• **Academic Integrity Links**
  
  - [https://uwaterloo.ca/academic-integrity/basic-info](https://uwaterloo.ca/academic-integrity/basic-info)
  - [https://uwaterloo.ca/library/get-assignment-and-research-help/academic-integrity/academic-integrity-tutorial](https://uwaterloo.ca/library/get-assignment-and-research-help/academic-integrity/academic-integrity-tutorial)

• For this and all subsequent assignments, you are expected to use the design recipe when writing functions from scratch, including helper functions.

• If you write a solution that includes a wrapper function, you are only required to include examples and tests for the wrapper function, and not for the function that it wraps.

• 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.

• Do **not** copy the purpose directly from the assignment description. The purpose should be written in your own words and include references to the parameter names of your functions.

• Unless otherwise indicated by the question, you may only use the built–in functions and special forms introduced in the lecture slides from CS115 up to and including the modules covered by this assignment. A list of functions described in each module of the lecture slides can be found at [https://www.student.cs.uwaterloo.ca/~cs115/built_in](https://www.student.cs.uwaterloo.ca/~cs115/built_in).

You are always allowed to use functions you define yourself.

• 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.

• Do **not** send any code files by email to your instructors or tutors. Course staff will **not** accept it as an assignment submission. Course staff will **not** debug code emailed to them.

• Read each question carefully for restrictions.

• Test data for all questions will always meet the stated assumptions for consumed values.

• You may post general assignment questions using the discussion groups on LEARN. Read the guidelines for posting questions. Do NOT post any code as part of your questions.

• Check MarkUss and your basic test results to ensure that your files were properly submitted. In most cases, solutions that do not pass the basic tests will not receive any correctness marks.

• Any string or symbol constant values must **exactly** match the descriptions in the questions. Any discrepancies in your solutions may lead to a severe loss of correctness marks. Basic tests results will catch many, but not necessarily all of these types of errors.

• Read the course Web page for more information on assignment policies and how to organize and submit your work. Follow the instructions in the Style Guide. Your solutions should be placed in files `a07qY.rkt`, where `Y` is a value from 1 to 3.

• Since each file you submit will contain more than one function, **it is very important that your code runs.** If your code does not run then none of the functions can be tested for correctness.

• You must provide the data definition and template in your solutions only when the question specifically indicates they are required for compound data types described in the question. If you create any additional data types that are beyond the question description, your program file should include a data definition and a template for each additional data type. Make sure to comment any templates that you include.

Copyright ©2018
Language level: Beginning Student With List Abbreviations
Coverage: Module 7 – Processing Two Lists

Useful data definitions:

(define-struct vehicle (manufacturer model fuelcapacity kmsperlitre))
;; A Vehicle is a (make-vehicle Str Str Nat Num)
;; requires:
;; fuelcapacity >0 is recorded in litres, and
;; kmsperlitre >0 is recorded in kilometres per litre.

;; A Message is a Str that contains only uppercase letters.

;; A CodeKey is a (listof Nat) that contains unique numbers
;; in ascending order.

1. (a) Write a function driven-distances which consumes a list of Vehicle (alovehicle) and a list of Num (alonum) (of the same length as alovehicle) which indicates, for the corresponding vehicle, the amount of fuel in its fuel tank in litres, and produces a new list containing the total distance that could be driven by each individual vehicle. If the stated amount of fuel in the tank exceeds the given fuel capacity, then the stated amount is in error. Use the fuel capacity instead of the stated amount of fuel in your computations instead in this situation.

Examples:
  • (driven-distances
    (list (make-vehicle "Mazda" "3" 40 20)
    (make-vehicle "Ford" "Explorer" 78.5 7.5))
    (list 30 52.5)) ⇒ (list 600 393.75)
  
  • (driven-distances
    (list (make-vehicle "Mazda" "3" 40 20))
    (list 50)) ⇒ (list 800)

(b) Write a function total-driven-distance which consumes a list of Vehicle (alovehicle) and a list of Num (alonum) (of the same length as alovehicle) which indicates, for the corresponding vehicle, the amount of fuel in its fuel tank in litres, and produces the total distance that could be driven by all the vehicles in alovehicle. If the stated amount of fuel in the tank exceeds the given fuel capacity, then the stated amount is in error. Use the fuel capacity instead of the stated amount of fuel in your computations instead in this situation. You may use your driven-distances function as a helper.

Examples:
2. Write a function `n-duplicates` which consumes a natural number \( n > 0 \) and a list of Int (`aloi`) (which is sorted in non-decreasing order) and returns a list of Int, sorted in non-decreasing order, which contains exactly one copy of every integer for which `aloi` contains at least \( n \) copies.

**Remark:** Having written the `n-duplicates` function as specified here, we could easily remove the constraint that `aloi` must already be sorted, by using a modified version of the `sort` function developed in Module 6.

**Examples:**

- `(n-duplicates 3
  (list -5 -1 0 1 1 2 2 3 4 5 5 7 8))`  
  ⇒ empty
- `(n-duplicates 3
  (list -5 -3 -3 -3 -3 -1 0 1 1 2 2 2 3 4 5 5 5 7 8))`  
  ⇒ (list -3 2 5)

3. Suppose someone was trying to conceal a secret message by embedding it in random extra letters. Write a function called `decode` that consumes a `Message` (`cypher`) and a `CodeKey` (`key`), and produces a `Message` that represents the decoded text. The `key` contains the index positions of the secret message in `cypher`. For example

```
(decode "KSGIAOELWIERUXOVQIERCZXIETSQNY"
  (list 3 7 14 19 21 27))  
⇒ "ILOVECS".
```

It is possible that the `key` contains numbers that are invalid index positions (i.e. they are greater than the length of `cypher`). These extra numbers should be ignored. For example,

```
(decode "OK"
  (list 0 1 2 3 4))  
⇒ "OK".
```