Important

1. Due Date: **Monday, February 13th**
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%.**

Introduction

The goal of this homework is for you to showcase your knowledge about how you can use indexing to manipulate strings and lists. The functions you are asked to code will all use some sort of indexing to accomplish the goal. Refer to the rubric to see how points will be rewarded for each function. You have been given HW4.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 Monday, February 13th to complete this assignment.

Function name: **`replaceStr`**
Parameters: original string (str), string to replace (str), replacement string (str)
Return value: new string (str)

Description: Write a function that takes three strings. The first string will be the original message. The second string will be another string that may or may not be present in the original string. If the second string is present in the original string, replace all instances of that string with the third string passed in as a parameter. Return the new string.

Note: **Do not use `.replace()`** you are, in essence, implementing this method yourself.

Test Cases:

```
     >>> newString = replaceStr("End To End Applications", "End",
"Back")
     >>> print(newString)
     Back to Back Applications

     >>> newString = replaceStr("Vehement Capital Partners",
"Vehement", "Mediaware")
     >>> print(newString)
     Mediaware Capital Partners
```

Function Name: **`breakEven`**
Parameters: A string (can be random letters, one word, or a sentence) and an int of either 1 or 2 (if it's 1 return the 1st half, if it's 2, return the 2nd half)
Return value: A string that contains either the front or back half of the string

Description: Write a function that divides a string into two halves. If the length is even, the front and back halves are the same length. If the length is odd, the extra character goes in the front half. If the string is of length 1, the letter goes to the front half and the back half remains empty. Return either the front or back half of the string if the second parameter is 1 or 2, respectively.

Test Cases:

```
     >>> breakEven("asdf", 1)—> "as"
     >>> breakEven("Corporate Earnings", 1)—› "Corporate"
     >>> breakEven("Wells Fargo", 1) -> "Wells "
     >>> breakEven("a", 2) -> ""
```

Function Name: **`stickTogether`**
Parameters: A list of arbitrary elements
Return value: A string that is the concatenation of all string elements in the list. Note that non-string elements should be skipped, including nested lists.

Description: Write a function called stickTogether that accepts a list as a parameter. It should return a string that is the concatenation of all string elements in the list. Note that non-string elements should be skipped, including nested lists.

Hint: Try using `type()` function. Note that `" "` is also a string (with only a space) and should be also included.

Test Cases:

```
>>> stickTogether( [4, "John", ["Lucas",3], " ", True, "Smith",
7] ) –›  "John Smith"
>>> stickTogether( [ "I am", 237846, " taking", " 1301", " ",
[":)"],9.72, "!" ) –›  "I am taking 1301 !"
```

Function Name: **`addNums`**
Parameters: A list of floats and/or integers. The list may be empty.
Return Value: The modified list.

Description: Define a function that adds each item in a list of numbers (either floats or ints) with the next element. If an item is the last element in the list, the item should be added with itself. The function should return the modified list.

Test Cases:

```
>>> testOne = addNums([1, 2, 3])
>>> print(testOne)
[3, 5, 6]

>>> testTwo = addNums([])
>>> print(testTwo)
[]

>>> testThree = addNums([2.2, 4.5, 5.1, 2])
>>> print(testThree)
>>> [6.7, 9.6, 7.1, 4]
```

Function Name: **`mergeAdjacent`**
Parameters: A list of ints
Return value: A list where all adjacent equal elements (same number next to each other) have been reduced to a single element.

Description: Write a function called mergeAdjacent that accepts a list of ints as a parameter. If the adjacent ints are equal, merge the two equals into one int of the same value. Return a new list that has no duplicate adjacent ints.

Test Cases:

```
>>> mergeAdjacent([1,2,2,3])–›[1,2,3]
>>> mergeAdjacent([1,0,5,2,6,6,7,7,12])–›[1,0,5,2,6,7,12]
>>> mergeAdjacent([1,1,1,1,1])–›[1]
>>> mergeAdjacent([1,4,2,1,4])–›[1,4,2,1,4]
```

Function name: **`findAgents`**
Parameters: regions (a list of lists); each region is a list of names (strings)
Return value: a list of all the first names that occur more than once in a single region

Description: Write a function that returns a list of all the first names that occur more than once in a given region (represented by a nested list). First names that are repeated once in a region, and are also in another region should not be included in the returned list. (See test cases for more explanation)

Test Cases:

```
>>> agentsList = [["John Knight", "John Doe", "Erik Peterson"],
["Fred Douglas", "John Stephans", "Mike Dud", "Mike Samuels"]]
>>> names = findAgents(agentsList)
>>> print(names)
['John', 'Mike']

>>> anotherAgentsList = [["John King", "Kane Hill", "Misha Fit"],
["Hank Fred", "Joe Biden", "Ellen Hip"]]
>>> namesTwo = findAgents(anotherAgentsList)
>>> print(namesTwo)
[]

>>> agentsListThree = [["Daniel Johnson", "Elana May", "Elana
Johnson"], ["John Denver", "John Doe", "John King", "Jerry
Jackson"], ["Caroline Rose", "Lucille King", "David Wang"]]
>>> namesThree = findAgents(agentsListThree)
>>> print(namesThree)
['Elana', 'John']
```

Function name: **`bestAthlete`**
Parameters: a nested list; each list will contain three values; the first value will be the string name of a sport, the second value will be the amount of points a player has scored in the sport, and the third parameter will be the name of a person who has played this sport
Return value: a string representation of the player with the high score, the sport that is being examined and the player's score

Description: You are helping a recruiter find an athlete for the last position open on a sports team. Return a string that tells the recruiter the name of the most promising candidate (the athlete who has scored the most amount of points), the sport the candidate plays, and the number of points that this player has scored. If two players are tied for the highest score, your function should return the player that the recruiter encounters first (the player that appears first in the list). If there are athletes of different sports within the list return None as you can't compare one sport with another.

Note: Failing to return the exact format of the string will make you fail the test cases. (Format: Name[comma][space]sport[comma][space] score)

```
>>> athletesOne = [["football", 400, "Jim"], ["football", 205,
"Joe"], ["football", 320, "Carl"]]
>>> message = bestAthlete(athletesOne)
>>> print(message)
Jim, football, 400

>>> athletesTwo = [["football", 10030, "Kim"], ["basketball",
4093, "Sarah"], ["basketball", 343, "Jane"]]
>>> message = bestAthlete(athletesTwo)
>>> print(message)
None

>>> athletesThree = [["soccer", 300, "Erica"], ["soccer", 200,
"John"], ["soccer", 2000, "Jim"], ["soccer", 3, "Molly"]]
>>> message = bestAthlete(athletesThree)
>>> print(message)
Jim, soccer, 2000
```

Grading Rubric

- `replaceStr`:	10 points
- `breakEven`:	10 points
- `stickTogether`:	10 points
- `addNums`:	10 points
- `mergeAdjacent:`	20 points
- `findAgents`:	20 points
- `bestAthlete`:	20 points
- **Total**	100/100 points

Provided

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

1. HW4.py

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

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. HW4.py

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