Assignment Guidelines.

- This assignment covers material in Modules 3 and 4.
- Submission details:
  - Solutions to these questions must be placed in files `a03q1.rkt`, `a03q2.rkt`, `a03q3.rkt`, and `a03q4.rkt`, respectively, and must be completed using Racket *Intermediate Student*.
  - 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 Modules 3 and 4.
  - 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. A Piecewise Function.

Exercise

Figure 1 shows a piecewise-linear function $f(x)$.
Write a function `(f x)` that implements this function in Racket.
For example,

```
(check-expect (f 0) 0)
(check-expect (f 4.5) 2)
```

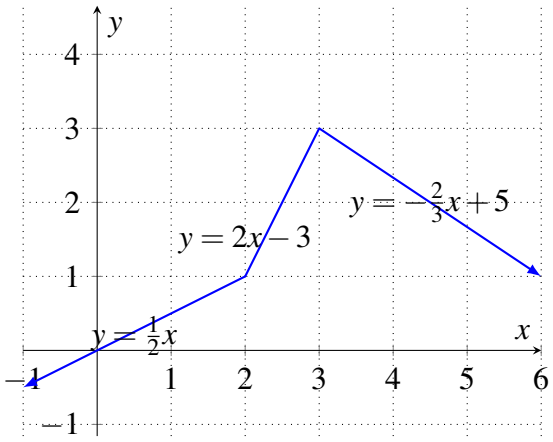

FIGURE 1. A plot of $f(x)$

2. Broadcast Standards.

It is desirable to transform all four-letter words from a body of text.

Exercise

Write a function `(censor L)` that consumes a non-empty `(listof Str)`. It returns a `Str` which contains all the words in `L`, where each `Str` of length four has had all the letters except first and last replaced with `"*"`, and spaces have been placed between the words.

```
(check-expect
 (censor (list "When" "the" "hurlyburly's" "done,"
               "When" "the" "battle's" "lost" "and" "won."))
 "W**n the hurlyburly's done, W**n the battle's l**t and w**.")
(check-expect
 (censor (list "Love" "the" "life" "you" "live."
               "Live" "the" "life" "you" "love."))
 "L**e the l**e you live. L**e the l**e you love.")
```

Hint

Remember that you can stick together a `(listof Str)` into a `Str` using **`foldr`**. You'll need a little more here....

3. Which Came First?.

The ISO 8601 standard describes a handful of compact date formats.
For the purposed of this assignment we consider dates including only the year, month, and day.
Dates are represented in the form YYYY-MM-DD.
For example, June 10, 2020 is represented as `"2020-06-10"`.

Exercise

Write a function `(before? d1 d2)` that consumes two `Str`, which must contain dates in ISO 8601 format. The function returns `#true` if `d1` comes before `d2`, and `#false` otherwise.

```
(check-expect (before? "1066-10-14" "1399-09-30") #true)
(check-expect (before? "1955-11-05" "1955-11-05") #false)
(check-expect (before? "1969-07-20" "0899-10-26") #false)
```

4. The `sort` function. Sorting algorithms are often studied in Computer Science classes because the problem is easy to describe, and there is a wide range of interesting algorithms.

In real life, however, you should almost certainly use the built-in `sort` function.

The function `sort` consumes a list `L`, and a predicate function `P`. `P` consumes two arguments; if it returns `#true`, the first argument will appear in the final list somewhere to the left of the second argument.

The `sort` function will repeatedly call the predicate with pairs of items from the list.

For example, consider `(sort (list 1 3 2 4) <)`. We use the predicate function `<`.

- Since `(< 1 4) => #true`, we know that `1` will appear in the answer somewhere to the left of `4`.
- Since `(< 2 3) => #true`, we know that `2` will appear in the answer somewhere to the left of `3`.

For example:

- `(sort (list 1 3 2 4) <) => (list 1 2 3 4)` since `(< 1 2)`, `(< 2 3)`, etc.
- `(sort (list 1 3 2 4) >) => (list 4 3 2 1)` since `(> 4 3)`, `(> 3 2)`, etc.

We want to sort the names of some musical groups. Since so many bands start with "the", we will ignore that word at the front of each band name. Otherwise, sort alphabetically.

Exercise

Write a function `(namefirst? n1 n2)` that consumes two `Str`. It returns `#true` if `n1` should appear before `n2` in the list, and `#false` otherwise.
For simplicity, consider only uppercase strings.
Test your function both by itself, and using `sort`.

For example,

- Alphabetically, "ARCADE FIRE" comes before "U2", so:
  `(check-expect (namefirst? "U2" "ARCADE FIRE") #false)`
- We ignore the "THE" in "THE BEATLES", so:
  `(check-expect (namefirst? "THE BEATLES" "MOXY FRUVOUS") #true)`

I have define the function `sort-bands` as follows. Here is an example of its usage:

```
;; (sort-bands L) Sort L alphabetically, ignoring the word "THE" at front.
;; sort-bands: (listof Str) -> (listof Str)
;; Examples:
(check-expect
 (sort-bands
  (list "THE BEATLES" "MOXY FRUVOUS" "THEY MIGHT BE GIANTS" "U2" "PAUL SIMON"
        "THE THE" "AD VIELLE QUE POURRA" "THE ROLLING STONES"
        "THE DUBLINERS" "ARCADE FIRE" "THE YARDBIRDS" "THE SMITHS"))
 (list "AD VIELLE QUE POURRA" "ARCADE FIRE" "THE BEATLES" "THE DUBLINERS"
       "MOXY FRUVOUS" "PAUL SIMON" "THE ROLLING STONES" "THE SMITHS"
       "THE THE" "THEY MIGHT BE GIANTS" "U2" "THE YARDBIRDS"))

(define (sort-bands L)
  (sort L namefirst?))
```