Lab: Regular Expressions

In the current lab we are going to introduce some restrictions in the data for our database. Below you can see the constraints to validate the input entry before adding it to the data structure.

Input format – the format for the input should be the following:

{Course Name}_{Course Instance}{One or more white spaces}{Username}{One or more white spaces}{Score}

Our task now is to write a regular expression that matches only valid entries, so we can add them to our data structure safely. Here is some example input data that may be given:

C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87

Course name – starts with a capital letter and may contain only small and capital English letters, plus '+' or hashtag '#'.

Course instance – should be in the format 'Feb_2015', e.g. containing the first three letters of the month, starting with a capital letter, followed by an underscore and the year. The year should be between 2014 and the current year.

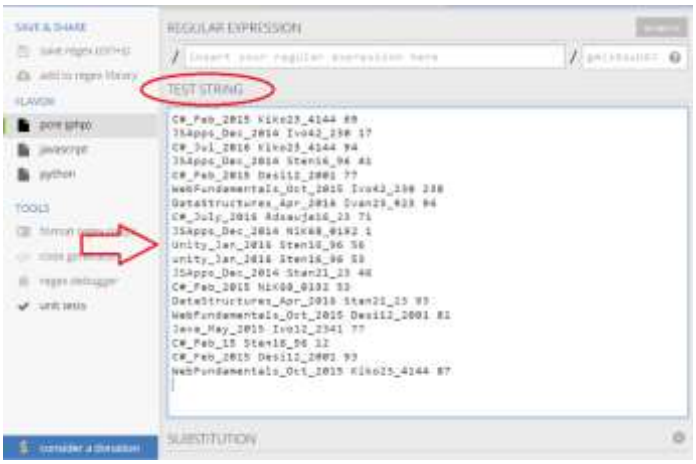
Username – starts with a capital letter and should be followed by at most 3 small letters after that. Then it should have 2 digits, followed by an underscore, followed by two to four digits. **Correct:** Ivan23_234, Nas12_4215, Re14_203. **Incorrect:** Ivana33_123, Stan_12, Мари31_421

Score – should be in the range from 0 to 100.

We are going to write a regular expressing for validating the input and implement it in the method for reading data from a file for the database of the university.

Problem 1. Start Using a Regex Editor

First we want to open some regex editor that will help us to complete our task. You can use whatever editor you like but you should be already familiar with https://regex101.com/ so we give you its link here. Next you may want to paste the sample data given above in the TEST STRING box:



Next you need to include the global modifier by simply adding a 'g' in the upper right corner:

Ok, that was pretty easy. Let's proceed with the next task.

Problem 2. Using Groups

We are going to have three groups, which are as follows:

1: Course name and instance

2: Student user name

3: Student score on task

First we will start with the first one and to be more specific with the course name. It should start with a capital letter so that is the first thing to add and you will be able to see as we go, how some data does not meet the conditions of the regex. So our regular expression by now should look like this:

And the matches are still quit unclear:

```
C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87
```

As you can see even from the first condition we don't catch the **unity** course written with small letters.

The next thing needed is that it **can** contain small and capital letters and also the symbols '+' and '#':

```
/([A-Z][a-zA-Z#+]*)/g
```

we have put an asterisk after the range, because the name of the course may be only of one capital letter. The result of the current modification looks like this:

```
C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87
```

Now it's time to add an underscore and the condition for the month name that follows, followed by an underscore:

```
/([A-Z][a-zA-Z#+]*_[A-Z][a-z]{2}_)/g
```

Here the condition after the range should be exactly two letters, because in the conditions the total number of letter for the month name is 3 and the first one should be capital. The result after this filter gives is now clearly showing where we are headed:

```
C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87
```

As you can see, now the C# course in July is no longer valid because the month is written with four letters.

Finally, it is time to add the year to the matching regular expression:

Now we should be ready with the group for the course and the result that we match by now should be as the one below:

```
C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87
```

As you can see, now we don't catch the C#_Feb_15, because the year is not in the valid format.

Now it is time to write the group for the user name which really similar to the one we just did.
So we must put a separator for one or more spaces, followed by the group, starting with a first capital letter:

The result after this filter is pretty much the same for the input we've chosen, but there could have been a person whose name starts with lower letter or an entry where there is no space between the course and the user name:

```
C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87
```

Now we should finish the regex for the rest of the name before the two numbers that follow. We should keep in mind that the we can have **at most** 3 letters after the first capital letter, but there may be 3 letters as there may be no letters after the first one. So this may be expressed as follows in the regular expression:


```
C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87
```

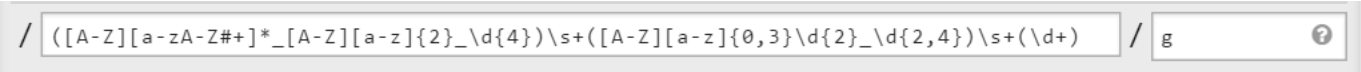
After this addition you can note that even if C#_July_2016 was written following the conditions, the user name would still be incorrect and of course the whole entry would be invalid.

So the only thing left for the username is the **two digits** that follow after the letters, followed by an **underscore**, followed by **two to four digits:**

You may see below that 2 more matches are now invalid, because they don't match the required format for the user name and more specifically for the numbers that are in the user name:

```
C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87
```

The final group we need to catch is the group for the score on a task for the given person and the given course:



The result with the given matches from the example should now look like this:

```
C#_Feb_2015 Kiko23_4144 69
JSApps_Dec_2014 Ivo42_230 17
C#_Jul_2016 Kiko23_4144 94
JSApps_Dec_2014 Sten16_96 41
C#_Feb_2015 Desi12_2001 77
WebFundamentals_Oct_2015 Ivo42_230 238
DataStructures_Apr_2016 Ivan23_923 94
C#_July_2016 Rdsauja16_23 71
JSApps_Dec_2014 NiK68_0192 1
Unity_Jan_2016 Sten16_96 56
unity_Jan_2016 Sten16_96 53
JSApps_Dec_2014 Stan21_23 46
C#_Feb_2015 NiK68_0192 53
DataStructures_Apr_2016 Stan21_23 93
WebFundamentals_Oct_2015 Desi12_2001 81
Java_May_2015 Ivo12_2341 77
C#_Feb_15 Sten16_96 12
C#_Feb_2015 Desi12_2001 93
WebFundamentals_Oct_2015 Kiko23_4144 87
```

So now that we've finally written the whole regular expression, it is time to implement it in C# in the method that reads all the students from the "database" file, but now you will be given a new file that will contain entries that do not match the given format.

We are going to refactor some code in the ReadData method in the StudentRepository class because we will now have to get the data from the groups of the current match.

First thing is to copy the pattern for the match of the entries and also create a `new Regex` with the given pattern:

```
string pattern = @"([A-Z][a-zA-Z#+]*_[A-Z][a-z]{2}_\d{4})\s+([A-Z][a-z]{0,3}\d{2}_\d{2,4})\s+(\d+)";
Regex rgx = new Regex(pattern);
string[] allInputLines = File.ReadAllLines(path);
```

Now that we have this instance of the Regex class, we can use it to check if there is a match with the current input line and if there is such, to get the data that we need from it in order to insert it in the data structure:

```
if (!string.IsNullOrEmpty(allInputLines[line]) && rgx.IsMatch(allInputLines[line]))
{
    Match currentMatch = rgx.Match(allInputLines[line]);
    string courseName = currentMatch.Groups[1].Value;
    string username = currentMatch.Groups[2].Value;
    int studentScoreOnTask;
    bool hasParsedScore = int.TryParse(currentMatch.Groups[3].Value, out studentScoreOnTask);
```

So the only thing left is to check if the score has been parsed and whether it is in the range between 0 and 100 and if all the three conditions are true, we can insert the data that we have extracted.

```
if (hasParsedScore && studentScoreOnTask >= 0 && studentScoreOnTask <= 100)
{
    if (!studentsByCourse.ContainsKey(courseName))
    {
        studentsByCourse.Add(courseName, new Dictionary<string, List<int>>());
    }

    if (!studentsByCourse[courseName].ContainsKey(username))
    {
        studentsByCourse[courseName].Add(username, new List<int>());
    }

    studentsByCourse[courseName][username].Add(studentScoreOnTask);
}
```

Finally, we should be ready with the **implementation** of the **regular expressions**.

Problem 3. Adding Features to the Command Interpreter

However **before testing** there is just one little thing we can **add to Command Interpreter**, because obviously we forgot adding it in the previous piece. The functionality is going to be a new command in the following format

show courseName (username) – user name may be omitted

What it does is show information for a given course or a given user name in a course from the data.

In the switch of the command interpreter, the case looks like the following:

```
case "show":
    TryShowWantedData(input, data);
    break;
```

Now it's time to add the show wanted data method and implement its functionality. We should only check for the number of parameters and depending on this, call the corresponding method.

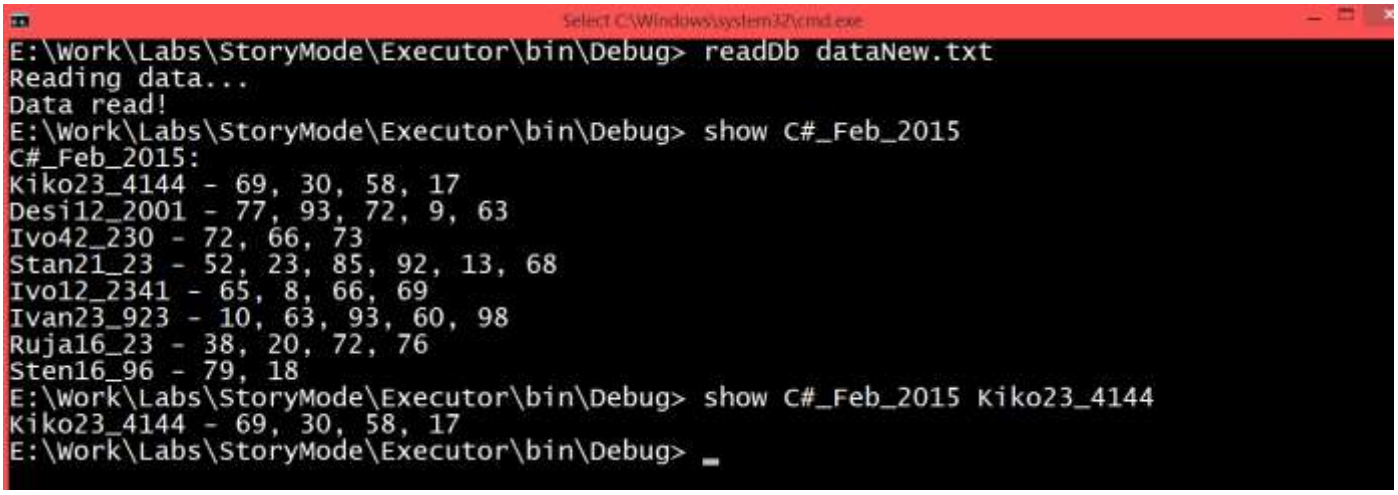
```
private static void TryShowWantedData(string input, string[] data)
{
    if (data.Length == 2)
    {
        string courseName = data[1];
        StudentsRepository.GetAllStudentsFromCourse(courseName);
    }
    else if (data.Length == 3)
    {
        string courseName = data[1];
        string userName = data[2];
        StudentsRepository.GetStudentScoresFromCourse(courseName, userName);
    }
    else
    {
        DisplayInvalidCommandMessage(input);
    }
}
```

Problem 4. Testing Your Code

Finally, we should be done with the corrections and now it's time for testing. Now the only thing we should call in the **main method** is **InputReader.StartReadingCommands();**

```
static void Main()
{
    InputReader.StartReadingCommands();
}
```

Since I've put the dataNew.txt in the Debug folder, I only read it, but you 've put it elsewhere, you'll have to navigate to the folder first.



You should be ready with the testing and next time you can expect to make the data we just read, filtered and ordered by some criteria we would like, using functional programming.

Congratulations! You've completed the lab exercise for Regular Expressions.