Getting started

You can find the skeleton code for the assignment in the file `hw02.py`. Complete the functions in the file as per the instructions below. Please do not rename the file: it should be called `hw02.py` when you submit it. See the full submission instructions in the end of this document.

You can test your solutions with `python ok` commands as specified below. *Note that these tests are not comprehensive and we will test your code with different examples. This means that* ***passing all the tests before submitting does not guarantee a perfect grade****. Be careful that your code works not just for a specific test case but in general.*

Using OK

We use a program called OK for code feedback. You should have OK in the starter files downloaded for this assignment. To use OK to test a specified function, run the following command:

```
python ok -q function_name
```

By default, only tests that did not pass will show up. You can use the `-v` option to show all tests, including tests you have passed:

```
python ok -q function_name -v
```

First part: Trading based on technical analysis

Some stock traders vouch for so-called "technical analysis": using mathematics and statistics to predict stock-price movements from historical data.

Others say that this is like driving a car by only looking at the rear-view mirror.

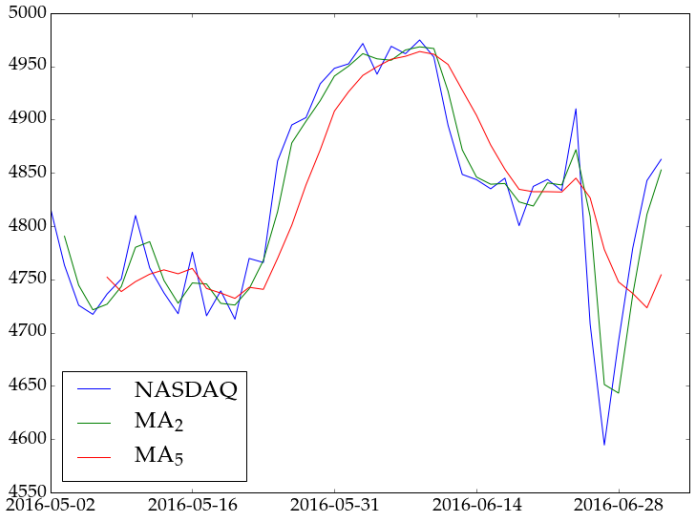
You will build a trading strategy based on technical analysis of stock prices and explore which group is right.

The technical analysis idea we'll study is using so-called moving-average strategies. A moving average is defined as follows. Consider a daily quoted time series of a stock price, p_t, eg the daily adjusted closing prices for the stock. An n-step (or n-lag) moving average is the average of the n last values of the price. More precisely, the n-step moving average is defined as follows

$$MA_n(t) = \frac{p_t + p_{t-1} + \ldots + p_{t-(n-1)}}{n}.$$

The one-step MA is simply the current price: $MA_1(t) = p_t$, and the two-step MA is the average of the current price and the previous day's price. Tf the moving average span n is short, the MA reacts quickly to any shocks to the stock price. If the span n is long, it reacts slowly. If the moving average spans the stock's entire period of existence, it is just the all-time average price.

The figure below shows an example of the Nasdaq index's development, along with two different moving averages. Notice how the two-step moving average follows the index more closely than the five-step one, which reacts to price changes more slowly and less strongly.



A theory put forward by proponents for technical analysis has it that moving averages give an indication of the "momentum" of a price. In particular, by comparing two moving averages of different lengths, one should be able to infer that the momentum of a the stock is changing, giving a signal when to buy or sell the stock.

This comparison between two moving averages is done by finding so-called *crossover points* between different moving averages. This works as follows:

- We keep track of two moving averages of different lengths: a shorter one (a lower lag n_L, eg $n_L = 2$ in the figure) and a longer one (higher lag n_H, eg $n_H = 5$ in the figure)
- **When the shorter MA "crosses" the longer one** (it has been lower but becomes higher), this is a **signal to buy the stock** as it seems to be gaining momentum. In the figure, these points are the ones where the green line has been lower than the red line but then crosses it to become higher, eg around 29 June. More specifically, these are defined as follows. If for any time t, $MA_{n_H}(t-1) \geq MA_{n_L}(t-1)$ and $MA_{n_L}(t) > MA_{n_H}(t)$, you should buy the stock.
- Conversely, when the longer MA "crosses" the shorter one in the same way, this is a signal to sell the stock as it seems to be losing momentum. In the figure, these are the points in time where the red line has been lower than the green one, but then becomes higher, eg around 10 June.

Inspecting the figure, it looks like those dates may indeed have been good times to trade. But in general, can you make money based on such strategies? Your task is to evaluate this claim by writing a Python script that calculates two moving averages of a (historical) stock/index price, finds all the crossover points, and finally performs trades based on those points.

Some advocates of technical analysis claim, for example, that by looking at the 20-day and 50-day moving averages of a stock price, one can identify longer trends and benefit from them. Let's create a trading strategy to this idea.

Question 1: Buy and hold (2p)

A standard benchmark strategy is buy and hold: you buy the stock in the beginning of the horizon and hold it until the end. Complete the function `buy_and_hold` in `hw02.py`. The function takes as input a list of prices, a starting index, and a starting cash position. It should return the value of the final position if the entire cash position is used to purchase stock at the starting index.

You can test your function with

```
python ok -q buy_and_hold -v
```

Question 2: Moving average (6p)

Complete the function `moving_average` in `hw02.py`. The function takes as input a list of prices and the step n, and return the n-step moving average as a list, calculated using the formula above. We cannot calculate the moving average for the first

$n-1$ steps so these values in the list should be set to `None`. For example, a three-step moving average can only be calculated for the third price, so there would be two `None` values in the beginning of the list.

Please do not use Python libraries such as `numpy` or `pandas` for calculating moving averages (but you may want to use the `sum` function on a list).

You can test your function with

```
python ok -q moving_average -v
```

Question 3: Comparing MAs for crossovers (6p)

Complete the function `compare_mas` in `hw02.py`. The function takes as input two lists `ma1` and `ma2` (of moving averages of prices) and returns a list of indicator numbers by comparing the elements pairwise. Each indicator in the returned list takes either the value 1 if the value in `ma1` is strictly greater or 0 if not. A crossover occurs when the indicator changes value. The moving averages may contain `None`-values in the beginning. If either value is `None`, the indicator is `None`.

You can test your function with

```
python ok -q compare_mas -v
```

Question 4: Trading based on moving averages (6p)

Complete the function `ma_strategy` in `hw02.py`. The function takes as input an initial cash position (a float), a list of prices, and a list of comparison indicators as described above. It uses these data to make trades at crossover points of the moving averages as follows.

- Start out with initial cash position. Look for the first crossover with an indication to buy stock.
- If the indicator variable at the previous index was 0 and becomes 1, this is an indication of a moving average crossover, to buy stock. `None` values count as neither 0 nor 1.
- If the indicator variable at the previous index was 1 and becomes 0, this is the opposite crossover and an indication to sell stock.
- When you buy stock, you buy with our entire cash position. You will then hold stock only, and zero cash. You may assume that whenever a crossover happens, you're able to make a trade at the current day's price, without waiting for the next day.
- If you hold stock, the value of your position fluctuates with the value of the stock price. You look for the next crossover which gives an indication to sell stock.
- If there is an indication to sell, you convert all stock to cash at the current stock price.
- As a result, you will always either hold cash or hold stock, but never both. If you hold cash, the value of your position stays constant.
- Assume you can buy fractional amounts of stocks, and there are no trading fees.

You can test your function with

```
python ok -q ma_strategy -v
```

Second part: Problem-solving practice

Question 5: Gallery (6p)

A friend is running an art gallery and is interested in collecting information about how visitors explore it. They are able to collect data such that each visitor's entry and exit from each room in the gallery is recorded. Your task is to produce summary statistics from the resulting data.

The data come in a list of tuples, such that each tuple contains the following information in string format: the unique number of the room, the unique number of the visitor, and the time of entry/exit. For example, this could be `('15', '3', '61')` where

visitor 3 entered or exited room 15 at time 61. The list containing the tuples is not sorted. Each visitor initially starts from outside any room, and finishes outside any room. This means that the first time in chronological order that we have a tuple with a visitor and a room, the visitor enters. The next time, they leave. A visitor may visit the same room multiple times. Rooms are integer numbered between 0 and 50. Visitors are integer numbered between 0 and up to 1000. Times are integer numbered between 0 and 1000. Some rooms may have no visitors. All visitors visit at least one room.

Complete the function `gallery` in `hw02.py`. Your function should produce the following summary statistics for each room in the data:

- The number of unique visitors the room.
- The average visit time of unique visits the room.
- The highest total time spent in the room by a single visitor.

The format in which the function should return this information is specified in the function docstring. Please note that the function has an `option` argument such that it may return one, two, or three of the above statistics. You may consider first implementing the first functionality (`option=0`) before moving on. The three option values will be roughly equally weighed in grading.

You can test your function with

```
python ok -q gallery -v
```

Question 6: Reverse engineering (6p)

This exercise is a more involved algorithm-design challenge. I recommend starting with pen and paper and thinking about how to approach the problem with different inputs before starting to code.

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`. Test cases may have up to twenty different letters to reverse engineer and up to four-digit integers as solutions. You will need to make sure your algorithm is efficient: in order to pass test cases on the autograder, it should solve them in under 10 seconds on a typical laptop.

You can test your function with

```
python ok -q reverse_engineer -v
```

All done!

To submit the assignment, follow these instructions:

1. Test all your functions using the above `ok` commands. Make sure to remove extra print statements etc that you may have used to test your code from `hw02.py`.
2. Run the command `python ok` on the command line. This is to check that your assignment file loads correctly in `ok`. If there is an error, fix the error in `hw02.py` before submitting.
3. Create a zip file called `submission.zip` containing `hw02.py` and nothing else. Please make sure that the names of the files are exactly these.
4. Upload the zip file in the submission area on the Hub.

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, compare_mas, ma_strategy, buy_and_hold
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.

Running the function `plot_ma` should compare your strategy with "buy and hold".

How does it work? It uses the `pandas-datareader` library to directly download stock prices from the web, and extract daily price data. We will learn more about `pandas` later in the module.

It then performs trading using the two strategies specified.

We can try different strategies. Suppose we have a list of `prices`. For example, we might like to buy whenever the shorter-term MA (say 20 days) becomes higher than the longer-term MA (say 50 days). This would mean that we calculate the moving averages for both, then the crossovers:

```
ma20 = moving_average(prices, 20)
ma50 = moving_average(prices, 50)
cos = compare_mas(ma20, ma50)
```

and finally get the position values over time as

```
values = ma_strategy(starting_cash, prices, cos)
```

> **Note.** If the pandas datareader library is not installed on your machine, you can get it by opening the command line on your machine and typing `conda install pandas-datareader`

Challenge: winning strategies?

Are moving-average strategies any good? Explore this question:

1. Pick a stock or index (eg Apple) and a period of time, eg the years 2000 -- 2010. This is your *training* data set.
2. Using your training data, go through all possible moving average strategies (n_H, n_L) for the stock, for example up to $n = 100$. Store the ones that perform best in terms of return to your investment.
3. Test the performance of these best strategies on the years following your training data set, eg 2011 - 2016. Do the strategies still make money in this time period? How would your analysis change if you had to pay trading fees of say 1%?
4. Repeat this for different stocks and time periods. If you find a strategy that consistently outperforms "buy and hold", I'd like to know - send me an email! Or keep it to yourself and start trading...
5. Finally, download cryptocurrency data using Pandas datareader and repeat the analysis. Does technical analysis look potentially more profitable for cryptocurrencies?