Below are several exercises that we will be covering in the upcoming tutorial (Friday, Feb. 4). 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: Caesar Cipher**

Given a string text and a natural number shift, write a function \((\text{encrypt text shift})\) that produces a new text encrypted using the Caesar cipher, which shifts each letter a fixed number of positions down the alphabet. For this problem, all characters in the input text will be capital letters from the alphabet A-Z. To produce the output text, add the shift value to each character in the input text. If the resulting character is not in the range A-Z, use modular arithmetic to produce a valid capital letter.

*Hint #1:* You may find the functions char->integer, integer->char, and modulo useful.

*Hint #2:* It is often easier to work with a list of characters than a string. Consider writing encrypt as a wrapper function to do this conversion, with a separate helper function to solve the problem.

*Example:*

\((\text{encrypt “ABCD” 2}) \rightarrow \text{“CDEF”}\)

\((\text{encrypt “ZAP” 1}) \rightarrow \text{“ABQ”}\)

Note: the Caesar cipher is a well-known encryption method, but it is not secure and can be easily hacked. If you would like to learn more, consider taking CS 458: Computer Security and Privacy.

**Question 2: Hill Climbing**

Write a function \((\text{find-peak hill})\) that finds the index (position, starting from 0) of the “peak” in the given hill. The function consumes a hill, a list of integers with the following requirements: the first part of the list will be strictly increasing, and the rest will be strictly decreasing. Either of these parts may be empty
(i.e., the list could just be strictly increasing or decreasing), but the list will always contain at least one element. Here are some examples of valid hills:

- (list 1 2 3 5 4 3)
- (list 1 2 3)
- (list -1 0 -4 -5 -6 -7)

Note: The peak is the maximum value in the list, but you may not use the function max in your solution.

Examples:

find-peak (list 1 2 3 5 4 3)) → 3
find-peak (list 1 2 3)) → 2
find-peak (list -1 0 -4 -5 -6 -7)) → 1

Question 3: Number Splitter

In this question, you will create two data definitions and a template in Parts A-C. In Part D, you will use them to write a function which splits a natural number into a list of its digits (e.g., 123 → (list 1 2 3)). You may wish to review Module 7 Slides 1-8 before solving this exercise.

Part A. A Digit is a natural number from 0 to 9 (inclusive). Write a data definition for a Digit.

Part B. In Module 7, you saw a data definition that defines a Nat as either 0 OR some natural number plus one. We can alternatively define a Nat as either a Digit OR (Nat * 10) + Digit. For example, 196 is just 19 * 10 + 6, and 19 is just 1 * 10 + 6.

Write a data definition for a Nat following these alternate rules.

Part C. Write a template for processing a Nat as defined in Part B.

Part D. Write a function (split-nat n) which consumes a natural number n and produces an ordered list of the digits of this natural number. Use the template and data definitions from Parts A-C to create your solution. Note: You may not use the function reverse in your solution.

Examples:

(split-nat 1234) -> (list 1 2 3 4)
(split-nat 581) -> (list 5 8 1)