Assignment Guidelines.

- This assignment covers material in Modules 6 and 7.
- Submission details:
  - Solutions to these questions must be placed in files a06q1.rkt, a06q2.rkt, a06q3.rkt, and a06q4.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:
  - 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. Extracting a value from a list.

   The built-in function (list-ref L i) consumes a (listof Any) and a Nat, and returns the ith item in L, where (first L) is item zero.

   (list-ref (list "a" "b" "c") 0) => "a"
   (list-ref (list "a" "b" "c") 2) => "c"

   Using recursion, write a function (my-list-ref L i) that duplicates the behaviour of list-ref.

2. Searching in Strings.

   Write a function (find s pattern) that looks through s and returns the smallest index in s where pattern exists as a substring.

   You may assume that s contains pattern somewhere.

   For example,
   (find "one fish two fish red fish blue fish" "fish") => 4
   (find "abcabcabc" "ab") => 0
3. **Skip Counting.**

Write a function `skip-sum L` that consumes a `(listof Num)` and returns the sum of the alternating values in the list, including the first value.

For example,

- `(skip-sum (list 2 3 5))` => `(+ 2 5)` => `7`
- `(skip-sum (list 2 3 5 7))` => `(+ 2 5)` => `7`
- `(skip-sum (list 2 3 5 7 11))` => `(+ 2 5 11)` => `18`

4. **Count like an Egyptian.** An **Egyptian Fraction** is the sum of distinct fractions where the numerators are all 1. As it turns out, every positive rational number can be expressed as an Egyptian fraction. For example, 

\[
\frac{1}{2} = \frac{1}{3} + \frac{1}{6}
\]

Write a function `egyptian-fraction n` that consumes a non-negative `Num` and returns a `(listof Num)`, where each value is a fraction with denominator 1, and which sums to `n`. There must be no duplicated values in the answer list.

For example,

- `(egyptian-fraction 1)` => `(list 1/2 1/3 1/6)`
- `(egyptian-fraction 7/6)` => `(list 1/2 1/3 1/4 1/12)`
- `(egyptian-fraction 0)` => `()`

On the other hand,

- `(egyptian-fraction 1)` => `(list 1/2 1/2)` is **incorrect** since `1/2` appears more than once in the answer.

Note: the answers can be extremely long, and take excessively long to find. Don’t test with random decimal numbers. Instead, choose your answer first, then work backwards to get a starting `n`.

You may not get the answer you expect! 

\[
\frac{7}{24} = \frac{1}{6} + \frac{1}{8} = \frac{1}{4} + \frac{1}{12}
\]

Either answer is valid, and will receive full marks.

**Hint**

Count the numbers in order 2, 3, 4..., and from these get the reciprocals: \( \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots \)

**Bonus**

For a bonus, write a function `egyptian? L n` that consumes a `(listof Num)` and a `Num`. It returns `#true` if `L` is an Egyptian Fraction that represents `n`.

For example,

- `(egyptian? (list 1/2 1/3 1/6) 1)` => `#true`
- `(egyptian? (list 1/3 1/4 1/5 1/6 1/20) 1)` => `#true`
- `(egyptian? (list 1/2 1/2) 1)` => `#false`