NB: these exercises are more involved algorithm-design challenges. I recommend in each case starting with pen and paper and thinking about how to approach the problem with different inputs before starting to code.

Question 4: Palindromes

Complete the function `palindrome` in `hw02.py`. The function takes two arguments: an integer in string form, for example `'1921'`, and an integer `k`, for example 2. The function returns the largest possible *palindrome* integer that one can find by changing at most `k` digits of the input number to any other digit 0-9. A palindrome is the same number read backwards.

For example, with at most two changes to 1921, the largest possible palindrome number is 1991 (with one change 2 -> 9). If we had three changes, we would get from 1921 to 9999, the highest possible integer value with four digits.

Note that test cases could have thousands of digits.

You can test your function with

```
python ok -q palindrome -v
```

Note that **the test cases provided for the function are not comprehensive**. Part of the challenge of solving the problem is thinking about possible cases that may come up. Before you start coding, it may be worth writing down some (say, six- or eight-digit) numbers on a piece of paper and working out what the result of the algorithm should be if you have one change, two changes, three changes, and so on.

In both this question and the next one, it is possible to design a solution approach that will eventually find the correct solution but might take a very long time to do so, for example by exhaustively enumerating all possible solution alternatives. The goal of the exercises is to find a better approach that would work quickly for even large problem sizes. Some of the test cases used to test your function will be much longer than those in the function (say, hundreds or thousands of digits). If the autograder times out, which will happen in approximately 20 seconds, your program is considered to have failed the test.

Question 5: Reverse engineering

In the first homework, you completed an exercise to calculate a value based on whether the input was divisible by 3, 5, or both. If we looped that function through integers from one, the first return values would be:

```
None # For input 1
None # For input 2
'3'
None
'5'
'3'
None
None
'3'
'5'
```

In this exercise, we'll try to reverse engineer the output of a similar process.

The sequence of numbers is generated as follows. Instead of checking division by 3 and 5, we have designated, say, three numbers 2, 4, 5. Whenever the input is divisible by 2, our output will include the letter `a`, when by 4, the letter `b`, when by 5, the letter `c`. Otherwise there is no output. So the generated sequence from one onwards would be

```
None # for input 1
'a' # for input 2
None # for input 3
'ab' # for input 4
'c'
'a'
None
'ab'
None
'ac' # for input 10
```

To make things interesting, we will not include the None values, giving us the sequence

```
'a'
'ab'
'c'
'a'
'ab'
'ac'
```

This is the input for the exercise. Your task is to "reverse engineer" `a, b, c` from the sequence. That is, you'll infer what integer values each letter corresponds to. There are multiple possible solutions: your solution should pick the lowest possible values.

Here's another example:

```
'a'
'b'
'a'
'a'
'b'
'a'
```

The solution is `a = 3, b = 5`. (You may confirm these are the lowest possible values that match the sequence.)

Complete the function `reverse_engineer` in `hw02.py`. Note that the test cases may have up to twenty different letters to reverse engineer and up to four-digit integers as solutions.

All done!

Submit your work with the command

```
python ok --submit
```

Before you submit, we recommend you test each function with

```
python ok -q function_name
```

Please note that you should not import any libraries to solve any of the problems in this assignment.

Optional Exercises

Evaluating your trading strategy

The file `hw02_extra.py` includes functions to try out your moving-average strategy and compare with a benchmark strategy. The benchmark is "buy and hold", which means simply buying in the beginning of the horizon and never selling the stock. A good alternative strategy should be able to consistently outperform such simple strategies.

Inspect the functions in the file. Running the file will import your functions from `hw02.py`. When you do this, you may get an error message in Spyder:

```
In [1]: from hw02 import moving_average, cross_overs, make_trades
Out[1]: ModuleNotFoundError: No module named 'hw02'
```

This happens if Spyder's working directory is not set to the directory where your files are. You can change the working directory from the folder icon in the top right corner of the window.