Below are several exercises that we will be covering in the upcoming tutorial (Friday, Feb. 11). 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: Gold, Silver, and Bronze**

A group of racers are competing in a foot race. A motion-sensing camera has recorded the finish times for each racer. You want to determine which racers should receive gold, silver, and bronze medals for winning the race. Given a list of numbers (for each racer, the number of seconds taken to complete the race), write a function *(top-three-racers finish-times)* to produce a list with the following three elements, in order.

- The time taken by the gold medalist (shortest time)
- The time taken by the silver medalist (second shortest time)
- The time taken by the bronze medalist (third shortest time)

Guarantee: There will always be at least three racers in every race and all finish times will be unique.

*Hint:* The finish times are given in arbitrary order. Would this problem be easier to solve if the finish times were ordered from lowest to highest?

*Examples:*

(top-three-racers (list 20 32.5 18 14 70)) \(\rightarrow\) (list 14 18 20)

(top-three-racers (list 10 30 20 40)) \(\rightarrow\) (list 10 20 30)

**Question 2: Decrypting Morse Code**

In morse code, letters are converted into a sequence of dots (“.”) and dashes (“-“). Unfortunately, you’ve received some morse code messages with all the separators removed, so it is impossible to
determine where each letter’s sequence begins and ends without some extra information. Thankfully, you also have a list of numbers to help you break up the sequence.

Write a function \textit{(decrypt-morse code code-lengths)} to decrypt morse code. The function consumes a string (code), which consists of dots “.” and dashes “-”, along with a list of natural numbers (code-lengths). Each number can range from 1 to 4, and the sum of all numbers in code-lengths is guaranteed to equal the length of the input code. Your goal is to decrypt the morse code string, given that each value in code-lengths indicates the number of code characters to use to produce the next character in the plain-text. The function should produce a single string, the plain-text version of the morse code sequence.

\textit{Hint:} You can use the provided association list \textit{morse-al} to map morse code sequences to characters. You may find the built-in function \textit{substring} useful.

\textit{Examples:}

\begin{verbatim}
(decrypt-morse “.-.” (list 1 1 2)) \rightarrow “TE”
(decrypt-morse “.... -. -.-” (list 3 1 4 3 1 1)) \rightarrow “SECRET”
\end{verbatim}

\textbf{Question 3: Column Sum}

Write a function \textit{(sum-columns matrix)} which consumes a numeric matrix with \(N\) rows and \(M\) columns and sums each column in the matrix. The function produces a list of \(M\) numbers, where the \(j^{th}\) element in the list is equal to the sum of the elements in the \(i^{th}\) column.

\textit{Examples:}

\begin{verbatim}
(sum-columns (list (list 1 2 3)
                (list -1 1 0))) \rightarrow (list 0 3 3)
\end{verbatim}