# Revision- Lists

## Q1: Reverse Words [ ** ]

Write a function called `reverse_words()` that takes in a piece of text that consists of multiple words. The function **returns** a string that is a transformed version of the text where each word is reversed.

You can assume that the words in the given text are separated by spaces, i.e., there is a single space between two adjacent words.

For example, `reverse_words("I study at SMU")` returns `"I yduts ta UMS"`. `reverse_words("Python is a programming language.")` returns `"nohtyP si a gnimmargorp .egaugnal"`. If the input text is an empty string, the function returns an empty string.

## Q2: Greatest Common Divisor [**]

Write a function called find_greatest_common_divisor(). The function takes in two positive integers as its parameters. (You can assume the two parameters are positive integers.) The function returns the greatest common divisor of the two integers. In mathematics, the greatest common divisor of two integers is the largest positive integer that divides each of the integers.

For example:
`find_greatest_common_divisor(8, 12)` returns 4
`find_greatest_common_divisor(100, 250)` returns 50
`find_greatest_common_divisor(13, 25)` returns 1

## Q3: List of Tuples [ **]

In the Jupyter Notebook flie, you are given a list of tuples, where each tuple represents the title, genre and duration (in minutes) of a movie. You are also given some testing code.

Implement the following functions:

- `get_average_duration()`: This function takes in a list of tuples as described above. The function returns the average duration of all the movies in the list. If the list is empty, the function returns 0.0.
- `get_num_movies_of_genre()`: This function takes in a list of tuples as described above together with a string that indicates a genre. The function returns the number of movies belonging to that genre in the give list. If the list is empty, the function returns 0.
- `get_title_of_longest_movie()`: This function takes in a list of tuples as described above and returns the title of the movie whose duration is the longest among all the movies in the list. If the list is empty, the function returns an empty string.
- `get_movies_with_keyword()`: This function takes in a list of tuples as described above together with a string that represents a keyword. The function returns a new list of

tuples where each tuple still represents a movie. The returned list contains those movies in the original list whose titles have the specified keyword as a substring. If the original list is empty, the function returns an empty list.

## Q4: Average Height [ ** ]

Write a function called `compute_avg_height`. The function takes in a single parameter, which a string that consists of some people's heights. An example of such a string is `"Jonathan Li:1.75m, Lim, Josephine : 1.59m , George Khoo:   1.7 m"`. We can see that for each person, the person's name is given, followed by a colon. After that there may be zero, one or more spaces, followed by the person height. The person's height is followed by the letter `'m'`, but there might be zero, one or more spaces before `'m'` and after `'m'`. The function `compute_avg_height` should return the average height of all the people listed in the string.

You can assume that the input string is always well formatted as described above. To be more specific, you can assume the following:

- Digits and the symbol `'.'` appear only as part of a person's height.
- A person's name doesn't contain the symbol `':'`.
- However, a person's name may be expressed as `'<surname>, <given name>'`, such as `'Lim, Josephine'`.
- The string parameter contains at least one person's height.

## Q5: List of Numbers

**Part (a) [ ** ]**

Define a function called `get_larger_values()`. This function takes in a list of float values as its parameter. The function **returns** a new list of float values, which are those values from the original list which are above the average of the values in the original list. The original list should remain unchanged.

You can assume that the original list contains at least one value.

For example, `get_larger_values([2.5, 3.5, 5.5, 1.0])` should return the list `[3.5, 5.5]`, because the average of the values in the original list is 3.125 ( (2.5 + 3.5 + 5.5 + 1.0) / 4 = 3.125), and 3.5 and 5.5 are the two values in the original list that are above 3.125.

**Part (b) [ ** ]**

Define a function called `merge_list()`. The function takes in two lists of numbers as its two parameters. It merges the two lists by alternating between the two lists to take their values one by one and inserting these values into a new list. It then returns the new list. The two original lists should remain unchanged.

The two lists may not be of the same length. The extra elements from the longer list are added to the returned list without merging with any element from the shorter list.

For example, `merge_list([1, 3, 10, 15, 4, 7, 12], [9, 5, 2])` returns the list `[1, 9, 3, 5, 10, 2, 15, 4, 7, 12]`. Note that 15, 4, 7 and 12 (the extra elements from the longer list) are included in the returned list at the end.

**Part (c) [ ** ]**

Define a function called `check_numbers()`. The function takes in two lists of positive integers as its two parameters. You can assume that both lists contain only positive integers. The function returns `True` if each integer in the first list is divisible by at least one of the integers in the second list; otherwise it returns `False`.

## Q6: Books [ *** ]

**Part (a)**

You are given a list of tuples, where each tuple represents a copy of a book in a library's collection. Specifically, each tuple consists of four elements:

(1) A string representing the title of the book.
(2) A string indicating the edition of the book. E.g., the third edition of a book would have this string set to "Ed-3".
(3) A string indicating whether this book is a hardcover book or a paperback book, i.e., the value of this string is either "hardcover" or "paperback".
(4) An integer indicating the number of copies of the book with the specified title, edition and type (hardcover vs. paperback).

Note that the library has multiple books having the same title.

Write a function called `get_unique_titles()` that takes in such a list of tuples as its parameter. The function returns a new list of strings, which are the unique book titles from the original list.

For example, `get_unique_titles([("Intro to Programming", "Ed-2", "paperback", 2), ("Intro to Python", "Ed-1", "paperback", 5), ("Intro to Programming", "Ed-3", "hardcover", 4)])` should return the list `["Intro to Programming", "Intro to Python"]`.

**Part (b)**

Write a function called `get_titles_and_counts()`. The function takes in the same kind of list of books as described above. The function should return a list of tuples, where each tuple contains two elements: (1) A book title, and (2) the number of copies of that book.

For example, `get_titles_and_counts([("Intro to Programming", "Ed-2", "paperback", 2), ("Intro to Python", "Ed-1", "paperback", 5), ("Intro to Programming", "Ed-3", "hardcover", 4), ("Intro to`

`Python", "Ed-3", "hardcover", 3)])` should return the list `[("Intro to Programming", 6), ("Intro to Python", 8)]`.

We get this result because there are six copies of “Intro to Programming”:

- Two in a 2nd edition in paperback and
- Four in a 3rd edition in hardcover

and there are eight copies of “Intro to Python”:

- Five in a 1st edition in paperback and
- Three in a 3rd edition in hardcover.

## Q7: Retrieve Numbers [ *** ]

Define a function called `retrieve_numbers()`. The function takes in a string that contains some numbers as its parameter. The function **returns** the sequence of these numbers separated by spaces as a string.

For example,

- `retrieve_numbers("12abc600$##0900AB 100X")`
  returns the string `"12 600 0900 100"`

- `retrieve_numbers("34.5689abc980")`
  returns the string `"34 5689 980"`

- `retrieve_numbers("xyz")`
  returns the string `""`

- `retrieve_numbers("abc25xyz")`
  returns the string `"25"`