Students: please show your TA your answers to 1 and 2 when you are done with those. As always, you’ll get credit for the lab if you make reasonable progress towards completing it: just do as much of the lab as you can.

1. Elevators are notoriously capricious. That’s partly because they’re really tricky to program well. Consider a building with two elevators and \( N_{\text{FLOORS}} \) floors. (Define a global constant variable \( N_{\text{FLOORS}} \) to store the number of floors in the building.)

Write a function with the signature

\[
elevator\_should\_stop(\text{flr}, \text{button}, \text{e1\_from}, \text{e1\_to}, \text{e2\_from}, \text{e2\_to})
\]

The function returns -1 if the input is invalid, 1 if elevator 1 should stop at floor \( \text{flr} \), 2 if elevator 2 should stop at floor \( \text{flr} \), and 0 if neither elevator should stop at floor \( \text{flr} \). The function returns 3 if both elevators should stop at floor \( \text{flr} \) according to the rules below.

- \( \text{e1\_from} \): the floor from which elevator 1 is going
- \( \text{e1\_to} \): the floor to which elevator 1 is going
- \( \text{e2\_from} \): the floor from which elevator 2 is going
- \( \text{e2\_to} \): the floor to which elevator 2 is going
- \( \text{button} \): “up”, “down”, or None, depending on whether the up button is pressed, the down button is pressed, or no buttons are pressed on floor \( \text{flr} \)

If elevator 1 (resp. 2) is not moving, then the floor on which the elevator is located is the same as \( \text{e1\_from} \) and \( \text{e1\_to} \) (resp., \( \text{e2\_from} \) and \( \text{e2\_to} \)). An elevator should stop at a given floor if:

- that floor is the elevator’s destination, or
- the floor is “on the way,” and the appropriate elevator button has been pressed (i.e., up or down, depending on which way the elevator is going), and the other elevator is not currently in a non-moving state at that floor.

2. Devise a testing strategy and test this function.

Note that engineering elevators to behave optimally is quite hard (if you don’t believe this, go to the Sidney Smith Building at 100 St George Str. and try to get to the sixth floor and back).
3a. Write a function that returns the number of days in a month, given the number of the month (1…12) and the year.

   Reminder: According to the **Gregorian calendar**, which is the civil calendar in use today, years evenly divisible by 4 are leap years, with the exception of centurial years that are not evenly divisible by 400. Therefore, the years 1700, 1800, 1900 and 2100 are not leap years, but 1600, 2000, and 2400 are leap years. *(Source: the US Naval Observatory website)*

3b. Write a function with the signature `next_day(y, m, d)`, which prints the date that follows the date `y/m/d`.

3c. Write a function that prints out, in order, all the dates between `fY/fM/fD` and `tY/tM/tD`.

3d. Implement a function which returns the number of days between `fY/fM/fD` and `tY/tM/tD`.

3e. Make the function from 3d as efficient as possible.