NYU, Tandon School of Engineering
CS-UY 1114 Introduction to Programming and Problem Solving — Fall 2017

# Homework #7
## Due by Friday 11/10, 11:55pm

**Submission instructions:**

1. You should submit your homework in the NYU Classes system.
2. **Create one '.py' file per question. Each '.py' file may contain multiple functions.**
3. For this assignment, you should turn in 6 '.py' files. Name your files 'YourNetID_hw7_q1.py', 'YourNetID_hw7_q2.py', etc.

**Question 1:**

Implement function `max_abs_val(lst)`, which returns the maximum absolute value of the elements in `lst`.

For example, given a list `lst`: `[-19, -3, 20, -1, 0, -25]`, the function should return `25`.

**Question 2:**

Write a function `find_all(lst, val)`, which returns a list containing indices of all occurrences of `val` in `lst`.

Note: You may assume that `val` is of the same type as all of the items in `lst`.

For example, given a list `lst`: `['a', 'b', '10 ', 'bab', 'a']` and a value `val`: `'a'`, the function should return a list `[0, 4]`.

**Question 3:**

Write a function `add_list(lst1, lst2)` that takes two lists of numbers of the same length and returns a list consisting of the sum of the first numbers, the sum of the second numbers, etc.

For example `add_list([1,2,3], [4,5,6])` should return the list `[5, 7, 9]` since 1+ 4 is 5, 2 + 5 is 7, etc.

Your main program should prompt user for input. It should read a list of numbers, one number per line, followed by "`done`", then another list of numbers, one per line, followed by done. If the lists have different lengths, your main program should print "Lists are different lengths." If they are the same length, the main program should call `add_list` and print the resulting list, one number per line.

Note: you may define additional functions.

**Question 4:**

Implement the function `create_prefix_lists(lst)` that will return a sequence of lists, each containing a prefix of `lst`. All the lists should be collected as one big list. For example, for `[2,4,6,8,10]` the function should return `[[], [2], [2,4], [2,4,6], [2,4,6,8], [2,4,6,8,10]]`

**Question 5:**

Design and implement encoder and decoder functions that perform run length compression of strings (https://en.wikipedia.org/wiki/Run-length_encoding).

a. Run length string encoder:

The encoder function should take a string as a parameter, and return the encoded string, represented in a list.

For example:

Given the string: "aadccccaa"

The encoded string will be: [['a', 2], ['d', 1], ['c', 4], ['a', 2]].

b. Run length string decoder:

The decoder function should take an encoded string (represented in a list) as a parameter, and return the original string.

For example:

Given the encoded string: [['a', 2], ['d', 1], ['c', 4], ['a', 2]]

The decoded string is: "aadccccaa".

Note: Use the top down design approach to break the problem into smaller sub-problems that can be used to solve it.

**Question 6:**

Implement following functions:

1. `reverse_to_new_list(lst)` - This function creates **a new list** containing the elements of `lst` in reverse order and returns it.
2. `reverse_in_place(lst)` – When called, this function **changes the order** of the elements in `lst` (in place) to be in the reverse order.
   Hint: think about the how the positions change before and after the reverse.

You may use the following main program to check your implementation of `reverse_to_new_list` and `reverse_in_place`.

```
def main():
    lst1 = [1, 2, 3, 4, 5, 6]
    rev_lst1 = reverse_to_new_list(lst1)
    print("After reverse_to_new_list, lst1 is", lst1,
          "and the returned list is", rev_lst1)

    lst2 = [1, 2, 3, 4, 5, 6]
    print("Before reverse_in_place, lst2 is", lst2)
    reverse_in_place (lst2)
    print("After reverse_in_place, lst2 is", lst2)
```

Expect output:

```
After reverse_to_new_list, lst1 is [1, 2, 3, 4, 5, 6] and the returned
list is [6, 5, 4, 3, 2, 1]
Before reverse_in_place, lst2 is [1, 2, 3, 4, 5, 6]
After reverse_in_place, lst2 is [6, 5, 4, 3, 2, 1]
```