Web Scraping Using PHP - Parse IMDB.com Movies HTML

clevertechie.com/php/97/web-scraping-php-parse-imdb.com-movies-html

PHP | 17 Feb, 2017 | Clever Techie
Google +2 39 0

In this tutorial we're going to be using regular expressions to parse HTML. This is a more advanced tutorial so you can check out my video on regular expressions before going through this. We're going to be parsing out the IMDb website, which is an Internet movie database, and I'm going to be using a website called www.regex101.com to test regular expressions against strings to make sure we're matching them correctly. Because this is an advanced tutorial, I'll be posting each portion of code and explaining how it works as we walk through it. Directly below is the full source code, but skip down further and I'll walk through each portion of the code.

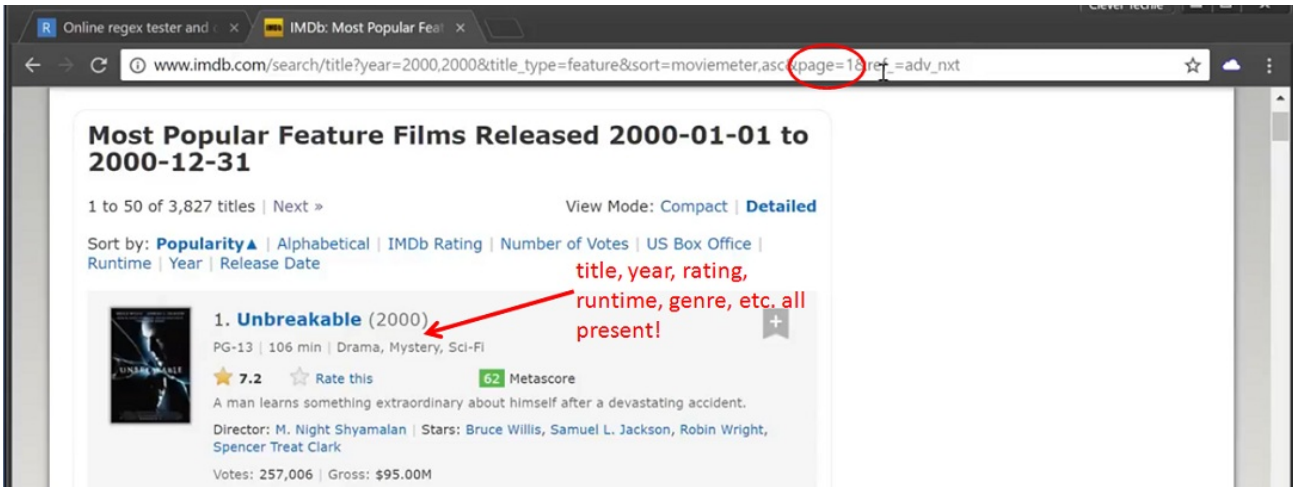
```
<?php

function scrape_imdb($year_start, $year_end, $page_start, $page_end){

$curl = curl_init();

$all_data = array();

for ($page = $page_start; $page <= $page_end; $page++){

$url =
"http://www.imdb.com/search/title?year=$year_start,$year_end&title_type=feature&sort=moviemeter,asc&page=$page&ref_=adv_nxt";
curl_setopt($curl, CURLOPT_URL, $url);

curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

$result = curl_exec($curl);

$movies = array();

preg_match_all('!<a href="\/title\/.*?\/\?ref_=adv_li_tt"\n>(.*?)<\/a>!',$result,$match);

$movies['title'] = $match[1];

preg_match_all('!<span class="lister-item-year text-muted unbold">.*?\((\d{4})\)<\/span>!',$result,$match);

$movies['year'] = $match[1];

preg_match_all('!loadlate="(.*?)"!',$result,$match);

$movies['image'] = $match[1];

preg_match_all("!<p class=\"text-muted\s\">(.*?)<\/p>!is",$result, $match);

for ($i=0;$i<count($match[1]);$i++){

if (preg_match('!<span class="certificate">(.*?)<\/span>!',$match[1][$i],$certificate)){

$movies['certificate'][$i] = $certificate[1];

}

else {

$movies['certificate'][$i] = '';

}

if (preg_match('!<span class="runtime">(\d{2,3}) min<\/span>!',$match[1][$i],$runtime)){

$movies['runtime'][$i] = $runtime[1];

}

else {

$movies['runtime'][$i] = '';
}

if (preg_match('!<span class="genre">\n(.*?)\s*?<\/span>!',$match[1][$i],$genre)){

$movies['genres'][$i] = $genre[1];

}

else {

$movies['genres'][$i] = '';

}

}

preg_match_all('!<div class="ratings-bar">(.*?)<\/span>!is',$result,$match);

for ($i=0;$i<count($match[1]);$i++)

{

if (preg_match("!data-value=\"(.*?)\"!i",$match[1][$i],$imdb_rating)){

$movies['imdb_rating'][$i] = $imdb_rating[1];

}

else {

$movies['imdb_rating'][$i] = '';

}

}

preg_match_all('!(<div class="inline-block ratings-metascore">

<span class="metascore (favorable|mixed|unfavorable)">(.*?)\s*?<\/span>\s*?Metascore\s*?<\/div>\s*?<\/div>\n)?<p
class="text-muted">(.*?)<\/p>!is',$result,$match);

for ($i=0;$i<count($match[0]);$i++){

if (preg_match('!metascore (favorable|mixed|unfavorable)">(.*?)\s*?</span>!',$match[0][$i],$metascore)){

$movies['metascore'][$i] = $metascore[2];

}

else {

$movies['metascore'][$i] = '';

}

if (preg_match('!<p class="text-muted\s?">\n(.*?)</p>!i',$match[0][$i],$description)){

$movies['description'][$i] = $description[1];

}

else {

$movies['description'][$i] = '';

}

}

preg_match_all('!<p class="">(.*?)<\/p>!is',$result,$match);

for ($i=0;$i<count($match[1]);$i++){

if (preg_match('!Directors?:\n<a href="/name/.*?/?ref_=adv_li_dr_0"\n>(.*?)</a>\n!s',$match[1][$i],$directors))
```

```
{
$clean_directors = preg_replace('!(<a href="\/name\/.*?\/?ref_=adv_li_dr_\d"\n>|<\/a>|\n)!','',$directors[1]);
$movies['directors'][$i] = $clean_directors;
}
else {
$movies['directors'][$i] = '';
}
if (preg_match('!Stars?:\n(.*?)<\/a>\n!is',$match[1][$i],$stars)){
preg_match_all('!>(.*?)<!',$stars[1],$all_stars);
$movies['stars'][$i] = implode(', ',$all_stars[1]);
}
else {
$movies['stars'][$i] = '';
}
}
$regex = '!(<p class="sort-num_votes-visible">
\s*?<span class="text-muted">Votes:<\/span>
\s*?<span name="nv" data-value="(\d*?)">.*?<\/span>)?.*?
(<span class="ghost">\|<\/span>\s*?<span class="text-muted">Gross:<\/span>
\s*?<span name="nv" data-value="(.*?)">.*?<\/span>
\s*?<\/p>)?(\s*?<\/div>\s*?<\/div>\s*?)(<div class="lister-item mode-advanced">|</div>\s*?<div class="nav">)!is';
preg_match_all($regex,$result,$match);
for ($i=0;$i<count($match[0]);$i++){
if (preg_match('!Votes:</span>\s*?<span name="nv" data-value="(\d*?)">!is',$match[0][$i],$votes)){
$movies['votes'][$i] = $votes[1];
}
else {
$movies['votes'][$i] = '';
}
if (preg_match('!Gross:</span>\s*?<span name="nv" data-value="(.*?)">!is',$match[0][$i],$gross)){
$movies['gross'][$i] = $gross[1];
}
else {
$movies['gross'][$i] = '';
}
}
foreach($movies as $key => $value) {
for ($i = 0; $i < count ($movies[$key]);$i++){
```

```
$data[$i][$key] = $movies[$key][$i];

}

}

$all_data = array_merge($data,$all_data);

}

return $all_data;

}

?>
```

Loading in an IMDb Search Page

The first thing to do when matching content from a website like this is to identify the search string from which to match the content. We're going to match movies released in the year 2000 and we'll be starting our search at page 1. For each movie there are a number of elements: title, year released, description, director, stars, etc.



www.imdb.com/search/title?year=2000,2000&title_type=feature&sort=moviemeter,asc&page=1&ref_=adv_nxt

Most Popular Feature Films Released 2000-01-01 to 2000-12-31

1 to 50 of 3,827 titles | Next »

View Mode: Compact | Detailed

Sort by: Popularity ▲ | Alphabetical | IMDb Rating | Number of Votes | US Box Office | Runtime | Year | Release Date

1. Unbreakable (2000)

PG-13 | 106 min | Drama, Mystery, Sci-Fi

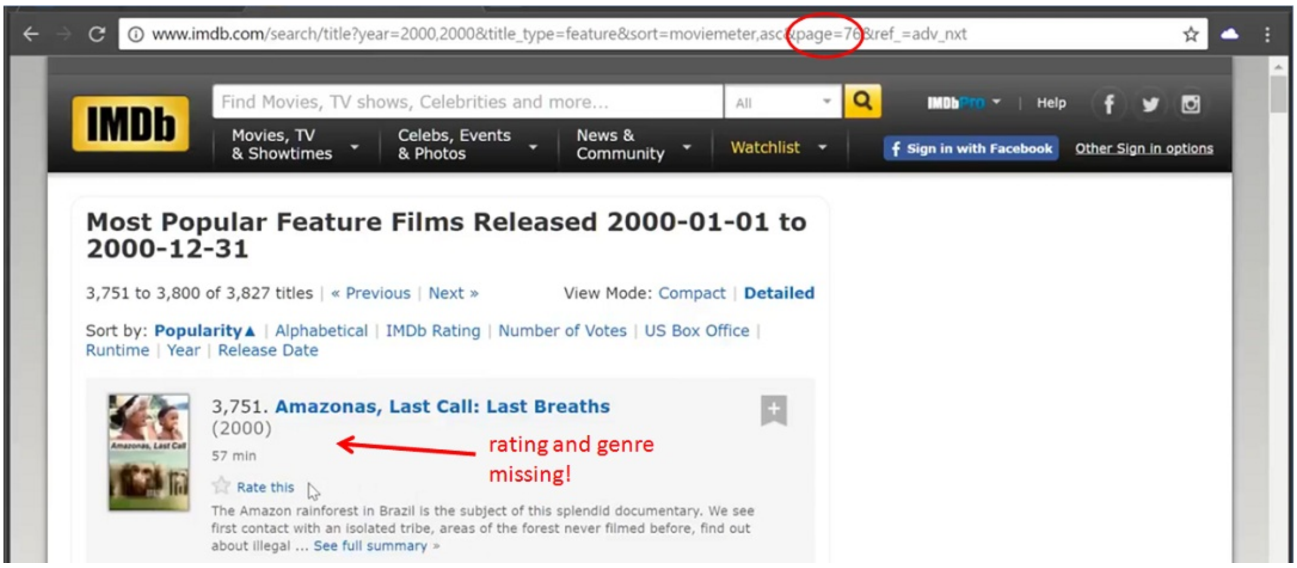
7.2 Rate this 62 Metascore

A man learns something extraordinary about himself after a devastating accident.

Director: M. Night Shyamalan | Stars: Bruce Willis, Samuel L. Jackson, Robin Wright, Spencer Treat Clark

Votes: 257,006 | Gross: $95.00M

The tricky thing is, if we go to a later page (we'll use page 76 in this example) we see that it is missing a lot of the content. Not every movie has content for every element, so we can't just match all the elements individually. Instead we have to match a block of elements at the same time. For example, we'll match the movie runtime, genre, and IMDb rating all at the same time and then check to see if any of those individual elements is empty. If they are, we'll set the array value to an empty string to make sure we have the right number of elements in the array.



www.imdb.com/search/title?year=2000,2000&title_type=feature&sort=moviemeter,asc&page=76&ref_=adv_nxt

Most Popular Feature Films Released 2000-01-01 to 2000-12-31

3,751 to 3,800 of 3,827 titles | « Previous | Next »

View Mode: Compact | Detailed

Sort by: Popularity ▲ | Alphabetical | IMDb Rating | Number of Votes | US Box Office | Runtime | Year | Release Date

3,751. Amazonas, Last Call: Last Breaths (2000)

57 min

Rate this

The Amazon rainforest in Brazil is the subject of this splendid documentary. We see first contact with an isolated tribe, areas of the forest never filmed before, find out about illegal ... See full summary »

The first piece of code shown below will load up our search results based on the function inputs. We want to load up page 76 of the movies released in 2000 for this example, so the function inputs will be (2000, 2000, 76, 76). We'll store everything inside the result variable and create an empty array called movies.

```
<?php
```

function scrape_imdb($year_start, $year_end, $page_start, $page_end){

$curl = curl_init();

$all_data = array();

for ($page = $page_start; $page <= $page_end; $page++){

$url =
"http://www.imdb.com/search/title?year=$year_start,$year_end&title_type=feature&sort=moviemeter,asc&page=$page&ref_=adv_nxt";
curl_setopt($curl, CURLOPT_URL, $url);

curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

$result = curl_exec($curl);

$movies = array();

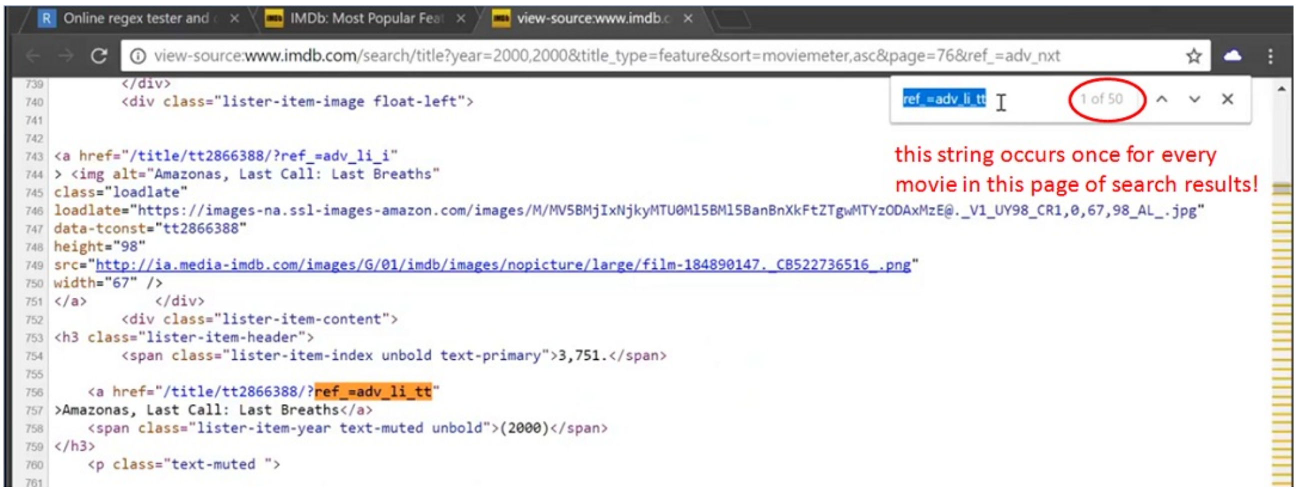
Matching the Movie Title

The movie title will be matched on its own using the code shown below.

preg_match_all('!<a href="\/title\/.*?\/\?ref_=adv_li_tt"\n>(.*?)<\/a>!',$result,$match);

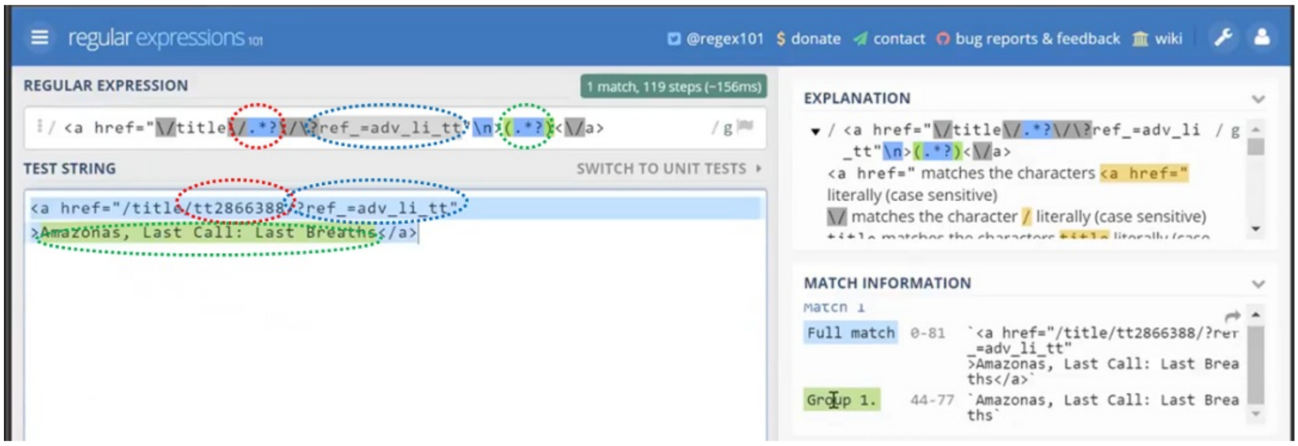
$movies['title'] = $match[1];

The regular expression is matching a phrase from the page source, and to verify that that is a reliable way to match the titles you can search a portion of that phrase in the page source of your IMDb search. It is clear in the screenshot below that the phrase appears 50 times in the page source from page 76 of the search, and that page has 50 movies displayed, confirming that it appears for each title.


Online regex tester and
IMDb: Most Popular Feat
view-source:www.imdb.c
view-source:www.imdb.com/search/title?year=2000,2000&title_type=feature&sort=moviemeter,asc&page=76&ref_=adv_nxt
ref_=adv_li_tt
1 of 50
this string occurs once for every movie in this page of search results!

```
        </div>
        <div class="lister-item-image float-left">


<a href="/title/tt2866388/?ref_=adv_li_i"
> <img alt="Amazonas, Last Call: Last Breaths"
class="loadlate"
loadlate="https://images-na.ssl-images-amazon.com/images/M/MV5BMjIxNjkyMTU0Ml5BMl5BanBnXkFtZTgwMTYzODAxMzE@._V1_UY98_CR1,0,67,98_AL_.jpg"
data-tconst="tt2866388"
height="98"
src="http://ia.media-imdb.com/images/G/01/imdb/images/nopicture/large/film-184890147._CB522736516_.png"
width="67" />
</a>          </div>
        <div class="lister-item-content">
<h3 class="lister-item-header">
        <span class="lister-item-index unbold text-primary">3,751.</span>

    <a href="/title/tt2866388/?ref_=adv_li_tt"
>Amazonas, Last Call: Last Breaths</a>
    <span class="lister-item-year text-muted unbold">(2000)</span>
</h3>
    <p class="text-muted ">
```

Now let's look at our regular expression. Paste it into regex101 along with the HTML from the page source. This is shown in the screenshot below, breaking down the pieces of the regular expression. Instead of “tt2866388” the regular expression contains “.*?”, shown in a red circle. The “.*?” means the expression will capture anything in that space. That number “tt2866388” may change from title to title, so we don't want to specify it in the regular expressin. In the blue circles is the string that we determined was constant for all the movie titles. There is a line break in our HTML code after that so we include a line break in our regular expression, followed by the portion in the green circle which captures the movie title. We use the same “.*?” as before, along with round brackets to capture that portion.



REGULAR EXPRESSION

1 match, 119 steps (~156ms)

```
/ <a href="\/title\/.*?\/\?ref_=adv_li_tt"\n>(.*?)<\/a>                /g
```

TEST STRING

```
<a href="/title/tt2866388/?ref_=adv_li_tt"
>Amazonas, Last Call: Last Breaths</a>
```

EXPLANATION

```
/ <a href="\/title\/.*?\/\?ref_=adv_li  / g
  _tt"\n>(.*?)<\/a>
```

`<a href="` matches the characters `<a href="` literally (case sensitive)
`\/` matches the character `/` literally (case sensitive)

MATCH INFORMATION

```
Match 1
Full match   0-81    `<a href="/title/tt2866388/?ref
                     _=adv_li_tt"
                     >Amazonas, Last Call: Last Brea
                     ths</a>`
Group 1.     44-77   `Amazonas, Last Call: Last Brea
                     ths`
```

Now that you've tested your regular expression, you can print the movies array to make sure there are 50 elements, each containing a movie title.

Matching the Year

The movie release year is matched in a similar manner to the movie title. The code block is shown below.

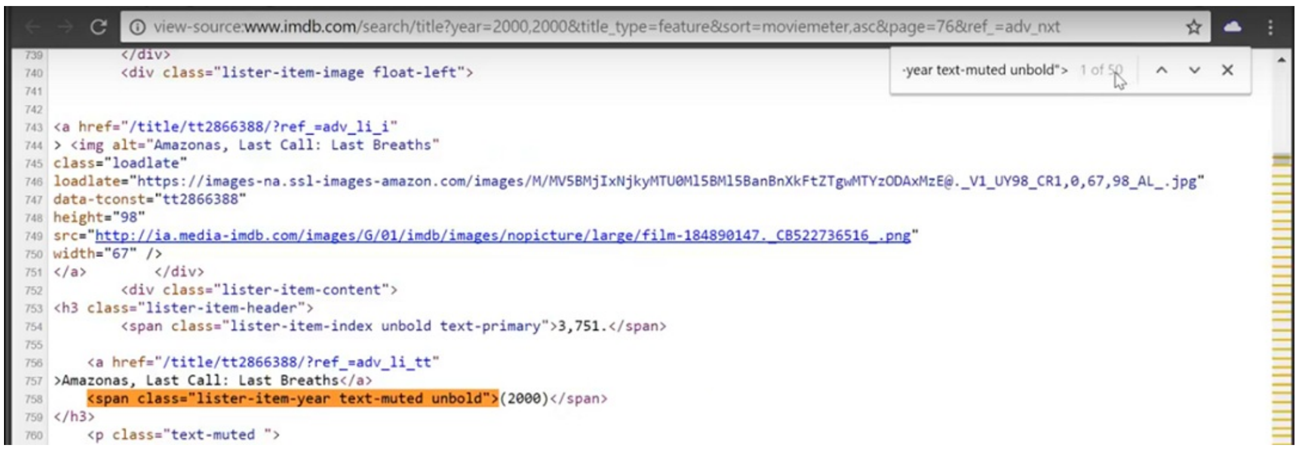
Similarly to matching the movie title, we'll copy the regular expression for matching the year into regex101 and search for the string in the page source. Once again it matches 50 elements, showing that it is a reliable way to match the date.

preg_match_all('!<span class="lister-item-year text-muted unbold">.*?\((\d{4})\)<\/span>!',$result,$match);

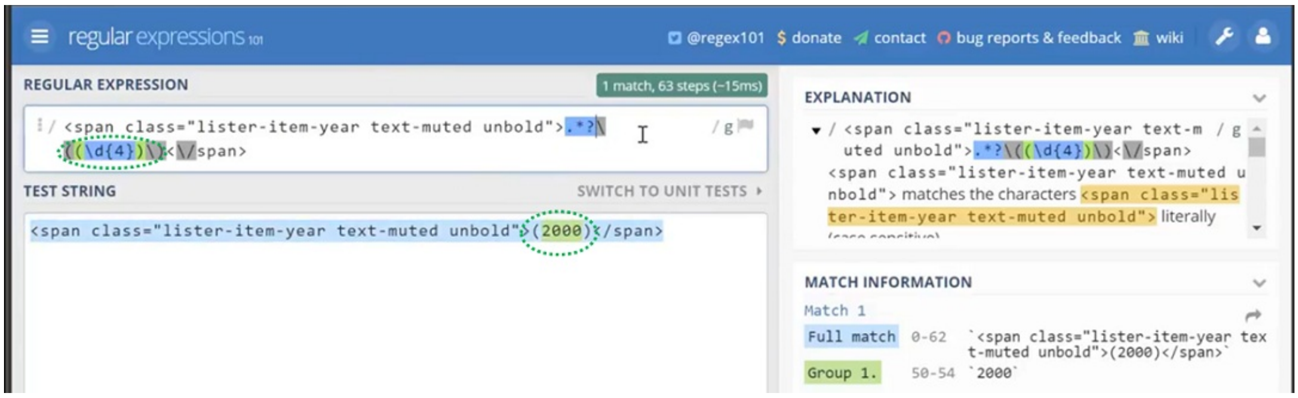
$movies['year'] = $match[1];

As seen in the page source below, the date is in a span tag that is always the same. The year is inside the round brackets and is the only thing that might change from movie to movie (although since we are looking at movies from 2000 it will be 2000 for all of the movies in this example).

Once again, paste the regular expression and HTML from the page source into regex to understand what it is matching. The "d{4}" phrase in the regular expression matches a four digit number. The ".*?" in the expression is in case there are spaces in the HTML.


view-source:www.imdb.com/search/title?year=2000,2000&title_type=feature&sort=moviemeter,asc&page=76&ref_=adv_nxt
-year text-muted unbold">
1 of 50

```
        </div>
        <div class="lister-item-image float-left">


<a href="/title/tt2866388/?ref_=adv_li_i"
> <img alt="Amazonas, Last Call: Last Breaths"
class="loadlate"
loadlate="https://images-na.ssl-images-amazon.com/images/M/MV5BMjIxNjkyMTU0Ml5BMl5BanBnXkFtZTgwMTYzODAxMzE@._V1_UY98_CR1,0,67,98_AL_.jpg"
data-tconst="tt2866388"
height="98"
src="http://ia.media-imdb.com/images/G/01/imdb/images/nopicture/large/film-184890147._CB522736516_.png"
width="67" />
</a>        </div>
        <div class="lister-item-content">
<h3 class="lister-item-header">
        <span class="lister-item-index unbold text-primary">3,751.</span>

    <a href="/title/tt2866388/?ref_=adv_li_tt"
>Amazonas, Last Call: Last Breaths</a>
    <span class="lister-item-year text-muted unbold">(2000)</span>
</h3>
    <p class="text-muted ">
```



REGULAR EXPRESSION

1 match, 63 steps (~15ms)

```
/ <span class="lister-item-year text-muted unbold">.*?\
  \((\d{4})\)<\/span>                                  /g
```

TEST STRING

```
<span class="lister-item-year text-muted unbold">(2000)</span>
```

EXPLANATION

```
/ <span class="lister-item-year text-m  / g
  uted unbold">.*?\((\d{4})\)<\/span>
```

`<span class="lister-item-year text-muted unbold">` matches the characters `<span class="lister-item-year text-muted unbold">` literally

MATCH INFORMATION

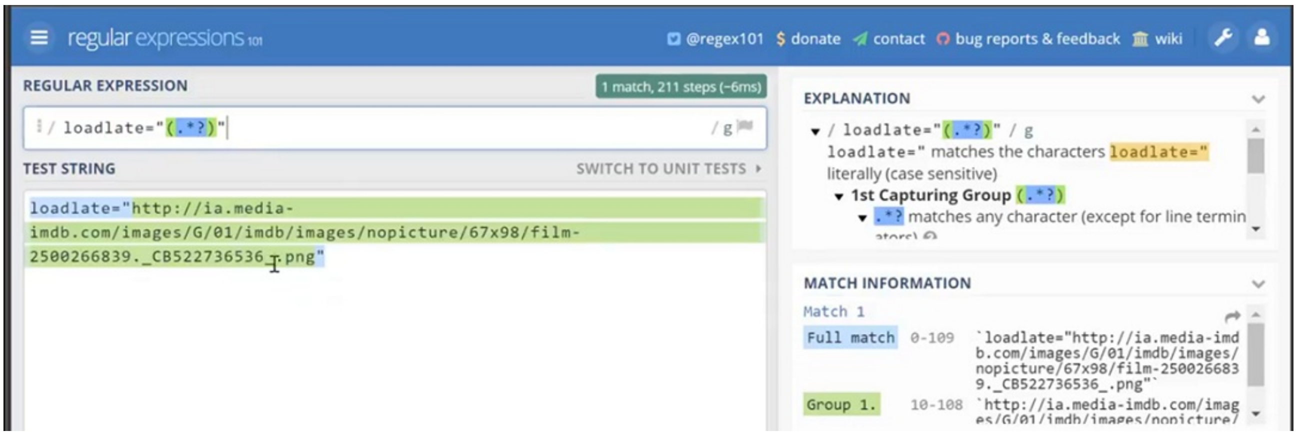
```
Match 1
Full match   0-62    `<span class="lister-item-year tex
                     t-muted unbold">(2000)</span>`
Group 1.     50-54   `2000`
```

Matching the Image URL

The code below for matching the image URL follows the same procedure as the movie title and year. Pasting the regular expression and HTML from the page source into regex101 shows that the simple regular expression grabs the image URL.

preg_match_all('!loadlate="(.*?)"!',$result,$match);

$movies['image'] = $match[1];



```
/ loadlate="(.*?)" / g
```

```
loadlate="http://ia.media-imdb.com/images/G/01/imdb/images/nopicture/67x98/film-2500266839._CB522736536_.png"
```

/ loadlate="(.*?)" / g
loadlate=" matches the characters loadlate=" literally (case sensitive)
1st Capturing Group (.*?)
.*? matches any character (except for line termin

Match 1
Full match 0-109 `loadlate="http://ia.media-imdb.com/images/G/01/imdb/images/nopicture/67x98/film-2500266839._CB522736536_.png"`
Group 1. 10-108 `http://ia.media-imdb.com/images/G/01/imdb/images/nopicture/

Matching Multiple Elements

Now for something a little more tricky! Instead of matching single elements we'll now match one HTML block which has the certificate, run time, and genre. The block of HTML is found on every page, but the specific elements inside the block may be empty. The code below matches the entire block and then loops through the block checking each element with preg_match. If any of the elements is empty it creates an empty string in place of that element. This is important because it ensures that the array has the proper number of elements so, for example, the title will line up with the correct genre even if the genre isn't listed for every movie.

preg_match_all("!<p class=\"text-muted\s\">(.*?)<\/p>!is",$result, $match);

for ($i=0;$i<count($match[1]);$i++){

if (preg_match('!<span class="certificate">(.*?)<\/span>!',$match[1][$i],$certificate)){

$movies['certificate'][$i] = $certificate[1];

}

else {

$movies['certificate'][$i] = '';

}

if (preg_match('!<span class="runtime">(\d{2,3}) min<\/span>!',$match[1][$i],$runtime)){

$movies['runtime'][$i] = $runtime[1];

}

else {

$movies['runtime'][$i] = '';

}

if (preg_match('!<span class="genre">\n(.*?)\s*?<\/span>!',$match[1][$i],$genre)){

$movies['genres'][$i] = $genre[1];

}

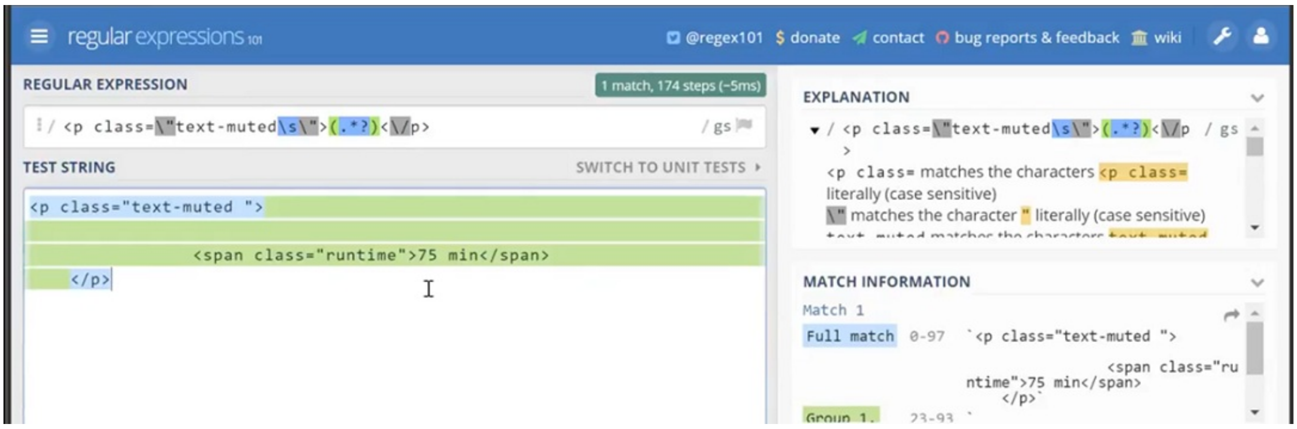
else {

$movies['genres'][$i] = '';

}

}

As we've done before, paste the HTML from the page source and the regular expression into regex101, making sure to set the single line parameter since we will be matching multiple lines. You can see that it is matching the HTML segment containing the run time while the other elements are empty.



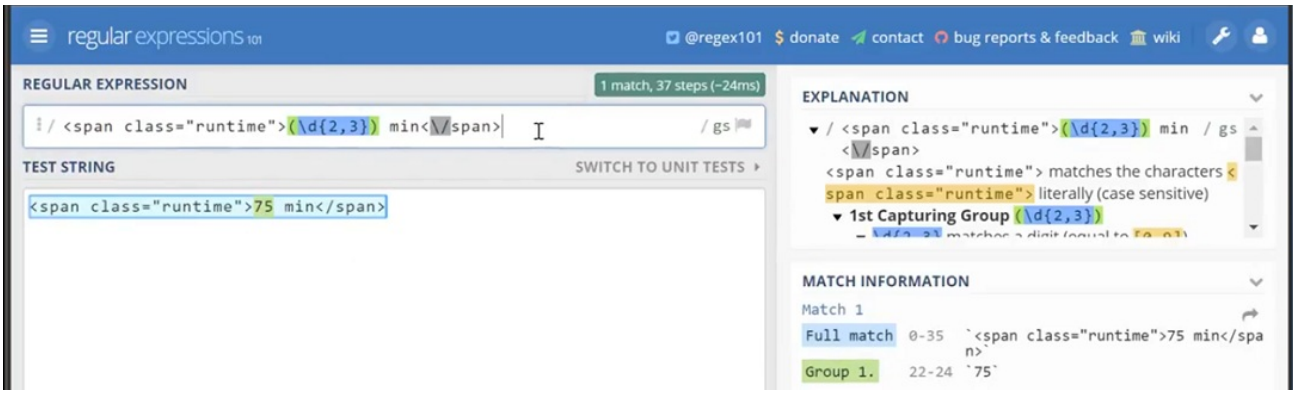
```
/ <p class=\"text-muted\s\">(.*?)<\/p> / gs
```

```
<p class="text-muted ">

                <span class="runtime">75 min</span>
        </p>
```

```
▾ / <p class=\"text-muted\s\">(.*?)<\/p / gs
  >
  <p class= matches the characters <p class=
  literally (case sensitive)
  \" matches the character " literally (case sensitive)
```

```
Match 1
Full match  0-97   `<p class="text-muted ">
                          <span class="ru
                   ntime">75 min</span>
                       </p>`
Group 1.    23-93  `
```

Whereas that regular expression grabbed the HTML content containing the runtime, the second regular expression inside the for loop pulls out the run time itself. Similar to using "d{4}" for the year, we use "d{2,3}" to pull out a 2 or 3 digit number for the run time. The regular expressions for the certificate and genre work similarly, but for this particular example those elements are empty. If you print your movie array you'll see that under "Certificate," "Genre," and "Run Time" you have lots of empty elements but the number of elements matches because we have filled the empty elements with empty strings.



```
/ <span class="runtime">(\d{2,3}) min<\/span> / gs
```

```
<span class="runtime">75 min</span>
```

```
▾ / <span class="runtime">(\d{2,3}) min / gs
    <\/span>
  <span class="runtime"> matches the characters <
  span class="runtime"> literally (case sensitive)
  ▾ 1st Capturing Group (\d{2,3})
```

```
Match 1
Full match  0-35   `<span class="runtime">75 min</spa
                   n>`
Group 1.    22-24  `75`
```

Matching the IMDb Rating

The IMDb rating is an interesting one to match, because even if the element is empty, there is an HTML block that will be present. Even though we are not matching multiple elements here, we will use a similar procedure to the way we matched the three elements previously, matching the whole HTML block and then using a loop to match the rating itself.

preg_match_all('!<div class="ratings-bar">(.*?)<\/span>!is',$result,$match);

for ($i=0;$i<count($match[1]);$i++)

{

if (preg_match("!data-value=\"(.*?)\"!i",$match[1][$i],$imdb_rating)){

$movies['imdb_rating'][$i] = $imdb_rating[1];

}

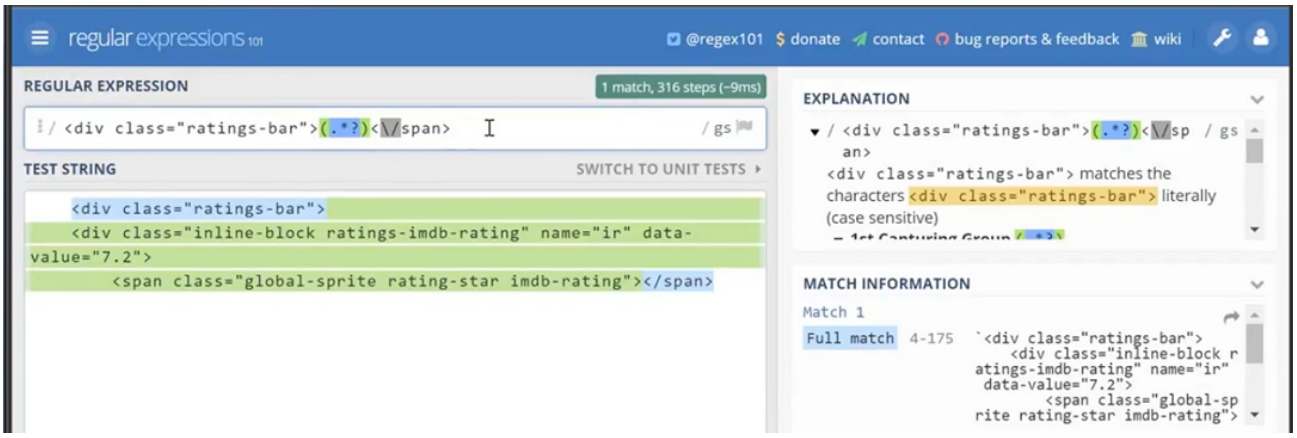
else {

$movies['imdb_rating'][$i] = '';

}

}

Following the same procedure we've used, paste the regular expression and HTML block into regex101 to see that it is matching the full block.



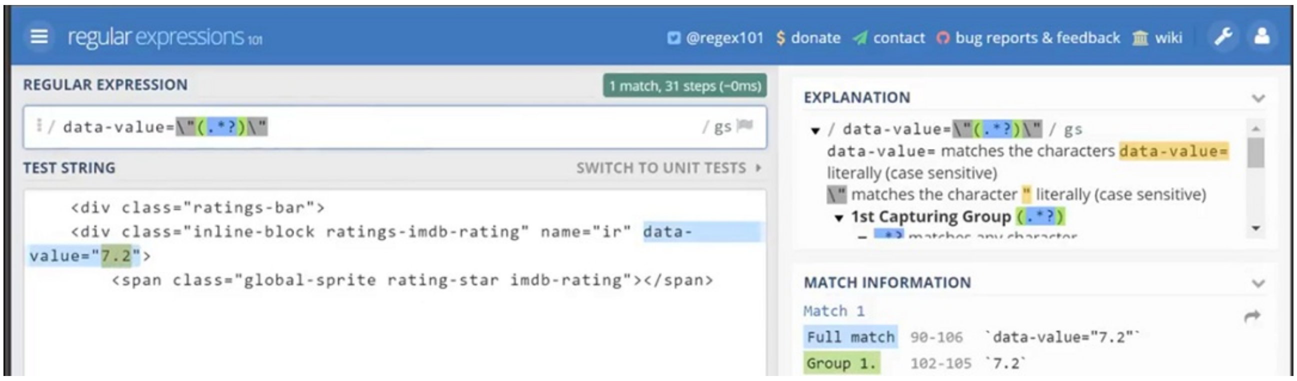
```
/ <div class="ratings-bar">(.*?)<\/span>   / gs
```

```
<div class="ratings-bar">
    <div class="inline-block ratings-imdb-rating" name="ir" data-
value="7.2">
        <span class="global-sprite rating-star imdb-rating"></span>
```

```
▾ / <div class="ratings-bar">(.*?)<\/sp  / gs
  an>
  <div class="ratings-bar"> matches the
  characters <div class="ratings-bar"> literally
  (case sensitive)
```

```
Match 1
Full match  4-175   `<div class="ratings-bar">
                         <div class="inline-block r
                    atings-imdb-rating" name="ir"
                     data-value="7.2">
                             <span class="global-sp
                    rite rating-star imdb-rating">
```

Once again, a for loop with an additional regular expression finds the data value itself. The sceenshot below shows this second regular expression matching the rating.



```
/ data-value=\"(.*?)\"   / gs
```

```
    <div class="ratings-bar">
    <div class="inline-block ratings-imdb-rating" name="ir" data-
value="7.2">
        <span class="global-sprite rating-star imdb-rating"></span>
```

```
▾ / data-value=\"(.*?)\" / gs
  data-value= matches the characters data-value=
  literally (case sensitive)
  \" matches the character " literally (case sensitive)
  ▾ 1st Capturing Group (.*?)
```

```
Match 1
Full match  90-106   `data-value="7.2"`
Group 1.    102-105  `7.2`
```

Matching Metascore and Description

Following the trend of matching progressively more tricky elements, the metascore and description have their own twist. Once again there is an HTML block that contains these elements, but the portion containing the metascore is not found at all on some pages of the search. To get around this we'll need to have make some pieces of the regular expression optional.

preg_match_all('!(<div class="inline-block ratings-metascore">

<span class="metascore (favorable|mixed|unfavorable)">(.*?)\s*?<\/span>\s*?Metascore\s*?<\/div>\s*?<\/div>\n)?<p class="text-muted">(.*?)<\/p>!is',$result,$match);

for ($i=0;$i<count($match[0]);$i++){

if (preg_match('!metascore (favorable|mixed|unfavorable)">(.*?)\s*?</span>!',$match[0][$i],$metascore)){

$movies['metascore'][$i] = $metascore[2];

}

else {

$movies['metascore'][$i] = '';

}

if (preg_match('!<p class="text-muted\s?">\n(.*?)</p>!i',$match[0][$i],$description)){

$movies['description'][$i] = $description[1];

}

else {

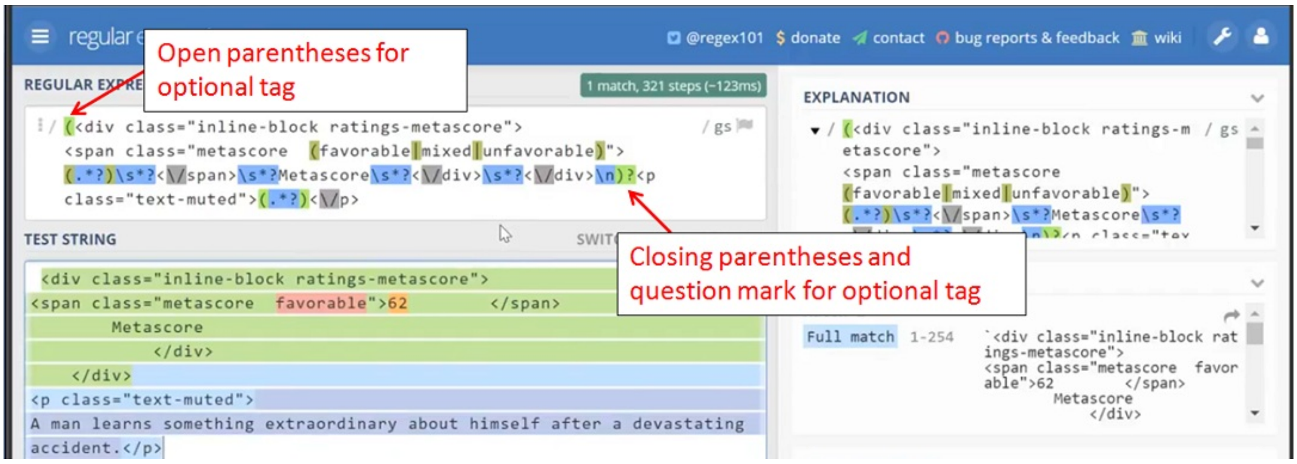
$movies['description'][$i] = '';

}

}

Paste the regular expression and HTML block (which can be found on page 1 of the IMDb search instead of page 76) into regex101. Notice that the majority of the regular expression is enclosed in round brackets with a question mark at the end. This makes it optional in case the block is not found on the page. The HTML tag including the “text-muted” line will always be found in the page source, so it does not need to be made optional.

Once again, the code loops through with preg_match to individually match the metascore and description.


regular e
@regex101
$ donate
contact
bug reports & feedback
wiki
REGULAR EXPRE
1 match, 321 steps (~123ms)
TEST STRING
SWITC
EXPLANATION
Open parentheses for optional tag
Closing parentheses and question mark for optional tag

```
/ (<div class="inline-block ratings-metascore">
<span class="metascore  (favorable|mixed|unfavorable)">
(.*?)\s*?<\/span>\s*?Metascore\s*?<\/div>\s*?<\/div>\n)?<p
class="text-muted">(.*?)<\/p>                                  / gs
```

```
 <div class="inline-block ratings-metascore">
<span class="metascore  favorable">62          </span>
          Metascore
               </div>
     </div>
<p class="text-muted">
A man learns something extraordinary about himself after a devastating
accident.</p>
```

```
Full match  1-254   `<div class="inline-block ratings-metascore">
                    <span class="metascore  favorable">62        </span>
                              Metascore
                                   </div>
```

Matching Directors and Stars

The principle here is exactly the same as for the previous elements so I won’t elaborate on it. After matching the main block, the code loops through and matches the directors and stars. The trick here is in cleaning up the name of the director and the list of stars for the array. Preg_match_all is used to match the individual star names from the HTML content. Then implode is used to create comma separated strings of the star names for each movie. Similar to using implode for the stars, we use preg_replace to clean up the names of the directors, replacing all the HTML that comes along with the director names.

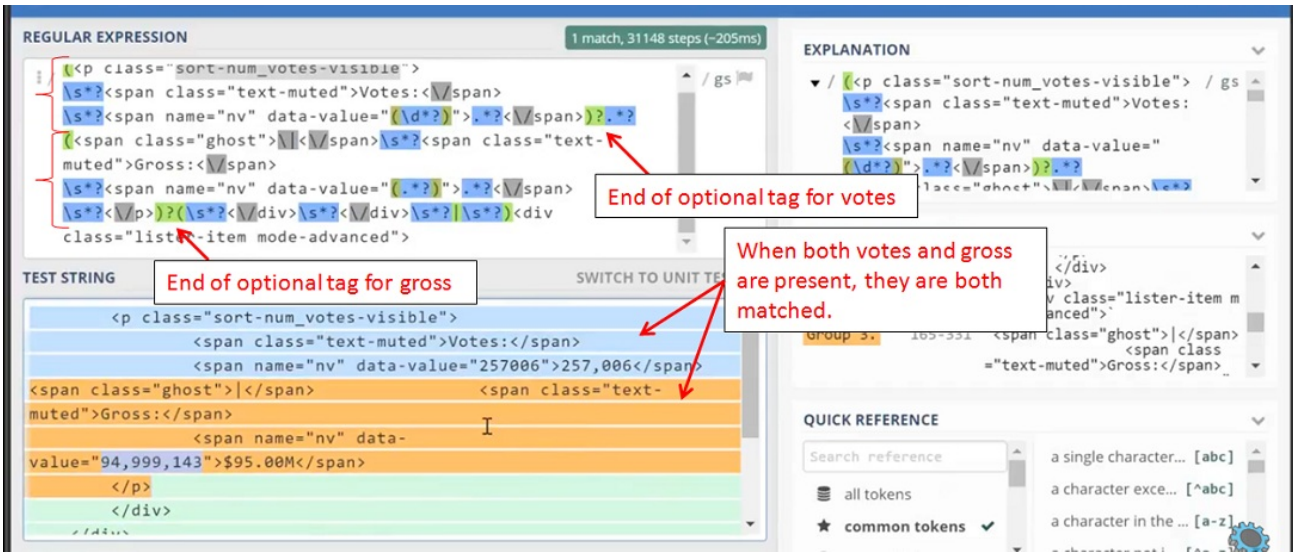
```
preg_match_all('!<p class="">(.*?)<\/p>!is',$result,$match);

for ($i=0;$i<count($match[1]);$i++){

if (preg_match('!Directors?:\n<a href="/name/.*?/?ref_=adv_li_dr_0"\n>(.*?)</a>\n!s',$match[1][$i],$directors))

{

$clean_directors = preg_replace('!(<a href="\/name\/.*?\/?ref_=adv_li_dr_\d"\n>|<\/a>|\n)!','',$directors[1]);

$movies['directors'][$i] = $clean_directors;

}

else {

$movies['directors'][$i] = ";

}

if (preg_match('!Stars?:\n(.*?)<\/a>\n!is',$match[1][$i],$stars)){

preg_match_all('!>(.*?)<!',$stars[1],$all_stars);

$movies['stars'][$i] = implode(', ',$all_stars[1]);

}

else {

$movies['stars'][$i] = ";

}

}
```

Matching Movie Votes and Gross

You’ve made it to the final elements! Matching the movie votes and gross is a little tricky because one or both of these may be absent. To handle that, the regular expression contains multiple optional pieces, as shown below.

```
$regex = '!(<p class="sort-num_votes-visible">

\s*?<span class="text-muted">Votes:<\/span>

\s*?<span name="nv" data-value="(\d*?)">.*?<\/span>)?.*?

(<span class="ghost">\|<\/span>\s*?<span class="text-muted">Gross:<\/span>

\s*?<span name="nv" data-value="(.*?)">.*?<\/span>

\s*?<\/p>)?(\s*?<\/div>\s*?<\/div>\s*?)(<div class="lister-item mode-advanced">|</div>\s*?<div class="nav">)!is';
```

If you paste in the full HTML block found in the page source from page 1 of the search results, it finds every block, as seen below. However, on some pages parts of the block may not be found, so we make each portion optional like we did with the metascore. You can see that the HTML piece containing the phrase “lister-item mode-advanced” is not made optional in the regular expression because it appears on all pages, even when both the votes and gross are absent.


REGULAR EXPRESSION
1 match, 31148 steps (~205ms)
/ gs
End of optional tag for votes
End of optional tag for gross
When both votes and gross are present, they are both matched.
TEST STRING
SWITCH TO UNIT TE
EXPLANATION
QUICK REFERENCE
Search reference
all tokens
common tokens
a single character... [abc]
a character exce... [^abc]
a character in the ... [a-z]

```
(<p class="sort-num_votes-visible">
\s*?<span class="text-muted">Votes:<\/span>
\s*?<span name="nv" data-value="(\d*?)">.*?<\/span>)?.*?
(<span class="ghost">\|<\/span>\s*?<span class="text-
muted">Gross:<\/span>
\s*?<span name="nv" data-value="(.*?)">.*?<\/span>
\s*?<\/p>)?(\s*?<\/div>\s*?<\/div>\s*?|\s*?)<div
class="lister-item mode-advanced">
```

```
        <p class="sort-num_votes-visible">
                <span class="text-muted">Votes:</span>
                <span name="nv" data-value="257006">257,006</span>
<span class="ghost">|</span>                <span class="text-
muted">Gross:</span>
                <span name="nv" data-
value="94,999,143">$95.00M</span>
        </p>
        </div>
```

```
Group 3.    165-331    <span class="ghost">|</span>
                                <span class
                    ="text-muted">Gross:</span>
```

Because we’ve made both the votes and gross elements optional, if you delete those in the HTML block in regex101 you see that the HTML content is still matched, but the array keys will be empty.

Since the regular expression is long and somewhat complex, it is stored in its own variable and then the variable is called in preg_match_all instead of pasting the expression directly into preg_match_all. This just serves to make the code simpler to read.

The remainder of the code is similar to the previous segments, looping through the movies to detmerine whether the votes and gross are blank and creating empty strings if that is the case.

```
preg_match_all($regex,$result,$match);

for ($i=0;$i<count($match[0]);$i++){

if (preg_match('!Votes:</span>\s*?<span name="nv" data-value="(\d*?)">!is',$match[0][$i],$votes)){

$movies['votes'][$i] = $votes[1];

}

else {

$movies['votes'][$i] = '';

}

if (preg_match('!Gross:</span>\s*?<span name="nv" data-value="(.*?)">!is',$match[0][$i],$gross)){

$movies['gross'][$i] = $gross[1];

}

else {

$movies['gross'][$i] = '';
}

}

foreach($movies as $key => $value) {

for ($i = 0; $i < count ($movies[$key]);$i++){

$data[$i][$key] = $movies[$key][$i];

}

}

$all_data = array_merge($data,$all_data);

}

return $all_data;

}

?>
```

Go on and test your code with page 76 of the search results and see that there are 50 elements for every field, even when some of them are empty. All of our regular expressions are working and the empty elements have empty strings as placeholders keeping everything in line.

I hope you enjoyed this tutorial. Make sure to watch the next video where we'll save all of this content to a MySQL database so we can display the whole thing on a webpage. As always, don't forget to comment, like, and subscribe to Clever Techie!