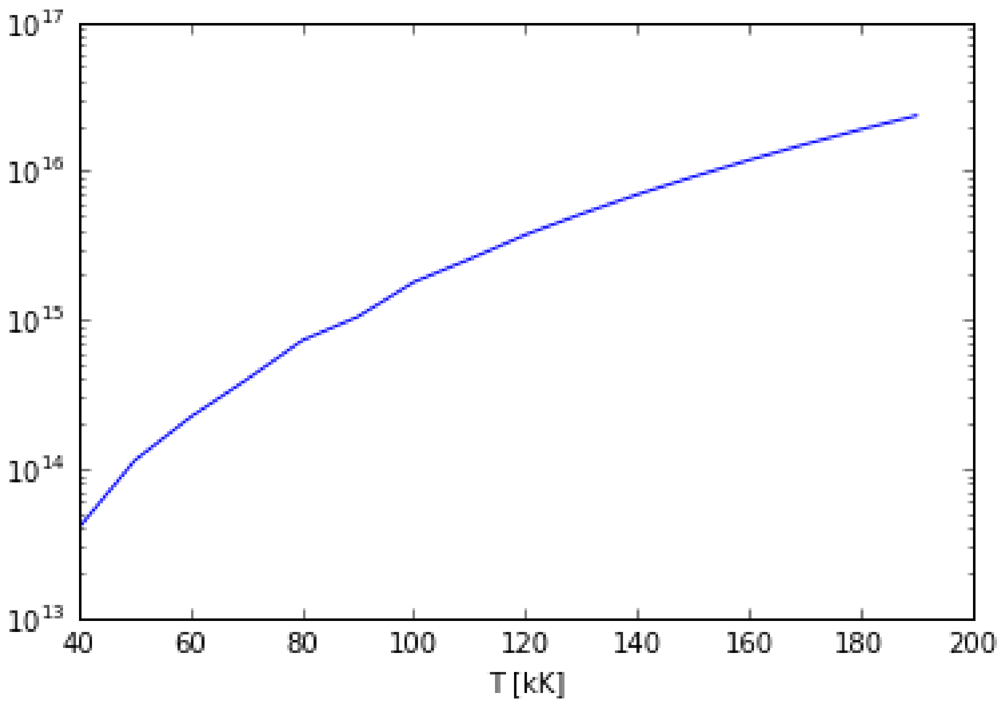


```
In [33]: for sp in spectra:
             print sp.T, sp.get_integ()
```

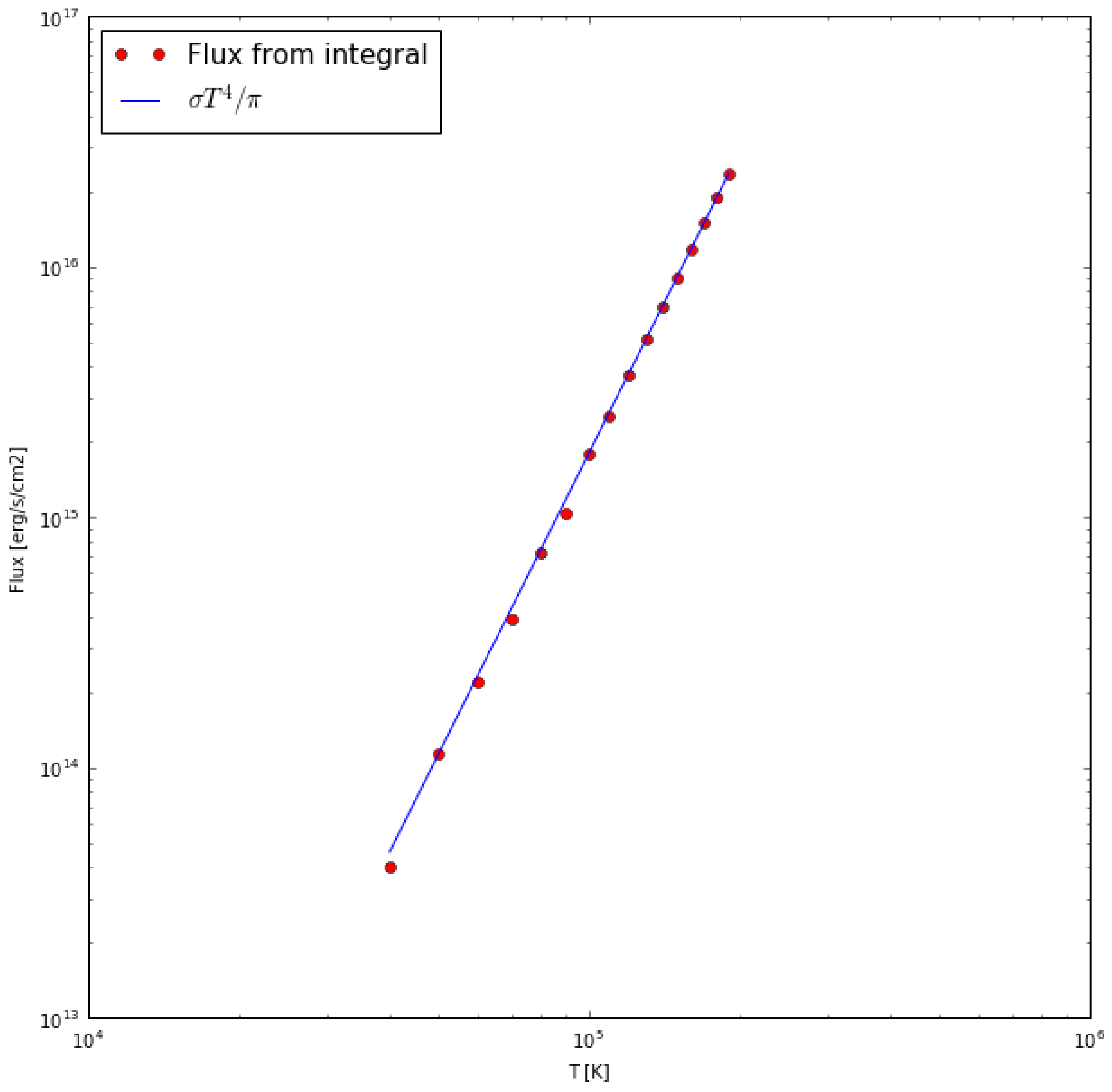
```
40000.0 3.9998068061e+13
50000.0 1.13703254293e+14
60000.0 2.22526357929e+14
70000.0 3.95696185164e+14
80000.0 7.25435743437e+14
90000.0 1.04784744936e+15
100000.0 1.79075718082e+15
110000.0 2.55483260388e+15
120000.0 3.7228781289e+15
130000.0 5.13212682334e+15
140000.0 6.93066619748e+15
150000.0 9.11406474868e+15
160000.0 1.18061135259e+16
```

```
170000.0 1.50574601651e+16
180000.0 1.89350279443e+16
190000.0 2.34906298088e+16
```

```
In [34]: # using list comprehension to compute on the fly the coordinates of the pl
         plt.semilogy([sp.T/1e3 for sp in spectra], [sp.get_integ() for sp in spect
         plt.xlabel('T [kK]');
```



```
In [35]: # Better to put the values into a numpy array:
         T = np.array([sp.T for sp in spectra])
         F = np.array([sp.get_integ() for sp in spectra])

In [36]: # check that the luminosity increase like sigma.T**4
         from astropy import constants # in real life, it is better to move this to
         sigma = constants.sigma_sb.to('erg/(s K4 cm2)') # convert Steffen-Boltzman
         fig, ax = plt.subplots(figsize=(10,10))
         ax.loglog(T, F, 'ro',label='Flux from integral')
         ax.loglog(T, sigma.value * T**4 / np.pi, label=r'$\sigma T^4 / \pi$') # ov
         ax.legend(loc=2, fontsize=15)
         ax.set_xlabel('T [K]')
         ax.set_ylabel('Flux [erg/s/cm2]');
```



1.0.3 Using *args and **kwargs in functions

This allows to pass arguments (without and with keyword respectively) to function. No need to know what are the arguments when desining the function.

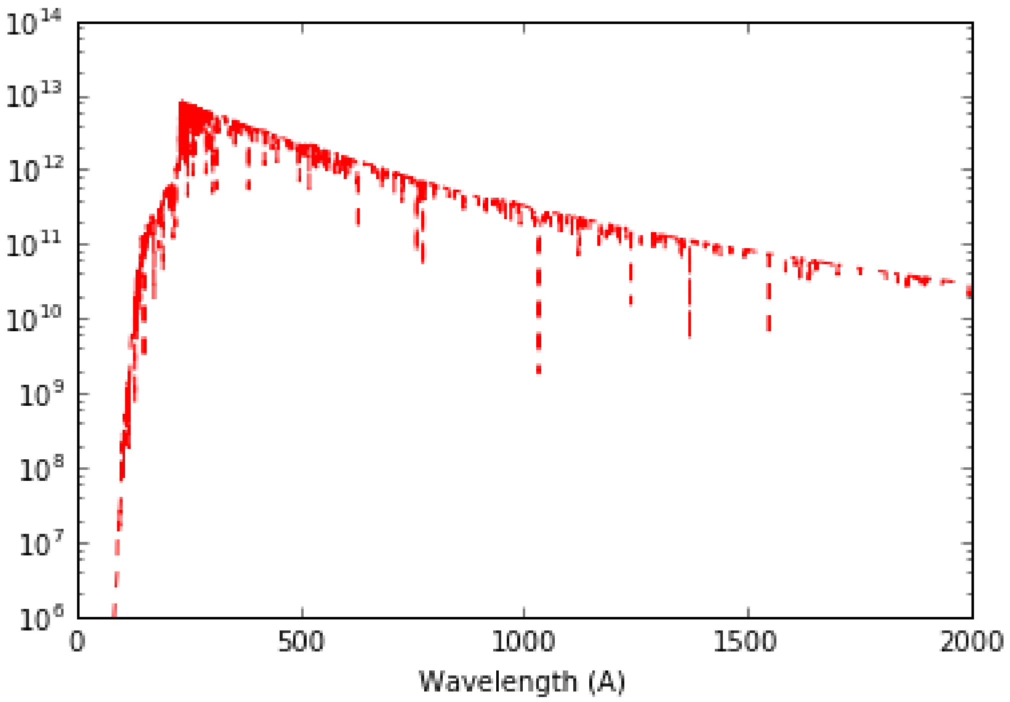
```
In [37]: class Stel_Spectrum(object):
             """
             This object downloads a file from http://astro.uni-tuebingen.de/~rauch
             and is able to make some plots.
             """
             def __init__(self, filename=None, T=None, logg=None):
                 """
                 Initialisation of the Stel_Spectrum object.
                 Parameter:
```

```
            - filename
            - T: temperature in K, e.g. 150000
            - logg: e.g. 7.5
        The wl variable is an array of wavelengths in Angstrom.
        The fl variable is the flux in erg/s/cm2/A
        """
        if filename is None:
            if T is not None and logg is not None:
                self.T = T
                self.logg = logg
                self.filename = '0{0:06.0f}_{1:.2f}_33_50_02_15.bin_0.1.gz
            else:
                raise TypeError("T and logg must be given")
        else:
            self.filename = filename
            self.T = float(filename.split('_')[0])
            self.logg = float(filename.split('_')[1])
        self.dlfile()
        if self.file_found:
            data = np.genfromtxt(self.filename, comments='*', names='wl, f
            self.fl = data['fl']
            self.wl = data['wl'] # in A
            self.fl /= 1e8 # F LAMBDA  GIVEN IN ERG/CM**2/SEC/CM -> erg/s/
        else:
            self.wl = None
            self.fl = None

    def dlfile(self):
        """
        Downloading file if not already here. Put it in the curremt direct
        """
        if not os.path.exists(self.filename):
            print('Downloading {}'.format(self.filename))
            try:
                stel_file = urllib2.urlopen('http://astro.uni-tuebingen.de
                                            self.filename)
                output = open(self.filename,'wb')
                output.write(stel_file.read())
                output.close()
                self.file_found=True
            except:
                print('file {} not found'.format(self.filename))
                self.file_found=False
        else:
            self.file_found=True

    def plot_spr(self, ax=None, *args, **kwargs):
        """
```

```
                Plot the spectrum.
                Parameter:
                    - ax: an axis (optionnal). If Noe or absent, axis is created
                    - any extra parameter is passed to ax.plot
                """
                if self.wl is None:
                    print('No data to plot')
                    return
                if ax is None:
                    fig, ax = plt.subplots()
                ax.plot(self.wl, self.fl,
                        label='T3={0:.0f}, logg={1}'.format(self.T/1e3, self.logg)
                        *args, **kwargs) # Here are the transmissions of extra par
                ax.set_yscale('log')
                ax.set_ylim(1e6, 1e14)
                ax.set_xlabel('Wavelength (A)')

            def print_info(self):
                """
                Print out the filename and the number of points
                """
                print self.__repr__()

            def __repr__(self):
                """
                This is what is used when calling "print <obj>" or <obj> ENTER
                """
                if self.wl is None:
                    return'Filename: {0}, No data'.format(self.filename)
                else:
                    return'Filename: {0}, number of points: {1}'.format(self.filen

            def get_integ(self):
                """
                Return the integral of Flambda over lambda, in erg/s/cm2
                """
                if self.wl is None:
                    print('No data')
                    return None
                return simps(self.fl, self.wl) # perform the integral
```

```
In [38]: sp1 = Stel_Spectrum(T=100000, logg=5)
         print sp1
         fig, ax = plt.subplots()
         sp1.plot_spr(ax, 'r', linestyle='--') # any extra argument is passed to pl
```

```
Downloading 0100000_5.00_33_50_02_15.bin_0.1.gz
Filename: 0100000_5.00_33_50_02_15.bin_0.1.gz, number of points: 19951
```



1.0.4 Class variables or class attributes

They are known and share between all the instanciations of a class. Usefull to count the number of objects of the same class.

```
In [39]: class Stel_Spectrum(object):
             """
             This object downloads a file from http://astro.uni-tuebingen.de/~rauc
             and is able to make some plots.
             """

             spec_count = 0 # This attibute is at the level of the class, not of t
             def __init__(self, filename=None, T=None, logg=None):
                 """
                 Initialisation of the Stel_Spectrum object.
                 Parameter:
                     - filename
                     - T: temperature in K, e.g. 150000
                     - logg: e.g. 7.5
                 The wl variable is an array of wavelengths in Angstrom.
                 The fl variable is the flux in erg/s/cm2/A
                 """
```

```
        if filename is None:
            if T is not None and logg is not None:
                self.T = T
                self.logg = logg
                self.filename = '0{0:06.0f}_{1:.2f}_33_50_02_15.bin_0.1.gz
            else:
                raise TypeError("T and logg must be given")
        else:
            self.filename = filename
            self.T = float(filename.split('_')[0])
            self.logg = float(filename.split('_')[1])
        self.dlfile()
        if self.file_found:
            data = np.genfromtxt(self.filename, comments='*', names='wl, f
            self.fl = data['fl']
            self.wl = data['wl'] # in A
            self.fl /= 1e8 # F LAMBDA  GIVEN IN ERG/CM**2/SEC/CM -> erg/s/
        else:
            self.wl = None
            self.fl = None
        Stel_Spectrum.spec_count += 1

    def dlfile(self):
        """
        Downloading file if not already here. Put it in the curremt direct
        """
        if not os.path.exists(self.filename):
            print('Downloading {}'.format(self.filename))
            try:
                stel_file = urllib2.urlopen('http://astro.uni-tuebingen.de
                                            self.filename)
                output = open(self.filename,'wb')
                output.write(stel_file.read())
                output.close()
                self.file_found=True
            except:
                print('file {} not found'.format(self.filename))
                self.file_found=False
        else:
            self.file_found=True

    def plot_spr(self, ax=None, *args, **kwargs):
        """
        Plot the spectrum.
        Parameter:
            - ax: an axis (optionnal). If None or absent, axis is created
            - any extra parameter is passed to ax.plot
        """
```

```
            if self.wl is None:
                print('No data to plot')
                return
            if ax is None:
                fig, ax = plt.subplots()
            ax.plot(self.wl, self.fl,
                    label='T3={0:.0f}, logg={1}'.format(self.T/1e3, self.logg)
                    *args, **kwargs) # Here are the transmissions of extra pa
            ax.set_yscale('log')
            ax.set_ylim(1e6, 1e14)
            ax.set_xlabel('Wavelength (A)')

        def print_info(self):
            """
            Print out the filename and the number of points
            """
            print self.__repr__()

        def __repr__(self):
            """
            This is what is used when calling "print <obj>" or <obj> ENTER
            """
            if self.wl is None:
                return'Filename: {0}, No data'.format(self.filename)
            else:
                return'Filename: {0}, number of points: {1}'.format(self.filen

        def get_integ(self):
            """
            Return the integral of Flambda over lambda, in erg/s/cm2
            """
            if self.wl is None:
                print('No data')
                return None
            return simps(self.fl, self.wl) # perform the integral

        def __del__(self):
            Stel_Spectrum.spec_count -= 1
```

```
In [40]: sp1 = Stel_Spectrum(T=100000, logg=5)
         sp2 = Stel_Spectrum(T=100000, logg=6)
         sp3 = Stel_Spectrum(T=100000, logg=7)
         print Stel_Spectrum.spec_count
         print sp1.spec_count
```

```
Downloading 0100000_7.00_33_50_02_15.bin_0.1.gz
3
3
```

```
In [41]: del sp1
         print Stel_Spectrum.spec_count

2
```

```
In [42]: for logg in (5, 6, 7, 8):
             sp = Stel_Spectrum(T=100000, logg=logg)
         print Stel_Spectrum.spec_count # the deleted objects are not count (it wou

Downloading 0100000_8.00_33_50_02_15.bin_0.1.gz
3
```

```
In [43]: sp = 3
         print Stel_Spectrum.spec_count

2
```

That can be used for example to change a value for a class variable used everywhere (e.g. the reddening correction to be applied to all the spectra before plotting them. . .)

1.0.5 Adding functionnality to classes and objects (monkey-patch)

```
In [44]: sp1 = Stel_Spectrum(T=100000, logg=5) # Instanciation of a class
         def print_ok(): # defining a function outside the class
             print 'ok'
         sp1.print_ok = print_ok # include the function to the object
         sp1.print_ok() # works, the instance is modified

ok
```

```
In [45]: def print_ok2(self):
             print self.T
         Stel_Spectrum.print_ok2 = print_ok2 # include the function to the class
         sp1.print_ok2() # the class has been modified, and it applies immediatly o

100000
```

```
In [46]: sp2 = Stel_Spectrum(T=100000, logg=6)
         sp2.print_ok2()
         sp2.print_ok() # ERROR : the print_ok was only included to an object, not

100000
```

```
    ---------------------------------------------------------------------------

    AttributeError                            Traceback (most recent call last)

    <ipython-input-46-9b9e27dbd5fd> in <module>()
      1 sp2 = Stel_Spectrum(T=100000, logg=6)
      2 sp2.print_ok2()
----> 3 sp2.print_ok() # ERROR : the print_ok was only included to an object, r


    AttributeError: 'Stel_Spectrum' object has no attribute 'print_ok'
```

```
In [47]: def print_T(self): # self could have been named otherwise
             print self.T
         sp1.print_T = print_T # adding to the object
         sp1.print_T() # ERROR: the object has no self reference
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-47-22b0360f0c63> in <module>()
      2     print self.T
      3 sp1.print_T = print_T # adding to the object
----> 4 sp1.print_T() # ERROR: the object has no self reference


    TypeError: print_T() takes exactly 1 argument (0 given)
```

```
In [48]: Stel_Spectrum.print_T = print_T # Adding to the class
         sp2 = Stel_Spectrum(T=100000, logg=5) # works immediatly
         sp2.print_T()

100000
```

```
In [49]: def print_T(self): # changing the definition of print_T
             print('T={}'.format(self.T))
         sp2.print_T() # does NOT affect the class nor the object

100000
```

```
In [50]: Stel_Spectrum.print_T = print_T # Adding to the class
         sp2.print_T() # now it changes the behaviour
```

```
T=100000
```

```
In [51]: # The same function can also be called passing the object. Good for testi
         print_T(sp2)
```

```
T=100000
```

The monkey patch is usefull for testing purpose. When everything is working fine, better to incorporate the method to the class definition.

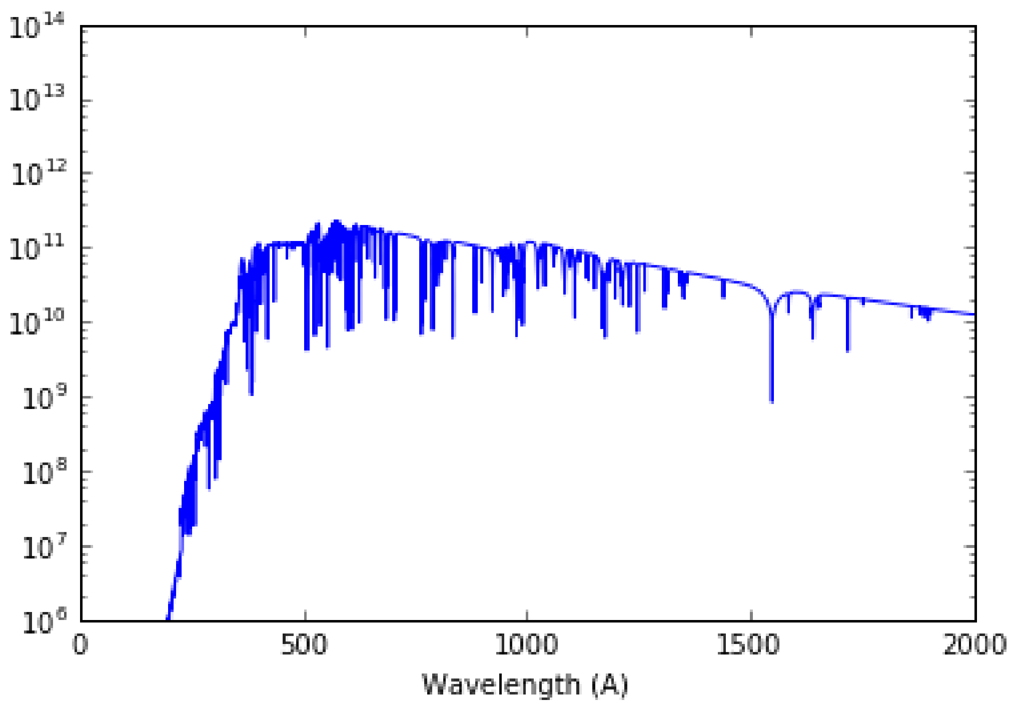
1.0.6 Class inheritance

This is very easy to create a new class from an existing one.

```
In [52]: class Stel_Sp2(Stel_Spectrum):

             def __init__(self, *args, **kwds):
                 super(Stel_Sp2, self).__init__(*args, **kwds)

             def print_logg(self):
                 print('logg = {}'.format(self.logg))
```

```
In [53]: print filename
         sp2 = Stel_Sp2(filename)
         sp2.plot_spr()
         sp2.print_logg()
```

```
0050000_7.00_33_50_02_15.bin_0.1.gz
logg = 7.0
```



```
In [54]: # One can even overwrite methods
         class Stel_Sp2(Stel_Spectrum):

             def __init__(self, *args, **kwds):
                 super(Stel_Sp2, self).__init__(*args, **kwds)

             def print_logg(self):
                 print('logg = {}'.format(self.logg))

             def print_info(self):
                 """
                 Print out new information
                 """
                 print('File: {}, T={}, logg={}'.format(filename, self.T, self.logg

         sp1 = Stel_Spectrum(T=100000, logg=5)
         sp2 = Stel_Sp2(T=100000, logg=5)
         sp1.print_info()
         sp2.print_info()

Filename: 0100000_5.00_33_50_02_15.bin_0.1.gz, number of points: 19951
File: 0050000_7.00_33_50_02_15.bin_0.1.gz, T=100000, logg=5
```

One can mix inheritances, using multiple parents to generate children (!). A lot of examples on the web. . .

1.0.7 Properties

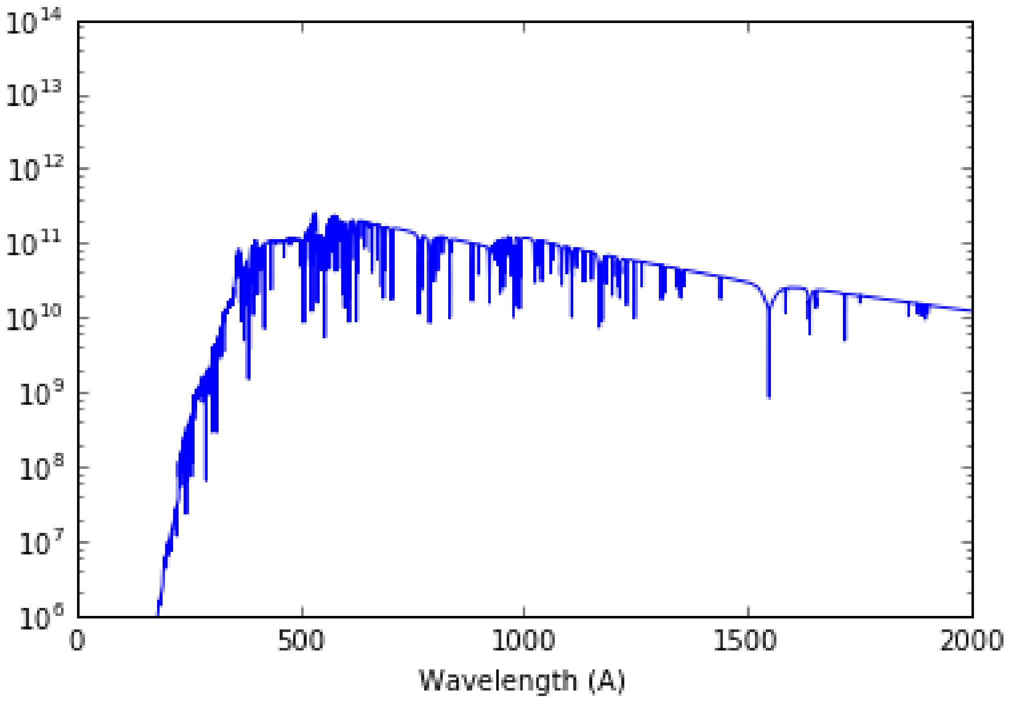
It is sometimes useful to have things that behave like attributes (print A.b, A.c = 2), but that call some routines. This is the goal of the properties.

For example here, we want the data to be updated if one change T or logg.

```
In [55]: class Stel_Spectrum(object):
             """
             This object downloads a file from http://astro.uni-tuebingen.de/~rauch
             and is able to make some plots.
             """

             spec_count = 0 # This attibute is at the level of the class, not of th
             def __init__(self, filename=None, T=None, logg=None, verbose=False):
                 """
                 Initialisation of the Stel_Spectrum object.
                 Parameter:
                     - filename
                     - T: temperature in K, e.g. 150000
                     - logg: e.g. 7.5
                     - verbose: if True, some info are printed out
                 The wl variable is an array of wavelengths in Angstrom.
                 The fl variable is the flux in erg/s/cm2/A
                 The variables T and logg are properties: changing them will reloa
                 """
                 self.verbose = verbose
                 if filename is None:
                     if T is not None and logg is not None:
                         self.__T = T # We need to initialize the hidden values, as
                         self.logg = logg
                         self.filename = '0{0:06.0f}_{1:.2f}_33_50_02_15.bin_0.1.gz
                     else:
                         raise TypeError("T and logg must be given")
                 else:
                     self.filename = filename
                     self.__T = float(filename.split('_')[0]) # We need to initiali
                     self.logg = float(filename.split('_')[1])
                 Stel_Spectrum.spec_count += 1
                 if self.verbose:
                     print('Instantiation done')

             def dlfile(self):
                 """
                 Downloading file if not already here. Put it in the current direct
                 """
```

```
        if not os.path.exists(self.filename):
            if self.verbose:
                print('Downloading {}'.format(self.filename))
            try:
                stel_file = urllib2.urlopen('http://astro.uni-tuebingen.de
                                            self.filename)
                output = open(self.filename,'wb')
                output.write(stel_file.read())
                output.close()
                self.file_found=True
            except:
                if self.verbose:
                    print('file {} not found'.format(self.filename))
                self.file_found=False
        else:
            if self.verbose:
                print('{} already on disk'.format(self.filename))
            self.file_found=True

    def read_data(self):
        """
        read the data from the file
        """
        if self.file_found:
            data = np.genfromtxt(self.filename, comments='*', names='wl, f
            self.fl = data['fl']
            self.wl = data['wl'] # in A
            self.fl /= 1e8 # F LAMBDA  GIVEN IN ERG/CM**2/SEC/CM -> erg/s/
            if self.verbose:
                print('Read data from {}'.format(self.filename))
        else:
            if self.verbose:
                print('file not found {}'.format(self.filename))
            self.wl = None
            self.fl = None

    def plot_spr(self, ax=None, *args, **kwargs):
        """
        Plot the spectrum.
        Parameter:
            - ax: an axis (optionnal). If Noe or absent, axis is created
            - any extra parameter is passed to ax.plot
        """
        if self.wl is None:
            print('No data to plot')
            return
        if ax is None:
            fig, ax = plt.subplots()
        ax.plot(self.wl, self.fl,
                label='T3={0:.0f}, logg={1}'.format(self.T/1e3, self.logg)
                *args, **kwargs) # Here are the transmissions of extra pa
        ax.set_yscale('log')
        ax.set_ylim(1e6, 1e14)
        ax.set_xlabel('Wavelength (A)')

    def get_integ(self):
        """
        Return the integral of Flambda over lambda, in erg/s/cm2
        """
        if self.wl is None:
            print('No data')
            return None
        return simps(self.fl, self.wl) # perform the integral

    def __getT(self):
        return self.__T

    def __setT(self, value):
        if not isinstance(value, (int, long, float)): # check the type of
            raise TypeError('T must be an integer or a float')
        if float(value) not in np.linspace(40000, 190000, 16): # check the
            raise ValueError('T value must be between 40000 and 190000K, 
        elif self.__T != value:
            self.__T = value
            self.filename = '0{0:06.0f}_{1:.2f}_33_50_02_15.bin_0.1.gz'.fo
            self.dlfile() # will download new data
            self.read_data() # will update the data

    def __delT(self):
        print('T is needed')

    T = property(__getT, __setT, __delT, "Stellar effective temperature")

    def __getlogg(self):
        return self.__logg

    def __setlogg(self, value):
        try:
            self.__logg
        except:
            self.__logg = -1
        if not isinstance(value, (int, long, float)):
            raise TypeError('logg must be an integer or a float')
        if float(value) not in (-1., 5., 6., 7. ,8., 9.):
            raise ValueError('Error, logg must be 6, 7, 8, or 9')
            self.__logg = None
```

```
                elif self.__logg != value:
                    self.__logg = value
                    self.filename = '0{0:06.0f}_{1:.2f}_33_50_02_15.bin_0.1.gz'.fo
                    self.dlfile() # will download new data
                    self.read_data() # will update the data

            def __dellogg(self):
                print('logg is needed')

            logg = property(__getlogg, __setlogg, __dellogg, "Stellar logg")

            def print_info(self):
                """
                Print out the filename and the number of points
                """
                print self.__repr__()

            def __repr__(self):
                """
                This is what is used when calling "print <obj>" or <obj> ENTER
                """
                if self.wl is None:
                    return'Filename: {0}, No data'.format(self.filename)
                else:
                    return'Filename: {0}, number of points: {1}'.format(self.filen

            def __del__(self):
                Stel_Spectrum.spec_count -= 1

In [56]: sp2 = Stel_Spectrum(T=50000, logg=6, verbose=True)
         print sp2.T
         sp2.plot_spr()

0050000_6.00_33_50_02_15.bin_0.1.gz already on disk
Read data from 0050000_6.00_33_50_02_15.bin_0.1.gz
Instantiation done
50000
```



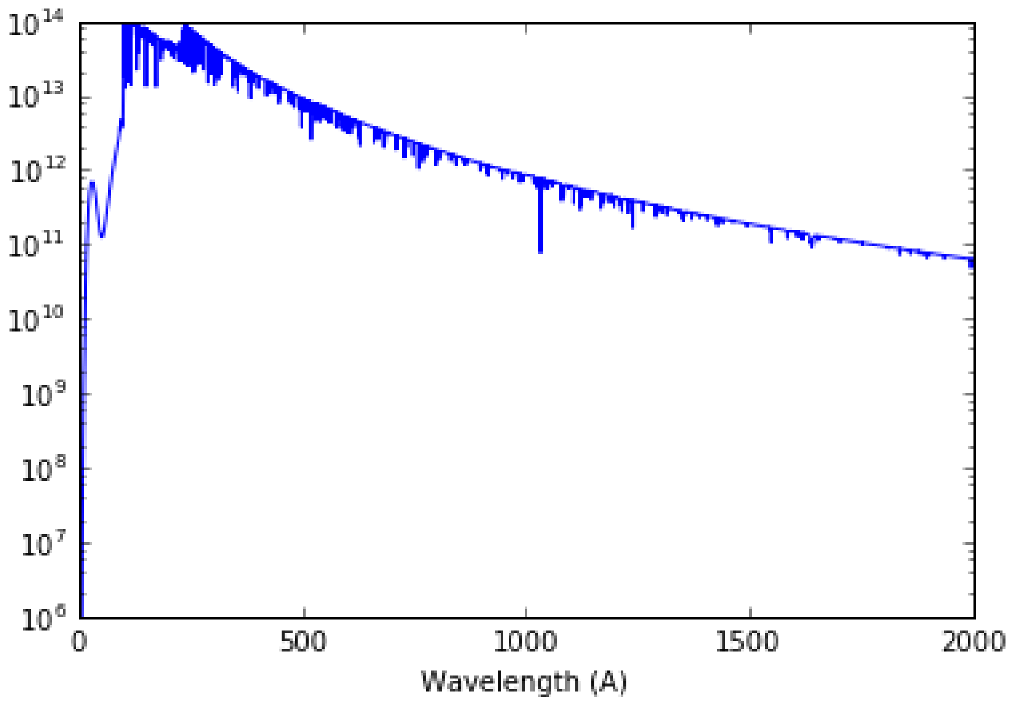
```
In [57]: # The properties are used to control type and values of the inputs
         sp2.T = 1800

        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-57-e0d29c1a86f4> in <module>()
          1 # The properties are used to control type and values of the inputs
    ----> 2 sp2.T = 1800


        <ipython-input-55-4f518246dde0> in __setT(self, value)
        109             raise TypeError('T must be an integer or a float')
        110         if float(value) not in np.linspace(40000, 190000, 16): # check
    --> 111             raise ValueError('T value must be between 40000 and 190000K
        112         elif self.__T != value:
        113             self.__T = value


        ValueError: T value must be between 40000 and 190000K, by 10000K steps


In [58]: sp2.logg = 'tralala'
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-58-cd5b2befd4fb> in <module>()
----> 1 sp2.logg = 'tralala'


    <ipython-input-55-4f518246dde0> in __setlogg(self, value)
    130                 self.__logg = -1
    131             if not isinstance(value, (int, long, float)):
--> 132                 raise TypeError('logg must be an integer or a float')
    133             if float(value) not in (-1., 5., 6., 7. ,8., 9.):
    134                 raise ValueError('Error, logg must be 6, 7, 8, or 9')


    TypeError: logg must be an integer or a float
```

```
In [59]: sp2.T = 180000
         sp2.logg = 7
         print sp2
         print sp2.T
```

```
0180000_6.00_33_50_02_15.bin_0.1.gz already on disk
Read data from 0180000_6.00_33_50_02_15.bin_0.1.gz
Downloading 0180000_7.00_33_50_02_15.bin_0.1.gz
Read data from 0180000_7.00_33_50_02_15.bin_0.1.gz
Filename: 0180000_7.00_33_50_02_15.bin_0.1.gz, number of points: 19951
180000
```

```
In [60]: sp2.plot_spr()
```



10^{14}
10^{13}
10^{12}
10^{11}
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
0
500
1000
1500
2000
Wavelength (A)

```
In [ ]:
```