REALFIVE
Учебно-практические курсы программирования

Практическое занятие №36 Regexp

Задание №1

Проверьте запущен ли у Вас Open Server. Папка с материалами урока (php36.loc) должна быть размещена в директории /domains сервера. Заходим на http://php36.loc и начинаем урок.

Задание №2

Регулярные выражения - это специальные шаблоны для поиска подстроки в тексте. С их помощью можно решить такие (а также другие) задачи:

1. Определить есть ли подстрока, соответствующая шаблону, в тексте.
2. Найти подстроки, соответствующие шаблону, в тексте.
3. Разбить строку по шаблону.
4. Заменить текст в строке по шаблону

Как видите задачи в общем похожи на те, которые мы решаем, используя стандартные функции для строк, только вместо конкретной строки у нас шаблон строки.

Регулярное выражение состоит из 2 частей: шаблона и модификаторов. Разделяются они символами-ограничителями. В качестве таких символов можно использовать любой не буквенно-цифровой символ кроме '\'. Чаще всего используют '/' или '#'. Если символ-ограничитель встречается в шаблоне, его необходимо экранировать.

Для того чтобы составлять шаблоны нужно знать спецсимволы регулярных выражений. Тестировать примеры можно тут https://regex101.com/.

**** - символ экранирования.

Пример: /qwe\/rty/ - соответствует строке, в которой есть qwe/try. Символ / мы заэкранировали, после чего он перестал выполнять в данном месте свое специальное значение (он являлся символом-ограничителем).

^ - символ начала данных.

Пример: /^pattern/ - соответствует строке, которая начинается словом pattern.

$ - символ конца данных.

Пример: /pattern$/ - соответствует строке, которая заканчивается словом pattern.

. - любой символ, кроме перевода строки.

Пример: /./ - соответствует любой не пустой строке.

Пример: /pat.ern/ - Соответствует и строке, содержащей pattern, или patdern, или pat3ern...

[] - внутри этих скобок перечисляются символы, любой один символ из которых может стоять на данном месте. Это называется символьным классом.

Пример: /pat[aoe]rn/ - под соответствие попадут только строки, содержащие patarn, patorn или patern.

? - одно или ноль вхождений предшествующего символа (символьного класса или подмаски).

***** - любое количество вхождений предшествующего символа (символьного класса или подмаски). В том числе и ноль.

+ - одно или более вхождений символа (символьного класса или подмаски).

Пример: /a?b*c+/ - соответствует строке в начале которой 1 символ a или без него, после произвольное кол-во символов b, после него после произвольное кол-во символов c, но хотябы 1: “abbbbcccc”, “c”, “acccc”, “bccc”…

| - Или

() – подмаска

Пример: /as+(es|du)?.*r/ - Буква a, потом одна или больше букв s, после этого сочетание es или du может быть один раз, а может и ни разу, потом любое количество любых символов и буква r.

{a,b} - количество вхождений предшествующего символа или подмаски от a до б. Если б не указан, считается, что верхней границы нет.

Пример: /a{5,8}/ - строка в которой есть 5, 6, 7 или 8 символов a: aaaaa, aaaaaa, aaaaaaa, aaaaa, aaaaaaaa

? – после *,+,{a,b} – переключатель жадности. По умолчанию количественные метасимволы – жадные, то есть захватывают в себя максимально доступное кол-во символов, если указать после них знак вопроса, они станут ленивыми – захватят, минимум символов

Пример: /.+/ - захватит всю строку kgjklfgjklgfjkl. /.+?/ - захватит только 1 символ.

Спецсимволы внутри символьного класса.

^ - отрицание.

Пример: [^da] - соответствует любому символу кроме d и a.

Пример: [^^] - соответствует любому символу кроме ^.

Пример: [d^a] - соответствует любому символу из перечисленных трёх. [\^da] - то же самое.

В последнем примере, как видно символ стоит не в начале перечисления и свою метафункцию теряет. И экранировать его, кстати, тоже тут не надо.

- - внутри символьного класса означает символьный интервал.

Пример: [0-9a-e] - соответствует любому символу от 0 до 9 и от a до e. Если в символьном классе надо перечислить сам символ дефиса, то следует либо заэранироватьего, либо разместить перед].

Осторожно в символьном классе надо использовать символ \. Если его поставить перед], она окажется заэкранирована. Также окажется заэкранированным любой символ, который может быть заэкранирован. Иначе символ \ является обычным символом.

Символ $ тоже является обычным символом внутри символьного класса. И скобки тоже.

Символ \. Одна из его функций - снятие специального значения с спецсимволов. А другая, наоборот придание специальных функций обычным символам.

\cx - ctrl + x. На месте x может быть любой символ.

\e - escape.

\f - разрыв страницы.

\n, \r, \t - это нам и так привычно. Перевод строки, возврат каретки и табуляция.

\d - любой символ, означающий десятичную цифру, аналог [0-9].

\D - любой символ, не означающий десятичную цифру.

\s - любой пробельный символ (пробел, переносы строк, табуляции).

\S - не пробельный.

\w - любая цифра, буква или знак подчеркивания.

\W - любой символ, но не \w.

\b - граница слова. Удобно использовать для разбиения строки по словам.

\B - не граница слова.

\xHH - символ с шестнадцатиричным кодом HH. Используется редко.

\”число” – обратная ссылка на подмаску. Обратная ссылка сопоставляется с частью строки, захваченной соответствующей подмаской, но не с самой подмаской.

/(sens|respons)e and **\1**ibility/ - соответствует "sense and sensibility", "response and responsibility", но не "sense and responsibility".

Модификаторы указываются после символа-разделителя. Есть следующие модификаторы:

i - если этот модификатор используется, символы в шаблоне соответствуют символам как верхнего, так и нижнего регистра. /[A-Z]/i - без модификатора – соответствует только символам верхнего регистра латинского алфавита, с модификатором всем символам латинского алфавита (любого регистра).

s – если указан спецсимвол «.» будет также включать символы переноса строки. /.+/s – без модификатора включает в себя только 1 строку многострочного текста, с модификатором все строки.

u – шаблон и целевая строка обрабатываются как UTF-8 строки. Добавлен в php (недоступен в других языках) для обработки utf-8 строк через регулярные выражения.

U – этот модификатор инвертирует жадность квантификаторов, таким образом они по умолчанию не жадные. Но становятся жадными, если за ними следует символ ?.

Это базовый синтаксис регулярных выражений подробнее смотрите в официальной документации:

1. Синтаксис http://php.net/manual/ru/reference.pcre.pattern.syntax.php

2. Модификаторы http://php.net/manual/ru/reference.pcre.pattern.modifiers.php

И после синтаксиса пару примеров шаблонов:

1. /^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$/ - шаблон ip адреса.
2. ^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$ - более правильный шаблон ip адреса (исключает числа большие 255).
3. /<img(?=\s|>)(?!(?:[^>=]|=(['"])(?:(?!\1).)*\1)*?\salt=['"])[^>]*>/ - все img без тега alt.
4. /^\(\d{3}\)\s\d{2,3}-\d{2}-\d{2}$/ - телефон в формате (777) 777-77-77 или (777) 77-77-77

Задание №3

Для работы с регулярными выражениями есть множество функций, поэтому рассмотрим основные из них (полный список http://php.net/manual/ru/ref.pcre.php).

preg_match - выполняет проверку на соответствие регулярному выражению.

Основные варианты использования:

1. Найти есть ли совпадение по строке.

```php
$str1= 'my id: 5353232';
$str2 = 'another string';
$pattern = '/:\s(\d+)/us';

$result1 = preg_match($pattern, $str1);
$result2 = preg_match($pattern, $str2);
var_dump($result1, $result2);
```

Как видите preg_match возвращает 1 если совпадение было, если нет 0. Если в регулярном выражении будет ошибка, функция вернет false.

2. Найти совпадения. Если в качестве третьего аргумента указать переменную, в нее будет записан массив найденных совпадений.

```php
$str1= 'my id: 5353232';
$str2 = 'another string';
$pattern = '/:\s(\d+)/us';

$result1 = preg_match($pattern, $str1, $matches1);
$result2 = preg_match($pattern, $str2, $matches2);
echo '<pre>';
var_dump($result1, $result2, $matches1,  $matches2);
```

Обратите внимание что результаты каждой подмаски записаны в отдельную строку, по индексу который соответствует номеру подмаски, а в 0 индексе общий результат регулярного выражения.

Полная документация: http://php.net/manual/ru/function.preg-match.php

Задание №4

В файле дан массив html ссылок. Используя регулярные выражения необходимо получить массив ссылок без html кода. Результат должен выглядеть так:

```
$array = [
```

```
    'http://example.com/example/',
    'http://php36.loc/task-list/',
    'https://www.facebook.com/realfive/',
    'http://realfive.eu/',
];
```

Задание №5

preg_match_all — выполняет глобальный поиск шаблона в строке. Если взять пример из 3 задания и изменить строку поиска добавив несколько вариантов соответствующих шаблону, мы увидим, что в результате поиска будет только первое совпадение:

```php
$str= 'my id: 5353232, my id: 454645';
$pattern = '/:\s(\d+)/us';

preg_match($pattern, $str, $matches);

echo '<pre>';
var_dump($matches);
```

Если нам нужны все совпадения, мы должны воспользоваться preg_match_all.

```php
$str= 'my id: 5353232, my id: 454645';
$pattern = '/:\s(\d+)/us';

preg_match_all($pattern, $str, $matches);

echo '<pre>';
var_dump($matches);
```

Как видите в этом примере были найдены все совпадения, а не только 1, а в результирующем массиве записаны не строки совпадений, а вложенные массивы.

Полная документация: http://php.net/manual/ru/function.preg-match-all.php

Задание №6

Используя регулярное выражение из 4 задания и функцию preg_match_all сделайте такой же массив как в результате 4 задания, но теперь все ссылки записаны в одной строке.

Задание №7

preg_split — разбивает строку по регулярному выражению. Функция похожа на explode, но вместо конкретной строки использует шаблон регулярного выражения.

```php
$str = '7777777 '
        . 'next phone: 8888888 '
        . 'another phone 4546656 '
        . 'and another one 5435436';
$pattern = '/\D+/us';

$phones = preg_split($pattern, $str);
echo '<pre>';
var_dump($phones);
```

Полная документация: http://php.net/manual/ru/function.preg-split.php

Задание №8

preg_replace — выполняет поиск и замену по регулярному выражению. Для примера напишем простой аналог функции trim

```php
function myTrim($str) {
    $pattern = '#(^\s+)|(\s+$)#us';
    return preg_replace($pattern, '', $str);
}

$str = " not trimmed str \n";
$trimmedStr = myTrim($str);
echo '<pre>';
var_dump($trimmedStr);
```

Также преимуществом этой функции является то что во втором аргументе (строка на которую мы заменяем текст, подходящий под шаблон) мы можем ссылаться на подмаски по номерам.

Для примера напишем функцию, которая будет менять формат даты с dd.mm.yyyy на yyyy/mm/dd

```php
function changeDateFormat($date){
    return preg_replace('#(\d{2})\.(\d{2})\.(\d{4})#', '$3/$2/$1', $date);
}

$date = '15.05.1999';
$newFormat = changeDateFormat($date);

var_dump($newFormat);
```

Полная документация: http://php.net/manual/ru/function.preg-replace.php

Задание №9

В JS также есть регулярные выражения. Создаются они через литералы или функцию-конструктор (но в основном используют литералы). Синтаксис практически совпадает с синтаксисом регулярных выражений в PHP.

Ключей всего 3:

g – глобальное сопоставление (при замене для замены по всей строке, а не только первого значения).

i – игнорирование регистра при сопоставлении.

m – сопоставление по нескольким строкам; символы начала и конца (^ и $) начинают работать по нескольким строкам (то есть, происходит сопоставление с началом или концом каждой строки (строки разделяются символами \n или \r), а не только с началом или концом всей вводимой строки).

Еще 1 ключ пока работает не во всех браузерах, так что пока мы его не рассматриваем.

```js
var rg  = /\w/gi;
```

Подробнее: https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/RegExp

Основные методы для работы с регулярными выражениями:

RegExp.prototype.exec() – выполняет поиск сопоставлений регулярного выражения в своём строковом параметре.

RegExp.prototype.test() – пытается сопоставить регулярное выражение своему строковому параметру.

String.prototype.replace() – возвращает новую строку с некоторыми или всеми сопоставлениями с шаблоном, заменёнными на заменитель.

String.prototype.split() - разбивает объект String на массив строк путём разделения строки указанным регулярным выражением.

Заметьте некоторые методы для работы с регулярными выражениями работают через как методы регулярного выражения, а некоторые как методы строки.

```
var rg = /\d+/gi;
var str = '543643 tfrd 64343 gdskgdlf 63464643';

console.log(rg.test('ddf 54435'));
console.log(rg.test('random string'));

console.log(rg.exec('ddf 43'));
console.log(rg.exec('random string'));

console.log(str.split(/\D+/));

console.log(str.replace(/\D+/, ' replaced '));
console.log(str.replace(/\D+/g, ' replaced '));
```

Задание №10 *

Парсер. Парсер – это скрипт который собирает информацию со сторонних сайтов.

Для написания парсера необходима библиотека cURL. С библиотекой можно ознакомиться тут: http://php.net/manual/ru/book.curl.php, а в общем функция запроса одной страницы стороннего сайта с использованием cURL написана в файле с заданием.

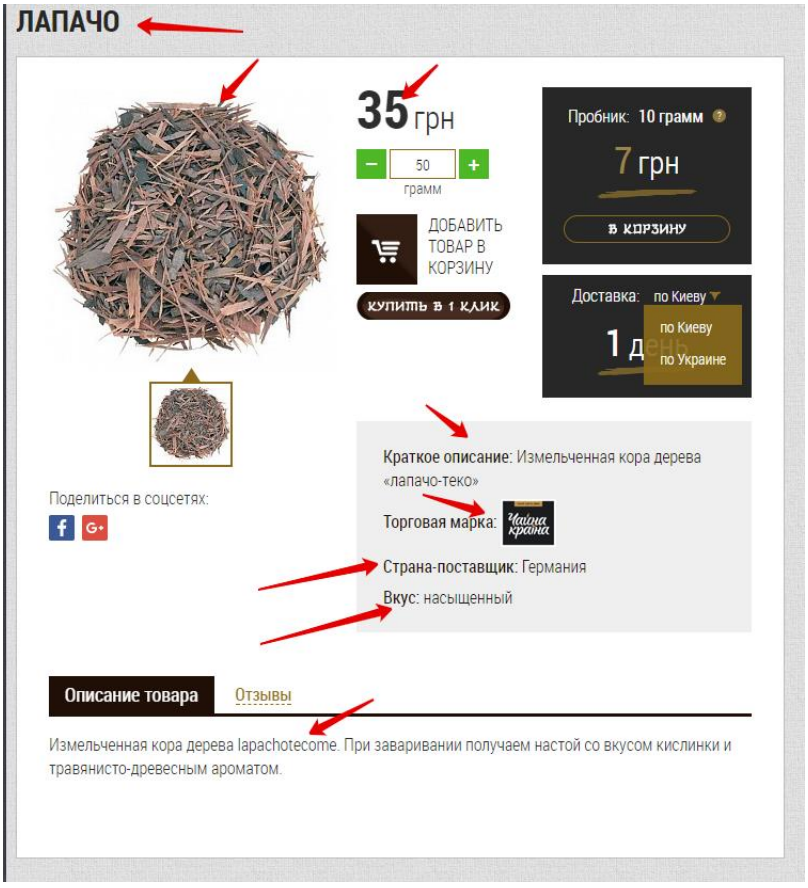
Универсального парсера в принципе существовать не может и решения необходимо подбирать под конкретный сайт. В данном случае мы будем парсить небольшой интернет магазин https://teaplace.com.ua/ и в этом случае алгоритм следующий (для многих интернет магазинов алгоритм приблизительно одинаков, но код будет разным в зависимости от сайта-источника).

1. Зайти на главную страницу и получить ссылки на подкатегории с описаниями из основного меню

В результате должен получиться массив вроде

```
[
        [
                'link' => 'https://teaplace.com.ua/catalog/tseylonskiy/',
               'category' => 'Черный чай',
               'sub_category' => 'Цейлонский',
        ],
        [
               'link' => 'https://teaplace.com.ua/catalog/kitayskiy/',
                'category' => 'Черный чай',
               'sub_category' => 'Китайский',
        ],
        …
];
```

2. Начинаем проход в цикле по подкатегориям.
3. Заходим на страницу подкатегории забираем товары с первой страницы и определяем кол-во страниц
4. Заходим на страницу товара и собираем следующую информацию: id, название, цену, краткое описание, название торговой марки (если есть), страну поставщика (если есть), вкус (если есть), полное описание, ссылку на картинку



Ид товара в исходном коде

```
<div itemscope itemtype="http://schema.org/Product">
  <div class="product-inside">
    <p class="product_id i-wai" data-id="590"></p>
    <p class="product_name i-wai" itemprop="name">Лапачо</p>
    <div class="product-inside_left">
      <div class="product-inside__slider">
        <div class="product-inside__gallery">
          <a href='/upload/iblock/e49/lapacho.jpg' class = 'cloud-zoom' id='zoom1' data-rel="zoomWidth:362,zoomHeight:420,adjustX:22,adjustY:-4">
            <img itemprop="image" src="/upload/resize_cache/iblock/e49/266_266_2/lapacho.jpg" alt="Лапачо" />
          </a>
        </div>
        <div class="product-inside__thumb-gallery">
          <a href='/upload/iblock/e49/lapacho.jpg' class='cloud-zoom-gallery' title='Лапачо' data-rel="useZoom:'zoom1',smallImage:'/upload/resize_cache/iblock/e49/266_266_2/lapacho.jpg'">
            <span class="ico-active"></span>
            <img src="/upload/resize_cache/iblock/e49/74_76_2/lapacho.jpg" alt="Лапачо"/>
          </a>
                  </div>
      </div>
      <span>Поделиться в соцсетях:</span>
      <div class="product-like__block">
                          <div class="ya-share2" data-services="facebook,gplus"></div>
      </div>
    </div>
```

ID товара

Используя информацию о подкатегории и со страницы товара должен получится следующий массив о каждом товаре

```
[

          'id' => '590',

          'category' => 'Этнический чай',

          'sub_category' => 'Лапачо',

          'img' => 'https://teaplace.com.ua/upload/resize_cache/iblock/e49/266_266_2/lapacho.jpg',

          'price' => '35',

          'short_desc' => 'Измельченная кора дерева «лапачо-теко»',

          'desc' => 'Измельченная кора дерева lapachotecome. При заваривании получаем настой со вкусом кислинки и травянисто-древесным ароматом.',

          'trade_mark' => 'Чайна країна',

          'country' => 'Германия',

          'taste' => 'Насыщенный',

];
```

5. Записываем информацию о товаре в бд (создайте бд и таблицу под эту задачу).
6. Проходим по всем страницам подкатегории забирая с каждой товары и выполняем пункт 4 и 5 для каждого товара.
7. Переходим к следующей подкатегории из цикла 2 и выполняем для всех подкатегорий пункты 3-6.