# Lab: Regular Expressions

Please, submit your source code solutions for the described problems to the [Judge System](https://judge.softuni.org/Contests/1742/Regular-Expressions-Lab).

## Match Full Name

Write a program to **match full names** from a sequence of characters 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**.
  + **Each** of the **two words** should be **at least two letters long**.
  + A **single space** separates the **two words**.

To help you out, we have outlined several steps:

1. Use the online regex tester [**regex101**](https://regex101.com/)
2. Check out how to use **character sets** (denoted with square brackets - "[]")
3. Specify that you want **two words** with a space between them (the **space character '** **'**, and **not** any whitespace symbol)
4. For each word, specify that it should begin with an uppercase letter using a **character set**. The desired characters are in a range – **from** '**A**' **to** '**Z**'.
5. For each word, specify that what follows the first letter are only **lowercase letters**, one or more – use another character set and the correct **quantifier**.
6. To prevent capturing letters across new lines, put "\b" at the beginning and the end of your regex. This will ensure that what precedes and what follows the match is a word boundary (like a new line).

To check your RegEx, use these values for reference (paste all of them in the **Test String** field):

|  |  |
| --- | --- |
| **Match ALL of these** | **Match NONE of these** |
| Peter Smith | peter smith Peter smith peter Smith PEter Smi7h Peter SmIth |

### Implementing the Solution

Import **re**, create your **pattern** (don't forget to **escape the special characters**), and use the **findall()** method to get all the matches. Then print them:



### Examples

|  |
| --- |
| **Input** |
| Peter Smith, peter smith, Peter smith, peter Smith, PEter Smith, Peter SmIth, Lily Everett |
| **Output** |
| Peter Smith Lily Everett |

## 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** ", ". The input will be a string containing phone numbers and other symbols.

### 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 is followed by **a number**:
  + The number consists of **7 digits** (separated into **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 RegEx properties to **help** with the matching:

* Use **quantifiers** to match a **specific number** of **digits**
* Use a **capturing group** to make sure the delimiter is **only one of the allowed characters** **(space or hyphen)** and **not** a **combination** of both (e.g., +359 2-111 111 has **mixed delimiters**, it is **invalid**). Use a **group backreference** to achieve this.
* Add a **word boundary** at the **end** of the match to avoid **partial matches**.
* **Ensure** that there is a **space** **before** the **'+'** **sign,** or it is positioned at the **beginning of the string**.

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

|  |  |
| --- | --- |
| **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

The input will be a string containing phone numbers and other symbols.

|  |
| --- |
| **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 |

### Implement the Solution

Import re, create your pattern, read the text, use the **findall()** method, and print the result:



## Match Dates

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

### Compose the Regular Expression

Every valid date has the following characteristics:

* It 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 one of these symbols: a period ("."), a hyphen ("**-**") or a forward slash ("**/**").
* The separator must be **the same** for the whole date (e.g., 13.03.2016 is valid, 13.03/2016 is **NOT**). Use a **group back reference** 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 **capturing groups** for the **day**, **month,** and **year**.

Since this problem requires more complex RegEx, which includes **named capturing groups**, we will take a look at how to construct it:

* First off, we do not want anything at the **start** of our date, so we're going to use a **word boundary** "\b":  
  
* Next, we are going to match the **day** by telling our RegEx to match **exactly two digits**,and since we want to **extract** the day from the match later, we're going to put it in a **capturing group**:  
  
* Next comes the separator – either a **hyphen**, **period,** or **forward slash**. We can use a **character class** for this:

  
Since we want to use the separator, we matched here to match the **same separator** further into the date. We're going to put it in a **capturing group**:  


* Next comes the **month**, which consists of a **capital Latin letter** and **exactly two lowercase Latin letters**:
* Next, we are going to match the **same separator** **we matched earlier**. We can use a **backreference** for that:  
  
* Next up, we are going to match the year, which consists of **exactly 4 digits**:  
  
* Finally, since we do not want to match the date if there's anything else **glued to it**, we're going to use another **word boundary** for the end:  
  

Now it is time to find all the **valid dates** in the input and **print each date** in the following format: "Day: {day}, Month: {month}, Year: {year}", each on a **new line**.

### Implement the Solution

First, import re, create the pattern, and read the text:



Then, we find all the matches:



Now in the matches, we have the following:



* We have an array of matches
* Each match has a tuple of all the matches (day, separator, month, year)

So, we print each match in the right format



### 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 |

## Match Numbers

Write a program that 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**](http://www.regular-expressions.info/lookaround.html)). 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 ("-").
* It 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 **$** or what's called a [**positive lookahead**](http://www.regular-expressions.info/lookaround.html)). The syntax for the **end** of the **RegEx** might look something like "($|(?=\s))".

Let's see how we would translate the above rules into a **regular expression**:

* First off, we need to establish what needs to exist **before** our number. We can't use \b here, since it includes "-", which we need to match **negative numbers**.   
  Instead, we'll use a **positive look behind**, which **matches** if there's something **immediately behind** it. We'll match if we're either at the **start** of the string (^)or if there's any **whitespace** **behind** the string:  
  
* Next, we'll check whether there's a **hyphen** signifying a **negative number**:  
  Since having a negative sign **isn't required**, we'll use the "?" quantifier, which means "**between 0 and 1 times**".
* After that, we'll match any integers – naturally, consisting of **one or more digits.** However, it **will match "00"**, but it is not what we want. So, we should be more specific:  
  
* Next, we'll match the **decimal** part of the number, which **might or might not exist** (note: we need to escape the **period** character, as it's used for something else in RegEx):  
  
* Finally, we're going to use the same logic for the end of our string as the start – we're going to match **only** if the number has **either whitespace or the end of the string ("**$**")**:  
  

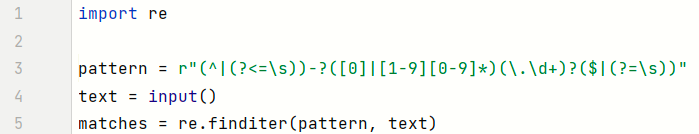
You can follow the table below to help with composing your RegEx:

|  |  |
| --- | --- |
| **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 00.5 |

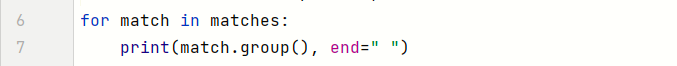
Find all the **numbers** from the string and **print them** on the **console**, separated by **spaces**.

### Implement the Solution

Now that we've written our regular expression, we can start by putting it in a variable and extracting the matches:



After that, it's only a matter of printing the numbers, 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 00.5 | 1 -1 123 -123 123.456 -123.456 |