CS 1410: Gas Mileage

Introduction

With gas prices constantly fluctuating and consumers being more conscientious of environmental concerns, fuel economy is becoming a higher priority for many. Nevertheless, the vast majority of drivers don't know how to accurately measure their vehicle's actual fuel economy, which most of us in the United States measure in miles per gallon. But with a little bit of programming help, it's easy to calculate and track gas mileage over time.

Assignment

Your program will gather and record information about a vehicle's mileage and gas consumption, and then use this information to calculate the vehicle's fuel economy in miles per gallon. Your program will feature a very simple menu that allows the user to perform any one of four operations, one at a time, repeatedly until the user chooses to quit the program. When prompted, the user may select one of the four operations by inputting a unique letter corresponding to one of the operations. The four operations (and their unique letters) are:

1. **[r] Record Gas Consumption:** This operation allows the user to record gas consumption when they fill their vehicle with gas. It asks the user for three pieces of information: the date, the number of miles traveled since the vehicle was last filled with gas, and the number of gallons of gas that was just added to the vehicle. You should store each of these values into a dictionary that contains three keys (one key for each of the three values), and then save the dictionary by appending it to a list.

2. **[l] List Mileage History:** This operation does not ask the user for any additional information. Your program will print a list of all recorded gas consumption entries, one per line. Each line should contain the date, the number of miles traveled, and the number of gallons added. Also print the gas mileage (miles per gallon) for this entry, calculated using the number of miles traveled and the number of gallons added (from this entry only).

3. **[c] Calculate Gas Mileage:** This operation does not ask the user for any additional information. Your program will calculate the average gas mileage from all recorded gas consumption entries and print the average gas mileage on a single line. If no gas consumption entries have been recorded, print a message that asks the user to first record their gas consumption.

4. **[q] Quit:** When this operation is selected, your program will print a friendly goodbye message to the user and then terminate.

5. **Something else?** If the user enters a different letter or phrase, print a message informing the user that this option is invalid, and allow them to try again.

Extra Challenges

- Because this assignment does not require you to save the gas consumption records to a file, you may assume that all records will be lost when the program quits. However, wouldn't it be useful if the records were *not* lost? Consider upgrading your program to write all records to a file just before the program quits, and then also load any records from the same file when the program is started again.

Hints

- Before starting, practice using dictionaries and lists. You'll need to know how to create a dictionary, set a value by key, retrieve a value by key, append a dictionary to a list, and iterate over a list of dictionaries.

- In order to calculate gas mileage, you should store the number of miles traveled and the number of gallons added as float values. Here is a simple way to convert a string or integer value to a float value: `gallons = float(gallons)`

Sample

Program execution:

```
What would you like to do?
[r] Record Gas Consumption
[l] List Mileage History
[c] Calculate Gas Mileage
[q] Quit
Enter an option: l
You first need to record your gas consumption!

What would you like to do?
[r] Record Gas Consumption
[l] List Mileage History
[c] Calculate Gas Mileage
[q] Quit
Enter an option: c
You first need to record your gas consumption!

What would you like to do?
[r] Record Gas Consumption
[l] List Mileage History
[c] Calculate Gas Mileage
[q] Quit
Enter an option: r
What is the date? [I hit enter on the keyboard]
What is the date?    1/1/17 [I add spaces before the date]
How many miles did you drive since last filling up? twelve
Please enter a number.
How many miles did you drive since last filling up? -17
Please enter a number greater than zero.
How many miles did you drive since last filling up? 0
Please enter a number greater than zero.
How many miles did you drive since last filling up? 300
How many gallons of gas did you add to your tank? zero
Please enter a number.
How many gallons of gas did you add to your tank? 0
Please enter a number greater than zero.
How many gallons of gas did you add to your tank? -1
Please enter a number greater than zero.
How many gallons of gas did you add to your tank? 10.0
Saved!

What would you like to do?
[r] Record Gas Consumption
[l] List Mileage History
[c] Calculate Gas Mileage
[q] Quit
Enter an option: r
What is the date? 1/25/17
How many miles did you drive since last filling up? 200
How many gallons of gas did you add to your tank? 8
Saved!

What would you like to do?
[r] Record Gas Consumption
[l] List Mileage History
[c] Calculate Gas Mileage
[q] Quit
Enter an option: l
On 1/1/17: 300.0 miles traveled using 10.0 gallons. Gas mileage: 30.0 MPG
On 1/25/17: 200.0 miles traveled using 8.0 gallons. Gas mileage: 25.0 MPG

What would you like to do?
[r] Record Gas Consumption
[l] List Mileage History
[c] Calculate Gas Mileage
[q] Quit
Enter an option: c
Average gas mileage: 27.77777777777778 MPG

What would you like to do?
```

```
[r] Record Gas Consumption
[l] List Mileage History
[c] Calculate Gas Mileage
[q] Quit
Enter an option: x
Sorry, that option is invalid.

What would you like to do?
[r] Record Gas Consumption
[l] List Mileage History
[c] Calculate Gas Mileage
[q] Quit
Enter an option: q
Bye! See you next time!
```

Instructions

Create your program in the file `gas_mileage.py`. Unit tests are available for download. Your `gas_mileage.py` needs to be in the same folder as the unittest files.

You must follow the specifications exactly, but may choose your own method for solving the problem described for each. Once you have completed a function you should run the unittest for that function and have it pass all tests. Fix any errors, warnings, and/or failures.

Because of the user input and output, not all functions are easily tested with unit tests. Be sure to test these functions by running your program and observing the correct behavior.

- `milesPerGallon`
- `createNotebook`
- `recordTrip`
- `listTrips`
- `calculateMPG`
- `formatMenu`
- `formatMenuPrompt`
- `getUserString`
- `getUserFloat`
- `getDate`
- `getMiles`
- `getGallons`
- `recordTripAction`
- `listTripsAction`
- `calculateMPGAction`
- `quitAction`
- `applyAction`
- `main`

`milesPerGallon`

The function `milesPerGallon` receives two parameters, `miles` and `gallons`, both numbers (floats or integers). It returns the float value from dividing miles by gallons. If the value of gallons equals zero the function should return `0.0`. There is no need to round the returned values.

```
milePerGallon(300.0, 10.0) -> 30.0
milesPerGallon(215.0, 8.765) -> 24.529378208784937
milesPerGallon(300, 11) -> 27.272727272727273
milePerGallon(300, 0) -> 0.0
```

`createNotebook`

The function `createNotebook` does not receive any parameters. It must return an empty list to use as a “notebook” to track your trip data.

```
createNotebook() -> []
```

`recordTrip`

The function `recordTrip` takes four parameters, the notebook list, the date of the trip (a string), the miles traveled (a float), and the gallons of gas pumped (a float). It creates a new dictionary for the trip and adds it to the notebook list. The function does not return anything, but it will update the notebook list. *Note: the order of the pairs in the trip dictionary does not matter.*

```
notebook = []
recordTrip(notebook, '01/01/2017', 300.0, 10.0)
print(notebook) -> [{'date': '01/01/2017', 'miles': 300.0, 'gallons': 10.0}]
recordTrip(notebook, '01/02/2017', 270.0, 8.6)
print(notebook) -> [{'date': '01/01/2017', 'miles': 300.0, 'gallons': 10.0}, {'date': '01/02/2017',
'miles': 270.0, 'gallons': 8.6}]
```

`listTrips`

The function `listTrips` take one parameter, the notebook list. The function returns a list of strings. Each string in the list is a line that contains the date of the trip, the miles travel, the gallons pumped, and the miles per gallon (mpg) for that trip. *Consider using the `milesPerGallon` function from above*. See the example from above for an example on how to format the line. If there are no trips in the notebook, the function returns an empty list. **The function should not modify the notebook.**

You may round your float numbers to 2 decimal places, but it is not required.

```
notebook = []
listTrips(notebook) -> []
notebook2 = [
    {'date': '1/15/16', 'miles': 300.0, 'gallons': 10.0},
    {'date': '1/25/16', 'miles': 200.0, 'gallons': 8.0},
]
listTrips(notebook2) -> ['On 1/15/16: 300.0 miles traveled using 10.0 gallons. Gas mileage: 30.0 MPG', 'On
1/25/16: 200.0 miles traveled using 8.0 gallons. Gas mileage: 25.0 MPG']
```

`calculateMPG`

The function `calculateMPG` takes one parameter, the notebook list. The function calculates the Average MPG (calculated from total miles and total gallons) for all trips recorded and returns it as a float. You should not round the values. If there are no trips in the notebook the function returns `0.0`. *Do not use milesPerGallon* function from above. You should sum all the miles and sum all the gallons. Using those sums, calculate the average. **The function should not modify the notebook.**

```
notebook = []
calculateMPG(notebook) -> 0.0
notebook2 = [
    {'date': '1/15/16', 'miles': 300.0, 'gallons': 10.0},
    {'date': '1/25/16', 'miles': 200.0, 'gallons': 8.0},
]
calculateMPG(notebook2) -> 27.77777777777778
```

`formatMenu`

The function `formatMenu` does not receive any parameters. It must return a list of strings that contains the lines of the menu.

```
formatMenu() -> ['What would you like to do?', '[r] Record Gas Consumption', '[l] List Mileage History',
'[c] Calculate Gas Mileage', '[q] Quit']
```

`formatMenuPrompt`

The function `formatMenuPrompt` does not receive any parameters. It must return a string that contains the prompt to ask the user which menu option they would like to select.

```
formatMenuPrompt() -> 'Enter an option: '
```

`getUserString`

The function `getUserString` receives one parameter, a string that contains a prompt for input. It must return a string that contains the text input by the user, with any leading and trailing whitespace removed. If the user gives an empty string, prompt them again, until they give a non-empty string. Note that this function interacts with the user, so there will be output to the screen and input from the keyboard when it is called.

```
getUserString("What is your name? ") -> 'It is Arthur, King of the Britons.'
getUserString("What is your quest? ") -> 'To seek the Holy Grail.'
getUserString("What is the air-speed velocity of an unladen swallow?") -> 'What do you mean? An African or European swallow?'
```

`getUserFloat`

The function `getUserFloat` receives one parameter, a string that contains a prompt for input. It must return a float that contains the number input by the user. If the user enters a non-number or a number less than or equal to zero it should prompt them again. To accomplish this you might consider using a try/except clause which allows you to try an execute some code (convert user's input to a float) and recover if it fails.

Example

```
try:
    # try some code
    float('this is a string')
except:
    # this gets called if anything above fails.
    print("you can't convert that string to a float")
```

```
getUserFloat('Type 1.7 ') -> 1.7
getUserFloat('Type 1 ') -> 1.0
getUserFloat('Type in an integer ') -> 10.0
getUserFloat('Type in a float ') -> 3.142
```

`getDate`

The function `getDate` does not receive any parameters. It must prompt the user for a date and return the date input by the user. The user's response should not contain leading or trailing whitespace. The function should continue to ask the user for input until the user gives a valid response. *Consider calling `getUserString`.*

```
getDate() -> '01/01/2107'
getDate() -> 'Jan 01'
getDate() -> 'user-typed-this'
```

`getMiles`

The function `getMiles` does not receive any parameters. It must prompt the user to input a number and return the float value of that number. The function should continue to prompt the user until their input is valid (float or integer greater than zero). *Consider calling `getUserFloat`.*

```
getMiles() -> 300.0
getMiles() -> 217.6
```

`getGallons`

The function `getGallons` does not receive any parameters. It must prompt the user to input a number and return the float value of that number. The function should continue to prompt the user until their input is valid (float or integer greater than zero). *Consider calling `getUserFloat`.*

```
getGallons() -> 12.0
getGallons() -> 9.99
```

`recordTripAction`

The function `recordTripAction` receives one parameter, the notebook list. It must prompt the user for the

date, miles traveled, and gallons pumped and record it in the notebook. All inputs must be valid. *Consider using functions from above.* Print a message to the user so they know the trip was saved.

```
recordTripAction(notebook) -> the notebook should be modified.
```

`listTripsAction`

The function `listTripsAction` receives one parameter, the notebook list. It will display all of the trips in the format shown in the examples. If there are no trips in the notebook, it must display a message to inform the user. The function does not return anything. The function must not change the notebook.

```
listTripsAction(notebook) -> the notebook should NOT be modified.
```

`calculateMPGAction`

The function `calculateMPGAction` receives one parameter, the notebook list. It should print the average MPG to the user. If there are no trips in the notebook it should display a message notifying the user there is no trip data. The function must not change the notebook.

You may round your float number to 2 decimal places, but it is not required.

```
calculateMPGAction(notebook) -> the notebook should NOT be modified.
```

`quitAction`

The function `quitAction` receives the notebook list as a parameter. This function will display a message to the user indicating the end of the program. It will then terminate the program using `sys.exit( 0 )`. Be sure to do the correct `import` statement. This function does not return anything.

```
quitAction(notebook) -> the program will end
```

`applyAction`

The function `applyAction` receives the notebook list and a choice string as parameters. This function will call the appropriate action function based on the choice string. If the choice string does not match any accepted choices, it will display a message to the user. This function does not return anything. The notebook may be changed as a result of the chosen action.

```
applyAction(notebook, "r") -> the notebook will have a new trip added.
applyAction(notebook, "l") -> the trips will be displayed.
applyAction(notebook, "c") -> the overall average MPG will be displayed.
applyAction(notebook, "q") -> the program will terminate.
applyAction(notebook, "x") -> the user will receive a message.
```

`main`

The function `main` receives no parameters, and returns nothing. This function ties everything together. Create a notebook, repeatedly asking the user their choice and taking appropriate action. *Note: everything you need to finish the main function should be contained in the functions above.*

```
main() -> the program runs.
```

Finishing Up

Lastly add this snippet at the bottom of your file which will execute your `main()` function when you run `gas_mileage.py` but will allow it to be imported into the unittest files without executing the main function.

```
if __name__ == '__main__':
    main()
```

Pass-of instructions

1. To pass off this assignment you need to show your completed program to the lab assistants.
   - Show them your `gas_mileage.py` code
   - Run `test_all.py` - All tests MUST pass!
   - Run `gas_mileage.py`
   - *The lab assistant may additional tests they want you to run*
2. Upload your `gas_mileage.py` file to canvas, please add a comment to the top of the file with your name and time your class meets.