Below are several exercises that we will be covering in the upcoming tutorial (Friday, Feb. 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: Summarizing Assignments**

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

- **id** (symbol): the assignment identifier, such as ‘A01.
- **max-marks** (natural number): the maximum number of marks that can be earned on the assignment, such as 100.
- **weight** (number greater than 0): the percentage of the final grade that the assignment is worth, such as 5%.
- **grades** (list of numbers): 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:** Create a data definition for an Assignment, implemented as a struct.

**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:

- The first element in the list is the minimum grade achieved by any student on the assignment.
- The second element in the list is the average grade achieved by any student.
- The final element in the list is the maximum grade achieved by students.

*Examples:*
Question 2: Checking Assignment Validity

Now that we have defined an Assignment, we can work with lists of 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 Assignment List is valid if the following conditions hold:

- No student has an invalid mark (a mark less than 0 or more than max-marks) 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) which consumes an AssignmentList assign-list. The function should produce true if all three of the above conditions hold and false otherwise.

Examples:

(valid? (list (make-assignment ‘A01 100 5 (list 20 30 50 40 10)))) → true
(valid? (list (make-assignment ‘A01 100 5 (list 300)))
  (make-assignment ‘A02 100 6 (list –4)))) → false

Question 3: Maximum Tree Depth

In Module 11, you learned how to represent trees using structs. Recall that the following definitions were provided on Slide 10:

(define-struct node (key left right))

`; A Node is a (make-node Nat BT BT)`

`; A binary tree (BT) is one of:`

`; * empty

`; * Node`
Using these definitions, write a function \((\text{max-depth } \text{btree})\) which consumes a binary tree \(\text{btree}\) and finds the maximum depth (the length of the longest path from the root to any leaf).

\textit{Examples:}

\[(\text{max-depth } (\text{make-node } 1 \text{ empty empty})) \rightarrow 0\]

\[(\text{max-depth } (\text{make-node } 1 (\text{make-node } 2 (\text{make-node } 3 \text{ empty empty}) \text{ empty}) \text{ empty}) \text{ empty})) \rightarrow 2\]