

Лекция 9

Регулярные выражения в РНР

Регулярные выражения.

Общая информация

Регулярное выражение (regular expression, regex, регэксп) - механизм, позволяющий задать шаблон для строки и осуществить поиск данных, соответствующих этому шаблону в заданном тексте. Кроме того, дополнительные функции по работе с regex'ами позволяют получить найденные данные в виде массива строк, произвести замену в тексте по шаблону, разбиение строки по шаблону и т.п.

Однако главной их функцией, на которой основаны все остальные, является именно функция поиска в тексте данных, соответствующих шаблону, описанному в синтаксисе регулярных выражений.

Пример самого простого регулярного выражения:

`'/веб-программирование/'` - соответствует строке, в которой есть слово 'веб-программирование'.

Регулярные выражения. Общая информация

Сутью механизма **регулярных выражений** является то, что они позволяют задать шаблон для нечеткого поиска по тексту. Например, если стоит задача найти в тексте определенное слово, то с этой задачей хорошо справляются и обычные функции работы со строками. Однако если вам нужно найти "то, не знаю что", о чем вы можете сказать только то, как приблизительно это должно выглядеть, придется воспользоваться регулярными выражениями.

Например, необходимо найти в тексте информацию, про которую вам известно только то, что это "3 или 4 цифры после которых через пробел идет 5 заглавных латинских букв".

```
/\d{3,4} \s[A-Z]{5}/
```

Синтаксис регулярных выражений

Регулярные выражения представляют собой строку. Строка всегда начинается с символа разделителя, за которым следует непосредственно регулярное выражение, затем еще один символ разделителя и потом необязательный список модификаторов. В качестве символа разделителя обычно используется слэш ('/'). Таким образом, в следующем регулярном выражении: `\d{3}-\d{2}/m`, символ '/' является разделителем, строка `\d{3}-\d{2}` - непосредственно регулярным выражением, а символ 'm', расположенный после второго разделителя - это модификатор.

Основой синтаксиса регулярных выражений является тот факт, что некоторые символы, встречающиеся в строке, рассматриваются не как обычные символы, а как имеющие специальное значение (так называемые метасимволы). Именно это решение позволяет работать всему механизму регулярных выражений. Каждый метасимвол имеет свою собственную роль в синтаксисе регулярных выражений.

Синтаксис регулярных выражений

Одним из самых важных метасимволов является символ обратного слэша ('\\'). Если в строке встречается этот символ, то парсер рассматривает символ, непосредственно следующий за ним двояко:

- если следующий символ в обычном режиме имеет какое-либо специальное значение, то он теряет это свое специальное значение и рассматривается как обычный символ. Это совершенно необходимо для того, чтобы иметь возможность вставлять в строку специальные символы, как обычные. Например, метасимвол '.' в обычном режиме означает "любой единичный символ", а '\\.' означает просто точку. Также можно лишить специального значения и сам этот символ: '\\\\'.
- если следующий символ в обычном режиме не имеет никакого специального значения, то он может получить такое значение, будучи соединенным с символом '\\'. К примеру, символ 'd' в обычном режиме воспринимается просто как буква, однако, будучи соединенной с обратным слэшем ('\\d') становится метасимволом, означающим "любая цифра".

Синтаксис регулярных выражений.

Метасимволы

Некоторые наиболее часто используемые метасимволы.

Метасимвол	Значение
Метасимволы для задания символов, не имеющих изображения	
<code>\n</code>	Символ перевода строки (код 0x0A)
<code>\r</code>	Символ возврата каретки (код 0x0D)
<code>\t</code>	Символ табуляции (код 0x09)
<code>\xhh</code>	Вставка символа с шестнадцатеричным кодом 0xhh, например <code>\x41</code> вставит латинскую букву 'A'

Синтаксис регулярных выражений.

Метасимволы

Некоторые наиболее часто используемые метасимволы.

Метасимвол	Значение
Метасимволы для задания групп символов	
<code>\d</code>	Цифра (0-9)
<code>\D</code>	Не цифра (любой символ кроме символов 0-9)
<code>`</code>	Пустой символ (обычно пробел и символ табуляции)
<code>\s</code>	Непустой символ (все, кроме символов, определяемых метасимволом <code>\s</code>)
<code>\w</code>	"Словесный" символ (символ, который используется в словах. Обычно все буквы, все цифры и знак подчеркивания ('_'))
<code>\W</code>	Все, кроме символов, определяемых метасимволом <code>\w</code>

Синтаксис регулярных выражений.

Метасимволы

Приведем несколько простейших примеров.

```
/\d\d\d/
```

Любое трехзначное число ('123', '719', '001').

```
/\w\s\d\d/
```

Буква, пробел (или табуляция) и двузначное число ('A 01', 'z 45', 'S 18').

```
/\d and \d/
```

Любая из следующих строк: '1 and 2', '9 and 5', '3 and 4'.

Синтаксис регулярных выражений. Метасимволы

Синтаксис регулярных выражений имеет средства для определения собственных подмножеств символов. Например, нам может понадобиться задать условие, что в этом месте строки должна находиться шестнадцатеричная цифра или еще что-то подобное. Для описания таких подмножеств применяются символы квадратных скобок '[]'. Квадратные скобки, встречающиеся внутри регулярного выражения, считаются одним символом, который может принимать значения, перечисленные внутри этих скобок.

Синтаксис регулярных выражений.

Метасимволы

Метасимволы, которые работают внутри таких секций, это:

- Обратный слэш ('\'). Т.е. все метасимволы из приведенной ранее таблицы будут работать.
- Минус ('-'). Используется для задания набора символов из одного промежутка (например все цифры могут быть заданы как '0-9').
- Символ '^'. Если этот символ стоит первым в секции задания подмножества символов (и только в этом случае!) он будет рассматриваться как символ отрицания. Т.о. можно задать все символы, которые не описаны в данной секции.

Синтаксис регулярных выражений.

Метасимволы

Приведем несколько простейших примеров.

```
[0-9A-Fa-f]
```

Цифра в шестнадцатеричной системе счисления

```
[\dA-Fa-f]
```

То же самое, но с использованием метасимвола

```
[02468]
```

Четная цифра

```
[^\d]
```

Все, кроме цифр (аналог метасимвола `\D`)

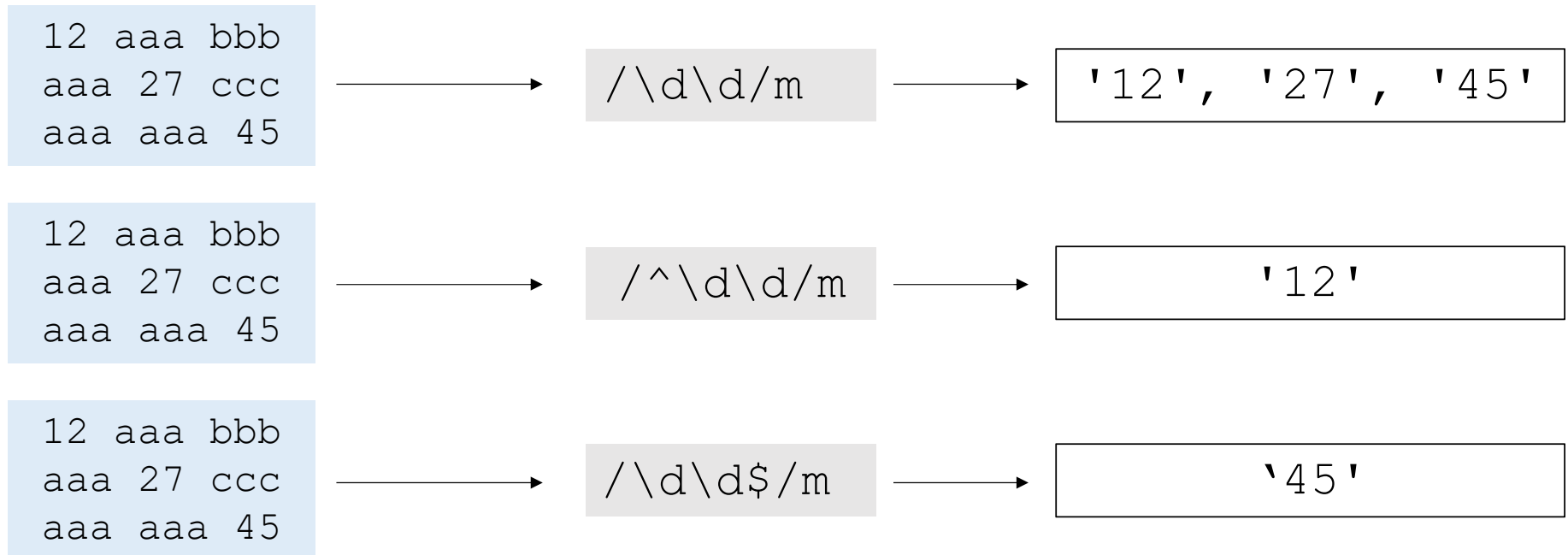
```
[a^b]
```

Любой из символов 'a', 'b', '^'. Заметьте, что здесь символ '^' не имеет какого-либо специального значения, потому что стоит не на первой позиции внутри квадратных скобок.

Синтаксис регулярных выражений.

Метасимволы

Символы '^' и '\$'. Они используются для того, чтобы указать парсеру регулярных выражений на то, чтобы он обратил внимание на положение искомого текста в строке. Символ '^' указывает, что искомый текст должен находиться в начале строки, символ '\$' наоборот, указывает, что искомый текст должен находиться в конце строки.



Синтаксис регулярных выражений.

Метасимволы

Символ точки `'.'`. Этот метасимвол указывает, что на данном месте в строке может находиться любой символ (за исключением символа перевода строки). Очень удобно использовать его, если вам нужно "пропустить" какую-нибудь букву в слове при проверке. Например регулярное выражение `/.bc/` найдет в тексте и `'abc'` и `'Abc'` и `'Zbc'` и `'5bc'`.

Синтаксис регулярных выражений. Метасимволы

Символ вертикальной черты '|'. Используется для задания списка альтернатив. Например, регулярное выражение: /(красное|зеленое) яблоко/ найдет в тексте все словосочетания 'красное яблоко' и 'зеленое яблоко'.

Синтаксис регулярных выражений.

Метасимволы

Символы круглых скобок '(' и ')'. Эти символы позволяют получить из искомой строки дополнительную информацию. Если парсер встречается внутри регулярного выражения круглые скобки, то он рассматривает содержимое этих скобок как еще одно регулярное выражение, по которому необходимо произвести поиск. Парсер рекурсивно вызывает сам себя для поиска по новому регулярному выражению и использует результаты поиска для дальнейшей обработки основного регулярного выражения.

При этом, если поиск хотя бы по одному из внутренних регулярных выражений не увенчался успехом - поиск по всему регулярному выражению считается безуспешным.

Синтаксис регулярных выражений.

Метасимволы

Рассмотрим пример

Необходимо проверить, является ли строка семизначным телефонным номером с указанием кода города, и получить из нее код города и номер телефона.

```
/\ ((\d{3,5}) \) \s+ (\d{3}-\d{2}-\d{2})) /
```

Первая круглая скобка здесь теряет свое специальное значение и будет рассматриваться как обычный символ:

```
/\ ( (\d{3,5}) \) \s+ (\d{3}-\d{2}-\d{2})) /
```


Синтаксис регулярных выражений.

Метасимволы

Рассмотрим пример

Далее идет регулярное выражение в скобках (проверка кода города):

```
/\ ( \d{3,5} ) \s+ ( \d{3} - \d{2} - \d{2} ) /
```

После этого идет закрывающая круглая скобка, которая также лишена своего специального значения из-за символа обратного слэша, стоящего перед ней:

```
/\ ( ( \d{3,5} ) \) \s+ ( \d{3} - \d{2} - \d{2} ) /
```

Синтаксис регулярных выражений.

Метасимволы

Рассмотрим пример

Затем идет пропуск пустого места:

```
/\ ((\d{3,5}) \) \s+ (\d{3}-\d{2}-\d{2}) /
```

И еще одно регулярное выражение в скобках, которое проверяет номер телефона:

```
/\ ((\d{3,5}) \) \s+ (\d{3}-\d{2}-\d{2}) /
```

Как видите, здесь есть 3 регулярных выражения - основное и два внутренних. При этом основное выражение позволяет нам проверить, имеет ли строка необходимый нам формат, а два внутренних - получить соответственно код города и номер телефона. Т.е. одним регулярным выражением мы можем решить сразу несколько задач.

Синтаксис регулярных выражений. Метасимволы

Рассмотрим пример

```
/\ ((\d{3,5}) \) \s+ (\d{3}-\d{2}-\d{2}) /
```

Посмотрим, как работает это регулярное выражение.

Пусть у нас есть строка: "My phone is (095) 123-45-67".

Результатами поиска будут 3 строки: '(095) 123-45-67', '095' и '123-45-67'.

Синтаксис регулярных выражений. Метасимволы (количественные показатели)

Группа метасимволов, определяющих количественные показатели (т.н. quantifiers).

Очень часто бывает необходимо указать, что какой-то символ должен повторяться определенное количество раз. Конечно, можно просто указать его необходимое количество раз непосредственно в строке, но это, естественно, не выход. Тем более, что очень часто встречаются ситуации, когда точное количество символов неизвестно. Поэтому синтаксис регулярных выражений содержит набор метасимволов, предназначенных именно для решения подобных задач.

Каждый из описанных ниже метасимволов определяет количественную характеристику символа, который находится непосредственно перед ним.

Синтаксис регулярных выражений.

Метасимволы (количественные показатели)

Звездочка '*'. Указывает, что символ должен быть повторен 0 или более раз (т.е. символ может отсутствовать или присутствовать в любых количествах).
Пример: выражение `/ab*c/` найдет строки `'ac'`, `'abc'`, `'abbc'` и т.д.

Плюс '+'. Указывает, что символ должен быть повторен 1 или более раз (т.е. символ обязан присутствовать и может присутствовать в любых количествах).
Пример: выражение `/ab+c/` найдет строки `'abc'`, `'abbc'`, `'abbbc'` и т.д., но не найдет строку `'ac'`.

Знак вопроса '?'. Указывает, что символ может как присутствовать, так и нет, но при этом не может повторяться более одного раза. Пример: выражение `/ab?c/` найдет строки `'ac'` и `'abc'`, но не найдет строку `'abbc'`.

Синтаксис регулярных выражений.

Метасимволы (количественные показатели)

Фигурные скобки '{' и '}'. Определяют количественную характеристику символа. Внутри скобок через запятую перечисляются минимальное и максимальное количество повторений символа. При этом любой из параметров может быть опущен, а кроме того можно задать точное количество повторений, указав только одно число. Примеры:

{2,4} - символ должен повториться минимум 2 раза, но не более 4.

{5} - символ может отсутствовать (т.к. не задано минимальное количество повторений), но если присутствует, то не должен повторяться более 5 раз.

{3,} - символ должен повторяться минимум 3 раза, но может быть и больше.

{4} - символ должен повторяться ровно 4 раза.

Синтаксис регулярных выражений.

Метасимволы (количественные показатели)

Есть еще одна тонкость в использовании метасимвола '?'. Посмотрите на такое выражение: `/.+a/`. Ожидается, что оно вернет нам часть текста до первого вхождения символа 'a' в этот текст. На самом деле оно будет работать несколько не так, как ожидается, и результатом поиска будет весь текст до последнего вхождения символа 'a'. Дело в том, что по умолчанию количественные метасимволы "жадничают" и пытаются захватить как можно больший кусок текста. Если это не нужно (как в нашем случае), то необходимо "отучить" их от жадности, указав знак '?' после количественного метасимвола: `/.+?a/`. После этого выражение будет работать так как надо.

Синтаксис регулярных выражений.

Модификаторы регулярных выражений

Механизм регулярных выражений позволяет добавлять модификаторы, влияющие на обработку регулярного выражения. Ниже рассмотрены наиболее употребительные.

Модификатор	Описание
i	Выполняет поиск без учета регистра. Например <code>/a/i</code> ищет и <code>a</code> , и <code>A</code> .
m	Выполняет многострочный поиск (шаблоны, которые ищут начало или конец строки, будут соответствовать началу или концу каждой строки)
u	Обеспечивает правильное сопоставление шаблонов в кодировке UTF-8 (для поиска русского текста например).
U	Инвертирует "жадность" (по умолчанию жадный, т.е. пытается захватить как можно большую строку, подходящую по условию).
s	Если используется, то символ точка (<code>.</code>) соответствует и переводу строки. Иначе она ему не соответствует.
x	Игнорировать пробелы. В этом случае пробелы нужно экранировать обратным слэшем <code>\</code> .

Синтаксис регулярных выражений.

Модификаторы регулярных выражений

При использовании модификаторов, можно использовать знак минус (-) для отключения модификатора. Например: (?m-i) — включаем многострочный поиск и отключаем регистронезависимый.

Синтаксис регулярных выражений.

Границы слов

Символ границы слова (`\b`) помогает искать слова, которые начинаются и/или заканчиваются определенным шаблоном. Например, регулярное выражение `/\bcar/` соответствует словам, начинающимся с шаблона `car`, и будет соответствовать `cart`, `carrot`, или `cartoon`, но не будет совпадать с `oscar`.

Аналогично, регулярное выражение `/car\b/` соответствует словам, оканчивающимся шаблоном `car`, и будет соответствовать `oscar` или `supercar`, но не будет соответствовать `cart`.

Аналогично, `/\bcar\b/` соответствует словам, начинающимся и заканчивающимся шаблоном `car` и будет соответствовать только слову `car`.

Синтаксис регулярных выражений. Просмотр вперед/назад

Большинство реализаций регулярных выражений поддерживают функцию просмотра вперёд и назад или опережающий и ретроспективный поиск (англ. — lookahead, lookbehind). Посмотрим на примере.

У нас есть следующее регулярное выражение, которое находит две подстроки:

```
/LudovicXVI/
```

LudovicXV, **LudovicXVI**, **LudovicXVIII**, LudovicLXVII, LudovicXXL

Синтаксис регулярных выражений.

Просмотр вперед/назад

Предположим, что нам не нужно включать в результаты поиска часть подстроки с римскими цифрами XVI. Для этого "обернём" её вот в такую конструкцию:

```
/Ludovic(?=XVI) /
```

LudovicXV, **Ludovic**XVI, **Ludovic**XVIII, LudovicLXVII, LudovicXXL

Как мы видим, условия сопоставления, заданные первоначальным выражением, не изменились. Сопоставились те же подстроки, что и в предыдущем примере. Однако символы XVI в совпавших подстроках не были включены в окончательный результат поиска. Такое поведение называется **позитивным просмотром вперёд** или **позитивным опережающим поиском**.

Синтаксис регулярных выражений.

Просмотр вперед/назад

Логика позитивного просмотра вперед можно описать следующим образом.

Регулярное выражение $a(?=b)$ находит совпадения таких a , за которыми следует b , не делая b частью сопоставления.

Просмотр вперед также может быть негативным - тогда он будет искать совпадения в тех подстроках, где указанная в скобках часть подстроки отсутствует. В нашем случае - это по-прежнему XVI. Чтобы из позитивного просмотра сделать негативный, заменим символ $=$ на $!$. Теперь у нас сопоставились три другие подстроки:

```
/Ludovic(?!XVI) /
```

LudovicXV, LudovicXVI, LudovicXVIII, LudovicLXVII, LudovicXXL

Синтаксис регулярных выражений.

Просмотр вперед/назад

Кроме просмотра вперед, существует **просмотр назад** или **ретроспективный поиск**. Он работает похожим образом, но ищет совпадения символов, расположенных после сгруппированной в скобках части регулярного выражения, которая не будет включена в сопоставление.

Иными словами, при позитивном просмотре назад регулярное выражение **(?<=b)a** находит совпадения таких **a**, перед которыми находится **b**, не делая **b** частью сопоставления.

Для позитивного просмотра назад используется дополнительный знак **<**. В этом примере мы находим совпадения подстроки **Two**, перед которыми следует **One**:

```
/ (?<=One ) Two /
```

One **Two**, Three Two

Синтаксис регулярных выражений. Просмотр вперед/назад

Чтобы изменить позитивный просмотр назад на негативный, точно так же, как и в просмотре вперед, меняем = на !:

```
/ (?<!One ) Two /
```

One Two, Three **Two**

Функции регулярных выражений

Функция preg_match()

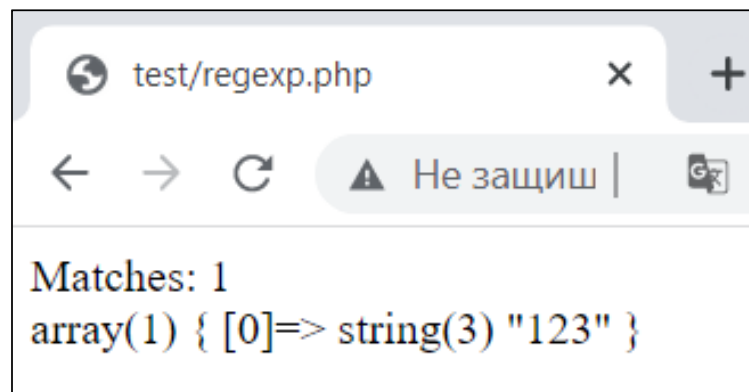
```
int preg_match (string pattern, string subject [, array matches])
```

Эта функция предназначена для проверки того, совпадает ли заданная строка (subject) с заданным регулярным выражением (pattern). В качестве результата функция возвращает 1, если совпадения были найдены и 0, если нет. Если при вызове функции был задан необязательный параметр matches, то после работы функции ему будет присвоен массив, содержащий результаты поиска по заданному регулярному выражению. Заметьте, что вне зависимости от того, сколько именно совпадений было найдено при поиске - вам будет возвращено только первое совпадение.

Функции регулярных выражений

Функция preg_match()

```
<?php
$str = "123 234 345 456 567";           // Строка для поиска
$result = preg_match('/\d{3}/', $str, $found); // Производим поиск
echo "Matches: $result<br>";           // Выводим количество найденных
совпадений
var_dump($found);                       // Выводим результат поиска
?>
```



Функции регулярных выражений

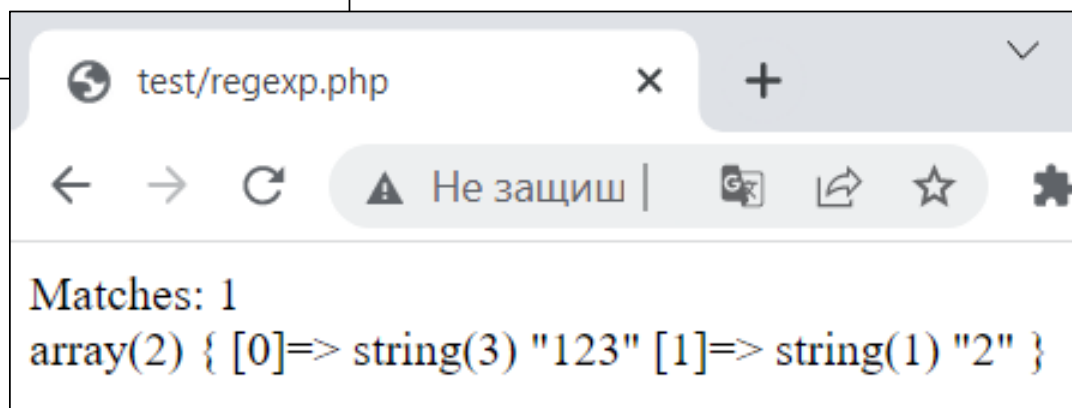
Функция `preg_match()`

Реально функция `preg_match()` обнаружила в заданной строке 5 совпадений с заданным выражением, но вернула только первое из них. Казалось бы, что в этом случае было бы логичнее возвращать результаты поиска в виде строки, а не в виде массива, но это не так. Вспомните, что регулярное выражение может содержать в себе внутренние регулярные выражения, которые также возвращают результат. А для того, чтобы вернуть результаты поиска по всем регулярным выражениям, нам как раз и требуется массив. Для того, чтобы проиллюстрировать сказанное выше, давайте немного изменим регулярное выражение и посмотрим на результат:

Функции регулярных выражений

Функция preg_match()

```
<?php
$str = "123 234 345 456 567";
// Теперь мы не просто ищем трехзначное число,
// но и получаем его среднюю цифру
$result = preg_match('/\d(\d)\d/', $str, $found);
echo "Matches: $result<br>";
var_dump($found);
?>
```



Функции регулярных выражений

Функция preg_match_all()

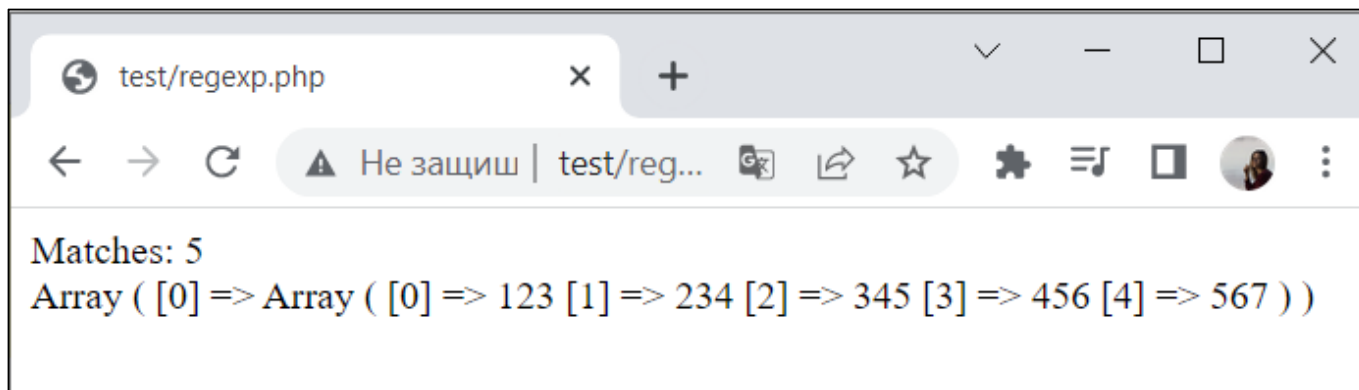
```
int preg_match_all (string pattern, string subject, array matches [, int order])
```

Эта функция очень похожа на `preg_match()` и предназначена для тех же самых целей. Единственное ее отличие от `preg_match()` состоит в том, что она осуществляет "глобальный" поиск в заданном тексте по заданному регулярному выражению и, соответственно, находит и возвращает все имеющиеся совпадения.

Функции регулярных выражений

Функция preg_match_all()

```
<?php
$str = "123 234 345 456 567";
$result = preg_match_all('/\d{3}/', $str, $found);
echo "Matches: $result<br>";
print_r($found);
?>
```



Функции регулярных выражений

Функция `preg_match_all()`

Необходимо обратить внимание на дополнительный параметр, появившийся в этой функции по сравнению с `preg_match()`: **`order`**. Значение этого параметра определяет структуру выходного массива с найденными совпадениями. Его значение может быть одним из перечисленных ниже:

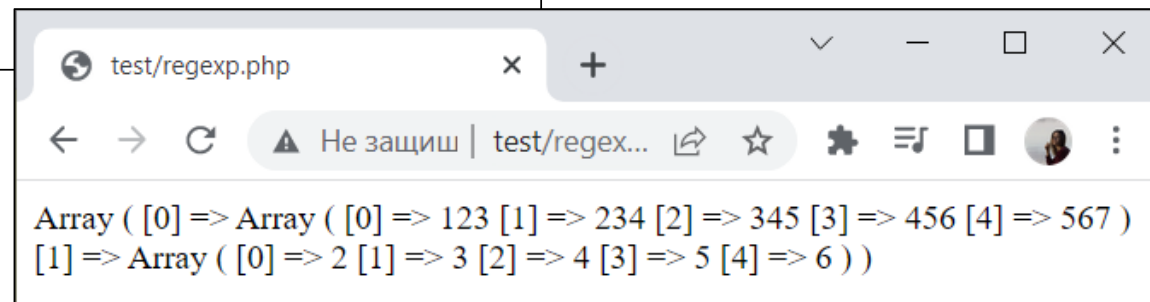
- **`PREG_PATTERN_ORDER`** - результаты поиска будут сгруппированы по номеру регулярного выражения, которое возвратило этот результат (это значение используется по умолчанию).
- **`PREG_SET_ORDER`** - результаты поиска будут сгруппированы по месту их нахождения в тексте.

Функции регулярных выражений

Функция preg_match_all()

```
<?php
$str = "123 234 345 456 567";
$order = PREG_PATTERN_ORDER;
$result = preg_match_all('/\d(\d)\d/', $str, $found, $order);
print_r($found);
?>
```

Как видите, массив результатов содержит внешние индексы, соответствующие номерам регулярных выражений, от которых получен результат (индекс 0 имеет основное регулярное выражение). По этим индексам в массиве расположены массивы, содержащие непосредственно найденную информацию, причем индекс в этом внутреннем массиве соответствует "порядковому номеру" данного фрагмента в исходном тексте.

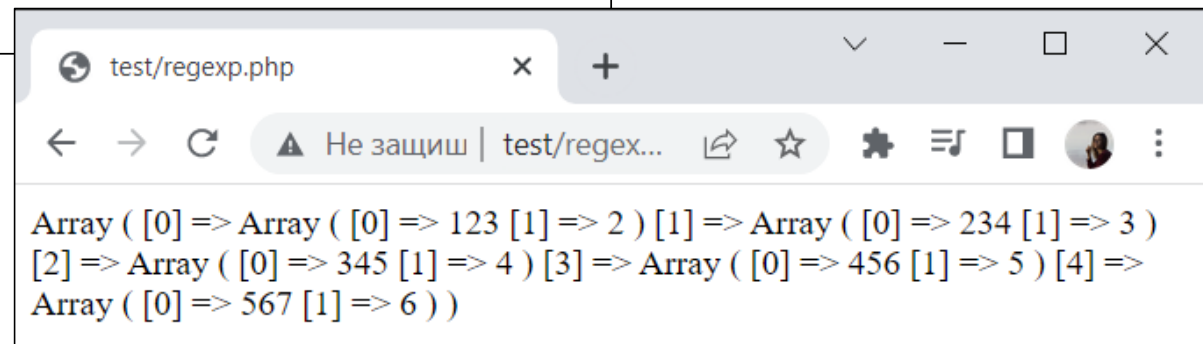


Функции регулярных выражений

Функция preg_match_all()

```
<?php
$str = "123 234 345 456 567";
$order = PREG_SET_ORDER;
$result = preg_match_all('/\d(\d)\d/', $str, $found, $order);
print_r($found);
?>
```

Здесь основной массив содержит результаты поиска, сгруппированные по порядку их нахождения в тексте, причем каждый результат представляет собой массив с результатами поиска по этому найденному фрагменту для всех имеющихся регулярных выражений.



Функции регулярных выражений

Функция preg_replace()

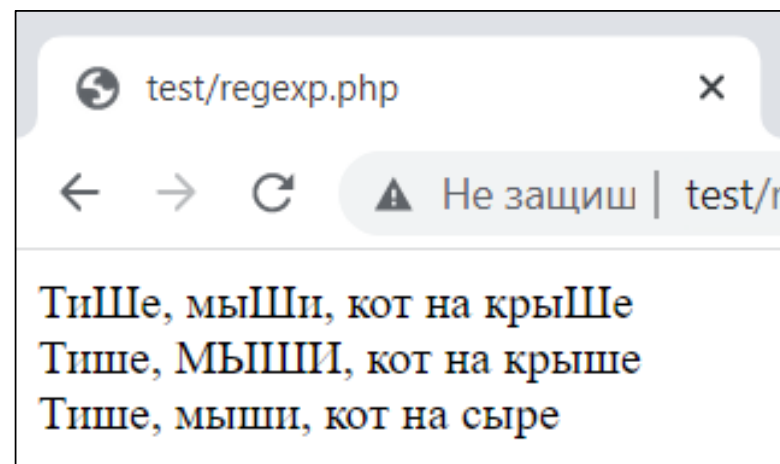
```
mixed preg_replace (mixed pattern, mixed replacement, mixed subject [, int limit])
```

Эта функция позволит произвести замену текста по регулярному выражению. Как и в предыдущих функциях, здесь производится поиск по регулярному выражению **pattern** в тексте **subject**, и каждый найденный фрагмент текста заменяется на текст, заданный в **replacement**. Задание необязательного параметра **limit** позволит ограничить количество заменяемых фрагментов в тексте.

Функции регулярных выражений

Функция preg_replace()

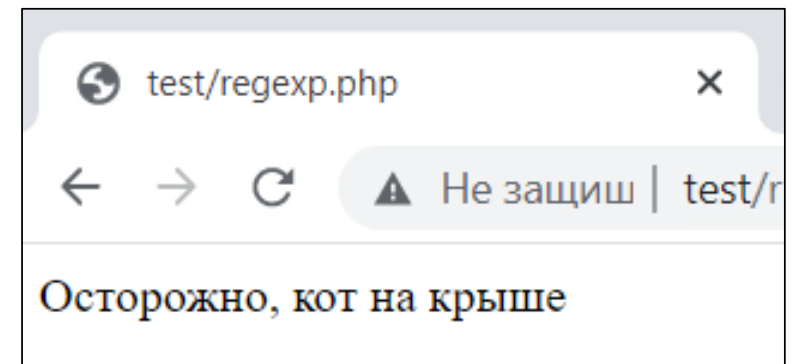
```
<?php
echo preg_replace('/ш/iu', 'Ш', 'Тише, мыши, кот на крыше');
echo preg_replace('#м.ш.#iu', 'МЫШИ', 'Тише, мыши, кот на крыше');
echo preg_replace('#к...е#iu', 'сыре', 'Тише, мыши, кот на крыше');
?>
```



Функции регулярных выражений

Функция preg_replace()

```
<?php
    echo preg_replace('#Т.+и#iu', 'Осторожно', 'Тише, мыши, кот на крыше');
?>
```



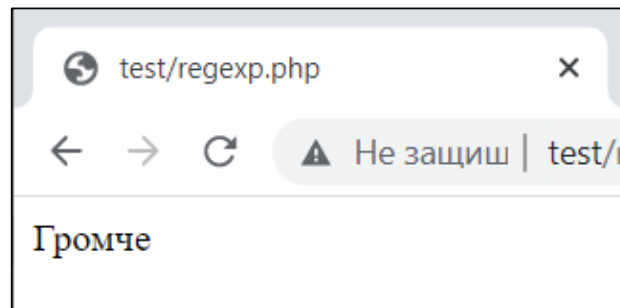
Функции регулярных выражений

Функция preg_replace(). Ограничение жадности

Регулярное выражение захватывает максимально возможное количество символов. Поэтому результат может быть неожиданным. К примеру, предположим, что нам надо заменить в строке "Тише, мыши, кот на крыше" слово "Тише" на "Громче". Для этого используем выражение:

```
<?php
    echo preg_replace('#T.+e#u', 'Громче', 'Тише, мыши, кот на крыше');
?>
```

Может показаться, что выражение работает правильно. Но по факту оно найдёт букву "Т" в начале строки и букву "е" не в конце первого слова, а в конце всей строки. Результат получится такой:

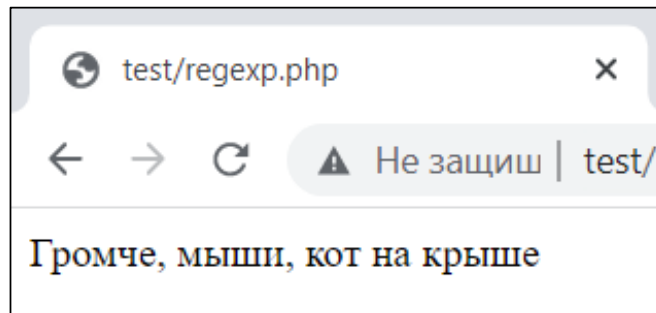


Функции регулярных выражений

Функция preg_replace(). Ограничение жадности

Чтобы этого не происходило, необходимо ограничить жадность регулярного выражения. Для этого необходимо после оператора повторения (звёздочки "*" или плюса "+") поставить знак вопроса "?", чтобы они перестали быть такими жадными.

```
<?php
    echo preg_replace('#T.+?e#u', 'Громче', 'Тише, мыши, кот на крыше');
?>
```



Ограничивать жадность можно всем оператором повторения. В том числе самому "?", делая так: "??".

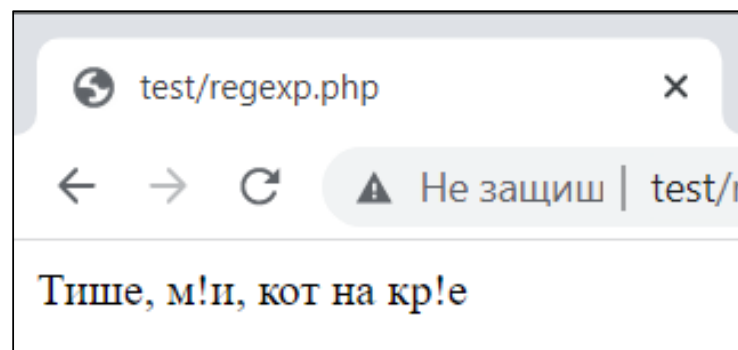
Функции регулярных выражений

Функция preg_replace(). Группирующие круглые скобки

Как было сказано ранее, операторы повторения работают только на символ, который стоит до них. Если же необходимо применить оператор повторения на несколько символов, то необходимо поставить скобки. Приведём пример:

```
<?php
    echo preg_replace('#(ыш)+#iu', '!', 'Тише, мыши, кот на крыше');
?>
```

Из кода видно, что оператор повторения будет применяться к двум символам, стоящим в круглых скобках - (ыш). Результат выполнения такого кода:



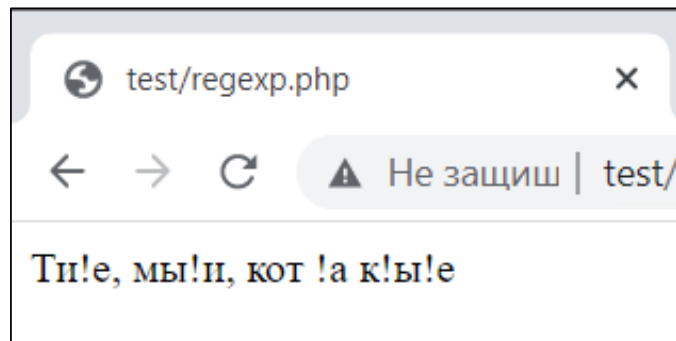
Функции регулярных выражений

Функция preg_replace(). Квадратные скобки

Существуют квадратные скобки. Если круглые являются "группирующими", или в терминах логики они делают соединение "и", то квадратные скобки - это логическое "или". Приведём пример:

```
<?php
    echo preg_replace('#[ншр]+#u', '!', 'Тише, мыши, кот на крыше');
?>
```

В результате выполнения этого кода все буквы "н", "ш", "р", которые стоят в квадратных скобках [ншр] будут заменены на восклицательный знак "!".

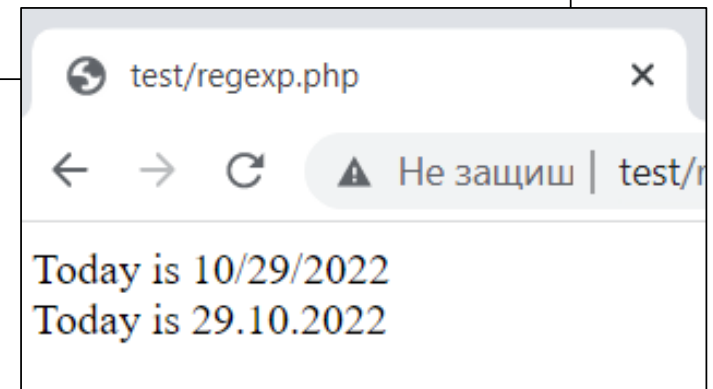


Функции регулярных выражений

Функция preg_replace()

Конвертация дат из одного формата в другой. Как вы знаете, на Западе обычно используется формат mm/dd/yyyy, тогда как у нас обычно - dd.mm.yyyy. Следующий пример осуществляет конвертацию дат между этими форматами в заданном тексте:

```
<?php
$text = 'Today is 10/29/2022';
echo $text, "<br>";
$text = preg_replace("/(\d{2})\/(\d{2})\/(\d{4})/", "\\2.\\1.\\3", $text);
echo $text;
?>
```



Функции регулярных выражений

Функция preg_split()

```
array preg_split (string pattern, string subject [, int limit [, int flags]])
```

Данная функция выполняет действие, аналогичное функциям `split()` и `explode()` - разбивает строку на части по какому-либо признаку и возвращает массив, содержащий части строки. Однако ее возможности по заданию правил разбиения больше, чем у этих функций, потому что в ее основе лежит механизм регулярных выражений. Если говорить более конкретно, то строка **subject** разбивается на части по разделителю, заданному регулярным выражением **pattern**.

При этом количество фрагментов может быть ограничено необязательным параметром **limit**. Специальное значение **limit**, равное -1 или 0, подразумевает отсутствие ограничения.

Кроме того эта функция поддерживает необязательный параметр **flags**, который позволяет в некоторой степени контролировать процесс разбиения строки.

Функции регулярных выражений

Функция preg_split()

```
array preg_split (string pattern, string subject [, int limit [, int flags]])
```

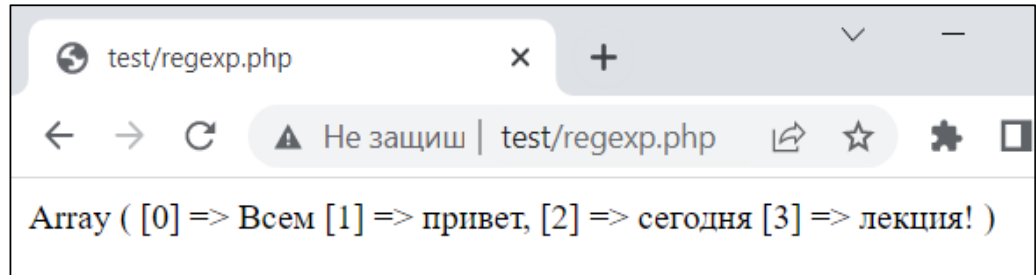
Параметр flags может принимать следующие значения (или их комбинации с использованием знака '|'):

- **PREG_SPLIT_NO_EMPTY** - возвращать только непустые части строк, полученные в результате разбиения.
- **PREG_SPLIT_DELIM_CAPTURE** - возвращать также результаты поиска по внутренним регулярным выражениям.
- **PREG_SPLIT_OFFSET_CAPTURE** - если указан этот флаг, для каждой найденной подстроки будет указана её позиция в исходной строке. Необходимо помнить, что этот флаг меняет формат возвращаемого массива: каждый элемент будет содержать массив, содержащий в индексе с номером 0 найденную подстроку, а смещение этой подстроки в параметре subject - в индексе 1.

Функции регулярных выражений

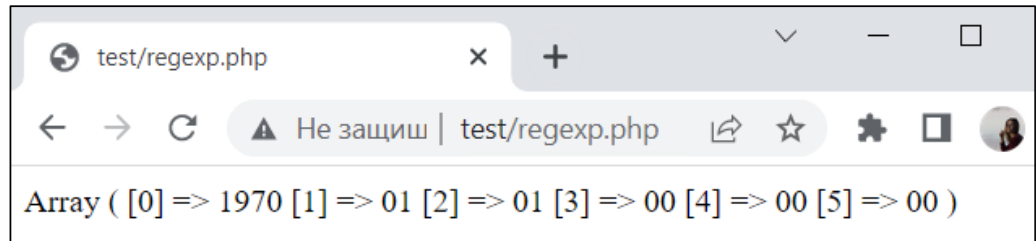
Функция preg_split()

```
<?php
$text = "Всем привет, сегодня лекция!";
$words = preg_split("/\s+/s", $text);
print_r($words);
?>
```



The screenshot shows a web browser window with the address bar displaying "test/regexp.php". The page content shows the output of the preg_split function: "Array ([0] => Всем [1] => привет, [2] => сегодня [3] => лекция!)".

```
<?php
$date = "1970-01-01 00:00:00";
$pattern = "/[-\s:]/";
$components = preg_split($pattern,
    $date);
print_r($components);
?>
```



The screenshot shows a web browser window with the address bar displaying "test/regexp.php". The page content shows the output of the preg_split function: "Array ([0] => 1970 [1] => 01 [2] => 01 [3] => 00 [4] => 00 [5] => 00)".

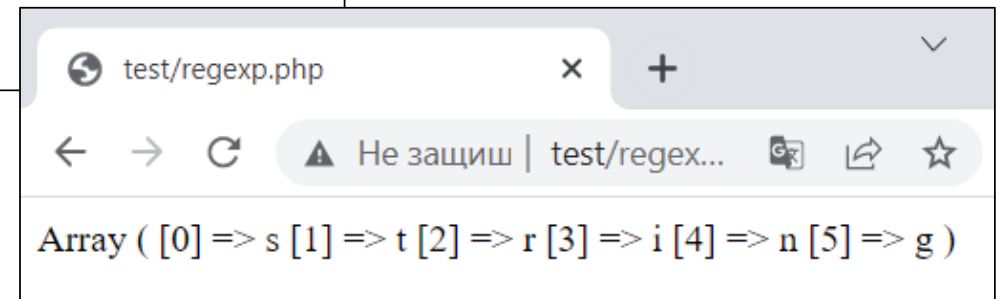
Функции регулярных выражений

Функция preg_split()

```
<?php
$str = 'string';
$chars = preg_split('//', $str, -1, PREG_SPLIT_NO_EMPTY);
print_r($chars);
?>
```

Здесь применен флаг `PREG_SPLIT_NO_EMPTY`, который возвращает только непустые части строк, полученные в результате разбиения.

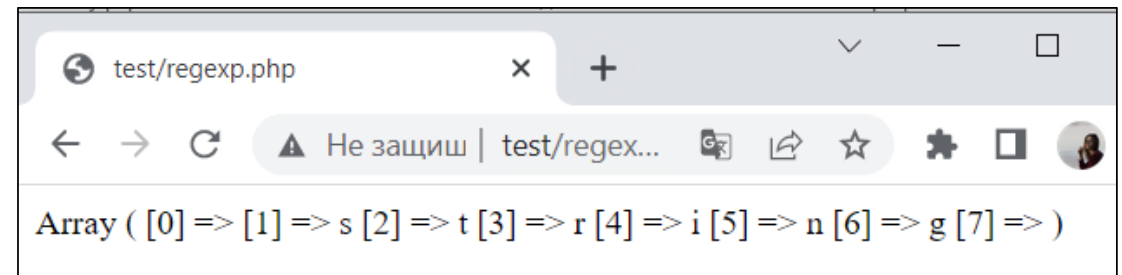
```
<?php
$str = 'string';
$chars = preg_split('//', $str, -1);
print_r($chars);
?>
```



test/regexp.php

← → ↻ ⚠ Не защищ | test/regex... 📄 🔗 ☆

Array ([0] => s [1] => t [2] => r [3] => i [4] => n [5] => g)



test/regexp.php

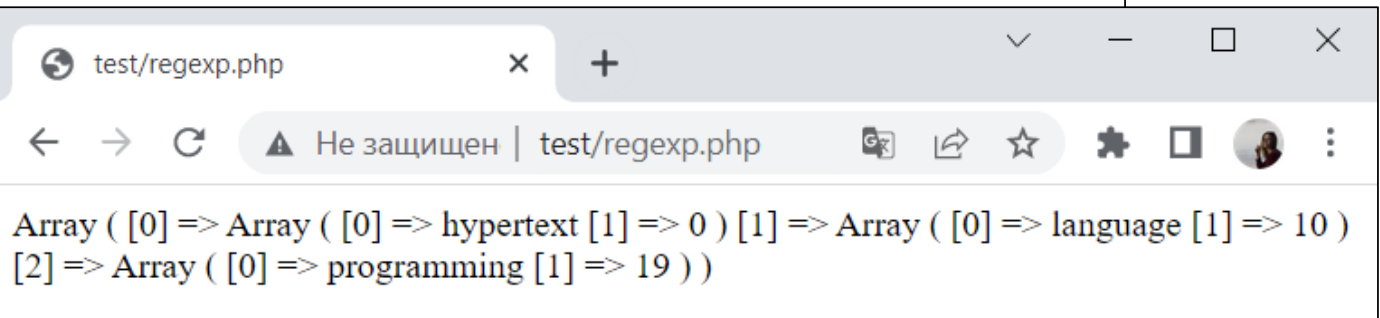
← → ↻ ⚠ Не защищ | test/regex... 📄 🔗 ☆ ⚙️ 🗑️ 👤

Array ([0] => [1] => s [2] => t [3] => r [4] => i [5] => n [6] => g [7] =>)

Функции регулярных выражений

Функция preg_split()

```
<?php
$date = "1970-01-01 00:00:00";
$pattern = "/([-\\s:])/";
$components = preg_split($pattern, $date, -1, PREG_SPLIT_DELIM_CAPTURE);
print_r($components);
?>
```



Здесь применен флаг `PREG_SPLIT_DELIM_CAPTURE`, который возвращает для каждой найденной подстроки её позицию в исходной строке.

Функции регулярных выражений

Функция preg_quote()

```
string preg_quote (string str [, string delimiter])
```

Экранирует символы в регулярных выражениях.

Функция **preg_quote()** принимает строку **str** и добавляет обратный слеш перед каждым служебным символом. Это бывает полезно, если в составлении шаблона участвуют строковые переменные, значение которых в процессе работы скрипта может меняться.

В регулярных выражениях служебными считаются следующие символы:

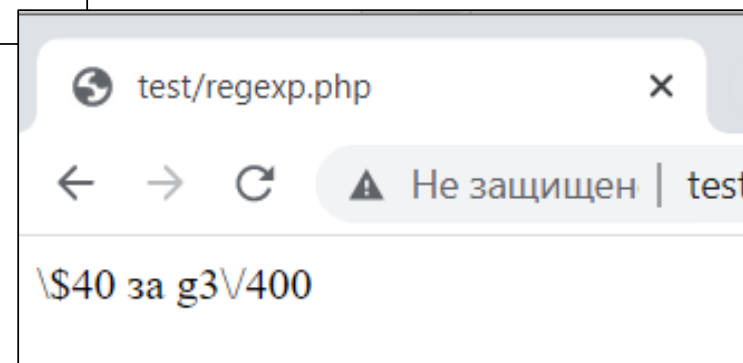
`.\+ * ? [^] $ () { } = ! < > | : - #`

Обратите внимание, что символ `/` не является служебным.

Функции регулярных выражений

Функция preg_quote()

```
<?php
$keywords = '$40 за g3/400';
$keywords = preg_quote($keywords, '/');
echo $keywords; // возвращает \$40 за g3\400
?>
```



Функции регулярных выражений

Функция preg_grep()

```
array preg_grep (string pattern, array input, int flags)
```

Она ищет текст по регулярному выражению `pattern`, в массиве `input` и возвращает новый массив, содержащий только элементы, в которых были найдены совпадения с заданным регулярным выражением.

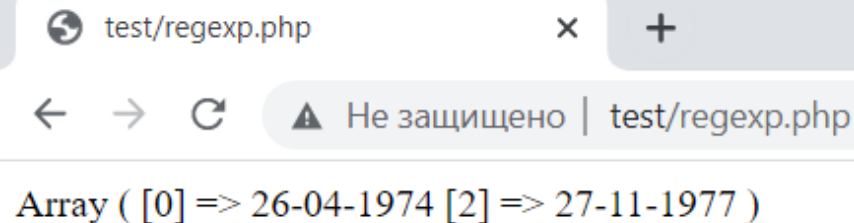
flags

В случае, если установлен в `PREG_GREP_INVERT`, функция `preg_grep()` возвращает те элементы массива, которые не соответствуют заданному шаблону `pattern`.

Функции регулярных выражений

Функция preg_grep()

```
<?php
$arr_input = array("26-04-1974", "Сергей", "27-11-1977", "Юля");
$arr_output = preg_grep("/(\d{2})-(\d{2})-(\d{4})/", $arr_input);
print_r($arr_output);
?>
```

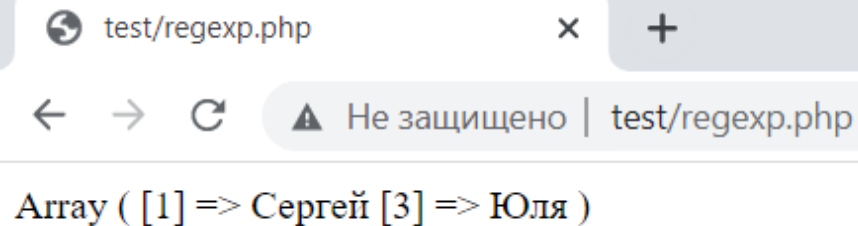


The screenshot shows a web browser window with a single tab titled 'test/regexp.php'. The address bar contains the same text. Below the address bar, there is a security warning icon and the text 'Не защищено | test/regexp.php'. The main content area of the browser displays the output of the PHP script: 'Array ([0] => 26-04-1974 [2] => 27-11-1977)'.

Функции регулярных выражений

Функция preg_grep()

```
<?php
$arr_input = array("26-04-1974", "Сергей", "27-11-1977", "Юля");
$arr_output = preg_grep("/(\d{2})-(\d{2})-(\d{4})/", $arr_input, PREG_GREP_INVERT);
print_r($arr_output);
?>
```



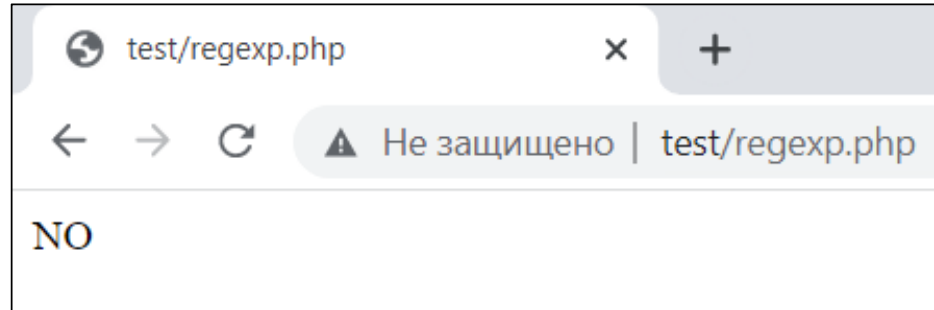
The screenshot shows a web browser window with a single tab titled "test/regexp.php". The address bar displays "test/regexp.php" with a warning icon and the text "Не защищено" (Not secure). The main content area of the browser shows the output of the PHP script: "Array ([1] => Сергей [3] => Юлия)".

Функции регулярных выражений

Примеры

Проверяем, является ли переменная числом.

```
<?php
$var = '123a';
if (!preg_match("|^\[\d\]+\$", $var))
{
    echo "NO";
}
else
{
    echo "OK";
}
?>
```

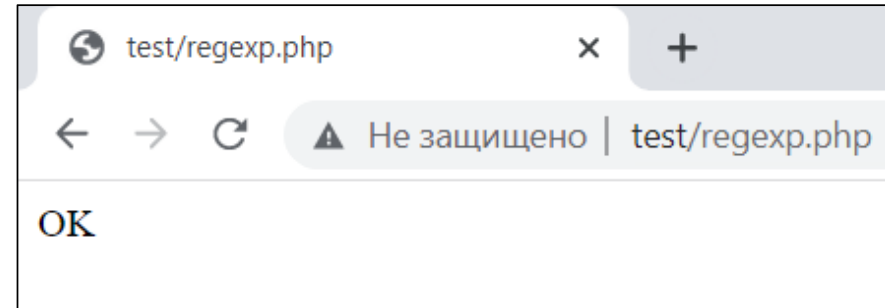


Функции регулярных выражений

Примеры

Проверяем, является ли переменная числом.

```
<?php
$var = '123';
if (!preg_match("|^\[\d\]+\$", $var))
{
    echo "NO";
}
else
{
    echo "OK";
}
?>
```

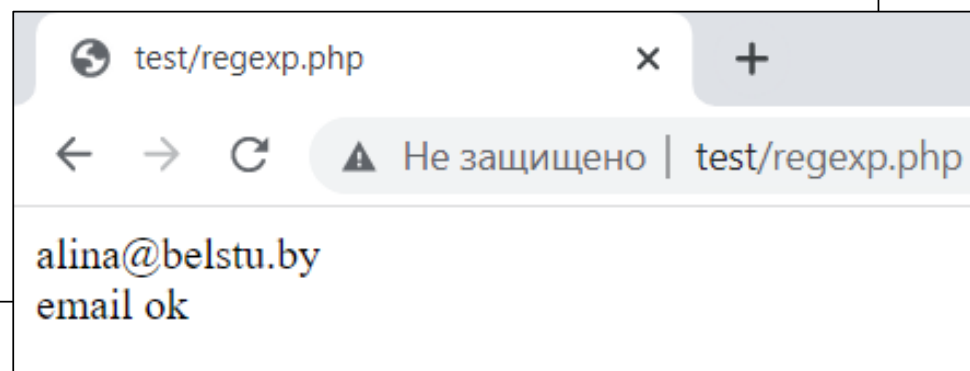


Функции регулярных выражений

Примеры

Проверка адреса e-mail.

```
<?php
$usermail= 'alina@belstu.by';
echo $usermail, "<br>";
if (preg_match("/^\w+([\.\w]+)*\w@\w((\.\w)*\w+)*\.\w{2,3}$/", $usermail))
{
    echo "email ok";
}
else
{
    echo "email invalid";
}
?>
```

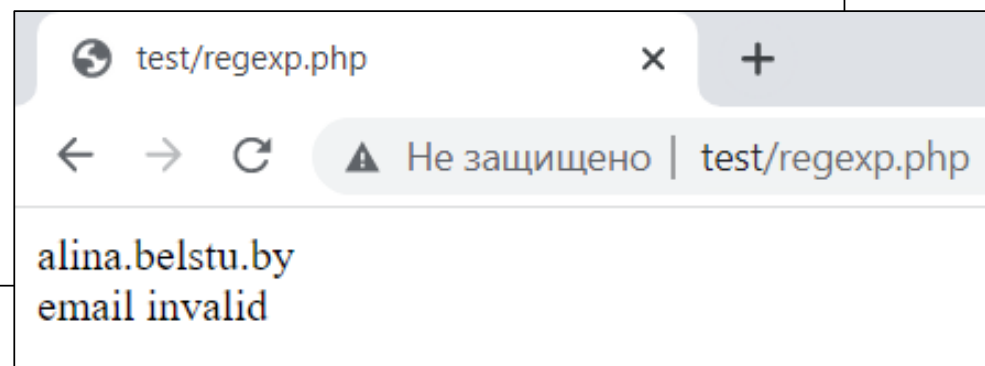


Функции регулярных выражений

Примеры

Проверка адреса e-mail.

```
<?php
$usermail= 'alina.belstu.by';
echo $usermail, "<br>";
if (preg_match("/^\w+([\.\w]+)*\w@\w((\.\w)*\w+)*\.\w{2,3}$/", $usermail))
{
    echo "email ok";
}
else
{
    echo "email invalid";
}
?>
```



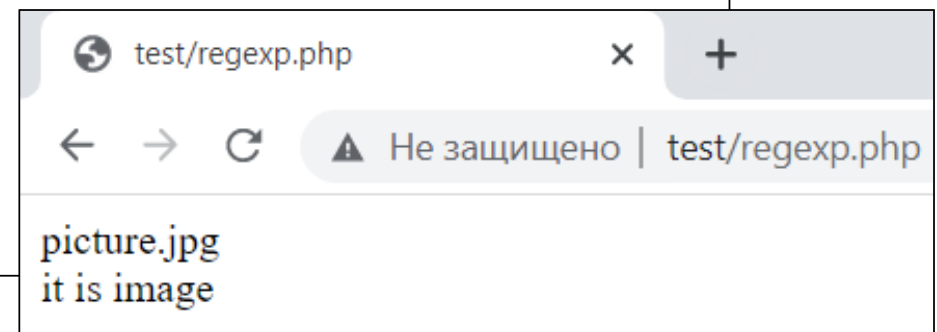
Функции регулярных выражений

Примеры

Проверка расширения файла

```
<?php
$my_image= 'picture.jpg';
echo $my_image, "<br>";
if
(preg_match("/\.(?:jp(?:e?g|e|2)|gif|png|tiff?|bmp|ico)$/i", $my_image))
{
    echo "it is image";
}
else
{
    echo "it is not image";
}
?>
```

(?:) - не сохраняющие скобки отличаются от обычных тем, что совпавший внутри них текст не захватывается механизмом регулярных выражений и не может быть использован, как ссылка на эту группу (*подмаску*). Не сохраняющие скобки служат для ограничения группы альтернативного выбора (*ИЛИ* "|")



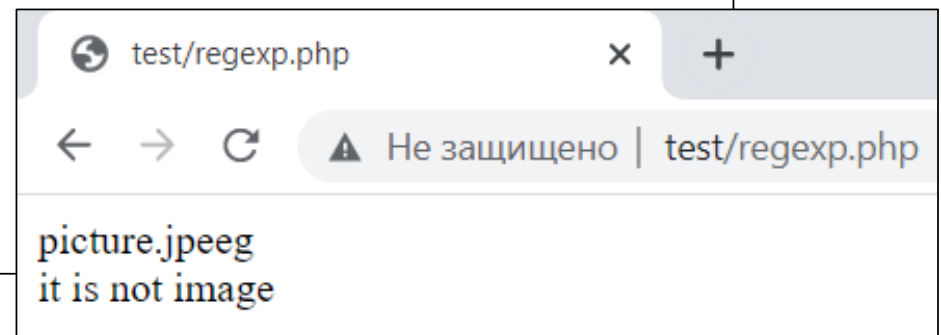
Функции регулярных выражений

Примеры

Проверка расширения файла

```
<?php
$my_image= 'picture.jpeeg';
echo $my_image, "<br>";
if
(preg_match("/\.(?:jp(?:e?g|e|2)|gif|png|tiff?|bmp|ico)$/i", $my_image))
{
    echo "it is image";
}
else
{
    echo "it is not image";
}
?>
```

(?:) - не сохраняющие скобки отличаются от обычных тем, что совпавший внутри них текст не захватывается механизмом регулярных выражений и не может быть использован, как ссылка на эту группу (*подмаску*). Не сохраняющие скобки служат для ограничения группы альтернативного выбора (*ИЛИ* "|")

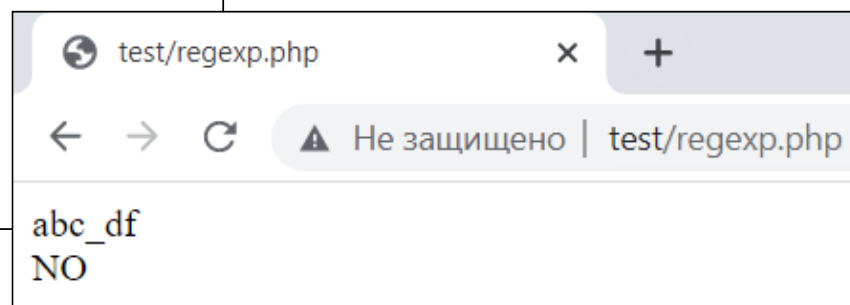


Функции регулярных выражений

Примеры

Состоит ли строка только из букв, цифр и "_", длиной от 8 до 20 символов

```
<?php
$string= 'abc_df';
echo $string, "<br>";
if (preg_match("/^[a-zA-я0-9_]{8,20}$/", $string))
{
    echo "YES";
}
else
{
    echo "NO";
}
?>
```

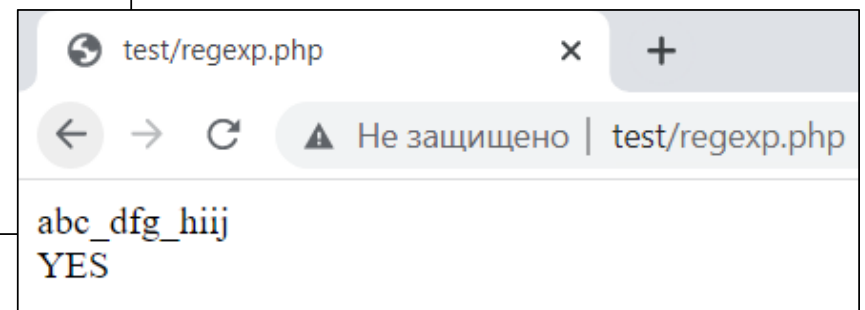


Функции регулярных выражений

Примеры

Состоит ли строка только из букв, цифр и "_", длиной от 8 до 20 символов

```
<?php
$string= 'abc_dfg_hiij';
echo $string, "<br>";
if (preg_match("/^[a-zA-я0-9_]{8,20}$/", $string))
{
    echo "YES";
}
else
{
    echo "NO";
}
?>
```



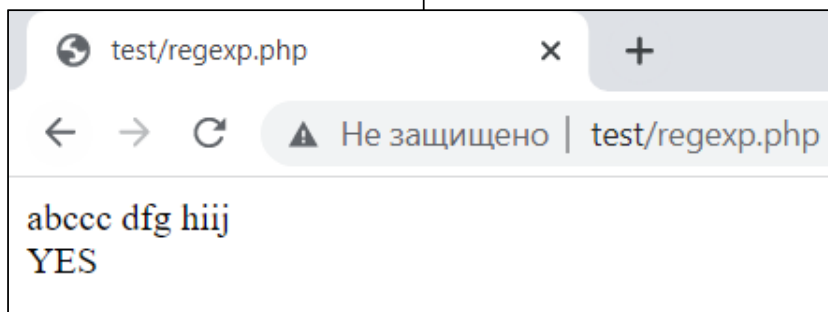
Функции регулярных выражений

Примеры

Проверка повторяющихся символов. Есть ли в строке идущие подряд символы, не менее 3-х символов подряд?

```
<?php
$string= 'abccc dfg hiij';
echo $string,"<br>";
if (preg_match("/(.)\\1\\1/", $string))
{
    echo "YES";
}
else
{
    echo "NO";
}
?>
```

\1 - обратная ссылка, ссылаемся на подмаску



Функции регулярных выражений

Примеры

Проверка повторяющихся символов. Есть ли в строке идущие подряд символы, не менее 3-х символов подряд?

```
<?php
$string= 'abcccc dfg hiij';
echo $string,"<br>";
if (preg_match("/(.)\\1{4}/",$string))
{
    echo "YES";
}
else
{
    echo "NO";
}
?>
```

\1 - обратная ссылка, ссылаемся на подмаску

