CS 2316 **Individual Homework 5 – Factory**
Out of 100 points

Files to submit: **1. HW5.py**

Due before 11:55pm on Thursday June 13th.

<u>This is an INDIVIDUAL Assignment:</u>
Collaboration at a reasonable level will not result in substantially similar code. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the lecturer. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. You should not exchange code or write code for others.

For Help:
- TA Helpdesk – Schedule posted on class website.
- Email TA's or use T-Square Forums

Notes:
- **Don't forget to include the required comments and collaboration statement (as outlined on the course syllabus).**
- ***Do not wait until the last minute*** *to do this assignment in case you run into* problems.
- **Read the entire specifications document before starting this assignment.**

Introduction
In this assignment, you will be tasked with reading data from files in CSV format. You will have to find the maximum of various rows and columns of numerical data and display that information on a GUI. Your GUI will also allow the user to select the file from which you should read data.

You will keep the GUI functions of your homework separated from the functions that perform the actual work. The GUI functions will call the "backend" functions that perform the actual work and then display the results. This separation of concerns allows you to change either the GUI or the backend without needing to modify the other. o

Your ENTIRE program must be built as a single Python class. Name the class HW5. At the very end of your code file, you will start the TK windowing system and instantiate your object. That may look something like this:
```
rootWin = Tk()
app = HW5( rootWin )
rootWin.mainloop()
```

Break your code up into several manageable methods as outlined below. You must use this suggested program organization and naming scheme, as we may auto-grade the "backend" functionality (readData, convertData, findSlowDay, findSlowStation) and need to know the exact name of the methods to call and the input/output of these functions.

The breakdown of functionality within your class should be in the following methods:

1. __init__
2. clicked
3. readData
4. convertData
5. findSlowDay
6. findBusyStation

File Format Information

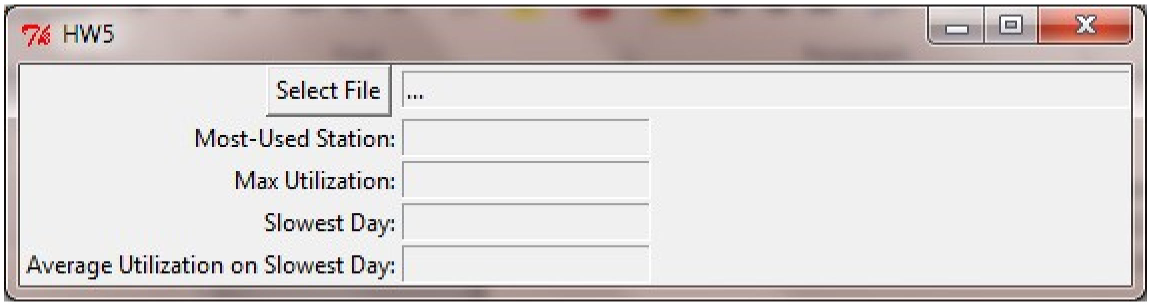
For this assignment, you will be given two comma separated value (CSV) files. These files will have different numbers of rows/columns, but the format will generally be the same. Each row starts with a day or date, and each column starts with the name of a station in the factory. The other values in the table are numbers representing what fraction of the day each station was used. See the factory1.csv and factory2.csv files for examples.

Your program must work with and produce the correct answers for any similarly formatted data file, regardless of size. You are allowed to use the csv module to read this data, or you may write your own function to read the data.

Here is what should be in each of your methods:

__init__(self,rootWin)

This method will be called automatically when your object is instantiated. It is responsible for setting up your GUI. It should create labels and text Entry widgets, along with a button to produce a GUI that looks like the following:

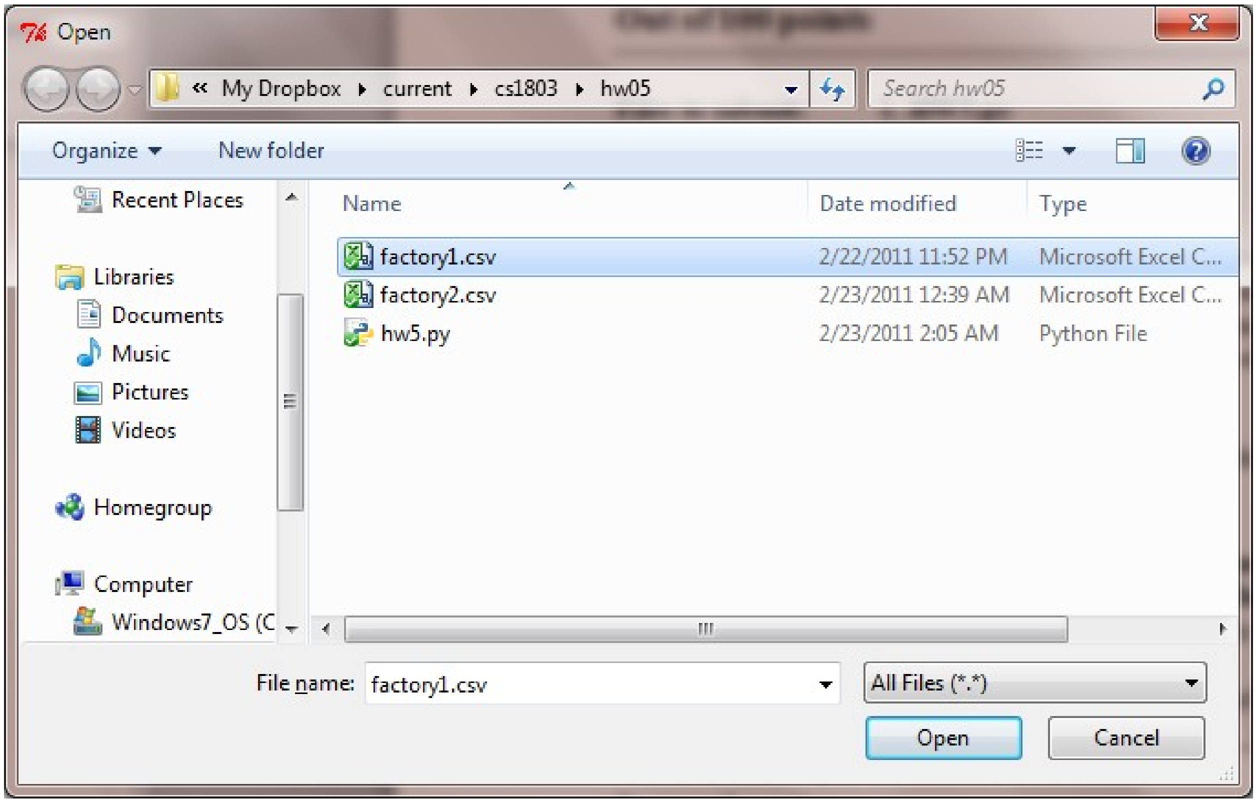


Note that the labels and text entry widgets are carefully arranged. The window is titled. The labels are right ("East") justified. The "Select File" button is close to the filename entry but has some padding space in between. The filename display text entry widget starts out with "..." in it, and is wide enough to show plenty of the file path (its width is 60). We suggest you use a grid layout to achieve this effect, although if you are able to do so by using multiple frames and the "pack" layout manager you may use that instead.

Note that all of the text entry widgets are set to "readonly" at this point.

The "Select File" button should trigger (call) the clicked() method.

clicked(self)



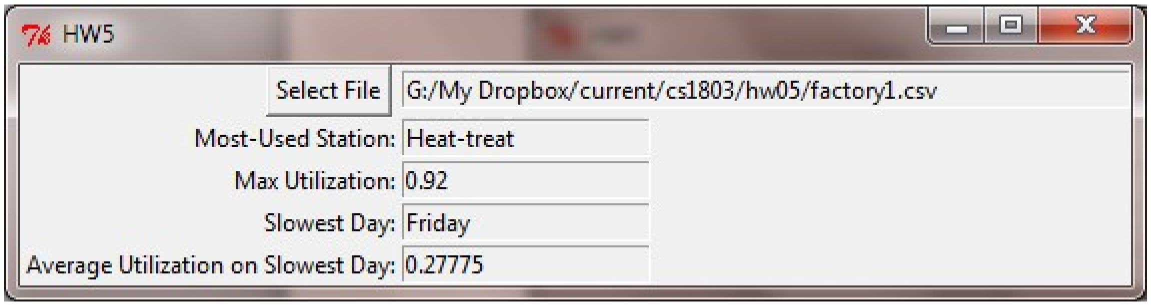
Name	Date modified	Type
factory1.csv	2/22/2011 11:52 PM	Microsoft Excel C...
factory2.csv	2/23/2011 12:39 AM	Microsoft Excel C...
hw5.py	2/23/2011 2:05 AM	Python File

File name: factory1.csv
All Files (*.*)

When the user clicks the "Select File" button, it executes the clicked() method. This method will present the user with a file open dialog box. If the user clicks "cancel" your program must gracefully wait for the user to again click the "Select a File" button. If the user actually selects a file and clicks "OK", the clicked method will do a lot more work.

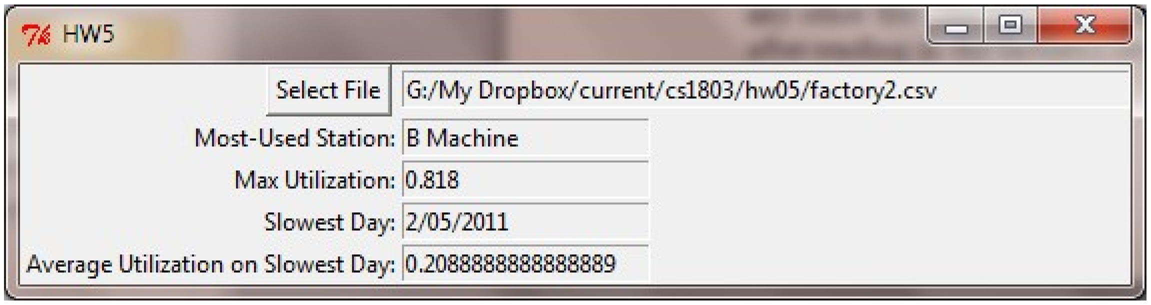
Specifically, it must cause the CSV data to be read in and stored, the textual representation of numbers converted to floats, the sum of rows and the average of columns calculated, and the appropriate data displayed to the screen. It will use the readData, convertData, findSlowDay and findBusyStation functions to do most of the heavy lifting for it.

Here is an example of the data filled in on the GUI after reading the factory1.csv file:



G:/My Dropbox/current/cs1803/hw05/factory1.csv
Most-Used Station: Heat-treat
Max Utilization: 0.92
Slowest Day: Friday
Average Utilization on Slowest Day: 0.27775

After loading one file, the user must be able to select the "Select File" button and load in any other file. The GUI must update itself appropriately. Here is an example of the GUI after reading in the factory2.csv file:



G:/My Dropbox/current/cs1803/hw05/factory2.csv

Most-Used Station: B Machine

Max Utilization: 0.818

Slowest Day: 2/05/2011

Average Utilization on Slowest Day: 0.2088888888888889

Note that to change an Entry widget that has been set to "readonly", you must first set the status to NORMAL, cut the existing text out, insert new text, and then return the status to "readonly". (You may want to create an additional method to take care of this.) We use Entry widgets so that the user can copy and paste the data out of our program.

readData(self, fileName)

This function will read in CSV data from the file name passed in as a string. You may assume that the user will only select "valid" CSV files, and do not have to do error checking. You may use the CSV module if you would like. Your readData function should return a list where each item in the list is a list of the data items in one row of the CSV File. If an entry has leading or trailing spaces, you *should strip the extra spaces before storing it*. (This will prevent leading spaces from showing up in your GUI.)

For example, if the CSV data file looks like this:

```
Date, Assembly,  Heat-treat,  Painting, Inspection
Monday, 0.352, 0.92, 0.182, 0.255
```

Your returned list would look like this:

```
 [ ['Date', 'Assembly', 'Heat-treat', 'Painting', 'Inspection'] ,
[ 'Monday', '0.352', '0.92', '0.182', '0.255' ] ]
```

convertData(self, listOfData)

This method will go through the listOfData (as returned by the readData function) , and convert all of the numbers from string representations to floats. This will allow you to do math on the numbers in the CSV files. You should replace the strings with floats in-place in the data structure. Note that you cannot (and should not try to) change the column headers or the dates in the first column to floats. You should return the updated list. Continuing the example above, the output of the convertData function would be:

```
[ ['Date', 'Assembly', 'Heat-treat', 'Painting', 'Inspection'] ,
[ 'Monday', 0.352, 0.92, 0.182, 0.255 ] ]
```

findSlowDay(self, listOfData)

This method will find the “slowest day” by averaging all utilization (all numbers in a particular row) for each day, and returning the day label (Entry in the first column of the row) and average of the remaining columns for the row with the smallest average utilization as a tuple. Note that the very first row of data is a header, so you should only examine the 2nd and subsequent rows. Because our example data only has one day's worth of data ("Monday") the return value from findSlowDay would be:

```
( 'Monday', 0.427250)
```

findBusyStation(self, listOfData)

This function will find the maximum utilization for each station. It finds the maximum of all numbers in a column and finds the column with the highest maximum. It then returns the top entry in the column (the data label, for the station) and the column's maximum as a tuple. In our sample data, on Monday, the Heat-treat station had the largest utilization, so findBusyStation will return:

```
( 'Heat-Treat', 0.92)
```

Grading:

You will earn points as follows for each piece of functionality that works correctly according to the specifications.

The GUI	**40**
Has all labels, text entry fields, and "Select File" button	10
Properly allows the user to specify a file to load, and displays the file name when selected.	10
Text Entry areas are "readonly", allowing selection and copying, but not modification by the user	10
The data displayed is updated correctly when each file is loaded	10
Data Loading / Calculations	**60**
Properly loads data from the CSV file	10
Correctly converts the string representation of numbers into floats that can be used for calculations	10
Correctly calculates the average of each row	10
Correctly find/displays the label and average for the row with the lowest average	10
Correctly calculates the maximum of each column	10
Correctly finds/displays the label/maximum for the column with the largest maximum	10