

# Lab: Regular Expressions PHP

Problems for lab for the ["PHP Fundamentals" course @ SoftUni](#).

You can check your solutions in [Judge](#).

## 1. Match Full Name

Write a PHP 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**.

To help you out, we've outlined several steps:

1. Use an online regex tester like <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 of letters across new lines, put "\b" at the beginning and at the end of your regex. This will ensure that what precedes and what follows the match is a word boundary (like a new line).

In order 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
Ivan Ivanov	ivan ivanov, Ivan ivanov, ivan Ivanov, IVan Ivanov, Ivan IvAnov, Ivan Ivanov

By the end, the matches should look something like this:

TEST STRING SWITCH TO UNIT TESTS ▶

Vankata IvAnov, Ivan ivanov, Ivan Ivanov, ivan ivanov, ivan Ivanov, IVan Ivanov, Ivan Ivanov

After you've constructed your regular expression, it's time to write the solution in PHP.

## Implementing the Solution in PHP

Create a new PHP project and copy your **regular expression** into a **string** variable:

```
<?php

$pattern = '/\b[A-Z][a-z]+ [A-Z][a-z]+/';
```

Note: It's usually a good idea to use a **verbatim string** (`/` in front and `/` at the end of the string literal) to store **regular expressions**, since characters like the backslash `"\"` can clash with **string escaping**.

Now, it's time to **read the input** and **extract all the matches full matches** from it. For this, we can use the `Matches[0]` class:

```
<?php

$pattern = '/\b[A-Z][a-z]+ [A-Z][a-z]+/';

$names = readline();
preg_match_all($pattern, $names, &$matches);
```

After we extract all the matches, we need to **iterate** over the `matches[0]` and **print** every match **full match** that we found:

```
<?php

$pattern = '/\b[A-Z][a-z]+ [A-Z][a-z]+/';

$names = readline();
preg_match_all($pattern, $names, &$matches);

foreach ($matches[0] as $fullName) {
    echo $fullName.PHP_EOL;
}
```

## 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 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** (the last example on the right-hand side).
- Ensure that before the '+' sign there is either a **space** or the **beginning of the string**.

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

## Implement the Solution in PHP

Now it's time to write the solution, so let's start writing!

First, just like in the previous problem, put your RegEx in an array:

```
<?php
$pattern = '/\+359([\ -| ])2(\1)\d{3}(\1)\d{4}\b/';
```

After that, let's make an array of matches and on index **0** will be array of our **full matches**:

```
<?php
$pattern = '/\+359([\ -| ])2(\1)\d{3}(\1)\d{4}\b/';

$input = readline();

preg_match_all($pattern, $input, &matches: $phones);

echo implode( glue: ', ', $phones[0]);
```

## 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 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**.

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

- First off, we don’t want anything at the **start** of our date, so we’re going to use a **word boundary** “\b”:

### REGULAR EXPRESSION

:/ \b

- Next, we’re 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**:

### REGULAR EXPRESSION

:/ \b(\d{2})

We’re also going to give our group a **name**, since it’s easier to navigate by **group name** than by **group index**:

### REGULAR EXPRESSION

:/ \b(?<day>\d{2})

- Next comes the separator – either a **hyphen**, **period** or **forward slash**. We can use a **character class** for this:

### REGULAR EXPRESSION

:/ \b(?<day>\d{2})[-.\//]

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**:

### REGULAR EXPRESSION

:/ \b(?<day>\d{2})([-.\//])

- Next comes the **month**, which consists of a **capital Latin letter** and **exactly two lowercase Latin letters**:

#### REGULAR EXPRESSION

```
:/\b(?<day>\d{2})([-.\./])(?<month>[A-Z][a-z]{2})
```

- Next, we're going to match the **same separator we matched earlier**. We can use a **backreference** for that:

#### REGULAR EXPRESSION

```
:/\b(?<day>\d{2})([-.\./])(?<month>[A-Z][a-z]{2})\2
```

- Next up, we're going to match the year, which consists of **exactly 4 digits**:

#### REGULAR EXPRESSION

```
:/\b(?<day>\d{2})([-.\./])(?<month>[A-Z][a-z]{2})\2(?<year>\d{4})
```

- Finally, since we don't want to match the date if there's anything else **glued to it**, we're going to use another **word boundary** for the end:

#### REGULAR EXPRESSION

```
:/\b(?<day>\d{2})([-.\./])(?<month>[A-Z][a-z]{2})\2(?<year>\d{4})\b
```

Now it's 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 in PHP

First off, we're going to put our matches in an array from the string:

```
<?php
```

```
$pattern = '/\b(?<day>\d{2})([-.\./])(?<month>[A-Z][a-z]{2})\2(?<year>\d{4})\b/';
$datesAsString = readline();
preg_match_all($pattern, $datesAsString, &matches: $dates);
```

We're going to **iterate** over the count of our full matches and **extract** the **day**, **month** and **year** from the **groups**. We can use a for loop full to get a match's group **value** by its **key**, the **same way** as when we access a **array's values**

```
$matchesCount = count($dates[0]);
for($i = 0 ; $i < $matchesCount; $i++){
    $day = $dates['day'][$i];
    $month = $dates['month'][$i];
    $year = $dates['year'][$i];

    echo "Day: $day, Month: $month, Year: $year".PHP_EOL;
}
```

## 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

## 4. 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) /`.

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 lookbehind**, 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:

**REGULAR EXPRESSION**

```
// (^|(?<=\s))
```

- Next, we'll check whether there's a **hyphen**, signifying a **negative number**:

**REGULAR EXPRESSION**

```
// (^|(?<=\s))-?
```

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 **one or more digits**:

**REGULAR EXPRESSION**

```
// (^|(?<=\s))-?\d+
```

- 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):

#### REGULAR EXPRESSION

```
// (^|(?<=\s))-?\d+(\.\d+)?
```

- 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 a whitespace or the end of the string ("")**:

#### REGULAR EXPRESSION

```
// (^|(?<=\s))-?\d+(\.\d+)?($|(?=\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**.

## Implement the Solution in PHP

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

```
<?php

$pattern = '/(^|(?<=\s))-?\d+(\.\d+)?($|(?=\s))/' ;

$numbers = readline();
preg_match_all($pattern, $numbers, &$matches);
```

After that, it's only a matter of printing the numbers, separated by spaces:

```
echo implode( glue: " ", $matches[0] );
```

## 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