Lab 9: Processing two lists

Create a separate file for each question. Keep them in your “Labs” folder, with the name `liqj` for Lab i, Question j.

Download the headers for each function from the file `labinterface9.rkt` linked off the “Labs” page on the course Web site.

After you have completed a question (except class exercises), including creating tests for it, you can obtain feedback by submitting it and requesting a public test. Follow the instructions given in the Style Guide.

This lab makes use of the following structure and data definitions:

(*define-struct event* (*type dur*))
;; An **event** is a structure (*make-event t d*), where
;; *t* is a symbol (the type of event) and
;; *d* is a positive integer (the duration in minutes).

After you have completed a question (except class exercises), including creating tests for it, you can obtain feedback by submitting it and requesting a public test. Follow the instructions given in the Style Guide.
Language level: Beginning Student with List Abbreviations

1. *[Class exercise with lab instructor assistance]*

   (a) Create a function *extract-values* that consumes a list of keys and an association list and produces the list of values associated with the keys.

   (b) Create a function *kth* that consumes a list and a number *k* and produces the item in the *k*th position in the list if it exists, or *false* otherwise. The first item in the list is in position 0.

2. Create a function *create-events* that consumes two lists of the same length, a list of symbols and a list of positive integers, and produces a list of *event*s.

3. Create a function *extract-bad* that consumes two strings, *bad-part* and *whole-string*, and produces the string formed by removing *bad-part* from *whole-string*. The characters in *bad-part* will appear in *whole-string*, in order, though not necessarily consecutively. For example, (*extract-bad* "aadba" "abracadabra") will produce "brcaar". Here the eliminated characters are shown in upper case: "AbrAcaDaBrA".

4. Create a function *subseq-string* that consumes two strings of letters in lowercase (a pattern and a target) and produces a string showing how much of the pattern can be found in the target. The string that is produced will have all the same letters as the target, but with some of them changed to uppercase to show where and how much of the pattern can be found. For example, with pattern "hat" and target "chordality", your function will produce "cHordAliTy". For the pattern "hat" and the target "hand", your function will produce "HAnd". The pattern will be found at most once, with each character as far to the left as possible: (*subseq-string* "hat" "hhat") produces "HhAT" and (*subseq-string* "hat" "hathat") produces "HAThat". Write a helper function that handles lists of characters.

5. *Optional open-ended question* In Lab 6 you were asked to consider functions on numbers represented in binary as lists of booleans. Now consider functions on pairs of numbers encoded this way. Write functions to add two such numbers, subtract one number from another, comparing two numbers to see which is larger, and so on.