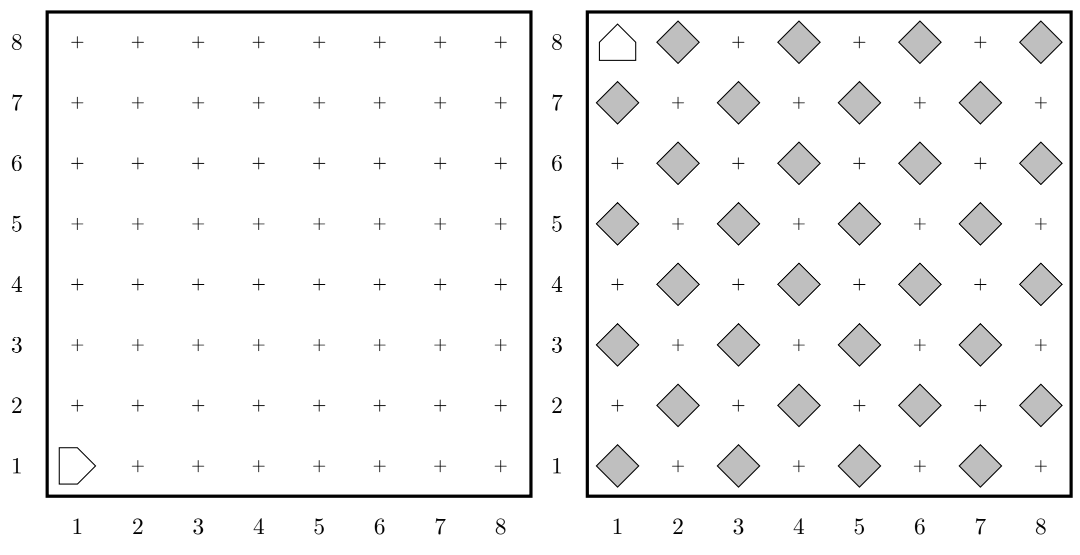
This problem set has one problem relating to Karel and then two problems concerning more generic Python programming. You should already be good to start working on the Karel problem, and the other problems you'll have most of the tools to solve after Wednesday. All problems have starting templates included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

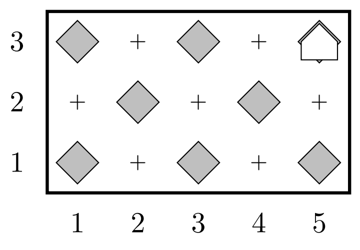
Get Assignment link: https://classroom.github.com/a/zZai6oXk

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 (but they should finish without erroring!) 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.

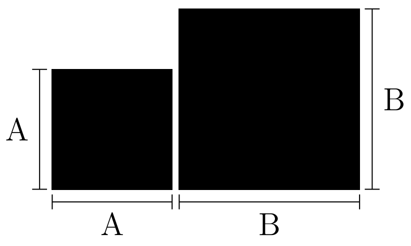
2. I'm supplying you with a template file for this problem, but it is largely empty. You thus have lots of flexibility in how you want to approach this, but make sure it achieves the objectives.

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

   Write a function called `divisible_by_six_or_seven` which takes two arguments as input. The first argument should represent the lower bound of a series of numbers to check, whereas the second argument should represent the upper bound. So for instance, calling `divisible_by_six_or_seven(1,100)` should check the numbers from 1 to 100 (including 1 and 100). Your function should print a number to the screen from the provided range if and only if it is evenly divisible by six or seven, *but not both.* These numbers do **not** need to be in strictly ascending order. In addition to printing these numbers, your function should keep track of how many numbers are printed, and *return* this value at the end of the function. So for instance, one example of running the program in the Python REPL might look like:

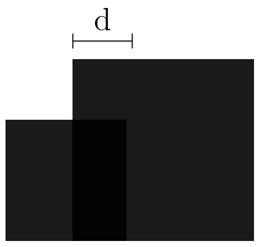
```
>>> count = divisible_by_six_or_seven(40, 60)
49
56
48
54
60
>>> print(count)
5
```

3. Consider the situation in which you have two perfect squares, one with a side length of A and the other with a side length of B, positioned as shown below.



Initially, there is no overlap and the total shaded area encompassed by both squares would be equal to $A^2 + B^2$.

Now suppose that the squares are moved towards one another such that they overlap a distance d, where d is 0 at a minimum and A at a maximum. (*See animated gif in the HW Images folder if you are having trouble picturing this over the full range.*) The total shaded area has clearly gone down.



Your objective in this problem is not to actually write a Python script, but to instead write out in plain English what your algorithm would be for computing the total area covered by the squares for any given values of A, B, and d, where you are promised that $A > 0$, $B > 0$ and $0 \leq d \leq A$.

In the provided template, write out using words and full sentences what your approach would be to find the total shaded area. No coding needs to be happening here, you just need to determine a method by which you can find the total covered area and convey that method clearly in English. Imagine that someone with no knowledge of this specific problem should be able to implement your described algorithm just from your description, and have it work for all possible values of A, B, and d.

Reminders and Tips:

- *You just need the shaded area, so be careful you don't "double count" the overlapping region.*
- *You don't know which square is larger, so you may need your algorithm to adjust depending on the inputs.*
- *It might be useful to map out the major possible configurations to ensure your algorithm works for all of them.*