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. Prime. A prime number is a Natural Number ≥ 2 that is divisible only by 1 and itself.

Exercise Write a function `(prime? n)` that consumes a `Nat`. It returns `#true` if `n` is prime, and `#false` otherwise.

```
(check-expect (prime? 17) #true)
(check-expect (prime? 6) #false)
(check-expect (prime? 5) #true)
(check-expect (prime? 1) #false)
```

2. More Books. Recall that in the previous assignment we defined a `Book`:

```
;; a Book is a (list Str Str Nat)
```

Here is a larger `(listof Book)` that we will use in examples:

```
(define booklist
  (list
   (list "Liu" "The Three Body Problem" 302)
   (list "Nawaz" "Songs for the End of the World" 400)
   (list "Heinlein" "The Moon Is a Harsh Mistress" 382)
   (list "Weir" "The Martian" 369)
   (list "Clancy" "The Sum of All Fears" 798)
   (list "Rowling" "Harry Potter and the Philosopher's Stone" 223)
   (list "Nawaz" "Bone and Bread" 445)
   (list "Heinlein" "Stranger in a Strange Land" 408)
   (list "Heinlein" "Starship Troopers" 263)
   ))
```

Exercise Write a function `(total-pages books author)`. It consumes a `(listof Book)` and a `Str`, and returns the total number of pages in `books` written by `author`.
For example,

```
(check-expect (total-pages booklist "Heinlein") 1053)
(check-expect (total-pages booklist "Nawaz") 845)
(check-expect (total-pages booklist "Seuss") 0)
```

3. Common Divisors.

Exercise Write a function `(common-divisors n1 n2)`. It consumes two positive `Nat` and returns a `(listof Nat)` containing all the numbers that divide both `n1` and `n2`.
The numbers should be in increasing order. For example,

```
(check-expect (common-divisors 24 60) (list 1 2 3 4 6 12))
```

4. Pascal's Triangle. Binomial coefficients are defined by the following function:

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

We define this only for $0 \leq k \leq n$.
An implementation of the factorial function $n!$ is provided in the interface file.
The binomial coefficients, when drawn in a table, form *Pascal's Triangle*.

$$\begin{array}{ccccccc} C(0,0) & & & & & & \\ C(1,0) & C(1,1) & & & & & \\ C(2,0) & C(2,1) & C(2,2) & & & \longrightarrow & \\ C(3,0) & C(3,1) & C(3,2) & C(3,3) & & & \\ C(4,0) & C(4,1) & C(4,2) & C(4,3) & C(4,4) & & \end{array} \begin{array}{ccccc} 1 & & & & \\ 1 & 1 & & & \\ 1 & 2 & 1 & & \\ 1 & 3 & 3 & 1 & \\ 1 & 4 & 6 & 4 & 1 \end{array}$$

Exercise

Write a function `(pascal n)` that returns a `(listof (listof Nat))` containing the first n rows of Pascals triangle.
For example,

```
(check-expect (pascal 5)
              (list (list 1)
                    (list 1 1)
                    (list 1 2 1)
                    (list 1 3 3 1)
                    (list 1 4 6 4 1)))
```