Lab: Regular Expressions (RegEx)

1. Match Full Name

Write a PHP Program to **match full names** from a list of names and **print** them.

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).

2. Match Phone Number

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

A valid number has the following characteristics:

- It starts with "**+961**" and **before** the **'+'** sign there is either a **space** or the **beginning of the string**.
- Then, it is followed by the code (**1 or 2 numbers**)
- Then, it **can be** followed by a **separator** (**either a space or a slash** ('**/**')).
- After that, it's followed by the **number** itself:
  - The number consists of **6 digits** (in **one group** or separated in **two groups** of **3 digits** or separated in **three groups** of **2 digits**).

You can use the following RegEx properties to **help** with the matching:

- Use **quantifiers** to match a **specific number** of **digits**
- Add a **word boundary** at the **end** of the match to avoid **partial matches**.

3. Match Dates

Write a program, which matches a date in the format:

`dd`

`separator`

`Mmm`

`separator`

`yyyy`

Every valid date has the following characteristics:

- Always starts with **two digits** (e.g. **07**, **11**) 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.

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**.

First off, we're going to put our RegEx in a variable and save matches in **`$matches`**

```
$re = '/\b(?<day>\d{2})(?<delimiter>[-.\/])(?<month>[A-Z][a-z]{2})(\k<delimiter>)(?<year>\d{4})\b/m';
$str = readline();
preg_match_all($re, $str, &$matches);
```

Next, we're going to **iterate** over every single **`match`** and **extract** the **day**, **month** and **year** from the **groups**.

```
for ($i = 0; $i < count($matches['day']); $i++) {
    echo "Day: " . $matches['day'][$i] . ', '
        . "Month: " . $matches['month'][$i] . ', '
        . "Year: " . $matches['year'][$i] . 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.

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**.

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

```
$re = '/(^|(?<=\s))-?\d+(\.\d+)?($|(?=\s))/m';
$str = readline();
preg_match_all($re, $str, &$matches);
```

After that, you can join the elements with space and print:

```
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