HS 608 Project 2: String and List Algorithms
Due Thurs, March 31, 2016, 11:59 pm
Upload to Canvas (algPack.py and testcases.txt)

This project will develop your algorithmic thinking as you code a package of string and list algorithms. You will define function bodies for each of the **10** algorithms described below. You may not change the function headers or descriptions. Be sure your code fulfills all the requirements of each function description. Your code should pass my limited doctests, but your **testing grade will be based on your own doctests**.

Your code may use the bracket operator, concatenation operator and the **len** function, but no other string or list functions/operations. NB: this means NO **slices**, NO **in** operator, etc, unless you write the code for the operator you use. The exception is that you are encouraged to use for loops throughout this assignment, so you may use the **in** operator within the for loop header (not the body).

For example, you may use:
for character **in** myString:
for i **in** range(start, **len**(lst), increment):

You must code the functions yourself, not use built-ins. Ex – you may not use str() in int2str, and may not use the repetition operator (*) in repeatList. However, you may use built-in functions ord and chr when you code functions str2int and int2str. Don't use another student's work. You should be able to explain your code in a code walk through.

Feel free to call on functions you've written for one algorithm as you later code another string or list algorithm.

Grading:

1. Correctness and quality of solution: 75% of your grade: Your program should return the correct value for all possible inputs that are valid according to the function description. Your program should **never crash on valid input such as the empty string**. For functions that require an integer input, test your program for negative, zero and positive integers.

2. Documentation: 5%: Include header documentation with your name, and a brief description of this collection of algorithms. Throughout your program, use identifiers that are meaningful. If there are any obscure constructs, be sure to clearly explain them. Each function **already has** a brief description of **what it accomplishes or returns**, **how its parameters are used (including type of parameters)**, **and preconditions**, if there are any. Therefore, you may keep these as your function descriptions; you do **not** have to write your own function descriptions.

3. Testing: 20%: For each function, test all its paths of execution, This will be in the range of 65-100 test cases. If you have an error in your code that you do not detect with testing, you will lose both points for testing and points for correctness. On the other hand, you get full testing when there is an error that you detect. Put all your test cases into doctests and **run in the terminal with**:

python algPack.py -v > testcases.txt

All test results will be written to file testcases.txt. They will not appear on the screen.

4. Do your own work, come up with your own solution to each algorithm.

Here are the **10** function descriptions and headers, along with very limited doctests:

```
#   NB: You may NOT use the repetition operator (*):  code this yourself.
def repeatList(lst, n):
    """
    Returns a new list that is equivalent to lst * n,  where * is the repetition operator.
    Assumes  n is a positive integer.
    Examples:
    >>> repeatList([1,2,3], 0)
    []
   >>> repeatList([1,2,3], 3)
    [1, 2, 3, 1, 2, 3, 1, 2, 3]

    """


def replace(lst, pos, item):
    """
    Modifies list lst so that its element at pos is replaced with item.
    Does not modify lst if pos is negative or pos >=  length of lst.
    Assumes n is an integer.
    Nothing is returned.
    Examples:
    >>> mylist = [1,2,3]
    >>> replace(mylist, 2, 'z')
    >>> mylist
    [1, 2, 'z']
    >>> mylist = [6,1,4,2,3]
    >>> replace(mylist, 5, 'z')
    >>> mylist
    [6, 1, 4, 2, 3]
    """


def countElem(lst, elem):
    """
    Returns the number of occurrences of element elem in list lst
    Examples:
    >>> countElem(['w', 'owl', 'w', 'awesome'],  'w')
    2
    """
```

```
def findLast(lst, elem):
    """
    Returns the index of the last occurrence of element elem in list lst or -1 if elem does not
occur in lst.
   Examples:
    >>> findLast(['w', 'owl', 'w', 'awesome'],  'w')
    2
    """
```

```
#   NB: You may not use built-in typecasting: code this yourself.
def string2list(s):
    """
    Returns a list whose elements are each of the characters in string s as ordered in s
    Examples:
    >>> string2list ('cat')
    ['c', 'a', 't']
    """
```

```
def lastOccurMostFreqElem(lst):
    """
    Returns the index of the last occurrence of the element that most frequently occurs in list lst
    or -1 if lst is empty. If elements occur with equal frequency, returns index of the last of these.
    Examples:
    >>> lastOccurMostFreqElem([0,0,2,2,0,2])
    5
   >>> lastOccurMostFreqElem([3,2,2,3])
    3
    """
```

```
#   NB: You may not use built-in slices: code this yourself.
def insert(s1, s2, pos):
    """
    Returns a new string that is equivalent to string s1 with string s2 inserted at index pos.
    Returns empty string if pos is negative or pos >  length of s1.
    Assumes n is an integer.
    Examples:
    >>> insert('', 'world!', 0)
    'world!'
    >>> insert('world', 's', 2)
    'wosrld'
    """
```

```
#   NB: You may not use built-in slices: code this yourself.
def substring(s, pos1, pos2):
    """
    Returns a new string consisting of the characters of string s from pos1 to pos2-1 inclusive.
    Returns the empty string if there is no valid substring beginning at pos1 and ending at pos2-1.
    Assumes both pos1 and pos2 are integers.

    Examples:
    >>> substring('cat', 1, 3)
    'at'
    """
```

```
#   NB: You may not use built-in typecasting: code this yourself.
def str2int(s):
    """
    Returns an integer that is the integer equivalent of the string s.
    Assumes that s is the string version of a valid integer.
    Examples:
    >>> str2int('-1000092')
    -1000092
    """
```

```
# NB: You may not use built-in str() or string template. Code this yourself.
def int2str(i):
    """
    Returns a string that is the string equivalent of valid integer i
    Examples:
    >>> int2str(-1000092)
    '-1000092'
    """
```