Below are several exercises that we will be covering in the upcoming tutorial (Friday, Mar 18). We will release these exercises in advance of the tutorial so that you get a chance to attempt the exercises yourself before we discuss them in tutorial. The course staff running the tutorial will go through each problem (time permitting) and show their process for how they would complete these questions in an assignment/exam setting. Being familiar with the questions before attending tutorial will help to ensure that you get the most out of our explanations.

Note that tutorials are not mandatory in CS135. We never cover any new material that you won’t have seen in lectures, and instead choose to highlight techniques and concepts from the most recent lectures to give you some extra practice. If you are very comfortable with the material that we covered and could easily complete the below questions, then you aren’t obligated to attend tutorials. Be very cautious with this though since sometimes questions can seem easy until you actually sit down and attempt them.

**Question 1: Perfect Squares**

Using filter and lambda (but no other named helper functions), write a function *(perfect-squares lst)* which consumes a list of natural numbers (lst) and produces a new list containing only numbers from the original list that are perfect squares.

*Example:*

(perfect-squares (list 2 4 8 16)) → (list 4 16)

**Question 2: Secret Messages**

You have a secret message and you’d like to make it impossible for anyone else to read it. Write a function *(obfuscate texts)* which consumes a list of strings (texts) and produces a new list of obfuscated strings. Each string should have all spaces removed and all lowercase vowels replaced according to the following mapping:

- a → @
- e → #
- i → !
- o → &
- u → %

Note: Your message only contains lowercase letters, numbers, and symbols so you don’t need to worry about mapping capitalized vowels.
Example:

obfuscate (list “hello” “how nice to see you”)) → (list “h#ll&” “h&wn!c#t&s##y&%”)

Question 3: Searching the Web

Web search algorithms use a rich set of features to decide which links are relevant to a given search query (such as “how do I pass CS 135?”). In this problem, you will simulate a web search algorithm. Thankfully, you only have to deal with queries that are one word long.

Your search algorithm may encounter two types of documents, either a WebPage or a Video. For this exercise, a WebPage is described by a list of symbols representing the words in the page and a Video by a description (list of symbols) and the number of seconds the video is long. You may use the following data definitions when writing your solution:

;;; A Document is a WebPage or a Video.

(define-struct webpage (words))
;;; A WebPage is a (make-webpage ne-(listof Sym))

(define-struct video (desc seconds))
;;; A Video is a (make-video ne-(listof Sym) Num)
;;; Requires: Num > 0.

Write a function (web-search query docs) which takes a one-word query (represented by a single symbol) and a list of documents (docs) and produces a list of booleans. For each document, the function should produce true if the document is relevant to the query and false otherwise.

• A WebPage is relevant to the query if the exact query symbol is in the list of words on the webpage.
• A Video is relevant to the query if it is at least 5 seconds long and the query symbol is in the list of words in its description.

Note: Your solution must use map, cond, and either local or lambda for all helper functions.

Example:

(define cat-page (make-webpage (list ’cats ’are ’very ’cool)))
(define dog-page (make-webpage (list ’dogs ’are ’mammals)))
(define cat-video (make-video (list ’video ’of ’my ’cats ’being ’cute) 4))
(web-search ’cats (list cat-page dog-page cat-video)) → (list true false false)