This tutorial will cover the following main ideas:

- Writing data definitions and templates for structures.
- Solving problems involving structures and lists of structures.
- Accumulative recursion.
**Question 1: Summarizing Assignments**

In CS 135, students complete a new Assignment almost every week. An Assignment has the following properties:

- **id**: a symbol assignment identifier, such as ‘A01.
- **total**: a natural number indicating the maximum number of marks that can be earned on the assignment, such as 100.
- **weight**: the percentage of the final grade (must be above zero) that the assignment is worth, such as 5%.
- **grades**: the list of the marks achieved by all students on this assignment. For example, (list 65 80 73 100) gives the marks for four students. At least one student completes every assignment, so this list is never empty.

**Part A**: Write a structure definition for assignment and an accompanying data definition for the type Assignment. The fields in your structure definition must match the names and order specified above.
Part B: Create a template for processing an assignment.
Part C: The CS 135 instructional team wants to understand whether students have been successful on recent assignments. Write a function (summarize assign) which consumes an Assignment and produces a list with three elements (in this order):
   1. The minimum grade achieved by any student on the assignment.
   2. The average grade achieved by all students.
   3. The maximum grade achieved by any student.

;; Example:
(check-expect (summarize (make-assignment 'A01 100 5
  (list 20 30 50 40 10)))
  (list 10 30 50))
Question 2: Checking Assignment Validity
Now that we have defined an Assignment, we can work with multiple assignments, using the following data definition:
;; An AssignmentList is a (listof Assignment).

Part A: Write a template for processing an AssignmentList.
Part B: The instructors of CS 135 want to make sure all their assignments are always valid (meaning that they make sense and contain no errors). An AssignmentList is only valid if all of the following conditions hold:

- No student has an invalid mark (a mark less than 0 or more than total) on any assignment.
- Every Assignment id in the list is unique.
- The sum of the weights for all assignments is not greater than 100.

Write a function (valid? assign-list) to perform this check.

;; Examples:
(define A01 (make-assignment 'A01 100 5
                             (list 20 30 50 40 10)))
(define A02 (make-assignment 'A02 100 6 (list -4))))
(check-expect (valid? (list A01)) true)
(check-expect (valid? (list A01 A02) false)
Question 3: Increasing Assignment Grades
The CS 135 instructors have decided to give everyone bonus marks!

Write a function called \((\text{bonus-marks assign-list inc})\) which adds \(\text{inc}\) bonus marks (always a positive number) to every grade in each assignment in \(\text{assign-list}\). The function should produce a new AssignmentList with the updated grades. However, marks are still capped at 100% - if a mark would go over the assignment’s total, set the mark to total instead.

;; Example:
(define A06 (make-assignment 'A06 100 15 (list 60 70 80 90)))
(define A07 (make-assignment 'A07 100 15 (list 90 80 75 99)))
(check-expect (bonus-marks (list A06 A07) 5)
  (list (make-assignment 'A06 100 15
                          (list 65 75 85 95))
        (make-assignment 'A07 100 15
                          (list 95 85 80 100))))