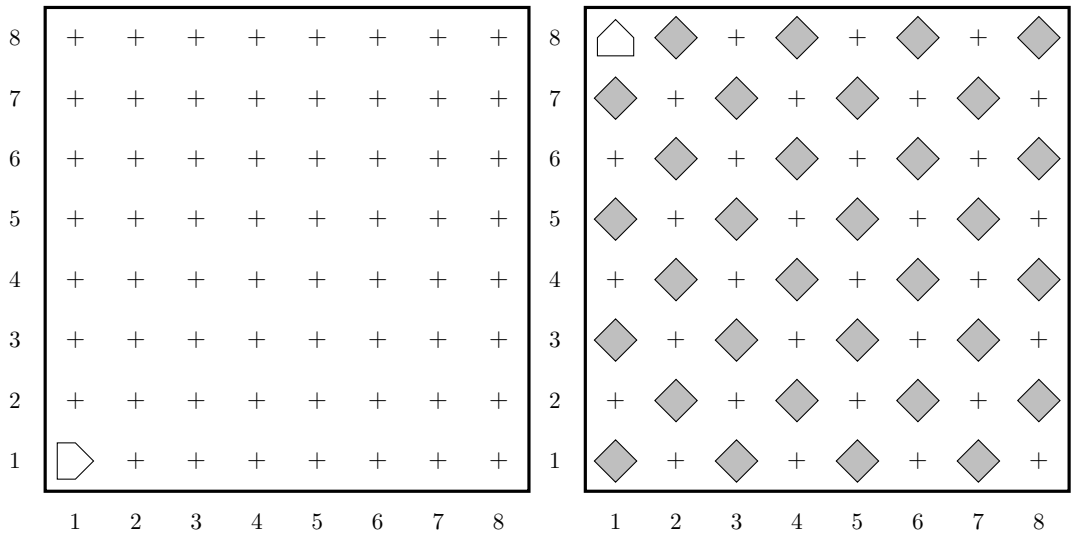
This problem set has one problem relating to Karel and then two problems concerning more generic Python programming. You should already be good to start working on the Karel problem, and the other problems you'll have most of the tools to solve after Monday. All problems have starting templates included in the repository.

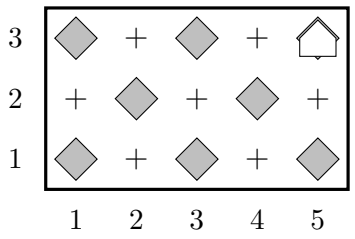
Get Assignment link: https://classroom.github.com/a/lIUbTb72

1. Karel, tired of painting, is thinking more abstractly, and wants to lay out a checkerboard pattern of beepers inside an empty rectangular world. An example of a before and after for an 8x8 world is shown below:



Before After

This problem has a nice decomposition structure along with some interesting algorithmic issues. As you think about how you will solve the problem, you should make sure that your solution works with checkerboards that are different in size from the standard 8x8 checkerboard shown in the example above. Odd-sized checkerboards are trickier, and you should make sure that your program generates the following pattern in a 5x3 world:



Another special case you need to consider is that of a world which is only one column wide or one row high. The repository includes several sample worlds for you to test your program against, including: `1x8.w`, `5x3.w`, `5x5.w`, and `8x1.w`. You can safely

assume that Karel always starts in the bottom left corner and facing to the east with an infinite supply of beepers in its bag. In does not matter where Karel finishes or which specific spaces have beepers, except that the world should be completely checkerboarded. To get full points on this problem, your program should show clear evidence of how you decomposed the problem and successfully recreate a checkerboard on any empty rectangular world.

2. Taken from Chapter 1, exercise 7.

> *It is a beautiful thing, the destruction of words.*
>
> —Syme in George Orwell's *1984*

In Orwell's novel 1984, Syme and his colleagues at the Ministry of Truth are engaged in simplifying English into a more regular language called *Newspeak*. As Orwell describes in his appendix entitled "The Principles of Newspeak," words can take a variety of prefixes to eliminate the need for the massive number of words we have in English. For example, Orwell writes,

> Any word—this again applied in principle to every word in the language—could be negatived by adding the affix *un-*, or could be strengthened by the affix *plus-*, or, for still greater emphasis, *doubleplus-*. Thus, for example, *uncold* meant "warm", while *pluscold* and *doublepluscold* meant, respectively, "very cold" and "superlatively cold."

Define three functions—`negate`, `intensify`, and `reinforce`—that take a string as input and add the prefixes `"un"`, `"plus"`, or `"double"` to that string, respectively, returning the result. For instance, if you were to enter in `negate("cold")` you should get out the string `"uncold"`. Below are several more examples of expected input and output, including chaining these functions together. These are all also printed in the given template so that you can check your code.

```
                    negate("cold") →  "uncold"
                 intensify("cold") →  "pluscold"
          reinforce(intensify("cold")) →  "doublepluscold"
reinforce(intensify(negate("good"))) →  "doubleplusungood"
```

3. Adapted from Chapter 2, exercise 3. I'm supplying you with a template file for this problem, but it is largely empty outside of start of some function definitions. So you otherwise have lots of flexibility in how you want to approach this.

> *Why is everything either at sixes or at sevens?*
>
> —Gilbert and Sullivan, *H.M.S. Pinafore*, 1878

   (a) Write a function called `print_divisible_by_six_or_seven` that displays the integers between 1 and 100 that are evenly divisible by *either* 6 or 7 *but not both*. Your function should print one number on each line, and nothing should be returned in the end.

   (b) Now revamp what you wrote above to construct a function that actually has inputs and outputs, called `list_divisible_by_six_or_seven`. This function should take two parameters as input, one indicating the lower bound and one indicating the upper bound of the range to check. It should return a *list* containing all the values within that range that are evenly divisible by 6 or 7, but not both. To create the list, you should start with an empty list and then use concatenation to add each new value (if matching the requirements) as you check them.