CS1110 Spring 2015

Assignment 7: Due Wednesday May 6 at 11pm

You must work either on your own or with one partner. If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group.

You may discuss background issues and general solution strategies with others, but the programs you submit must be the work of just you (and your partner). We assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website. Any doubts that you have about "crossing the line" should be discussed with a member of the teaching staff before the deadline.

Objectives. Reading data from a file. Numpy 1D and 2D arrays and pylab graphics. Objects and classes. Creating images. Having an appreciation for the cost of energy.

1 Set-Up

Set up a folder `A7` by unzipping the file `A7.zip` that can be downloaded from the assignments page. It includes (a) a folder with 30 building energy use files, (b) a folder with 10 building jpegs, and (c) various modules. Do not fiddle with this set-up because (for simplicity) various codes have file path conventions that depend upon this organization. LET A7 BE THE CURRENT WORKING DIRECTORY WHENEVER YOU ARE WORKING ON THIS ASSIGNMENT.

2 The Energy Files

In the folder `EnergyData` you will find 30 `.txt` files that house energy consumption statistics for 10 campus buildings:

`Gates_E.txt`	`Gates_S.txt`	`Gates_C.txt`
`Upson_E.txt`	`Upson_S.txt`	`Upson_C.txt`
`Duffield_E.txt`	`Duffield_S.txt`	`Duffield_C.txt`
`Statler_E.txt`	`Statler_S.txt`	`Statler_C.txt`
`Uris_E.txt`	`Uris_S.txt`	`Uris_C.txt`
`UrisLibrary_E.txt`	`UrisLibrary_S.txt`	`UrisLibrary_C.txt`
`GoldwinSmith_E.txt`	`GoldwinSmith_S.txt`	`GoldwinSmith_C.txt`
`Sage_E.txt`	`Sage_S.txt`	`Sage_C.txt`
`Rockefeller_E.txt`	`Rockefeller_S.txt`	`Rockefeller_C.txt`
`Bailey_E.txt`	`Bailey_S.txt`	`Bailey_C.txt`

The "E" files house electricity consumption data, the "S" files house steam consumption data, and the "C" files house chilled water consumption data. Every 15 minutes throughout the year an energy consumption snapshot is taken of every building on campus. Each of the above files has $365 \times 24 \times 4 = 35040$ lines of energy consumption data. In this assignment you will be working with this data set performing computations and displaying results.

The first step is for you to get familiar with how the data is laid out in these files. Run the given demonstration module `ShowTheData.py`. Notice how it uses the `split` method to "take apart" file lines. Even though it just displays the first ten lines of the three Upson Hall energy files, `ShowTheData.py` points to some critical data layout features:

1. We only care about file lines that do NOT begin with the character "`#`". These are the *energy data lines* and there are 35040 of them in each file.
2. Column 0 of an energy data line houses the snapshot date in the form `dd-MMM-2014`.
3. Column 1 of an energy data line houses the snapshot clock time in the form `hh:mm:ss`.
4. The E and S files have just one column of energy consumption data.
5. The C files have six columns of energy consumption data. The last column houses the data we need in this assignment.
6. If an energy source is shut off when the snapshot is taken, then that is indicated with the character "`-`", not a zero.

3 Time stamps

The notion of a "time stamp" is important in what follows and it will be handy to have a single string representation that combines date and clock information[1]. Write a helper function `TimeStamp(date,clock)` that takes a date string `'dd-MMM-2014'` and a clock string `'hh:mm:ss'` and returns a time stamp string of the form `dd-MMM-2014-hh-mm`. Here are the first three time stamps in July:

```
01-Jul-2014-00-00
01-Jul-2014-00-15
01-Jul-2014-00-30
```

Set up a module `MyEnergy.py` and within it place your implementation of the function `TimeStamp`.

4 The Energy Class

Next, you will define a class `Energy` that is to be included in the `MyEnergy.py` module. It should have these attributes:

```
class Energy:
"""
  Name:  a string that is the name of the building
 Image:  a string that specifies the path of the building's jpeg image
E_rate:  a float that is the cost of electricity per unit of consumption
S_rate:  a float that is the cost of steam per unit of consumption
C_rate:  a float that is the cost of chilled water per unit of consumption
     A:  a 35040-by-3 numpy array that houses all the energy snapshots
TS_list:  a length-35040 list of time stamp strings
TS_dict:  a 35040-item time stamp index dictionary
 """
```

The constructor definition should have the form

```
def  __init__(self,BuildingName)
```

where `BuildingName` is a string that names the building. Thus, `M = Energy('Upson')` sets up an `Energy` object that encodes all the data associated with Upson Hall energy consumption.

The `Name` attribute is just the building name. The `Image` attribute is to specify the complete path of the jpeg for the building. The jpegs are in the folder `BuildingImages`. Thus, `BuildingImages/Upson.jpeg` is the complete path of the Upson Hall jpeg file.

We will use the same energy rates for each building and we will assume that rates are constant throughout the year with these values:

Electricity:	0.02
Steam:	0.006625
Chilled Water	0.04

Set up the `E_rate`, `S_rate`, `C_rate` attributes accordingly.

The `A` attribute is used to house the consumption data. It is a 35040-by-3 numpy array with these properties:

Column 0 houses the E data
Column 1 houses the S data
Column 2 houses the C data.

Setting up a column of `A` requires reading data from the appropriate energy file. Here are the steps that would apply for `A[:,0]`:

[1] In thinking about time stamps and the consumption stats that go along with it, assume that the consumption stats apply to the 15-minute period that begins at the moment specified by the time stamp.

1. Open the building's E-file. This requires putting together the file path, e.g.,

   `'EnergyData/Upson_E.txt'`.

   The building's name is an argument for the constructor so you have everything you need for the required concatenation.

2. Read the file line by line and maintain a counter `k` that keeps track the energy data lines. Take apart each energy data line using `split`, grab the electricity consumption string, convert it to a float, and store in `A[k,0]`. To be clear, the value of `A[k,0]` is the E consumption associated with the k-th time stamp of the year. Midnight Jan 1, 2014 is time stamp 0.

3. Close the file after this process is complete.

Repeat this data acquisition process for steam (which goes into `A[:,1]`) and chilled water (which goes into `A[:,2]`). Some tips:

- Where to "grab" the consumption data from a file line is discussed in §2.
- Do not build up `A` using `append`. Instead, use `A = zeros((35040,3))` and then "fill it up" as you read in the data.
- Don't forget to look for the "`-`" characters that signal "energy shut off". You will get an error if you try `float('-')`.
- If a file line begins with "`#`", then it is not an energy data line and it should be ignored.

The `TS_list` attribute is a list of strings, each of which is a time stamp. Its construction can "live" off of any of the three file reads that are used to set up `A`. That is because an energy data line in an E-file or an S-file or a C-file always begins with date and clock information. So, for example, you can pull off the date and clock information every time you read a line from the E-file. `TS_list` can be built up using repeated appending. Use the `TimeStamp` helper function that you developed in §3.

The `TS_dict` attribute is more complicated and it can be ignored for the time being. It has to do with the fact that daylight savings time creates problems when it comes to using time stamps to acquire information out of the `A` array.

The test modules `CheckConstructor` and `CheckMonthStarts` can be used to confirm that things are (probably) working.

5 Annual Costs

Write a method `annualBill(self)` for the `Energy` class that returns three floats `E`, `S`, and `C`. The value of `E` should be the total cost of electricity for the building represented by `self`. The value of `S` should be the total cost of steam for the building represented by `self`. The value of `C` should be the total cost of chilled water for the building represented by `self`. The test module `CheckAnnualBill.py` can be used to check things out.

These calculations involve summing values in the columns of `A` and multiplying by the appropriate rate. For example, if you sum the values in column 1 of `A`, you get the annual consumption of steam. By multiplying that number by `S_rate` you get the annual bill for steam.

6 Monthly Costs

Write a method `monthlyBill(self)` for the `Energy` class that returns three, 1D, length-12 numpy arrays `E`, `S`, and `C`. The values in `E` should be the monthly cost of electricity for the building represented by `self`. Thus, `E[4]` would be the May electric bill for the building represented by `self`. Ditto for the values in `S` and `C`.

To implement this method you should make use of the lists

```
StartIndices = [0,2976,5664,8636,11516,14492,17372,20348,23324,26204,29180,32064]

EndIndices   = [2975,5663,8635,11515,14491,17371,20347,23323,26203,29179,32063,35039]
```

These lists tell us (for example) that the energy data for May sits in

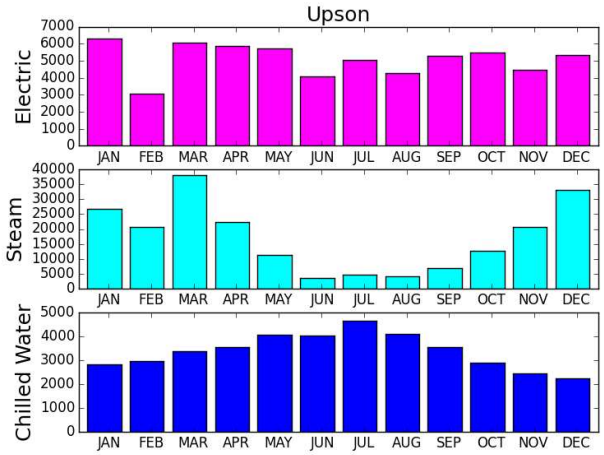
`A[StartIndices[4]:EndIndices[4]+1,:]`.

(Note that `StartIndices` and `EndIndices` are used in `CheckMonthStarts.py` and so you can cut and paste them into your implementation `monthlyBill`.)

The test module `CheckMonthlyBill.py` can be used to check things out.

7 A Bar Plot

Complete the module `ShowMonthlyBill.py` so that after it prompts the user for a building name, creates an energy object for the building named, calls the method `monthlyBill`, and then produces three bar plots that displays the values in the the returned lists. Make it nice like this:



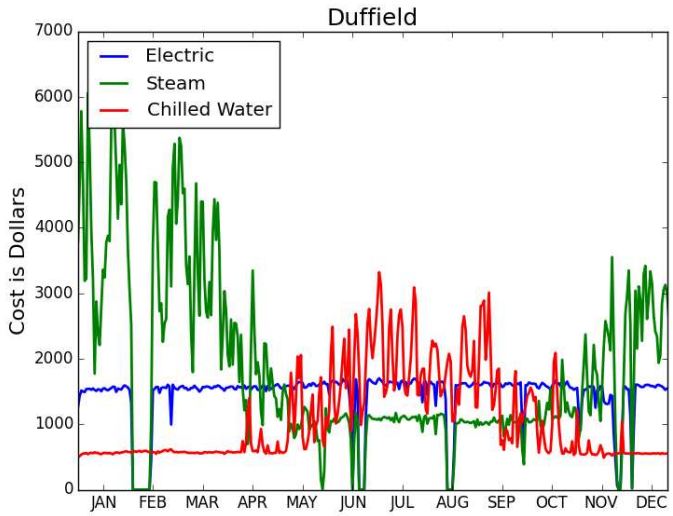
8 Daily Costs

Implement a method `dailyBill(self)` for the `Energy` class that returns three, 1D, length-365 numpy arrays `E`, `S`, and `C`. The values in `E` should be the daily cost of electricity for the building represented by `self`. Thus, `E[100]` would be the electric bill for the building represented by `self` on day 100. Ditto for the values in `S` and `C`.

Note that there are 96 time stamps each day. Thus, the energy consumption data for the kth day of the year is situated in rows $96k$ to $96(k+1)-1$ in the `A` array. (Note: January 1 is day 0.)

9 A Graph

Complete the module `ShowDailyBill.py` so that after it prompts the user for a building name, creates an energy object for the building named, calls the method `dailyBill`, and then produces a plot like this:



10 The Problem With Daylight Savings

Regarding the attribute `TS_dict` that is to be part of the `Energy` class, we require it to be a dictionary whose keys are time stamps and whose values are integers that indicate the row in `A` that houses the corresponding energy consumption data. Thus, if `tau` is a time stamp string and `k = TS_dict[tau]`, then `A[k,0]`, `A[k,1]`, and `A[k,2]` would house the E, S, and C data associated with that particular time stamp. At first, this looks like an easy addition to the constructor:

```
self.TS_dict = {}
k = 0
for t in self.TS_list:
    self.TS_dict[t] = k
```

Unfortunately, Daylight Savings Time creates a problem with this. To see why, look at the time stamps when we switch to Daylight Savings Time in March,

```
TS_list[6438] = 09-Mar-2014-01-30
TS_list[6439] = 09-Mar-2014-01-45
TS_list[6440] = 09-Mar-2014-03-00
TS_list[6441] = 09-Mar-2014-03-15
```

and when we go back to standard time in November,

```
TS_list[29279] = 02-Nov-2014-00-45
TS_list[29280] = 02-Nov-2014-01-00
TS_list[29281] = 02-Nov-2014-01-15
TS_list[29282] = 02-Nov-2014-01-30
TS_list[29283] = 02-Nov-2014-01-45
TS_list[29284] = 02-Nov-2014-01-00
TS_list[29285] = 02-Nov-2014-01-15
TS_list[29286] = 02-Nov-2014-01-30
TS_list[29287] = 02-Nov-2014-01-45
TS_list[29288] = 02-Nov-2014-02-00
TS_list[29289] = 02-Nov-2014-02-15
```

Notice that four time stamps are "missing" in March and we have four repeat time stamps in November.

This can be rectified with a bit of string slicing and list concatenation. First, you need to produce a "standard time" list of time stamp strings from `s = self.TS_list`. It is simply the list concatenation of

(a) all the standard times up to the March switch time, i.e., `s[:6440]`
(b) the list of four "missing" time stamps
(c) all the daylight savings time time stamps, i.e., `s[6440:29284]`
(d) all the standard times that follow the November switch time, i.e., `s[29288:]`

Using these tips, complete the development of the constructor by setting up the `TS_dict` attribute.

11 Arbitrary Energy Bills

Implement a method `arbitraryBill(self,T1,T2)` for the `Energy` class that returns three floats `E`, `S`, and `C`. The value in `E` should be the cost of electricity for the building represented by `self` from time stamp `T1` through up to time stamp `T2`. Likewise for the values returned in `S` and `C`. Make effective use of `TS_dict`.

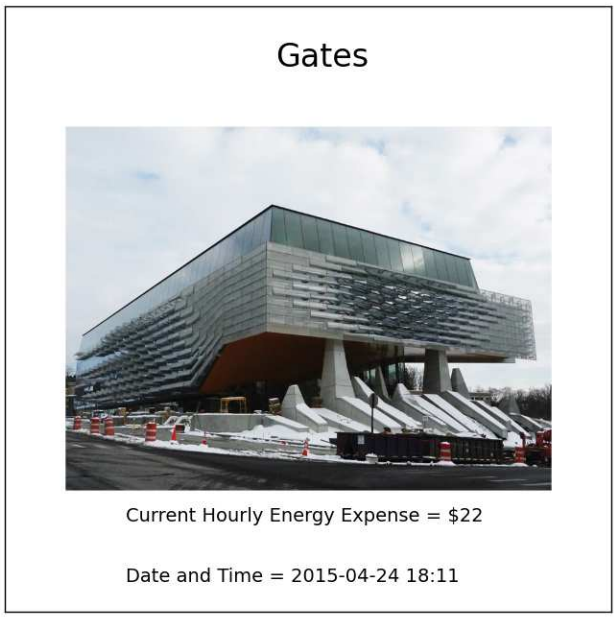
As an example,

```
M = Energy('Gates')
(E,S,C) = M.arbitraryBill('15-May-2014-08-00','16-May-2014-11-45')
x = E+S+C
```

would assign to `x` the total energy cost of running Gates Hall from 8AM May 15 up to noon May 16.

12 Online Snapshot

It is possible to get an "instantaneous" reading of energy consumption of any building on campus using a web service that is provided by the Facilities Department. It is also possible to write Python code that can "read" information off of a webpage and then perform computations with that data. In this last part of the assignment we give you the tools to do just that and to produce a graphic like this:



We have provided a template module `EnergyNow.py` that you must complete using these facts

- We added a graphics procedure `MyText` to `simpleGraphicsE,py` that can be used to display text in the figure window. The call

  ```
  MyText(x,y,s,c,fs)
  ```

  displays the string `s` at location (x, y) with color `c` and fontsize `fs`. We have also added a procedure `DrawImage` that can be used to place a jpeg in the figure window. The call

  ```
  DrawImage(p,x,y,w,h)
  ```

  positions the jpeg with path `p` in the figure window with lower left corner at (x, y). The image will be shaped to have width `w` and height `h`.

- The modules `datetime.py` and `time.py`can be used to get the current date and time. In particular

  ```
  datetime.datetime.now()
  ```

  returns a string with that information.

13 What to Submit to CMS and Scoring

You are to submit five files to CMS:

1. The module `MyEnergy`. It will include

   (a) The helper function `TimeStamp`

   (b) The class `Energy` that will include its constructor and the methods `annualBill`, `monthlyBill`, `dailyBill`, and `arbitraryBill`.

2. The completed module `ShowMonthlyBill.py`.

3. The completed module `ShowDailyBill.py`.

4. The completed module `EnergyNow.py`.

5. A file `MyImage.png` file that is the image YOU produced using `EnergyNow`. To get this file, just click on the save image icon on the figure window.

Scoring:

`TimeStamp`	1 pt
`Constructor`	20 pts
`annualBill`	2 pts
`monthlyBill`	3 pts
`dailyBill`	3 pts
`arbitraryBill`	3 pts
`ShowMonthlyBill.py`	5 pts
`ShowDailyBill.py`	5 pts
`EnergyNow.py`	3 pts
`MyImage.png`	5 pts