Assignment Guidelines.

- This assignment covers material in Module 5.
- Submission details:
  - Solutions to these questions must be placed in files a05q1.rkt, a05q2.rkt, a05q3.rkt, and a05q4.rkt, respectively, and must be completed using Racket Intermediate Student with lambda.
  - Unless otherwise indicated in the question you may use only the built-in functions and special forms introduced in the lecture slides from CS115 up to and including the modules covered by this assignment.
  - 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 CS115 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page.
  - For the design recipe, helper functions only require a purpose, a contract and an example.
- Restrictions:
  - Unless the question specifically describes exceptions, you are restricted to using the functions and special forms covered in or before Module 5.
  - 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. Keeping Count.

Write a function \((\text{count-over } L \ n)\). It consumes a \((\text{listof } \text{Num})\) and a \(\text{Num}\), and returns the number of values in \(L\) that are greater than \(n\).

For example,
\[
\text{(count-over (list 2 4 5 9 7 6 3 8) 4)} \Rightarrow 5
\]
\[
\text{(count-over (list 2 4 5 9 7 6 3 8) 10)} \Rightarrow 0
\]

Write a function \((\text{keep-counts } L \ \text{threshold} \ \text{mincount})\), where \(L\) is a \((\text{listof (listof Num)})\), and the other parameters are both \(\text{Num}\).
This function returns a list containing all the values in \(L\) which contain at least \(\text{mincount}\) values greater than \(\text{threshold}\).

For example,
\[
\text{(keep-counts (list (list 1 1 2 5) (list 1 4 4 1) (list 6 6 4 7)) 2 2)} \Rightarrow (\text{list (list 1 4 4 1) (list 6 6 4 7)})
\]

Since only these two lists contain at least two values greater than 2.

2. Pascal’s Triangle. Binomial coefficients are defined by the following function:

\[
C(n,k) = \frac{n!}{k!(n-k)!}
\]

We define this only for \(0 \leq k \leq n\).

(From assignment 2 recall that the factorial of a natural number is \(n! = 1 \times 2 \times 3 \times 4 \times \ldots (n-1) \times n\).)

The binomial coefficients, when drawn in a table, form Pascal’s Triangle.

\[
\begin{array}{cccc}
C(0,0) & 1 \\
C(1,0) & C(1,1) & 1 & 1 \\
C(2,0) & C(2,1) & C(2,2) & \Rightarrow 1 & 2 & 1 \\
C(3,0) & C(3,1) & C(3,2) & C(3,3) & 1 & 3 & 3 & 1 \\
C(4,0) & C(4,1) & C(4,2) & C(4,3) & C(4,4) & 1 & 4 & 6 & 4 & 1 \\
\end{array}
\]

Write a function \((\text{pascal } n)\) that returns a \((\text{listof (listof Nat)})\) containing the first \(n\) rows of Pascal’s triangle. (That is, the rows from 0 to \(n - 1\).)

For example,
\[
\text{(pascal 5)} \Rightarrow
(\text{list (list 1)}
\text{ (list 1 1)}
\text{ (list 1 2 1)}
\text{ (list 1 3 3 1)}
\text{ (list 1 4 6 4 1)})
\]
3. Substrings.

Exercise
Write a function (substrings-w-len s n). It consumes a Str and a Nat, and returns all the substrings of length n.
For example, (substrings-w-len "foobar" 4) => (list "foob" "ooba" "obar")

Exercise
Write a function (last-substring-len s n). It consumes a Str and a Nat, and returns the substring of s with length n that comes last alphabetically.
Remember to consider if your function Requires anything of its arguments!
For example,
(last-substring-len "foobar" 4) => "ooba"
(last-substring-len "thequickbrownfoxjumpsoverthelazydogs" 7) => "xjumpso"

Hint
(string<=? "" s) => #true for every string s.


There are many circumstances where computer programs need to be able to identify the words in text. You could imagine a program that interprets computer languages, or analyses poetry, or any number of other tasks.
The first step of this is to use the spaces between words to split a long string into single words.

Exercise
Write a function (string-split s) that consumes a Str and returns a (listof Str) containing all the individual words in s.
For example,
(split-string "I'll be back") => (list "I'll" "be" "back")
(split-string "Arma virumque cano, Troiae qui primus ab oris")
=> (list "Arma" "virumque" "cano," "Troiae" "qui" "primus" "ab" "oris")
You may assume that s does not start or end with a space, and that it does not contain duplicate spaces.