

# Programming Assignment 1

Deadline: Friday March 5th 2021 at 23:59

Submit to elearn a tarball with name *yourFirstName.yourLastName.tar*. The tarball should contain:

1. The source code of your solutions
2. Your report with answers to assignment questions that mention “Write in your report”. Do not forget to include at the beginning of your report your personal details (name, ID, etc).

## 1 Scala Warming up (30 points)

In this section we are going to write simple scala functions to get used to the language. **No mutable variables** (i.e. “var” variables) can be used in any question. For questions that do not explicitly mention the use of tail recursive functions and your solution uses recursion, feel free *NOT* to enforce *tail* recursion. For this section, you are expected to create object `ScalaWarmUp` with member methods the ones that you are going to develop in response to the questions of this section. The object may contain any other method you create for assistance. Name your file `ScalaWarmUp.scala`.

1. Write a function that returns the second to last element of a list of integers. The function must be tail recursive. *Hint!* Use list pattern matching. The signature of the function should be:

```
def getSecondToLast(lst: List[Int]): Int
```

You can test the function for correctness by comparing it's output with:

```
getSecondToLast(lst) == lst(lst.size - 2)
```

2. Write the same function but now use *zipWithIndex* and *filter* to obtain the same result. The signature of this function should be:

```
def getSecondToLastZip(lst: List[Int]): Int
```

3. Eliminate consecutive duplicates of a list elements. If a list contains repeated elements they should be replaced with a single copy of the element. The element order should not be changed. The signature of this function should be:

```
def filterUnique(l: List[String]): List[String]
```

Example outputs:

```
filterUnique(List("c", "c", "d", "e", "e", "a", "a")) = List("c", "d",  
"e", "a")  
filterUnique(List("a", "c", "b", "b", "a", "c")) = List("a", "c", "b",  
"a", "c")
```

4. Find the most frequent substring of size *k* on a list of strings. Assume all strings have length greater than *k*. The signature of this function should be:

```
def getMostFrequentSubstring(lst: List[String], k: Int)
```

Example output:

```
getMostFrequentSubstring(List("abcd", "xwyzfs", "klmbco"), 2) = "bc"
```

Tip: A string of length  $n$  contains  $n - k + 1$  substrings of length  $k$ .

5. **Bonus 5pts.** Given a list of strings, write a function that returns true if its parentheses are balanced. For simplicity assume that a list element is either “(”, “)”, or an alphanumeric string. You are expected to use the `reduceLeft` function of `scala`. The signature of this function should be:

```
def checkBalancedParentheses(lst: List[String]): Boolean
```

Example output:

```
checkBalancedParenthesis(List("(", "(", "hello", "543", ")", " ")) = true
checkBalancedParenthesis(List(")", ")", "(", "(")) = false
```

## 2 Fun with Apache Spark (70 points)

In this exercise we will use Apache Spark to analyze a (small) web server log. We are going to use a web server log from NASA.

The dataset of this exercise is rather small, so you will have no problems answering all questions from Spark local installations, either on your personal laptops/desktops or on CSD-provided infrastructure.

Of course, you are highly encouraged to deploy your solutions to a cloud setup. In that case, start developing locally and experiment with a small subset of the data; You may move to a cluster as soon as you are confident that what you have is working.

**ATTENTION!!!** Do not forget to destroy an AWS cluster that you do not need... It costs unnecessary \$\$\$ when idle.

### 2.1 Environment setup

If you use a local Spark installation in your desktops/laptops, Spark can easily access files from your local file system.

You can also have the choice to use the EC2 service of AWS to create a Spark cluster with 2 nodes (1 master, 1 slave).

From a terminal, you can download the dataset of this assignment as follows:

```
curl -O ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip *gz
```

On Amazon, you can access the dataset and upload it to S3. To upload a local file (say `nasa.txt`) to a bucket on S3 (say bucket `s3://cs543YourName`) type:

```
aws s3 cp NASA_access_log_Jul95 s3://cs543YourName/nasa.txt
```

Start a spark-shell and import and test the required libraries. In this exercise we are going to need the regular expression library of Scala.

```
import scala.util.matching.Regex

val pattern = "([0-9]+) ([A-Za-z]+)".r
pattern.findAllIn("100 bananas 200 mangoes").toList
```

### 2.2 Data Exploration

To load the file from a Spark shell: First make sure that `AWS_ACCESS_KEY_ID`, and `AWS_SECRET_ACCESS_KEY` are set in your environment and then open a spark shell:

```
val baseRdd = sc.textFile("s3a://cs543YourName/nasa.txt")
```

This is a text file of *unstructured* data. In order to give some structure, observe that it follows the Common Log Format. Table 1 shows the field description.

First you need to create a case class to convert the unstructured lines you just loaded into a list of objects:

```
case class Log (host: String, date: String, requestURI: String, status: Int, bytes: Int)
```

field	meaning
remotehost	Remote hostname
rfc931	Don't worry about that
authuser	The user name of the remote user, as authenticated by the HTTP server
date	The date and Time of the request
requestURI	The request, exactly as it came from the client
status	The HTTP status code the server sent back to the client
bytes	The number of bytes (content-length) transferred to the client.

Table 1: Field description of the Common Log Format

The supplementary files of this homework provide function `getLogFields` that returns a `Log` object from a string of the Common Log Format.

Transform `baseRdd` from `RDD[String]` to `RDD[Log]` and name the output as “`splitRdd`”.

```
val splitRdd = ?
```

Does this work? Use “count” to trigger a spark action. What are all these exceptions? `NoSuchElementException`? No good! So, your next task is to locate which regular expression fails. For each field of the Common Log Format that we are trying to extract, use a combination of transformations + count of the `baseRdd` to “zoom in” the problematic column.

If you debug this situation properly you will notice there is a single problematic column. Let's try to figure out what is wrong with that. But first go back to the regular expression that was used to produce that column and figure out what kind of strings that expression is looking for.

Now that we understand the regular expression, there are two things that might have gone wrong: Is it possible there are lines without a valid content size? Or is there something wrong with the regular expression that we used? Let's take a look at a few lines for which the regular expression fails.

We are going to use `findFirstIn` function of regular expressions to look for some bad lines. In the code snippet below, *patternX* is the pattern of `getLogFields` for the problematic column (you should replace it with the correct one).

```
val badRdd = baseRdd.filter(x => patternX.findFirstIn(x) == None)
badRdd.count
badRdd.take(3)
```

Whoops all lines are 404 errors. What is the value of the problematic field for those errors?

Let's fix those errors and try again to produce a `RDD[Log]`. We do not want to discard those entries, but we will replace the values in problematic columns with 0.

Rewrite function `getLogFields` so that it assigns 0 to each field where *patternX* fails.

Reapply this function to `baseRdd` to transform it to `cleanRdd` and run `count` to materialize it. We are going to use this a lot, so cache it in memory.

**To summarize:** For this subsection, you are expected to:

1. Write in your report which field of Table 1 produces exceptions.
2. Write in your report, what kind of patterns that field is expected to contain, and what happens in the problematic lines.
3. Implement a modified version of function `getLogFields` that has been provided to you, so that it replaces with 0 the problematic fields. The signature of the function should be:

```
def getImprovedLogFields(str: String): Log
```

4. Implement a function that takes an `RDD` of strings and converts it to `RDD` of `Log` using the improved function that you wrote in the step above. Then use this function to produce `cleanRDD` from `baseRDD`. Materialize `cleanRDD` and cache it; you will need to access it multiple times in the rest of this exercise. The function signature should be:

```
def convertToLog(base: RDD[String]): RDD[Log]
```

## 2.3 Walk through on the Web Server Log File

For each of the questions below implement a scala function that takes as input an RDD of Log and prints the requested values. You must also include all of those results in your report. Some of the RDDs you are about to create below might need caching as they can be used to more than one questions.

1. Explore content size: Write in your report the min, max, and average content size.
2. HTTP status analysis: Write in your report the 100 most frequent status values and their frequencies.
3. Frequent hosts: Write in your report 10 hosts that accessed the server more than 10 times.
4. Top-10 error paths: Write in your report the top 10 requestURIs that did not have a return code of 200.
5. Unique hosts: Write in your report how many unique there are in the entire log.
6. Write in your report the count of 404 Response codes.
7. Write in your report 40 distinct requestURIs that generate 404 errors.
8. Write in your report a list of the top twenty paths (in sorted order) that generate the most 404 errors.