Project 11: Classes

This assignment is worth 55 points (5.5% of the course grade) and must be completed and turned in before 11:59pm on 4/23/2018.

Overview

In this assignment you will practice classes and file handling.

Background

In this project, you design two classes to help with reading and writing CSV files. A CSV file is a comma-separated values file that is used to store data. We provide College_Scorecard.csv which includes data from 1996 through 2016 for all undergraduate degree-granting institutions of higher education. The data about the institution will help the students to make decision about the institution for their higher education such as student completion, debt and repayment, earnings, and more.

Project Specifications

There are two classes: a CsvWorker class that represents a whole csv spreadsheet, and a Cell class that represents one cell in that spreadsheet. That is, the CsvWorker class contains a bunch of Cell objects.

Your solution must include the following classes with their methods:

1. **`Cell(object):`** This class defines a cell component of a CSV and has four attributes: a reference to a CsvWorker instance, a column number, a value, and its alignment for printing. These 4 attributes should be private. The CsvWorker instance describes that csv spreadsheet that this cell is in. You should implement the following methods:
    a. `__init__(self, csv_worker, value, column, alignment):` This method initializes the object and its 4 private attributes. The default value of the parameters should be `None`, empty string, 0 and `'^'`, respectively.
    b. `__str__(self):` This method builds and returns a formatted string (to be used for printing the value). To build the string we will need to find the cell width from the csv worker instance by calling the worker's `get_width` method and passing this cell's `column` number attribute (The CsvWorker stores the widths for each column which is explained in the next section.) See Notes below for information on how to use width and alignment as parameters in formatting a string. Some values may need to be formatted specially (Hint: skip the special formatting for your first version). For example, a column of fractions could be formatted as a

percentage. Special functions are also stored in CsvWorker for the formatting of each column. If a function is defined for the column of a given cell, __str__ should call that function to get a pre-formatted string. The function should not crash when trying to format a title cell or an empty cell that cannot be converted to a floating point, it should return the original value unmodified.

c. `__repr__(self):` This method should return a shell representation of a Cell object. No need to do anything special, just call `__str__` and return that value.
d. `set_align(self, align):` This method takes in an alignment string, left/right/center. (left: `'<'`, center: `'^'`, right: `'>'`) This method redefines the cell's alignment attribute from the parameter.
e. `get_align(self):` This method returns the object's alignment.
f. `set_value(self, value):` This method takes in a new value and redefines its value attribute.
g. `get_value(self):` This method returns the object's value.

2. **`CsvWorker(object):`** This class is the actual CSV Worker class that will take a csv file as input and process it, then stores its data for output.
    a. `__init__(self, fp=None):` This initializes the worker class. The object can be instantiated with a file pointer to process it, or a file can be processed after instantiating the object—in both cases using the `read_file` method described next. The attributes of this class are columns, rows, data, widths, and special functions assigned to format each column. Columns and rows are just integer values representing the size of the csv table. The `__data` attribute will be a list of lists containing all the data in the CSV. How you organize it is up to you, but a format of list of rows is recommended, such that `__data[row][column]` will give an individual cell. The attribute `__widths` is a list of integers defining the formatted width of each column. The attribute `__special` is a list of functions defining the special formatting functions for each column. Initialize ints to zero and lists to be empty.
    b. `read_file(self, fp):` This method takes a file object, `fp`, which is for a CSV file and iterates over it, filling in all the attributes along the way. The `__data` should store the data in all the cells. If the value in the file is "Null", the value in the cell should be the empty string. The `__widths` should store the width of the widest element in each column. Determine `__rows` and `__columns` values from the number of rows and columns in the file. Note that all row in the csv file may not have the same number of values, i.e. not the same number of columns. Be sure that you account for this. (Hint: for your first version assume that all rows have the same number of columns and add logic to account for different columns later.) All

values in `__special` should be set to `None`. Be sure to close your file once you're done with it.

c. `__getitem__(self, index):` This method overloads the [] operator to allow you to access values in `__data`. We provide it for you.
d. `__setitem__(self, index, value):` This method overloads the [] operator to allow you to set values in `__data`. We provide it for you.
e. `__str__(self):` Overload the `__str__` method to allow the class to convert its data to a string (so it can be called by print). Just convert the cells in each row in `__data` to strings, append a carriage return (`'\n'`) to each row, and return the resulting string. Note that for an `item` in `__data`, `str(item)` will call `__str__` for that cell so you get a formatted string for that cell.
f. `__repr__(self):` This method should return a shell representation of a CsvWorker object. No need to do anything special, just call `__str__` and return that value.
g. `limited_str(self, limit):` This method does a similar job to `__str__`, but it should print maximum of `limit` number of lines. CSV files can get quite large, so this allows you to see just a small portion of your data.
h. `remove_row(self, index):` This method allows you to remove a row from the `__data` at the particular index. (Hint: you can `pop` a whole row.)
i. `set_width(self, index, width):` This method sets width of the column of `__data` at `index`. The width information of every column is stored in the `__widths` attribute so you can simply modify the value at that particular `index`.
j. `get_width(self, index):` This method returns information about the width of any column by specifying its `index` in `__widths`.
k. `set_special(self, column, special):` This method allows you to link a special formatting function to the column. In this project, there are two special functions: percentage and currency that convert values to those particular formats. We provide this method.
l. `get_special(self, column):` This function returns the special formatting function assigned to a column. We provide this method.
m. `set_alignment(self, column, align):` This method sets the alignment of the specified column by providing a string value such as left/right/center. This method calls the `set_align(alignment)` method of the Cell object (note that you need to set the alignment for every cell in the specified column). If the alignment is not one of the 3 valid alignments '<', '^', '>', this method should raise `TypeError`.
n. `get_columns(self):` This method returns number of the columns in the CsvWorker object (i.e. returns that attribute).

o. `get_rows(self):` This method returns number of the rows in the CsvWorker object (i.e. returns that attribute).
p. `minimize_table(self, columns_list):` This function returns a new CsvWorker object which is a minimized version of the original CsvWorker. The new object only contains the columns specified in the `columns_list` parameter (which is a list of colum indices). Hint: create a new CsvWorker instance, and for each row append new cell instances whose values are initialized with values from the original CsvWorker instance.
q. `write_csv(self, filename, limit = None):` This method writes the data into a CSV file named by the `filename` parameter. Only the values are written, i.e. no formatting. The `limit` parameter optionally limits the number of rows to be written in the CSV file. Remember to close the file.
r. `write_table(self, filename, limit = None):` This method writes the data into a tabular formatted text file named `filename`. Number of the rows to be written is controlled by `limit` parameter. Hint: for this method you can simply open a file, write using the `limited_str` method described above, and close the file.
s. `minimum(self, column):` This method returns the cell with minimum value of a `column` . You can only find the minimum of cells that are numbers so skip any values that are not numbers. Hint: `try` to convert the value to a float and if a `ValueError` is raised, simply `continue` to the next cell.
t. `maximum(self, column):` This method returns the cell with maximum value of a `column`. You can only find the maximum of cells that are numbers so skip any values that are not numbers. Hint: `try` to convert the value to a float and if a `ValueError` is raised, simply `continue` to the next cell.

You need to implement of the following functions:

1. `open_file():` This function prompts the user to enter a filename. The program will try to open a CSV file. An error message should be printed if the file cannot be opened. This function will loop until it receives proper input and successfully opens the file. It returns a file pointer.
2. `percentage(value):` The value parameter is a float. This method will convert the value to string and convert it into percentage format with one decimal place. Ex: 3.443 to 3.4%. If value is not a float, simply return the value unchanged (Hint: `try` to convert value to a float; if you get a `ValueError`, simply return the value.) Hint: there is a percentage string format.
3. `currency(value):` The value parameter is a float. This method will convert value to a string and convert it into currency format with two decimal places. Ex: 3.443 to $3.44.

If value is not a float, simply return the value unchanged (Hint: `try` to convert value to a float; if you get a `ValueError`, simply return the value.)

4. `main():` We provide this function. There are several tasks to complete for the main implementation of the function.
    a. Instantiate a CsvWorker with the input file and minimize it to contain columns INSTNM, STABBR, ACTMMID, PCIP14, MD_EARN_WNE_P10, GRAD_DEBT_MDN_SUPP, and C150_4_POOLED_SUPP. These are the column numbers: 3, 5, 40, 55, 116, 118, and 122 (zero indexed). The new CsvWorker instantiated will have 7 columns.
    b. Percentage columns should be formatted as percentages, and currency columns should be formatted as currency (USD).
    c. After minimizing and formatting, the data should be written to "output.txt" and "output.csv" using appropriate methods.
    d. Here are the titles for each column, to clarify the data:

INSTNM	Institution name
STABBR	State code
ACTMMID	Median ACT composite score
PCIP14	Percentage of graduates receiving engineering degrees
MD_EARN_WNE_P10	Median earnings 10 years after entry
GRAD_DEBT_MDN_SUPP	Median debt after graduation
C150_4_POOLED_SUPP	Completion rate

Notes:

In the Strings section we provided a link to a string formatting summary site which has a section on named placeholders (https://pyformat.info/#named_placeholders). That shows how you can use parameters for width and alignment. For example. If `some_value = 4`, `AAA = '>'` and `WWW = 10`, you can print a formatted string with parameterized alignment and width. Try this in the Python shell.

```
S = "{:{align}{width}}".format(some_value, align=AAA, width=WWW)
print(S)
```

Deliverables:

The deliverable for this assignment is the following file:

proj11.py – the source code for your Python classes and functions.

Be sure to use the specified file name and to submit it for grading via Mimir before the project deadline.

Grading Rubrics

General Requirements:

__0__ (5 pts) Following all the coding standards

Implementation:

__0__ (3 pts) open_file function (no Mimir test)

__0__ (3 pts) percentage function

__0__ (3 pts) currency function

__0__ (10 pts) Testing Cell class

__0__ (15 pts) Testing CsvWorker class

__0__ (8 pts) Test 1

__0__ (8 pts) Test 2 (hidden test)