Lab: Regular Expressions (RegEx)

This document defines the homework assignments from the "Programming Fundamentals" Course @ Software University. Please submit your solutions (source code) of all below described problems in the Judge System.

1. Match Full Name

Write a C# Program to **match full names** from a list of names and **print** them on the console.

Writing the Regular Expression

First, write a regular expression to match a valid full name, according to these conditions:

- A valid full name has the following characteristics:
  - It consists of **two words**.
  - Each word **starts** with a **capital letter**.
  - After the first letter, it **only contains lowercase letters afterwards**.
  - **Each** of the **two words** should be **at least two letters long**.
  - The **two words** are **separated** by a **single space**.

Examples

Input
`Ivan Ivanov, Ivan ivanov, ivan Ivanov, IVan Ivanov, Test Testov, Ivan  Ivanov`
Output
`Ivan Ivanov Test Testov`

2. Match Phone Number

Write a regular expression to match a **valid phone number** from **Sofia**. After you find all **valid phones**, **print** them on the console, separated by a **comma and a space** "**,** ".

Compose the Regular Expression

A valid number has the following characteristics:

- It starts with "**+359**"
- Then, it is followed by the area code (always **2**)
- After that, it's followed by the **number** itself:
  - The number consists of **7 digits** (separated in **two groups** of **3** and **4 digits** respectively).
- The different **parts** are **separated** by **either a space or a hyphen** ('**-**').

You can use the following table of values to test your RegEx against:

Match ALL of these	Match NONE of these
`+359 2 222 2222` `+359-2-222-2222`	`359-2-222-2222, +359/2/222/2222, +359-2 222 2222` `+359 2-222-2222, +359-2-222-222, +359-2-222-22222`

Examples

Input
+359 2 222 2222,359-2-222-2222, +359/2/222/2222, +359-2 222 2222 +359 2-222-2222, +359-2-222-222, +359-2-222-22222 +359-2-222-2222
Output
+359 2 222 2222, +359-2-222-2222

3. Match Hexadecimal Numbers

Write a program, which finds all **valid hexadecimal numbers** in a **string** and **print** them **space-separated**.

Compose the Regular Expression

A valid hexadecimal number follows these conditions:

- Can have "**0x**" in front of it (not required)
- Has **one or more hexadecimal digits** after it (**0-9** and **A-F**).
- Doesn't have anything on **either** of its sides (use **\b**).

You can follow the table below to help with composing your RegEx:

Match ALL of these	Match NONE of these
0x10 0xAB 0x1F 10 AB 1F FF	0xG G 0x4G 4G 0xFG FG

Find all the **hexadecimal numbers** from the string and **print them** on the **console**, separated by **spaces**.

Examples

Input	Output
1F 0xG 0x1F G 0x4G 4G 0xAB 0xFG FG 0x10 10 AB FF	1F 0x1F 0xAB 0x10 10 AB FF

4. Match Dates

Write a program, which matches a date in the format "**dd{separator}MMM{separator}yyyy**". Use **named capturing groups** in your regular expression.

Compose the Regular Expression

Every valid date has the following characteristics:

- Always starts with **two digits**, followed by a **separator**
- After that, it has **one uppercase** and **two lowercase** letters (e.g. **Jan**, **Mar**).
- After that, it has a **separator** and **exactly 4 digits** (for the year).
- The separator could be either of three things: a period ("**.**"), a hyphen ("**-**") or a forward slash ("**/**")
- The separator needs to be **the same** for the whole date (e.g. 13**.**03**.**2016 is valid, 13**.**03**/**2016 is **NOT**). Use a **group backreference** to check for this.

You can follow the table below to help with composing your RegEx:

Match ALL of these	Match NONE of these
13/Jul/1928, 10-Nov-1934, 25.Dec.1937	01/Jan-1951, 23/sept/1973, 1/Feb/2016

Use **named capturing groups** for the **day**, **month** and **year**.

Examples

Input
13/Jul/1928, 10-Nov-1934, , 01/Jan-1951,f 25.Dec.1937 23/09/1973, 1/Feb/2016
Output
Day: 13, Month: Jul, Year: 1928 Day: 10, Month: Nov, Year: 1934 Day: 25, Month: Dec, Year: 1937

5. Match Numbers

Write a program, which finds all **integer** and **floating-point numbers** in a string.

Compose the Regular Expression

A number has the following characteristics:

- Has either **whitespace** before it or the **start** of the string (match either **^** or what's called a positive lookbehind). The entire syntax for the **beginning** of your **RegEx** might look something like "**(^|(?<=\s))**".
- The number might or might not be negative, so it might have a hyphen on its left side ("**-**").
- Consists of **one or more digits**.
- Might or might not have **digits after the decimal point**
- The decimal part (if it exists) consists of a period ("**.**") and **one or more digits** after it. Use a **capturing group**.
- Has either **whitespace** before it or the **end** of the string (match either **$** or what's called a positive lookahead). The syntax for the **end** of the **RegEx** might look something like "**($|(?=\s))**".

You can follow the table below to help with composing your RegEx:

Match ALL of these	Match NONE of these
1 -1 123 -123 123.456 -123.456	1s s2 s-s -1- _55_ s-2 s-3.5 s-1.1

Find all the **numbers** from the string and **print them** on the **console**, separated by **spaces**.

Examples

Input	Output
1 -1 1s 123 s-s -123 _55_ _f 123.456 -123.456 s-1.1 s2 -1- zs-2 s-3.5	1 -1 123 -123 123.456 -123.456

6. Replace <a> Tag

Write a program that replaces in a HTML document given as string **all the tags …** with corresponding **tags [URL href=…>…[/URL]**. Read an input, until you receive the **"end" command**. **Print** the lines on the **console**, but with the **<a>** tags replaced.

Examples

Input
        SoftUni   **end**
Output
        [URL href="**http://softuni.bg**"]SoftUni[/URL]