

Important

1. Due Date: **Thursday October 12th**
2. This homework is graded out of 100 points.
3. This is an Individual Assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 1301, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
 - TA Helpdesk (Schedule posted on class website.)
 - Email TA's or use Piazza Forums Notes:
 - How to Think Like a Computer Scientists [<http://openbookproject.net/thinkcs/python/english3e/>]
 - CS 1301 Python Debugging Guide [http://www.cc.gatech.edu/classes/AY2016/cs1301_spring/CS-1301-Debugging-Guide/index.html]
5. Don't forget to include the required collaboration statement (outlined on the syllabus). Failing to include the Collaboration Statement will result in no credit.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. Comment or delete all your function calls. When your code is run, all it should do is build without any errors. Having function calls or extraneous code outside the scope of functions will result in a lower grade. (import statements and global variables are okay to be outside the scope of a function)
8. **Read the entire specifications document before starting this assignment.**
9. **IF YOUR CODE CANNOT RUN BECAUSE OF AN ERROR, IT IS A 0%.**
10. **IF YOUR DELIVERABLES ARE NOT NAMED EXACTLY AS STATED IN THE INSTRUCTIONS IT IS A 0%.**

Introduction

The goal of this homework is for you to showcase your knowledge about how to properly manipulate files. Refer to the rubric to see how points will be rewarded for each function. You have been given HW6.py to fill out with instructions in the docstrings. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin. You have until **Thursday, October 12th** to complete this assignment. Don't forget to include your name and your collaboration statement. Re-download your submission from T-Square after you submit it to make sure that your code runs successfully.

Part 1: CSV and Data Analysis

You have been provided with a comma-separated values file (a .csv). In this section of the assignment you will be reading this file and writing functions to get statistics and data from it. You are not required to use the csv module, but it might be useful to do so. In order to get some more practice with CSV in Python, these resources might be helpful:

- <https://pythonprogramming.net/reading-csv-files-python-3/>
- <https://docs.python.org/3/library/csv.html>

Description of the Dataset:

The file, data.csv, contains ~900 entries. It has the Real Estate data from house sales in Sacramento over a two-month period. This data is in no way representative of larger data sets, but it has very good and interesting data we can use. It includes the following variables/columns: street, city, zip, state, beds, baths, sq_ft, typ, sale_date, price, latitude, and longitude.

Assignment:

Do not modify in any way the dataset that has been given to you as small changes might make your assignment fail test cases. For the purpose of this assignment you can assume that the strings of the house types are case sensitive. This means that for example the house type 'Condo' is not the same as 'condo'.

Complete the following functions:

Function name: affordable_homes

Parameters: max_price (int), min_size (int), num_homes (int), out_file (string)

Return value: None

Description: You will write to the file, outFile, a certain number of homes, given by the parameter num_homes, that cost strictly less than the max_price and are strictly greater than the min_size. If no homes meet the qualifications, then leave out_file empty. The homes that meet the qualifications will be written to out_file in order from cheapest to most expensive in the following format:

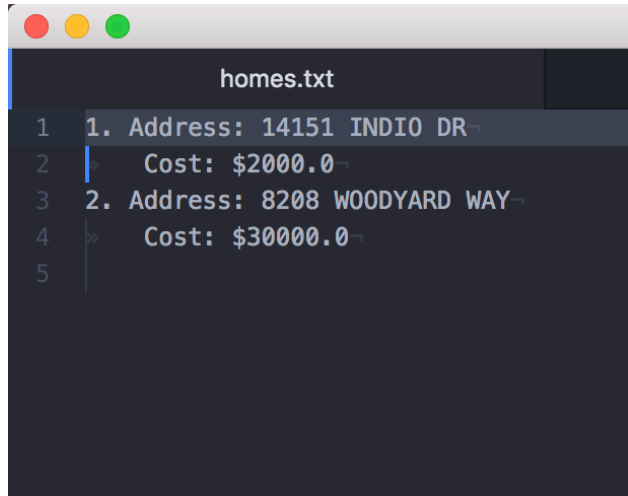
```
1. ADDRESS: [HOUSE ADDRESS]
   COST: $[HOUSE COST]
2. ADDRESS: [HOUSE ADDRESS]
   COST: $[HOUSE COST]
...
...
```

Notes on Formatting:

1. Every line of the file must end in a newline character '\n' including the last line of the file.
2. The Cost line must be indented by a tab character '\t' from the left side of the file, and each monetary amount must be preceded by a dollar sign.
3. Run HW6_test.py to verify that your formatting is correct.

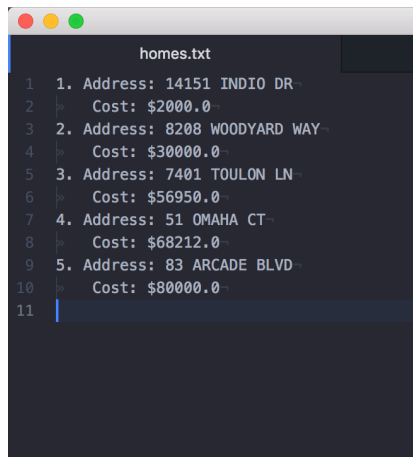
Test Cases:

```
>>> affordable_homes(200000, 1000, 2, 'homes.txt')
```



```
homes.txt
1 1. Address: 14151 INDIO DR
2   Cost: $2000.0
3 2. Address: 8208 WOODYARD WAY
4   Cost: $30000.0
5
```

```
>>> affordable_homes(100000, 1100, 5, 'homes.txt')
```



```
homes.txt
1 1. Address: 14151 INDIO DR
2   Cost: $2000.0
3 2. Address: 8208 WOODYARD WAY
4   Cost: $30000.0
5 3. Address: 7401 TOULON LN
6   Cost: $56950.0
7 4. Address: 51 OMAHA CT
8   Cost: $68212.0
9 5. Address: 83 ARCADE BLVD
10  Cost: $80000.0
11
```

Function name: home_profile

Parameters: address (string), redo_file (boolean)

Return value: None

Description: Write a profile of the home whose address (string) is passed into the function to a file named profileFile.txt. Assume the address given is always valid. If the boolean value redo_file is True, then start the file over (begin at a blank file and write the house description to the initially empty file). If redo_file is False, then simply append the description of the address passed in to the bottom of the file. Write the address of the home to profileFile.txt in the following format:

Address:

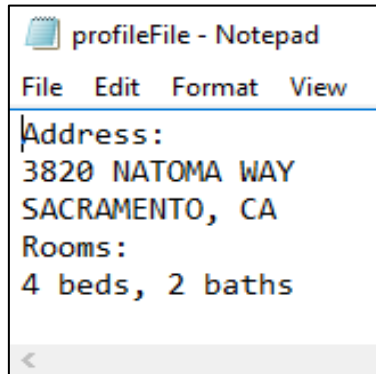
```
[ADDRESS OF HOME PASSED IN]
[CITY], [STATE]
Rooms:
[NUMBER OF BEDROOMS] beds, [NUMBER OF BATHS], baths
```

Notes on Formatting:

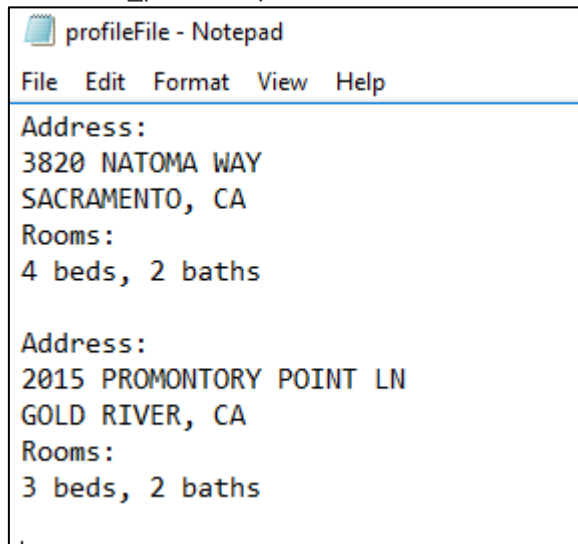
1. Every line of the file must end in a newline character '\n'.
2. The last line of the file must end in two newline characters (to make a distinction between houses when more than one description is contained in the file) '\n\n'.
3. Run HW6_test.py to verify that your formatting is correct.

Test Cases:


```
>>> home_profile('3820 NATOMA WAY', True)
```



```
>>> home_profile('2015 PROMONTORY POINT LN', False)
```



```
>>> home_profile('1500 ORANGE HILL LN', True)
```

 profileFile - Notepad
File Edit Format View Help
Address:
1500 ORANGE HILL LN
PENRYN, CA
Rooms:
3 beds, 2 baths

Function name: sold_day

Parameters: day (string)

Return value: int

Description: Write a function called sold_days that will take in a three letter abbreviation of a day of the week. Count up how many houses were sold on that day of the week. Return the number of houses sold on that day as an int. If an invalid string is passed in as the day, return -1.

Test Cases:

```
>>> test1 = sold_day("Tue")
>>> print(test1)
177
```

```
>>> test2 = sold_day("Mon")
>>> print(test2)
268
```

```
>>> test3 = sold_day("Tuesday")
>>> print(test3)
-1
```

Part 2: Writing to Files

Function name: make_roster

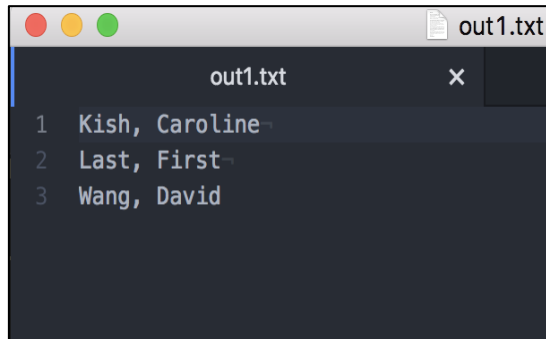
Parameters: students (str), filename (str)

Return value: None

Description: Take in a string of students. The format of the students will be "First Last, First Last, First Last". Write the names of the students to the file specified by the parameter. Write each line in the format "Last, First". Each student should be in a different line, and the students should be sorted alphabetical by last name. Make sure there is no newline character after the last student

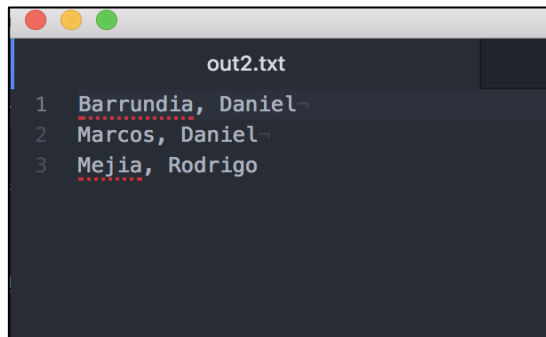
Test Cases:

```
>>> make_roster("Caroline Kish, David Wang, First Last", "out1.txt")
```



```
out1.txt
1 Kish, Caroline
2 Last, First
3 Wang, David
```

```
>>> make_roster("Rodrigo Mejia, Daniel Marcos, Daniel Barrundia", "out2.txt")
```



```
out2.txt
1 Barrundia, Daniel
2 Marcos, Daniel
3 Mejia, Rodrigo
```

Function name: choose_group

Parameters: aList(str), num (int), filename (str)

Return value: Boolean (True or False)

Description: Write a function that takes in a list of student names. You will form a group of size num from the group of names. You will choose the last num students in the list. You will write out the names of the students who will form the group to the filename passed in. This will be the format:

Team: [Name1], [Name2], ..., [NameNum]

If there are not enough students for a group, write 'Not enough people for a group.' to the file. If there are zero people on a team, then leave the file empty. Finally, return True if there are enough people to form a group and False otherwise.

Notes on Formatting:

1. The file will only be one line no matter how many names there are so it should contain no newline characters ('\n').

Test Cases:

```
>>> test_01 = choose_group(["Cathy", "Kelly", "Caroline", "Elena"], 4,
'group_test_01.txt')
>>> print(test_01)
True
```

```
>>> test_02 = choose_group(["David", "John", "Daniel", "Cory"], 10,
'group_test_02.txt')
>>> print(test_02)
False
```

```
>>> test_03 = choose_group(["Betty", "Veronica", "Jughead", "Archie", "Zack"], 2,
'group_test_03.txt')
>>> print(test_03)
True
```

Part 3: Reading Files

Assume the text files in this section are formatted in the following way:

```
Student 1
Test 1: score
Test 2: score
...
```

```
Student 2
Test 1: score
Test 2: DNT
Test 3: score
...
```

The student's name will be followed by a variable number of tests, each on its own line. After the test number and a colon, if the student took that test then the float of their score is given. If the student did not take a certain test, DNT will be written instead.

Function name: `get_roster`

Parameters: `file_name` (str)

Return value: `roster` (list)

Description: Read in a file (that has the same format shown above). Generate a list with every student's name. Make sure the names do not contain newline characters (`\n`) and are in the order they appear in the file. Assume the passed in filename is a valid file.

Test Cases:

```
>>> test1 = get_roster("scores1.txt")
>>> print(test1)
['Iron Man', 'Hulk', 'Captain America']
>>> test2 = get_roster("scores2.txt")
>>> print(test2)
['Batman', 'Wonder Woman', 'Superman']
```


Function name: tests_missed

Parameters: file_name (str)

Return value: tests_missed (list)

Description: Read in a file (that has the same format shown above). For every DNT scores, add the test number to a list. Return the list at the end. The order of tests should be the same order as the DNT scores are encountered. If multiple people did not take the same test, the test number should be included in the list multiple times. Assume the passed in filename is a valid file.

Test Cases:

```
>>> tests_missed("scores1.txt")
[4, 5]
>>> tests_missed("scores2.txt")
[4, 5, 2, 3, 5]
```

Function name: find_avg

Parameters: file_name (str), student_name (str)

Return value: student_avg (float)

Description: Read in a file (that has the same format shown above). Find the student specified by the parameter. Generate the average of all the student's score and return the average. If the student has a DNT, disregard that score (do not use that to compute the average). Round the answer to 2 decimal places. Assume not two students have the same name and every student will have at least one valid test. If the name of the student passed in is not in the file, return 0. Assume the passed in filename is a valid file.

Test Cases:

```
>>> find_avg("scores1.txt", "Iron Man")
95.44
>>> test1 = find_avg("scores1.txt", "Iron Man")
>>> print(test1)
95.44
>>> test2 = find_avg("scores2.txt", "Wonder Woman")
>>> print(test2)
97.86
>>> test3 = find_avg("scores1.txt", "Superman")
>>> print(test3)
0.0
```

Grading Rubric

- affordable_homes:	15 pts
- home_profile:	10 pts
- sold_day:	10 pts
- make_roster:	15 pts
- choose_group:	10 pts
- get_roster:	10 pts
- tests_missed:	10 pts
- find_avg:	20 pts

- Total	100/100 pts
---------	-------------

Provided

The following file(s) have been provided to you. There are several, but you will only edit one of them:

1. `HW6.py`

This is the file you will edit and implement. All instructions for what the methods should do are in the docstrings.

2. `HW6test.py`

This is a file containing tests for you to use if you will like to debug and test your code. You are not required to edit, submit, or even use this file.

Deliverables

You must submit all of the following file(s). Please make sure the filename matches the filename(s) below. Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder and run them.

1. `HW6.py`

If this file does not run (if it encounters an error while trying to run), you will get no credit on the assignment.