Assignment Guidelines.

- This assignment covers material in Module 6.
- Submission details:
  - Solutions to these questions must be placed in files a6q1.rkt, a6q2.rkt, a6q3.rkt, and a6q4.rkt, respectively, and must be completed using Racket.
  - 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. A list of functions described in each module of the lecture slides can be found at [https://www.student.cs.uwaterloo.ca/~cs115/built_in](https://www.student.cs.uwaterloo.ca/~cs115/built_in)
  - 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:
  - You should expect to use recursion on every question.
  - Read each question carefully for additional restrictions.

Do not use `map`, `foldr`, `filter`, `length`, `append`, or `range` on this assignment.

- 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. The first and the last.

   (a) last. Write a function `last` that consumes a non-empty `(listof Any)` and returns the last item in the list. (Like `first`, but in reverse.)
   
   (last (list 4 9 7 5)) => 5

   (b) all-but-last. Write a function `all-but-last` that consumes a non-empty `(listof Any)` and returns the same list with every item except the last. (List `rest`, but in reverse.)
   
   (all-but-last (list 4 9 7 5)) => (list 4 9 7)

2. Counting up and down.

   Write a recursive function `my-range` that duplicates the behaviour of the builtin `range` function.

   (my-range 4 13 3) => (list 4 7 10)
   (my-range 17 8 -2) => (list 17 15 13 11 9)
3. **Rhythms and Logarithms.**

The logarithm base $b$ of a number $n$ is the number power to which $b$ must be raised in order to get $n$. In other words, if $f(x) = b^x$, then the inverse function is $f^{-1}(x) = \log_b x$.

**Exercise**

Write a function `(floor-log b n)` that consumes two `Num` returns the greatest `Nat` less than or equal to $\log_b n$.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(floor-log 2 31)</code></td>
<td>4</td>
</tr>
<tr>
<td><code>(floor-log 2 32)</code></td>
<td>5</td>
</tr>
<tr>
<td><code>(floor-log 2 33)</code></td>
<td>5</td>
</tr>
<tr>
<td><code>(floor-log 3 100)</code></td>
<td>4</td>
</tr>
</tbody>
</table>

You may assume that $n \geq 1$ and that $b > 1$.

!! Do not use the `log` function or any similar function.

4. **Catalan numbers with Recursion.** Earlier we defined the Catalan numbers by the equation:

\[
C(n) = \frac{(2n)!}{n!(n+1)!}
\]

The Catalan numbers may also be define recursively, in at least two ways:

\[
(2) \quad C(0) = 1 \text{ and } C(n) = \sum_{i=0}^{n-1} C(i)C(n-i-1)
\]

\[
(3) \quad C(0) = 1 \text{ and } C(n) = \frac{2(2n-1)}{n+1}C(n-1)
\]

**Exercise**

(a) **Generating using recursion.** Write a function `(catalan-1 n)` that returns the $n$th Catalan number. Use the definition of Catalan numbers given by one of Equations 1, 2, or 3.

For example,

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(catalan-1 3)</code></td>
<td>5</td>
</tr>
<tr>
<td><code>(catalan-1 4)</code></td>
<td>14</td>
</tr>
</tbody>
</table>

(b) **A different solution.** Write a function `(catalan-2 n)` that returns the $n$th Catalan number. Use the definition of Catalan numbers given by one of Equations 1, 2, or 3, but a different definition than you used in part (a).

For example,

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(catalan-2 3)</code></td>
<td>5</td>
</tr>
<tr>
<td><code>(catalan-2 4)</code></td>
<td>14</td>
</tr>
</tbody>
</table>

!! This is the same question as on a previous assignment, but here you must use recursion for all parts. Do not use `map` or `foldr`, anywhere (not even in a helper function). Ensure `catalan-1` works even if `catalan-2` is not defined, and vice-versa.