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.

Figure 1 shows a piecewise-linear function \( f(x) \).
Write a function \((f \ x)\) that implements this function in Racket.
For example,
\[
\begin{align*}
(f 1) &= 2 \\
(f 2) &= 1.5 \\
(f 4.5) &= 2
\end{align*}
\]

2. Add or Multiply.

Write a function \((\text{add-mul} \ L)\). The function consumes a \((\text{listof Int})\).
It performs a series of mathematical operations. It starts with an answer of zero, then starting at the right, walks through the list \( L \), and does the following operations:

(1) If the next item in \( L \) is even, it adds the item to the answer so far.
(2) If the next item on \( L \) is odd, it multiplies the answer by the item.

For example, given \((\text{list} 4 \ 8 \ 5 \ 3 \ 2)\):
- 2 is even, so add 0 and 2, giving 2;
- 3 is odd, so multiply 2 and 3, giving 6;
- 5 is odd, so multiply 6 and 5, giving 30.
- 8 is even, so add 30 and 8, giving 38.
- 4 is even, so add 38 and 4, giving 42.

\((\text{add-mul} \ (\text{list} 4 \ 8 \ 5 \ 3 \ 2))\) \(\Rightarrow 42\)
3. **Transit Pass.**

In Berlin, you can buy transit passes that are good for 1 day, 7 days, or 30 days. When you first use a pass, a machine stamps it with the week and weekday. We will represent the weekday by a number 0–6, 0 being Monday, 1 Tuesday, and so on.

For example, a pass started on Wednesday of the 5th week of the year would be stamped “5-2”.

Write a function `(good-ticket? type stamp-week stamp-day curr-week curr-day)`. The function returns `#true` if the ticket is still good, and `#false` otherwise.

- `type` is a `Str`, one of "day", "week", or "month".
- `stamp-week` and `stamp-day` represent the date on which the ticket was stamped.
- `curr-week` and `curr-day` store the current date.

Any ticket that is stamped in the future is invalid.

For example,

- `(good-ticket? "day" 16 4 16 4) => #true` Stamped today.
- `(good-ticket? "day" 16 4 16 5) => #false` Stamped yesterday.
- `(good-ticket? "day" 16 4 16 3) => #false` Stamped tomorrow!
- `(good-ticket? "week" 16 4 16 4) => #true` Stamped today.
- `(good-ticket? "week" 15 4 16 3) => #true` Stamped on Friday. It’s now Thursday, so still good.
- `(good-ticket? "week" 15 4 16 4) => #false` Stamped a whole week ago, so expired.

**Hint**

Write a function to calculate the number of days between one week-day and another. For example, day 5 of week 6 is 9 days after day 3 of week 5.

4. **Pig Latin.**

Pig Latin is a way of rearranging letters in English words for fun. For example, the sentence “pig latin is stupid” becomes “igpay atinlay isway upidstay”.

Vowels (‘a’, ‘e’, ‘i’, ‘o’, and ‘u’) are treated separately from the consonants (any letter that isn’t a vowel). For simplicity, we will consider ‘y’ to always be a consonant.

Although various forms of Pig Latin exist, we will use the following rules:

1. Words of two letters or less simply have “way” added on the end. So “a” becomes “away”.
2. In any word that starts with consonants, the consonants are moved to the end, and “ay” is added. If a word begins with more than two consonants, move only the first two letters. So “hello” becomes “ellohay”, and “string” becomes “ringstay”.
3. Any word which begins with a vowel simply has “way” added on the end. So “explain” becomes “explainway”.

Write a function `(pig-latin L)` that consumes a non-empty `(listof Str)` and returns a `Str` containing the words in L converted to Pig Latin. Each value in L should contain only lowercase letters and have a length of at least 1.

For example,

- `(pig-latin (list "this" "is" "a" "crazy" "exercise"))` => "isthay isway away azycray exerciseway"

**Hint**

You may use the `join` function developed on a previous assignment: either your own code, or the code from the posted solution.