CS 116 Winter 2013

Assignment 07

Due at 10.00am Wednesday, March 20

- Each question requires the use of recursion in a nontrivial way.
- Do **NOT** use Python iteration (loops). Any repetition should be implemented using recursion or abstract list functions.
- Do not import any modules except the `check` and `math` module.
- Do not use any global variables.
- Download the testing module from the course Web page. Include `import check` in each solution file.
- Be sure to use the strings that exactly match those specified on the assignment and interface. Changing them in any way may result in the loss of correctness marks.
- You are encouraged to use helper functions in your solutions as needed. Include them in the same file as your solution, but make helper functions separate functions from the main function, i.e. do NOT make them local functions. You do not need to provide examples and tests for these helper functions.
- Do not copy the purpose directly from the assignment description. The purpose should be written in your own words and include reference to the parameter names of your functions.
- The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.
- Assignments will not be accepted through email. Course staff will not debug code emailed to them.
- Test data for all questions will always meet the stated assumptions for consumed values.
- Read the course Web page for more information on assignment policies and how to organize and submit your work. Follow the instructions in the style guide. Specifically, your solutions should be placed in files `a07qY.py`, where `Y` is a value from 1 to 4.
- Download the interface file from the course Web page.
- For full marks, it is not sufficient to have a correct program. Be sure to follow all the steps of the design recipe given in the Python Style Guide for CS116, including the definition of constants and helper functions where appropriate.

Coverage: Module 7
Language: Python

1. The Microsoft Word **Find and Replace** function finds the target in the whole document and replaces it with the given replacement text.

   Write a function *`replace_str`* to implement this functionality. Your function will consume 3 non-empty strings, *`base`*, *`target`* and *`rep`*. The first string, *`base`*, represents a base string that you want to update. The second string *`target`* represents the target string that you want to replace and the third string *`rep`* represents a string that will replace the *`target`* in the updated string. The function produces a new string in which the *`target`* string is replaced by the *`rep`* string in the *`base`* string, but produces the same *`base`* string if either of the following conditions hold true.

   - If the *`target`* string is not found in the *`base`* string or,
   - If the *`target`* and *`rep`* are the same strings.

   For example,

   ```
   replace_str("This is a book","a","the")=>'This is the book'

   replace_str("This is my book","a","the")=>'This is my book'
   ```

Assignment 07

Due at 10.00am Wednesday, March 20

```
replace_str("I like this book","I","I")=> 'I like this book'

replace_str ("my brother reads books and sometimes he reads magazines",

"reads", "likes")

=>'my brother likes books and sometimes he likes magazines'

replace_str("Apple is a  fruit", "f" , "t" )=>'Apple is a truit'

replace_str("aaaaa","aa","x")=>'xxa'
```

Note:

- You are not allowed to use the string methods `replace` and `find` for this question

2. A HyperText Markup Language (HTML) is used to create webpages. A valid HTML code for creating a webpage is defined as.

   - "HEADING" is a valid html code
   - If s is a valid html code, so is "<html>s</html>"
   - If s is a valid html code, so is "<h1>s </h1>"

   Write a function `is_html_code`, which consumes a string `s`, and produces `True` if `s` is a valid html code, and `False` otherwise.

   For example,

   ```
   is_html_code("")=>False

   is_html_code("HEADING")=>True

   is_html_code("<html>HEADING</html>") =>True

   is_html_code("<h1>HEADING</h1>")=>True

   is_html_code("<h1></h1>") =>False

   is_html_code("<h1>HEAD</h1>") =>False

   is_html_code("<html><h1><html>HEADING</html></h1></html>") =>True
   ```

3. Consider the following data definition for `coins_needed`:

   `coins_needed` is a list of length 2 : `[coin_value, num_coin_value]`, where `coin_value` and `num_coin_value` are non negative integers, corresponding to the value of a coin and the number of coins of that value that we need .

   Write a function called `coin_change`, which consumes a non-empty list of integers `avail_change` (in decreasing order), and an integer `total_cents[>0]`, and produces a list of `coins_needed`. The

Assignment 07

Due at 10.00am Wednesday, March 20

avail_change is a list of integers, which contains any number of denominations **(in cents)** in strictly decreasing (i.e. no duplicate values) order, with the last coin always equal to 1, and *total_cents* is the money in cents that we want to represent as change.

The function finds the *coin_value* and *num_coin_value* needed to make the exact change of *total_cents* by using as many of coins of the first value as possible, as many of the coins of the second value, and so on.

For example:

```
coin_change([50,5,1],108)=>[[50,2],[5,1],[1,3]]

coin_change([10,5,1],25)=>[[10,2],[5,1]]

coin_change([50,25,10,5,1],200)=>[[50,4]]

coin_change([90,85,20,1],108)=>[[90,1],[1,18]]
```

Note:

- *total_cents* contains the total cents that we want to convert. For example, if we want to convert $1 we will give 100 cents instead of $1.

4. The Median of the numbers in a list divides a set of elements into (roughly) equal sets of numbers, such that half of the numbers are less than the median, and half of the numbers are greater than median.

   Write a python function *find_median,* which consumes two non-empty lists of integers of equal lengths *lst1* and *lst2* (can be an odd or even number of elements) sorted in a non-decreasing order, and produces an integer that represents the median of the two sorted lists [you are **NOT** allowed to use the *sort* method for the lists].

   The function **MUST USE** the following algorithm to compute the median of the two lists:

   1. Compute the median, *med1*, of *lst1* using the following rules:

      a. If *lst1* has an odd number of elements, *med1* is the middle element. For example,

      Median of [2,5,7] => 5

      b. If *lst1* has an even number of elements, *med1* is the integer (average) of the two middle elements. For example,

      Median of [10,15,20,21] => 17

Assignment 07

Due at 10.00am Wednesday, March 20

2. Compute the median, *med2*, of *lst2* using the same technique as computing *med1*.

3. If *lst1* and *lst2* are both of length 1, return the integer average of *med1* and *med2*.

4. If *lst1* and *lst2* are both of length 2, return the median calculated using the formula:

$$\textbf{Median} = (\textbf{max}(\textbf{lst1}[\textbf{0}], \textbf{lst2}[\textbf{0}]) + \textbf{min}(\textbf{lst1}[\textbf{1}], \textbf{lst2}[\textbf{1}])) \,/\textbf{2}$$

5. If *med1* and *med2* are equal numbers, return either *med1* or *med2*.

6. If *med1* is greater than *med2*, call the function *find_median* on the sublists of *lst1* and *lst2*. [Each time a function *find_median* is called, both sublists of *lst1* and *lst2* should be of equal lengths].

   The following are the rules for creating sublists of *lst1* and *lst2*.

   a. Sublist of *lst1* should be a list starting from first element up to and including *med1* (including both middle entries when *lst1* contains an even number of entries).

   b. Sublist of *lst2* should be a list starting from *med2* (including both middle entries when *lst2* contains an even number of entries) to the end of the list.

7. If *med1* is less than *med2*, call the function *find_median* on the sublists of *lst1* and *lst2*. [Each time a function *find_median* is called, both sublists of *lst1* and *lst2* should be of equal lengths].

   The following are the rules for creating sublists of *lst1* and *lst2*.

   a. Sublist of *lst1* should be a list starting from *med1* (including both middle entries when *lst1* contains an even number of entries) to the end of the list.

   b. Sublist of *lst2* should be a list starting from first element up to and including *med2* (including both middle entries when *lst2* contains an even number of entries).

For example,

```
find_median ([4],[3])  => 3
find_median ([4,6],[3,4])  => 4
find_median ([4,6,7],[3,4,5])  => 4
find_median([1,12,15,19],[2,13,17,19])=>14
```

Reminder:

- All the median calculations are integer division.