Assignment Guidelines:

- This assignment covers material up to Module 5 slide 28.
- Submission details:
  - Solutions to these questions must be placed in files a05q1.py, a05q2.py, a05q3.py, and a05q4.py, respectively, and must be completed using Python 3.
  - Download the interface file from the course Web page to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the Python section of the CS116 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page
  - Helper functions need design recipe elements but not examples and tests.
- Download the testing module from the course web page. Include import check in each solution file.
  - When a function produces a floating point value, you must use check.within for your testing. Unless told otherwise, you may use a tolerance of 0.00001 in your tests.
  - Test data for all questions will always meet the stated assumptions for consumed values.
- Restrictions:
  - Do not import any modules other than math and check.
  - You are always allowed to define your own helper functions, as long as they meet the assignment restrictions. Do not use Python constructs from later modules (e.g. loops) or the command zip. Use only the functions and methods as follows:
    * abs, len, max and min
    * Any method or constant in the math module
    * Type casting including int(), str(), float(), bool(), list() 
    * The command type()
    * Any basic arithmetic operation (including +, -, *, /, //, %, **) 
    * Any basic logical operators (not, and, or)
    * String or list slicing and indexing as well as string or list operations using the operators above and any methods.
    * input and print as well as the formatting parameter end and method format. Note that all prompts must match exactly in order to obtain marks so ensure that you do not alter these prompts.
    * Map, filter and lambda functions.
  - Do not mutate any passed parameters unless instructions dictate otherwise.
  - While you may use global constants in your solutions, do not use global variables for anything other than testing.
  - Read each question carefully for additional restrictions.
  - 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.
1 Word growth

Write a function

\[
grow_word(s)
\]
that consumes a string \(s\) and returns a string where every letter of \(s\) has been repeated a specified number of times according to its position in the original string \(s\) plus one. For example, the word bet would have the \(b\) repeated once, the \(e\) repeated twice and the \(t\) repeated thrice.

Sample

\[
grow_word("banana") => "baannnaaaaannnnnaaaaaa"
\]

You must use abstract list functions to solve this problem. Recursion is prohibited.

2 Benford’s Law

When trying to determine whether or not data is being forged, Benford’s Law can be used to help determine if a distribution of the numbers appears random. A collection of numbers is said to satisfy Benford’s law if and only if the proportion \(p_d\) of the leading digits \(d\) satisfies

\[
p_d = \log_{10} \left( \frac{d + 1}{d} \right)
\]

that is, \(p_d\) is the base 10 logarithm of \(d + 1\) divided by \(d\), for each value of \(d\) from 1 to 9. Translated, this means that non-zero numbers should have roughly \(p_1 = \log_{10}((1 + 1)/1) \approx 0.301\) or 30.1% of the numbers beginning with a 1, \(p_2 = \log_{10}((2 + 1)/2) \approx 0.176\) or 17.6% of the numbers beginning with a 2 and so on.

Your job is to write a function:

\[
\text{obey_benford}(L)
\]
that consumes a list of positive integers and determines whether or not the leading digits fall in a distribution that is within a tolerable distance form the actual data. To quantify what is a tolerable distance, statisticians use something called the \(\chi^2\) test which we simplify here for our purposes. Let \(n\) be the length of the list of numbers and let \(a_d\) be the total number of values in the list with leading digit \(d\). For each of the values \(d\) from 1 to 9, compute

\[
\frac{(a_d - n \cdot p_d)^2}{n \cdot p_d}
\]

(namely, the squared difference of \(a_d\) minus \(n\) times \(p_d\) all divided by \(n\) times \(p_d\)) and sum up these 9 values. It is this value that you are returning and you should check to within a tolerance of 0.00001.\(^1\)

As a sample, if your list of numbers was \(L = [1, 2, 1337]\), then you would return

\[
\frac{(2 - 3p_1)^2}{3p_1} + \frac{(1 - 3p_2)^2}{3p_2} + 3p_3 + 3p_4 + 3p_5 + 3p_6 + 3p_7 + 3p_8 + 3p_9 = 3.32219532272341...
\]

You must use abstract list functions to solve this problem. Recursion is prohibited.

Sample:

\[
L = [1, 2, 1337]
\]

\[
\text{obey_benford}(L) => 3.32219532272341
\]

\[
\begin{array}{c}
+ [6] * 67
\begin{array}{c}
\begin{array}{c}
+ [7] * 58
\begin{array}{c}
\begin{array}{c}
+ [8] * 51
\begin{array}{c}
\begin{array}{c}
+ [96] * 46
\begin{array}{c}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\text{obey_benford}(L) => 0.002362217189127143
\]

\(^1\)Note in practice, if this value is less than or equal to 15.507, the data is said to be within a tolerable distance to random data. The 15.507 comes from the 8 degrees of freedom we have in this problem and from standard \(\chi^2\) distribution tables.
3 Products

Write a function

\[ \text{num_zeroes}(L) \]

that consumes a list of integers \( L \) and returns the following value. First let \( p \) be the product of all values in \( L \). This function will return the number of zeroes at the end of the number \( p \). Note that an empty product has a value of 1 (and 1 does not end in any zeroes) and if the above product is 0 then you should return 1. All helper functions in this problem must use accumulative recursion. You are forbidden from using abstract list functions.

Sample:

\[ \text{num_zeroes}([2, 5, 6, 100]) \Rightarrow 3 \]

4 Recamán’s Function

Recamán’s function is defined on the natural numbers as follows:

\[ r(n) = \begin{cases} 
0 & \text{if } n = 0 \\
(r(n - 1) - n) & \text{if } n > 0 \text{ and } r(n - 1) - n \geq 0 \text{ and } r(i) \neq r(n - 1) - n \text{ for any } 0 \leq i < n \\
r(n - 1) + n & \text{otherwise}
\end{cases} \]

If we think of this as a sequence, then the above can be reworded as follows: every \( n \)th term is equal to the previous term minus \( n \) if this is positive and hasn’t already appeared in the sequence yet and is equal to the previous term plus \( n \) otherwise. Below is a table with the first few terms:

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r(n) )</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>13</td>
</tr>
</tbody>
</table>

For example, to compute \( r(3) \), notice that \( r(2) = 3 \) and \( r(2) - 3 = 0 \) which is equal to \( r(0) \) and hence, we make \( r(3) = r(2) + 3 = 6 \). Write a function

\[ \text{recaman}(n) \]

that returns \( r(n) \).

Sample:

\[ \text{recaman}(0) \Rightarrow 0 \]
\[ \text{recaman}(6) \Rightarrow 13 \]

You must use accumulative recursion to solve this problem. You are forbidden from using abstract list functions.