Coursework 6 (Scala)

This coursework is about Scala and is worth 10%. The first and second part are due on 16 November at 11pm, and the third part on 21 December at 11pm. You are asked to implement three programs about list processing and recursion. The third part is more advanced and might include material you have not yet seen in the first lecture. Make sure the files you submit can be processed by just calling `scala <<filename.scala>>`.

Important: Do not use any mutable data structures in your submissions! They are not needed. This means you cannot use `ListBuffers`, for example. Do not use `return` in your code! It has a different meaning in Scala, than in Java. Do not use `var`! This declares a mutable variable. ??? Make sure the functions you submit are defined on the "top-level" of Scala, not inside a class or object. Also note that the running time of each part will be restricted to a maximum of 360 seconds on my laptop.

Disclaimer

It should be understood that the work you submit represents your own effort. You have not copied from anyone else. An exception is the Scala code I showed during the lectures or uploaded to KEATS, which you can freely use.

Part 1 (3 Marks)

This part is about recursion. You are asked to implement a Scala program that tests examples of the $3n + 1$-*conjecture*, also called *Collatz conjecture*. This conjecture can be described as follows: Start with any positive number n greater than 0:

- If n is even, divide it by 2 to obtain $n/2$.
- If n is odd, multiply it by 3 and add 1 to obtain $3n + 1$.
- Repeat this process and you will always end up with 1.

For example if you start with 6, respectively 9, you obtain the series

$$6, 3, 10, 5, 16, 8, 4, 2, 1 \quad (= 9 \text{ steps})$$
$$9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 \quad (= 20 \text{ steps})$$

As you can see, the numbers go up and down like a roller-coaster, but curiously they seem to always terminate in 1. The conjecture is that this will *always* happen for every number greater than 0.[1]

[1] While it is relatively easy to test this conjecture with particular numbers, it is an interesting

Tasks (file collatz.scala):

(1) You are asked to implement a recursive function that calculates the number of steps needed until a series ends with 1. In case of starting with 6, it takes 9 steps and in case of starting with 9, it takes 20 (see above). In order to try out this function with large numbers, you should use `Long` as argument type, instead of `Int`. You can assume this function will be called with numbers between 1 and 1 million. [2 Marks]

(2) Write a second function that takes an upper bound as argument and calculates the steps for all numbers in the range from 1 up to this bound. It returns the maximum number of steps and the corresponding number that needs that many steps. More precisely it returns a pair where the first component is the number of steps and the second is the corresponding number. [1 Mark]

Test Data: Some test ranges are:

- 1 to 10 where 9 takes 20 steps
- 1 to 100 where 97 takes 119 steps,
- 1 to 1,000 where 871 takes 179 steps,
- 1 to 10,000 where 6,171 takes 262 steps,
- 1 to 100,000 where 77,031 takes 351 steps,
- 1 to 1 million where 837,799 takes 525 steps

Part 2 (4 Marks)

This part is about list processing—it's a variant of "buy-low-sell-high" in Scala. It uses the online financial data service from Yahoo.

Tasks (file trade.scala):

(1) Given a list of prices for a commodity, for example

```
List(28.0, 18.0, 20.0, 26.0, 24.0)
```

open problem to *prove* that the conjecture is true for *all* numbers (> 0). Paul Erdös, a famous mathematician you might have hard about, said about this conjecture: "Mathematics may not be ready for such problems." and also offered a $500 cash prize for its solution. Jeffrey Lagarias, another mathematician, claimed that based only on known information about this problem, "this is an extraordinarily difficult problem, completely out of reach of present day mathematics." There is also a xkcd cartoon about this conjecture (click here). If you are able to solve this conjecture, you will definitely get famous.

you need to write a function that returns a pair of indices for when to buy and when to sell this commodity. In the example above it should return the pair `(1, 3)` because at index 1 the price is lowest and then at index 3 the price is highest. Note the prices are given as lists of `Doubles`.

[1 Mark]

(2) Write a function that requests a comma-separated value (CSV) list from the Yahoo websevice that provides historical data for stock indices. For example if you query the URL

http://ichart.yahoo.com/table.csv?s=GOOG

where `GOOG` stands for Google's stock market symbol, then you will receive a CSV-list of the daily stock prices since Google was listed. You can also try this with other stock market symbols, for instance AAPL, MSFT, IBM, FB, YHOO, AMZN, BIDU and so on.

This function should return a List of strings, where each string is one line in this CVS-list (representing one day's data). Note that Yahoo generates its answer such that the newest data is at the front of this list, and the oldest data is at the end. [1 Mark]

(3) As you can see, the financial data from Yahoo is organised in 7 columns, for example

```
Date,Open,High,Low,Close,Volume,Adj Close
2016-11-04,750.659973,770.359985,750.560974,762.02002,2126900,762.02002
2016-11-03,767.25,769.950012,759.030029,762.130005,1914000,762.130005
2016-11-02,778.200012,781.650024,763.450012,768.700012,1872400,768.700012
2016-11-01,782.890015,789.48999,775.539978,783.609985,2404500,783.609985
....
```

Write a function that ignores the first line (the header) and then extracts from each line the date (first column) and the Adjusted Close price (last column). The Adjusted Close price should be converted into a `Double`. So the result of this function is a list of pairs where the first components are strings (the dates) and the second are doubles (the adjusted close prices).

[1 Mark]

(4) Write a function that takes a stock market symbol as argument (you can assume it is a valid one, like GOOG or AAPL). The function calculates the dates when you should have bought the corresponding shares (lowest price) and when you should have sold them (highest price). [1 Mark]

Test Data: In case of Google, the financial data records 3077 entries starting from 2004-08-19 until 2016-11-04 (which is the last entry on the day when I prepared the course work...namely on 6 November; remember stock markets are typically closed on weekends and no financial data is produced then; also I

did not count the header line). The lowest shareprice for Google was on 2004-09-03 with $49.95513 per share and the highest on 2016-10-24 with $813.109985 per share.

Advanced Part 3 (3 Marks)

A purely fictional character named Mr T. Drumb inherited in 1978 approximately 200 Million Dollar from his father. Mr Drumb prides himself to be a brilliant business man because nowadays it is estimated he is 3 Billion Dollar worth (one is not sure, of course, because Mr Drumb refuses to make his tax records public).

Since the question about Mr Drumb's business acumen remains, let's do a quick back-of-the-envelope calculation in Scala whether his claim has any merit. Let's suppose we are given $100 in 1978 and we follow a really dumb investment strategy, namely:

- We blindly choose a portfolio of stocks, say some Blue-Chip stocks or some Real Estate stocks.
- If some of the stocks in our portfolio are traded in January of a year, we invest our money in equal amounts in each of these stocks. For example if we have $100 and there are four stocks that are traded in our portfolio, we buy $25 worth of stocks from each.
- Next year in January, we look how our stocks did, liquidate everything, and re-invest our (hopefully) increased money in again the stocks from our portfolio (there might be more stocks available, if companies from our portfolio got listed in that year, or less if some companies went bust or de-listed).
- We do this for 38 years until January 2016 and check what would have become out of our $100.

Tasks (file drumb.scala):

(1.a) Write a function that queries the Yahoo financial data service and obtains the first trade (adjusted close price) of a stock symbol and a year. A problem is that normally a stock exchange is not open on 1st of January, but depending on the day of the week on a later day (maybe 3rd or 4th). The easiest way to solve this problem is to obtain the whole January data for a stock symbol as CSV-list and then select the earliest entry in this list. For this you can specify a date range with the Yahoo service. For example if you want to obtain all January data for Google in 2000, you can form the query:

```
http://ichart.yahoo.com/table.csv?s=GOOG&a=0&b=1&c=2000&d=1&e=1&f=2000
```

For other companies and years, you need to change the stock symbol (`GOOG`) and the year `2000` (in the `c` and `f` argument of the query). Such a request might fail, if the company does not exist during this period. For example, if you query for Google in January of 1980, then clearly Google did not exists yet. Therefore you are asked to return a trade price as `Option[Double]`.

(1.b) Write a function that takes a portfolio (a list of stock symbols), a years range and gets all the first trading prices for each year. You should organise this as a list of lists of `Option[Double]`'s. The inner lists are for all stock symbols from the portfolio and the outer list for the years. For example for Google and Apple in years 2010 (first line), 2011 (second line) and 2012 (third line) you obtain:

```
List(List(Some(313.062468), Some(27.847252)),
     List(Some(301.873641), Some(42.884065)),
     List(Some(332.373186), Some(53.509768)))
```

[1 Mark]

(2.a) Write a function that calculates the *change factor* (delta) for how a stock price has changed from one year to the next. This is only well-defined, if the corresponding company has been traded in both years. In this case you can calculate

$$\frac{price_{new} - price_{old}}{price_{old}}$$

(2.b) Write a function that calculates all change factors (deltas) for the prices we obtained under Task 1. For the running example of Google and Apple for the years 2010 to 2012 you should obtain 4 change factors:

```
List(List(Some(-0.03573991820699504), Some(0.5399747522663995))
        List(Some(0.10103414428290529), Some(0.24777742035415723)))
```

That means Google did a bit badly in 2010, while Apple did very well. Both did OK in 2011. [1 Mark]

(3.a) Write a function that calculates the "yield", or balance, for one year for our portfolio. This function takes the change factors, the starting balance and the year as arguments. If no company from our portfolio existed in that year, the balance is unchanged. Otherwise we invest in each existing company an equal amount of our balance. Using the change factors computed under Task 2, calculate the new balance. Say we had \$100 in 2010, we would have received in our running example

```
$50 * -0.03573991820699504 + $50 * 0.5399747522663995
                                                = $25.211741702970222
```

as profit for that year, and our new balance for 2011 is \$125 when converted to a `Long`.

(3.b) Write a function that calculates the overall balance for a range of years where each year the yearly profit is compounded to the new balances and then re-invested into our portfolio. [1 Mark]

Test Data: File `drumb.scala` contains two portfolios collected from the S&P 500, one for blue-chip companies, including Facebook, Amazon and Baidu; and another for listed real-estate companies, whose names I have never heard of. Following the dumb investment strategy from 1978 until 2016 would have turned a starting balance of \$100 into \$23,794 for real estate and a whopping \$524,609 for blue chips.

Moral: Reflecting on our assumptions, we are over-estimating our yield in many ways: first, who can know in 1978 about what will turn out to be a blue chip company. Also, since the portfolios are chosen from the current S&P 500, they do not include the myriad of companies that went bust or were de-listed over the years. So where does this leave our fictional character Mr T. Drumb? Well, given his inheritance, a really dumb investment strategy would have done equally well, if not much better.