Practice Programming Assignment: Scaffolding material

You have not submitted. You must earn 8/10 points to pass.

Deadline Pass this assignment by January 14, 11:59 PM PST

Note: If you have paid for the Certificate, please make sure you are submitting to the required assessment and not the optional assessment. If you mistakenly use the token from the wrong assignment, your grades will not appear

Download the assignment and the dataset and extract them somewhere on your file system. The assignment archive contains an sbt project starter, while the dataset only contains the data you are going to use.

Note that the sbt project requires Java 8. Be sure that you have this version of the JVM installed on your environment.

First, copy the content of the dataset (the "resources" directory) into the directory **observatory/src/main/**, so that the **.csv** files are located under the **observatory/src/main/resources/** directory.

While the whole capstone project is graded via a single assignment, we divided the whole project into 6 milestones (one per week). Concretely, this means

How to submit

Copy the token below and run the submission script included in the assignment download. When prompted, use your email address **serg.astapov@gmail.com**.

9GEWQZUjjE6UXXMw

Generate new token

Your submission token is unique to you and

that you will hack on the same code base from the beginning to the end, and that the starter project already contains some files that are going to be used by later milestones. Furthermore, you will have to configure which milestones should be graded. You can achieve that by changing the **Grading.milestone** value to the milestone / week number that you completed. This value is initially set to 1. After you completed the 1st milestone you should set it to 2, and so on. In the grader output, you can easily identify the tests which are relevant to a given milestone: tests names are prefixed by the number of the milestone and its name (e.g. "#2 - Raw data display").

To grade your work, run the following sbt command (after sbt is launched and shows its prompt):

```
> submit <your-email> <your-token>
```

Where, **<your-email>** is the email associated with your Coursera's account, and **<your-token>** is the token shown in the "How to submit" section on the top of this page.

The grader uses a JVM with limited memory: only 1.5 GB are available. It means that even if your code runs on your machine, it might fail on the grader. It is your job to design a program that fits in the available memory at run-time.

Our goal in this project is to give you as much freedom as possible in the solution space. Unfortunately, in order to be able to grade your work, we will ask you to implement some methods, for which we have fixed the type signature, and which may influence your solution. For instance, most of them are using Scala's standard **Iterable** datatype, but not all concrete implementations of **Iterable** may scale to the high volume of data required by the project (the grader is not going to use a high volume of data, though). So, while you must not change the code that is provided with the project, your actual implementation should use appropriate and efficient data types in order to perform incremental and

should not be shared with anyone. You may submit as many times as you like.

parallel computations. For this purpose, we have added several dependencies to the build: Spark, Akka Streams, Monix and fs2. You can use any of these libraries, if you want to, or just use the standard Scala library. However, note that the provided build just makes these libraries available in the classpath, but it does not configure their execution environment.

If you decide to use one of the above libraries, you might find it easier to implement the actual functionality in separate methods with different signatures. You can just consider the provided methods as "probes" into your code that the grader can use. For instance, if you want to use Spark RDDs and we provide a **locateTemperatures** method that requires an **Iterable**, you could implement it in the following way:

```
1  // Provided method:
2  def locationYearlyAverageRecords(
3    records: Iterable[(LocalDate, Location, Temperature)]
4  ): Iterable[(Location, Temperature)] = {
5    sparkAverageRecords(sc.parallelize(records)).collect().toSeq
6  }
7
8  // Added method:
9  def sparkAverageRecords(
10   records: RDD[(LocalDate, Location, Temperature)]
11 ): RDD[(Location, Temperature)] = {
12   ??? // actual work done here
13 }
14
```

Ultimately, you'll have to write your own **Main** to generate the map, so you have the freedom to organize your code the way you want. The **Main** doesn't have to use the provided methods, you can simply chain your own methods together instead. We want you to think about program structure, so don't feel too boxed in by the provided methods.

Last, note that there is a **src/test/** directory with test files for each milestone.

We recommend that you write tests here to check your solutions, but do not remove the existing code.

Before you start coding

Before you start coding, there are two files that you should know about. First, **package.scala** introduces type aliases that will be used throughout the project:

```
type Temperature = Double
type Year = Int
```

We will introduce them again when appropriate. Second, **models.scala** has a few useful case classes:

```
case class Location(lat: Double, lon: Double)
case class Tile(x: Int, y: Int, zoom: Int)
case class GridLocation(lat: Int, lon: Int)
case class CellPoint(x: Double, y: Double)

case class Color(red: Int, green: Int, blue: Int)
```

There is one case class for every coordinate system that we use, and an additional **Color** case class that may be useful for creating images. We will introduce each one of these again when relevant. **You are free to add methods to these case classes if you deem it appropriate, but you must not change their parameters.**

Notes for Spark users

The output of the grader is limited to 64 kB, but Spark produces a lot of logs by default. Often, this makes it impossible to read the whole grader feedback. Consequently, we suggest you to tune the log level of Spark like so:

```
import org.apache.log4j.{Level, Logger}
Logger.getLogger("org.apache.spark").setLevel(Level.WARN)
```

You may also want to refer to the Dataset vs DataFrame vs RDD discussion at the end of the last video of the Spark course so that you can choose the best API for this project.